A feladatok általános követelményei

Architektúra

A feladatok tipikusan olyan egyszerű alkalmazás integrációk, amelyek a JBoss vagy Wildfly alkalmazás szerver, vagy docker segítségével megvalósíthatók.

Feladatok beadása

A feladatot a félév végén kell leadni, személyesen bemutatva. Lehet saját laptopon is vagy a labor gépein. Csak email-ben elküldött megoldásokat nem fogadunk el.

Beadási határidő

Az utolsó tanítási héttel bezárólag minden gyakorlaton. Ezután pótlás is lehetséges.

Feladatok

1.

Készítsen egy alkalmazást, amely 2 kliensből áll. Az első kliens a '/queue/colorQueue' üzenetsorra pont-pont csatlakozással véletlenszerűen RED, GREEN és BLUE paraméterrel ellátott üzeneteket küld 1 másodpercenként. Készítsen három MDB-t (üzenet vezérelt bean) amelyek filterrel a 'RED', 'GREEN' és a 'BLUE' paraméterrel ellátott üzeneteket kapják kizárólag. Minden 10 megkapott üzenet után az MDB-k a '/queue/colorStatistics' sorra küldenek egy üzenetet, ami azt jelzi, hogy 10 (adott színű) üzenetet feldolgoztak. Készítsen egy második klienst, ami a '/queue/colorStatistics' sorrol olvassa a statisztikát és a konzolba kiírja hogy pl. '10 'RED' messages has been processed'

2.

Készítsen egy alkalmazást, amely 3 kliensből áll. Az első kliens a '/queue/colorQueue' üzenetsorra pont-pont csatlakozással véletlenszerűen RED, GREEN és BLUE paraméterrel ellátott üzeneteket küld 1 másodpercenként. Készítsen három MDB-t amelyek filterrel a 'RED', 'GREEN' és a 'BLUE' paraméterrel ellátott üzeneteket kapják kizárólag. Az MDB-k véletlenszerűen átlagosan 10 ből 3 szor,

rollback-elik az üzenetet, ami így a halott levél csatornára kerül. Minden 10 sikeresen megkapott (nem rollback-elt) üzenet után az MDB-k a '/queue/colorStatistics' sorra küldenek egy üzenetet, ami azt jelzi, hogy 10 (adott színű) üzenetet feldolgoztak. Készítsen egy második klienst, ami a '/queue/colorStatistics' sorrol olvassa a statisztikát és a konzolba kiírja hogy pl. '10 'RED' messages has been processed'. A harmadik kliens a '/queue/DLQ' halott levél csatornáról a konzolon jelzi, ha egy üzenetet nem dolgoztak fel.

3.

Készítsen alkalmazást, amely az 1. feladat alapján működik, annyi különbséggel, hogy a kliens egy webszolgáltatás, amely SOAP-on keresztül küldi a véletlenszerű színeket a a Wildfly webszolgáltatásnak, és a webszolgáltatás küldi tovább '/queue/colorQueue' üzenetsorra pont-pont csatlakozással az üzeneteket. Az ez utáni teendők megegyeznek az 1.-es feladatban leírtakkal. A lényeges különbség az, hogy a kliens nem kapcsolódik közvetlenül az üzenetsorra, hanem a Wildfly-ban futó szolgáltatás küldi tovább az üzenetet a sorra.

4.

Készítsen alkalmazást, amely az 2. feladat alapján működik, annyi különbséggel, hogy a kliens egy webszolgáltatás, amely SOAP-on keresztül küldi a véletlenszerű színeket a a Wildfly webszolgáltatásnak, és a webszolgáltatás küldi tovább '/queue/colorQueue' üzenetsorra pont-pont csatlakozással az üzeneteket. Az ez utáni teendők megegyeznek az 2.-es feladatban leírtakkal. A lényeges különbség az, hogy a kliens nem kapcsolódik közvetlenül az üzenetsorra, hanem a Wildfly ban futó szolgáltatás küldi tovább az üzenetet a sorra.

5.

Készítsen alkalmazást amelynek egyik kliense a '/queue/colorQueue' üzenetsorra véletlen szöveges üzeneteket küld. ('RED', 'GREEN', 'BLUE') szöveggel 0.5 másodpercenként. Készítsen 3 további klienst amelyek folyamatosan 'hallgatják' a '/queue/colorQueue' üzenetsort, ha valamelyik üzenetet olvas a sorról, akkor véletlenszerűen (2000-9000) milliszekundum ideig várakozik, majd utána újra hallgatja az üzenetsort. (A véletlenszerű feldolgozási idő miatt -e közben egy másik üzenetet a következő kliens fogja feldolgozni). A 3 kliensnek ne legyen konzol outputja, azaz ne írjon ki semmit a konzolra. Minden üzenet feldolgozása után a '/queue/processedColors' sorra küldjenek egy üzenetet, ami azt jelzi, hogy feldolgozták a feladatot. Készítsen egy klienst, ami a '/queue/processedColors' sort olvassa és a megjelenő üzeneteket kiírja az outputra. pl: 'Client 2, processed a 'BLUE' message'. Ennél a feladatnál nem kell a Wildfly-ban komponenst létrehozni.

6.

Készítsen egy képzeletbeli prepaid mobilfeltöltő alkalmazást amely egy tcp alapú socket szerver lesz egy adott porton fogadja a kliensen kéréseit. A kliensek atm automaták, amelyek a következő protokoll szerint működnek: 1.) a kliens egy tesztüzenetben elküldi a feltölteni kívánt telefonszámot, 2.) ha a szám feltölthető, akkor OK egyébként ERROR üzenettel megszakítja a tranzakciót. OK üzenet esetén egy tranzakció azonosítót kapunk, amely a következő 3. lépésben azonosítja a folyamatot. 3.) a kliens a tényleges feltöltést kezdeményezi a korábban kapott tranzakció azonosítóval, de újra el kell küldeni a telefonszámot és a feltöltési összeget.

A lehetséges feltöltési összegek: 3000,5000,10000,15000 Készítsen egy lekérdező klienst, amely egy adott telefonszámhoz tartozó korábbi tranzakciókat listázza. Használjon adatbázist a tranzakciók tárolására a szerver alkalmazásban.

Indítson 3 klienst, amelyek egy előre adott telefonszám halmazt használnak és véletlenszerűen teszt és feltöltés tranzakciókat indítanak.

7.

Készítsen egy kliens alkalmazást, ami http POST kérésekkel szöveges adatokat küld egy üzenetsor vezérelt mintarendszerbe. Úgy tervezze meg az alaprendszert, hogy később nemcsak szöveges, hanem más, pl. bináris állományokat is tudjon kezelni. A mintarendszer alapfeladata, hogy megszámolja a szöveges állományokban a szavak számát és eredményként visszadja a kliensnek. Készítsen egy futtató környezetet amiben több kliens párhuzamosan, véletlenszerű állományokkal használja a szolgáltatást. Készítsen legalább 3 mintaszöveget a teszteléshez. Tételezzük fel, hogy a működő rendszert ki kell egészíteni egy új funkcióval: mostmár bmp formátumú képeket is lehet küldeni, amelyeknél a rendszer a kép méretét adja vissza ereményként. Készítse el úgy a rendszert, hogy menet közben lehessen kicserélni a kliens és szerver komponenseket, azaz nem lehet leállítani a rendszert. A megoldáshoz használja az itt bemutatott elveket: mintapélda

8.

Adott egy A és egy B szerverkomponens. A B komponens hozzáfér egy SQL adatbázishoz, amiben 1 db tábla van, ami személyek adatait tartalmazza (név, szül idő, stb..). Az 'A' komponensnek nincs hozzáférése az adatbázishoz. Az 'A' komponens rendelkezik viszont egy browser-ben megjelenő felülettel, ahol személyek adatait lehet lekérdezni (összes személyt egyszerre) és új személyt lehet felvenni. A 'B' komponens két funkciója: egy új személy felvétele és a személyek lekérdezése az adatbázisból. Készítse el az A és B komponenseket és hozzon létre kapcsolatot közöttük SOAP felhasználásával. Az A és B komponens tetszőleges technológia lehet.

9.

Adott egy A és egy B szerverkomponens. A B komponens hozzáfér egy NO-SQL adatbázishoz, amiben 1 db tábla van, ami könyvek adatait tartalmazza (cím, szerzők, kiadó, év, stb..). Az 'A' komponensnek nincs hozzáférése az adatbázishoz. Az 'A' komponens rendelkezik viszont egy browser-ben megjelenő felülettel, ahol könyvek adatait lehet lekérdezni (összes könyvet egyszerre) és új könyvet is fel lehet venni. A 'B' komponens két funkciója: egy új könyv felvétele és a könyvek lekérdezése az adatbázisból. Készítse el az A és B komponenseket és hozzon létre kapcsolatot közöttük JAX/RS felhasználásával. Az A és B komponens tetszőleges technológia lehet.

10.

Készítsen olyan mintarendszert, ami képes működés közben updgradel-ni. Adott egy kliens komponens (A), ami 2 másodpercenként üzenetet küld a *tasks* üzenetsornak. A kezdeti üzenetek verziója V1.0. Készítsen egy consumer komponenst (B), ami feldolgozza az üzeneteket és a *finishedTasks* üzenetsorra küldi a kész üzeneteket, az üzenetekhez hozzáfűzi az aktuális időt. A B komponens fel legyen készítve arra az esetre, ha az üzenet verziószáma nem V1.0, ekkor az üzenetet az invalidTasks üzenetsorra továbbítja. A (C) consumer komponens az *invalidTasks* üzenetsorról leveszi az üzeneteket és 5 másodperc késleltéssel visszateszi a *tasks* üzenetsorra. Mutassuk be, ha a (A) komponens módosításával v2.0 verziószámú üzeneteket küldünk és a (B) komponensben feldogozzuk a V2.0 üzeneteket (azaz nem küldjük az *invalidTasks* sorra őket) akkor a rendszer kis késleltetéssel, de leállítás nélkül tud üzemelni.

---

11. Fény intenzitás monitorozó rendszer

Készítsen egy alkalmazást, amely a beltéri fényszintet (lux) figyeli. A rendszer három külön kliensből áll: egy adatgenerátorból, egy feldolgozóból és egy riasztás-kezelő kliensből.

Komponens 1: Light Intensity Generation Client

1. Csatlakozás: A kliens a *lightIntensityQueue* pontról-pontra típusú (point-to-point) üzenetsorhoz csatlakozik.
2. Feladat: 3 másodpercenként véletlenszerű fényszint-adatokat (lux értékeket) küld, például 0 és 2000 lux között.
    1. Példa: 300 lux, 1500 lux, stb.

Komponens 2: Light Intensity Processor

1. Üzenetfogadás: Egy processzor, amely kizárólag a *lightIntensityQueue* üzeneteit kapja meg (fényszint mérési adatok).
2. Feldolgozás: Meghatározza, hogy a fényerő értéke túl alacsony-e. Például dönthetünk úgy, hogy 100 lux alatti érték esetén „sötét” állapotot észlelünk.
3. Riasztás küldése: Ha 3 egymást követő mérés alatt marad 100 luxon, a processzor egy

riasztásüzenetet küld a *lightAlertQueue* üzenetsorba azzal a szöveggel, hogy pl. “Low light alert: 3 consecutive readings below 100 lux.”

Komponens 3: Alert Reporting Client

1. Fogyasztás: A kliens a *lightAlertQueue* üzenetsorból olvassa a riasztásokat.
2. Kimenet: A kapott értesítéseket kiírja a konzolra, pl. “Low light alert: 3 consecutive readings below 100 lux.”

Működés tesztelése:

- Üzenetküldés és -fogadás tesztelése: Ellenőrizd, hogy a fényszint-adatokat helyesen küldi és fogadja a rendszer.
- Alacsony fényszint felismerése: Teszteld, hogy 3 egymást követő <100 lux érték esetén a riasztás helyesen továbbítódik.
- Riasztási mechanizmus: Ellenőrizd, hogy a lightAlertQueue-ba kerül-e az üzenet, és a kliens kiírja-e a figyelmeztetést a konzolra.

---

12. Hangerő figyelő rendszer (Sound Level Alert System)

Készítsen egy alkalmazást, amely zajszint (decibel) értékeket figyel egy adott környezetben. Három kliensből áll: egy adatokat generáló kliensből, egy zajszint-feldolgozóból és egy riasztási kliensből.

Komponens 1: Sound Level Generation Client

1. Csatlakozás: A kliens a *soundLevelQueue* pontról-pontra típusú üzenetsorhoz csatlakozik.
2. Feladat: 2 másodpercenként küld véletlenszerű zajszint-adatokat decibelben, pl. 30 dB és 120 dB között.

Komponens 2: Sound Level Processor

1. Üzenetfogadás: Kizárólag a *soundLevelQueue*-ból olvas.
2. Feldolgozás: Meghatározza, hogy a zajszint magas-e. Például 80 dB felett „túl hangos” értéknek tekintjük.
3. Riasztás: Ha egy időn belül 5 db “túl hangos” érték (80 dB felett) érkezik, küld egy üzenetet a *soundAlertQueue*-ba: “High noise alert: 5 high decibel readings detected.”
    1. Megjegyzés: Dönthetünk úgy, hogy egymást követő 5 mérés is elegendő, vagy a feldolgozó számlálja, amíg össze nem gyűlik 5 hangos mérés, majd riaszt.

Komponens 3: Alert Reporting Client

1. Fogyasztás: Feliratkozik a *soundAlertQueue* üzenetsorra.
2. Kimenet: Kiírja a konzolra: “High noise alert: 5 high decibel readings detected.”

Tesztelési feladat:

- Üzenetküldés és -fogadás tesztelése: Győződjön meg róla, hogy a decibel értékek áramlása zavartalan.
- Magas zajszint azonosítása: Ellenőrizze, hogy 80 dB felett történő mérésekből 5 után riasztás jön létre.

- Riasztási üzenet: Vizsgálja meg, hogy megjelenik-e a kívánt üzenet a *soundAlertQueue*-ban és a konzolon.

---

13. Nyomás figyelő rendszer (Pressure Monitoring System)

Készítsen egy alkalmazást, amely folyadék- vagy gáznyomás-adatokat figyel egy rendszerben. A feladat során három klienssel dolgozunk: egy nyomásgeneráló klienssel, egy nyomást figyelő processzorral és egy riasztást kiértékelő klienssel.

Komponens 1: Pressure Generation Client

1. Csatlakozás: A kliens a *pressureQueue* pontról-pontra típusú üzenetsorhoz csatlakozik.
2. Feladat: 4 másodpercenként küld véletlenszerű nyomásértékeket (pl. 0 és 10 bar között).

Komponens 2: Pressure Alert Processor

1. Üzenetfogadás: Kizárólag a *pressureQueue* üzeneteit kapja.
2. Feldolgozás: Meghatározza, hogy a nyomás értéke veszélyesen magas-e (például 8 bar felett).
3. Riasztás küldése: Ha 2 egymást követő mérés 8 bar felett van, akkor a processzor riasztást küld a *pressureAlertQueue* üzenetsorba: “High pressure alert: 2 consecutive readings above 8 bar detected.”

Komponens 3: Alert Reporting Client

1. Fogyasztás: A kliens olvassa a *pressureAlertQueue* üzeneteit.
2. Kimenet: Konzolra írja a riasztás szövegét, pl. “High pressure alert: 2 consecutive readings above 8 bar detected.”

Tesztek

- Üzenetek küldése és fogadása: Ellenőrizze, hogy a nyomásértékek megfelelően jutnak el az első kliensből a processzorig.
- Magas nyomás azonosítása: Tesztelje, hogy a processzor 8 bar felett helyesen detektálja-e a veszélyes értékeket.
- Consecutive readings logika: Vizsgálja meg, hogy pontosan akkor jön-e létre riasztás, ha 2 egymást követő mérés magas.
