Service Oriented Architecture

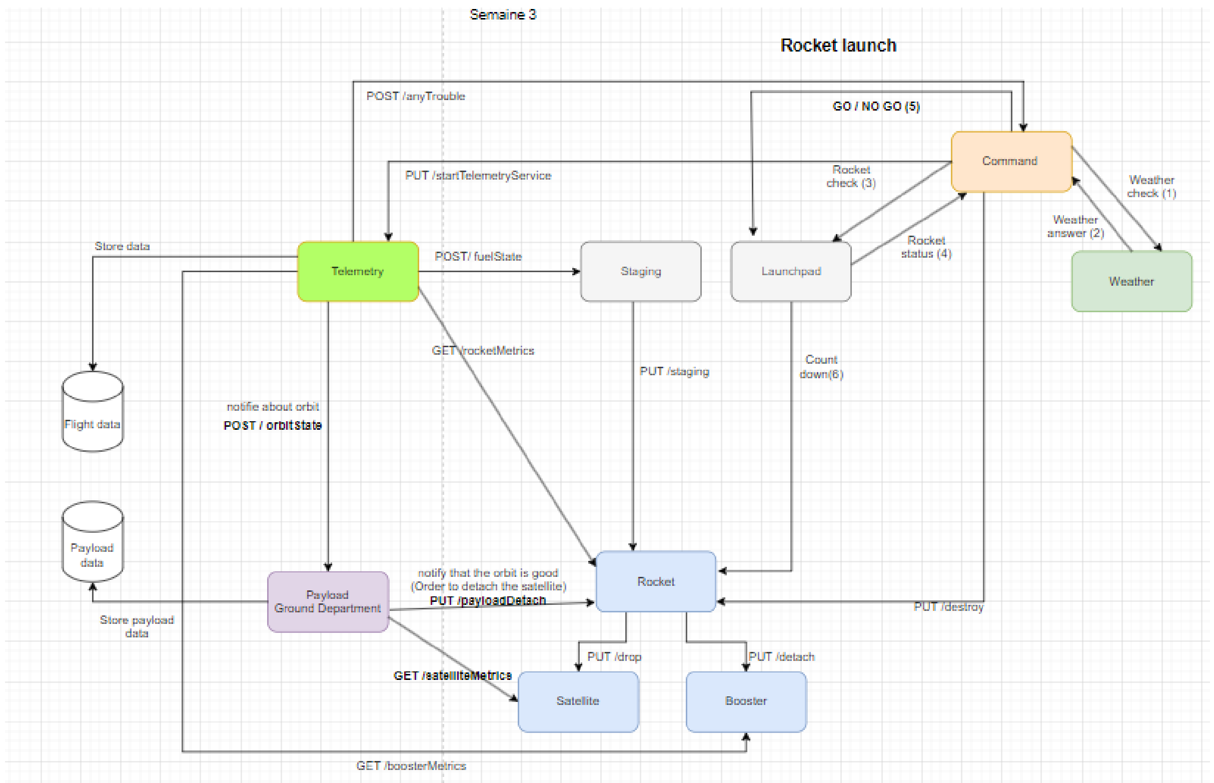
Week 39

Groupe member :

- Bassy Ayman
- Bonifay Tobias
- Melnyk Igor
- Schalkwijk Mathieu

Summary

Current architecture



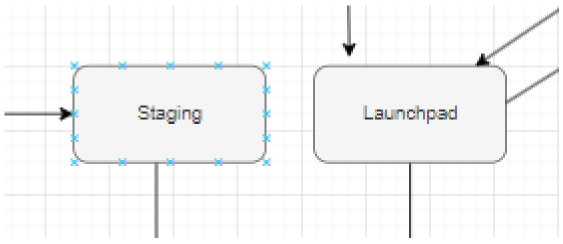
In this architecture we decided to have for each actor of the Marsy mission, a service that represents his own business logic. We decided to cut back heavily on the first versions of the services we had in order to separate the different, very specific responsibilities.

Our motivations were multiple, first of all in the manner of the chaos monkey, we imagined several scenarios where randomly such or such service no longer worked. This led to a certain amount of decoupage during the previous week, and other services were added with the new user stories, which led to the same approach for the decoupage of services we felt were too big.

The other motivation was mission completion, as having several separate departments meant that we could move forward with the mission with 5 active departments at the start and 6 active departments towards the end, for example, so that the other departments could delegate the rest of the mission to the other departments for completion.

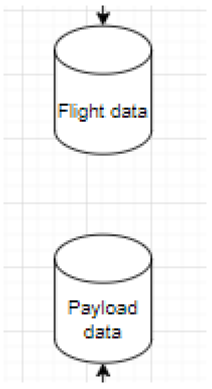
This has three advantages: firstly, as stated above, if a service that has already fulfilled its role no longer works, it doesn't hinder the other services; secondly, in order to add more specificity and new features linked to an increase in the system's complexity, the code will be added to the existing services and set up properly. Finally, in a future implementation of an event bus, having all these services properly isolated will allow us to delude more specific subscribers, instead of having a service subscribed to the retrieval of several uncorrelated data.

Detailed choice of architecture



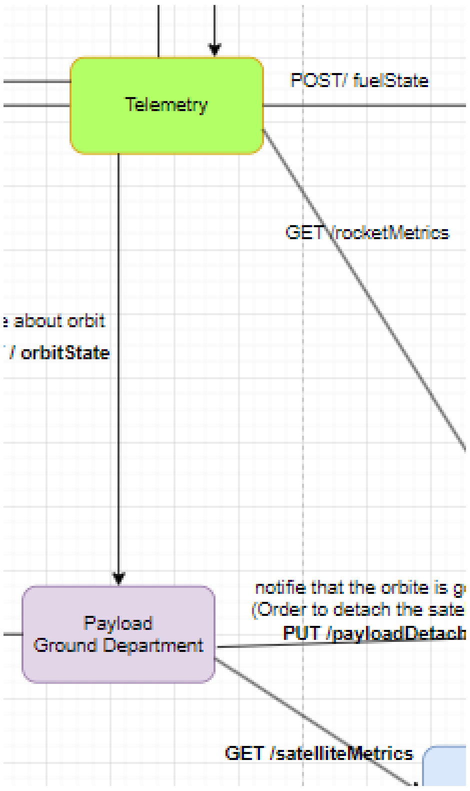
Decision to separate the basic launchpad into a launchpad and a staging service to avoid a single point of failure that would prevent the separation of the satellite from the rocket. This way, even if the launchpad fails, we will still have another service running to successfully carry out the launch of the payload in the Marsy mission context. Another thing is that we also wanted to not have the telemetry service having the staging responsibility as he already manage the rocket data.

.



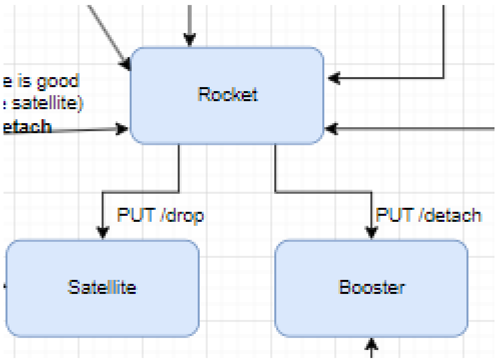
Two databases to avoid having too many interactions with a single database that multiple services must repeatedly call, especially in the case of significant scaling.

Additionally, assuming that the rocket encounters an issue during the course of the mission that could potentially impact the stored information, the data from the satellite will still be stored without any disruption



Decision to separate the responsibility for data retrieval between the Telemetry and Payload services is due to the fact that if one service goes down, we can continue to collect data to be placed in one of the two databases.

Furthermore, the data related to the rocket is distinct from that collected by a satellite. In the context of adding more complex features, we can isolate the satellite's case. The satellite data can also be sent to the client.



We have decided to represent each module of the rocket as a separate service so that it can represent new hardware modules from which we can retrieve data. Simulating the rocket as a whole after real detachments did not seem to be a viable alternative approach. Having a service for each module allows us to be as close to the physical representation as possible (cf. Conway's Law)

Services

Command service

Location: At the command center.
Role: Checks if the different services are going ok and gives order to launch. Sends an autodestruction signal to the rocket if a problem is reported by the telemetry service.

Weather service

Location: In the weather station next to the launch site.
Role: Gives the current weather (weather data is simulated).

Launchpad service

Location: At the launchpad on the launch site.
Role: Check if everything is ok on the launch site.

Rocket service

Location: In the rocket (the upper part).
Role: Consults the rocket's captors to obtain the metrics (obviously mocked), detaches the booster and detaches the satellite.

Telemetry service

Location: In the telemetry office at the command center.
Role: Periodically collects and stores data from the rocket and gives them to staging service and the payload ground service. Also collects and stores data from booster service. Sends a signal to the command service if there is a critical problem with the rocket.

Payload service

Location: At the command center.
Role: Analyzes data collected from telemetry service and gives the order to detach the satellite from the rocket at the right moment. Collects and stores data directly from the satellite.

Staging service

Location: At the command center.
Role: Receives rocket metrics from the telemetry service and gives order to stage the booster at the right moment.

Satellite service

Location: In the satellite (beside the customer's own system to which we don't have access).
Role: Detaches from the rocket and gives his metrics to the ground payload service.

Booster service

Location: In the booster (lower part of the rocket).
Role: Detaches from the rocket and gives his metrics to the telemetry service. Lands back on the launchpad.

About Scénario

Based Scénario Flow

We have one scenario demonstrating the features. It consists in the following steps:

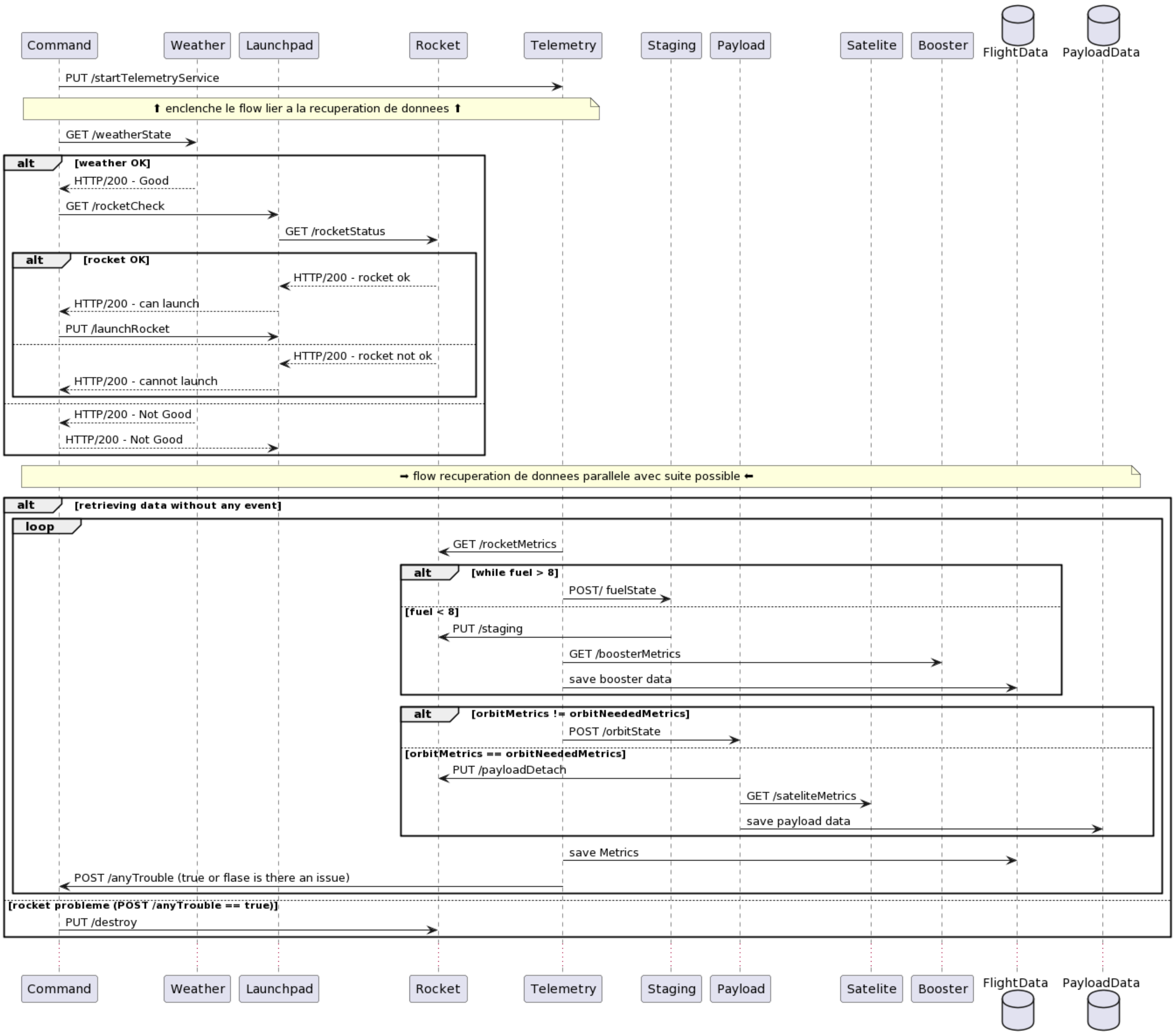
- Poll from the command center to check if everything is ok (weather and rocket)
- Rocket launch
- Booster stage from the rocket
- Satellite detachment
- Booster landing
- Simulated critical problem on the rocket (an external call directly on the rocket service)
- Order of command center to destroy the rocket and then auto-destruction (rocket-service auto-shutdown)

Possible Scenario

1) The first and basic one is when everything goes well, so both weather and rocket are ready to be launched, and when it's launched, the telemetry service keeps retrieving hardware data from the rocket and broadcasts them to both staging service and payload service. It first sends fuel data to the staging service and when the booster is detached, it's the payload service who receives the data about the orbit, so he can launch the payload.
   When the booster is detached the telemetry service starts retrieving data from it and saves it in the database. The booster land after a while.

2) The second scenario is when the weather is not good, and so the rocket cannot GO
3) The third on is if the rocket have a hardware issue with conduct to a abort of the mission.
4) The last scenario occur when the rocket is launched and a hardware issue appear, so the command center have to destroy it

Sequence Diagram



(diagram made with PlantUML, code in the annexe)

.

Limit of this implementation

Mock sensors

For obvious reasons, we had to mock the sensors located in the rocket. It is the rocket service that manages to do this so it doesn't accurately represent reality and it makes the rocket service stateful.

Some Stateful Service

The Rocket service and Telemetry service are actually statefull, as they need to keep in memory the actual state of the rocket which can be not deployed yet or separated in two parts, meaning that the booster has been separated, and the last state being the one where the satellite is sended in orbite.
This statefull capability permit us to know when to stop sending data to some or other services so we limit all HTTP calls during the marsy mission

Limit Case single point of failure

We've identified a single point of failure that's difficult to cut or change, because it's an important part of our service, and it's the command service. In fact, it comes into play in the second part of the mission, after the rocket has been launched. In the event of a problem, he's the one who blows up the rocket. The problem here is that if the service command fails, the rocket cannot explode. But in the current implementation, it's the command service that's best placed to take this action. As the launchpad remains linked to the rocket's departure, we don't know whether the complexity linked to the rocket's departure will subsequently increase or not. So the responsibility for supervising the mission in general, and the major decisions linked to it, lies with the command service.

What's missing

- We didn't implement the user story number 12. In fact, we were short of time and chose to invest it in ./prepare.sh, which was causing us a few problems. We identified the user story as being a new implementation of the rocket's sensors, which is currently mocked up in the poc demonstration, and therefore also didn't seem relevant to us for the week 39 rendering.
- The booster is actually landing, but we miss some log about altitude and fuel metric that demonstrate the landing.

Annexe

PlantUml code (https://www.planttext.com/)

```
@startuml
participant Command
participant Weather
participant Launchpad
participant Rocket
participant Telemetry
participant Staging
participant Payload
participant Satelite
participant Booster

database FlightData
database PayloadData

Command->Telemetry: PUT /startTelemetryService
note over Command, Telemetry: ⬆️ enclenche le flow lier a la recuperation de donnees ⬆️
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
    Telemetry->Rocket: GET /rocketMetrics
    alt while fuel > 8
      Telemetry-> Staging: POST/ fuelState
    else fuel < 8
      Staging->Rocket: PUT /staging
      Telemetry->Booster: GET /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)
  Command -> Rocket: PUT /destroy
end
...

@enduml
```