Dronazone

Rapport Intermédiaire

—

25 Octobre 2019

Alexandre Bolot - Grégoire Peltier - Hugo François - Prune Pillone

Sommaire

Introduction

Ce rapport a pour but de présenter l'état d'avancement de notre projet Dronazone en tant que rendu intermédiaire.

Nous aborderons dans un premier temps les choix d'interprétation du sujet que nous avons établis, pour donner un périmètre plus précis au projet, en respectant les UserStories.

Nous présenterons ensuite notre architecture en micro-services, en rappelant ce qui était présent lors du premier rendu de projet, ainsi que les modifications et nouveautés apportées durant les semaines suivantes. Dans la continuité nous ferons également une prise de recul de cette architecture.

Enfin, nous mettrons en avant un diagramme de séquence représentant un parcours de bout en bout de notre solution.

Interprétation du sujet

Quelques hypothèses ont été émises sur le fonctionnement du projet et donc sur ses fonctionnalités et limitations :

- Il n'y a qu'un seul entrepôt dont la localisation est connue.
- Le stock de l'entrepôt est illimité
- Le passage de flotte d'un drone (active, en charge etc), ne se fait pas automatiquement mais à l'action de Elena
- Le drone retient en interne sa position d'origine, où il doit retourner, par un mécanisme interne à son démarrage
- Les produits sont stockés dans le même entrepôt d'où partent les drones.

Au niveau technique, nous avons choisi d'utiliser :

- Java Spring. Largement utilisé et correspondant au besoin d'accès aux données et de création de service, ceci nous paraissait le choix idéal.
- Postgres. Référence dans les bases de données et respecte les normes.
- InfluxDB. Base de données performante et scalable, qui nous servira pour le monitoring des statistiques, car créée pour traiter un grand nombre de petites données.
- Grafana. Référence pour le monitoring et l'affichage de statistiques, peut bien s'interfacer avec InfluxDB.
- Cucumber pour la validation des User Story.

User Story couvertes

Notre application possède actuellement les fonctionnalités des User Stories 1 à 11.
Les 12 et 13 ont été mises de côté et repoussées au prochain sprint car nous avions du retard sur les tests et le test d'intégration. Ce scénario d'intégration couvre les fonctionnalités des user stories 1 à 9.

Evolution de l'architecture

Au début du projet, nous avions 3 services : Order, Warehouse et Drone. Ces services communiquaient entre eux à l'aide de WebServices REST et des MOM.
Nous avons enrichi notre architecture en prenant en compte les nouvelles user stories, tout en voulant garder une grande adaptabilité aux fluctuations de charge et une bonne répartition des responsabilité.
Pour cela, nous proposons de nouveaux services, et des enrichissements des anciens :

1. Order Service

Il est dédié au domaine de commerce en ligne. Il peut être vu comme notre chargé de relation client étant donné qu'il a pour rôle principal de recevoir les commandes des clients, de faire une demande de livraison à l'entrepôt de stockage et de notifier le client lors de l'arrivée imminente du drone de livraison. De plus, le service Order suit les états de la commande et de la livraison.
À terme, ce sera à lui de connaître les informations relatives aux clients et au catalogue.

Ce service est l'un de ceux qui devra pouvoir supporter une montée en charge (et descente de charge) la plus critique étant donné que le nombre de commandes effectuées peut varier fortement de façon imprévisible. Un client doit toujours avoir accès à notre service, sans se soucier du nombre d'autres clients qui sont en train de commander en même temps. Toutefois, il est important de prendre en considération le fait que toute montée en charge ici se répercutera sur l'ensemble du système, bien que plus diluée car Kafka permettra d'empiler les messages le temps qu'ils soient traités par le Warehouse Service.
Nous considérons ce service comme un SinglePointOfFailure de notre système, car s'il tombe, un client ne pourra plus commander, ce qui est le besoin primaire de Dronzone.

Nous avons choisi d'établir une interface de communication avec contrat fort (Remote Procedure Call) pour la réception des commandes à effectuer étant donné qu'il s'agit d'envois critiques et que nous voulons nous assurer du format et de la validité des données que nous recevons. En effet, le temps d'attente de l'utilisateur avant qu'il ne reçoive la confirmation que sa commande est bien enregistrée et prête à être traitée doit être minimal. L'utilisation de communications RPC nous permet également d'avoir un temps de réponse allégé lors des échanges entre Roger et notre système.

2. Warehouse Service

Il est dédié au domaine de la gestion de colis. Il a pour rôle principal d'être notre buffer des commande : il permet aux techniciens gérant les commandes (représentés par Klaus) de consulter la liste des commandes à préparer pour l'expédition, et de notifier le système une fois la tâche effectuée.

Ce service est découplé de celui qui gère les commandes car il ne suit pas toujours la même temporalité. En effet, une commande effectuée par un Client à un temps donné ne sera pas nécessairement traitée immédiatement, en fonction des contraintes liées au domaine de la gestion des colis : stock manquant, etc.

Nous avons choisi d'établir des interfaces de type ressource (REST) pour la réception d'une commande à préparer et pour les interactions de Klaus avec le système étant donné qu'il s'agit de requêtes simples et contenant une version simplifiée de nos Order, ne nécessitant donc pas de contrat fort comme dans le service Order.

3. Drone Service

Il est dédié à la gestion de la flotte de drones de Dronazone. Il a pour rôle l'affectation d'un drone à l'expédition d'une commande, mais également de permettre à Elena d'avoir une vue d'ensemble sur la flotte.
Ce service est découplé des deux autres, d'une part en raison de son domaine différent : celui de la gestion logistique de livraisons, mais également par sa spécificité technique.
En effet, la communication avec les nombreux drones doit se faire par des flux de messages fréquents et nombreux. Nous voulons permettre à Elena d'avoir une vue d'ensemble du statut et du niveau de batterie des drones de la façon la plus approchante du temps réel[1].

Nous avons choisi d'établir des interfaces orientées vers l'envoi de messages et de documents, utilisant des queues de messages pour permettre aux drones de publier à intervalle régulier leur position et leur statut, sans avoir à établir une connection forte à notre système : la perte d'un message de position ou de batterie n'est pas critique puisque l'envoi est prévu toutes les 10 secondes.

4.Statistics Service

Ce service s'occupe de la surveillance des statistiques.
Nous avons choisi 5 statistiques à surveiller :

- Le nombre de commandes total, assez simple à implémenter côté code, apporte beaucoup de valeur métier combiné à d'autres stats (nombres de paquets emballés, nombre de drones partant)
- Le nombre de paquets empaquetés,
- Nombre de commandes livrées
- Nombre de drones qui reçoivent des demandes
- Temps de livraison : montre notre capacité à suivre la commande du départ à son arrivée (encore plus combinée avec des étapes intermédiaires)

Pour cela, nous avons ajouté des Listeners sur les topics Kafka concernant les orders, qui enregistrent chaque élément dans la base de données Influx. (voir schéma ci dessous)
Actuellement nous sommes capables de restituer chaque donnée enregistrée, mais nous n'appliquons pas encore de traitement afin de raffiner les données de ressortir les statistiques mentionnées précédemment. Nous avons fait le choix de repousser cet traitement des données au prochain sprint car il demandera peu de travail, et nous avons préféré privilégier un rendu stable (notamment les tests d'intégration).
Ce service est indépendant des autres, et embarque sa propre base de données.

[1] Une communication temps réel avec les drones n'est pas envisageable étant donné qu'ils se trouvent dans un environnement extérieur non contrôlé (perte de réseau temporaire, changement d'antenne réseau, etc). Cela entraînerait par ailleurs une consommation d'énergie qui n'est pas nécessaire selon nous.

5.WatchDog Service

Comme son nom l'indique, ce service a pour rôle de surveiller les événements du système et d'alerter en cas de comportement suspect.
Il répond à la demande de Haley d'être avertie si un client effectue un trop grand nombre de commandes dans un temps court : il y a une forte chance qu'il s'agisse d'une fraude ou d'un abus d'identité.

Ce service est indépendant des autres services et est un bon exemple de ce que nous pouvons obtenir avec une architecture en micro-services. En effet, aucun autre service ne s'adresse directement au WatchDog Service, et il n'y a pas non plus de partage de base de donnée nécessaire.

WatchDog Service va consommer les messages publiés par OrderService à chaque commande pour en extraire l'heure de commande, et l'identifiant unique du client. Il va ensuite mémoriser temporairement cette information dans une base de données.

Lors d'une prochaine commande du même client, WatchDog Service vérifie combien de commandes ont été effectuées dans une période de surveillance donnée (par exemple les 20 dernières minutes), et notifie Haley si le nombre est trop élevé.

Enfin, la base de donnée est très régulièrement nettoyée en retirant les commandes dont la date est au delà de la période de surveillance.

Cela nous permet d'avoir un service complètement indépendant, embarquant une base de donnée légère.

Prise de recul

L'ajout de Kafka dans notre projet nous a permis de nous rendre compte de son utilité et de sa puissance. Nous sommes également satisfait du peu de changement d'architecture qu'a entraîné sa venue, puisque que nous avons uniquement ajouté de nouveaux services.
L'ajout et l'utilisation des topics Kafka n'a pas été très homogène au moment de la transition, et nous avons dû passer par une phase de conception et de prise de recul sur nos choix d'implémentation.

À ce jour, nous avons une architecture établie, orientée événement et qui est adaptée aux besoin, mais son implémentation est toujours en cours : il faut implémenter les bons topics et les bons formats de messages dans notre projet.

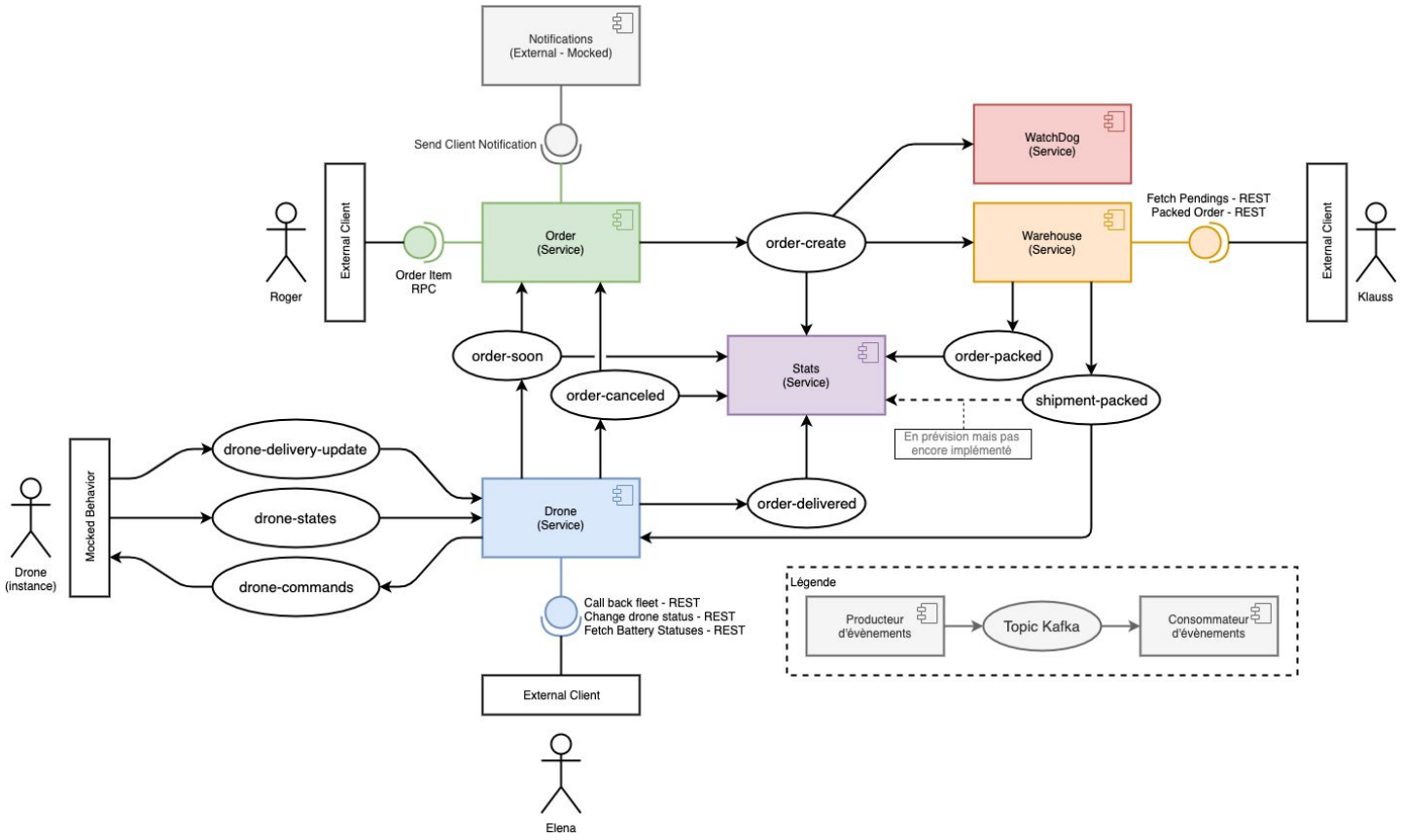
Au niveau de l'aspect scalabilité de notre projet, nous pensons que le gros de la charge se concentrera sur le service Order (périodes d'affluences telles que Noël) et dans le service Drone (envoi régulier des données de position et de batterie.) Toutefois, dans le cadre du service Drone il n'y aura pas de montée en charge comme il pourra y en avoir dans le

service Order. Nous sommes donc conscients que ce service sera un point critique de notre application et qu'il sera important de le renforcer par la suite.

Les défis que nous rencontrons désormais sont l'incertitude de fonctionnement de notre solution. En effet il nous manque la vision du fonctionnement dans l'ensemble, de bout-en-bout, et notre couverture de test est trop légère sur certains points. Pour remédier à cela nous sommes en train de mettre en place des scénarios de test d'intégration qui valideront le comportement bout-en-bout de notre solution et nous donnerons confiance dans notre système complet.

A l'heure actuelle, le scénario d'intégration n'est pas encore entièrement fonctionnel.

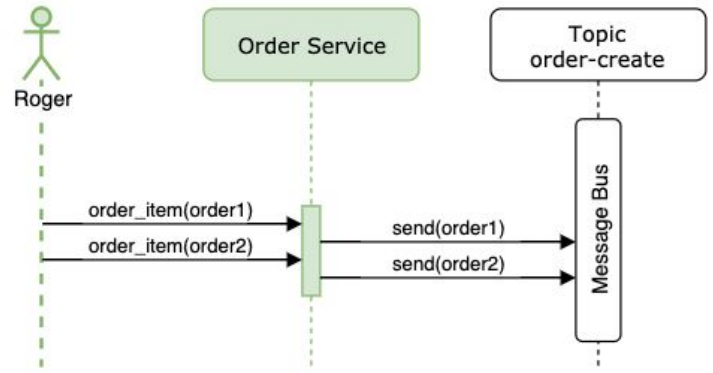
Diagramme de services

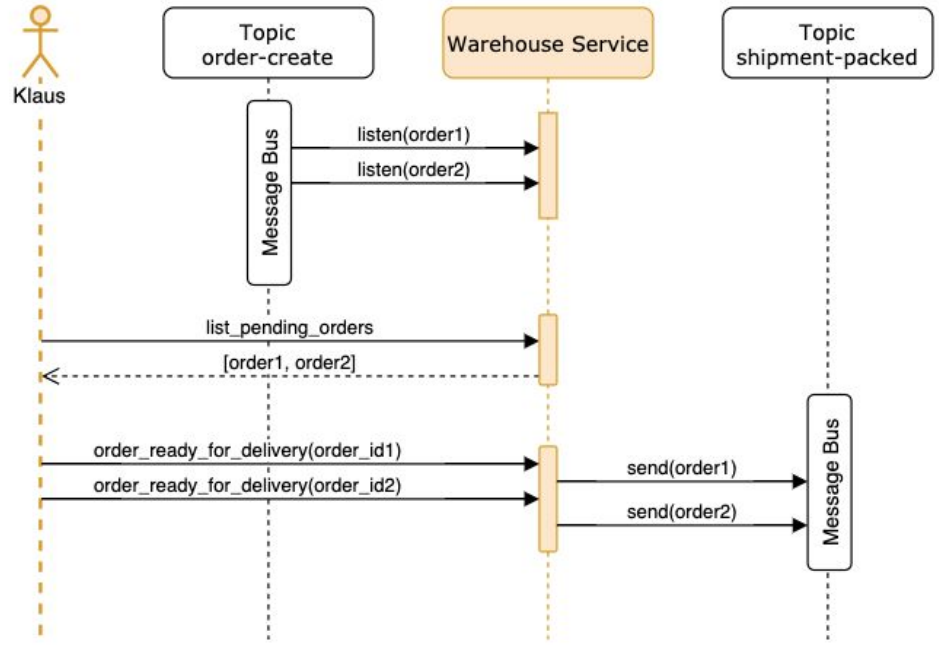


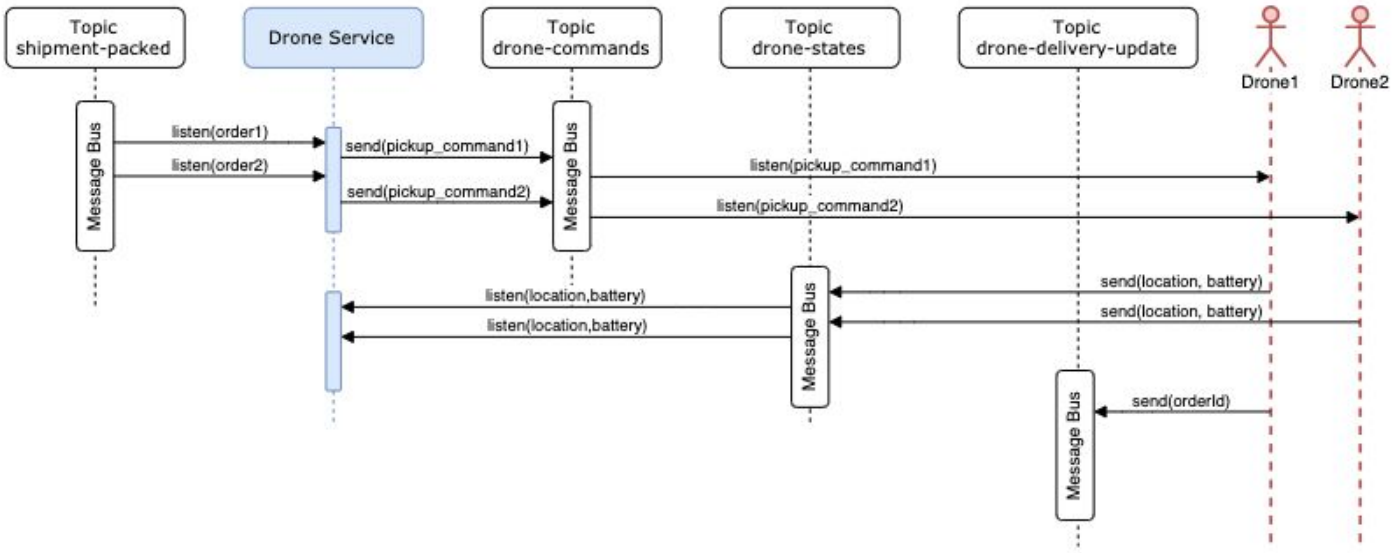
Diagrammes de séquence

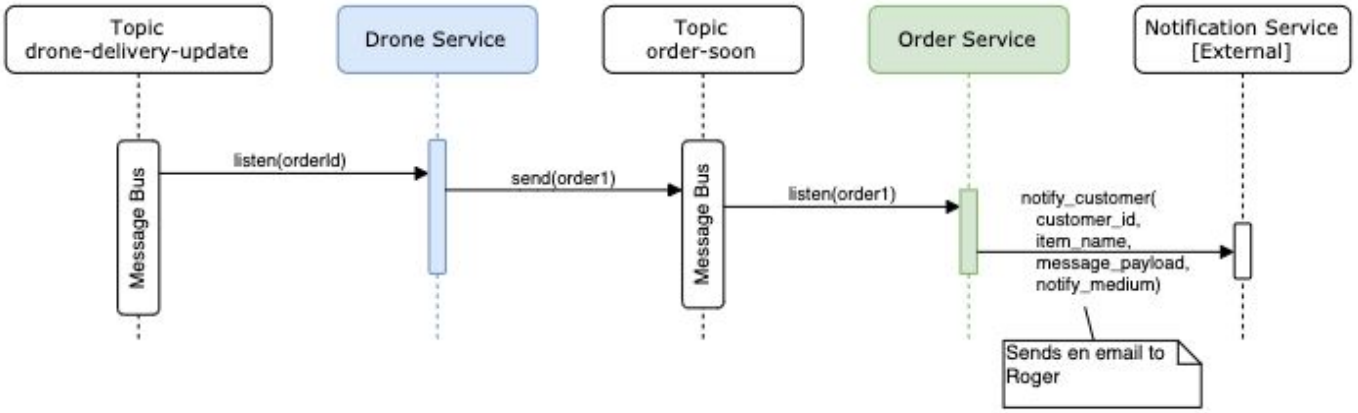
Scénario :

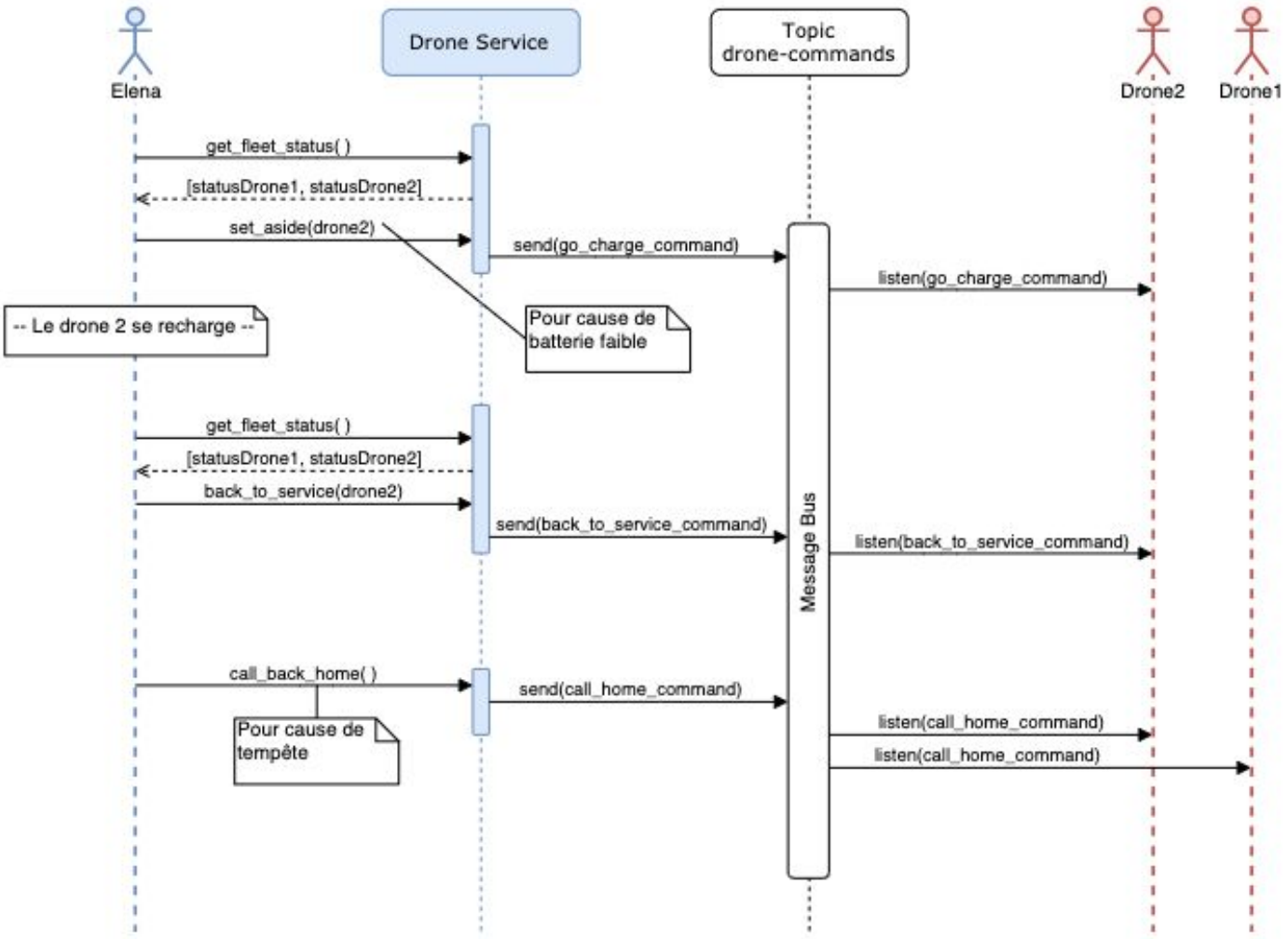
- 2 Commandes passées
- Klaus demande la liste des commandes => 2 commandes affichées
- Klaus notifie qu'une commande est prête x2 => Drone assigné à la livraison x2
- Drone 1 en approche => notification de livraison
- Drone 1 livre tout ok
- Roger change son media notif (Email → SMS)
- Drone 2 batterie faible, Elena le rappelle
- Drone 2 batterie pleine, Elena le remets dans la flotte active
- Elena rappelle tous les drones avant que le drone2 livre son colis => notif par sms






Topic shipment-packed
Drone Service
Topic drone-commands
Topic drone-states
Topic drone-delivery-update
Drone1
Drone2
Message Bus
listen(order1)
listen(order2)
send(pickup_command1)
send(pickup_command2)
listen(pickup_command1)
listen(pickup_command2)
send(location, battery)
send(location, battery)
listen(location,battery)
listen(location,battery)
send(orderId)


Topic drone-delivery-update
Drone Service
Topic order-soon
Order Service
Notification Service [External]
Message Bus
listen(orderId)
send(order1)
listen(order1)
notify_customer(
customer_id,
item_name,
message_payload,
notify_medium)
Sends en email to Roger


Elena
Drone Service
Topic
drone-commands
Drone2
Drone1
get_fleet_status( )
[statusDrone1, statusDrone2]
set_aside(drone2)
send(go_charge_command)
listen(go_charge_command)
-- Le drone 2 se recharge --
Pour cause de
batterie faible
get_fleet_status( )
[statusDrone1, statusDrone2]
back_to_service(drone2)
send(back_to_service_command)
listen(back_to_service_command)
Message Bus
call_back_home( )
send(call_home_command)
listen(call_home_command)
Pour cause de
tempête
listen(call_home_command)