Service Oriented Architecture

Rapport Final

Bassy Ayman

Bonifay Tobias

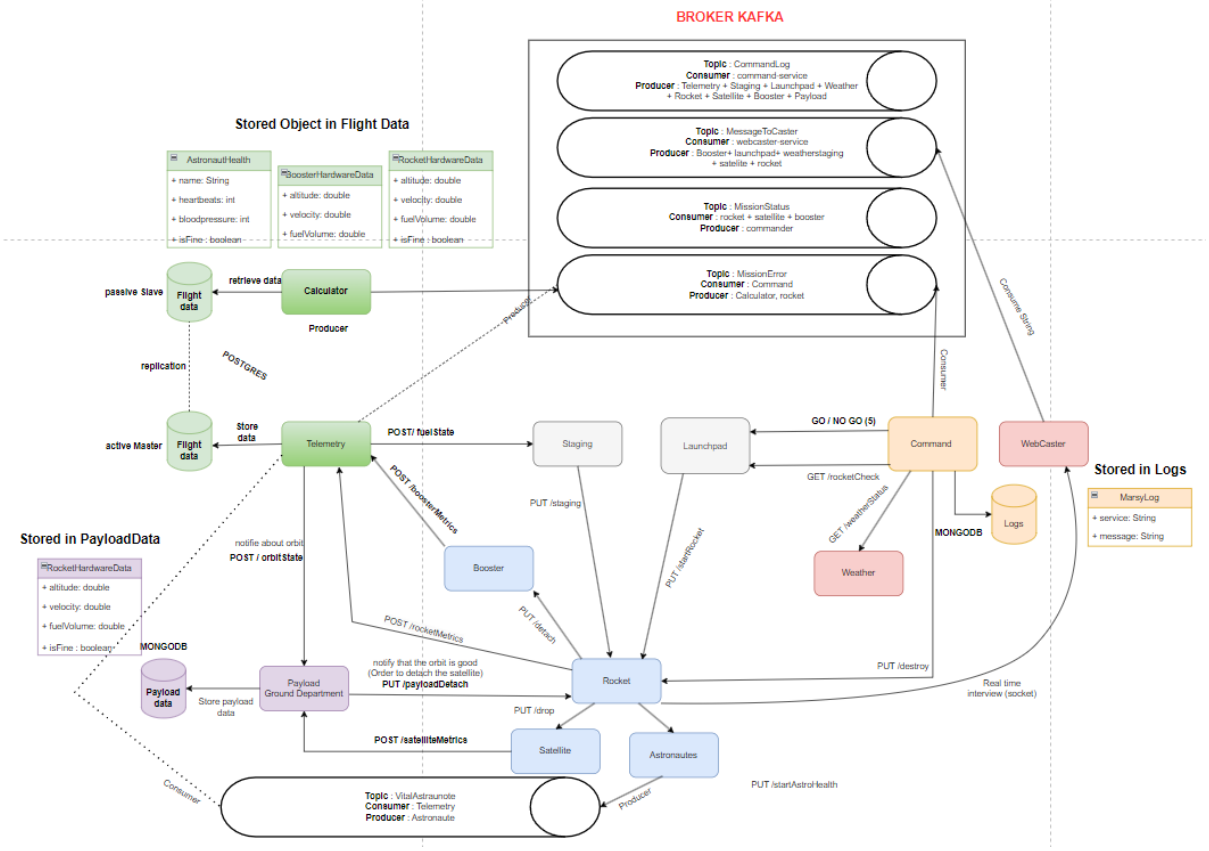
Melnyk Ihor

Schalkwijk Mathieu

Sommaire

Architecture

Plan de l’architecture actuelle



Dans cette architecture, en appliquant au mieux la loi de Conway, nous avons décidé de séparer les différents éléments intervenant dans le processus de lancement d'une fusée en services appartenant à des domaines différents.

En bleu, nous avons nos services stateful/Mock qui nous permettent de simuler les différentes parties de la fusée durant son lancement.

En orange, le service responsable de l’orchestration du lancement de la fusée.

En vert, les services responsables du stockage et de la distribution des données de métriques, ainsi que du calcul de ces dernières.

En gris, les services permettant de passer à la prochaine étape dans le processus de lancement de la fusée.

En mauve, le service de gestion des métriques du payload et de son détachement.

Et enfin, en rouge, les services “extérieurs” dans le cadre du lancement de la fusée, services ne voyant pas les données de la fusée passer.

Dans le cadre de la réalisation de cette architecture, nous nous sommes imaginés des scénarios à la “chaos monkey”, ce qui nous a permis de mieux découper nos services. L’objectif dans la conception de ces services était de pouvoir découper en plusieurs étapes le processus de lancement de la fusée, de telle manière que même si les services de départ se plantent, on puisse tout de même continuer la mission.

Au fur et à mesure que la mission avance, nos services vont peu à peu prendre leur responsabilité de manière indépendante et stateless afin d’assurer la continuité de celle-ci. En effet, même le service “rocket” va lui aussi déléguer la responsabilité d’envoi de métriques (metrics) vers de nouveaux services, ainsi la fusée n’a pas à assurer l’envoi de tout, car le moment où elle arrêterait de fonctionner, toute la mission s'arrêterait également. Cette isolation permet à nos services de rester stables, même dans le cas d’un ajout de nouvelles fonctionnalités à la mission.

Nouvelles Histoires utilisateurs

Pour ajouter plus de contexte à la suite du rapport, nous allons énoncer les user stories supplémentaires que nous avons décidé d'implémenter. À noter que l'objectif de la mission est finalement l'envoi d'un “payload” et d'un astronaute :

Astronaute vers l’au-delà

Un nouveau personnage entre en scène : Thomas Pesquet, un astronaute avec une grande expérience dans le domaine, souhaite rejoindre la mission MARSY, ayant pour ambition d'atteindre Mars prochainement.

US : En tant que Jeff (responsable de la télémétrie), je veux connaître l'état de santé de Thomas peu importe les événements qui surviennent. Sa sécurité, jusqu'à son envoi dans l'espace, est ma priorité. Il est essentiel que Thomas puisse quitter la fusée en cas de problèmes graves.

Interview spatiale

L'objectif final de la mission est de permettre à la fusée, et donc à Thomas, de rejoindre l'espace, et éventuellement un satellite tampon avant de poursuivre vers Mars. Le monde est curieux de connaître l'état d'esprit et l'expérience vécue par Thomas.

US : En tant que Marie (webcaster), je veux pouvoir réaliser une interview exclusive avec Thomas à la fin de la mission.

Nos Services

Command-Service :

Location : Au niveau du centre de commandement.
Rôle : Vérifier si la fusée et la météo sont en condition adéquate pour lancer la fusée. Reçoit des messages des autres services, y compris des logs pour suivre l'état de la mission et des messages d'erreur pour savoir si la mission doit être arrêtée (par exemple, en cas d'explosion de la fusée). Notifie également les services mock du numéro de la mission actuelle.

Weather-service :

Location : À la station météorologique.
Rôle fonctionnel : Vérifier la météo pour approuver ou non le départ de la fusée en mission.

Launchpad-service :

Location : Sur le site de lancement.
Rôle : Vérifier que tout est prêt et lancer la fusée.

WebCaster-service :

Location : À proximité du site de lancement.
Rôle : Commenter la mission et réaliser une interview avec l'astronaute.

Rocket-service :

Location : Sur la fusée
Rôle : Fournir des informations en temps réel sur la fusée au service de télémétrie.

Telemetry-service :

Location : Dans les locaux de la télémétrie.
Rôle : Collecter et stocker les données de télémétrie en temps réel de la fusée et du propulseur, ainsi que les informations de santé de l'astronaute.

Staging-service :

Location : Au centre de commandement.
Rôle : Recevoir les métriques de la fusée et décider du moment optimal pour séparer la fusée du propulseur.

Booster-service :

Location : Sur le propulseur.
Rôle : Envoyer des informations de métriques sur le propulseur après sa séparation avec la fusée, lors de sa descente vers le site de lancement.

Astronaut-service :

Location : Sur la combinaison de l'astronaute.
Rôle : Transmettre des informations de santé liées à l'astronaute. Permettre une réaction rapide, telle que l'éjection des astronautes, en cas de signal critique indiquant une potentielle explosion de la fusée.

Satellite-service :

Location : Sur le satellite largué par la fusée.
Rôle : Envoyer des informations spécifiques de métriques du satellite vers le service de payload.

Payload-service :

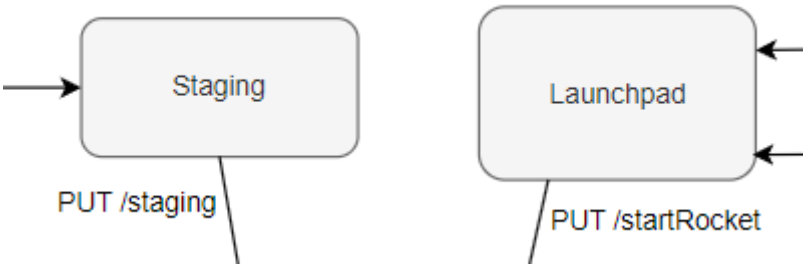
Location : Au centre de traitement du payload.
Rôle : Récupérer les métriques de la fusée du service de télémétrie pour décider du moment de détachement du satellite. Ensuite, récupérer les métriques envoyées par le satellite.

Calculator-service :

Location : Dans les locaux de la télémétrie.
Rôle : Récupérer automatiquement toutes les 2 secondes les dernières données enregistrées dans la base de données de télémétrie et évaluer les risques encourus à cet instant de la mission.

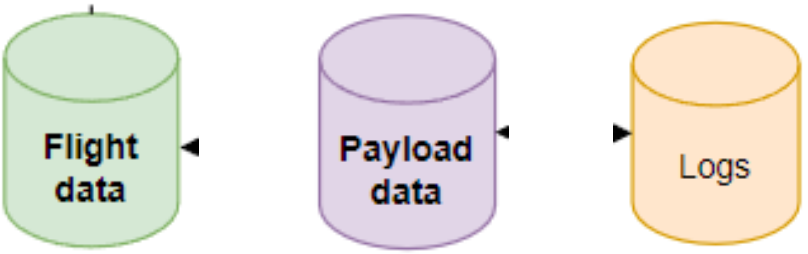
Plan détaillé de l’architecture

Séparation des services launchpad et staging

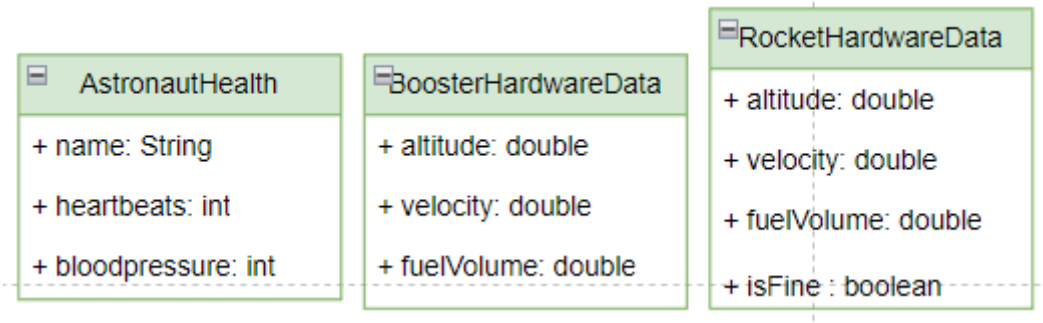


Nous avons observé que la validation de la possibilité de poursuite de la mission et le déclenchement du détachement du propulseur sont soumis à des temporalités différentes. En effet, la décision d'envoyer la fusée est prise en premier, et ce n'est qu'après que l'activation du détachement des propulseurs est réalisée. Dans notre modèle, nous surveillons les données d'altitude de la fusée et, en fonction de ces données, nous procédons au détachement du “booster”. Il nous a donc semblé pertinent de séparer le service initial en deux : un pour la décision du détachement du “booster” (propulseurs) et l'autre pour recueillir toutes les autorisations nécessaires à l'envoi de la fusée en mission.

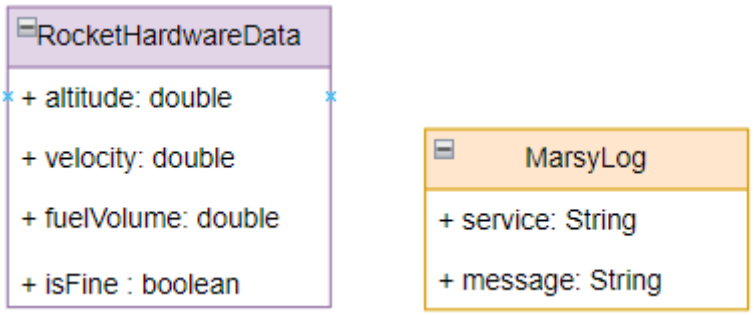
Séparation des données liées à la mission



En effet, pendant la mission, de nombreuses données sont stockées :



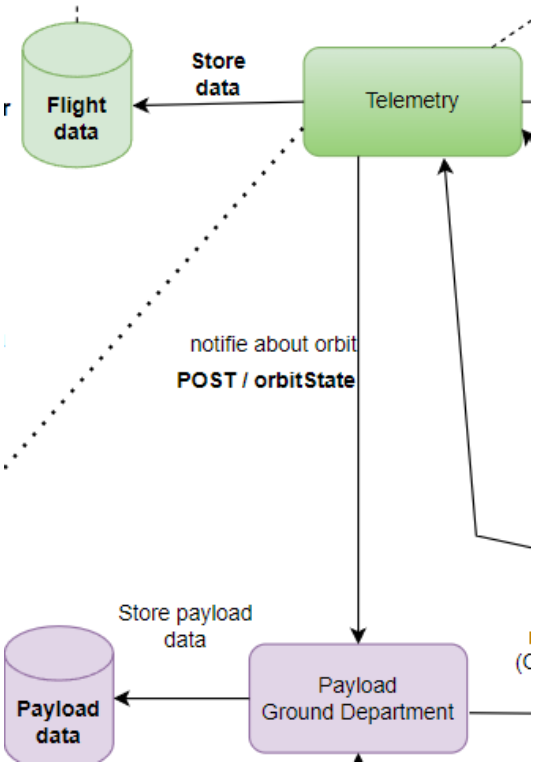
- Nous disposons tout d'abord des données liées à la fusée, qui sont stockées dans la base de données nommée “Flight Data”. Cette base de données comprend les mesures de la fusée, du propulseur, ainsi que les données relatives à l'état de santé des astronautes durant la mission. Ces différentes données seront ensuite transmises aux autres services qui en ont besoin.



- Dans la base “Payload Data”, nous trouvons les données de mesure liées au payload. Ces données permettent au service de gestion du payload (Payload Service) de déterminer la meilleure configuration pour le détachement du payload. Ce service se base sur des calculs spécifiques et plus précis, d'où la nécessité d'un service isolé traitant ses propres données.
- Enfin, dans “Logs”, nous avons les journaux (logs) des différents services, stockés dans la base de données du service de commandement (Command Service). Ce dernier peut ainsi récupérer à tout moment ces données en cas de panne ou d'autres incidents.

Au final, en fédérant nos bases de données, nous évitons la perte totale de nos données en les centralisant en un seul endroit. De plus, cela réduit les interactions des divers services avec une unique base de données, optimisant ainsi la gestion et la sécurité des données.

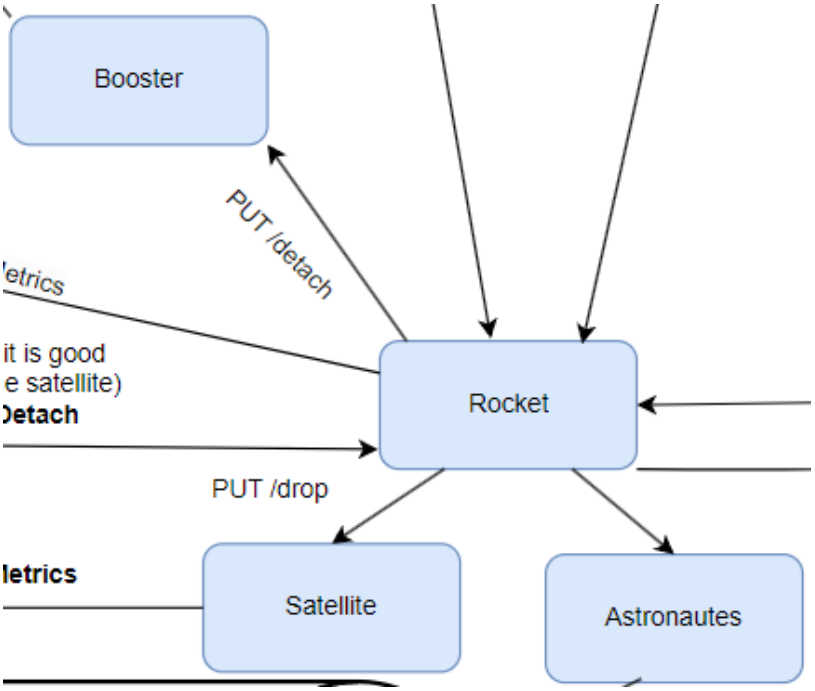
Séparation entre les services de télémétrie et de payload.



L'objectif de la séparation entre ces deux services est de garantir que, même si le service de télémétrie venait à tomber, nous puissions continuer à récupérer les données de mesure envoyées par le payload si celui-ci a été envoyé.

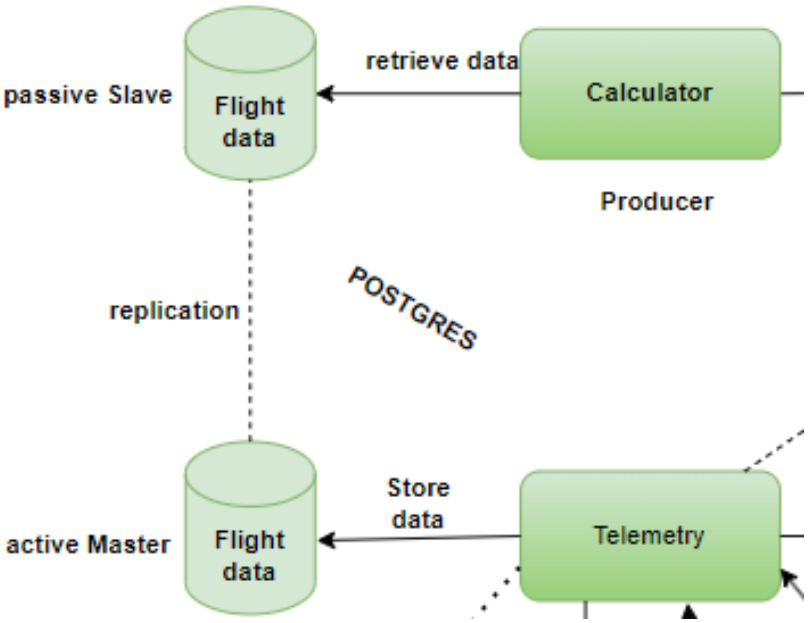
De plus, il est important de noter que les données envoyées par la fusée (Rocket) sont différentes de celles envoyées par le satellite. Cette distinction est cruciale car la fusée et le booster sont conçus pour retomber sur Terre, tandis que le satellite est destiné à rester en orbite.

Séparation des différents modules de la fusée



Nous avons divisé en plusieurs services Mock les différents composants de la fusée, dont l'objectif principal est d'envoyer des données de métriques. Cette approche permet de répartir la responsabilité de l'envoi de métriques sur différents services, dans le cas où l'un d'eux serait défaillant. Sur le plan temporel, les différents événements de détachement, servant de déclencheurs pour ces nouveaux services, nous permettent également de déléguer efficacement de nouvelles responsabilités.

Mise en place d’un modèle CQRS

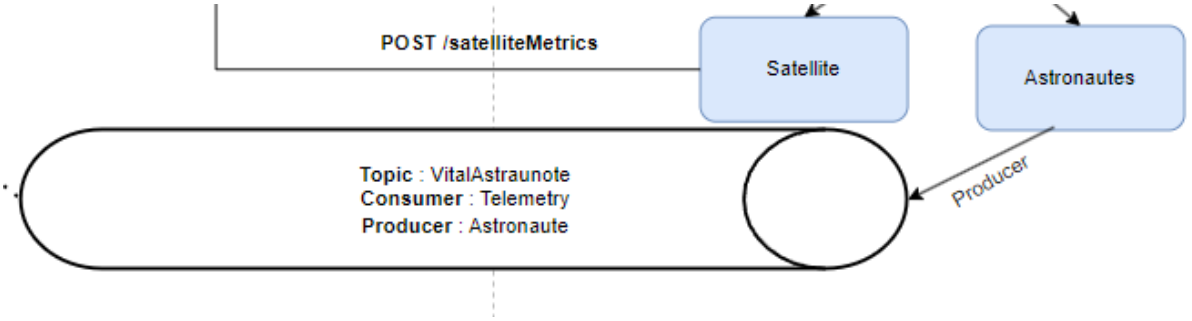


Dans le contexte des données de télémétrie, celles-ci arrivent en très grand nombre et sont fréquemment écrites dans la base de données associée. Initialement, tout était géré par le service de Télémétrie : le stockage des données, leur redirection et leur analyse. Cependant, ce service était surchargé par un grand nombre d'interactions en lecture et écriture.

Nous avons constaté, presque par accident, qu'après une vingtaine de minutes de mission, le service de Télémétrie a cessé de fonctionner. En effet, la lecture et l'écriture constantes dans la base de données, et la nécessité de persister et d'analyser une donnée chaque seconde, ont rendu le service incapable de supporter la charge.

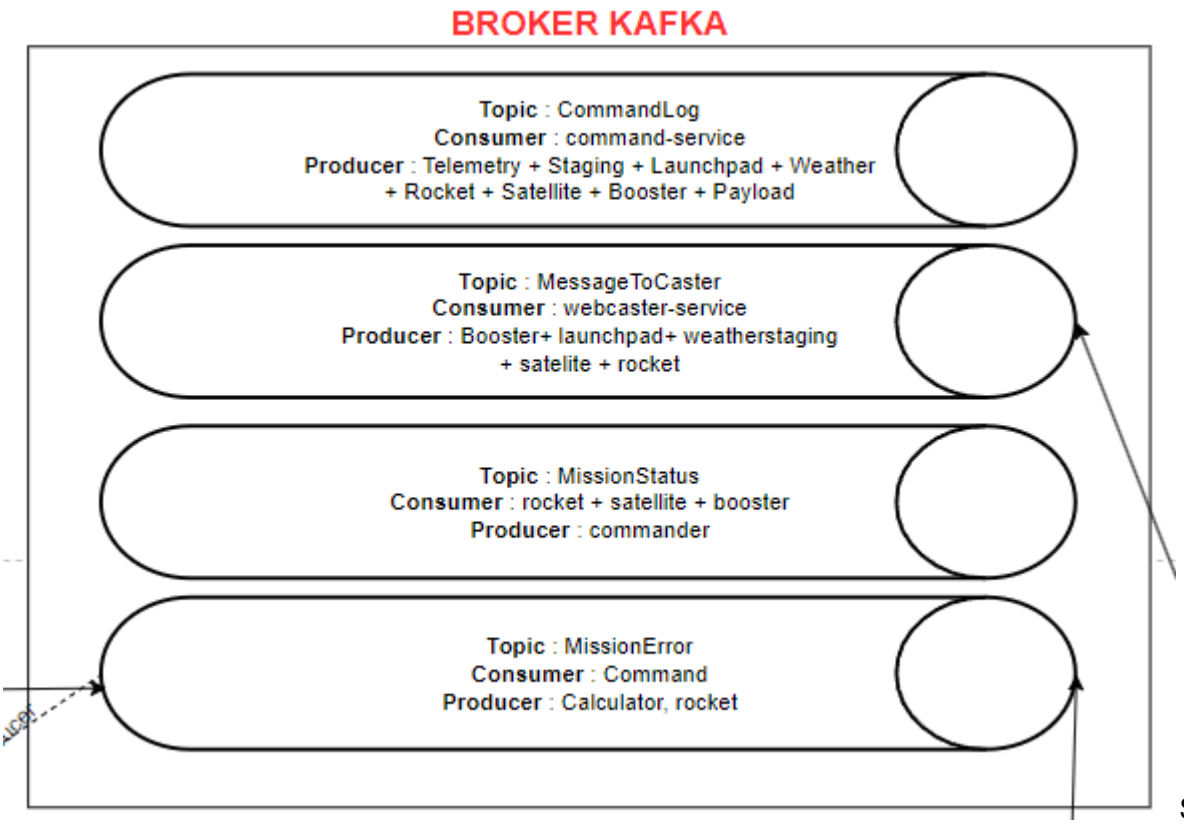
Par conséquent, nous avons décidé d'ajouter un nouveau service dédié exclusivement à l'analyse des données en temps réel pour détecter d'éventuelles erreurs durant la mission. De plus, nous avons mis en place un système de réplication de la base de données en lecture, selon un mode master-slave. Cela nous permet de mieux gérer un grand nombre de métriques, ce qui est essentiel dans des cas d'utilisation plus réalistes. Le CQRS (Command Query Responsibility Segregation) nous permet ainsi de faire évoluer notre système et garantit une meilleure résilience de nos services, ayant chacun moins de responsabilités.

Envoie des métriques de l’astronaute via un topic kafka



Les données de santé de l'astronaute sont plus critiques que les données métriques liées à la fusée. En effet, si le service de Télémétrie venait à tomber pendant quelques instants, les données envoyées par la fusée pourraient être temporairement négligées, car sur la base d'un large ensemble de données, nous pouvons suivre l'état de la mission. Cependant, pour ce qui est de l'humain, les choses peuvent évoluer très rapidement. L'utilisation de Kafka nous permet de conserver un historique des métriques de l'astronaute, même en cas de défaillance du service de Télémétrie.

Event Sourcing



S

La meilleure réponse à la conception du système d'envoi de la fusée MarsY était l'adoption d'une architecture basée sur les événements. Une série d'événements déclencheurs est à l'origine de l'envoi de la fusée dans l'espace. Pour cela, nous avons mis en place un Bus de

données Kafka, qui nous permet de transmettre des événements et des données au sein de notre système.

Avantages et inconvénients dans notre contexte

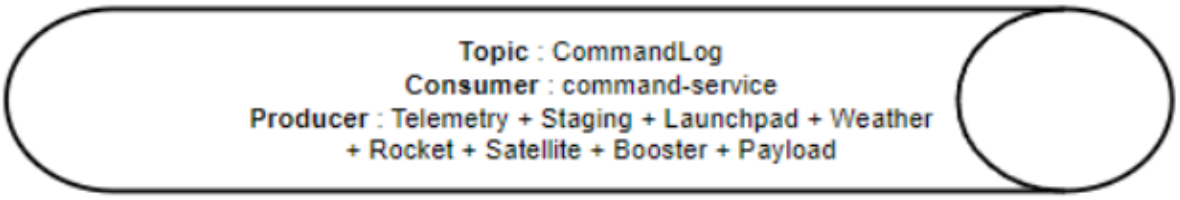
Notre utilisation principale de Kafka a été pour l'envoi de données considérées importantes (cf. topic : Vital Astronaut), ainsi que pour la récupération de données à travers tout le système. Concernant les déclencheurs faisant progresser la mission, nous avons décidé de maintenir un modèle REST pour deux raisons :

- **Sur le plan organisationnel** : Repasser tout en événementiel nécessiterait un grand refactoring que nous n'avions pas le temps de réaliser.

- **Gestion de la Temporalité** : L'utilisation d'un modèle événementiel introduit une marge asynchrone et une méthode d'envoi d'événements de type "fire and forget". Nous craignions, par exemple, que le payload se détache avant le booster, posant ainsi un problème de temporalité dans l'exécution des étapes de la mission. Avec REST, nous nous assurons que la demande de détachement a bien été envoyée avant de passer à la suite. L'action synchrone de "REST/PUT" garantit son exécution, et en cas d'échec, nous retentons jusqu'à la réussite de l'opération, permettant ainsi de poursuivre la mission.

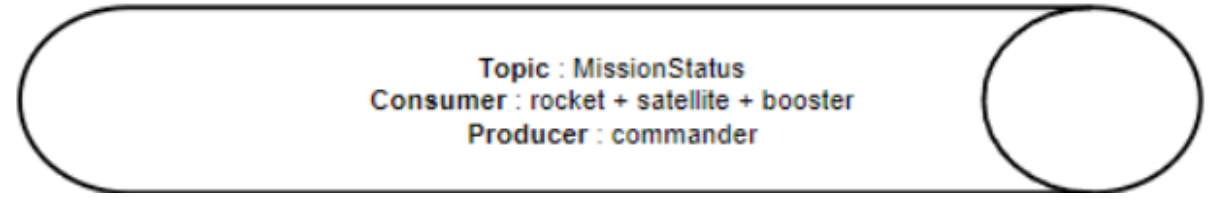
Les avantages de l'utilisation de Kafka résident dans le fait que les données envoyées persistent jusqu'à leur consommation. L'usage de topics nous a permis de faire produire des données par plusieurs services, qui peuvent être consommées par d'autres services. Dès qu'un nouveau service a besoin de consommer ou d'envoyer des données, il lui suffit de se synchroniser avec le topic correspondant. Cette simplification est avantageuse car elle permet de réduire le nombre de dépendances évitables entre les différents services. En effet, avoir plusieurs services envoyant des requêtes REST vers quelques autres peut complexifier le système et augmenter le risque d'erreurs.

Les topics

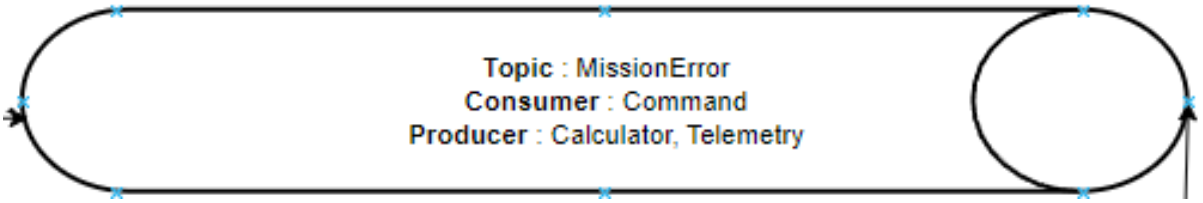


Permet à l'ensemble des services de notre système d'envoyer des logs de leur état vers le service "Commander" qui va pouvoir ensuite les afficher et les stocker.

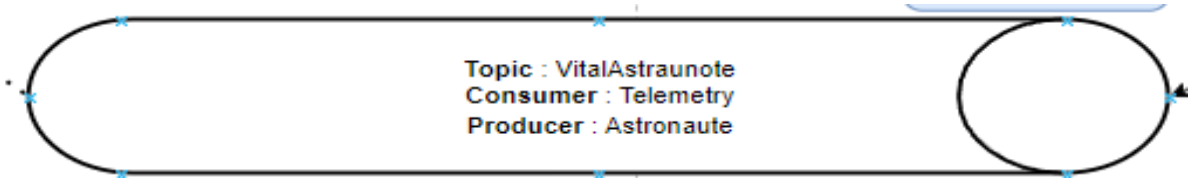
Permet à nos services “visibles durant la mission” d’envoyer des Strings représentant les différents commentaires qui seront faits par le WebCaster durant la mission.



Permet d’envoyer un signal de l'état de la mission au service responsable de la génération de métriques (service mock), afin que les données stockées soient associées à une des multiples missions pouvant être effectuées à la suite.



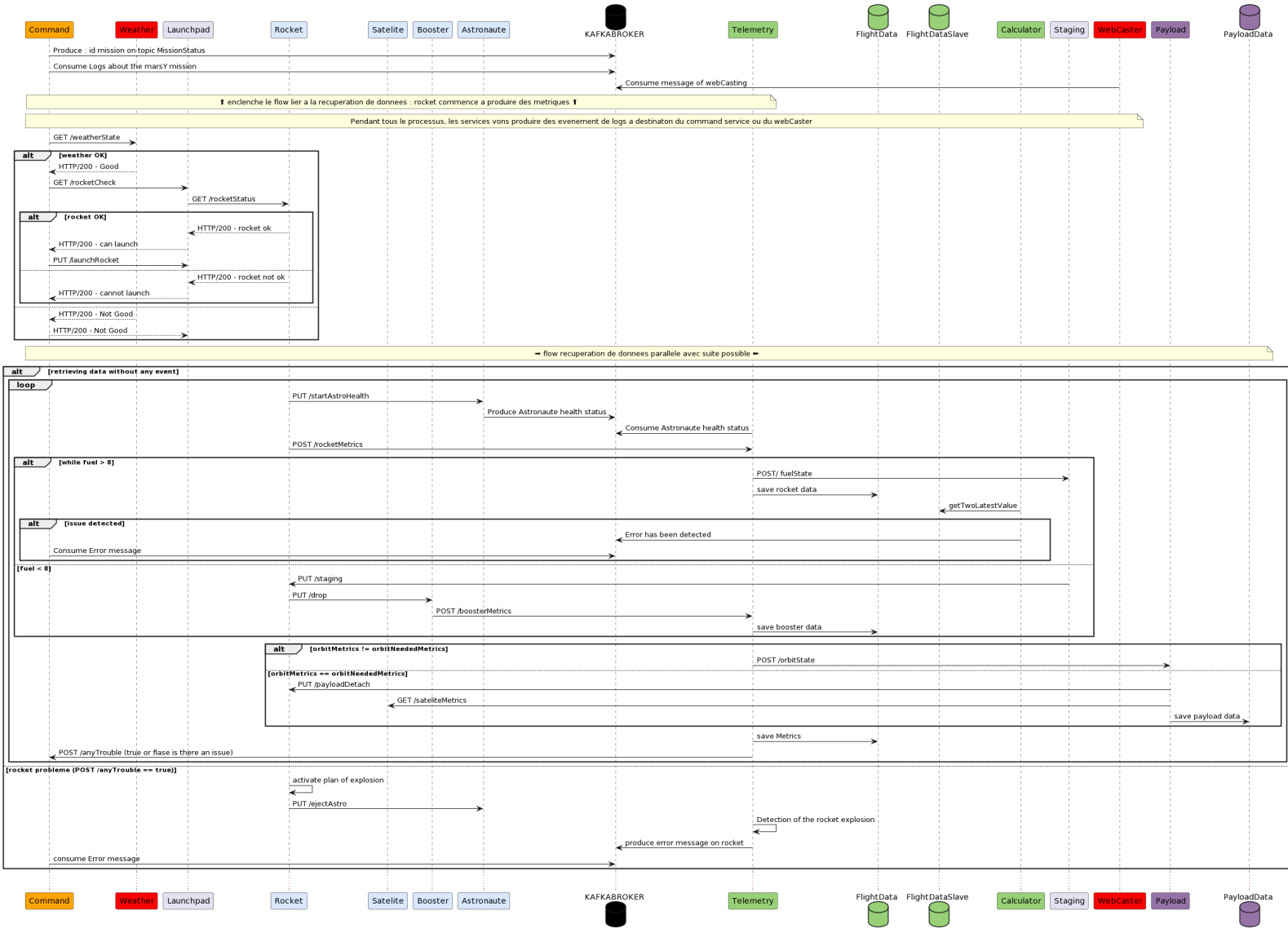
Permet d’envoyer des messages d’erreurs détectés vers le service Commander afin qu’il puisse être notifié et qu’il prenne éventuellement une décision.



Permet d’envoyer les données de métriques liées à l’astronaute.

Scénario de démonstration

Dans nos scénarios, nous présentons les différents logs de nos services en lisant les logs issus du centre de commandement, qui témoignent du déroulement de la mission. Nous affichons, à côté de ceux-ci, les logs des commentaires faits par le WebCaster. Finalement, nous montrons les métriques en temps réel fournies par la fusée et par les astronautes sur la partie basse de la console du run.sh. Le tout a été réalisé grâce à Tmux, sous linux .



Déroulement normal de la mission

- Le scénario commence par le service de commandement qui vérifie auprès du service météorologique, si les conditions sont ensoleillées. Une fois cette première condition validée, la seconde consiste à vérifier si la fusée est en état de se lancer. Finalement, une fois toutes les conditions remplies, le lancement peut commencer.
- Une fois la fusée lancée, nous commençons à récupérer des données sur celle-ci et son équipage. Parmi les données de la fusée, il y a notamment le niveau de carburant, l'altitude et la vitesse. Ces données nous permettront de déterminer le moment idéal pour effectuer les séparations de la fusée.
- Au cours de sa montée vers l'espace, arrivée à une certaine pression atmosphérique, la vitesse sera réduite de 20%. Lorsque le niveau de carburant sera passé en dessous de 8%, la séparation des propulseurs de la fusée sera déclenchée. Un second processus consistera ensuite à faire atterrir le propulseur et à récolter les informations qui lui sont liées.
- Arrivée à un certain stade de la mission, la vitesse et l'altitude seront adéquates pour déclencher l'envoi du payload dans l'espace. De nouvelles métriques sur le payload seront alors récupérées par le service de payload.
- Après le détachement du payload, pendant la phase restante de la mission, l'astronaute sera interviewé par les WebCasters sur Terre.

Explosion de la fusée

Une fois le scénario précédent terminé, nous enclenchons un nouveau scénario avec l'explosion de la fusée. En effet, 30 secondes après le lancement de la fusée, nous simulerons une erreur matérielle au niveau de la fusée

- À la détection d'un grave problème, la fusée enverra un signal vers le cockpit.
- À la réception du signal d'erreur, les astronautes seront éjectés de la fusée dans une capsule qui retombera sur Terre, la chute étant amortie grâce à des parachutes.
- La télémétrie détectera un problème dans le matériel de la fusée et enverra alors un message d'erreur vers le service de commandement indiquant que la fusée s'est auto-détruite.

Faiblesses de l'architecture

Il y a tout de même des faiblesses dans notre architecture :

Single Point of Failure

Notre service rocket est au cœur de toutes les actions qui se déroulent pendant la mission. Bien que nous ayons créé de nouveaux services capables de prendre la relève dans l'envoi de métriques, ils ne commencent à en envoyer que selon une logique temporelle, après que le service Rocket ait débuté ses transmissions. Or, le service rocket est stateful pour des besoins de mock. Si ce service venait à tomber, notre système serait compromis car il faudrait le réinitialiser aux valeurs par défaut du service de la Rocket.

Un surplus de responsabilités

Le service Payload a de multiples responsabilités et pourrait être divisé. Il est chargé de la consommation des métriques de la fusée pour déterminer quand le payload peut être envoyé dans l'espace, et il a aussi pour rôle de récupérer les métriques liées à celui-ci. Ils pourraient être séparés en mode CQRS, à l'image de ce qui a été réalisé pour le service de télémétrie.

Un grand nombre de données dans une même base de données

Actuellement, les données de santé de l'astronaute ainsi que les données métriques de la fusée et du propulseur sont toutes stockées dans la même base de données "Flight Data". Cela pourrait poser problème, car ces données pourraient saturer la base de données. Le choix de cette base a été PostgreSQL, afin de répartir les différentes données dans des tables distinctes. Néanmoins, à l'instar des autres bases de données, MongoDB pourrait être une option viable pour assurer une haute disponibilité du service.

Equipe B : répartition et tâche

Ayman Bassy 100 pts :

- Service Telemetry + Calculator
- Service Astronaut
- Service Launchpad
- DB flight Data
- Mise en place de Kafka

Igor Melnyk 100 pts :

- Service Command
- Service Satellite
- Service Payload
- DB Payload + Logs
- Mise en place des logs

Tobias Bonifay 100 pts :

- Service Rocket (MaxQ)
- Service Weather
- migration vers linux
- Mise en places des logs
- CI

Mathieu Schalkwijk 100 pts :

- Service Rocket
- Service Booster
- Mise en place Logs
- Mise en place socket
- Mise en place Kafka

Annexes

Code plantUML du diagramme de séquence

Meilleur visibilité sur le site : https://www.planttext.com/

```
@startuml
participant Command #orange
participant Weather #red
participant Launchpad

participant Rocket #DAE8FC
participant Satellite #DAE8FC
participant Booster #DAE8FC
participant Astronaute #DAE8FC

database KAFKABROKER #black

participant Telemetry #97D077
database FlightData #97D077
database FlightDataSlave #97D077
participant Calculator #97D077

participant Staging
participant WebCaster #red

participant Payload #9673A6
database PayloadData #9673A6


Command->KAFKABROKER: Produce : id mission on topic MissionStatus
Command->KAFKABROKER: Consume Logs about the marsY mission
WebCaster->KAFKABROKER: Consume message of webCasting
note over Command, Telemetry: ⬆️ enclenche le flow lier a la recuperation de donnees : rocket commence a produire des metriques ⬆️
note over Command, WebCaster: Pendant tous le processus, les services vons produire des evenement de logs a destinaton du command service ou du webCaster
Command->Weather: GET /weatherState
alt weather OK
  Weather-->Command: HTTP/200 - Good
  Command->Launchpad: GET /rocketCheck
  Launchpad->Rocket: GET /rocketStatus
  alt rocket OK
    Rocket-->Launchpad: HTTP/200 - rocket ok
    Launchpad-->Command: HTTP/200 - can launch
    Command->Launchpad: PUT /launchRocket

  else
    Rocket-->Launchpad: HTTP/200 - rocket not ok
    Launchpad-->Command: HTTP/200 - cannot launch
  end
else
  Weather-->Command: HTTP/200 - Not Good
  Command-->Launchpad: HTTP/200 - Not Good
end

note over Command, PayloadData: ➡️ flow recuperation de donnees parallele avec suite possible ⬅️

alt retrieving data without any event
  loop
    Rocket -> Astronaute: PUT /startAstroHealth
    Astronaute -> KAFKABROKER: Produce Astronaute health status
    Telemetry -> KAFKABROKER: Consume Astronaute health status
    Rocket->Telemetry: POST /rocketMetrics
    alt while fuel > 8
      Telemetry-> Staging: POST/ fuelState
      Telemetry->FlightData: save rocket data
      Calculator->FlightDataSlave: getTwoLatestValue
      alt issue detected
        Calculator->KAFKABROKER : Error has been detected
        Command -> KAFKABROKER : Consume Error message
      end
    else fuel < 8
      Staging->Rocket: PUT /staging
      Rocket -> Booster: PUT /drop
      Booster->Telemetry: POST /boosterMetrics
      Telemetry->FlightData: save booster data
    end
    alt orbitMetrics != orbitNeededMetrics
      Telemetry-> Payload: POST /orbitState
    else orbitMetrics == orbitNeededMetrics
      Payload->Rocket: PUT /payloadDetach
      Payload->Satelite: GET /sateliteMetrics
      Payload->PayloadData: save payload data
    end
    Telemetry-> FlightData: save Metrics
    Telemetry->Command: POST /anyTrouble (true or flase is there an issue)
  end
else rocket probleme (POST /anyTrouble == true)
  Rocket->Rocket : activate plan of explosion
  Rocket->Astronaute : PUT /ejectAstro
  Telemetry->Telemetry: Detection of the rocket explosion
  Telemetry->KAFKABROKER : produce error message on rocket
  Command->KAFKABROKER : consume Error message
end
...


@enduml
```