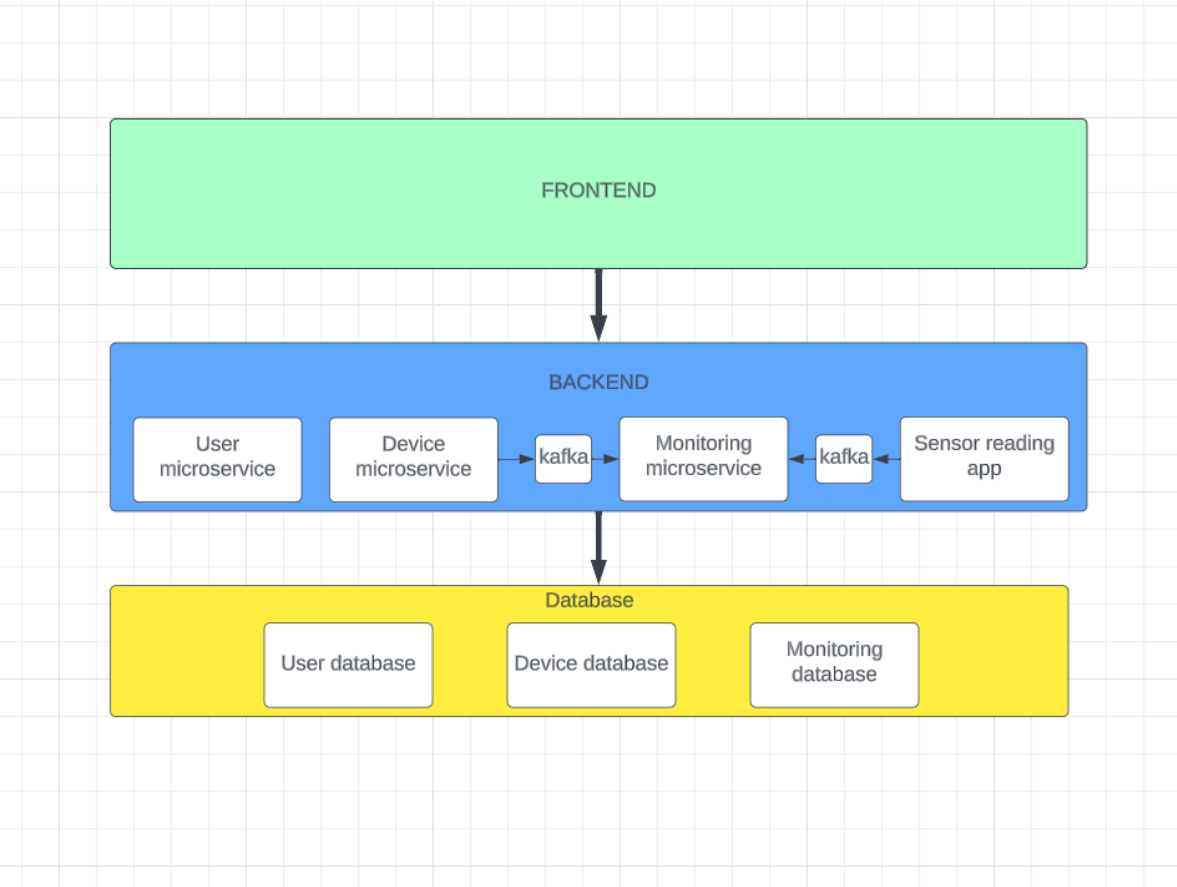
Energy management system

Beiusanu Horia

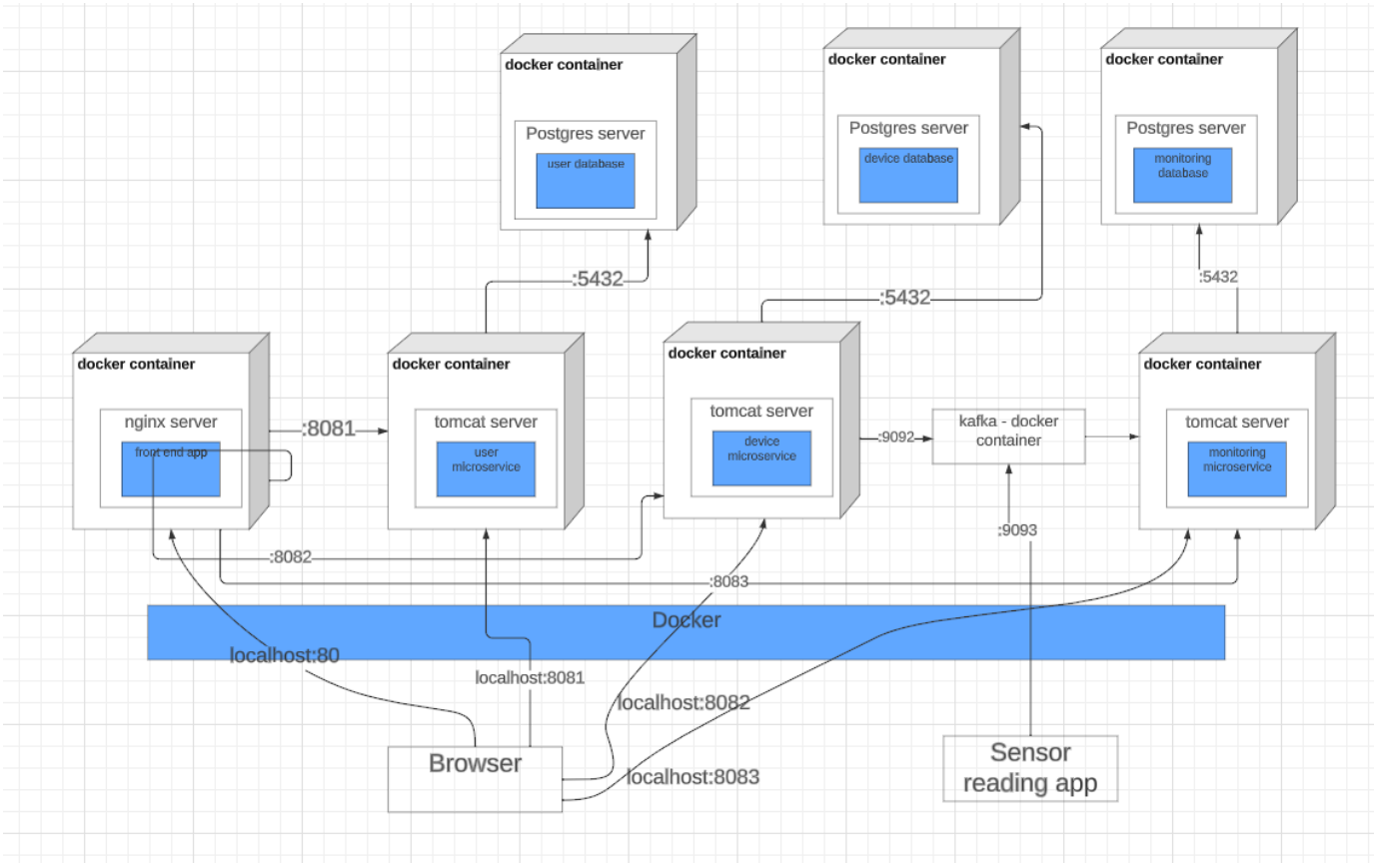
Arhitectura 3-layer



Aplicatia web foloseste arhitectura 3-layer, daca ar fi sa o privim abstractizat top-view. Mai in detaliu, backendul foloseste o arhitectura pe microservicii, mai exact din 2: user si device management. Microserviciile comunica intre ele atat sincron prin HTTP cat si asincron prin kafka si produc side-effects in bazele de date. Fiecare microserviciu are la randul lui cate o baza de date si comunica strict doar cu cea asignata lui. Aceasta arhitectura pe microservicii ne permite sa decuplam functionalitatea, si sa permitem echipe diferite sa lucreze si sa evolueze in mare parte indepependent. De asemenea permita ca fiecare microserviciu sa fie implementat in orice fel de tehnologie, neexistand nicio cuplare intre cele folosite de catre servicii diferite. Drept comparatie, arhitectura monolitica ofera initial o mare viteza de development si o performanta mai ridicata, dar aceste degradeaza semnificativ odata cu evolutia proiectului. Aceasta decuplare permite development independent, dar si o scalare mult mai eficienta, fiecare serviciu putand fi scalat independent, in functie de nevoie, in special prin tehnologii de

orchestrate precum Kubernetes. Sarcina frontendului de a gasi mereu ip-ul si portul corespunzator pentru un anumit request, s-ar putea simplifica prin introducerea unui gateway intre frontened si backend. Frontend-ul comunica cu backend-ul majoritar sincron prin HTTP, dar exista si un use case de notificare a clientilor in caz ca un device depaseste limita maxima de consum, iar acesta are loc prin comunicare bidirectionala asincrona folosind websockets. Pentru simularea citirilor de senzori, avem un alt desktop app care doar citeste valori dintr-un excel, le pune pe un kafka topic(numit energy-topic), iar mai departe aceste mesaje sunt preluate spre procesare de catre montoring microservice. Acest microserviciu citeste timestamp-ul si valoarea, le stocheaza, iar pe urma verifica daca citire depasesc limita curent a acelui device si notifica client-ul prin websockets, daca e nevoie. Notificarile se pot observa atat pe pagina unui client(unde se pot vedea notificari de pe toate device-urile lui), cat si de pe pagina unui device, loc unde se gaseste si graficul cu citirile per ora. In cazul in care un user sterge un device, microserviciul de device trimite un eveniment pe un alt kafka topic(numit device-change-topic), care este citit tot de catre monitoring microservice, care la randul lui, incearca sa sincronizeze bazele de date si sa stearga datele retinute pana atunci pentru acel device.

Diagrama de deployment



Pentru deployment am folosit tehnologia de containerizare Docker. Din browser serviciile se pot accesa prin port-urile externe ale containerelor, dar containerele intre ele vor comunica prin porturile lor

interne. Fiecare micro serviciu detine o imagine proprie si se instatiaza intr-un container. Pentru fiecare microserviciu exista si un container care se ocupa cu baza de date corespunzatoare. Toate aceste containere sunt configurate sa comunice printr-un network extern care trebuie creat inainte de pornire. Acesti pasi sunt descrisi in sectiunea 'pasi pentru build'. In acest network, containerele se adreseaza unul altuia prin numele containerului si prin portul expus de catre acestea. Serverul DNS integrat in docker preia responsabilitatea de a mapa numele containerului la ipul adevarat al acestuia. La start-up containerele de kafka executa un shell script care creeaza topic-urile in caz ca aceste nu exista.

Pasii pentru build:

Configurare initiala: docker network create mynetwork

Pentru microservicii:

Din root-ul proiectului, se executa:

1) ./gradlew clean build –x test –parallel
2) Docker build –t [user-app/device-app/monitoring-app] .

Pentru frontend:

Din root-ul proiectului, se executa

1) npm run build

2) docker build –t energy-fe .

Pentru simulatorul de senzori:

Din root-ul proiectului, se executa:

1) din aplication.properties se introduce id-ul deviceului pentru care vrem sa simulam

2) ./gradlew clean build –x test –parallel

3) rulam din IDE sau cu java –jar "jarul"

Pasii pentru executie:

Pentru a porni toate containerele deodata, din root-ul microserviciului se executa comanda:

docker-compose up --build