Semestral project (40+5 points)

1. Codebase – one codebase tracked in revision control, many deploys
    - separate repositories for each microservice (1)
    - continuous integration to build, test and create the artefact (3)
    - implement some tests and test each service separately (unit tests, integration tests) (5)
2. Dependencies – explicitly declare and isolate dependencies
    - preferably Maven project with pom.xml
    - eventually gradle project or other
3. Config
    - configuration of services provided via environmental properties (1)
    - eventually as configuration as code (bonus: *0.5*)
4. Backing services – treat backing services as attached resources
    - backing services like database and similar will be deployed as containers too (1)
5. Build, release, run – strictly separate build and run stages
    - CI & docker
    - eventually upload your images to docker hub (bonus: *1*)
6. Processes – execute the app as one or more stateless processes (1)
7. Port binding – export services via port binding (1)
8. Disposability – maximize robustness with fast startup and graceful shutdown
    - ability to stop/restart service without catastrophic failure for the rest (2)
9. Dev/prod parity – keep development, staging, and production as similar as possible
    - repository for integration testing and system demonstration (2)
    - services will be deployed as containers
10. Logs – treat logs as event streams
    - log into standard output (1)
    - eventually collect logs in Elastic (bonus: *0.5*)
11. Communication
    - REST API defined using Open API standard (Swagger) (2)
    - auto-generated in each service (1)
    - clear URLs (2)
    - clean usage of HTTP statuses (2)
    - eventually message based asynchronous communication via queue (bonus: *1*)
12. Transparency – the client should never know the exact location of a service.
    - service discovery (2)
    - eventually client side load balancing (bonus: *0.5*) or workload balancing (bonus: *0.5*)
13. Health monitoring – a microservice should communicate its health
    - Actuators (1)
    - eventually Elastic APM (bonus: *1*)
14. Design patterns – use the appropriate patterns (2)
15. Scope – use domain driven design or similar to design the microservices (5)
16. Documentation – visually communicate architecture of your system (5)
    - https://c4model.com/ (https://github.com/RicardoNiepel/C4-PlantUML)