Présentation du projet Marsy

Groupe C

Auteurs :
Théophile YVARS - Hamza ZOUBAIR - Roméo David AMEDOMEY -Jules PRINCE

19 novembre 2023

Table des matières

1 Architecture

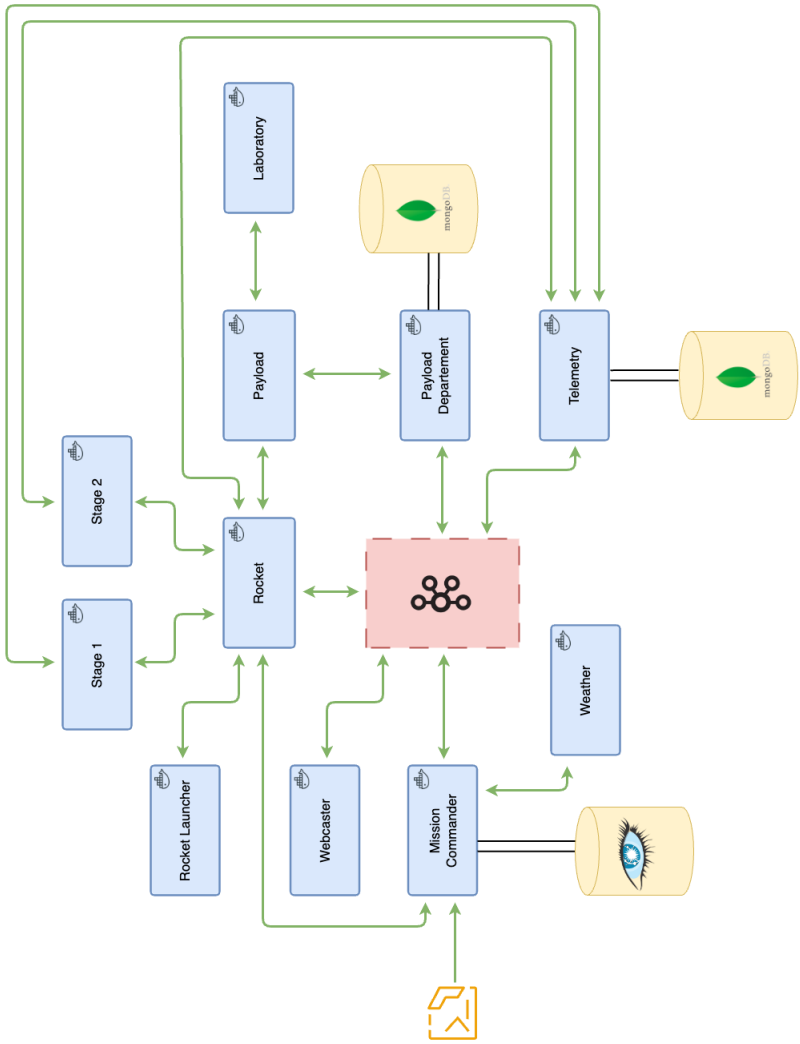

FIGURE 1.0.1 – Diagramme d'Architecture

Voici notre diagramme d'architecture. Ce diagramme a beaucoup évolué en fonction des nouvelles Users Storys et des problèmes de conceptions que l'on rencontrait. Chaque Micro-Service représente une partie physique de la fusée ou bien du centre de contrôle.

Le bus Kafka représente le lien physique qui lie les différents départements qui gere le déroulement de la mission.

Ce bus kafka nous as posé problème au cours du projet. En effet suite à une mauvaise compréhension des consignes, nous avions mis le bus Kafka au centre de notre projet.

Cette conception nous as permis de comprendre ce que c'est d'implémenter Kafka. C'est une opération complexe et qui prends du temps. De plus il apporte une logique événementiel et nous étions partis pour adapter tout notre code ainsi. Heureusement nous avons réussi à prendre du recul et à réfléchir sur quel service ont besoin de Kafka. Nous sommes alors parti sur l'hypothèse que notre bus kafka représenterai un câble liant tous les services présent au sol entre eux. Le services de telemetry qui reçoit un flux de donnée

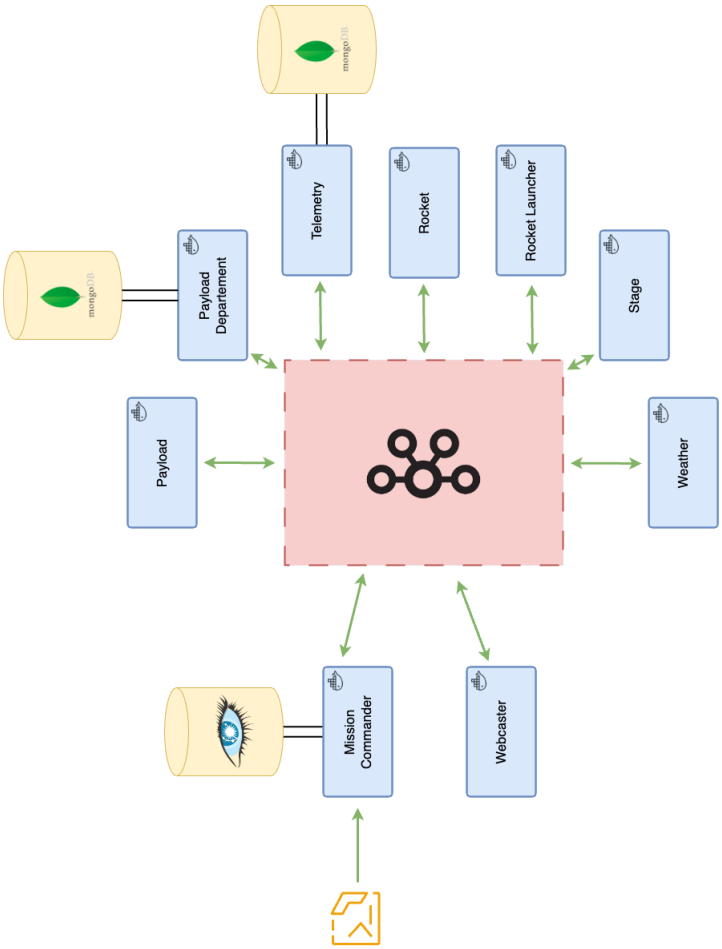

FIGURE 1.0.2 – Diagramme d'Architecture avec Kafka au centre

important il utilise donc le bus Kafka pour produire sur son topic tous les messages de telemetry venant des stages, du payload ou de la fusée.

La base de données Cassandra permet au service Mission Commander, recevant un grand nombre d'événements Kafka, de stocker rapidement ces données dans sa base de données. MongoDB a été choisi comme solution NoSQL performante en raison de sa structure de document, idéale pour notre modèle de données Telemetry ou Position pour le payload.

Notre architecture est cohérente et facile de prendre en main, nos servies communique efficacement entre eux et s'échange uniquement les données nécessaire. Nous n'avons pas de Single Point Of Failure et le responsabilités sont correctement réparti.

Nous pouvons critiquer l'absence de mécanisme prévenant la résilience de notre système car faute de temps.

2 Découpage des microservices :

2.1 Description des services :

1) Mission Commander : c'est le service qui gère la mission, il va permettre de vérifier l'état du Weather et de la fusée pour commencer la mission, ensuite il pourra donner des ordres de destruction. Il recoit tous les évènements de la Rocket, des stages et du Payload. Il est capable de stocker et consulter les logs de la mission.

2) Rocket : C'est le service qui modélise la fusée, et qui contient un simulateur qui va imiter l'avancement de la fusée.

3) Telemetry : Le rôle de ce service est de questionner la fusée et les 2 stages pour récupérer leurs métriques. Ce choix est fait, parceque la communication en REST est plus envisageable en espace.

4) Web Caster : C'est le service qui va recevoir tous les évènements qui se passent dans le système (Rocket, Stages, Payload, ...).

5) Stage : Ce service va récupérer les informations des deux stages de la fusée après la séparation. Ensuite, il va créer un simulateur qui va avancer les métriques des stages. Il envoie les métriques des stages à Telemetry qui va les stocker dans sa base de données.

6) Payload et Payload Department : Le premier service fonctionne comme le service Stage, il va simuler le payload après sa livraison. Payload Department va communiquer avec le service Rocket pour approuver le dépot du payload et ensuite commencer la telemetry du payload.

7) Weather : c'est le service qui communique l'état du climat pour commencer la mission.

8) Rocket Launcher : Service qui simule le lancement de la fusée après la confirmation du Mission Commander.

2.2 Communication entre les services :

Dans notre architecture nous utilisons deux styles de services pour couvrir toutes les fonctionnalités. Nous avons pris en considération la répartition physique des différents services. On a supposée que la fusée, les stages et le payload ne peuvent plus communiquer en événementiel après le décollage, donc la communication avec cette dernière se fait en appels REST dans ce cas. Ce choix revient au difficulté de communiquer à travers le bus Kafka à cause des contraintes réseau dans l'espace.

La Rocket va commencer au début par envoyer les évènements suivants à travers le Bus Kafka :

1) Rocket preparation

2) Rocket on internal power

3) Startup

4) Main engine start

5) Liftoff / Launch

Ces évènements seront envoyés dan le topic "launch" et ils seront consommés par Mission Commander et Web Caster. Tous ces évènements se passe sur sol.

Une fois le décollage effectué, le simulateur de la fusée générera les métriques, les transmettant ensuite au service Telemetry. Ce dernier utilisera le bus Kafka pour relayer ces métriques vers Mission Commander.

Ce processus permet à Mission Commander de récupérer l'ensemble des métriques, une nécessité pour le contrôle global de la mission.

Pour l'envoie des métriques, il va s'effectuer à travers le topic "monitoring". Cette séparation des topics est fonctionnelle, puisque le topic "launch" va gérer les évènements clé de la mission et "monitoring" pour l'envoie des métriques de Rocket et Stage. Pour réduire la charge sur le service Telemetry, Payload Department va récupérer les métriques du payload et les envoyer dans le bus Kafka, elles seront consommées ensuite par Telemetry et Mission Commander. Nous avons choisi de transmettre les statuts du payload via le bus Kafka au lieu de les envoyer directement vers Telemetry, ce qui nous permettra de les récupérer en cas de défaillance du service Telemetry.

Le service Rocket va envoyer les appels REST suivantes au service Telemetry :

7) Main engine cut-off

8) Stage separation

9) Second engine start

10) Fairing separation

11) Second engine cut-off

12) Payload separation/deploy

Ces évènements vont modéliser les actions de séparation des stages et la livraison du payload. Ensuite, Telemetry va produire ces évènements dans le bus Kafka.

2.3 Argumentation :

Nous avons pris la décision de diviser les responsabilités en trois services distincts : Rocket, Stage et Payload. Cette division intervient après la séparation et la livraison du payload. Les deux stages sont gérés de manière indépendante par le service Stage, chacun possédant son propre simulateur, de même pour le payload. Cette approche est pertinente car, après la séparation, la notion de fusée est perdue. Ainsi, notre choix nous permet de conserver les états des stages et du payload, assurant une meilleure gestion des données et des responsabilités.

Nous avons opté pour MongoDB comme base de données pour stocker les métriques en raison de l'absence de relations entre les modèles. De plus, le schéma des métriques de stage et rocket est flexible et extensible, ce qui nous permet d'ajouter de nouveaux attributs au modèle. Ce choix assure la stabilité de la logique métier, même lors du stockage de nouvelles métriques avec un modèle différent.

Nous avons choisi d'utiliser Cassandra comme base de données pour Mission Commander en raison de son aptitude à gérer efficacement l'écriture des logs, notamment lorsqu'il y a plusieurs missions lancées en même temps. Cette base de données offre une grande résilience en matière d'écriture tout au long de la mission, assurant une fiabilité accrue pour la gestion des logs.

2.4 Lecture écriture, CQRS :

- Ecriture

  Dans Payload Department, la responsable peut observer les datas du Payload. Ici, les datas sont directements reçu de paylaod avec des requêtes http. Elles sont sauvegardées en DB. Ensuite, ces informations sont converties en event pour les diffusées dans Kafka.

Dans Telemetry, des requêtes http permettent d'avoir des informations sur les stages et la rocket. Ces datas sont sauvegardées en DB, et sont ensuite converties en event pour les diffusées dans Kafka.

Le rôles de Telemetry est de sauvegarder toutes les informations de Rocket et des Stages. Il produit des events en fonction des informations qu'il recoit. La lecture de ces datas est faite à partir des micro-services qui consomment ces events, avec pour responsabilité de restituer de l'information à un certain moment.

Le rôles de Payload department est de sauvegarder les datas produites par le payload, pour que la responsable de ce service puisse faire ces analyses, mais également de produire des events à partir de ces informations pour que la lecture de ces datas puisses se faire à partir des micros-service consomateur.

- Lecture

  MissionCommander est notre plus gros consommateur. Il stock absolument tout ce qu'il se passe dans la mission pour être en capacité de restituer les logs, les métriques et les events si un contrôle est demandé. Il capture donc tous les events Kafka produits. Webcaster consomme également les events de la mission pour les affichés sur une page web. Il offre une lecture temps réel du déroulé de la mission.

- CQRS

  Il y un un pattern CQRS entre Telemetry, Payload Department, Webcaster et MissionCommander

  Telemetry et Payload Department sont l'écriture, la data stockée sert de backup et la lecture se fait sur d'autre micro-service afin de reduire la charge. L'écriture dans Telemetry se fait toutes les 100ms, donc une quantité significative de datas est géré par ce micros services. Payload Department recoit beaucoup moins d'informations, et une lecture se fait par le responsable de ce départmenet épisodiquement. Malgré tout, d'autre service comme MissionCommander et Webcaster on besoin de ces informations, donc ces datas sont aussi poussées dans le bus Kafka. Ansi, on réduit la charge de lecture/écriture dans ce micro-service.

  Webcaster et MissionCommander sont la lecture. Webcaster affiche en temps réel les events de la missions et MissionCommander sauvegarde toutes les informations issue de la mission. La lecture sur mission commander est rare car elle correspond au souhait d'obtenir ces informations pour investiguer l'origine d'un problème. Donc elle ne se fait qu'une fois et s'il y a un probleme.

2.5 Passage à l'echelle

FIGURE 2.5.1 – Test de charge

Le test est effectuer sur Telemetry. La charge est evalué en considérant que Telemetry récupère les informations de Rocket et des 2 Stages.

Jusqu'a 99 utilisateurs en simultanés, notre application tiens le coup avec un temps de réponse de 6 ms. Mais au dela, l'application perds des données jusqu'a arrivé à saturation.

L'hypothése de cette rupture est : On atteint la limite d'utilisation de Mongo DB. Il est probable qu'avec une cassandra, que la charge soit mieux géré par la DB. Ici, il y a une saturation dans l'utilisation de Mongo DB en écriture.

Puisque que nous avons au maximum 3 utilisateur en simultané (Rocket, Stage1 et Stage2), l'application est

totalement en mesure de supporter la charge avec un temps de réponse moyen évalué à 6ms. Les informations sont demandées aux différents éléments toutes les 100ms.

3 Scénarios

3.1 Prise des statuts pour le décollage

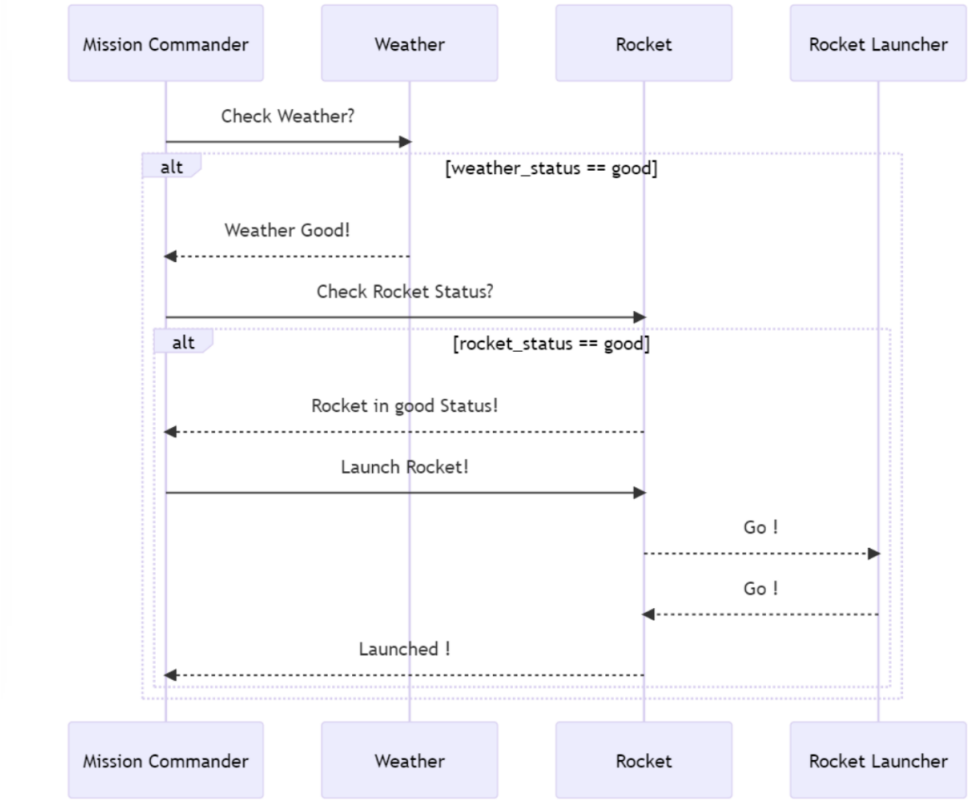

FIGURE 3.1.1 – Diagramme de séquence - Prise des statuts pour le décollage

Pour commencer la mission, le Mission Commander vérifie le statut de la Météo, ensuite on vérifie si le statut de la Rocket est bon et retourne la réponse au Mission Commander. Ce dernier lance un ordre de lancement de la fusée, le service Rocket récupère cette requête et la transmet au service Rocket Launcher qui se charge de lancer la fusée. Enfin, on envoie une confirmation de lancement au Mission Commander.

3.2 Décollage de la fusée

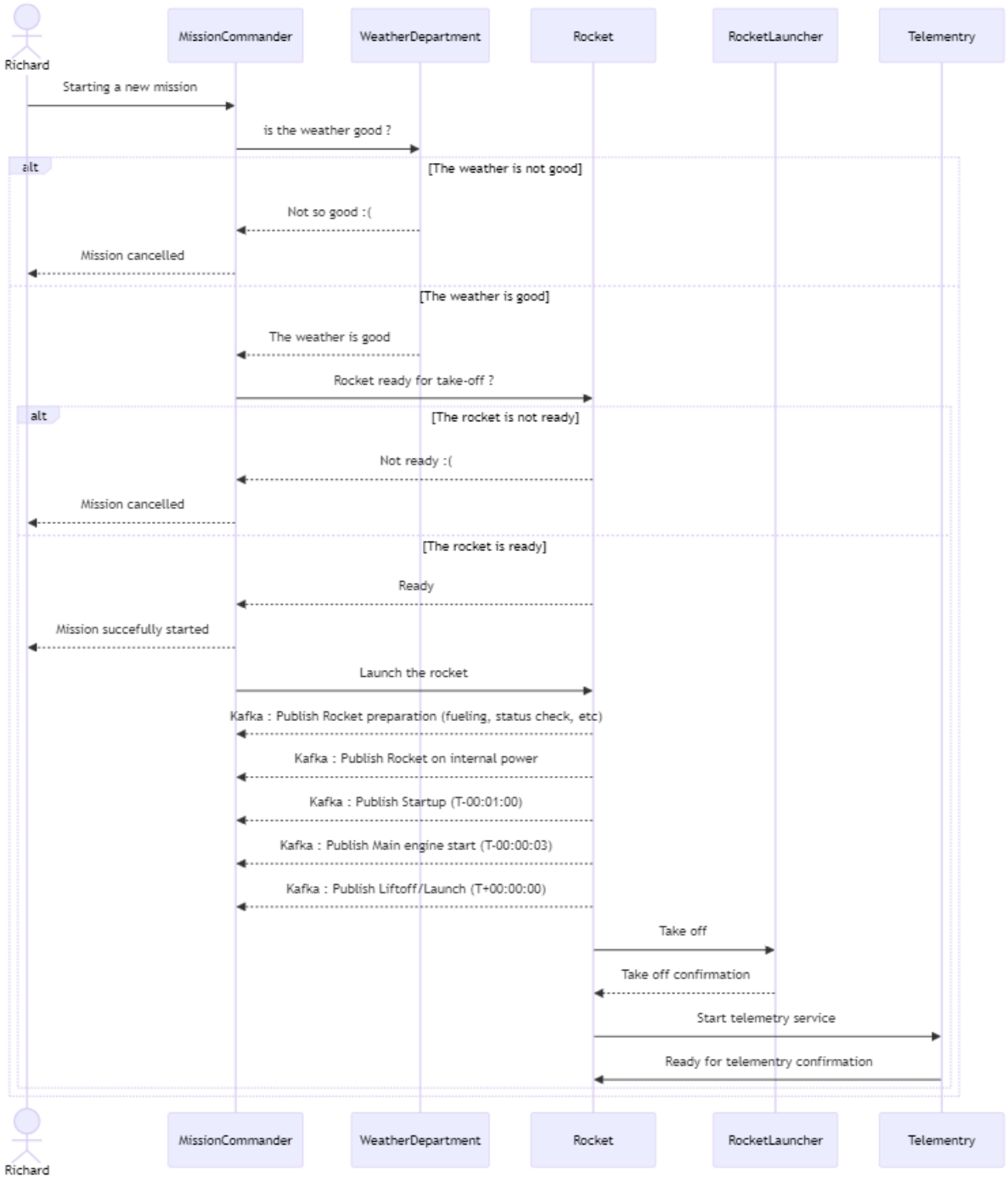

FIGURE 3.2.1 – Diagramme de séquence - Décollage de la fusée

Richard envoie l'ordre de décollage au système, puis il demande à WeatherDepartment son statut. Si les conditions météorologiques sont favorables, Richard interroge ensuite Rocket sur son statut. Si la fusée est prête à décoller, Richard donne l'ordre de décollage à la fusée. La fusée s'envole alors et transmet à Richard l'information que le décollage a eu lieu. Webcaster souscrit également au topic kafka pour recevoir les évènements publié par Rocket lors de son lancement.

3.3 Lancement de plusieurs fusée (multiple launch)

Mission commander est capable de lancer plusieurs fusées (les unes après les autres). En effet, le processus reste le même, il y a le contrôle de l'état de la fusée et du wheather. A la fin de chaque mission, les simulateurs sont remis dans leur état initial (un reset), ce qui permet de pouvoir faire un nouveau lancement. Chaque fusée a un nom (choisi aléatoirement dans une liste prédéfinie) et un identifiant unique qui permet d'indentifier les rockets. Ainsi donc, on peut facilement lancer plusieurs fusées et pouvoir les identifier dans les logs de missions.

3.4 Métrique de la fusée

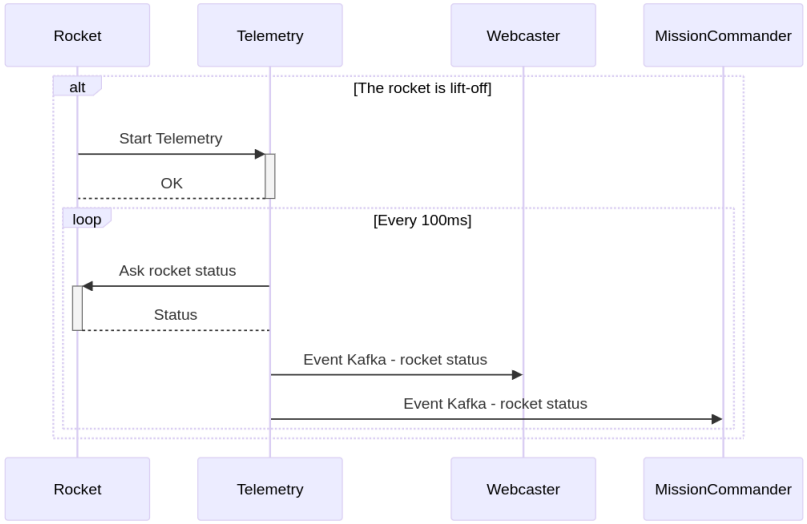

FIGURE 3.4.1 – Diagramme de séquence - Métrique de la fusée

Si la mission est bien lancée, le service Rocket envoie un message au Telemetry pour commencer l'enregistrement des métriques de la fusée. Rocket envoie à Telemetry un objet RocketStatusDTO avec la hauteur, l'état, l'inclinaison de la fusée et d'autres informations. On envoie bien évidemment des informations sur le niveau de carburant de chaque étage de la fusée. Chaque statut envoyé est persisté automatiquement dans la base de données des métriques de la fusée. Ensuite, Telemetry devient producteur d'event et en envoie un contenant le status de rocket. Ansi, Webcaster et MissionCommander consomme cette information. Il y a ici un pattern CQRS, où l'écriture se fait dans Telemetry, et où la lecture se fait avec MissionCommander et Webcaster.

3.5 Explosion de la fusée

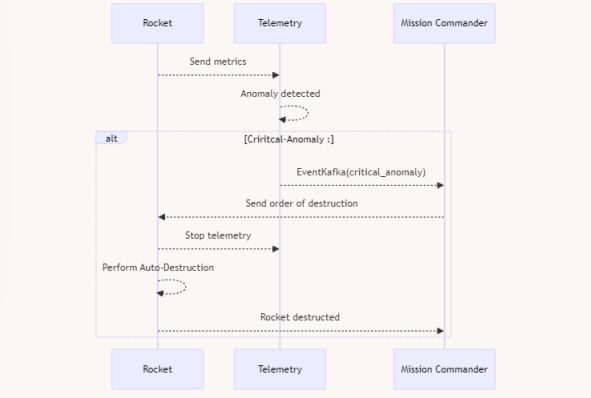

FIGURE 3.5.1 – Diagramme de séquence - Explosion de la fusée

Tant que la fusée n'a pas atteint le bon Orbite pour déposer le payload, le Telemetry interroge le service Rocket par rapport au statut de la fusée. On vérifie à chaque fois si l'inclinaison par rapport au plan de la Terre est bonne. Si on détecte qu'elle est supérieur à la valeur 20 degrés, on la considère comme anomalie donc on l'envoie au Mission Commander comme un évènement Kafka . Ce dernier va envoyer directement un ordre de destruction au service Rocket. La Rocket exécute l'ordre après avoir arrêté l'envoie des métriques au service Telemetry.

3.6 Max Q

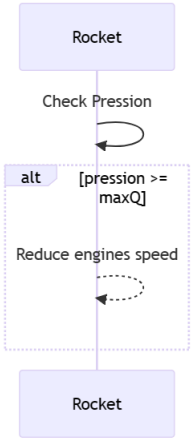

FIGURE 3.6.1 – Diagramme de séquence - Max Q

Le service Rocket effectue une vérification toutes les secondes pendant son cycle de vie pour s'assurer que sa vitesse actuelle ne dépasse pas la limite de MaxQ. Si la vitesse de la fusée commence à se rapprocher de la limite de MaxQ, elle active alors ses freins.

3.7 Largage du stage I et II

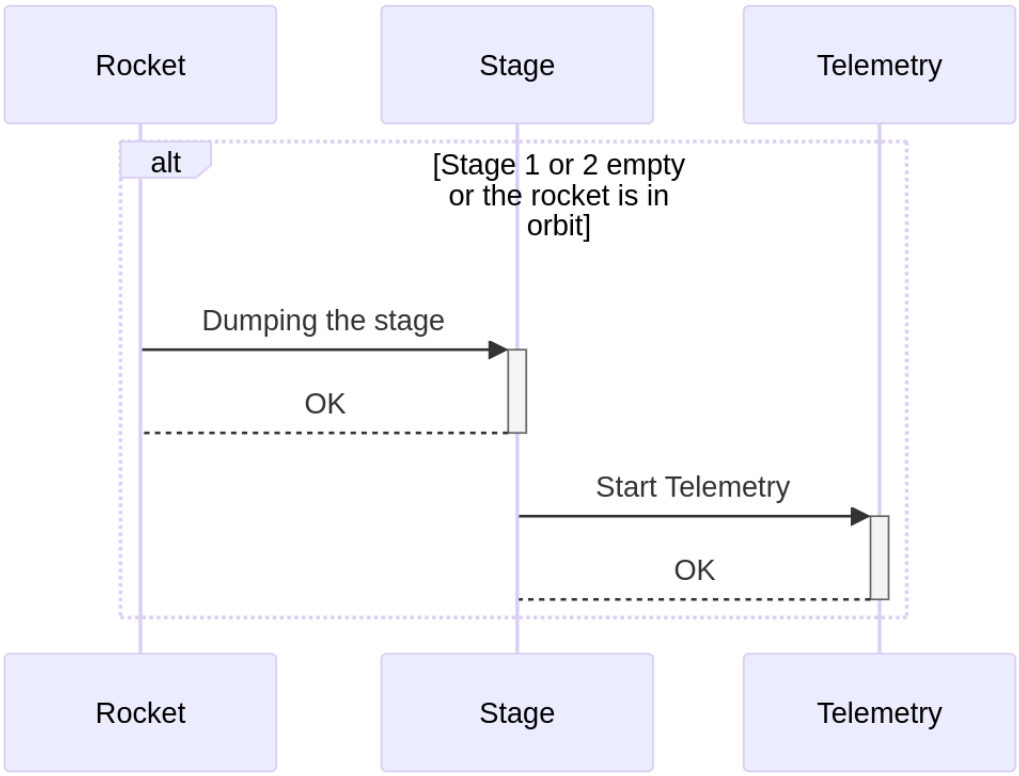

FIGURE 3.7.1 – Diagramme de séquence - Largage du stage I

Une fois que le stage 1 n'a plus de carburant, Rocket le détache. Une fois que stage 1 chute vers la terre, il demande à la Telemetry de démarrer la prise de métriques.

Idem pour le stage II, s'il a plus de carburant ou si la mission est fini, la fusée largue le stage. Le stage 2 fait partie des US que nous avons créé.

3.8 Métriques du stage I et II

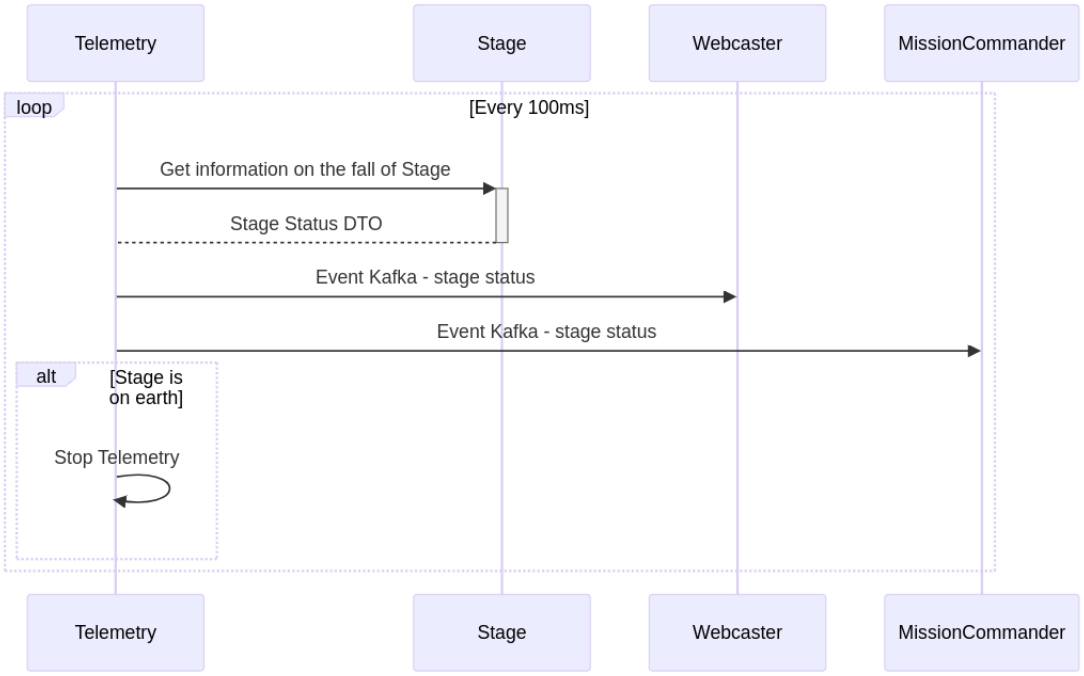

FIGURE 3.8.1 – Diagramme de séquence - Métriques du stage I

Telemetry va interroger toutes les 100ms le Stage 1 ou 2 lors de sa chute pour connaître sa position ainsi que l'état du parachute(ouvert/fermé). Une fois qu'il sera sur terre, Telemetry arrêtera sa demande de métrique.

Telemetry enregistre les datas provenant des Stages dans sa DB.

Telemetry Envoie ensuite l'event contenant les informations de la chute du stage. Webcaster et MissionCommander consomme ces informations.

L'opération de largage et de récupération des métriques est identique pour le Stage 2.

3.9 Déposer le payload

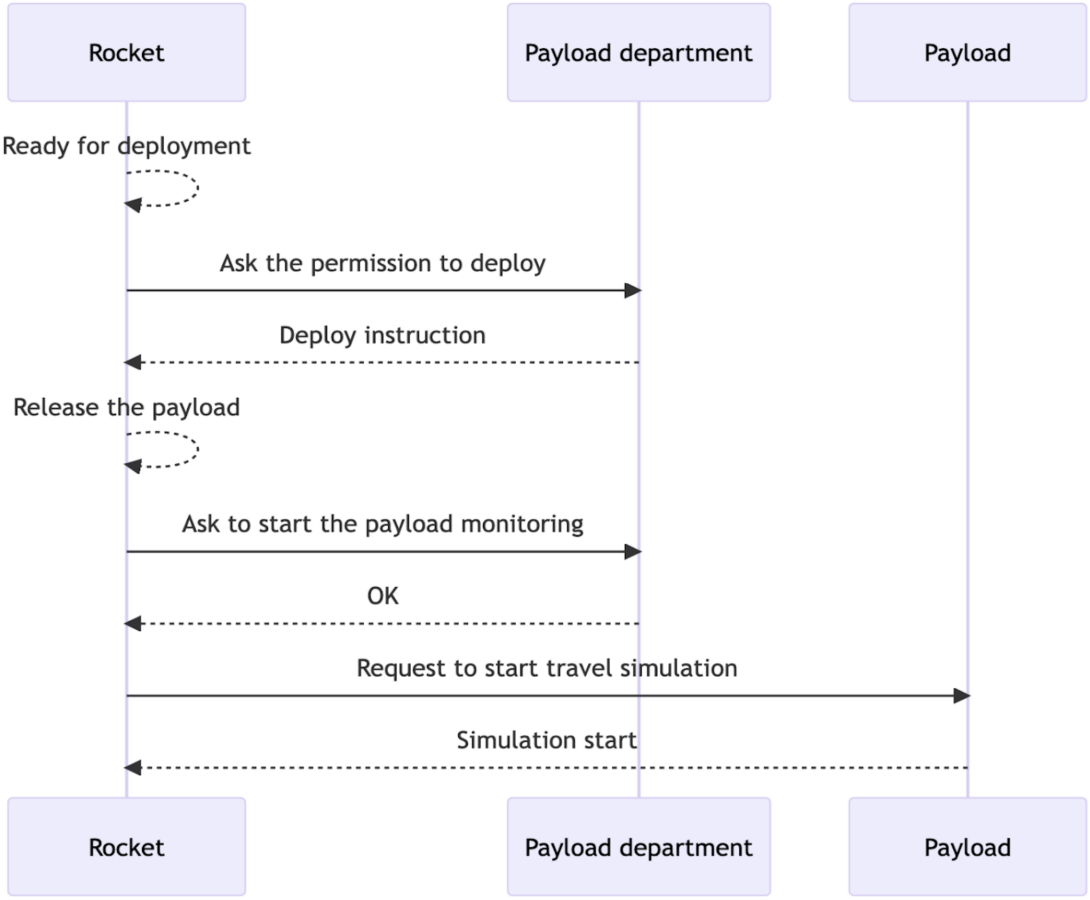

FIGURE 3.9.1 – Diagramme de séquence - Déposer le payload

Une fois arrivé à la bonne altitude pour le déploiement du payload, la rocket demande l'autorisation de larguer le payload. Une fois le payload lancer, il demande au département du payload de suivre le déplacement du payload et au payload de commencer la simulation de son déplacement

3.10 Suivi de la telemetry (payload)

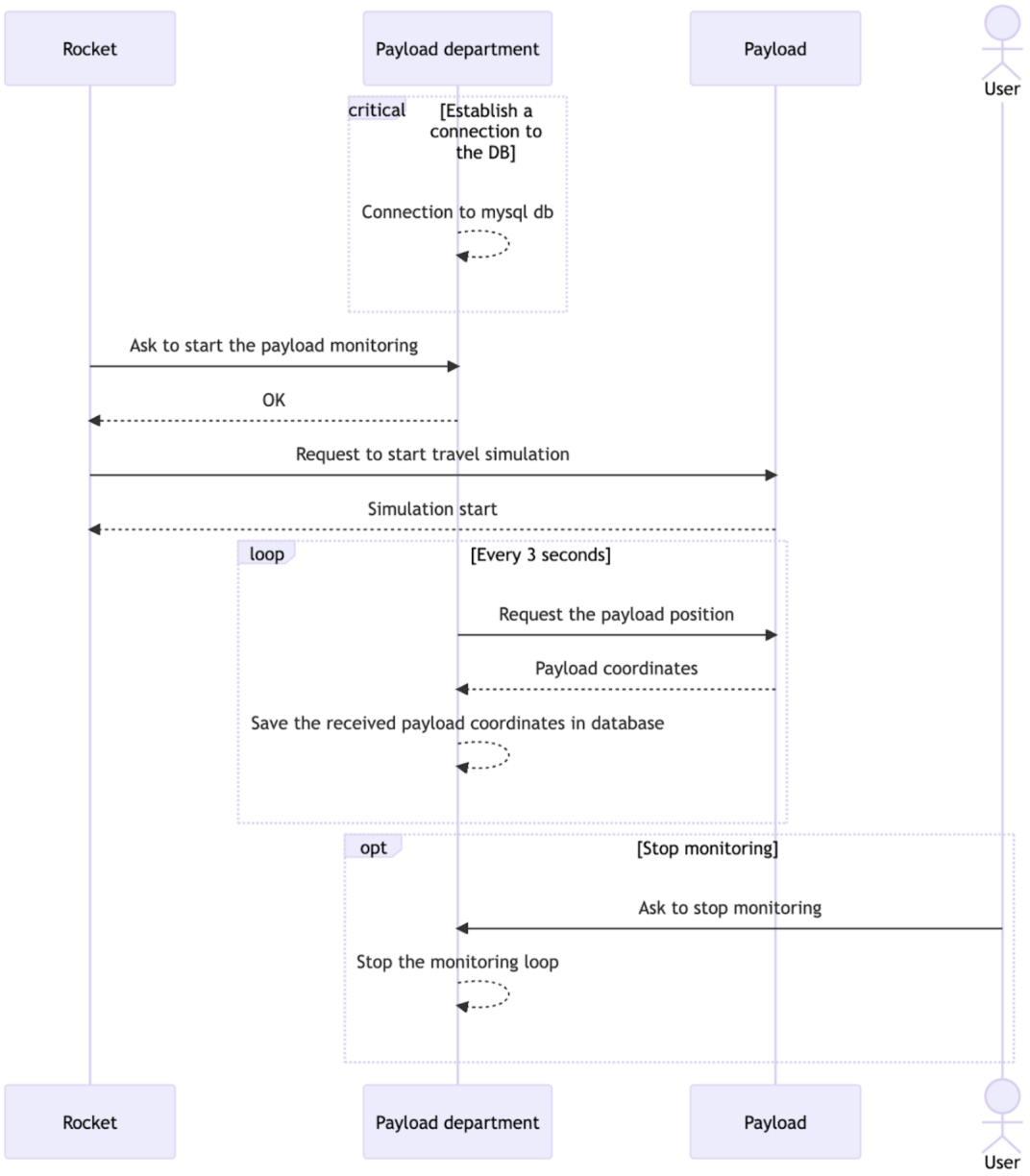

FIGURE 3.10.1 – Diagramme de séquence - Déposer le payload

Après le démarrage de la simulation et de son suivi, le service payload departement demande au payload sa position chaque 3 secondes, et ce, jusqu'à ce qu'une commande directe (externe) lui demande d'arrêter. Les données de position du payload sont sauvegardées dans une base de données et émises sur le topic de monitoring dans kafka pour que mission commander et les autres services puisse recevoir les données de télémétrie du payload.

3.11 Auto-déstruction de la Rocket

Le service Rocket vérifie à l'aide de son simulateur la valeur de l'inclinaison, dès qu'il détecte qu'il y a une anomalie , il exécute l'auto-destruction. Ensuite, il informe le mission Commander.

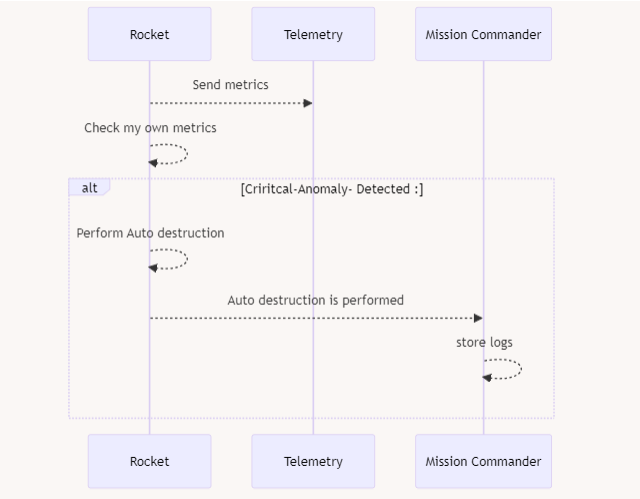

FIGURE 3.11.1 – Diagramme de séquence - Auto déstruction de la Rocket

3.12 Détection et envoie des anomalies :

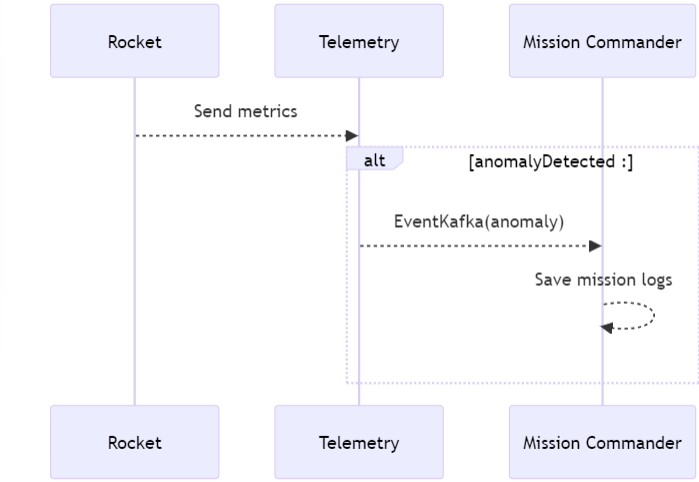

FIGURE 3.12.1 – Diagramme de séquence - Détection des anomalies de la fusée

Ce scénario est simple, si la Telemetry détecte une anomalie dans les métriques de la fusée, il crée un évènement Kafka qui sera envoyé dans le bus et consommé par Mission Commander. Ce dernier va se servir de cet évènement pour enregistrer les logs de la mission.

3.13 Mesure des éruptions solaires

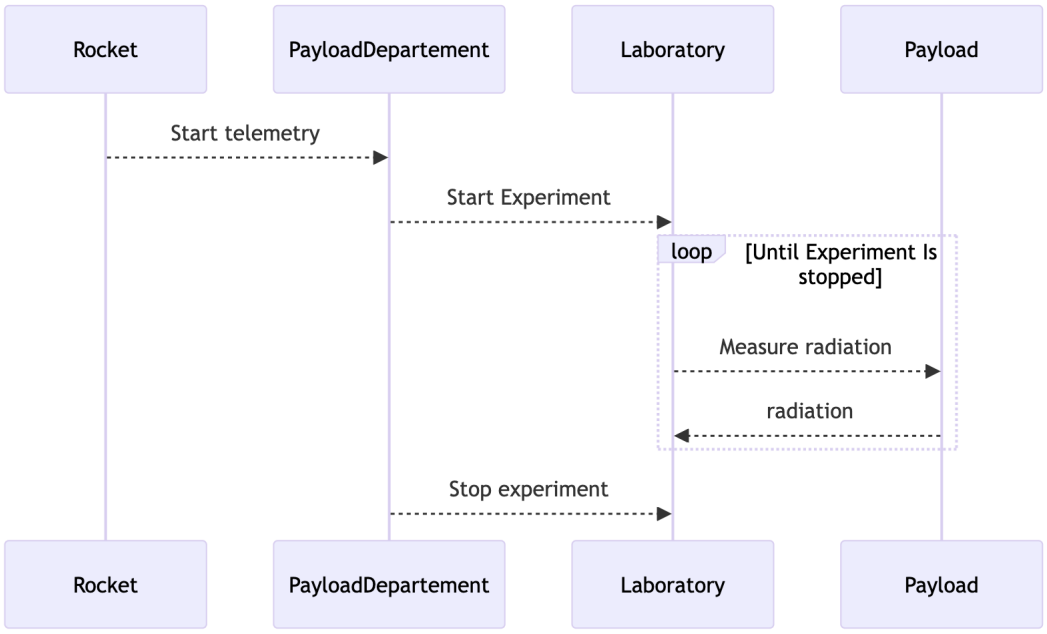

FIGURE 3.13.1 – Diagramme de séquence - Mesure des radiations solaires

Lorsque le payload est en place, la fusée signale au département Payload qu'il peut démarrer la télémétrie avec ce dernier. Simultanément, elle informe le laboratoire qu'il peut débuter la mesure des radiations solaires. En conséquence, des requêtes sont envoyées au satellite pour mesurer les radiations et renvoyer les résultats. Ces données sont ensuite analysées par le laboratoire.

4 Interprétation du sujet et analyse de l'existant

4.1 Interprétation du sujet

Richard (le Mission Commander), avant chaque mission va demander à Tory (du département de la météo) si la météo est valide pour un lancement, ainsi qu'à Elon (qui s'occupe de l'état de la fusée d'avant décollage), si la fusée est en état de pouvoir décoller.

Si et seulement si Richard reçoit le feu vert de ces deux collaborateurs (Tory et Elon), il informe Elon qui peut entamer la procédure de mise à feu. Dès que Elon reçoit cette ordre, il démarre la procédure de mise à feux du launcher, il prévient Jett (du département de la telemetry de la fusée) de commencer la prise de métriques de la fusée jusqu'au largage du payload, et la fusée décolle.

La fusée, lors de son voyage, est soumise à plusieurs stress. D'une part, des soucis divers et variés peuvent diminuer l'état de santé de cette fusée. (Modélisé par des points de vie dans notre simulateur). Mais globalement, il n'y a jamais vraiment de problème à ce niveau. Par contre, si son inclinaison devient inquiétante, voire problématique, le département de la telemetry (là où travaille Jeff) envoie automatiquement une alerte à Richard (Mission commandée) pour lui informer de la situation. Richard prend alors la décision de détruire ou non la fusée.

La fusée, lorsqu'elle atteint Qmax, ralentit sa vitesse pour ne pas subir trop de dégât.

Le stage 1 de la fusée, lorsqu'il n'a plus de carburant, est lâché pour qu'il rejoigne la terre. Lorsque l'ordre d'abandon est reçu, la stage 1 est en chute libre et il demande à la télémétrie de commencer sa prise de métrique. De la sorte, Peter (Chief executive) récupéra son stage et nous aurons toutes les métriques de l'atterrissage du stage.

La fusée, une fois en orbite à la bonne hauteur et dans la bonne trajectoire, demande à Gwynne (Chief Payload) si le largage du payload est maintenu. Si Gwynne donne le feu vert, la fusée demande au département du payload de démarrer la prise de métrique. Ensuite la fusé lâche le payload vers sa mission et demande à la télémétrie(Jeff) d'arrêter sa prise de métrique de la fusée .

Le département du payload va collecter des métriques sur le payload jusqu'à ce qu'il atteigne son objectif.

4.2 Analyse de l'existant

4.2.1 Les métriques

Nous avons considéré que ce sont les départements concernés au sol qui font les requêtes vers les cibles. C'est Telemetry qui fait des requêtes à Rocket et au Stage pour avoir des informations et Payload département qui fait des requêtes au Payload. Nous le justifions en exposant la volonté de diminuer la charge des systèmes embarqués. Moins ils sollicitent de puissance de calcul, moins ils consomment d'énergie et moins ils consomment de mémoire. Donc pas de cron qui tourne pour envoyer des informations à partir des systèmes embarqués. Nous avons estimé que cette vision était une bonne approximation du monde réel des entreprises du spatial.

4.2.2 Les limites

Les systèmes embarqués (Rocket et Payload) envoie une requête à Telemetry et à Payload department pour démarrer le cron qui à pour charge de récupérer des informations. Si cette requête est perdue, il n'y a pas de rappel et la Telemetry ou Payload département ne démarre pas leur cron.

Globalement, il n'y a pas de "retry" si une requête est perdue. C'est une dette technique.

Nous atteignons une limite en ce qui concerne la gestion des logs. Avec un grand nombre de services, la quantité de logs générée devient considérable. La multitude d'User Stories rend difficile la recherche d'une méthode optimale pour présenter la séquence d'événements de nos services.

Pour aller plus loin, il est impératif de mettre en place des services capables de consommer ces logs. L'intégration de technologies telles qu'OpenTelemetry pour surveiller nos services et logs s'avère essentielle. De plus, l'intégration de Prometheus et Grafana serait pertinente pour créer des tableaux de bord et réaliser des analyses approfondies de nos services

4.2.3 Explications de nos simulateurs

— **Payload** : Le payload intègre un simulateur basé sur le temps. On lui attribue une vélocité de 7.66 km/s. Lorsqu'il reçoit une requête lui demandant sa position actuelle, il fait une simulation de sa position en se basant sur le temps écoulé depuis sont lancement et d'autres données constantes.

— **Rocket** : Le simulateur de la Rocket intègre plusieurs capteurs (fuel, health, height, incline, speed). A chaque tour (a chaque appel de la telemetry) on déroule l'algo qui donne "vie" à la rocket. Il y a de l'aléatoire concernant l'inclinaison et pour de potentiels petits problèmes qui pourraient affecter la "vie" de la fusée (comme la collision avec de petits obstacles par exemple). En fonction de la vitesse, la fusée monte plus ou moins vite et une condition contrôle le non dépassement de Qmax (en réduisant la vitesse si nécessaire).

5 Explications des logs et Scénarios exécutées :

5.1 Affichage des logs :

Le script "run.sh" va exécuter six scénarios qui vont parcourir tous les User stories que nous avons implémenté. On a décidé d'ajouter des couleurs pour distinguer les actions clés. Voici chaque couleur et sa correspondance :

1) Jaune : Les évènements de Rocket.

2) Violet : Pour signaler que Mission Commander a reçu une anomalie non critique.

3) Rouge : Les évènements des Stages.

4) Marron : les informations du Payload.

Les scénarios qui vont s'exécuter sont les suivants :

- Exécution d'une mission qui va réussir.
- Récupération des positions du Payload en fin de mission.
- Mission Commander va consulter les logs de la mission.
- Lancement d'une mission qui échoue après une destruction manuelle (Après ordre du Mission Commander).
- Lancement d'une mission qui échoue après une auto-destruction.
- Mission Commander consulte encore les logs de la mission et constate que l'évènement "auto destruction" .
- Rediffusion des évènements de la mission dans Web-Caster.

Pour passer entre les différents scénarios, on doit appuyer sur le bouton **entrer**.

5.2 Scénario 1

Ce scénario va lancer une mission normale du lancement de la fusée jusqu'au dépôt du payload.

Ce scénario va référencer les diagrammes suivants :

- Prise des statuts pour décollage : 3.1
- Décollage de la fusée : 3.2
- MAxQ : 3.6
- Envoie des métriques de la fusée : 3.4
- Largage du stage : 3.7
- Dépôt du payload : 3.9
- Envoie des métriques du stage 1 : 3.8
- Suivi des métriques de payload : 3.10
- Enregistrement des logs : 3.11
-

5.3 Scénario 2

La visualisation des logs de la mission correspond au diagramme 3.11

5.4 Scénario 3

Dans ce scénario, nous avons lancer le même qu'avant juste, il y aura la détection d'une anomalie critique qui va causer la destruction de la fusée. Le Mission Commander va donner l'ordre de destruction. La destruction suit le diagramme 3.5 .

5.5 Scénario 4

Comme le scénario 3 sauf que la destruction sera automatique quand la fusée va détecter une anomalie, ensuite elle va exécuter l'auto-destruction suivant le diagramme : 3.12

5.6 Scénario 5

Dans ce scénario nous allons récupérer tous les logs après la destructions de la fusée.

5.7 Scénario 6

Dans ce scénario on va venir récupérer tous les évènements reçu par le webcaster.