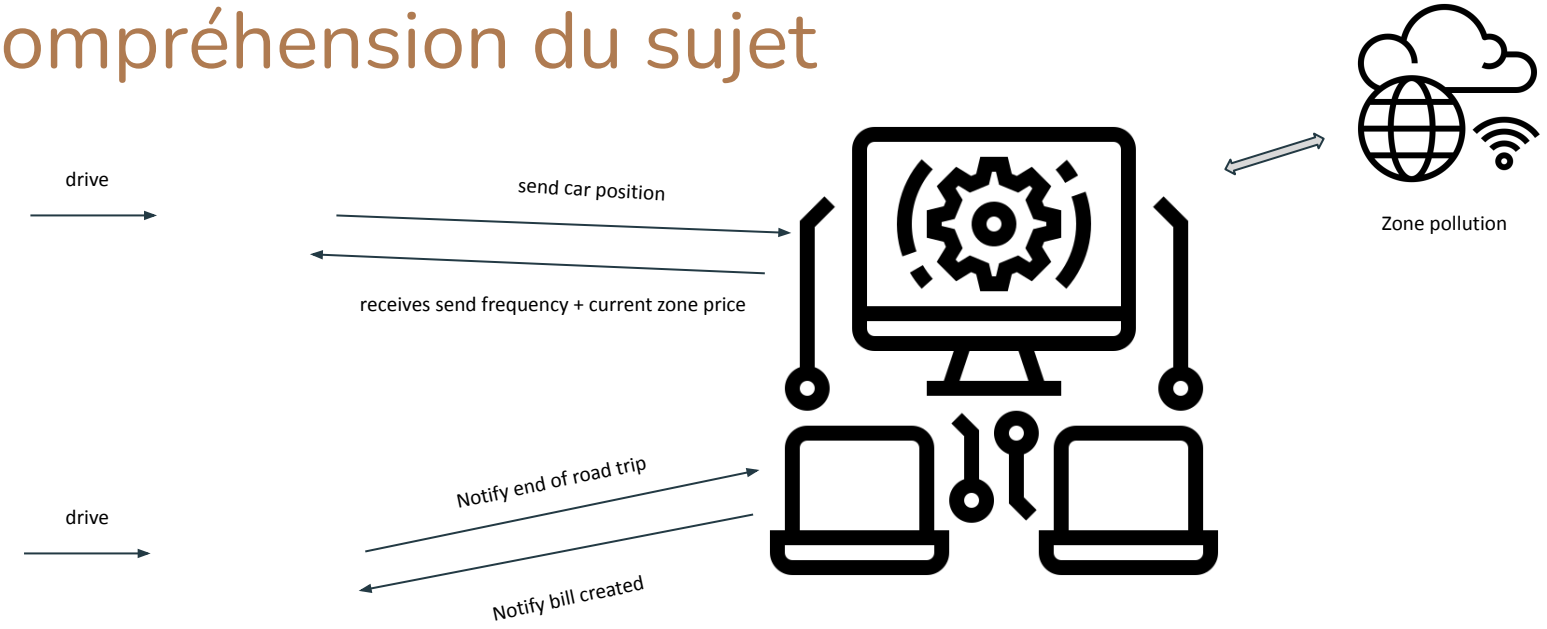
Architecture Logicielle : Construction

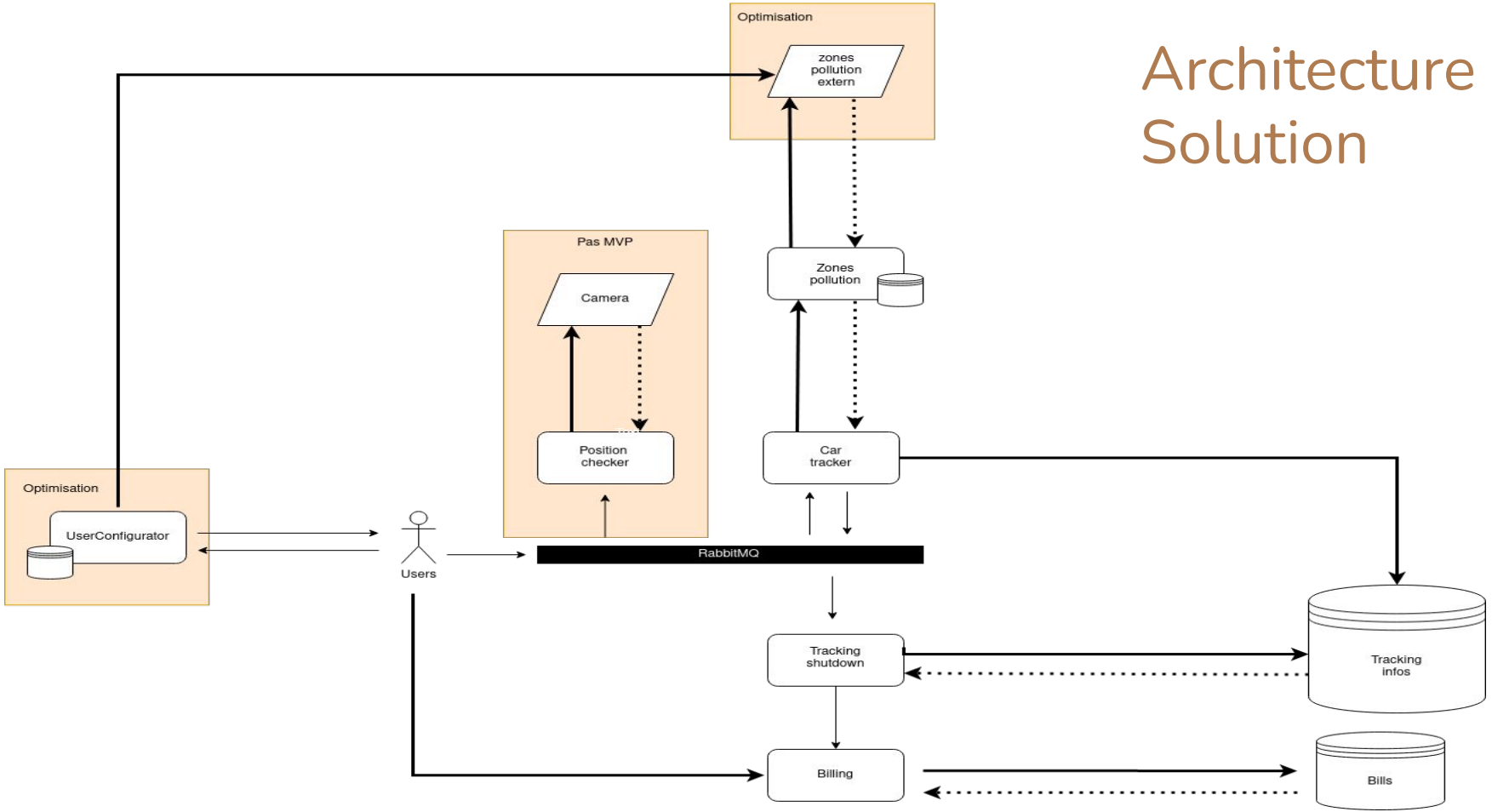
V6 : Pay as you pollute

Equipe B : Guillaume Piccina, William D'Andrea, Nicolas Fernandez, Yann Brault

Compréhension du sujet



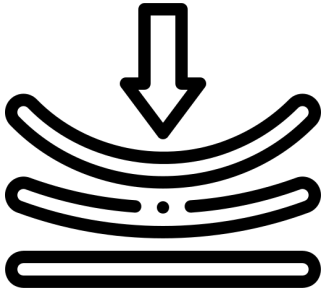
Architecture : Solution



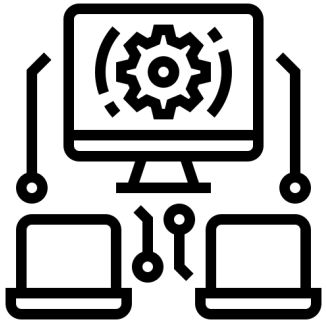
Problème et caractérisation

- Le système doit être disponible 24h/24, 7j/7
- Potentiellement des milliers de voitures qui envoient leur position en même temps.
- Les données de positions ne doivent pas être perdues ni corrompu
- L'utilisateur ne doit pas pouvoir tricher
- La facture doit refléter exactement le trajet effectué (pas de perte de données)

-> Problématiques : Volumétrie et résilience

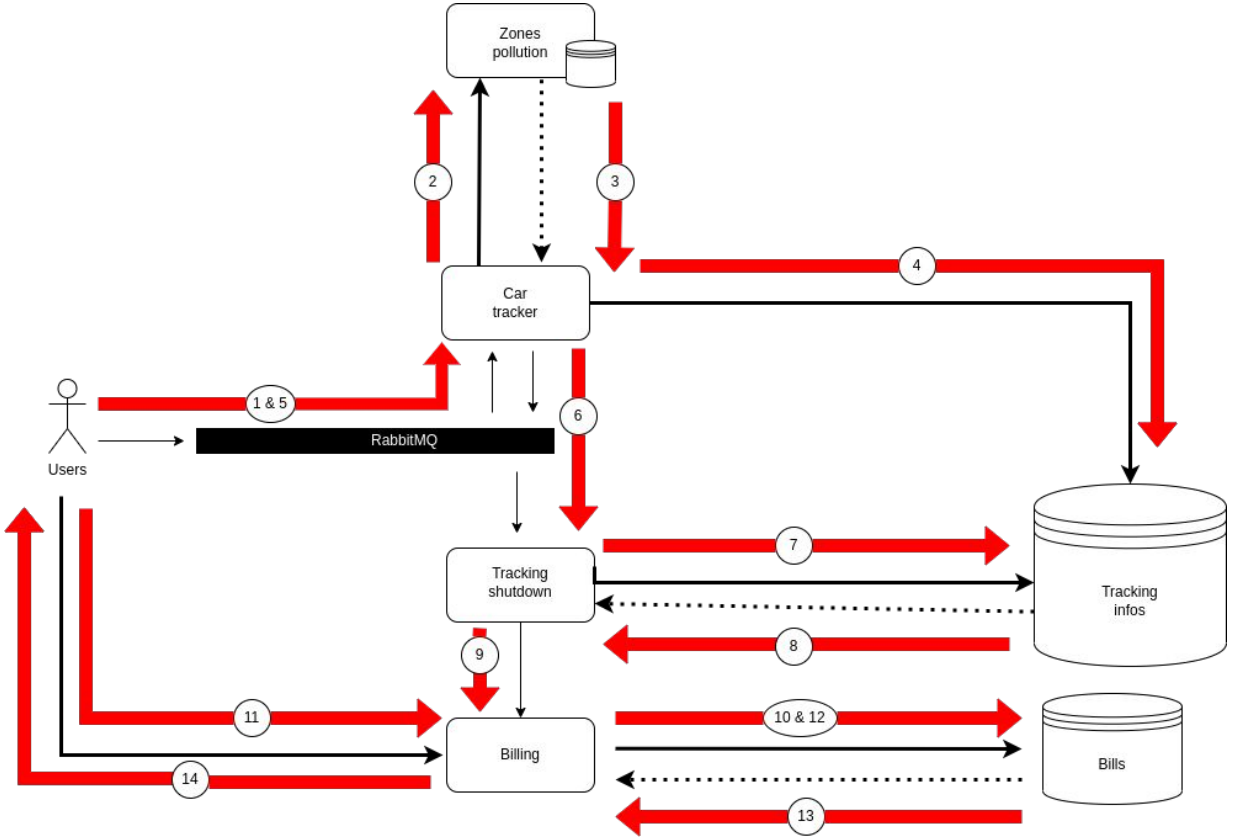


Choix technologiques



RabbitMQ

Scénario de démonstration



Justification des rôles

- L’application utilisateur doit rester simple : pratiquement pas de logique métier, seul rôle : envoyer la position de l’utilisateur toutes les n secondes
- Utilisation de RabbitMQ pour gérer un gros volume de message
- Le rôle de Car tracker est simple et il est également stateless car il doit pouvoir être facilement scalable en cas de forte montée en charge et pareil pour Zones pollution
- Car shutdown réalise la récupération des positions en DB et le calcul de la facture
- Billing est un service à part avec sa propre DB pour que la possibilité de voir ses factures soit disponible même quand le reste du système n’est pas disponible

Scénario : hypothèses que nous avons fait

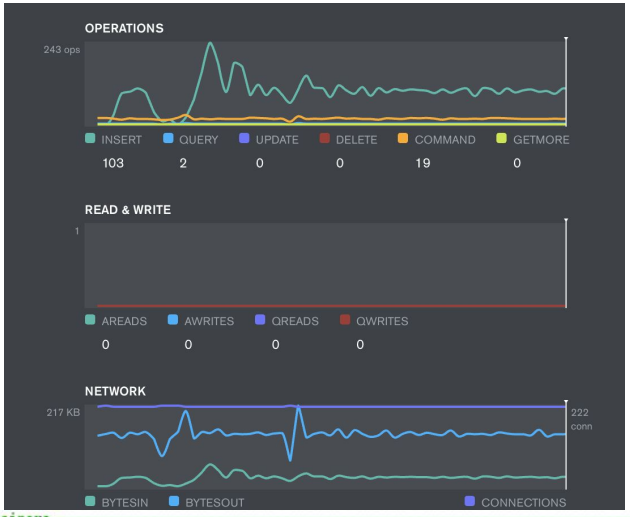
- POINT NOIR : Pas de retour en temps réel pour l'utilisateur, il envoie juste sa position
- Pas "d'application" pour l'utilisateur, nous avons créer un service qui simule une voiture et qui envoie la position dans le bus (on crée N threads qui envoient chacun une position tout les X secondes)
- Les zones de pollution ont déjà été récupérées auparavant auprès d'un service externe

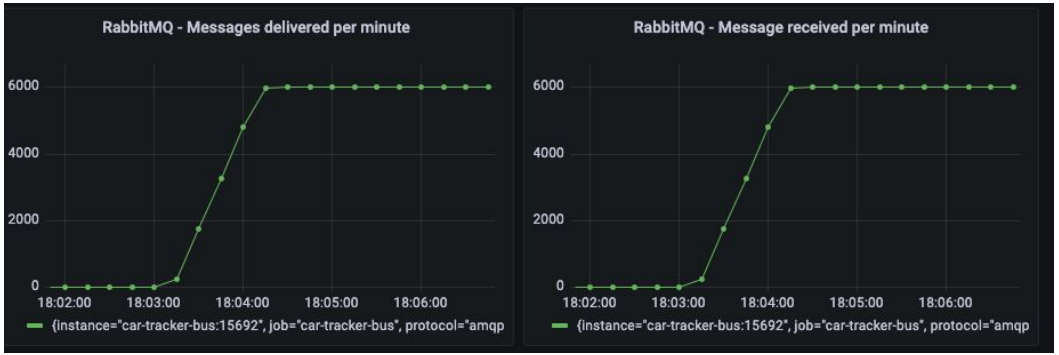
Montée en charge du système

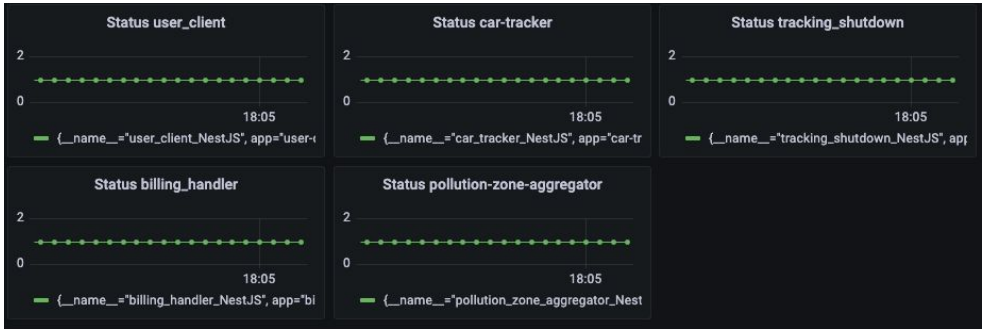
Qu'avons nous implémenté pour mesurer la résilience du système ?
=> Script python qui crée N voitures
=> Prometheus + Grafana

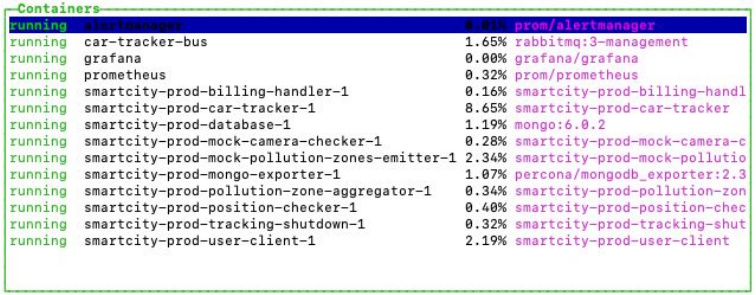
Montée en charge du système - Test

100 voitures qui envoient leur position chaque seconde :









Montée en charge du système - Test

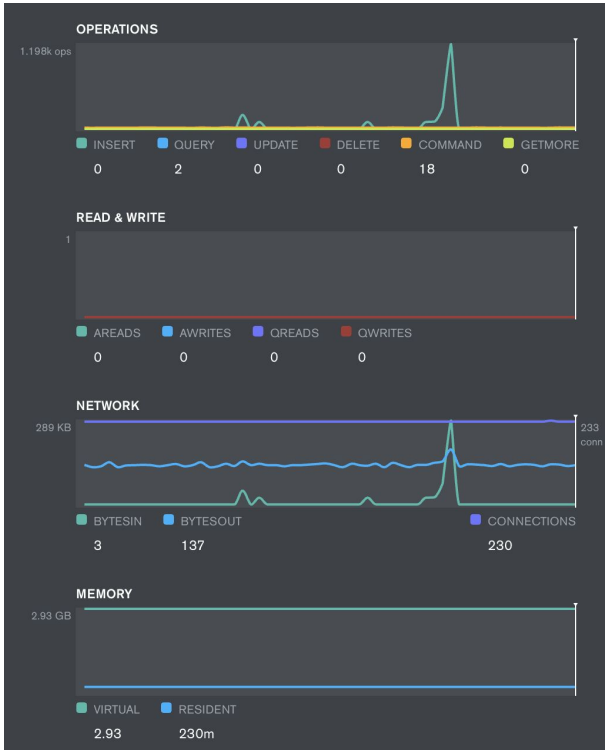
100 voitures qui envoient leur position chaque seconde :

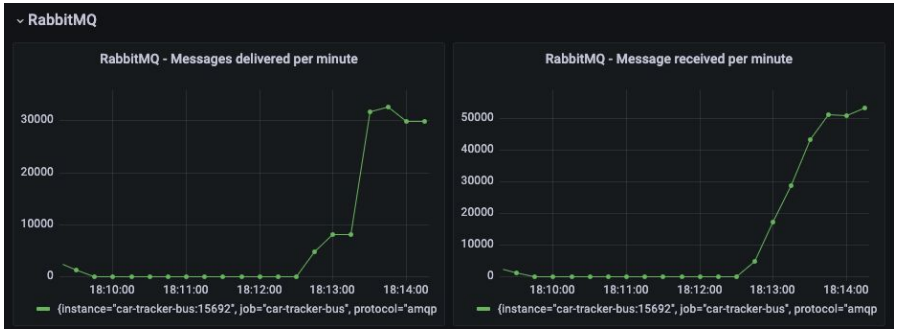

```
{
    "_id": "6372761c80b06238961ba39c",
    "license_plate": "80",
    "price": 16500,
    "is_paid": false,
    "__v": 0
},
{
    "_id": "6372761c80b06238961ba3a2",
    "license_plate": "90",
    "price": 16500,
    "is_paid": false,
    "__v": 0
},
{
    "_id": "6372761c80b06238961ba3a4",
    "license_plate": "86",
    "price": 16500,
    "is_paid": false,
    "__v": 0
},
{
    "_id": "6372761c80b06238961ba3ae",
    "license_plate": "99",
    "price": 16500,
    "is_paid": false,
    "__v": 0
},
{
```

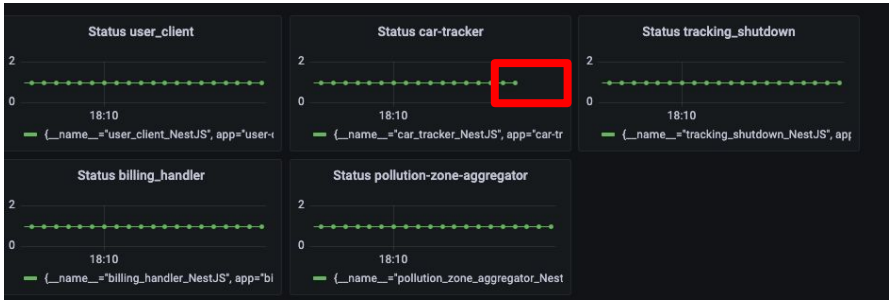

Les 100 factures sont générées, pas de problème

Montée en charge du système - Test

1000 voitures qui envoient leur position toute les secondes :

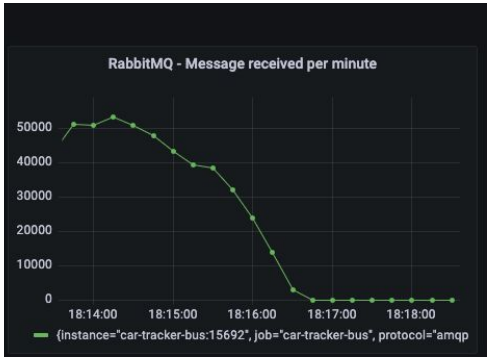


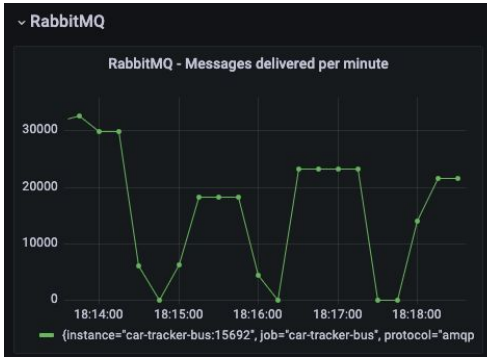




Montée en charge du système - Test

- car-tracker ne suit pas la charge -> il tombe
- il n'arrive pas à se relever (les pics), perte de donnée énorme (uniquement 365 factures / 1000)
- dès qu'il se relève, des milliers de messages l'attendent dans la Queue





Interprétation

- Avant le test, nous pensions que ce serait la base de donnée car-position qui ne suivrait pas la charge
  - On s'est rendu compte que l'on avait tort, et que car-tracker ne suit pas.
- Pourquoi ?
  - Énormément de messages du bus à traiter -> RabbitMQ est capable de largement gérer la charge mais pas car-tracker
  - Dès qu'il tombe, et qu'il redémarre, il se noie à nouveau sous la charge
- Conséquences
  - Quand il tombe, des requêtes avec la base de donnée ne sont pas faite et donc perte de data (325 factures alors qu'on devrait en avoir 1000)
  - Les positions dans car-tracker ne sont pas supprimer => 63.5k document inutiles et non traitable

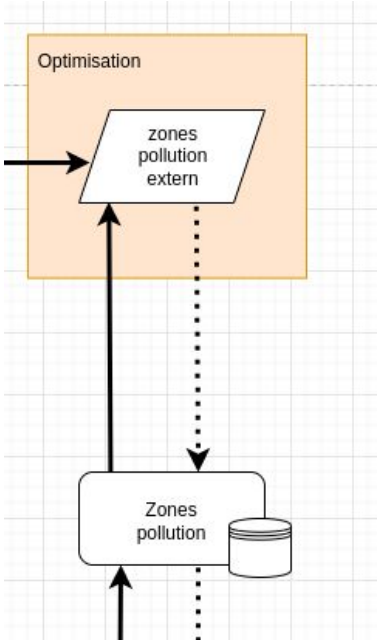
Leçons tirées et évolutions

- Leçons
  - Notre système, telle qu'il l'a été conçu n'est pas résilient, si car-tracker tombe, impossible de générer des factures cohérentes
  - Avec aucun réplica de micro-service, il est impossible que notre système tienne la charge
  - RabbitMQ (queue de messages) est pratique dans notre situation car normalement, les messages ne sont pas perdu

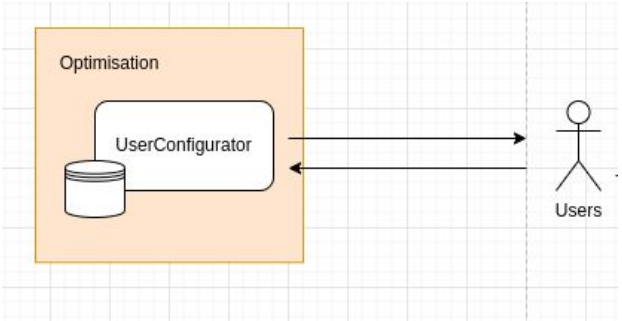
- Evolutions
  - Revoir le système de factures du point de vue métier, il faudrait qu'il puisse s'exécuter que quelques fois par jour
  - Ne plus faire passer le message tracking-shutdown par la car-tracker, mais directement l'envoyer de la voiture à tracking-shutdown
  - Implémenter des réplicats sur car-tracker, zones pollution, et tracking-shutdown
    - La base de donnée ne suivra probablement pas la charge
      - Implémenter un sharding vertical sur la MongoDB
      - Tester la capacité de charge avec une réplication + sharding
  - Implémenter le retour en temps réel de la facturation à l'utilisateur (pendant qu'il roule)
  - Implémenter un circuit breaker ou un système de back-pressure pour éviter de surcharger car-tracker avec des milliers de messages dès qu'il se relance (voir ce qui est possible de faire avec RabbitMQ)

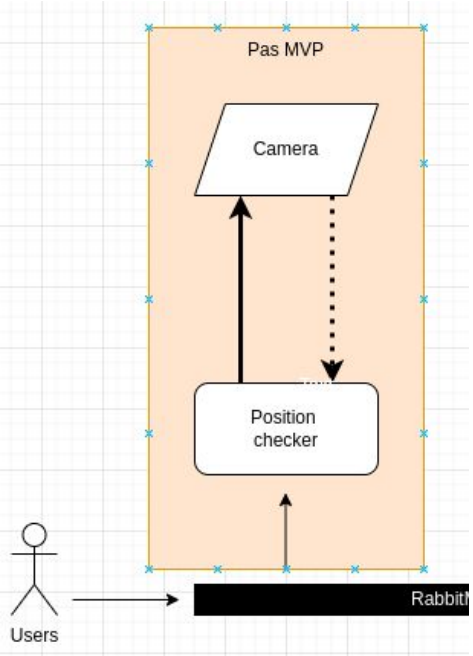
Future de l’application

Zones pollution extern



User configurator





Position checker

Merci