Service-oriented Architecture

MarsY, to the space and beyond

Team A

(comme astronaute)

Antoine BUQUET

-

Benoit GAUDET

-

Ayoub IMAMI

-

Mourad KARRAKCHOU

Sommaire

I. Architecture

1. Explication de l'architecture

Services	*weather-service*
	mission-service
	rocket-department-service
	telemetry-service
	payload-service
	executive-service
	logs-service
	robot-department-service
	webcaster-service
	telemetry-reader-service
	scientific-department-service
Mocks	*rocket-hardware-mock-service*
	payload-hardware-mock-service
	stage-hardware-mock-service
	robot-hardware-mock-service
Bases de données	*database-telemetry*
	database-telemetry-slave
	database-logs
	database-payload
Plateforme	*Kafka*

Figure 1. - Composition de l'architecture

L'architecture de notre projet est composée d'un certain nombre de micro-services que nous pouvons classer selon deux catégories distinctes : les services proprement dits et les mocks. Plusieurs bases de données sont également présentes dans notre architecture. Enfin, nous avons mis en place Kafka à certains endroits clés de notre application.

- Les services sont les composants individuels de l'application. Ils sont responsables de tâches spécifiques, telles que la gestion de la mission, la collecte de données météorologiques ou la surveillance des métriques de la fusée.

- Les mocks sont des services fictifs utilisés pour simuler le comportement physique réel de la fusée ou de sa cargaison par exemple. Cela permet de tester d'autres services en envoyant des données régulièrement.

- Les bases de données sont utilisées pour stocker les données de l'application. Il en existe quatre dans notre projet : deux bases de données de télémétrie, une base de données pour la cargaison et une pour les logs.

- L'intégration de Kafka permet d'assurer une communication efficace et fiable entre les différents services de notre application. Kafka permet aux services de publier et de consommer des événements en temps réel. Cela permet de coordonner les opérations au sein de l'application.

- Les services interagissent entre eux grâce à l'utilisation d'APIs REST, particulièrement lors de la transmission de messages exceptionnels. En ce qui concerne des échanges d'informations réguliers, nous avons préféré choisir une communication événementielle via Kafka.

2. Diagramme d'architecture

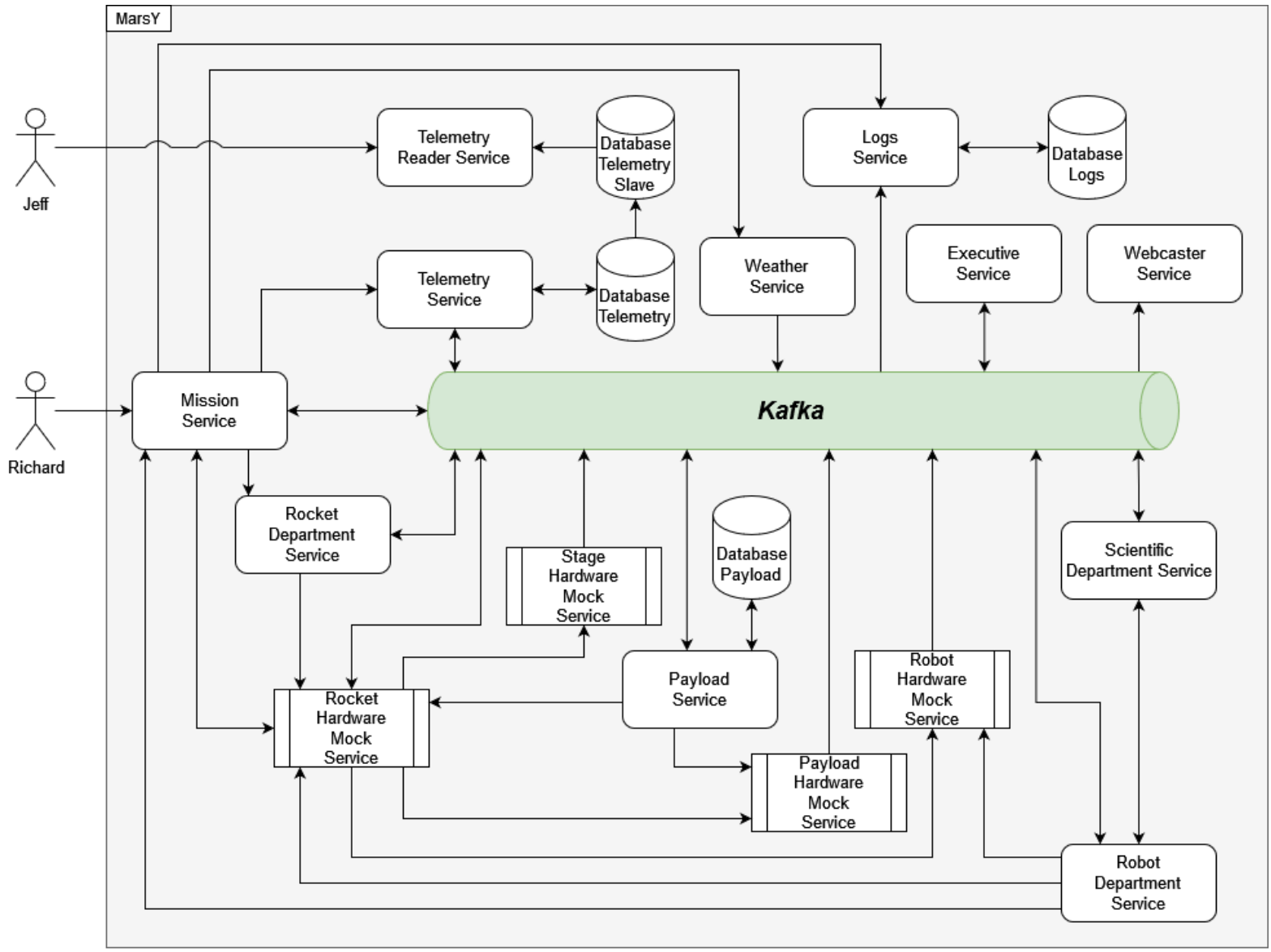

Figure 2. - Diagramme d'architecture

À des fins de comparaison, notre diagramme d'architecture lors du rendu du MVP est disponible en *Figure 3* ci-dessous.

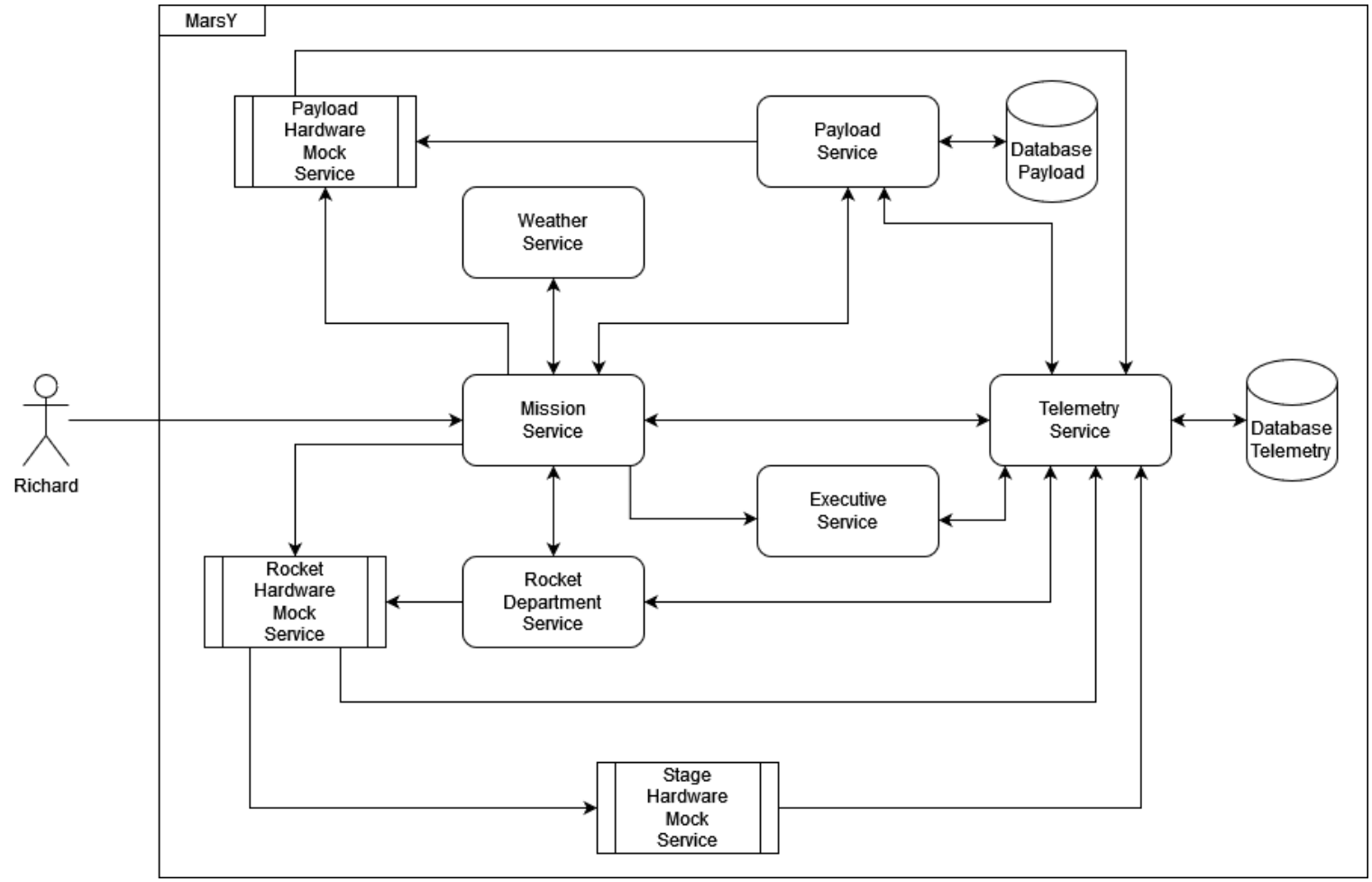

Figure 3. - Ancien diagramme d'architecture

L'ajout de nouvelles User stories dans notre application a nécessairement mené à l'évolution de notre architecture avec l'ajout de nouveaux micro-services et de nouvelles bases de données. Mais l'évolution entre ces deux diagrammes d'architecture réside aussi dans le changement d'approche quant à la communication entre les services. En effet, nous sommes passés à une approche événementielle avec la mise en place de Kafka. Cela permet notamment une communication asynchrone entre les micro-services, ce qui offre une flexibilité accrue et une meilleure extensibilité de notre système. Grâce à l'approche événementielle, les services peuvent désormais échanger des informations de manière décentralisée, sans être directement couplés les uns aux autres.

Cette approche événementielle présente des avantages significatifs, particulièrement pour gérer des charges de travail variables et garantir une plus grande résilience face aux pannes potentielles. Les topics Kafka servent de canaux de communication spécialisés, permettant à chaque micro-service de s'abonner et de réagir aux événements qui lui sont pertinents, tout en ignorant ceux qui ne le sont pas.

Enfin, Kafka agit alors comme un journal de transactions distribué, enregistrant de manière immuable chaque événement produit par les services. Ainsi, il nous permet de jouer un rôle de source de vérité si un service venait à mal fonctionner pendant le traitement d'événements ou que nous venions à perdre une partie d'une base de données.

Nous l'avons implémenté de manière stratégique aux endroits les plus sollicités par notre programme, à savoir le service de *télémétrie*, de *logs* et du *webcaster*.

Une autre évolution a été l'utilisation du CQRS pour la base de données du *telemetry-service*. En effet, la problématique était le nombre important d'écritures et potentiellement de lecture sur la base de données *telemetry-service*. Dans la vraie vie, les employés du service de télémétrie seraient amenés à souvent lire les données de la mission, nous avons donc créé un 2ème service : *telemetry-reader-service.* Ce nouveau service lit sur une base de données slave de la base de données de *telemetry-service*. Cela permet notamment de séparer la lecture et l'écriture et ainsi de permettre un scaling horizontal plus ciblé lors de l'évolution de notre application. De plus, la base de données master est donc seulement accédé en écriture, ce qui permet d'éviter des verrous potentiels lors des lectures. Enfin, la synchronisation entre les deux bases de données implique que la base de donnée slave aura des données pas forcément à jour. Cependant, si l'on considère que la revue de ces informations se fait de manière postérieure pour vérifier certaines données ou le bon déroulement de la mission, cela n'est pas un problème.

3. Description des services

Services

- *weather-service* : s'assure que les conditions météorologiques sont sûres sur l'ensemble des sites
    - Répond GO ou NO-GO au *mission-service* lorsqu'il demande l'état des conditions météorologiques avant le lancement de la fusée

- *logs-service* : Enregistre l'ensemble des logs de la mission
    - Reçois l'ensemble des logs de tous les services liés à la mission, puis les enregistre
    - Permets d'accéder à l'historique des logs de chacune des fusées pour différentes missions

- *mission-service* : supervise toute la mission
    - Informe le *log-service* et le *telemetry-service* que la mission va commencer
    - Demande le remplissage en essence des deux étages de la fusée
    - Demande l'ajout de la cargaison au sein de la fusée
    - Demande au *weather-service* et au *rocket-department-service* un GO ou NO-GO pour lancer la fusée
    - Envoie un GO ou NO-GO au *rocket-department-service* en fonction des réponses du *weather-service* et du *rocket-department-service*
    - Informe le *payload-service*, l'*executive-service* et le *robot-department-service* du début de la mission
    - Demande au *telemetry-service* d'être averti si la fusée présente une anomalie sévère ou une anomalie de pression
    - Donne l'ordre à la fusée de se détruire en cas d'anomalie sévère
    - Mets fin à la mission si la fusée s'est autodétruite automatiquement en cas d'anomalie critique

- *rocket-department-service* : gère la fusée et de ses systèmes
    - Répond GO ou NO-GO à *mission-service* lorsqu'il demande l'état de la fusée avant le lancement
    - Lance le décompte du décollage de la fusée après avoir reçu la réponse GO de *mission-service*
    - Demande au *telemetry-service* d'être informé de certains événements au cours de la mission
    - Découpe la fusée en plein vol lorsqu'il n'y a plus de carburant dans le premier étage
    - Ralenti la fusée lors du passage dans la zone Max Q pour éviter d'endommager le matériel et la cargaison
    - Ralenti la fusée quand celle-ci atteint la bonne altitude orbitale

- *payload-service* : responsable de la cargaison du client et de son largage sur la bonne la trajectoire ou orbite
    - Demande au *telemetry-service* d'être informé lorsque la fusée atteint la bonne hauteur pour larguer la cargaison
    - Demande au *rocket-hardware-mock-service* de larguer la cargaison une fois sur la bonne trajectoire ou orbite
    - Reçois et enregistre les données du *payload-hardware-mock-service* via *telemetry-service*
    - Vérifie que l'altitude de la cargaison est toujours correcte après son largage

- *telemetry-service* : gère les communications de données entre la fusée et le sol
    - Reçois des événements de demandes de notification d'événements des autres services
    - Reçois, surveille et enregistre les événements de données produits par *rocket-hardware-mock-service*, *stage-hardware-mock-service*, *payload-hardware-mock-service* et *robot-hardware-mock-service*
    - Informe, via des événements Kafka, le *mission-service*, l'*executive-service*, le *payload-service*, le *rocket-department-service, scientific-department-service* et le *robot-departement-service* de certains événements
    - Envoi les données des échantillons de sol prélevé par le robot via un évènement Kafka

- *executive-service* : dirige et pilote stratégiquement les activités et les objectifs de l'entreprise
    - Demande au *telemetry-service* d'être informé du retour du premier étage de la fusée sur Terre

- *robot-department-service :* responsable de la gestion du robot et de son largage sur la Lune
    - Demande au *telemetry-service* d'être informé lorsque la fusée atteint la bonne hauteur pour déployer le robot
    - Demande au *rocket-hardware-mock-service* de larguer le robot puis lance la procédure d'atterrissage du robot en demandant au *telemetry-service* d'être informé lorsque celui-ci atterrit sur la Lune
    - Demande au *scientific-department-service* les coordonnées où procéder à des analyses du sol
    - Lance l'auto-pilotage du robot auxdites coordonnées
    - À l'endroit voulu, demande au *robot-hardware-mock-service* de procéder à des analyses du sol

- *webcaster-service* : Est notifié des différentes étapes de la mission puis les retransmet en direct sur le web
    - Est informé quand des étapes clés de la mission sont atteintes
    - Retransmets en direct ces informations

- *scientific-department-service* : analyse les données des échantillons prélevée par le robot
    - Lorsque demandé, envoi les coordonnées où procéder à des analyses du sol
    - Reçois de la part de *telemetry-service* les données du sol prélevé par le robot

Mocks

- *rocket-hardware-mock-service* : représentation fictive du matériel de la fusée
    - Commence à envoyer des données au *telemetry-service* après le début de la mission
    - Envoie des données sur l'altitude, la vitesse, le carburant de chaque étage, l'état et l'accélération de la fusée toutes les demi-secondes
    - Largue le premier étage de la fusée
    - Largue la cargaison de la fusée
    - Largue le robot sur la Lune
    - En cas de détection d'une anomalie critique, la fusée informe *mission-service* qu'elle va s'autodétruire en raison de cette anomalie, puis procède à l'autodestruction

- *stage-hardware-mock-service* : représentation fictive du matériel des étages de la fusée
    - Commence à envoyer les données d'un étage spécifique au *telemetry-service* après que l'étage a été détaché
    - Envoie des données sur l'altitude, la vitesse, le carburant et l'accélération de l'étage toutes les secondes

- *payload-hardware-mock-servic*e : représentation fictive du matériel de la cargaison
    - Commence à envoyer des données au *telemetry-service* après avoir été lâché par la fusée
    - Envoie des données sur la position actuelle de la cargaison au *telemetry-service* toutes les secondes

- *robot-hardware-mock-servic*e : représentation fictive du matériel du robot
    - Commence à envoyer des données au *telemetry-service* après avoir été lâché par la fusée
    - Envoie des données sur la position du robot au *telemetry-service* toutes les secondes
    - Attend l'ordre de se déplacer vers un point précis pour utiliser son autopilote
    - Prélève des échantillons du sol afin de les analyser

Databases

- *database-telemetry* :
    - Enregistre les données de la fusée reçues par *rocket-hardware-mock-service*
    - Enregistre les données du premier étage de la fusée reçues par le *stage-hardware-mock-service*
    - Enregistre les données du robot reçues par le *robot-hardware-mock-service*
    - Enregistre les demandes de notification reçues par les autres services
    - Enregistre les noms de fusée reçus par *mission-service*

- *database-telemetry*-slave :
    - Réplication de *database-telemetry* accessible en lecture seule

- *database-payload* :
    - Enregistre les données de la cargaison reçues par le *payload-hardware-mock-service*

- *database-logs* :
    - Enregistre les noms de fusée reçus par *mission-service*
    - Enregistre les données de l'ensemble des logs des services liés à la mission, donc tous sauf *webcaster-service*

II. Scénarios

1. Couvertures des users stories

Rappel des rôles :

- Richard ➜ Mission Commander
- Elon ➜ Chief of the Rocket Department
- Tory ➜ Launch Weather Officer
- Jeff ➜ Telemetry Officer
- Gwynne ➜ Chief of the Payload Department
- Peter ➜ Chief Executive Officer of Mars Y
- Marie ➜ Webcaster of the launches

Rôles associés aux users stories 19, 20 et 21 que nous avons ajoutés :

- Wally ➜ Robot Commander
- Issac ➜ Chief Scientist

☑ 1) Tory : Vérifier la météo pour assurer le lancement sûr de la fusée.
☑ 2) Elon : Surveiller la fusée avant le lancement pour assurer son bon fonctionnement.
☑ 3) Richard : Effectuer un sondage Go/No Go avant le lancement pour validation.
☑ 4) Elon : Émettre l'ordre de lancement après validation pour livrer la charge.
☑ 5) Jeff : Recevoir, stocker et monitorer les données télémétriques pour assurer le bon déroulement de la mission.
☑ 6) Elon : Séparer la fusée en deux parties pour se séparer de l'étage sans fuel.
☑ 7) Gwynne : Placer la cargaison sur la bonne orbite pour satisfaire le client.
☑ 8) Richard : Ordonner la destruction en cas d'anomalie sévère pour éviter les risques.
☑ 9) Peter : Faire atterrir le premier étage pour réutilisation et réduction des coûts.
☑ 10) Jeff : Recevoir, stocker et consulter les données du booster depuis le lancement jusqu'à l'atterrissage.
☑ 11) Gwynne : Vérifier les données télémétriques pour certifier les paramètres orbitaux.
☑ 12) Elon : Passer le Max Q sans stress excessif pour assurer la sécurité globale.
☑ 13) Richard : Superviser les procédures de lancement détaillé de la mission.
☑ 14) Richard : Stoker les logs de l'ensemble de la mission.
☑ 15) Marie : Suivre en temps réel les événements du lancement pour la diffusion web.
☑ 16) Peter : Gérer plusieurs lancements avec diverses fusées.
☑ 17) Richard : Déclencher des alertes pour assurer le contrôle en cas de dysfonctionnement.
☑ 18) Richard : Abandonner la mission et destruction automatique de la fusée en cas d'anomalie critique pour éviter les risques.

Nous avons également ajouté trois user stories supplémentaires :

- ☑ 19) En tant que Wally (Commandant du Robot), je veux que le robot atterrisse en toute sécurité sur la lune afin qu'aucune partie du robot ne soit endommagée.
- ☑ 20) En tant que Wally (Commandant du Robot), une fois que le robot a atterri, je souhaite demander au département scientifique les coordonnées de la position à analyser, être en mesure de contrôler le robot pour qu'il se rende sur le site à analyser, et le faire prélever des échantillons de son environnement.
- ☑ 21) En tant qu'Isaac (Chef Scientifique), je souhaite que toutes les données des échantillons me soient envoyées afin que je puisse analyser la lune.

2. Réussite de la mission

Réussite de la mission avec passage à travers Max Q, livraison du satellite, retour du 1er étage sur Terre, atterrissage du robot sur la Lune et analyse des sols.

US couvertes : *1 / 2 / 3 / 4 / 5 / 6 / 7 / 9 / 10 / 11 / 12 / 13 / 14 / 15 / 19 / 20 / 21*

Tout au long du déroulement de la mission, les étapes importantes vont être retransmises sur le web par *webcaster-service*.

Tout commence par la réception d'une demande pour commencer la mission, puis *mission-service* signale le début de la mission. En même temps, il contacte *rocket-hardware-mock-service* pour activer les logs du hardware de la fusée. Le *rocket-hardware-mock-service* commence la journalisation et envoie des données de fusée à *telemetry-service*. Ensuite, *mission-service* indique le début de la préparation de la fusée, avant d'indiquer à *rocket-hardware-mock-service* de remplir le réservoir de ses étages. Pendant cette phase, il charge aussi la cargaison, un satellite dernière génération, dans la fusée. Puis *mission-service* initie un sondage auprès de *weather-service* pour obtenir le statut météorologique "GO". De manière similaire, il effectue un sondage auprès de *rocket-department-service* pour vérifier le statut de la fusée. Une fois que tous les services sont prêts et que les réponses du sondage sont positives, *mission-service* est autorisée à lancer la fusée. Après cette phase de préparation, mission-service demande à *telemetry-service* de le prévenir dans le cas où la fusée présenterait une anomalie sévère ou de pression. Finalement, il ordonne *rocket-department-service* d'effectuer la mise à feu de la fusée après 60 secondes. Au même moment, *playload-service*, *executive-service* et *robot-department-service* sont prévenus du début de la mission et demandent chacun à *telemetry-service* de les prévenir dans le cas où certaines conditions seraient remplies. À la fin du compte à rebours, *rocket-department-service* gère le lancement de la fusée en activant l'étage inférieur de la fusée avant d'initier son décollage et de confirmer le lancement de la fusée. Il demande ensuite à *telemetry-service* de le notifier lorsque le carburant du premier étage est épuisé, lorsque la fusée atteint la phase Max Q, lorsqu'elle la quitte et enfin quand l'altitude de largage de la cargaison est atteinte.

```
INFO  mission-service: Weather service is OK to launch rocket
INFO  mission-service: Rocket department service is OK to launch rocket
INFO  mission-service: Mission service is OK to launch rocket
INFO  mission-service: Rocket Appolo 11 preparation complete
```

Figure 4. - Logs correspondants au sondage permettant de déterminer la possibilité de lancer la fusée

```
INFO  mission-service: Order the rocket department to launch the rocket in 60 seconds
INFO  rocket-department-service: Request received to launch the rocket in 60 seconds
INFO  rocket-department-service: The mission will start in 60 seconds
```

Figure 5. - Logs correspondants à la demande de démarrage de la fusée après le compte à rebours

```
INFO  rocket-department-service: The rocket department wants the telemetry service to notify it when the rocket enters the Max Q
INFO  rocket-department-service: Ask to the telemetry service to be notified when the rocket reaches the altitude of 1300.0
INFO  rocket-department-service: The rocket department wants the telemetry service to notify it when the rocket leaves the Max Q
INFO  rocket-department-service: Ask to the telemetry service to be notified when the rocket reaches the altitude of 1800.0
INFO  rocket-department-service: The rocket department wants the telemetry service to notify it when the rocket reaches the orbit altitude
INFO  rocket-department-service: Ask to the telemetry service to be notified when the rocket reaches the altitude of 3000.0
```

Figure 6. - Logs correspondants à la demande de notification de rocket-department à telemetry dans le cas où certaines conditions seraient atteintes

```
INFO  mission-service: The rocket department has launch the rocket
```

Figure 7. - Log correspondant au décollage de la fusée

Pendant le vol de la fusée, un premier événement apparaît lorsque la fusée atteint la zone de Max Q (où la pression est maximale), *telemetry-service* prévient *rocket-department-service*, qui fera ralentir la fusée pour éviter de potentiels dégâts sur la fusée ou sa cargaison. Peu après, lorsque la fusée sort de la zone de Max Q, *telemetry-service* prévient *rocket-department-service* que la fusée quitte cette zone, afin qu'elle accélère de nouveau. Après cela, un nouvel événement apparaît quand le carburant du premier étage de la fusée est presque épuisé. *telemetry-service* prévient *rocket-department-service* du niveau faible de carburant sur l'étage inférieur de la fusée. La fusée va ensuite se séparer de cet étage, avant d'activer le deuxième étage pour continuer à avancer.

```
INFO  telemetry-service: Notify the rocket department that the rocket has reached a specific height
INFO  rocket-department-service: Notification received, rocket enters Max Q => slow down
INFO  rocket-department-service: Inform the rocket hardware service to activate the slow down
INFO  rocket-hardware-mock-service: Request received for deactivate lowest stage of the rocket
INFO  rocket-hardware-mock-service: Deactivate lowest stage of the rocket, speed will decrease
```

Figure 8. - Logs correspondants à l'entrée de la fusée en Max Q

```
INFO  telemetry-service: Notify the rocket department that the fuel has reached a specific level
INFO  rocket-department-service: Notification received, fuel condition reached for staging the rocket
INFO  rocket-department-service: Inform the rocket hardware service to stage the rocket
INFO  rocket-hardware-mock-service: Request received for stage the rocket
INFO  rocket-hardware-mock-service: Main engine cut-off
INFO  rocket-hardware-mock-service: Stage rocket physically
```

Figure 9. - Logs correspondants à la séparation du 1er étage de la fusée après que son réservoir est vide

Pour finir, lorsque la fusée est proche de la hauteur de largage, *telemetry-service* prévient *rocket-department-service* qui va procéder au ralentissement de la fusée afin de larguer la cargaison en toute sécurité. Peu après, lorsque la fusée atteint la hauteur de largage, *telemetry-service* prévient *payload-service* qui va lâcher la cargaison à la bonne altitude. Suite au largage, *payload-service* va émettre 10 échantillons de positions de la cargaison pour vérifier qu'il se trouve bien à la bonne orbite et donc qu'il a bien été livré.

```
INFO  telemetry-service: Notify the rocket department that the rocket has reached a specific height
INFO  rocket-department-service: Notification received, rocket reaches the orbit altitude
INFO  rocket-department-service: The rocket department wants the rocket hardware to eject the fairing
INFO  rocket-department-service: Inform the rocket hardware service to deploy the fairing
INFO  rocket-hardware-mock-service: Request received to deploy the fairing
INFO  rocket-hardware-mock-service: Eject fairing physically
INFO  rocket-department-service: The rocket department wants the second engine to cut-off
INFO  rocket-department-service: Inform the rocket hardware service to activate the slow down
INFO  rocket-hardware-mock-service: Request received for deactivate lowest stage of the rocket
INFO  rocket-hardware-mock-service: Deactivate lowest stage of the rocket, speed will decrease
```

Figure 10. - Logs correspondants au ralentissement de la fusée à l'approche de l'altitude de largage sur satellite

```
INFO  telemetry-service: Notify payload service that the correct height has been reached
INFO  payload-service: Notification received, requesting to drop the payload
INFO  payload-service: Inform the rocket hardware service to drop the payload
INFO  webcaster-service: Marie: 'The following news is whispered to me: "Payload drop requested"'
INFO  rocket-hardware-mock-service: Request received for drop the payload
INFO  rocket-hardware-mock-service: Drop payload physically
```

Figure 11. - Logs correspondants au largage du satellite après que la fusée est atteinte l'altitude de largage

```
INFO  payload-service: Number of payload data: 10
INFO  payload-service: Sample of 10 payload data available
INFO  payload-service: Verifying that the payload altitude was always between 3500 and 3900
                       Sample informations
INFO  payload-service: The orbital parameters desired by the customer are ensured!
```

Figure 12. - Logs correspondants à la validation de l'orbite du satellite

Après le largage de la cargaison, *payload-service* prévient *rocket-hardware-mock-service* que le largage a été effectué et qu'il peut accélérer de nouveau. La fusée continue son trajet et lorsque la fusée atteint la hauteur de largage du robot, *telemetry-service* prévient *robot-department-service* que la condition a été atteinte. *robot-department-service* demande le largage du robot au *rocket-hardware-mock-service*. Après le largage, *robot-department-service* prévient *mission-service* de la réussite du largage. *mission-service* pourra par la suite affirmer le succès de la mission et demander à *rocket-hardware-mock-service* d'arrêter la journalisation des informations de la fusée et de récupérer les logs de la mission auprès de logs-service afin de les analyser, dans le but d'améliorer les prochaines missions.

```
INFO  telemetry-service: Notify the robot department that the height has reached a specific level
INFO  robot-department-service: Notification received, requesting to drop the robot
INFO  robot-department-service: Inform the rocket hardware service to drop the robot
INFO  webcaster-service: Marie: 'The following news is whispered to me: "Robot drop requested"'
INFO  rocket-hardware-mock-service: Request received for drop the robot
INFO  rocket-hardware-mock-service: Drop robot physically
```

Figure 13. - Logs correspondants au largage du robot

```
INFO  robot-department-service: Notify mission-service that the robot has been dropped and the mission is a success
WARN  mission-service: The mission has succeed !!!
INFO  mission-service: Inform the rocket hardware service to stop logging
INFO  rocket-hardware-mock-service: Request received to stop logging
INFO  rocket-hardware-mock-service: Stop logging
INFO  webcaster-service: Marie: 'The following news is whispered to me: "The mission has succeed !!!"'
INFO  mission-service: Ask logs service for all logs
INFO  logs-service: Request received to get all logs for current rocket
```

Figure 14. - Logs correspondants à la réussite de la mission

Après son largage, le robot se dirige vers la Lune en indiquant sa position à *telemetry-service*. Lorsqu'il a atterri, *telemetry-service* prévient *robot-department-service* que le robot a correctement atterri sur la Lune. Ensuite, *robot-department-service* demande à *scientific-department-service* la destination à laquelle le robot doit se déplacer. Ces informations sont ensuite transmises à *robot-hardware-mock-service*, afin que le robot lance son autopilote en direction de la destination. Par la même occasion, *robot-department-service* demande à *telemetry-service* d'être prévenu lorsque le robot aura atteint la destination souhaitée. Quand cette dernière a été atteinte, *robot-department-service* est prévenu avant de demander à *robot-hardware-mock-service* de prendre les échantillons souhaités. Les informations de ces derniers sont envoyées à

telemetry-service qui va les transférer au *scientific-department-service* qui va les analysés afin d'en tirer des conclusions exceptionnelles.

```
INFO  robot-hardware-mock-service: Request received to start logging for robot
INFO  robot-hardware-mock-service: Robot has started its its descent to the Moon
```

Figure 15. - Logs correspondants au commencement de la descente du robot en direction de la lune

```
WARN  robot-hardware-mock-service: Robot has landed on the Moon
```

Figure 16. - Log correspondant à l'arrivée du robot sur la Lune

```
INFO  telemetry-service: Notify the robot department that the robot has landed successfully
INFO  robot-department-service: Notification received, robot has landed
INFO  robot-department-service: Ask the scientific department service the destination
INFO  webcaster-service: Marie: 'The following news is whispered to me: "Robot has landed"'
INFO  scientific-department-service: Warned that the robot has landed
INFO  scientific-department-service: Sending to the robot department service the position where the robot has to go to
INFO  robot-department-service: Destination received from Scientific Department
INFO  robot-department-service: Telling to the robot hardware service to use autopilot to move to the spot
INFO  robot-hardware-mock-service: Request received to start logging the movement of the robot
INFO  robot-department-service: Ask telemetry to being notify when the robot reach a x position of 70.0 and a y position of 60.0
```

Figure 17. - Logs correspondants à l'atterrissage et au choix de la destination du robot

```
INFO  telemetry-service: Notify the robot department that the robot has reached the position successfully
INFO  robot-department-service: Notification received, robot has reached the position
INFO  robot-department-service: Telling to the robot hardware service to take samples
INFO  robot-hardware-mock-service: Request received to take samples
INFO  webcaster-service: Marie: 'The following news is whispered to me: "Robot has reached the position"'
INFO  robot-hardware-mock-service: Robot has started to take samples
INFO  robot-hardware-mock-service: Robot has finished to take samples
INFO  telemetry-service: Receive robot hardware data sample: RobotData{position={x=70.0, y=60.0, altitude=0.0}, sample=Sample{minerals=[Pyroxene, Olivine, Ilmenite]}}
INFO  telemetry-service: Transferring Sample to the robot department service: Sample{minerals=[Pyroxene, Olivine, Ilmenite]}
INFO  scientific-department-service: Sample received for analyze
INFO  scientific-department-service: Sample analyzed: Sample{minerals=[Pyroxene, Olivine, Ilmenite]}
INFO  scientific-department-service: New Discovery: there might be presence of Water!
```

Figure 18. - Logs correspondants à l'arrivée du robot à la destination et de la récupération et de la transmission des échantillons au département scientifique

Lorsque le premier étage de la fusée est largué, il va envoyer ses informations à *telemetry-service*. Lorsque l'étage atteint une altitude de 900 mètres, il va réduire sa vitesse de chute afin de ne pas bruler et d'atterrir en douceur. Ensuite, quand il atteint une altitude inférieure à 250 mètres, *stage-hardware-mock-service* va automatiquement activer ses pieds qui lui permettront d'atterrir sans encombre. Finalement, puisque le stage a atterri, *telemetry-service* va prévenir *executive-service* que le stage a correctement atterrit sur Terre.

```
INFO  stage-hardware-mock-service: Request received to start logging for stage 1
INFO  stage-hardware-mock-service: Stage 1 has started its flight back at altitude 2360.0 decameters
```

Figure 19. - Logs correspondants au commencement du retour vers la Terre de l'étage 1 de la fusée

```
INFO  stage-hardware-mock-service: Stage 1 has started its entry burn at altitude 890.0 decameters. Starting engine to slow down
```

Figure 20. - Log correspondant au ralentissement de l'étage 1 avant son arrivée

```
INFO  stage-hardware-mock-service: Stage 1 has deployed its legs and started its landing burn at altitude 242.0 decameters. More power in the engine
```

Figure 21. - Log correspondant au déploiement des jambes d'atterrissage de l'étage 1

```
INFO  stage-hardware-mock-service: Stage 1 has landed. End of logging
```

Figure 22. - Log correspondant à l'atterrissage de l'étage 1 de la fusée

```
INFO  telemetry-service: Notify the executive that the stage has landed
INFO  executive-service: The stage has successfully landed
```

Figure 23. - Logs correspondants à la notification d'atterrissage de l'étage 1 à executive-service

3. Échec de la mission : anomalie sévère

Échec de la mission dû à une anomalie sévère et une demande de destruction de la fusée pour éviter des dégâts collatéraux

US couvertes : *1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 13 / 14 / 15 / 16*

Le début de la mission est le même que durant le scénario de la mission réussite. Les explications détaillées de ce scénario commencent après le lancement de la fusée, pendant que la fusée est en vol et qu'elle n'a pas encore terminé sa mission.

Durant le vol de la fusée, le maléfique Elune Mars est venu saboter la fusée. *rocket-hardware-mock-service* détecte alors que la fusée à une anomalie sévère. Lorsque *telemetry-service* reçoit les données de la fusée, il prévient *mission-service* qui va ainsi envoyer l'ordre à la fusée de se détruire afin d'éviter tout dégât collatéral que l'anomalie pourrait provoquer. Il indique ensuite que la mission a échoué.

```
WARN  rocket-hardware-mock-service: Elune Mars from SpaceY comes to sabotage the rocket :(
WARN  rocket-hardware-mock-service: Rocket successfully sabotaged
```

Figure 24. - Logs correspondants au sabotage sévère de la fusée par Elune Mars

```
INFO  telemetry-service: Notify mission service that the rocket has a specific status
INFO  mission-service: Order the rocket hardware to destroy the rocket
INFO  rocket-hardware-mock-service: Request received for destroy the rocket
WARN  rocket-hardware-mock-service: Rocket destroyed to prevent potential damage
INFO  mission-service: The rocket hardware has been destroyed
WARN  mission-service: The mission has failed due to unexpected events
INFO  mission-service: Inform the rocket hardware service to stop logging
INFO  rocket-hardware-mock-service: Request received to stop logging
INFO  rocket-hardware-mock-service: Stop logging
INFO  mission-service: Ask logs service for all logs
INFO  logs-service: Request received to get all logs for current rocket
INFO  mission-service: Number of logs of the missions : 109
```

Figure 25. - Logs correspondants à l'ordre de destruction de la fusée et de l'échec de la mission causé par une anomalie sévère de celle-ci

4. Échec de la mission : anomalie critique

Échec de la mission dû à une autodestruction de la fusée après la détection d'une anomalie critique

US couvertes : *1 / 2 / 3 / 4 / 5 / 6 / 7 / 13 / 14 / 15 / 16 / 18*

Le début de la mission est le même que durant le scénario de la mission réussite. Les explications détaillées de ce scénario commencent après le lancement de la fusée, pendant que la fusée est en vol et qu'elle n'a pas encore terminé sa mission.

Durant le vol de la fusée, le maléfique Palpatooine est venu rediriger la fusée en direction de la Terre afin de la détruire et d'assouvir sa soif de méchanceté. *rocket-hardware-mock-service* détecte alors que la fusée à une anomalie critique. Dès lors qu'il détecte cette anomalie, il prévient *mission-service* de l'état des événements avant de s'autodétruire dans le but d'éviter tout dégât collatéral que l'anomalie critique pourrait provoquer. *mission-service* indique ensuite que la mission a échoué pour cause d'événements imprévisible.

```
WARN  rocket-hardware-mock-service: Palpatooine comes to redirect the rocket and destroy the earth
WARN  rocket-hardware-mock-service: Rocket is redirected towards the earth
```

Figure 26. - Logs correspondants au sabotage critique de la fusée par le terrible Palpatooine

```
WARN  rocket-hardware-mock-service: Rocket critical anomaly detected
WARN  rocket-hardware-mock-service: Inform the mission service that the rocket has a critical anomaly and will self-destroy
WARN  mission-service: The mission has failed due to unexpected events
INFO  mission-service: Inform the rocket hardware service to stop logging
INFO  rocket-hardware-mock-service: Request received to stop logging
INFO  rocket-hardware-mock-service: Stop logging
INFO  mission-service: Ask logs service for all logs
INFO  logs-service: Request received to get all logs for current rocket
INFO  mission-service: Number of logs of the missions : 115
```

Figure 27. - Logs correspondants à l'autodestruction de la fusée et de l'échec de la mission causé par une anomalie critique de celle-ci

III. Compréhension et interprétation du sujet

1. Compréhension du sujet et choix de conception

Le projet MarsY consiste à modéliser la gestion d'une fusée en utilisant plusieurs micro-services. En plus de la création de nos services, nous avons dû simuler plusieurs composants matériels et prendre des décisions sur leur fonctionnement. L'objectif était de maintenir la cohérence avec le fonctionnement réel de ces composants tout en répondant aux besoins de nos différents services.

Pour le service rocket-hardware-mock, nous avons décidé qu'il renverrait diverses informations sur la fusée : le niveau de carburant de chaque étage, sa hauteur, sa vitesse et son accélération. Ces informations sont transmises toutes les demi-secondes au service de télémétrie pour simuler un signal récurrent envoyé par la fusée vers la Terre.

Un autre composant que nous avons simulé est le service stage-hardware-mock, représentant le premier étage de la fusée qui retourne sur Terre. Nous avons choisi de lui attribuer son propre service, car nous avons considéré que les deux composants de la fusée étaient totalement séparés pendant cette phase. Pour simuler cette séparation lors du largage du premier étage, le service rocket-hardware-mock envoie une requête à stage-hardware-mock pour l'informer que le largage a eu lieu et qu'il est désormais en autonomie. Ce service suit le même schéma d'envoi d'informations périodiques vers le service de télémétrie et il en va de même pour le troisième composant matériel, le payload-hardware-mock. Ce dernier envoie des positions plus détaillées, car il est nécessaire de pouvoir suivre la trajectoire du payload par la suite. Enfin, le quatrième composant matériel robot-hardware-mock envoie les informations sur la position du robot.

Ces services envoient leurs données au service de télémétrie de manière régulière. Celui-ci traite alors un nombre très important de données qu'il doit sauvegarder et il doit aussi notifier certains services quand des conditions sur ces données sont atteintes. Nous avons décidé de faire en sorte de créer une base de donnée slave à la base de données du service telemetry pour séparer l'écriture et lecture et ainsi éviter les verrous sur la base de données lorsque quelqu'un du service de télémétrie voudrait avoir accès à l'historique des données reçues. Nous avons alors créé un service telemetry-reader qui a accès uniquement à la base de données slave en lecture. La mise en place de ce CQRS permet donc d'éviter une charge excessive sur le telemetry-service et sa base de données.

Nous avons également dû faire des choix concernant la découpe des services que nous devions implémenter. Nous avons créé nos différents services en suivant les différents acteurs qui constituent l'expédition : Telemetry, Payload, Weather, Rocket, Executive, Mission, etc. Les décisions que nous avons prises étaient basées sur les responsabilités de chaque service et la cohérence des actions qui y sont effectuées. Par exemple, nous avons envisagé de créer deux services distincts, un pour le lancement de la fusée (rocket-launch) et un pour le largage d'un étage (rocket-stage). Cependant, nous avons opté pour un seul service rocket, car nous estimons que ces deux phases auront actuellement trop peu de responsabilités pour justifier la création de deux services distincts. Il était donc cohérent qu'un seul service gère les différentes phases de la fusée.

Enfin, il était nécessaire de faire des choix concernant le déroulement de la mission elle-même. Nous avons décidé que la fusée larguerait le payload à une hauteur de 3600 et qu'elle cesserait d'accélérer entre 1300 et 1800 mètres afin de passer sans problème la zone de Max Q. Pour le payload, nous avons convenu que nous pourrions considérer qu'il était sur la bonne trajectoire si 10 données consécutives émises par le payload-hardware-mock étaient comprises entre 3500 et 3900. Tous ces choix ont principalement été faits dans le souci de garantir la clarté du déroulement de la simulation.

Concernant les choix technologiques, les micro-services sont développés en Java avec l'aide de Maven et du framework Spring Boot. Cela nous a permis de grandement faciliter le processus de mise en place de l'application au début du projet. L'architecture interne des micro-services est représentée par les couches de contrôleurs, composants, services, connecteurs et enfin répertoires.
La communication entre les différents micro-services s'effectue en partie à travers les API des contrôleurs et l'aide du protocole HTTP REST. Nous avons choisi cette manière de faire également pour sa simplicité et son efficacité au début du projet et cela correspond parfaitement à nos besoins en termes d'échange de données entre micro-services. Pour les communications beaucoup plus récurrentes telles que les envois de données depuis les services "hardware" vers la télémétrie ou bien les logs vers le logs-service, cela s'effectue au travers de notre bus Kafka. En effet, comme évoqué dans la justification du diagramme d'architecture, Kafka permet une plus grande flexibilité et une meilleure extensibilité de notre système par le biais de la mise en place d'une communication asynchrone et du système de topic qui servent de canaux de communication spécialisés.

La persistance a été réalisée grâce au module Spring Data JPA relié à des bases de données PostgreSQL. Cette approche nous permet d'éviter d'écrire tout le code nécessaire pour se connecter aux bases de données, faire les transactions ou les requêtes SQL et autres, car JPA le fait déjà et nous offre une interface simple à utiliser.

2. Forces des micro-services

Les micro-services permettent d'avoir une architecture qui est plus à même de répondre à une demande nécessitant la gestion d'un grand nombre de requêtes. Les users stories implémentées auraient pu être gérées avec une application monolithe, cependant, si les services étaient amenés à être surchargés, la mise en place des micro-services nous permet de dupliquer les services nécessitant plus de ressources. Pour prendre un exemple concret, le service de télémétrie permet de stocker les données de plusieurs fusées et de monitorer ces données avec la mise en place d'un système de notification notamment. Celui-ci est beaucoup plus susceptible de devenir surchargé lorsque l'on fait décoller et qu'il faut monitorer plusieurs fusées en même temps ou bien des données plus complexes et plus nombreuses. Il sera alors possible de le dupliquer sans que le weather-service, par exemple, qui lui n'est pas surchargé, ne le soit aussi.

Le 2ème atout des micro-services est que le métier est réparti entre différents services et que la défaillance d'un des services n'entraîne pas forcément la défaillance des autres. En effet, si par exemple le payload service est défaillant et ne peut plus larguer la cargaison, il faut que la fusée continue de pouvoir être monitorée. Certains services sont centraux et sans eux, il n'est pas possible que la mission continue en toute sécurité. Néanmoins, le fait de limiter ces points d'engorgement permet d'avoir des micro-services pouvant continuer en attendant que les autres soient réparés.

3. Faiblesses et limites

Bien que l'utilisation des micro-services nous permette de séparer notre code en petits services indépendants, elle nous force aussi à correctement définir des interfaces et des routes de communication. Nous avons choisi d'utiliser le protocole HTTP pour une partie de la communication entre nos différents services, car il est simple à mettre en place et nous permet de transférer des données.

Enfin, comme visible sur la *Figure 3*, le service de télémétrie était trop central à notre application et pouvait constituer un Single Point Of Failure. Si celui-ci venait à être dysfonctionnel, plus aucune communication sur les données de la fusée vers la base spatiale sur Terre serait possible. La seule option envisageable serait alors de détruire la fusée pour éviter son comportement imprévisible lié à la perte de connaissance de ses métriques. La mise en place de Kafka permet de limiter le nombre de connexions du service de télémétrie qui n'a plus qu'à se brancher sur le bus, cependant, c'est désormais ce bus Kafka qui est devenu central à notre application et par conséquent un Single Point Of Failure potentiel.

La mise en place du CQRS permet de séparer la lecture de l'écriture, mais elle entraîne aussi le fait que les données lues par le Telemetry reader ne seront pas consistantes. Nous avons fait l'hypothèse que cela ne posait pas un problème, car les données de télémétrie sont stockées pour une analyse ultérieure. Cependant, dans le cas où elles seraient utilisées pour un monitoring nécessitant des données toujours très consistantes, le CQRS devrait être modifié.