My Booking Service

Infrastructure	**honora_j**
CI/CD	**menneg_l**
Développement & tests	**mahtal_m**
Développement & tests	**feldma_l**

Dans une optique d'une équipe de développement rapide et efficace, nous avons mis en place une méthodologie Agile. Cette dernière était calquée sur des sprints de trois semaines. Des dailys étaient prévus à 10h avec celui du lundi transformé en weekly plus conséquent et complet. Le weekly suivant la fin d'un run était quant à lui une plus grande réunion pour faire le point sur les deadlines, points à revoir et améliorer mais aussi pouvoir s'encourager pour le run futur et se féliciter des réussites du run passé.

Sur et avec l'aide de GitLab, nous avons défini des issues taguées pour permettre un classement de priorité en amont et nous aider dans la priorisation des tâches. Avec ceci, une politique de branche a été mise en place pour empêcher les push sur main afin de s'assurer de la cohérence des modifications et de la qualité du code. Des Merge Request depuis une branche annexe ont dû être réalisées et revues par les paires avant de pouvoir être push sur main. Les commits ont été, eux aussi, en accord avec les issues, tagués pour permettre une meilleure traçabilité.

Nous avons également intégré des agents de CI/CD avec GitLab pour avoir une pipeline de construction d'image Docker, de publication sur un registre de conteneurs, puis un déploiement avec des charts Helm dans le cluster Kubernetes. En aval de cette pipeline, des tests d'intégration ont été lancés à partir de Postman. Une fois ces derniers validés, une seconde campagne de tests end-to-end a été lancée, toujours depuis un runner Postman.

Un certain nombre d’outils de mis en production ont été utilisés tels que :

- Configuration de l’infra avec Ansible
- Ansible Vault pour la sécurisation des données
- Pipeline GitLab pour la CI/CD
- Des conteneurs Docker pour les applications (API & databases)
- Kubernetes pour l’ordonnancement des conteneurs (appelés pods)
- Gestion du trafic avec Ingress NGINX Controller
- Stack TICK (Telegraf, InfluxDBv2) pour le logging

voici les raisons qui nous ont poussés à choisir ses technologies:

-Choix de Ansible & Ansible Vault:
Ansible permet d'automatiser l'ensemble du cycle de vie d'une infrastructure, de la configuration initiale à la gestion continue; il permet de regrouper l'ensemble du cycle de vie d'une infrastructure dans des playbooks, qui décrivent les tâches à effectuer pour chaque étape. Cela facilite l'automatisation des tâches répétitives, la cohérence des configurations et la gestion efficace de l'infrastructure à grande échelle.
Il peut être utilisé dans les pipelines CI/CD pour automatiser le déploiement des microservices à chaque étape du processus de développement et de déploiement. Cela permet une livraison continue et une intégration transparente des microservices dans l'environnement.
Ansible Vault à la capacité de chiffrer les données sensibles, garantissant ainsi leur sécurité, et la possibilité de les intégrer de manière transparente dans les playbooks et les fichiers de configuration Ansible, simplifiant ainsi la gestion des informations confidentielles au sein de l'infrastructure automatisée.

-Choix de la Stack TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) pour le logging:
Elle permet une gestion des logs complets (collecte, visualisation, stockage et analyse).
Elle permet aussi une collecte facile des données provenant de différentes sources (système d’exploitation, applications et services, infra réseau, microservices etc.), elle offre d'autres avantages mais en ce qui nous concerne sachant que nous sommes dans un environnement de conteneurisation, elle permet une gestion centralisé des logs.

-Choix de Ingress NGINX Controller pour la gestion du trafic:
Ingress NGINX Controller se distingue des autres technos grâce à sa haute performance, sa flexibilité et sa personnalisation des règles de routage, son intégration avec l'écosystème, sa prise en charge étendue des protocoles, sa scalabilité et sa disponibilité. Ces fonctionnalités font de NGINX un choix solide pour la gestion du trafic dans une architecture microservices.

-Choix de Kubernetes:
Déjà premièrement c'est une technologie plus utilisée que docker swarm ce qui fait de lui une technologie plus mature, il est devenu la norme pour l'orchestration des conteneurs, il détient des fonctionnalités avancées que ce soit en flexibilité, gestion des ressources etc.

-Choix de docker pour la conteneurisation:
En raison de son large adoption, de sa facilité d'utilisation, de sa portabilité, de son isolation sécurisée et de sa gestion efficace des versions et des dépendances on a utilisé docker.

Pour le développement des API, elles sont toutes en Java avec le framework de création d'API Spring Boot. Les tests unitaires vont de paires en restant en Java et l'intégration de Swagger permet aux API d'être documentées et d'avoir une interface graphique pour les agrémenter. L'utilisation du Java et son framework Spring Boot est une évidence tant son efficacité n'est plus à prouver, à laquelle on peut rajouter le fait que sa vaste utilisation en fait un outil très bien documenté et facile à utiliser.

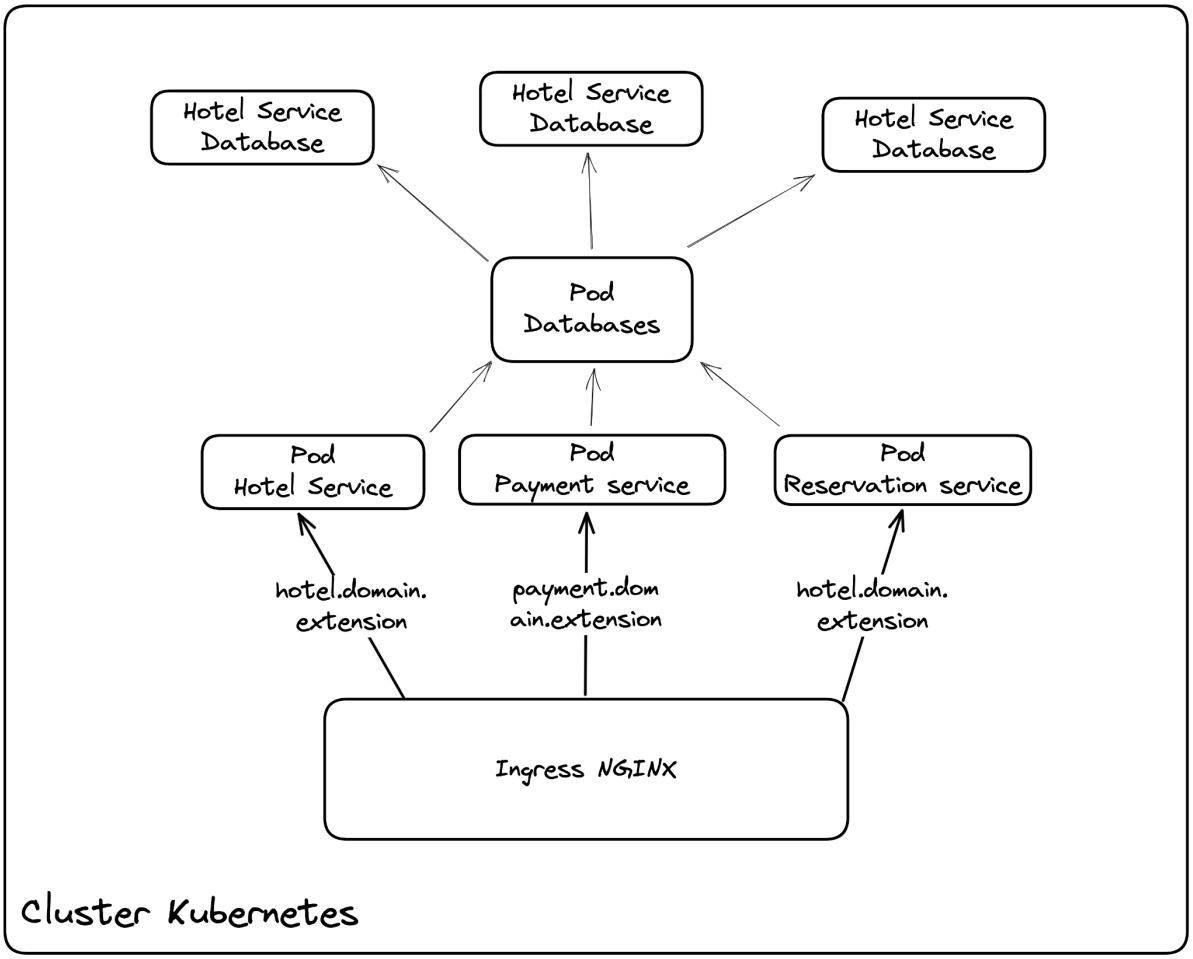
À ces API s'ajoutent des bases de données MySQL. Nous avons choisi cette technologie comme étant facile d'installation, de configuration et d'utilisation. Se basant sur une architecture en micro services, nos requêtes sont assez simples, cas d'usage où MySQL est le plus performant. Il a été vraiment évident pour nous de nous tourner vers cette solution d'autant plus que nous l'avions déjà tous utilisé et étions à l'aise avec.

Concernant le découpage de l'application en microservices, voici ce à quoi nous pensons :

- Service de gestion des hôtels : gestion de toutes les fonctionnalités liées à la gestion des hôtels, notamment l'ajout de nouveaux hôtels, la mise à jour des informations de l'hôtel, la gestion des chambres.

- Service de réservation : gestion de toutes les fonctionnalités liées à la réservation de chambres, y compris la recherche de disponibilité, l'annulation et la confirmation de la réservation.

- Service de gestion des paiements : gestion de toutes les fonctionnalités liées aux paiements, notamment la tarification, le traitement des paiements, la gestion des remboursements et la facturation.



Voici l'infrastructure schématisée du projet final tel qu'il a été imaginé dans un premier temps. Les pods d'API sont stateless tandis que les pods des databases sont stateful avec leurs volumes respectifs.

Pour la répartition des tâches, nous faisons en fonction des compétences de chacun. En effet :

Jean-Baptiste est orienté Ops. Il s'est concentré sur la configuration du cluster Kubernetes et des différents ingress. Il a aussi été chargé des stratégies de déploiement.

Loïc est développeur avec une appétence pour l'Ops, il apporte ses connaissances sur la CI/CD et apporte du soutien sur la mise en place de l'infrastructure. Il est le

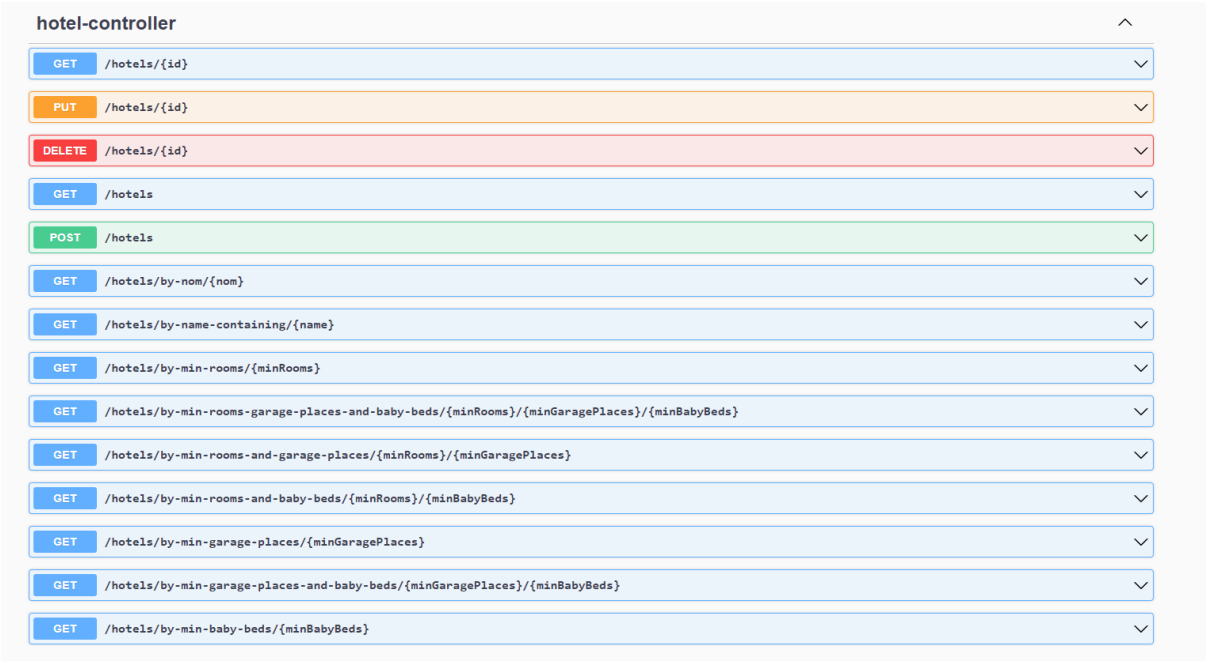
pont principal entre les membres de l'équipe. Il a notamment aussi entièrement développé l'api de réservation.

Louis est développeur back-end Java. Il a permis de facilement détecter et remonter les anomalies et faire appliquer les bonnes pratiques de développement. Il a été chargé du développement de l'api de facturation.

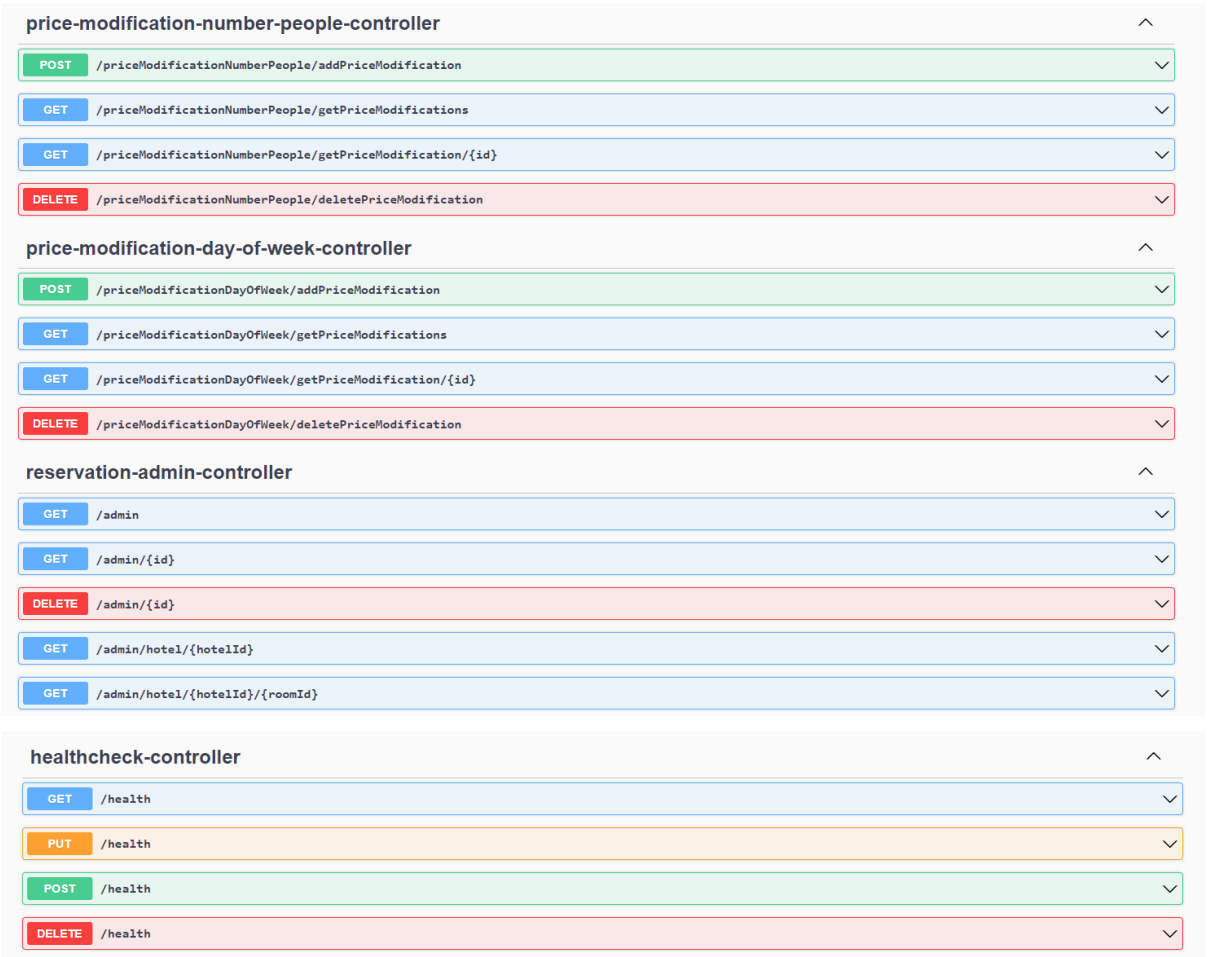
Mohamed est développeur Full-Stack. Il s'est chargé du développement de l'api de gestion des hôtels. Il a aussi grandement aidé pour le développement des tests autos sur l'ensemble des services.

Documentation des API :

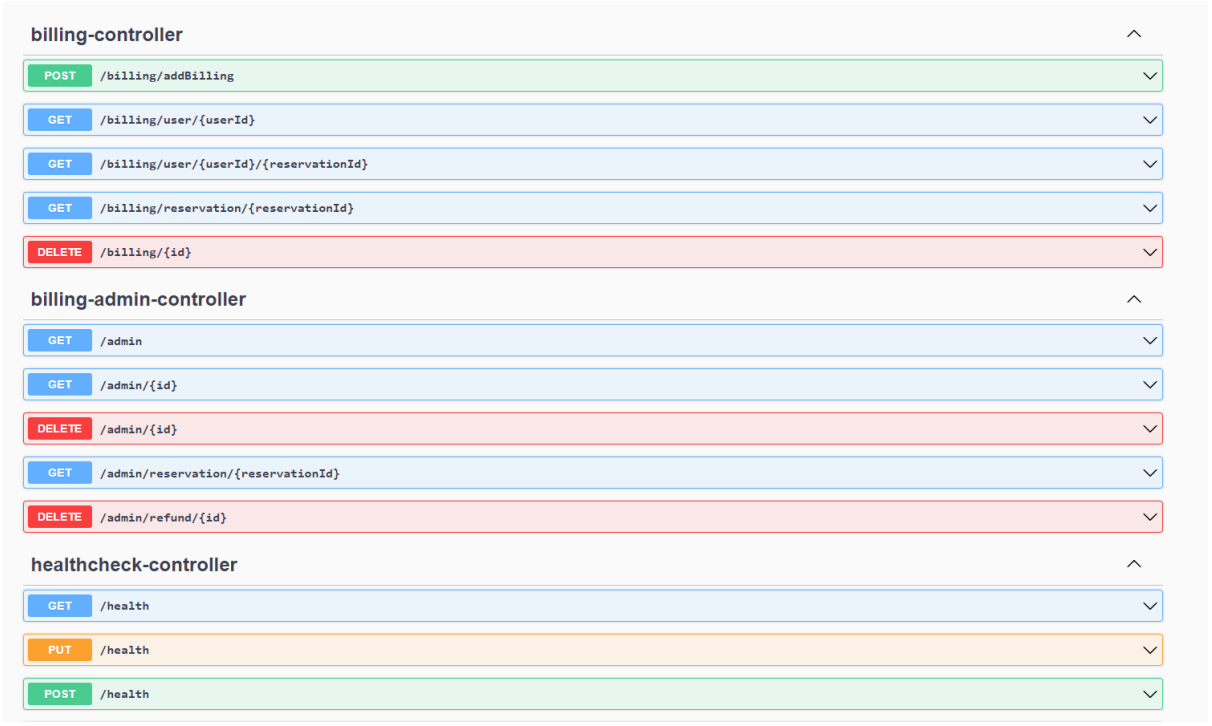
HotelApi :



Reservation Api :


price-modification-number-people-controller
POST /priceModificationNumberPeople/addPriceModification
GET /priceModificationNumberPeople/getPriceModifications
GET /priceModificationNumberPeople/getPriceModification/{id}
DELETE /priceModificationNumberPeople/deletePriceModification
price-modification-day-of-week-controller
POST /priceModificationDayOfWeek/addPriceModification
GET /priceModificationDayOfWeek/getPriceModifications
GET /priceModificationDayOfWeek/getPriceModification/{id}
DELETE /priceModificationDayOfWeek/deletePriceModification
reservation-admin-controller
GET /admin
GET /admin/{id}
DELETE /admin/{id}
GET /admin/hotel/{hotelId}
GET /admin/hotel/{hotelId}/{roomId}
healthcheck-controller
GET /health
PUT /health
POST /health
DELETE /health

BillingApi:


billing-controller
POST /billing/addBilling
GET /billing/user/{userId}
GET /billing/user/{userId}/{reservationId}
GET /billing/reservation/{reservationId}
DELETE /billing/{id}
billing-admin-controller
GET /admin
GET /admin/{id}
DELETE /admin/{id}
GET /admin/reservation/{reservationId}
DELETE /admin/refund/{id}
healthcheck-controller
GET /health
PUT /health
POST /health