Architettura dei sistemi software

Progetto per l'A.A. 2025/2026

4 novembre 2025

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)

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

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

BetterMusic viene usato come segue.

- Gli utenti del sistema possono registrare nuovi album musicali.
- Gli utenti del sistema possono scrivere e pubblicare delle recensioni di questi album.
- Gli utenti possono poi specificare quali sono le recensioni di album a cui sono interessati e che vogliono seguire. In particolare, possono seguire le recensioni scritte da specifici recensori e/o le recensioni di album di specifici artisti e/o le recensioni di album di specifici generi musicali. Per esempio, Alice potrebbe seguire le recensioni degli album dei Pink Floyd, Bob potrebbe seguire le recensioni scritte da Woody e Carlo potrebbe seguire le recensioni degli album Rock.
- Quando un utente accede alla pagina delle recensioni che segue, gli vengono mostrate le recensioni degli album degli artisti, dei recensori e dei generi musicali che segue.

Ogni album è caratterizzato da:

- un id univoco – per es., *1*
- un titolo – per es., *The Dark Side of the Moon*
- l'artista che lo ha realizzato – per es., *Pink Floyd*
- uno o più generi musicali – per es., *Rock* e *Progressive Rock*

Inoltre, ogni recensione è caratterizzata da:

- un id univoco – per es., *101*
- un recensore (ovvero chi ha scritto la recensione) – per es., *Woody*
- l'id dell'album musicale a cui si riferisce – per es., *1* (che in questo esempio corrisponde a *The Dark Side of the Moon* dei *Pink Floyd*)
- 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 **BetterMusic** è composta dai seguenti microservizi:

- Il servizio **album** gestisce gli album musicali. Ogni album ha un titolo ed è stato realizzato da un artista, e può essere relativo ad uno o più generi musicali. Inoltre, ogni album ha un id numerico univoco. Anche titolo ed artista identificano univocamente gli album. (Per semplicità, tutti gli attributi sono stringhe, tranne l'id che è numerico e la lista di generi, che è una lista di stringhe.) Operazioni:
  - POST /album aggiunge un nuovo album (dati titolo, artista e una lista di generi) e gli assegna un id univoco
  - GET /album/{id} trova un album, dato il suo id
  - GET /album trova tutti gli album
  - GET /cercaalbum?titolo={titolo}&artista={artista} trova un album, dati titolo e artista
  - GET /cercaalbum/artista/{artista} trova tutti gli album di un certo artista
  - GET /cercaalbum/genere/{genere} trova tutti gli album di un certo genere
- Il servizio **recensioni** gestisce le recensioni degli album. Ogni recensione è scritta da un recensore, si riferisce ad un album (tramite il suo id), e contiene il testo della recensione (che può essere molto lungo) ed un suo sunto. Inoltre, ogni recensione ha un id numerico univoco. (Per semplicità, tutti gli attributi sono stringhe, tranne l'id che è numerico.) Inoltre, 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. Operazioni:
  - POST /recensioni aggiunge una nuova recensione (dati recensore, titolo dell'album, artista dell'album, testo della recensione e sunto) e gli assegna un id univoco
  - GET /recensioni/{id} trova una recensione, dato il suo id
  - GET /recensioni trova tutte le recensioni (in formato breve)
  - GET /cercarecensioni/album/{idAlbum} trova tutte le recensioni (in formato breve) di un certo album, dato il suo id
  - GET /cercarecensioni/recensore/{recensore} trova tutte le recensioni scritte da un certo recensore
- 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 **album** sul path /album; per es., GET /album/album/{id}

- 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**, **recensioni** e **album**, 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 viene fatto quanto segue, per trovare le recensioni seguite dall'utente U:
  - per trovare le recensioni dei recensori nell'insieme RR, si trovano le recensioni di ogni recensore R in RR (questo richiede l'esecuzione di molte operazioni GET /recensioni/cercarecensioni/recensore/{recensore}, una per ogni recensore)
  - per trovare le recensioni degli album nell'insieme AA, si trovano prima gli album dell'artista A per ogni artista A in AA (questo richiede l'esecuzione di molte operazioni GET /album/cercaalbum/artista/{artista}, una per ogni artista), e poi si cercano tutte le recensioni di ogni album in questo insieme di album (questo richiede l'esecuzione di molte operazioni GET /recensioni/cercarecensioni/album/{idAlbum}, una per ogni album)
  - infine, per trovare le recensioni dei generi nell'insieme GG, si trovano prima gli album del genere G per ogni genere G in GG (questo richiede l'esecuzione di molte operazioni GET /album/cercaalbum/genere/{genere}, una per ogni genere), e poi si cercano tutte le recensioni di ogni album in questo insieme di album (questo richiede l'esecuzione di molte operazioni GET /recensioni/cercarecensioni/album/{idAlbum}, una per ogni album)
- viene infine restituito l'insieme di tutte queste recensioni (in formato breve), che sono proprio quelle seguite dall'utente U

Inoltre, l'implementazione dell'operazione POST /recensioni del servizio **recensioni**, per creare una nuova recensione, richiede anche un'invocazione dell'operazione GET /album/cercaalbum?titolo={titolo}&artista={artista} per trovare l'id dell'album a cui la recensione si riferisce.

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 effettuare molte invocazioni remote (potrebbero essere centinaia o anche di più!) e movimentare in rete una grande quantità di dati.
- Inoltre, questo servizio dipende fortemente dai servizi **connessioni**, **recensioni** e **album**, e se anche uno solo di questi 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 **BetterMusic**, e pertanto va implementata in modo da sostenere prestazioni, scalabilità e disponibilità.

Sicuramente l'accesso alle recensioni seguite da un utente può essere implementato in modo da migliorare le prestazioni (per esempio, facendo meno invocazioni remote a grana più grossa), ma questo non risolve comunque i problemi di disponibilità e scalabilità

Inoltre, anche l'implementazione del servizio **recensioni** soffri di alcuni problemi. In particolare:

- Ogni volta che si vuole creare una nuova recensione è necessario effettuare un'invocazione remota per trovare l'id dell'album a cui si riferisce la recensione.
- Dunque, questo servizio dipende dal servizio **album** e, se questo servizio non è disponibile allora il servizio **recensioni** è disponibile solo in modo parziale.

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 dei servizi

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

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

Per evitare che la creazione di una nuova recensione richieda di effettuare un'invocazione remota al servizio degli **album**, si può invertire la logica del servizio **recensioni** come segue:

- Quando viene aggiunto un nuovo album, il servizio **album** deve notificare un evento **AlbumCreatedEvent** (con tutti i dati dell'album), su un apposito canale per messaggi.
- Il servizio **recensioni** deve gestire nella propria base di dati anche una tabella per gli album.
- Ogni volta che il servizio **recensioni** riceve un evento **AlbumCreatedEvent**, allora deve aggiornare di conseguenza la propria tabella degli album.
- Il servizio **recensioni** potrà poi rispondere alle richieste POST /recensioni accedendo solo alla propria base di dati, senza effettuare nessuna invocazione remota.

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

Una **seconda 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 e a grana più grande possibile (anche aggiungendo nuove operazioni agli altri servizi). 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** continuerebbe a dipendere fortemente dai servizi **connessioni**, **recensioni** e **album**. Dunque, bisogna cercare una soluzione migliore.

Una soluzione probabilmente migliore consiste invece nell'invertire la logica del servizio **recensioni-seguite** (in modo simile a quanto già fatto per il servizio **recensioni**), come segue:

- Quando viene aggiunto un nuovo album, il servizio **album** deve notificare un evento **AlbumCreatedEvent** (con tutti i dati dell'album), su un apposito canale per messaggi. (Questa attività in realtà dovrebbe essere stata gli svolta nell'implementazione della prima modifica.)
- 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 da quelle degli altri servizi), con una tabella per gli album, una tabella per le recensioni (in formato breve) e una tabella per le connessioni.
- Ogni volta che il servizio **recensioni-seguite** riceve un evento **AlbumCreatedEvent**, allora deve aggiornare di conseguenza la propria tabella degli album.
- 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-seguite** 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, senza effettuare nessuna invocazione remota.

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 aggiunti nuovi album, 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**.

Una **terza modifica** (anche questa obbligatoria) riguarda la modifica della configurazione dei servizi che compongono l'applicazione, come segue:

- Nei servizi **album**, **recensioni**, **connessioni** e **recensioni-seguite** 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.

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 **BetterMusic** 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-bettermusic/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.