POLYTECH
NICE-SOPHIA

Université Nice Sophia Antipolis

Membre de UNIVERSITÉ CÔTE D'AZUR

Blue Origin X

Team E

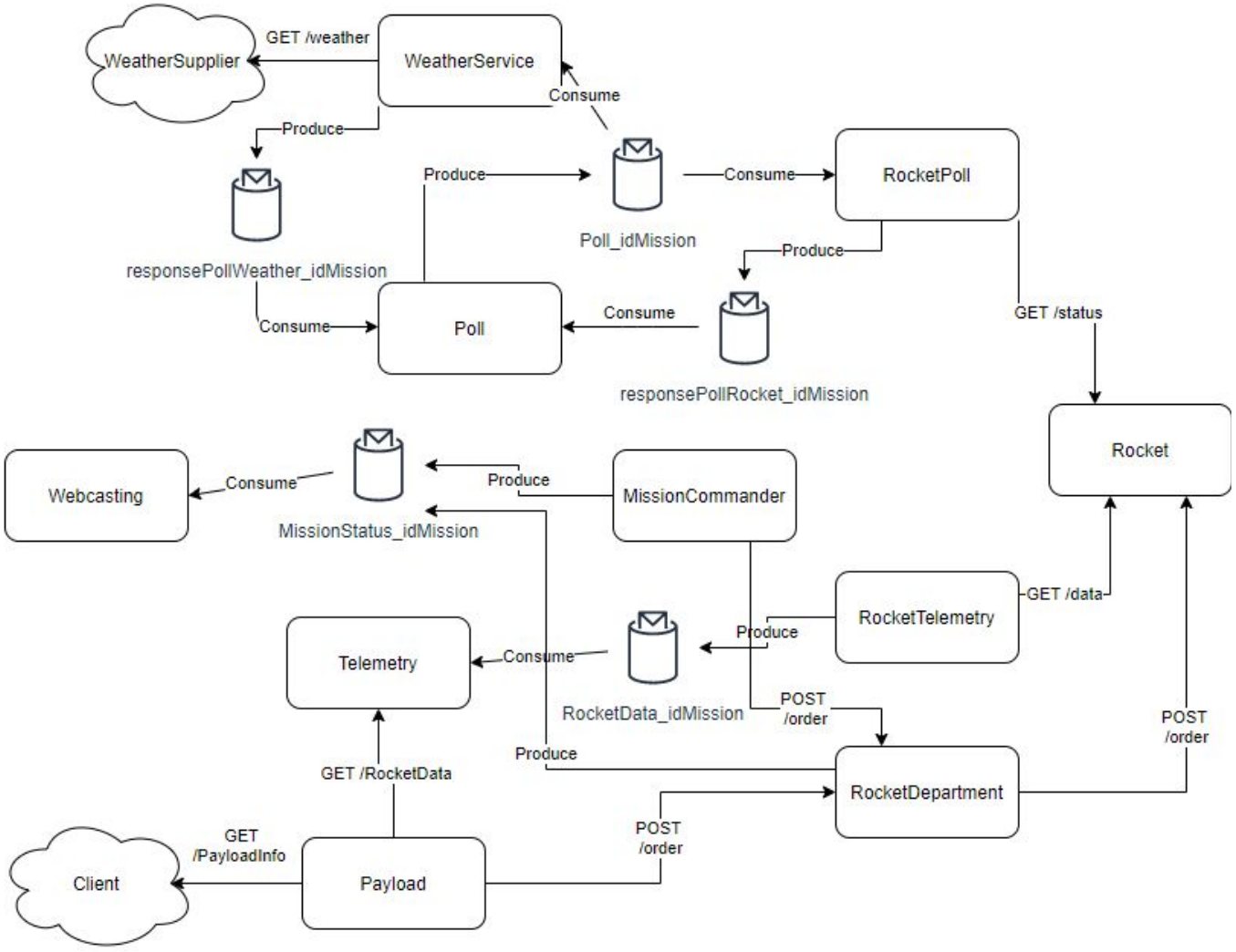
DELABY Maël
MAZOUZ Othmane
SIMON Fabrice
VAILLANT--BEUCHOT Maël

Final Delivery Report

Année universitaire : 2020 - 2021

Table des matières

Architecture Actuelle



Liste des services

Services	Ports	Routes	Paramètres	Retour de route	Dépendances
Rocket	4001	/status GET		“GO”	
		/order POST	“LAUNCH”, “TRAJCHANGE”, “FAIL”, “CHANGEMAXQ”		
		/success GET			
		/data GET		RocketData	
Chief Rocket Department	4002	/order POST	“LAUNCH”, “TRAJCHANGE”, “FAIL”, “CHANGEMAXQ”		4001/order
		/success GET			
		/rocketInfo POST	“MAXQ”, “SPLIT”		Topic MissionStatus
Poll	4003	/poll GET		“GO”, “NO GO”	4012/startKafka 4004/startKafka Topic Poll Topic ResponsePollRocket Topic ResponsePollWeather
Weather Department	4004	/startKafka GET			4005/status Topic Poll Topic ResponsePollWeather
Weather Supplier	4005	/status GET		"Clear", "Raining", "Windy", "Thunderstorm"	
Mission	4006	/rocketLaunch GET		“NO GO”	4003/poll 4002/order Topic MissionsStatus
		/payloadStatus			Topic MissionsStatus
Telemetry	4007	/start GET			Topic RocketData
		/stop GET			
		/reset			Topic RocketData
		/rocketData GET		RocketData	
Payload Department	4008	/sendPayloadInformation GET		“Payload at place”	4002/order 4006/payloadStatus 4007/rocketData 4009/getPayloadInformation
Client	4009	/getPayloadInformation GET		“Traj” : “Orbital "FutureSpeed":100, "FutureAngle":90	
Webcasting	4010	/start		"Starting reading mission status"	Topic MissionStatus

		/stop			
Rocket Telemetry	4011	/start			4001/data Topic RocketData
		/stop			
Rocket Poll	4012	/startKafka		“GO”, “NO GO” (in the topic)	4001/status Topic Poll Topic ResponsePollRocket

Modifications dans l'architecture au cours du temps

Lors de notre première approche, nous avons créé une classe pour chaque UserStory. Dans une problématique de responsabilités, nous avons ensuite séparé le département de Richard en deux services : *Poll* qui s'occupe de faire le sondage et *RichardInterface* qui communique simplement avec *Poll* et le *ChiefRocketDepartment*.
Dans les semaines qui ont suivi, nous avons progressé vers une deuxième approche en continuant à séparer des services en plusieurs services pour se détacher de plus en plus du côté "un service est égal à une User Story". Nous avons par exemple, séparé notre service *Rocket* en services *Rocket*, *RocketPoll* et *RocketTelemetry*. Cela permet de découper un maximum les responsabilités de chaque service. Ensuite, dans une troisième approche, nous avons essayé d'implémenter une relation avec Kafka dans tous les services dans lesquels c'était approprié, par exemple, le sondage est effectué entre le service *Poll*, le service WeatherDepartment et les différents services concernés via des messages transitant dans Kafka.
Nous avons également essayé de corriger un maximum de points sensibles de notre architecture. Par exemple, le service *Telemetry* avait une trop grande responsabilité dans la gestion de la Rocket, cette dernière devait attendre des réponses de *Telemetry* pour se séparer. Ce sont des actions qui sont à présent effectuées en interne dans chaque service concerné. La responsabilité de *Telemetry* est maintenant moins importante et ne représente plus un *SPOF*.

Etat de l'architecture finale

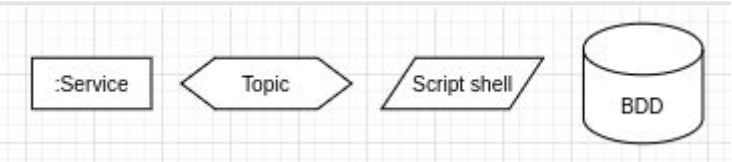
Dans notre architecture finale, nous avons des contrats faibles au niveau de tâches non critiques comme le sondage avant le lancement, la récupération des données de la fusée ou encore la récupération des statuts de la mission à but informel (les décisions prises par rapport à l'avancement de la mission ne le sont pas en fonction de ces contrats faibles). Pour les décisions prises par rapport au statut de la fusée, nous ne nous basons pas sur les contrats faibles non plus, donc pas via les événements Kafka. Nous prenons ces décisions à partir de contrats plus forts via des requêtes entre services.

Dans les éléments qui nous restent à faire :

- Rendre la fusée en tant que classe et non en tant qu'un ensemble de variables
- Séparer encore le service Rocket en d'autre services pour gérer les instances de la classe fusée (RocketCreate, RocketPopulate, RocketRemove)
- Séparation de Payload en 2 services: un s'occupant de la télémétrie (savoir si la fusée a été split…) et l'autre de la gestion du payload (récupération des infos auprès du client, envois à la fusée, vérification si le payload est au bon endroit…)
- Nous avons implémenté qu'une partie des différents événements du lancement de la fusée mais aucun de l'atterrissage du booster. (User story 13)
- Nous n'avons pas implémenté de services permettant l'atterrissage du booster de la fusée

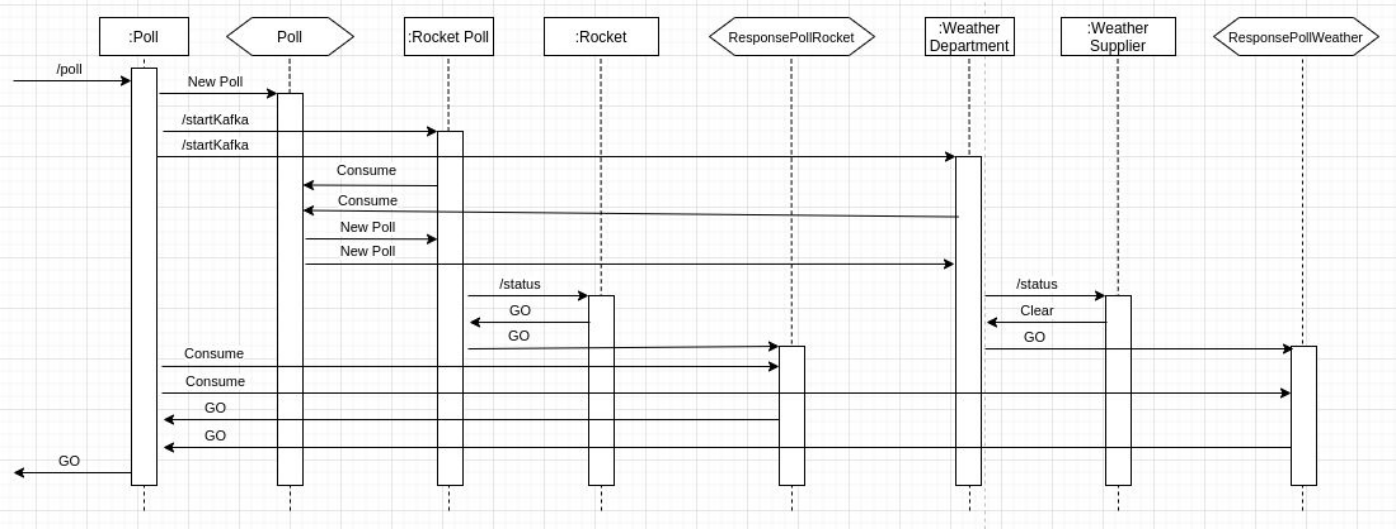
- Découper chief rocket department pour séparer le fait de transmettre des ordres à la fusée et le fait de publier le state de la fusée
- Créer un service qui s'occupe exclusivement des issues et qui peut décider selon les issues qui ont été déclenchées, de détruire la fusée en vol. Cela se fera *via* un topic spécifique auquel n'importe quel autre service pourrait envoyer des issues.
- Nous n'avons pas implémenté le lancement de plusieurs fusées. Cependant nous avons réfléchi à l'implémentation:
    - Chaque mission est associée à un ID unique.
    - Cet ID sera utilisé dans les bases de données (pour différencier les télémétries), et de nouveaux topics Kafka sont créés à partir de cet ID.

Diagrammes de séquence



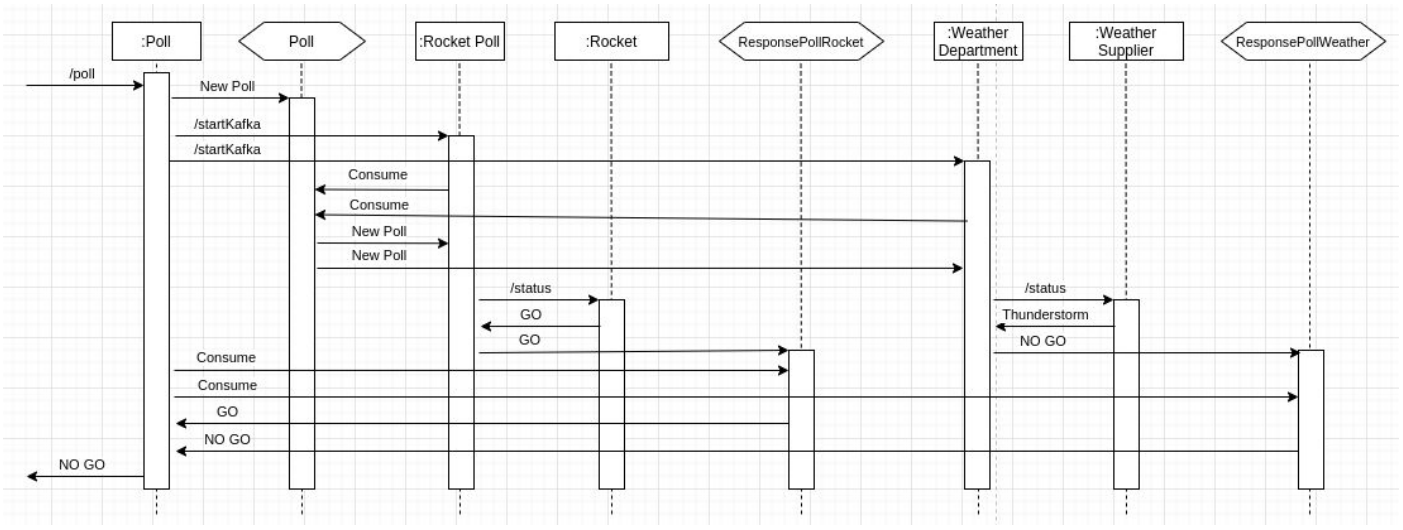
Représentation des éléments des diagrammes de séquence suivants

Poll avec un temps calme



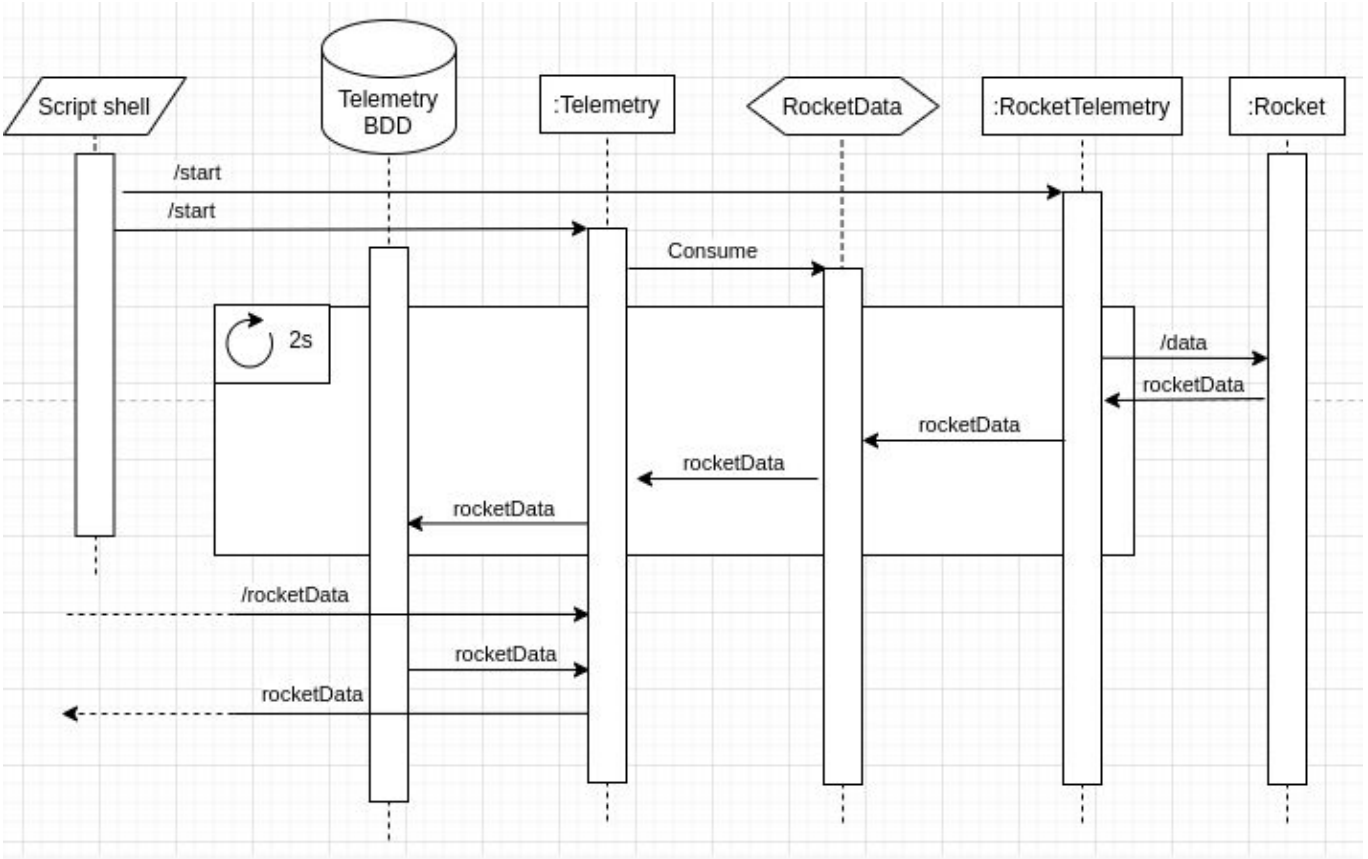
Ce diagramme représente les liaisons qui se font entre les différents services et topics lors de la phase de poll lorsque le temps est calme. Nous avons décidé d'utiliser Kafka dans cette partie pour que lorsqu'un poll est demandé tous les services devant y répondre puissent être au courant sans avoir à leur demander un par un.

Poll avec une tempête



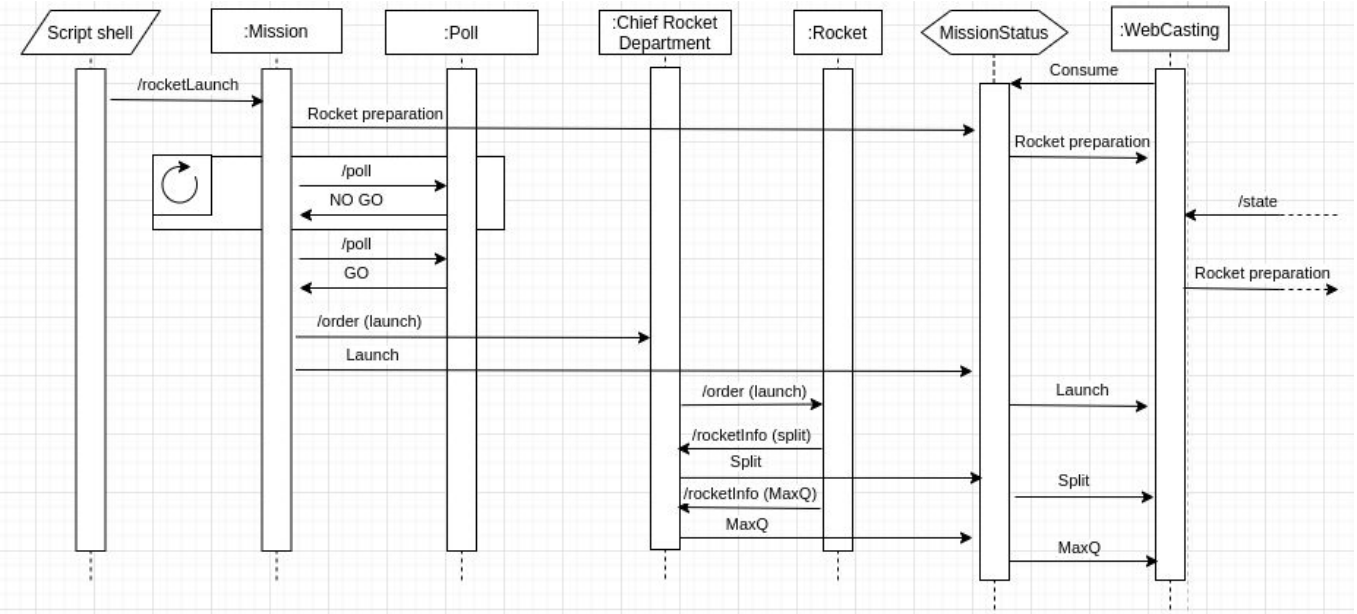
Ce diagramme représente les liaisons qui se font entre les différents services et topics lors de la phase de poll lorsqu'il y a une tempête.

Telemetry



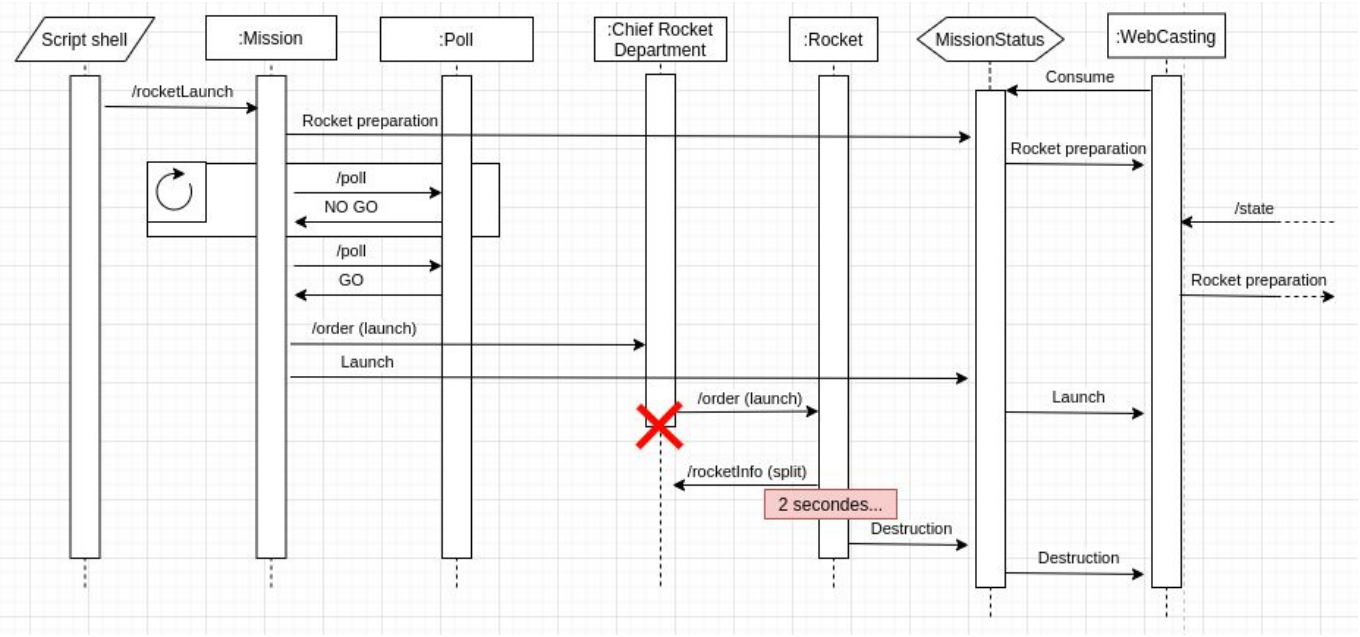
Ce diagramme représente les liaisons qui se font entre les différents services, topics et bases de données pour que les données de la fusée soient stockées dans la base de données de la telemetry, ainsi que par quel biais nous pouvons accéder aux données de la telemetry.

Mission status



Ce diagramme représente les liaisons qui se font entre les différents services et topics lors du lancement de fusée, nous pouvons aussi y voir comment webCasting obtient ses informations sur l'état de la mission : cela est effectué *via* un topic nommé MissionStatus, ce topic reçoit des informations de la part de plusieurs services tel que Mission et ChiefRocketDepartment, nous imaginons que par la suite d'autre services fourniraient ce topic.

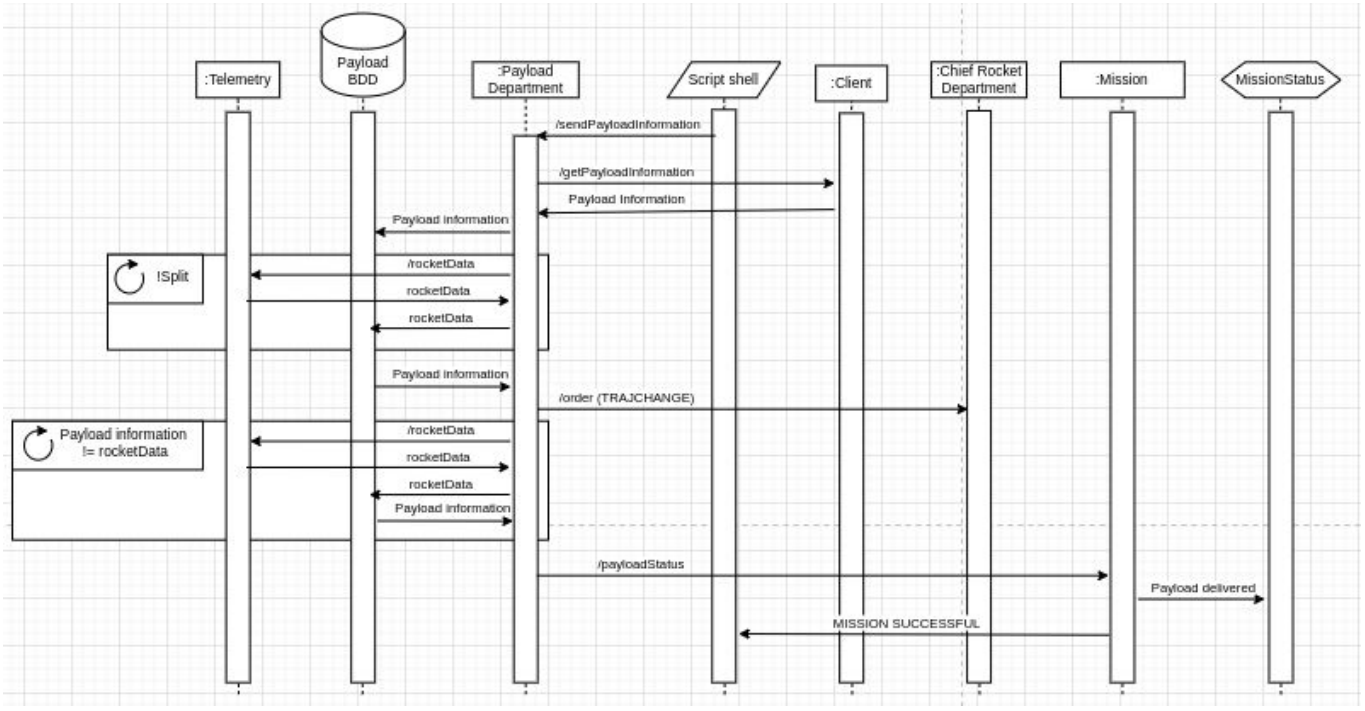
Mission fail



Ce diagramme représente les liaisons qui se font entre les différents services et topics lorsqu'il se passe le même scénario que le scénario 2, lors de ce scénario le ChiefRocketDepartment est victime d'une attaque zombie et n'a donc plus d'électricité malgré les systèmes électriques de secours, lorsque la rocket arrive donc à sa phase de

split elle envoie comme habituellement une requête au ChiefRocketDepartment qui ne lui réponds pas, au bout de deux seconde la rocket décide toute seule de mettre fin à ses jours et envoie en dernier message un “Destruction” au topic MissionStatus avant de se donner la mort car elle sait que maintenant sans maître, elle est devenue un danger pour la société.

Payload department



Ce diagramme représente les liaisons qui se font entre les différents services, topics et bases de données afin que le PayloadDepartment puisse effectuer au mieux sa mission : mettre en orbite son objet galactique. Le département commence par récupérer les informations utiles pour savoir comment positionner sa charge, puis il attend que la fusée se soit divisée, il ordonne alors le changement de trajectoire de la fusée, puis il attend qu’elle soit positionnée comme souhaitée pour annoncer à mission que ça tâche est réussie. Une fois toutes les tâches nécessaires au succès de la mission accomplie, la mission est une réussite !

Scénarios

Les deux scénarios couvrent toutes les séquences décrites dans les diagrammes précédents, sauf le diagramme sur l'échec de la mission pour le premier scénario et le diagramme sur le *payload department* pour le deuxième scénario.

Scénario 1 : Mission successful

Dans ce scénario, nous simulons le succès d'une mission. Les services ayant besoin de constamment récupérer ou exposer des informations sont lancés en premiers. Puis, une demande est faite pour effectuer un *poll*, si le temps le permet, l'ordre est donné de lancer le décollage de la fusée. Pendant son temps de vol, nous affichons les données reçues par le service *telemetry*, cela permet de visualiser en temps réel les différentes variables de la fusée. Les événements concernant la fusée sont gérées automatiquement par celle-ci, par exemple, lorsque le taux de carburant de la première partie de la fusée passe sous les 10%, cette dernière se sépare (la variable *split* passe alors à 1). Lorsque la charge utile est positionnée correctement, la fusée a réussit sa mission et met alors à disposition dans ses données le texte "MISSION SUCCESSFUL !". Les bases de données sont ensuite effacées.

Ce scénario couvre ces différentes User Stories :

1. As Tory (Launch Weather Officer), I need to check the weather status, so that I can be sure that the conditions are in a valid range for a safe operation of the rocket.

2. As Elon (Chief Rocket Department), I need to monitor the status of the rocket, so that I can be sure that the rocket is behaving correctly before launch.

3. As Richard (Mission Commander), I have to perform a Go/No Go poll to every monitoring department before giving the final go ahead with the launch, so that I can be sure that everything is nominal before launch.
The Go/No Go poll has to be done in the following order:

Weather Department (Tory)
Rocket Department (Elon)
Mission Commander (Richard)

4. As Elon (Chief Rocket Department), I have to send the launch order to the rocket after the GO from Richard, so that the rocket can launch into space and deliver the payload.

5. As Jeff (Telemetry Officer), I want to receive, store and consult the telemetry data of the rocket of the whole launch sequence (from before the launch, to the end of the mission), so that I can monitor that everything is working as intended, and that if anything goes wrong, I can find the root cause of the anomaly.

6. As Elon (Chief Rocket Department), I want to stage the rocket mid-flight (separate the rocket in 2 parts), so that the rocket remains as efficient as possible, by leaving behind the

first stage, now empty in fuel, and continuing with the second stage and the payload, full in fuel.

7. As Gwynne (Chief Payload Department), I want to deliver the payload (satellite/probe) in space on the right orbit or trajectory, so that the customer's desires have been successfully fulfilled by the mission.

11. As Gwynne (Chief Payload Department), I want to receive, store and consult the telemetry data of the payload, so that Blue Origin X can certify that the orbital parameters desired by the customer are ensured.
N.B.: Pour cette User Story, les informations de la télémétrie sont bien enregistrées mais sont ensuite supprimées pour permettre au deuxième scénario de se lancer.

12. As Elon (Chief Rocket Department), I want the rocket to go through Max Q harmlessly so that the total stress on the payload and the flight hardware stay in a safe level. In order to do so, the rocket engines must throttle down to reduce the load. Max Q is the atmospheric flight phase where the vehicle's flight reaches maximum dynamic pressure because of the air density and the speed of the rocket.

13. As Richard (Mission Commander), I want the launch procedure to follow the subsequent events in order to have a fine grain overview of the mission

15. As Marie (Webcaster), I want to be aware of the launch procedure events in real time so that I can tell on the web stream what is actually happening.

Scénario 2 : Mission aborted with destruction

Dans ce scénario, nous simulons l'échec d'une mission. Pour ce faire, nous reproduisons les mêmes manipulations qu'au scénario précédent mais à un moment nous arrêtons le service avec lequel la fusée communique pour vérifier que tout se passe correctement. Après un délai de deux secondes sans retour de la part de ce service, la fusée s'autodétruit et expose la donnée "MISSION FAILED !".

En plus de certaines User Stories énoncées précédemment, ce scénario couvre les suivantes :

8. As Richard (Mission Commander), I want to be able to issue an order for the destruction of the flight hardware in case of a severe anomaly, so that the rocket can't follow an uncontrolled trajectory and to prevent potential damage on the ground.

18. As Richard (Mission Commander), I want the mission to be aborted (flight hardware destruction if in-flight, etc) whenever an anomaly with a critical severity is detected in order to not endanger the surrounding area with and uncontrollable rocket without any human intervention.

Annexe

Lien vers les diagrammes de séquence :
https://drive.google.com/file/d/12_XxcBxftJGWgezN9ABdR5VH43KpXiSJ/view?usp=sharing