OC Pizza solution technique

OC Pizza

# Choix de l'architecture.

Deux possibilités techniques peuvent être mise en œuvre pour l'application une version monolithique ou une version à base de microservices.

## Architecture Monolithique.

Toute l'application est implémentée dans un seul programme et fonctionne sur une seule machine. On a une grosse application complexe qui demande beaucoup de ressource pour démarrer et fonctionner. En outre l'application n'est pas flexible et ne peut pas répondre à un accroissement de la demande des utilisateurs. Elle n'est pas scalable. Si on a besoin de répondre à plus de demande on va devoir dupliquer l'application mais cela pause un autre problème pour la gestion des données.

On doit donc avoir au minimum un serveur sur lequel fonctionne l'applicatif et un autre pour gérer la base de données. On pourra ainsi dupliquer l'applicatif sur un autre serveur qui communiquera avec la même base de données. On accroit ainsi le problème d'utilisation de ressources car on duplique même des services que l'on n'a pas besoin comme la consultation de la liste des clients ou le lancement de la préparation d'une commande.

Il est aussi difficile de demander à son fournisseur de service internet d'allouer un nouveau serveur pour dupliquer notre application dans un temps record.

Le premier inconvénient de la configuration monolithique est l'utilisation massive de ressources qui doivent être multipliées pour répondre à la demande sous réserve de la passation d'un nouveau contrat avec le fournisseur.

Toute application doit être maintenue avec des mise à jour et c'est difficile de savoir où se trouve le problème dans une application volumineuse et complexe. On doit en outre l'arrêter et la redémarrer complètement ce qui n'est pas seulement une perte de temps mais aussi de profit.

La réflexion sur une application monolithique a soulevé de gros désavantages et montré que nous devions avoir au moins deux serveurs :

* Un pour la base de données.
* Un pour le programme.

La seconde approche est le découpage de l'application en plusieurs parties : les microservices.

## Architecture microservices.

Dans cette architecture on découpe l'application en services indépendant les uns des autres. Chaque microservice est développé, testé et déployé séparément des autres. Ils échangent des données avec les autres à travers les différentes APIs qu'ils exposent. Ils suivent les règles suivantes :

* Ils implémentent une seule fonctionnalité.
* Ils peuvent être écrit avec des langages différents.
* Ils sont faiblement couplés car séparé physiquement les uns des autres.
* Ils sont facilement testés, déployés et maintenus.

Les microservices sont petits comparé à une application monolithique et ils peuvent fonctionner sur un même serveur ou des serveurs très proches. L'accroissement du temps de réponse dû à leur séparation est négligeable par rapport à une architecture monolithique. En outre comme ils sont petits, ils répondent plus vite.

### Scalabilité.

Pour répondre à la demande d'un site internet il est plus facile de dupliquer un microservice qu'une application monolithique. En plus les contrats cher l'hébergeur permet d'avoir des configurations flexibles. En outre on peut séparer les services qui demandent plus de mémoire et ceux qui demandent plus de puissance de calcul. On n'a pas besoin de refaire un contrat avec l'hébergeur pour faire passer de 2 à 5 microservices.

### Maintenance.

Par sa définition un microservice est un programme petit donc facile à maintenir et à faire évoluer. Il est plus facile aussi de rechercher et d'isoler une erreur dans 100 lignes de code que dans 10000 lignes. On peut aussi en cas de problème facilement arrêter et relancer un microservice sans bloquer les autres. On peut aussi porter un microservice dans un autre langage sans que cela ne perturbe les autres.

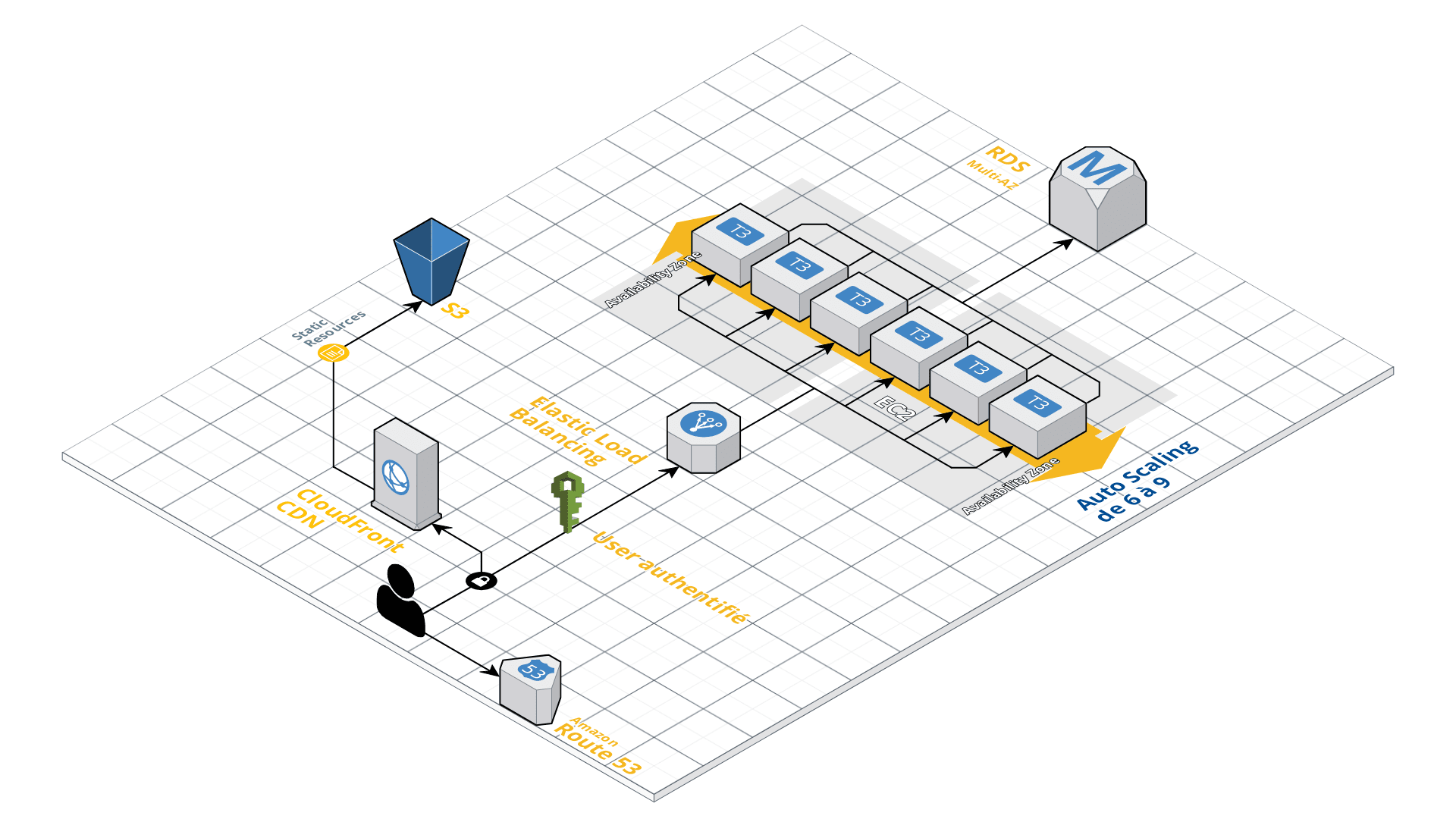
### Encapsulation.

On utilise des conteneurs comme Docker pour encapsuler les microservices avec les ressources dont ils ont besoin. Ils peuvent être écrit dans des langages différents et faire appel à des librairies ou frameworks différents. En cas de changement de ressource d'un microservice, les autres ne seront pas impactés.

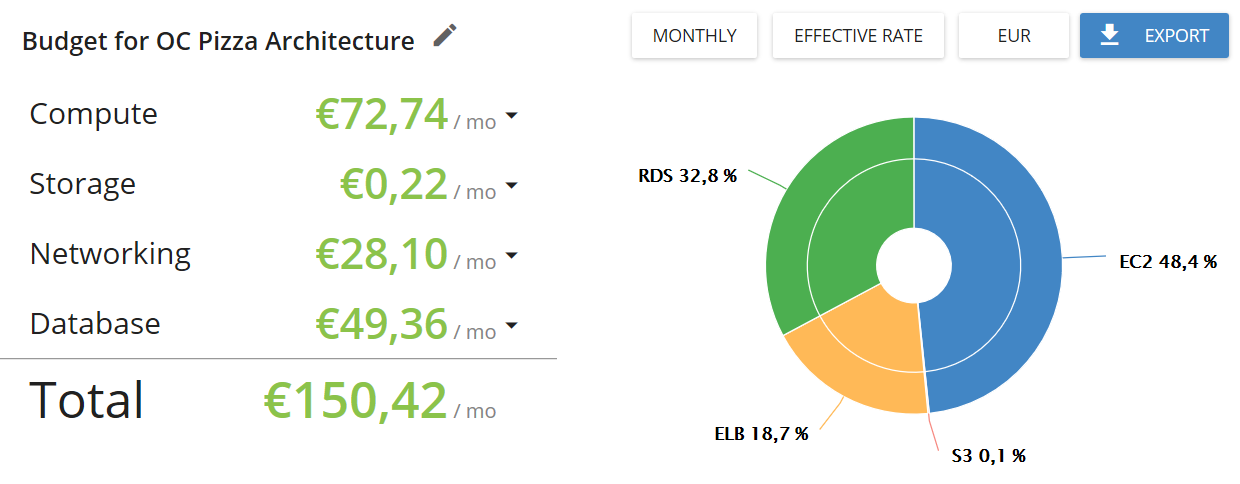
### Proxy.

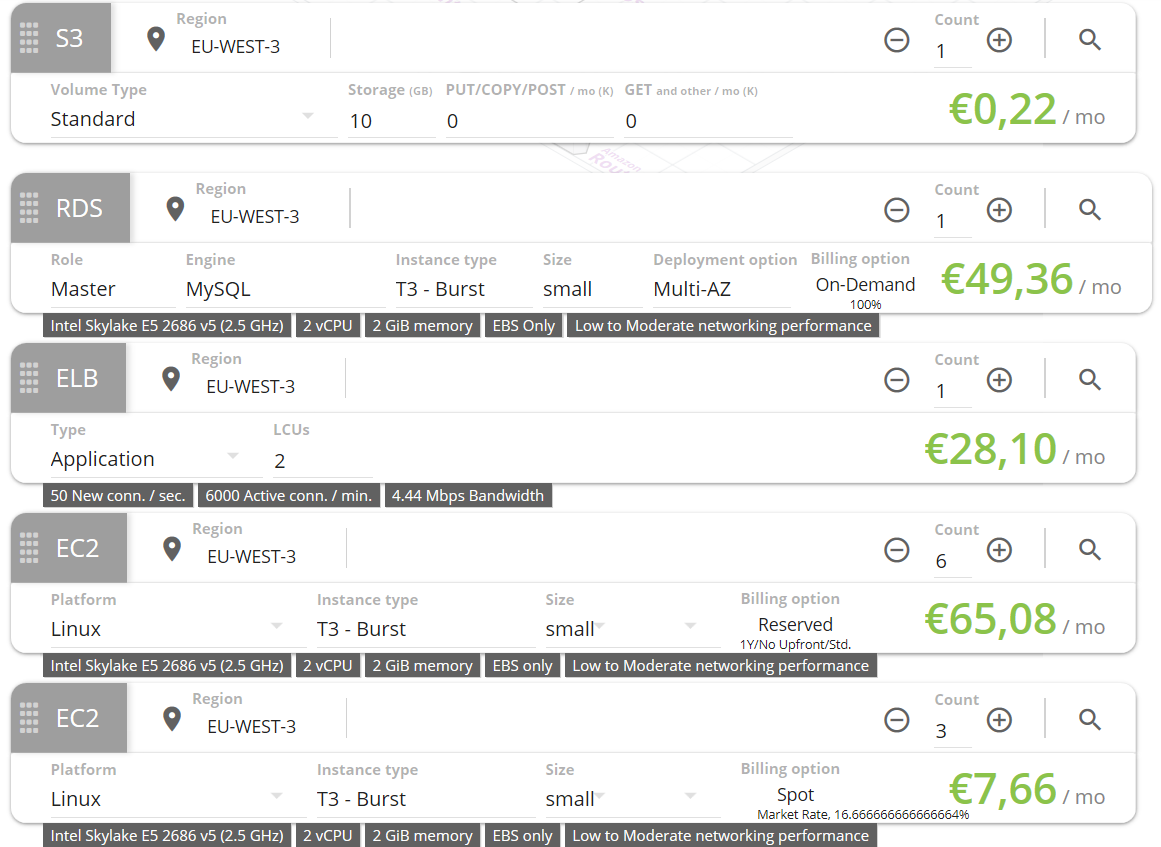
Il est préférable d'utiliser un microservice proxy pour contrôler les requêtes qui sont envoyées à notre application et ainsi la prémunir des attaques malveillantes. Cela permet aussi de découpler le système de l'extérieur et permettre ensuite de le changer si nécessaire dans le futur sans perturber l'utilisateur.

### Représentation de l'architecture.



### Evaluation des coûts avec Cloudcraft.





# Décomposition de l'application.

On scinde l'application en microservices qui sont peu couplés les uns aux autres. Suivant la montée en charge on pourra cibler les APIs à dupliquer pour pouvoir répondre à la demande. On peut faire un premier découpage en dédiant un microservice par package de la spécification fonctionnelle.

## Microservice commande.

C'est la partie utilisée pour la prise de commandes soit par le Client ou l'Employé. C'est lui aussi qui permet de faire évoluer le statut au cours de la vie d'une commande suivant les intervenants. C'est celui qui sera dupliqué pour répondre à la demande croissante des clients.

## Microservice stock.

C'est par se microservice que toutes les modifications et consultations de stock passent. Il modifie la base de données du stock lors de la validation d'une commande où la réception d'une commande fournisseur. C'est lui aussi qui permet de savoir si la quantité d'un produit est suffisante pour être vendu.

## Microservice client.

C'est par lui que transitent toutes créations, modifications, consultations et suppressions de client. Il renvoie les données de livraison du client lors de la création d'une commande. Il permet d'accéder à des informations sur les anciennes commandes des clients à but statistique ou marketing.

## Microservice gestion.

Il regroupe tous les indicateurs et les informations sur les ventes, les coûts de revients, les profits, les pertes pour faire des statistiques. Il permet aussi d'effectuer de la gestion administrative du magasin avec la production de documents pro-formats.

## Microservice authentification.

C'est le service indispensable pour gérer les accès au système pour les Clients et les Employés d'OC Pizza.

## Description de l'architecture

Le choix du prestataire s'est orienté vers Amazone qui offre de nombreux services et outils. Il permet en plus de travailler par zone géographique dans le but de développer progressivement l'activité au niveau mondial.

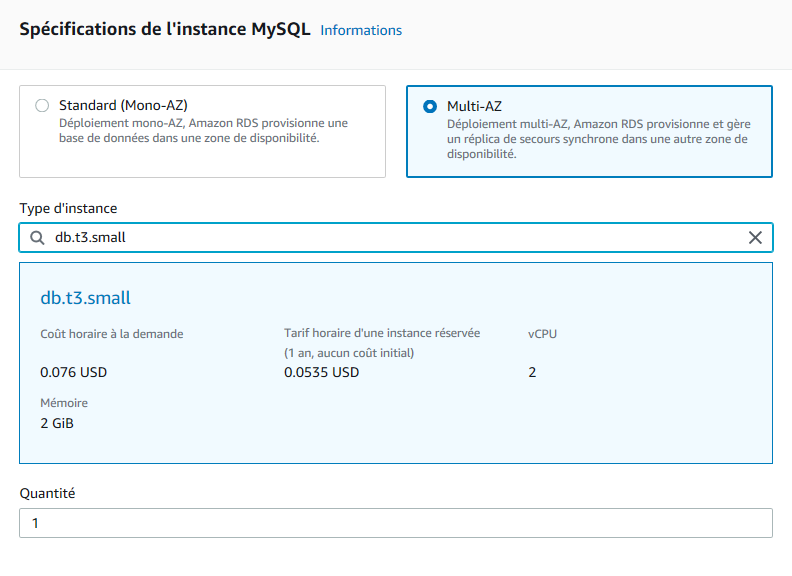
Le site **aws.amazon.com** nous permet de calculer les coûts de déploiement sur le cloud d'Amazone suivant les options choisies.

### Base de données.

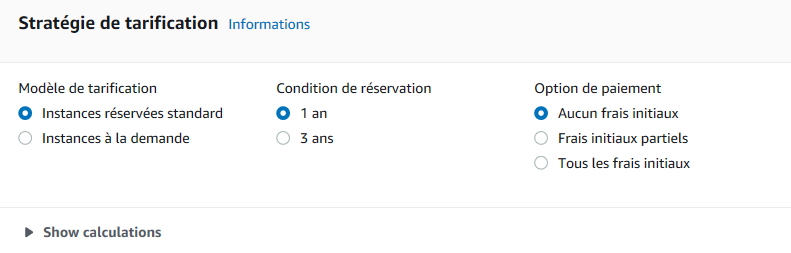
Le service **Amazon RDS (Relational Database Service)** permet de configurer les besoins en base de données en utilisant les plus grands **SGBD-R** mais aussi des services de migration et de réplication. Le service **RDS** permet d'accompagner la croissance de notre base de données et la faire évoluer facilement.

La gestion du service se fait soit par la console **RDS** **Management Console**, par la ligne de commandes ou par des appels **API**. On spécifie les détails de notre instance **MySQL** sur **AWS**.

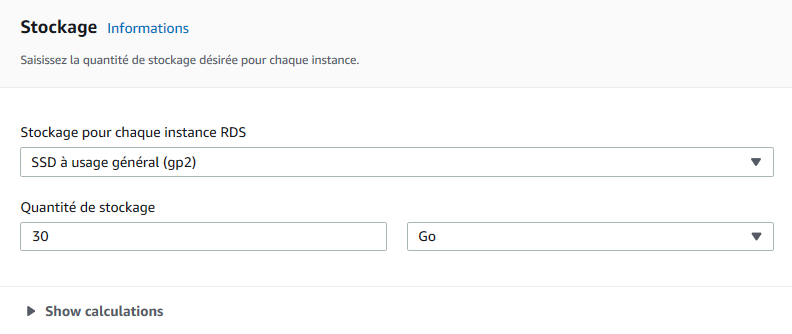
* On prend l'option **Multi-AZ** pour avoir une réplication synchrone de la base dans une autre zone par sécurité. Cela augmente la latence des transactions mais les données sont protégées. On pourra toujours ajouter par la suite des réplications en lecture de la base pour accroitre la capacité de réponse de la base de données.
* Une instance de type **db.t3.small** suffit au départ pour notre base de données. On pourra par la suite augmenter la taille si nécessaire.
* On prend donc une instance qui est en fait constituée d'une base de données principale et une secondaire de secours.



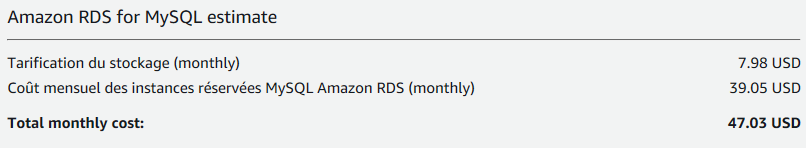
* On réserve l'instance pour une année pour réduire les coûts et permettre de réajuster au bout d'un an de production le choix d'hébergement de la base de données.
* On pourra faire une réservation sur 3 ans quand la configuration sera stable et en période de non expansion du nombre de magasin.
* Pour ne pas accroître les coûts de départ, on coche l'option "Aucun frais initiaux".



* On utilise un stockage de type Solid State Disk qui sont plus performant que des disques mécaniques.
* Une capacité de 30Go est largement suffisante pour contenir la base pour plusieurs années.



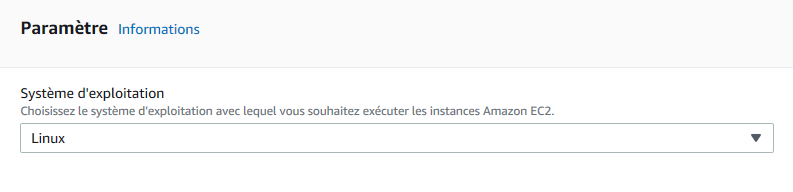
* Le coût total pour la base de données seule est de 47,03$ soit 42,44€.

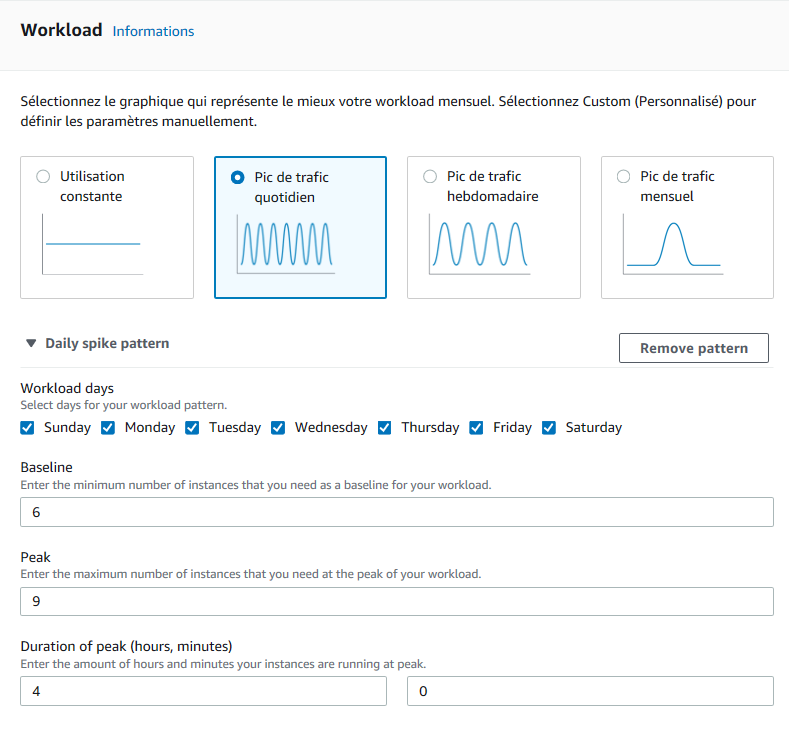


### Application.

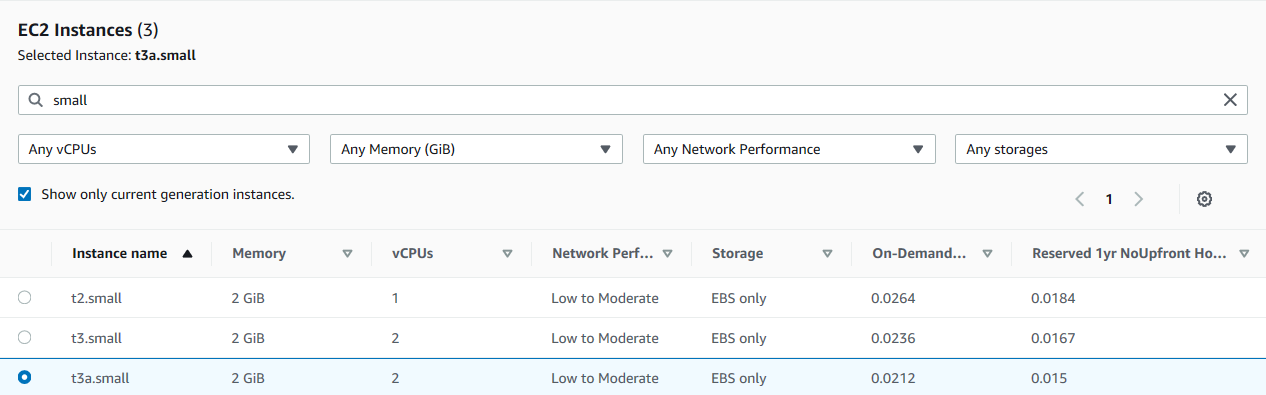
On a besoin d'un serveur pour faire fonctionner les microservices. On choisit un contrat flexible qui permet d'accroitre le nombre d'instance de 2 à 5 pour répondre à la demande aux heures de pointes. On utilise le service **Amazon EC2** (Elastic Cloud) et un contrat sur une année pour permettre de le modifier si nécessaire après une année de fonctionnement.

* On utilise le système d'exploitation **Linux** pour exécuter les instances de nos microservices.
* On paramètre 6 instances par défaut qui fonctionnent 24/24 et 7/7.
  + 1 x API Authentification.
  + 1 x API Gestion infocentre.
  + 1 x API Stock.
  + 1 x API Clients
  + 2 x API Commande qui est plus sollicité.
* On rajoute 3 instances pendant 4 heures pour les heures de fortes activités le soir pour répliquer les API Commande.

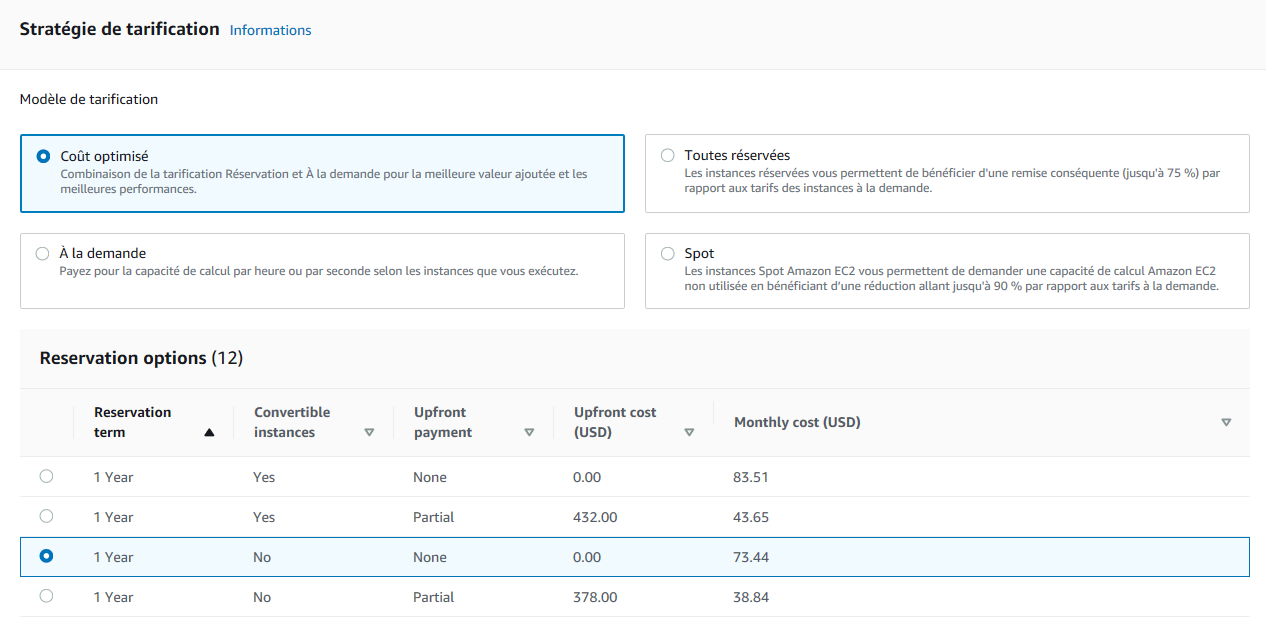




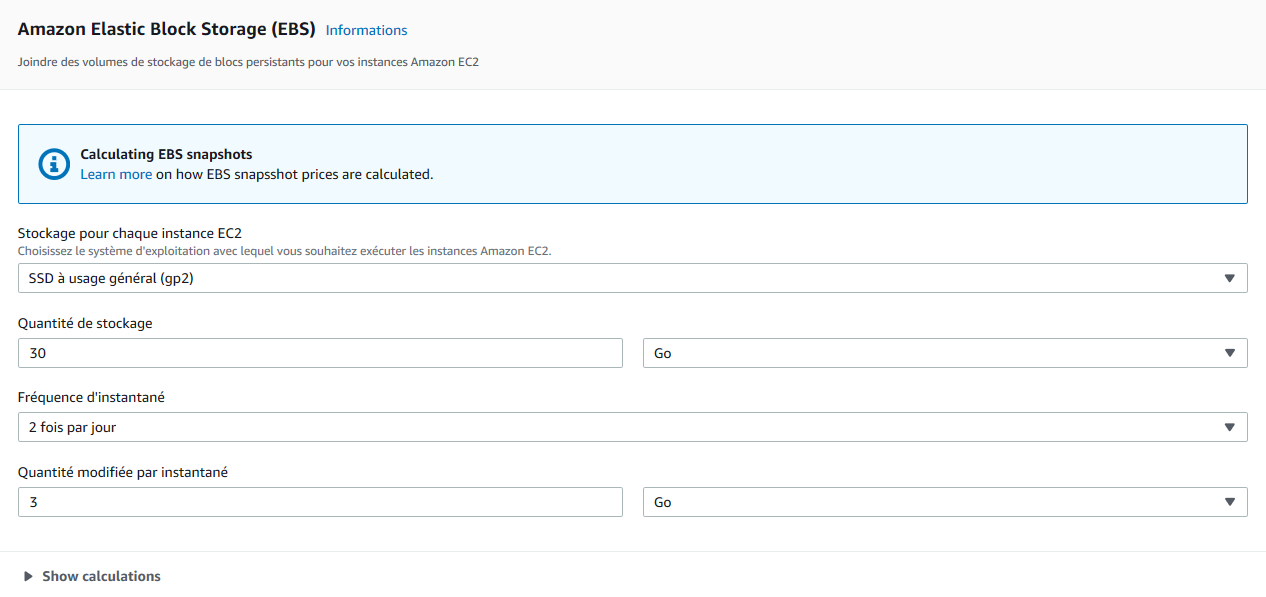
* On utilise des instances **t3a.small** qui suffisent en capacité de mémoire et de cpu pour nos microservices.



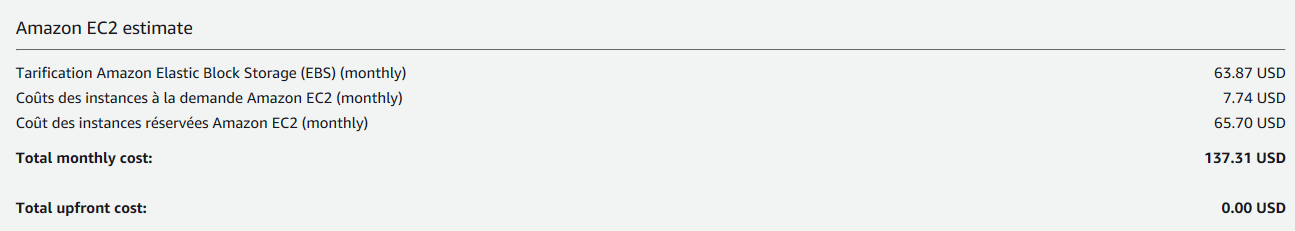
* On prend l'option de coût optimisé et une réservation de 1 an pour pouvoir modifier la configuration après une années d'exploitation.



* 30Go pour chaque instance des **EC2** suffit largement pour contenir le système d'exploitation et un microservice.
* On utilise l'option **EBS** (Elastic Block Storage) pour pouvoir mutualiser le stockage des instances **EC2**.

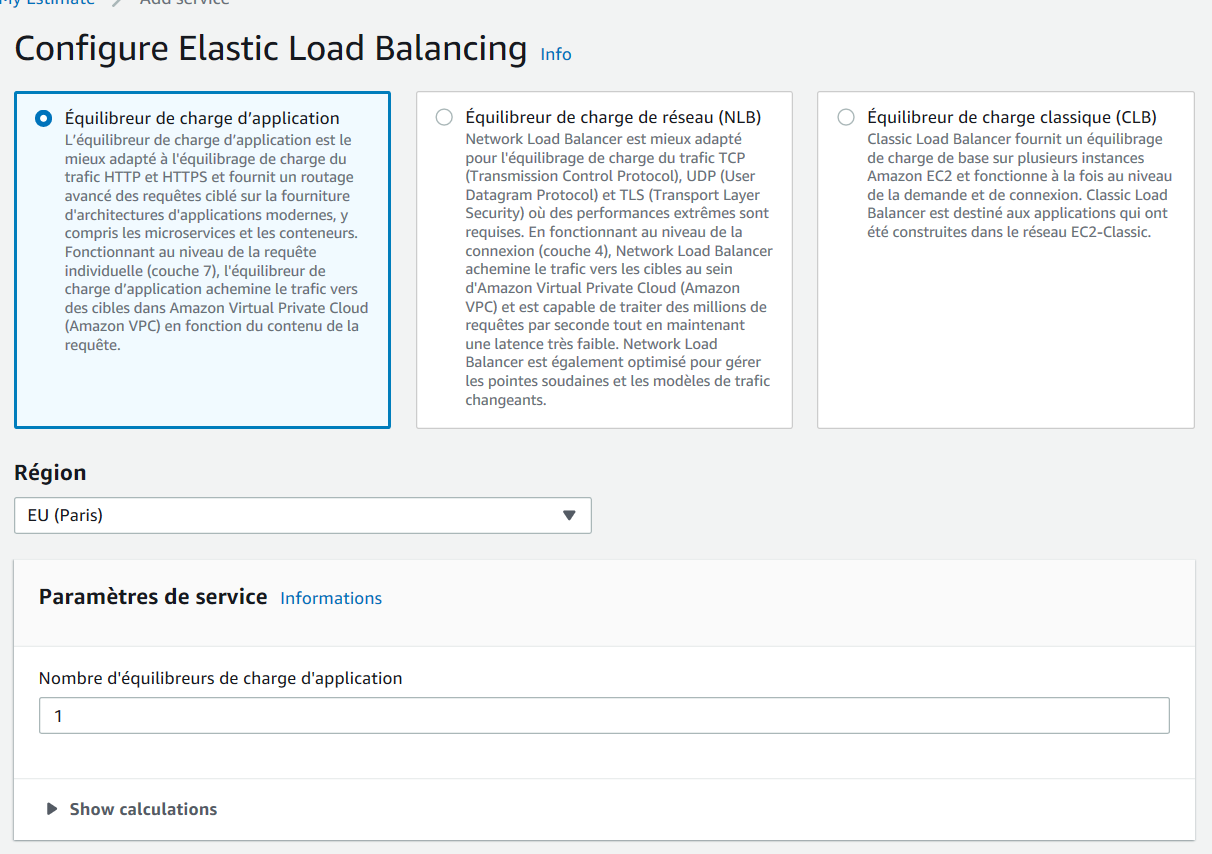


* Le coût total pour les microservices est de 137,31$ soit 123,92€.



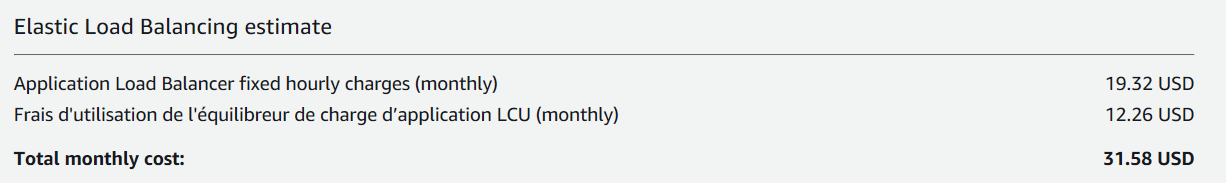
### Load balancer.

On utilise **Amazon ELB** (Elastic Load Balancer) pour répartir la charge sur les différentes instances des microservice, notamment entre les API Commande. Son accès est sécurisé par le service **Amazon IAM** (Identity and Access Management) qui est compris dans le package.





* Le coût total pour la base de données seule est de 31.58$ soit 28,50€.



### CDN CloudFront

**Amazon CloudFront** est un réseau de diffusion mondial rapide de contenu. Dans la version gratuite de base il comprend 50Go de transfert sortant et 2 millions de requêtes HTTP/HTTPS. Il accélère la distribution des contenus des sites web statiques et dynamiques, tels que des fichiers .html, .css, .js, multimédias et image, aux utilisateurs.

Il est intégré aux services **Amazon EC2** et **Amazon ELB** et fonctionne de façon transparente.

### Stockage global

On utilisera **Amazon S3** (Simple Storage Service) pour stocker quelques fichiers de configuration qui seront utilisés de manière globale par l'application. Il servira à placer les données qui n'ont pas besoin d'être sécurisées comme les fichiers .html, .css… On n'a pas besoin d'une grande quantité de stockage donc son coût est négligeable dans notre étude.

### Système DNS Route 53

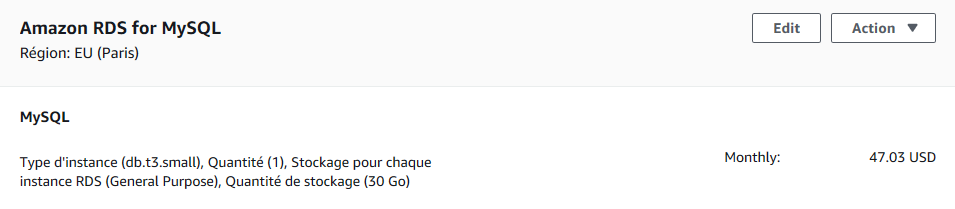
**Amazon Route 53** permet de faire le routage de façon optimale des utilisateurs vers notre application en garantissant la haute disponibilité. On peut gérer facilement l'acheminement des utilisateurs vers les microservices de notre application.

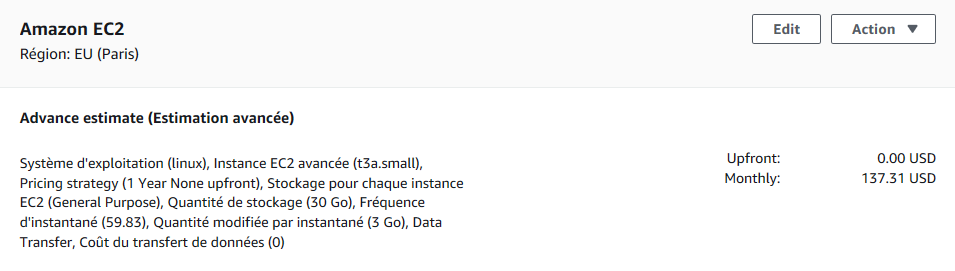
Il offre la possibilité de switcher sur une autre route à la volée si l'on a plusieurs serveurs en cas de panne ou de maintenance sur l'un d'eux.

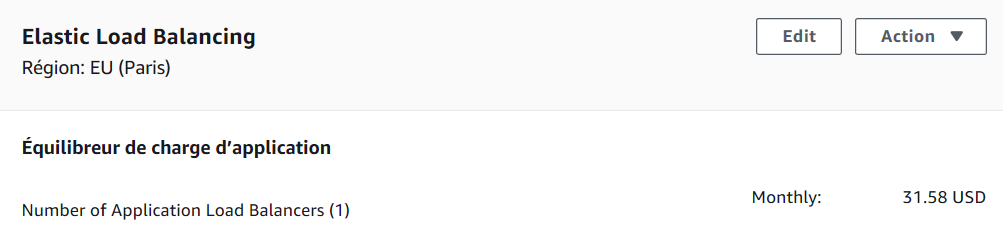
### Récapitulatif.

|  |  |
| --- | --- |
| Service Amazon | Quantité |
| Relational Database Service (RDS) for MySQL multi A-Z | 1 |
| Elastic Cloud (EC2) | 6+3 |
| Elastic Load Balancier (ELB) | 1 |
| CloudFront | 1 |
| Simple Storage Service (S3) | 1 |
| Route 53 | 1 |
| Identity and Access Management (IAM) | 1 |

On arrive à un total annuel de 2593,76$ soit **2 340,93€** ou encore mensuellement 216.15$ soit **195.07€**. D'autres possibilités existent comme **Microsoft Azure**, **Google Cloud** ou **Alibaba** qu'il serait possible de comparer dans une étude plus poussée de la solution technique qui n'est pas l'objet de ce document.









# Table des matières.

[Choix de l'architecture. 2](#_Toc17994717)

[Architecture Monolithique. 2](#_Toc17994718)

[Architecture microservices. 3](#_Toc17994719)

[Scalabilité. 3](#_Toc17994720)

[Maintenance. 3](#_Toc17994721)

[Encapsulation. 3](#_Toc17994722)

[Proxy. 4](#_Toc17994723)

[Décomposition de l'application. 7](#_Toc17994724)

[Microservice commande. 7](#_Toc17994725)

[Microservice stock. 7](#_Toc17994726)

[Microservice client. 7](#_Toc17994727)

[Microservice gestion. 7](#_Toc17994728)

[Microservice authentification. 7](#_Toc17994729)

[Description de l'architecture 8](#_Toc17994730)

[Base de données. 8](#_Toc17994731)

[Application. 10](#_Toc17994732)

[Load balancer. 13](#_Toc17994733)

[CDN CloudFront 15](#_Toc17994734)

[Récapitulatif. 16](#_Toc17994735)

[Table des matières. 18](#_Toc17994736)