Architettura dei sistemi software

Progetto per l'A.A. 2024/2025

11 novembre 2024

Premessa

Il progetto del corso di Architettura dei sistemi software è relativo alla sperimentazione pratica delle tecnologie studiate durante il corso, e riguarda la realizzazione di una semplice applicazione a microservizi e il rilascio di questa applicazione in un ambiente di esecuzione a container.

Il progetto è basato sullo svolgimento di diverse attività (alcune delle quali sono obbligatorie mentre altre sono opzionali) e prevede anche alcune varianti, come descritto nel seguito di questo documento.

Progetto (punto di partenza)

GoodMusic è un semplice social network per la condivisione di recensioni di album musicali.

Il codice sorgente di **GoodMusic** è disponibile sul repository GitHub del corso (https://github.com/aswroma3/asw/tree/main/projects/asw-goodmusic).

GoodMusic viene usato come segue.

- Gli utenti del sistema possono scrivere e pubblicare delle recensioni di album musicali.
- Gli utenti possono anche seguire le recensioni di album, relative a specifici recensori e/o a specifici artisti e/o a specifici generi musicali. Per esempio, Alice potrebbe seguire le recensioni scritte da Woody, Bob potrebbe seguire le recensioni degli album dei Pink Floyd e Carlo potrebbe seguire le recensioni degli album Pop.
- Quando un utente accede alla pagina delle recensioni che segue, gli vengono mostrate le recensioni degli album dei recensori, degli artisti e dei generi musicali che segue.

Ogni recensione è caratterizzata da:

- un recensore (ovvero chi ha scritto la recensione) – per es., *Woody*
- un album musicale, a cui si riferisce – per es., *The Dark Side of the Moon*
- l'artista che ha realizzato l'album – per es., *Pink Floyd*
- il genere musicale dell'album – per es., *Rock*
- il testo della recensione – che potrebbe essere molto lungo
- un sunto della recensione – per es., *Il lato buio dell'animo umano*

Una recensione in formato breve è una recensione che non contiene il testo completo della recensione, ma solo il suo sunto. Per motivi di prestazioni, le operazioni che restituiscono più recensioni, le restituiscono in formato breve.

L'applicazione **GoodMusic** è composta dai seguenti microservizi:

- Il servizio **recensioni** gestisce le recensioni di album musicali. Ogni recensione ha un recensore (chi ha scritto la recensione), un album musicale, l'artista che ha realizzato l'album, il genere dell'album, il testo della recensione e un sunto della recensione. (Per semplicità, questi attributi sono tutte stringhe.) Operazioni:
  - POST /recensioni aggiunge una nuova recensione (dati recensore, album, artista, genere, testo della recensione e sunto)

    - GET /recensioni/{id} trova una recensione, dato il suo id
    - GET /recensioni trova tutte le recensioni (in formato breve)
    - GET /cercarecensioni/album/{album} trova tutte le recensioni (in formato breve) di un certo album
    - GET /cercarecensioni/recensore/{recensore} trova tutte le recensioni scritte da un certo recensore
    - GET /cercarecensioni/recensori/{elenco-di-recensori} trova tutte le recensioni di un insieme di recensori
    - GET /cercarecensioni/artista/{artista} trova tutte le recensioni degli album di un certo artista
    - GET /cercarecensioni/artisti/{elenco-di-artisti} trova tutte le recensioni degli album di un insieme di artisti
    - GET /cercarecensioni/genere/{genere} trova tutte le recensioni degli album di un certo genere musicale
    - GET /cercarecensioni/generi/{elenco-di-generi} trova tutte le recensioni degli album di un insieme di generi
- Il servizio **connessioni** gestisce le connessioni degli utenti con le recensioni che seguono, e più precisamente con gli artisti, con i recensori e con i generi musicali che essi seguono. Una connessione è una terna *utente-seguito-ruolo*, in cui l'*utente* è chi segue, *seguito* è chi o che cosa è seguito (un artista oppure uno che scrive recensioni oppure un genere musicale) e *ruolo* rappresenta il ruolo del seguito, e può essere *ARTISTA* oppure *RECENSORE* oppure *GENERE*. (Per semplicità, questi attributi sono tutte stringhe.) Operazioni:
    - POST /connessioni aggiunge una nuova connessione utente-seguito-ruolo (dati utente, seguito e ruolo)
    - GET /connessioni trova tutte le connessioni
    - GET /connessioni/{utente} trova tutte le connessioni di un certo utente
    - GET /connessioni/{utente}/{ruolo} trova tutte le connessioni di un certo utente relative a un certo ruolo
    - DELETE /connessioni cancella una connessione utente-seguito-ruolo (dati utente, seguito e ruolo)
- Il servizio **recensioni-seguite** consente a un utente di trovare le recensioni degli artisti e dei recensori e dei generi musicali che segue. Operazioni:
    - GET /recensioniseguite/{utente} trova tutte le recensioni seguite da un certo utente, ovvero le recensioni di artisti di album e di recensori e di generi musicali seguiti da quell'utente (le recensioni sono in formato breve)
- Il servizio **api-gateway** (esposto sulla porta 8080) è l'API gateway dell'applicazione, che:
    - espone il servizio **recensioni** sul path /recensioni; per es., GET /recensioni/recensioni
    - espone il servizio **connessioni** sul path /connessioni;
      per es., GET /connessioni/connessioni/{utente}
    - espone il servizio **recensioni-seguite** sul path /recensioni-seguite;
      per es., GET /recensioni-seguite/recensioniseguite/{utente}

Sul repository GitHub del corso, l'implementazione dell'operazione GET /recensioniseguite/U del servizio **recensioni-seguite**, per trovare le recensioni seguite dall'utente U, è basata su invocazioni remote REST ai servizi **connessioni** e **recensioni**, come segue:

- prima viene invocata l'operazione GET /connessioni/U di **connessioni** per trovare l'insieme CC delle connessioni relative all'utente U
- a partire da CC vengono determinati gli insiemi AA degli artisti, RR dei recensori e GG dei generi seguiti dall'utente U

- poi vengono invocate le operazioni GET /cercarecensioni/autori/AA di **recensioni** (se AA non è vuoto), GET /cercarecensioni/recensori/RR di **recensioni** (se RR non è vuoto) e GET /cercarecensioni/generi/GG di **recensioni** (se GG non è vuoto), per trovare le recensioni seguite da U (in formato breve)
- viene infine restituito l'insieme di tutte queste recensioni (in formato breve), che sono proprio quelle seguite dall'utente U

Discussione

L'implementazione del servizio **recensioni-seguite** soffri di alcuni problemi. In particolare:

- Ogni volta che si vuole accedere alle recensioni seguite da un certo utente è necessario movimentare in rete una grande quantità di dati.
- Inoltre, questo servizio dipende fortemente dai servizi **recensioni** e **connessioni**, e se anche uno solo di questi due servizi non è disponibile allora non lo è nemmeno il servizio **recensioni-seguite**.
- Infine, questo servizio non è molto scalabile, perché nessuno dei servizi è replicato e perché i diversi servizi comunicano in modo sincrono.

L'accesso alle recensioni seguite da un certo utente è probabilmente l'operazione eseguita più frequentemente nel social network **GoodMusic**, e pertanto va implementata in modo da sostenere prestazioni, scalabilità e disponibilità.

Progetto (attività)

Il progetto consiste nel modificare l'applicazione presente sul repository GitHub del corso, svolgendo le seguenti attività (alcune delle quali sono obbligatorie mentre altre sono opzionali).

Modifica del codice e della configurazione dell'applicazione

Una prima modifica (obbligatoria) riguarda la modifica del codice dell'applicazione, come segue:

- In effetti, mentre gli altri servizi funzionano correttamente, il servizio **recensioni-seguite** non prende in considerazione le connessioni relative ai generi musicali, che vengono ignorate. Il codice con cui questo servizio accede alle recensioni basate sui generi musicali seguiti da un utente è presente solo in parte, e va completato.

Una seconda modifica (obbligatoria) riguarda la modifica della configurazione dell'applicazione, come segue:

- Nei servizi **recensioni** e **connessioni**, bisogna usare delle basi di dati PostgreSQL o MySQL al posto delle basi di dati HSQLDB. In particolare, ciascun servizio deve avere una propria base di dati, che va eseguita in un container Docker separato.

Una terza modifica (anche questa obbligatoria) riguarda la modifica del codice e della configurazione dell'applicazione, come segue:

- Modificare la logica del servizio **recensioni-seguite**, per migliorare le sue prestazioni, scalabilità e disponibilità, come descritto nel seguito.

Si potrebbe pensare di modificare la logica del servizio **recensioni-seguite** utilizzando delle invocazioni remote *asincrone* (anziché *sincrone*), e di eseguire l'algoritmo per il calcolo delle recensioni seguiti da un utente nel modo più concorrente possibile. Anche questo approccio richiede però di movimentare una grande quantità di dati in rete quando si vogliono trovare le recensioni

seguite da un utente. Inoltre, la disponibilità del servizio **recensioni-seguite** continua a dipendere fortemente dai servizi **recensioni** e **connessioni**. Dunque, bisogna cercare una soluzione migliore.

Una soluzione probabilmente migliore consiste invece nell'invertire la logica del servizio **recensioni-seguite**, come segue:

- Quando viene aggiunta una nuova recensione, il servizio **recensioni** deve notificare un evento **RecensioneCreatedEvent** (con tutti i dati della recensione breve), su un apposito canale per messaggi.
- Quando viene aggiunta una nuova connessione utente-seguito-ruolo, il servizio **connessioni** deve notificare un evento **ConnessioneCreatedEvent** (con tutti i dati della connessione), su un apposito canale per messaggi.
- Quando viene cancellata una connessione utente-seguito-ruolo, il servizio **connessioni** deve notificare un evento **ConnessioneDeletedEvent** (con tutti i dati della connessione), su un apposito canale per messaggi.
- Il servizio **recensioni-seguite** deve gestire una propria base di dati (separata dalle precedenti, ed eseguita in un container Docker separato), con una tabella per le recensioni (in formato breve) e una tabella per le connessioni.
- Ogni volta che il servizio **recensioni-seguite** riceve un evento **RecensioneCreatedEvent**, allora deve aggiornare di conseguenza la propria tabella delle recensioni.
- Ogni volta che il servizio **recensioni-seguiti** riceve un evento **ConnessioneCreatedEvent** oppure **ConnessioneDeletedEvent**, allora deve aggiornare di conseguenza la propria tabella delle connessioni.
- Il servizio **recensioni-seguite** potrà poi rispondere alle richieste GET /recensioniseguite/{utente} accedendo solo alla propria base di dati.

I canali per messaggi possono essere gestiti con Apache Kafka (in esecuzione in un container Docker separato) oppure anche con un message broker differente.

Si noti che anche questa soluzione può richiedere di movimentare molti dati in rete quando vengono aggiunte nuove recensioni o nuove connessioni, che però sono operazioni meno frequenti che non il trovare le recensioni seguite da un utente. Invece l'accesso alle recensioni seguite da un utente viene effettuato solamente nell'ambito del servizio **recensioni-seguite**.

Nel realizzare queste modifiche, si raccomanda di mantenere l'architettura esagonale dei diversi servizi; in particolare:

- la logica di business (comprese le porte) va collocata nell'interno di ogni servizio (ovvero nel package **domain**); nessuna logica infrastrutturale nell'interno di un servizio;
- le responsabilità infrastrutturali (gli adattatori) vanno collocate nell'esterno di ogni servizio (ovvero, in tutti gli altri package); nessuna logica di business negli adattatori.

Modifica della modalità di rilascio dell'applicazione

Le seguenti ulteriori modifiche riguardano la modifica della modalità di rilascio dell'applicazione, sulla base di diverse possibili attività e varianti:

- Eseguire i diversi servizi ciascuno in un proprio container Docker (obbligatorio).
- Mandare in esecuzione più istanze di ciascun servizio (obbligatorio). Per semplicità, le basi di dati non vanno replicate.
- Eseguire i diversi servizi in container Docker, utilizzando Docker Compose (opzionale).
- Eseguire i diversi servizi in container, utilizzando Kubernetes (opzionale).

Utilizzo di Kafka con Docker e Kubernetes

Nel rilascio dell'applicazione **GoodMusic** con Docker e Kubernetes, uno degli aspetti più difficili è probabilmente l'utilizzo di Kafka. A questo link è possibile trovare alcuni utili suggerimenti in proposito: https://github.com/aswroma3/asw/tree/main/projects/asw-goodmusic/kafka/.

Modalità di svolgimento e di consegna del progetto

Il progetto va svolto in gruppi composti preferibilmente da 3-5 studenti.

Ciascun gruppo dovrà interagire con il docente in questo modo:

- Ciascun gruppo dovrà comunicare al docente, per posta elettronica, appena possibile, la composizione del proprio gruppo, la soluzione che si intende implementare e le tecnologie che si intendono utilizzare. Sia la soluzione che le tecnologie potrebbero essere diverse da quelle proposte in questo documento. Questa comunicazione non è vincolante, perché ogni gruppo potrà poi decidere diversamente da quanto indicato, purché nel rispetto dei requisiti del progetto.
- Poi, ciascun gruppo dovrà realizzare e verificare (sul proprio computer) la propria applicazione distribuita.
- Infine, ciascun gruppo dovrà caricare la propria applicazione distribuita su GitHub (o altro servizio di condivisione del codice). In particolare, dovrà caricare tutto il codice sorgente dell'applicazione, oltre a ogni script necessario per la compilazione e costruzione dell'applicazione (Gradle), i file Dockerfile e gli eventuali file per Docker Compose o Kubernetes, nonché gli script per mandare in esecuzione l'applicazione.
- Al completamento del progetto, il gruppo dovrà comunicare al docente, per posta elettronica, l'URI su GitHub del codice dell'applicazione, insieme a una descrizione sintetica della soluzione effettivamente implementata, delle attività svolte e delle tecnologie effettivamente utilizzate.

Valutazione del progetto

La valutazione del progetto riguarderà i seguenti aspetti:

- Le realizzazioni delle modifiche indicate come "obbligatorie", e la loro correttezza.
- Quante e quali varianti indicate come "opzionali" sono state realizzate, e la loro correttezza.
- Il rispetto dell'architettura esagonale.

Altri progetti

Ciascun gruppo può formulare, se vuole, una propria proposta di progetto (che deve comunque avere finalità simili a quelle del progetto illustrato in questo documento). In ogni caso, queste proposte di progetti alternativi devono essere autorizzate preventivamente dal docente.