Documentatie pentru Aplicatii cu Sisteme Distribuite

Suciu Radu

January 11, 2024

1 Cerinta

Am dezvoltat un Sistem de Gestionare a Energiei care constă într-un frontend și două microservicii proiectate pentru gestionarea utilizatorilor și a dispozitivelor lor de măsurare a energiei inteligente asociate. Sistemul poate fi accesat de două tipuri de utilizatori după un proces de autentificare: administrator (manager) și client. Administratorul poate efectua operațiuni CRUD (Creare-Citire-Actualizare-Ștergere) asupra conturilor de utilizator (definite prin ID, nume, rol: admin/client), dispozitivelor de măsurare a energiei inteligente (definite prin ID, descriere, adresă, consum maxim de energie pe oră) și asupra asociării utilizatorilor cu dispozitivele (fiecare utilizator poate deține unul sau mai multe dispozitive inteligente în diferite locații).

Implementați un microserviciu de monitorizare și comunicare pentru Sistemul de Gestionare a Energiei. Microserviciul se bazează pe un middleware cu broker de mesaje care adună date de la dispozitivele de măsurare inteligente, procesează datele pentru a calcula consumul de energie la nivelul orar și le stochează în baza de date a microserviciului de monitorizare și comunicare.

Sincronizarea între bazele de date ale Microserviciului de Gestionare a Dispozitivelor și noul Microserviciu de Monitorizare și Comunicare se realizează printr-un sistem bazat pe evenimente care utilizează un topic pentru modificările dispozitivului (trimite informații despre dispozitiv printr-o coadă către Microserviciul de Monitorizare și Comunicare).

O aplicație Simulator pentru Dispozitive de Măsurare Inteligente va fi implementată ca Producător de Mesaje. Aceasta va simula un contor inteligent citind date de energie dintr-un fișier sensor.csv (adică o valoare la fiecare 10 minute) și trimite date sub forma < timestamp, device_id, measurement_value > către Brokerul de Mesaje (adică o coadă). Timestamp-ul este preluat de la ceasul local, iar device_id este unic pentru fiecare instanță a Simulatorului pentru Dispozitive de Măsurare Inteligente și corespunde device_id-ului unui utilizator din baza de date (așa cum este definit în Asignarea 1). Simulatorul de dispozitiv ar trebui dezvoltat ca o aplicație independentă (adică o aplicație de desktop).

Dezvoltați un microserviciu de chat și un component de autorizare pentru Sistemul de Management Energetic.
Componenta de autorizare ar trebui să ofere acces securizat al utilizatorilor la microserviciile sistemului.
Microserviciul de chat ar trebui să permită comunicarea între utilizatori și administratorul sistemului, permițându-le să pună întrebări și să primească răspunsuri.

2 Introducere

Sistemele distribuite reprezintă aplicații software complexe care rulează pe mai multe servere sau dispozitive și cooperează pentru a furniza funcționalități extinse. Aceste aplicații sunt elaborate pentru a împărți sarcinile și datele între diverse componente, crescând astfel scalabilitatea, redundanța și performanța. Ele se bazează pe interacțiuni strânse între servere și dispozitivele implicate, permitân d realizarea unor operațiuni complexe și necesare.

Prin intermediul sistemelor distribuite, este posibil să se coordoneze și să se administreze resursele într-un mod mai eficient, asigurându-se că sarcinile sunt distribuite uniform între diferitele componente. Aceasta duce la o mai mare flexibilitate și capacitate de redresare în caz de eșecuri sau perturbări, deoarece aplicația poate să continue să funcționeze fără întreruperi notabile. Totodată, performanța este îmbunătățită prin utilizarea mai multor servere pentru a distribui sarcinile, accelerând astfel procesele și asigurând o experiență mai rapidă și mai eficientă pentru utilizatori

3 Arhitectura sistemului disribuit

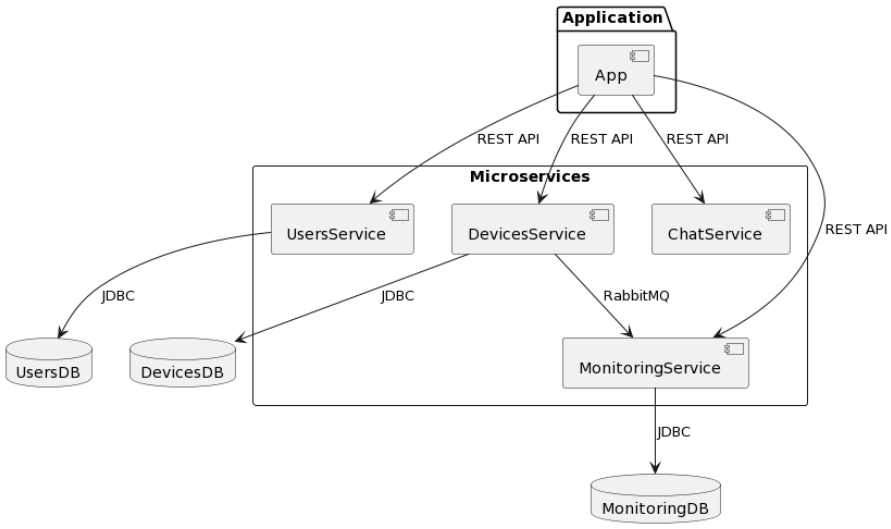
Microserviciile sunt unități independente de dezvoltare și deservire a aplicațiilor, care îndeplinesc funcționalități specifice ale unei aplicații mai mari. Acestea sunt caracterizate de unitate și focus în funcționalitate, comunicare prin API, independență tehnică, scalabilitate separată, dezvoltare și implementare agilă, gestionare simplă a erorilor.

Arhitectura sistemului se bazează pe o aplicație client dezvoltată în ReactJS TypeScript, care se folosește de REST APIs ale a 4 microservicii Java Spring: Users, Devices, Chat și Monitoring. Datele sunt stocate în trei baze de date separate pentru fiecare microserviciu: UsersDB, DevicesDB și MonitoringDB. O reprezentare a arhitecturii poate fi observată în diagrama de mai jos.

În plus, a fost adăugat un nou microserviciu, Monitoring, care se ocupă de monitorizarea și colectarea datelor relevante. Acest microserviciu interacționează cu o bază de date dedicată, MonitoringDB, pentru stocarea informațiilor colectate. Pentru a facilita comunicarea între microservicii, s -a implementat și un serviciu RabbitMQ, care asigură schimbul de mesaje între acestea.

Microserviciul de Chat se ocupa cu transmiterea de semnale prin WebSocket a mesajelor transmise intre admin si client , cat si a semnalelor pentru mesaje citite, mesaje de typing atat al adminului cat si a userului. Transmiterea de mesaje se face prin captarea mesajelor din textbox si la apasarea de send se trimite prin websocket-ul deschis in chat service mesajul spre username-ul selectat. Partea de typing este implementata prin functia onFocusCapture din React care la deschiderea textbox-ului trimite un semnal prin acelasi websocket spre username-ul corespunzator. Partea de read este diferita in functie de rol. Pentru user, la deschiderea dashboard-ului sau se trimite semnalul de read catre admin, cat si de fiecare data cand primeste un mesaj si este logat. Pentru admin, daca primeste un mesaj si are chat-ul deschis pe tab-ul aceluiasi user care a trimis mesaj-ul , se trimite semnal-ul de read. Altfel, cand schimba tab-ul de chat pe alt user, in cazul in care exista un mesaj primit,se trimite semnal-ul de read.

A fost adaugata o parte de security bazata pe un serviciu de autorizare care generează tokenuri recunoscute de alte microservicii care împart aceeași cheie secretă cu serviciul de autorizare. UserService genereaza un JWT token la logare bazat pe un secret detinut de toate microserviciile. Token-ul este trimis spre frontend si retinut in sessionStorage. La orice call spre backend , se trimite si token-ul ca header si se decodeaza cu acelasi secret. Fiecare endpoint este securizat in functie de rol-ul care este atasat token-ului.



3.1 Users Service and DB

Microserviciul Users este responsabil pentru gestionarea informațiilor legate de utilizatori și autentificare. Acesta expune un API la ruta /persons pentru a permite gestionarea entităților de tip "persons". În baza de date usersdb, există o tabelă numită "persons" cu următoarele coloane: id, username, password și role. Acest microserviciu gestionează informațiile despre autentificare și rolurile utilizatorilor în sistem

3.2 Devices Service and DB

Microserviciul Devices se concentrează pe gestionarea dispozitivelor și consumului acestora. Furnizează un API la ruta /devices pentru gestionarea entităților de tip "devices". Datele despre dispozitive sunt stocate în baza de date devicesdb, în tabela "devices", care conține următoarele coloane: id, description, address, maxHourlyConsumption și personUsername. Acest microserviciu se ocupă de înregistrarea, actualizarea și ștergerea dispozitivelor, precum și monitorizarea consumului acestora.

3.3 React Routes

Ruta /login: Această rută este destinată autentificării utilizatorilor. Aici, utilizatorii pot furniza numele de utilizator și parola pentru a accesa aplicația. Microserviciul Users va gestiona procesul de autentificare și va verifica datele împotriva bazei de date usersdb. În funcție de autentificare, utilizatorii vor fi redirecționați către ruta /admin sau /client.

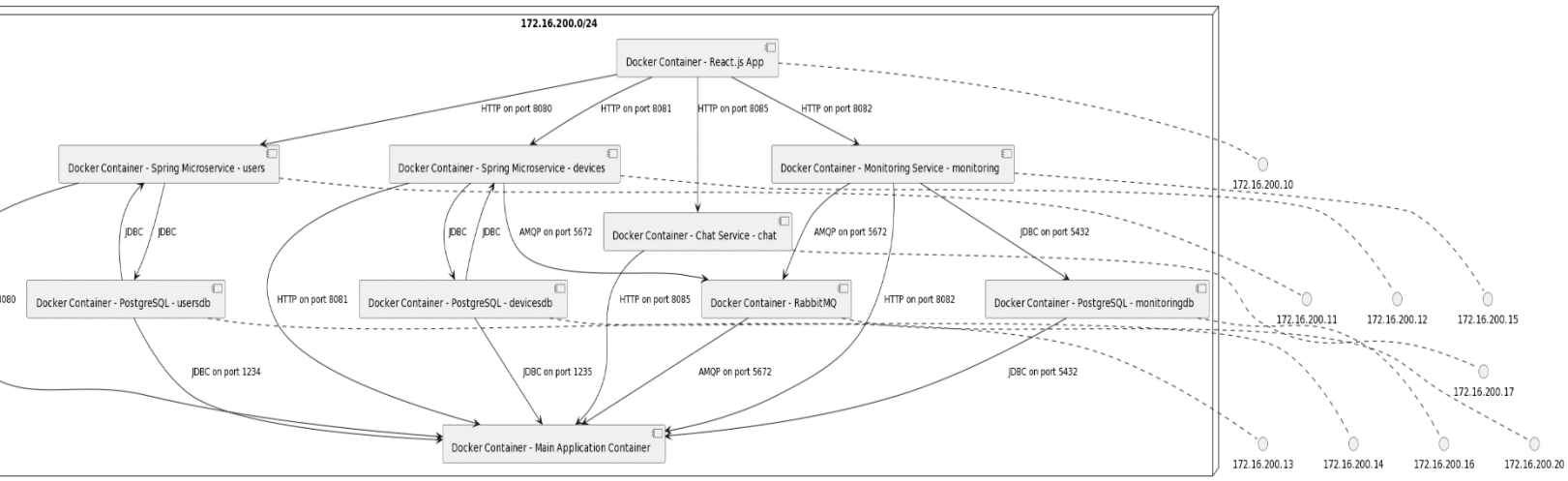
Ruta /admin: Această rută este destinată u tilizatorilor cu rol de administrator. Aici, administratorii pot accesa și gestiona datele legate de utilizatori și dispozitive, inclusiv adăugarea, modificarea și ștergerea acestora. De asemenea, pot face legătura între utilizatori și dispozitive.

Ruta /client: Această rută este destinată utilizatorilor cu rol de client.
Utilizatorii pot accesa informații despre dispozitive și consumul acestora.

4 Deployment

În imaginea de mai sus este prezentată modalitatea de implementare a microserviciilor, bazelor de d ate și aplicației client utilizând Docker.

Deployment-ul unei aplicații este un proces crucial pentru a asigura funcționarea corespunzătoare a acesteia. Un instrument puternic folosit în acest scop este Docker, care permite izolarea și gestionarea eficient ă a containerelor. În cadrul acestui proces, am utilizat Docker Compose pentru a construi un container care să cuprindă întreaga noastră aplicație.



Pentru partea de backend, am creat patru servicii separate, ambele dezvoltate în Java Spring. Am definit Dockerfiles pentru acestea, configurându-le să folosească imaginea openjdk:17. Aceste servicii rulează pe adresele IP 172.16.200.12,
172.16.200.17, 172.16.200.13 si 172.16.200.15, pe porturile 8080, 8081, 8085 si 8082.

Pentru partea de frontend, am folosit un serviciu dezvoltat în React, care a fost configurat într-un Dockerfile pentru a utiliza imaginea node:14. Acest serviciu este disponibil la adresa IP 172.16.200.10 și portul 3000.

Pentru stocarea datelor, am creat trei containere separate pentru bazele de date PostgreSQL. Primul, "usersdb," rulează la adresa IP 172.16.200.13 și portul 1234, al doilea, "devicesdb," rulează la adresa IP 172.16.200.14 și portul 1235 iar al treilea „monitoringdb” la adresa IP 172.16.200.16 Aceste containere asigură persistența datelor aplicației noastre.

Pentru comunicarea intre microservicii avem si un server de RabbitMq ruland pe 5672 la adresa IP 172.16.200.20.

În final, toate aceste componente sunt integrate și comunică între ele în cadrul rețelei definite la adresa IP 172.16.200.0/24. Acest proces de deployment asigură o gestionare ușoară a resurselor și un mediu consistent pentru dezvoltare și testare.

5 Concluzie

În concluzie, aplicațiile cu sisteme distribuite reprezintă o abordare avansată în dezvoltarea software-ului, aducând beneficii semnificative în ceea ce privește scalabilitatea, redundanța și performanța. Aceasta oferă soluții viabile pentru aplicații complexe și necesa