EQUIPE F
*Hadil AYARI*
*Nicolas GUIBLIN*
*Chahan MOVSESSIAN*
*Floriane PARIS*

# Rapport Service-oriented Architecture

## Rendu Final

*Rendu du 19 novembre 2024*

# Table des matières

# Notre projet

## Exécution

- ./prepare.sh : préparation des images
- ./run.sh : exécution des scénarios

Le premier scénario va s'exécuter jusqu'à ce que, le payload soit déployé à la bonne orbite, et le booster s'est posé.

Le second scénario va s'exécuter jusqu'à ce qu'une erreur pendant le détachement du premier étage provoque la destruction de la fusée.

Le troisième scénario vu s'exécuter jusqu'à ce qu'un problème de fuite de carburant pousse le chef de la mission à déclencher la destruction de la fusée manuellement.

## Users stories

Les 18 Users Stories décrites dans le sujet ont été implémentées.

Les Users stories supplémentaires que nous avions choisies sont :

- Point de rendez-vous : Envoi de deux fusées à la même position
- Mission Circum-Lunaire: Décollage de la Terre, injection en orbite autour de la terre, puis poussée vers une orbite croisant la lune avant de retourner sur Terre grâce à une trajectoire de retour libre.

Cependant par manque de temps et problèmes rencontrées expliqués plus loin, nous avons décidé de nous concentrer sur la bonne exécution des US demandées et préféré ne pas rajouter d'US supplémentaires.

## Informations supplémentaires

Le projet est fait avec NestJS et NodeJS ainsi qu'un service réaliser en python.

Vous pouvez également trouver les logs détaillés dans le service *scripts* dans logs/*.log.

# Diagramme de l’architecture actuelle

## Diagramme

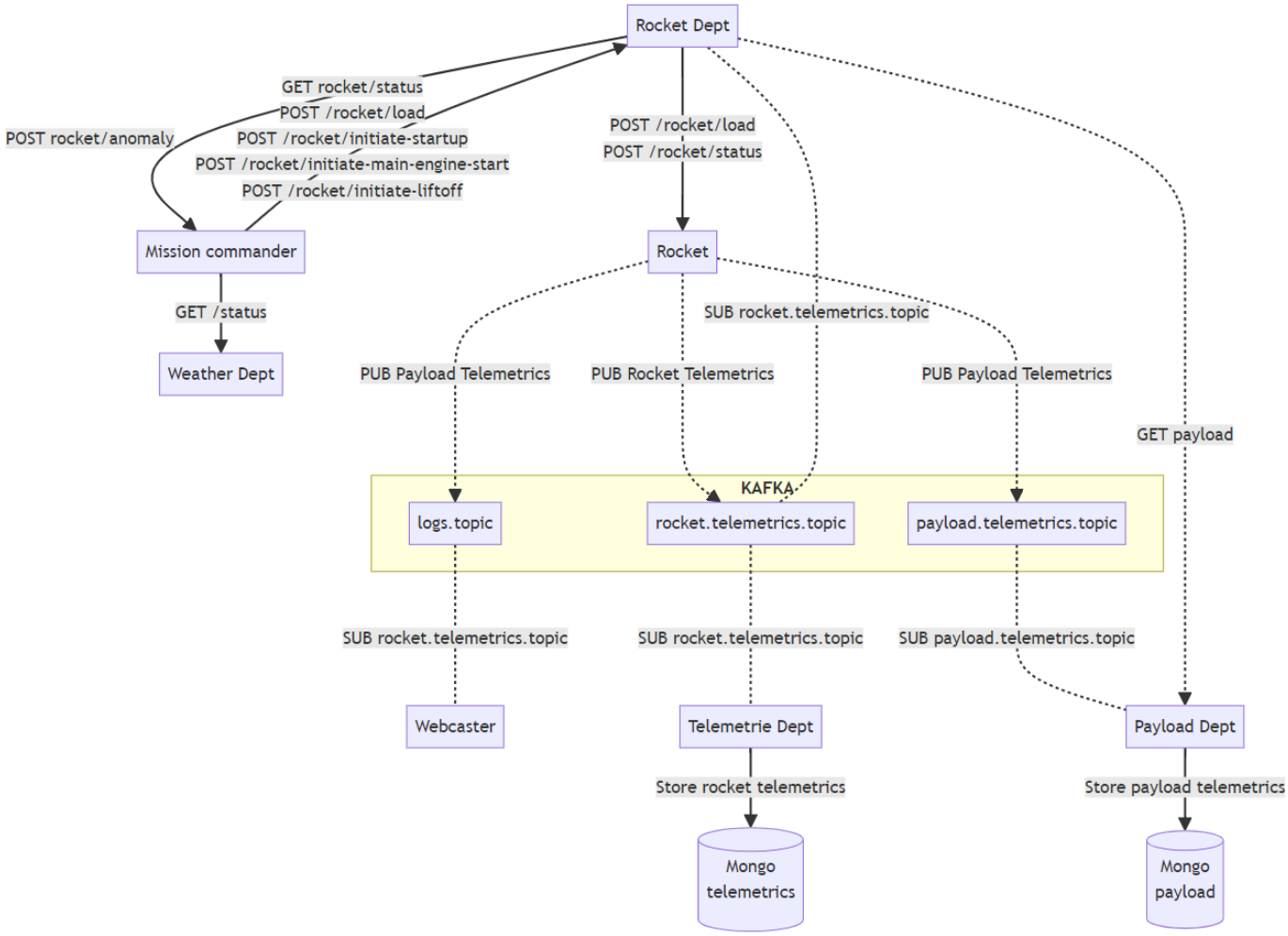


*Diagramme d'architecture*

## Explications de l’architecture actuelle

Nous avons commencé par mettre au centre de notre architecture le service mission-commander, qui récupère des informations du département weather et du département rocket. Seul le département rocket est en communication directe avec l’objet Rocket, ainsi le mission-commander doit passer par le département rocket s’il veut agir sur l’objet Rocket.

Ensuite, les informations de l’objet Rocket sont envoyées au bus Kafka. On sépare les informations concernant l’objet Rocket entier, dans le topic rocket.telemetrics.topic, et les informations qui concernent uniquement le payload, qui sont dans le topic payload.telemetrics.topic. Le département payload est abonné au topic payload.telemetrics.topic, ce qui lui permet de recevoir et de stocker dans sa base de données toutes les mises à jour concernant le payload. De même, le département telemetrie est abonné au topic rocket.telemetrics.topic afin de recevoir et de stocker dans sa base de données toutes les informations relatives à la fusée.

Les logs concernant l’état de la fusée sont publiés de l’objet Rocket au topic logs.topic. Le service Webcaster est abonné à ce topic afin d’être informé en temps réel de l’état de la fusée, ainsi que mission-controller.

Voici les informations reçues par Webcaster :

---

**Launch Event Messages**

| Rocket Name | Status | Altitude | Stages Info | Payload Info |
|---|---|---|---|---|
| MarsY-1 | **Liftoff** | 0 meters | Stage 0: Liftoff, Fuel: 1760 Stage 1: Liftoff, Fuel: 3000 | Payload: Liftoff, Weight: 100 kg |
| MarsY-1 | **In flight** | 14.715 meters | Stage 0: In flight, Fuel: 1720 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 58.86 meters | Stage 0: In flight, Fuel: 1680 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 132.435 meters | Stage 0: In flight, Fuel: 1640 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 235.44 meters | Stage 0: In flight, Fuel: 1600 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 367.875 meters | Stage 0: In flight, Fuel: 1560 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 529.74 meters | Stage 0: In flight, Fuel: 1520 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 721.035 meters | Stage 0: In flight, Fuel: 1480 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 941.76 meters | Stage 0: In flight, Fuel: 1440 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |
| MarsY-1 | **In flight** | 1191.915 meters | Stage 0: In flight, Fuel: 1400 Stage 1: In flight, Fuel: 3000 | Payload: In flight, Weight: 100 kg |

# Explications des scénarios

Tous les scénarios réalisés commencent de la même manière, lors de l'exécution leur différence commence lorsque la fusée est en vol.

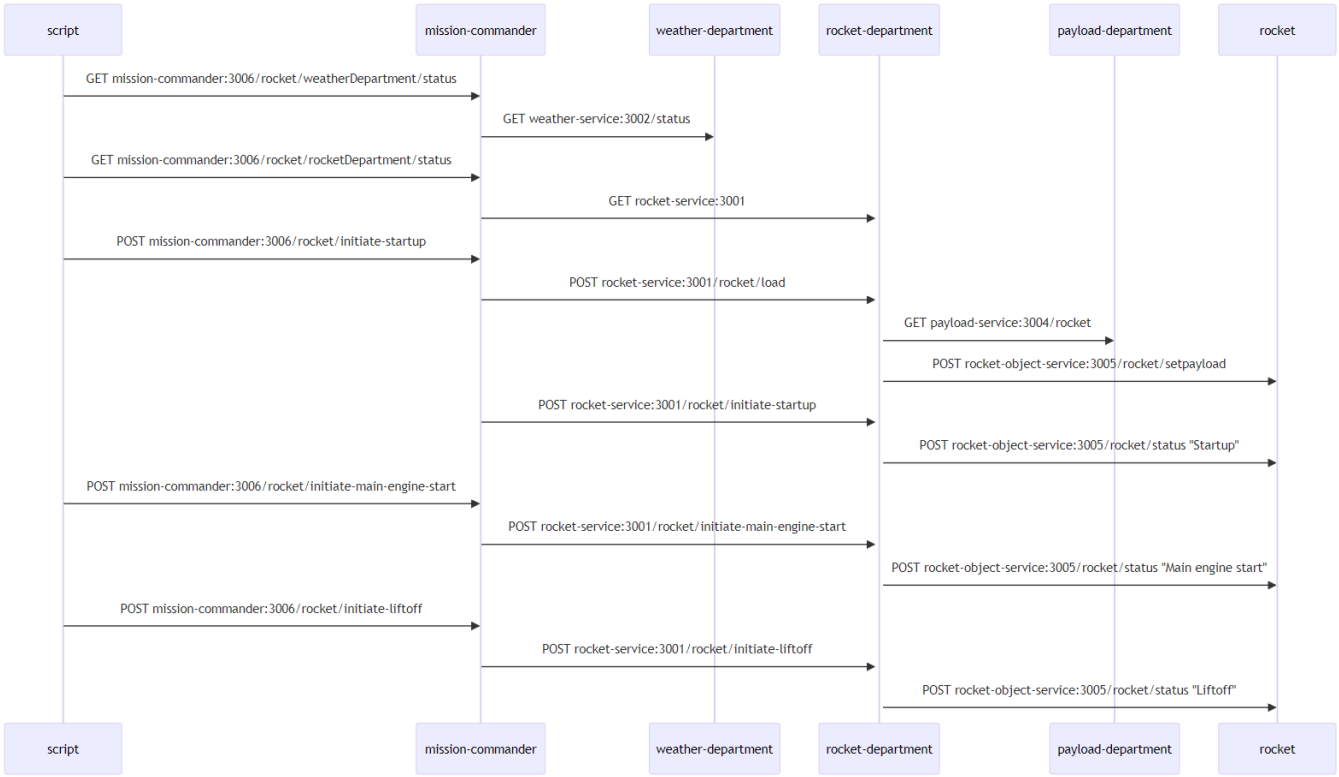


*Diagramme de séquence 1: lancement de la fusée*

Le script représente d'abord le mission-commander, puisqu'il va demander au services rocket-department et weather-department si le status est ok. Si c'est le cas il va envoyer l'instruction 'initiate-stratup' au rocket-department (qui s'occupe à ce moment-là de charger le payload dans la rocket pour simplifier le flow -- POST rocket/setpayload ). Le mission-commander va ensuite pouvoir initier le la séquence de lancement, faire démarrer le moteur puis donner l'ordre de décollage en aux instants corrects de la séquence de décollage.

Ensuite le scénario va représenter l'ordinateur de bord de la rocket, il va donc poster au service rocket-object sur la route /rocket/status. Le principe est que l'ordinateur poste une rocket sous la forme d'un objet JSON contenant les infos de la rocket et le status de vol suivant, il met à jour également le temps interne de la rocket ce qui permet au service roket-object de mettre à jour la position et la vitesse de la rocket en fonction de l'accélération et du temps.

La séquence de démarrage est la même pour les 3 scenarios.

## Scénario 1 : Insertion en orbite

La rocket va décoller et consommée son fuel jusqu'à ce que le premier étage ne garde qu'un dixième de son fuel, à ce moment-là l'étage se sépare automatiquement (géré dans rocket-object-service). Ensuite l'étage supérieur de la rocket va continuer sa course pour aller déposer le payload en orbite, après avoir au préalable atteint MaxQ, séparer la coiffe et éteins le second moteur.

Le scénario s'arrête lorsque le booster (premier étage) a atterri de nouveau.

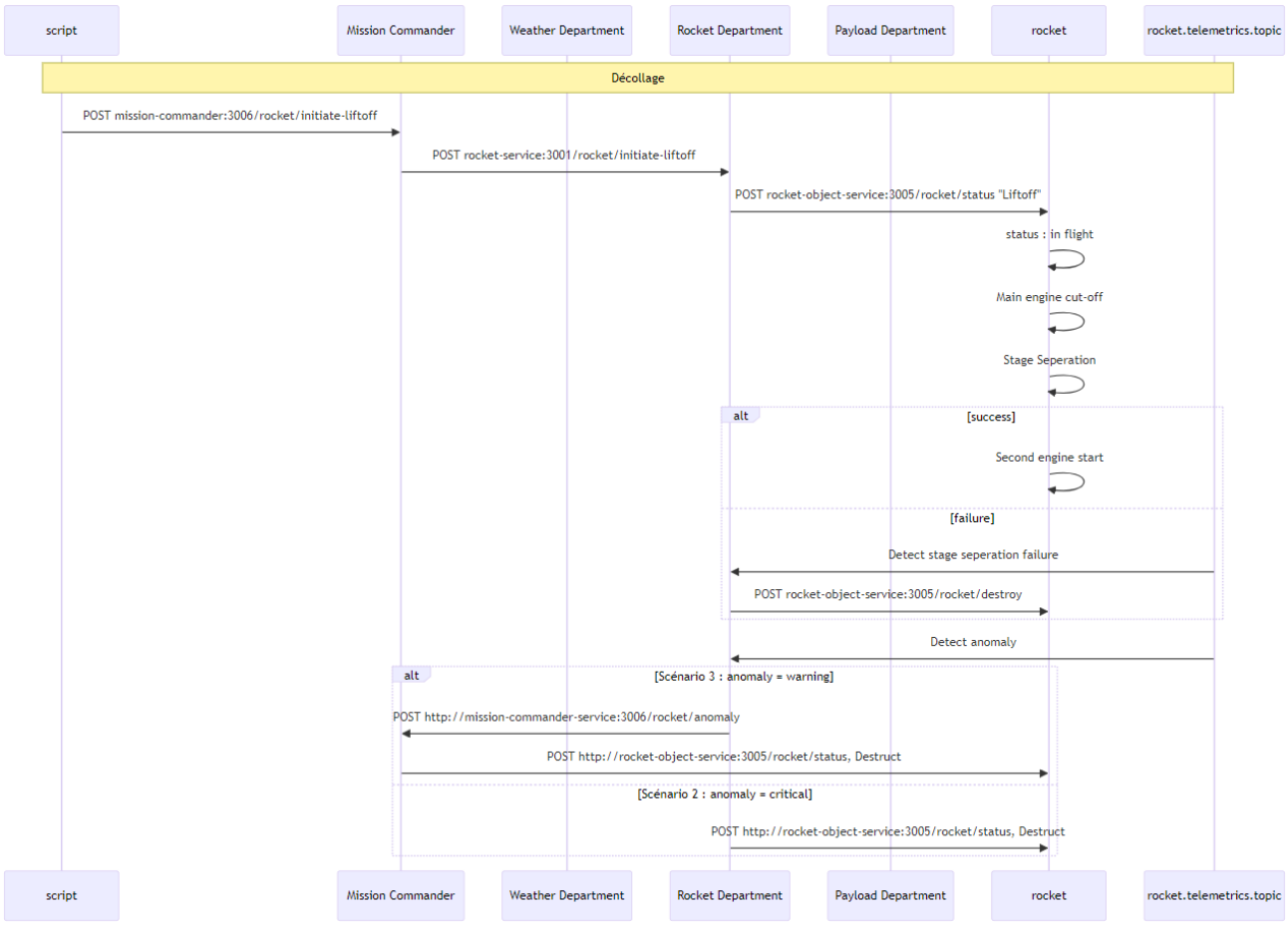


*Diagramme de séquence 2: Cas de destruction de la fusée*

## Scénario 2 : Erreur pendant la séparation

Ce scénario se comporte comme le précèdent jusqu'à la séparation du premier étage, cependant la rocket détecte que cette séparation ne se passe pas comme prévu et rocket département prend la décision de le détruire. La rocket est également équipée d'une fonctionnalité d'autodestruction en tant que mesure préventive

## Scénario 3 : Destruction manuelle

Dans ce scénario c'est mission-commander qui prend la décision d'ordonner la destruction de la rocket en fonction de la sévérité des anomalies reçues.

# Notre approche

## Nos choix

Nous avons fait le choix d'implémenter un service par département, car les départements sont déjà découpés de manière logique en termes de besoin et de mission principale.

Quant à Kafka, utiliser trois topics nous permet de séparer les informations selon les besoins des différents services.

Nous avons utilisé Mongo DB, qui fonctionne correctement pour stocker des documents.

## Nos contraintes

Lors de la mise en place de l'US 16, c'est à dire pour lancer plusieurs fusées à la suite, nous avons été bloqués par notre implémentation de départ. Nous avons detectés que plusieurs services n'étaient pas stateless notamment rocket-object-service qui est en charge de géré la simulation. Nous avons donc perdu beaucoup de temps pour corriger ces services, réimplémenter les fonctionnalités et corriger les scénarios.

Cette contrainte de temps nous a également empêché de mettre en place la mise en place de rendez-vous orbital et la mission circum-lunaire, nos US supplémentaires choisies.

## Les limites de notre implémentation

La limite principale de notre implémentation est la résilience des services. Les évènements Kafka ne sont pas correctement gérés dans tous les services, nous n'avons pas implémenter de fonctionnement pour relire les événements en cas de panne/reboot d'un service. Cela suffit à l'échelle de ce projet mais cela pose de nombreux problèmes.

De ce fait, le passage à l'échelle pourrait être difficile à mettre en place.

Nous n'avons pas non plus consolidé rocket-object-service, qui est le service les plus utilisé de notre implémentation et donc le plus fragile en cas de problème de passage à l'échelle. Même si de par son implémentation il peut déjà gérer plusieurs rocket et peut être utilisé pour reprendre un scénario en cours ce qui simule le fait qu'en cas de reboot de l'ordinateur de bord, le service pourra continuer de fonctionner dès le redémarrage.

## Notre compréhension du sujet

Ce sujet met en évidence l'efficacité d'une architecture micro-service, dans ce cas ou beaucoup de service bien différent doivent communiquer entre eux et recevoir beaucoup d'informations, cette architecture permet de passer à l'échelle plus facilement les services nécessaires tout en gardant un fonctionnement optimal. Pour finir ce sujet demande une grande compréhension des principes des architectures micro-service et particulièrement des principes stateless.