UNIVERSITÀ
degli STUDI
di CATANIA

DIPARTIMENTO DI INGEGNERIA
ELETTRICA ELETTRONICA E INFORMATICA

DISTRIBUTED SYSTEM AND BIG DATA

EDU CONNECT APP

Piero Galatà 1000066734

Indice

1. Abstract

L’elaborato proposto si focalizza sulla progettazione ed implementazione di una piattaforma a **microservizi**, sviluppata per supportare la gestione di corsi erogati da una scuola che offre formazione linguistica ed informatica (es. corsi di inglese, francese, informatica, etc.). Il sistema consente agli studenti di registrarsi, iscriversi ai corsi di interesse e ricevere notifiche in tempo reale sui corsi d'interesse, tramite un meccanismo **Pub/Sub** implementato con **Apache Kafka**. L’architettura si basa su un insieme di microservizi containerizzati con **Docker** e orchestrati tramite **Kubernetes**, ognuno con responsabilità specifiche: autenticazione, gestione utenti, gestione corsi, notifiche ed elaborazione predittiva delle metriche di sistema.

Docker, attraverso la sua capacità di creare container isolati con tutte le dipendenze necessarie, semplifica la distribuzione e l'esecuzione dell'applicazione. Ciò non solo favorisce la portabilità su diverse infrastrutture, ma migliora anche la scalabilità e la gestione efficiente delle risorse, ottimizzando il deployment.

Mentre Kubernetes, indicato anche con k8s, offre un ambiente orchestrato che semplifica la gestione dei container in un'infrastruttura distribuita. Questo arricchisce l'architettura basata, consentendo una gestione avanzata e dinamica dell'infrastruttura distribuita, garantendo una distribuzione efficiente, scalabile e affidabile.

Inoltre, il sistema integra **Prometheus e cAdvisor** per il monitoraggio sia di metriche white-box (tempi di risposta, numero di richieste) che di metriche black-box (consumo di memoria, tempo di avvio container, errori di rete). Tali dati vengono analizzati dal microservizio Predictor, che utilizza modelli **ARIMA** per prevedere l’andamento delle prestazioni delle API.

L'architettura basata su microservizi non solo offre una maggiore modularità e scalabilità ma facilita anche la manutenzione e l'espansione dell'applicazione con la possibilità di implementazione di nuove funzionalità future. L'approccio adottato mira a fornire un sistema efficiente e flessibile, adattabile alle esigenze dinamiche degli utenti e alle evoluzioni del contesto informativo.

2. Architettura del sistema

L’**architettura** di EduConnect è basata su un approccio a microservizi, in cui ogni componente del sistema è indipendente e responsabile di una specifica funzionalità. L’obiettivo è garantire modularità, scalabilità e semplicità di manutenzione, riducendo al minimo le dipendenze tra i vari servizi. Tutti i microservizi sono containerizzati tramite Docker e orchestrati in Kubernetes, che gestisce il deployment, il bilanciamento del carico e la resilienza dei pod.

Il sistema si compone di diversi microservizi principali:

- **Api Gateway**: nella distribuzione basata su Docker Compose, l’API Gateway rappresenta l’entry point centralizzato dell’applicazione. Si occupa di ricevere tutte le richieste esterne e di indirizzarle dinamicamente verso il microservizio corretto, semplificando la comunicazione e nascondendo la complessità interna del sistema. In ambiente **Kubernetes**, questo ruolo non è più svolto da un microservizio dedicato, ma viene sostituito dall’**Ingress**

Controller, che fornisce funzionalità di routing e bilanciamento direttamente a livello di orchestratore.

- **Auth Service**: si occupa della gestione dell'autenticazione, della generazione dei token JWT, consentendo l'accesso sicuro alle API, e della validazione dei token, precedentemente emessi, verificandone l'autenticità e la scadenza. Questo meccanismo garantisce che solo gli utenti correttamente autenticati possono accedere ad API specifiche del sistema.

- **User Service**: gestisce le informazioni sugli utenti registrati, permettendo la creazione, la modifica e la consultazione dei profili. Questo microservizio è fondamentale per garantire una corretta gestione delle identità e per mantenere separati i dati degli utenti dagli altri servizi, in linea con i principi di indipendenza e modularità tipici dell'architettura a microservizi.

- **Course Service**: gestisce tutte le funzionalità legate ai corsi disponibili nella piattaforma. Consente la creazione di nuovi corsi, la gestione delle iscrizioni e l'associazione studenti-corsi, ovvero permette agli studenti di registrarsi ai corsi e quindi automaticamente di iscrversi al topic relativo a quel corso per ricevere aggironamenti real time. Ogni corso può avere più studenti iscritti, e la relazione molti-a-molti tra corsi e utenti è gestita tramite una tabella di iscrizione dedicata.

- **Publisher Service**: è responsabile della gestione dei topic Kafka relativi ai corsi. Ogni volta che viene creato un nuovo corso, il Publisher genera un **topic dedicato** (course-<titolo>) e si occupa della **pubblicazione dei messaggi** al suo interno, trmaite API apposita. In questo modo è possibile informare gli studenti iscritti riguardo aggiornamenti, annunci o notifiche legate al corso. Grazie a questo microservizio, la comunicazione asincrona tra sistema e studenti risulta affidabile e scalabile.

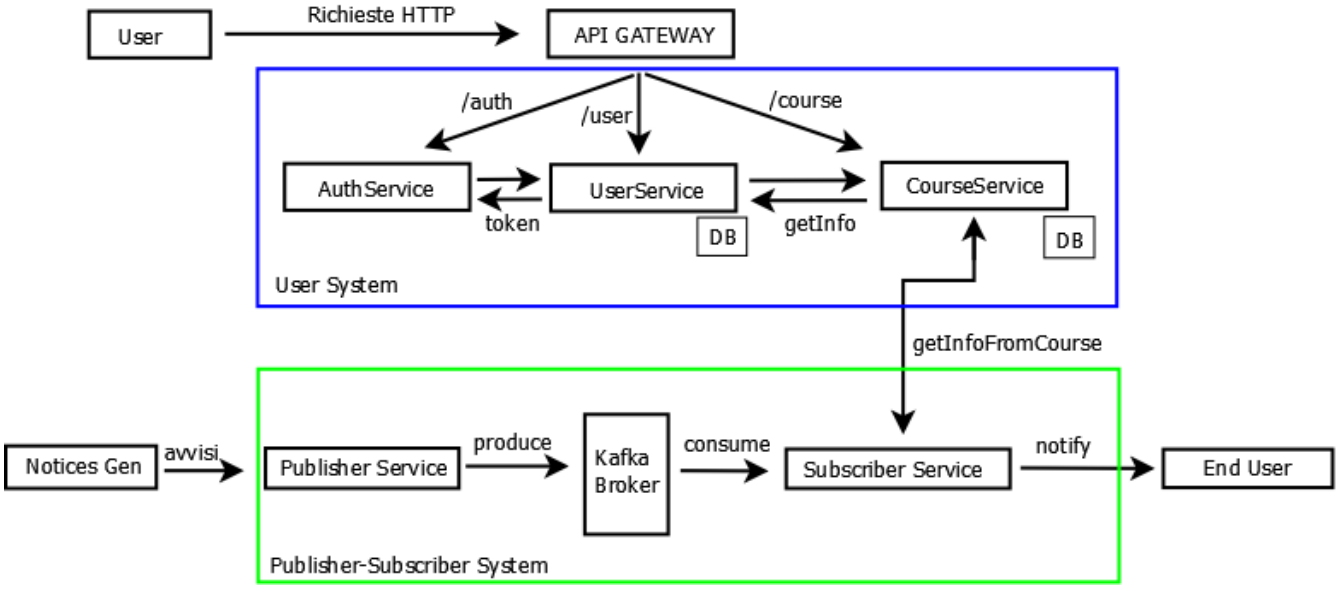
- **Subscriber Service**: ha il compito di consumare i messaggi pubblicati nei topic Kafka relativi ai corsi. Quando un utente è iscritto ad un corso, il Subscriber recupera dal Course Service l'elenco degli studenti associati a quel corso e i rispettivi indirizzi email e garantisce che possano ricevere le notifiche pubblicate sul relativo topic. In questo modo si realizza un sistema di **notifiche push scalabile** che permette agli studenti di restare costantemente aggiornati in tempo reale sulle attività dei corsi a cui sono iscritti.

- **Predictor Service**: rappresenta il componente dedicato al **monitoraggio e all'analisi predittiva** delle performance del sistema. Si integra con **Prometheus** e **cAdvisor** per raccogliere metriche sia di tipo *white-box* (ad esempio il tempo medio di risposta delle API e il numero di richieste gestite) sia di tipo *black-box* (come l'utilizzo di memoria, il tempo di avvio dei container e gli errori di rete).
  Queste metriche vengono elaborate dal Predictor, che utilizza modelli di serie temporali come **ARIMA** per prevedere l'andamento futuro delle prestazioni, in particolare i tempi di risposta delle API.
  Grazie a questo microservizio, EduConnect può anticipare eventuali problemi di performance e migliorare la gestione proattiva delle risorse.

Il traffico in ingresso verso la piattaforma è gestito, in ambiente Kubernetes, da un **Ingress Controller**, che svolge la funzione di instradamento delle richieste verso i microservizi appropriati. In questo contesto, l'Ingress sostituisce l'**API Gateway** utilizzato durante la fase di sviluppo in

Docker Compose, fornendo una gestione nativa e scalabile dell'accesso esterno al cluster.
Ogni microservizio è inoltre dotato di un proprio **database MySQL dedicato**, scelta che consente di mantenere un'architettura **loosely coupled** e di garantire la piena indipendenza dei dati, evitando punti di contatto o dipendenze dirette tra i vari componenti.

Grazie a questa architettura, EduConnect è in grado di garantire un'elevata modularità e di supportare facilmente l'aggiunta di nuovi microservizi o funzionalità, mantenendo al contempo una gestione centralizzata delle metriche e della comunicazione tra i vari componenti.



1. **Diagramma architetturale del software**

3. API Gateway Service

L'**API Gateway** è stato utilizzato in fase di sviluppo e test con Docker Compose come punto di accesso unico al sistema EduConnect, agendo da intermediario tra i clients e i microservizi sottostanti, facilitando la comunizazione e l'inoltro delle richieste in modo efficiente.
È stato implementato in **Python**, utilizzando il framework **Flask** insieme alla libreria **Flask-RESTful** per la gestione delle rotte HTTP. L'API Gateway funge da proxy centralizzato, instradando le richieste verso i microservizi interni (Auth, User, Course) e semplificando l'accesso al sistema in ambienti non orchestrati.

L'**API Gateway** di EduConnect adotta un approccio basato su **routing dinamico**, che permette di esporre in maniera trasparente le API di tutti i microservizi senza doverle dichiarare staticamente una per una. Il meccanismo si fonda su una mappatura (SERVICE_MAP) che associa ciascun microservizio al proprio indirizzo interno; quando una richiesta arriva al Gateway con il pattern */api/<service>/<subpath>,* il sistema individua automaticamente il microservizio corrispondente e ricostruisce la chiamata HTTP verso la destinazione corretta, inoltrando metodo, parametri, corpo e intestazioni originali. In questo modo, l'API Gateway non possiede endpoint propri, ma si

comporta come un **reverse proxy intelligente**, esponendo dinamicamente tutte le API dei microservizi registrati nella mappa.
Questo approccio offre diversi vantaggi: il client interagisce sempre con un unico entry point (/api/...), senza conoscere indirizzi e porte dei singoli servizi; l'aggiunta di un nuovo microservizio richiede unicamente l'aggiornamento della mappa, senza modifiche al codice delle rotte; i meccanismi trasversali, come logging, monitoraggio e validazione dei token JWT, possono essere applicati centralmente nel Gateway. In fase di sviluppo con Docker Compose il Gateway risulta fondamentale per semplificare l'accesso al sistema e mantenere una gestione coerente delle richieste. Ma in ambiente Kubernetes, l'API Gateway viene sostituito dall'**Ingress Controller**, che assume la funzione di instradamento del traffico esterno verso i microservizi interni.

4. Authentication Service

L'**Auth Service** fornisce i meccanismi di **autenticazione** e **autorizzazione** dell'intera piattaforma. Sviluppato in **Python** con **Flask**, espone API per il **login** e per la **validazione dei token** basati su **JWT (JSON Web Token)**. In fase di login, dopo la verifica delle credenziali, il servizio emette un token firmato con una chiave segreta applicativa (gestita come **Secret**), che incorpora le informazioni essenziali sull'utente (ad es. identificativo, scadenza). Gli altri microservizi utilizzano questo token in modalità **bearer** per proteggere le proprie API; quando necessario, invocano l'endpoint di validazione dell'Auth Service per verificarne integrità e validità temporale.

L'adozione di JWT consente di mantenere **stateless** la propagazione dell'identità all'interno del sistema, riducendo la necessità di sessioni centralizzate e semplificando la scalabilità. La responsabilità dell'Auth Service include anche gli aspetti di **sicurezza operativa**: gestione sicura del segreto di firma, definizione della **durata** dei token (expire/refresh) e delle **policy** di accesso. Gli endpoint di salute e le probe di liveness/readiness ne facilitano l'osservabilità e l'integrazione con l'orchestratore.

Dal punto di vista dell'integrazione, l'Auth Service funge da **autorità di fiducia** per l'intero ecosistema: centralizza il rilascio e la verifica delle credenziali, lasciando ai microservizi applicativi la sola responsabilità di controllare la presenza del token. In questo modo si riducono duplicazioni di logica, si migliora la coerenza dei controlli e si semplifica l'evoluzione delle regole di sicurezza nel tempo.

Le principali API esposte includono:

API	**Method**	**Path**	**Description**
Health check	GET	/auth/health	Controllo dello stato del servizio
Verify token	POST	/auth/verify_token	verifica la validità di un token
Authenticate customer	POST	/auth/authenticate	autenticazione dell'utente e la generazione del token JWT

5. User Service

Lo **User Service** è il microservizio dedicato alla gestione del ciclo di vita degli utenti all'interno della piattaforma EduConnect. È sviluppato in **Python** con **Flask** e utilizza la libreria **mysql-connector-python** per la connessione al database MySQL dedicato. Il modello dati prevede campi fondamentali come user_id, name, email e password, con user_id auto-incrementale e gestito dal database, mentre l'indirizzo email è definito come campo univoco per garantire l'identificazione sicura di ciascun utente.

Il servizio espone API REST per la **creazione di nuovi account**, la **consultazione dei profili** tramite ID o email, l'**aggiornamento delle informazioni** e la **cancellazione di un utente**. È inoltre presente un endpoint dedicato all'**autenticazione applicativa**, che riceve email e password in ingresso e valida le credenziali, rappresentando un'integrazione diretta con l'Auth Service e garantendo la coerenza nel processo di login.

Dal punto di vista della sicurezza, lo User Service adotta un sistema di logging per tracciare gli eventi rilevanti (creazioni, modifiche, errori), mentre le password devono essere gestite in forma cifrata (hashing con salt) per prevenire rischi legati alla memorizzazione in chiaro. Gli endpoint sono progettati per restituire risposte standardizzate (200, 201, 400, 404, 409) in modo da facilitare l'integrazione con gli altri microservizi.

Lo User Service si integra inoltre con gli altri componenti della piattaforma, in particolare con il Course Service per associare studenti ai corsi e con l'Auth Service per la validazione delle sessioni. La modularità del microservizio consente di evolverlo facilmente introducendo funzionalità aggiuntive, come statistiche sugli utenti registrati o strumenti di profilazione avanzata, mantenendo l'architettura flessibile e scalabile.

Le API esposte dal microservizio sono:

API	**Method**	**Path**	**Description**
Health check	GET	/users/health	Controllo dello stato del servizio
Autenticazione	POST	/users/auth	Autentica l'utente tramite email e password, richimando il microservizio auth. Risponde 200 se l'esito è positivo o 400 se mancano dei campi
Elenco utenti	GET	/users/getAll	Restituisce l'elenco completo degli utenti registrati nella piattaforma
Dettagli utente per ID	GET	/users/getById/<user_id>	Ritorna i dati dell'utente identificato da ID

Dettagli utente per email	GET	/users/getByEmail?email=<mail>	Ritorna i dati dell'utente per email
Creazione utente	POST	/users/new	Crea un nuovo utente cn name, email, password
Aggiornamento utente	PUT	/users/update	Aggirona un utente esistente.
Eliminazione utente	DELETE	/users/delete/<user_id>	Elimina l'utente per ID.

6. Course Service

Il **Course Service** è il microservizio responsabile della gestione dei corsi e delle iscrizioni degli studenti. Anch'esso sviluppato in **Python** con **Flask** e **mysql-connector-python**, utilizza un database MySQL dedicato per memorizzare informazioni sui corsi (course_id, title, description, duration) e sugli abbinamenti con gli utenti tramite la tabella delle iscrizioni.

Il dominio è rappresentato dalle classi:

- **Course** che rappresenta i corsi erogati e presenta i campi course_id (auto-incrementale), title, description, duration
- **Enrollment** rappresenta l'iscrizione degli utenti ai corsi, trmaite relazione molti-a-molti. Presenta i campi enrollment_id (auto-incrementale), course_id, user_email

Le funzionalità principali includono la **creazione di nuovi corsi**, la **visualizzazione dell'elenco dei corsi disponibili**, il recupero dei **dettagli di un corso specifico** e la gestione delle **iscrizioni degli studenti**. Gli endpoint dedicati alle iscrizioni permettono sia di aggiungere un nuovo studente a un corso (operazione protetta tramite token JWT nell'header della richiesta), sia di recuperare la lista dei corsi a cui un utente è iscritto, sia di ottenere l'elenco delle email degli studenti registrati a un determinato corso. Quest'ultimo meccanismo è particolarmente importante per l'integrazione con il **Subscriber Service**, che lo utilizza per inviare notifiche ai destinatari corretti quando viene pubblicato un avviso.

Il Course Service implementa controlli di validazione sui campi obbligatori (titolo, descrizione, durata) e restituisce codici di errore significativi (400 per richieste incomplete, 401 per token mancanti o invalidi, 404 per corsi inesistenti, 409 per conflitti come iscrizioni duplicate). La gestione delle iscrizioni è progettata per evitare duplicati grazie a controlli applicativi e vincoli di database, assicurando la consistenza dei dati.

Dal punto di vista dell'integrazione, il Course Service riveste un ruolo centrale nell'ecosistema EduConnect: è il punto di riferimento sia per gli studenti, che lo consultano per scoprire e seguire corsi, sia per i microservizi di comunicazione (Publisher e Subscriber), che ne sfruttano le API per associare i messaggi Kafka agli utenti iscritti. Grazie a questa architettura, il servizio mantiene una separazione chiara tra la logica applicativa e la gestione delle notifiche, garantendo un sistema modulare ed estensibile.

Le API esposte dal microservizio sono le seguenti:

API	Method	Path	Description
Health check	GET	/courses/health	Controllo dello stato del servizio
Elenco corsi	GET	/courses/getAll	Restituisce la lista di tutti i corsi disponibili
Dettaglio corso per ID	GET	/courses/getById/<course_id>	Restituisce le info del corso specificato tramite ID.
Creazione corso	POST	/courses/new	Crea un nuovo corso
Iscrizione ad un corso	POST	/courses/new_enroll	Iscrive l'utente autenticato al corso indicato dal course_id inserito nel body. Richiede *Authorization: Bearer JWT* dal quale recuperare le info dell'utente.
Elenco iscrizioni utente	GET	/courses/user_enrollments?user_email=<email>	Restituisce tutte le iscrizioni per l'utente indicato (parametro query user_email).
Email iscritti al corso	GET	/courses/enrollments/emails?course_title=<title>	Restituisce la lista di email degli utenti iscritti al corso.

7. Publisher Service

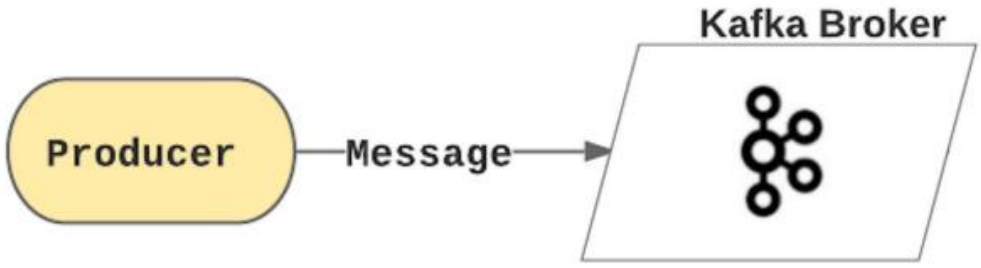
Il **Publisher Service** è il microservizio incaricato di pubblicare su **Apache Kafka** gli avvisi relativi ai corsi (annunci, comunicazioni, aggiornamenti). È sviluppato in **Python** con **Flask** per l'esposizione dell'API REST e utilizza la libreria **confluent-kafka** (basata su *librdkafka*) per l'invio affidabile dei messaggi ai broker. L'API principale riceve dal client i dati dell'avviso (titolo, contenuto, data) e il riferimento al corso; a partire dal titolo del corso, il servizio costruisce in modo deterministico il **nome del topic** (ad es. course-lingua_inglese, normalizzando spazi e maiuscole) così che Publisher e Subscriber condividano una convenzione unica.

Quando l'endpoint viene invocato, il Publisher serializza l'avviso in **JSON** e lo invia in **modo asincrono** al topic Kafka del corso. La pubblicazione è accompagnata da una **callback applicativa** che registra nei log l'esito positivo o eventuali errori di consegna (utile per diagnosi e retry). La configurazione del producer è orientata all'affidabilità (es. gestione degli *acks*, timeout e code interne); a fine richiesta viene effettuato un **flush** per garantire che i messaggi in coda vengano effettivamente inviati al broker prima di rispondere al client.

Dal punto di vista operativo, il servizio è pensato per essere **idempotente** rispetto al canale: se il topic esiste già, viene semplicemente riutilizzato; in caso contrario può essere previsto un meccanismo di creazione lato amministrazione o bootstrap del sistema. L'osservabilità è garantita da log strutturati e può essere arricchita con metriche Prometheus (numero di messaggi pubblicati, latenza di publish, error rate) per integrare dashboard e alerting.

Le API esposte sono le seguenti:

API	**Method**	**Path**	**Description**
Health check	GET	/publisher/health	Controllo dello stato del servizio
Pubblicazione messaggio	POST	/publisher/publish_notice	Pubblica il messaggio sul topic kafka.



8. Subscriber Service

Il **Subscriber Service** è il microservizio responsabile della ricezione e gestione dei messaggi pubblicati sui topic Kafka relativi ai corsi. È stato sviluppato in **Python**, utilizzando **Flask** per esporre endpoint di controllo e la libreria **confluent-kafka** per implementare la logica di consumo dai broker Kafka.

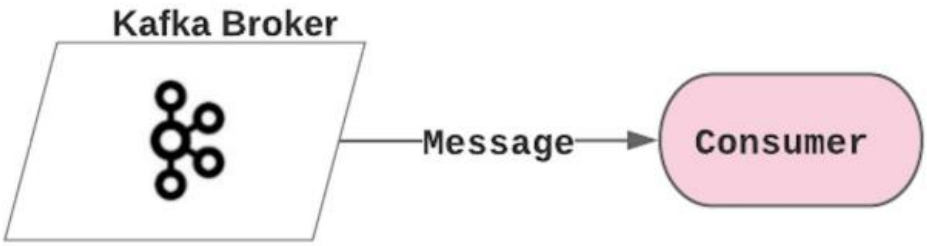
All'avvio, il servizio esamina la lista dei topic associati ai corsi (TOPIC_LIST) e, per ciascuno di essi, ne verifica l'esistenza nel cluster Kafka tramite l'**AdminClient**. Qualora un topic non fosse ancora presente, viene creato dinamicamente con un numero predefinito di partizioni e un fattore di replica minimo, così da garantire la disponibilità del canale di comunicazione. Una volta che il topic risulta disponibile, viene attivato un **consumer dedicato**, eseguito in un thread separato.

Ogni consumer si iscrive al proprio topic Kafka e rimane in ascolto continuo dei messaggi in arrivo. I messaggi, pubblicati in precedenza dal **Publisher Service**, vengono estratti, decodificati e interpretati come avvisi destinati agli studenti iscritti al corso corrispondente. Per identificare gli utenti destinatari, il Subscriber interroga il **Course Service** attraverso un'apposita API che restituisce l'elenco delle email degli studenti iscritti a quel corso. Una volta ottenuti i destinatari, il servizio si occupa di inviare loro la notifica: nella versione attuale la notifica è simulata tramite logging, ma l'architettura prevede facilmente l'estensione futura con l'invio di email reali, la memorizzazione in un database o l'utilizzo di WebSocket per notifiche in tempo reale.

Il Subscriber Service si basa su un modello di consumo asincrono e concorrente: ogni topic ha il proprio thread consumer, il che consente di gestire in parallelo più corsi e più flussi di messaggi. Grazie all'uso dei **consumer group**, è possibile scalare orizzontalmente il microservizio distribuendo il carico di elaborazione tra più istanze, ciascuna delle quali consuma un sottoinsieme delle partizioni dei topic. In questo modo il sistema rimane flessibile e scalabile al crescere del numero di corsi e degli avvisi generati.

Dal punto di vista dell'affidabilità, il servizio implementa meccanismi di gestione degli errori sia nella fase di consumo dei messaggi (log degli errori Kafka) sia nella fase di notifica agli utenti (retry e gestione delle risposte non valide da parte del Course Service). Inoltre, ogni consumer chiude correttamente le proprie connessioni ai broker Kafka in caso di interruzione, garantendo il rilascio ordinato delle risorse.

In sintesi, il **Subscriber Service** rappresenta il ponte tra la pubblicazione asincrona di eventi su Kafka e la consegna di notifiche agli utenti finali, integrandosi strettamente con il Course Service per recuperare la lista degli iscritti e mantenendo un'architettura aperta a diverse modalità future di distribuzione degli avvisi.



9. Predictor Service

Il **Predictor Service** è il microservizio dedicato al **monitoraggio** e all'**analisi predittiva** delle prestazioni di EduConnect. È sviluppato in **Python** con **Flask** e utilizza la libreria **statsmodels** per i modelli di serie temporali (**ARIMA**), mentre i dati di osservabilità sono raccolti da **Prometheus**. Il servizio combina metriche **white-box** (esposte dai microservizi, ad esempio: tempo medio di risposta, throughput) e metriche **black-box** (raccolte a livello container, ad esempio: memoria utilizzata, tempo di avvio, errori di rete).

In ambiente **Docker Compose**, le metriche container arrivano da **cAdvisor** (etichettate con container_label_com_docker_compose_service), mentre in **Kubernetes** è Prometheus a scrappare il kubelet all'endpoint /metrics/cadvisor, ottenendo label coerenti con l'orchestratore (namespace, pod, container). Il Predictor adotta query "**dual mode**" che funzionano in entrambi i contesti (Docker o K8s), selezionando automaticamente la sorgente disponibile. Le funzionalità principali esposte via API includono: recupero del **tempo medio di risposta** in una finestra mobile, calcolo del **numero totale di richieste** in un intervallo, lettura della **memoria** usata da un container/pod, del **tempo di start** e degli **errori di rete**. Per le previsioni, il servizio costruisce la serie temporale dei tempi medi di risposta (minuto per minuto) e applica un modello **ARIMA** per stimare l'andamento nei prossimi minuti, restituendo una **forecast** utile ad anticipare possibili degradazioni.

Il modello **ARIMA (AutoRegressive Integrated Moving Average)** è una delle tecniche più diffuse per l'analisi e la previsione di serie temporali. Combina tre componenti principali: la parte autoregressiva (AR), che utilizza la correlazione tra i valori attuali e quelli passati della serie; la parte integrata (I), che applica differenziazioni successive per rendere la serie stazionaria; e la parte a media mobile (MA), che cattura la dipendenza tra i valori osservati e gli errori residui di previsioni precedenti. L'uso di ARIMA consente al Predictor Service di stimare in anticipo variazioni nei tempi di risposta, fornendo così un supporto utile al rilevamento proattivo di possibili colli di bottiglia o anomalie.

Il design privilegia semplicità e portabilità: endpoint REST chiari, logging strutturato, gestione degli errori (nessun dato, parametri non validi, problemi di parsing), e indipendenza dall'ambiente di esecuzione grazie a selettori PromQL compatibili. In prospettiva, il servizio può essere esteso con: alerting su soglie predittive (integrazione Prometheus/Alertmanager), e dashboard dedicate (Grafana) per visualizzare storici e previsioni.

API	**Method**	**Path**	**Description**
Health check	GET	/predictor/health	Controllo dello stato del servizio.
Tempo medio di risposta	GET	/predictor/avg_response_time	Recupero del tempo medi di risposta.
Totale richieste	GET	/predictor/total_requests	Recupero numero di richieste in range temporare specificato.
Memoria usata	GET	/predictor/memory_usage	Recupero della memoria usata dal container/pod
Tempo di avvio	GET	/predictor/start_time	Recupero del tempo di avvio di un container/pod in formato UTC.
Totale errori di rete	GET	/predictor/network_errors	Recupero del numero di errori di rete per il servizio indicato.
Predittore ARIMA tempo medio di risposta	GET	/predictor/predict_response_time	Predizione tramite modello ARIMa sul tmepo medio di risposta per i prossimi x minuti specificati nella richiesta.

10. Databases MySql

Nel sistema EduConnect i microservizi **User Service** e **Course Service** utilizzano un database relazionale per la gestione dei dati. Ogni microservizio ha il proprio database **MySQL**, che viene avviato come container dedicato in Docker Compose e come Pod separato in Kubernetes.

Lo User Service mantiene nel proprio database le informazioni sugli utenti registrati (identificativo, nome, email e password), garantendo vincoli di unicità sull'email e fornendo le basi per le operazioni di autenticazione e gestione profilo.
Il Course Service gestisce, invece, i dati relativi ai corsi (titolo, descrizione, durata) e le iscrizioni degli studenti, modellate tramite una tabella dedicata che implementa la relazione molti-a-molti tra utenti e corsi.

All'interno della sottocartella db di ciascun microservizio *<microservizio>/db/* è presente un file init.sql, che viene eseguito automaticamente all'avvio per inizializzare il database. Questo script si occupa di creare il database e le relative tabelle se non ancora esistenti, e di inserire alcune entry di esempio utili ai test iniziali. In questo modo, l'applicazione può essere avviata rapidamente con un dataset coerente senza richiedere configurazioni manuali.

Per il database **mysql-userdb** viene creata una tabella **users** con i campi:

- *user_id:* di tipo intero, Primary Key ed auto-increment;
- *name:* di tipo stringa
- *email:* di tipo stringa;
- *password:* di tipo stringa.

Lo script permette l'inserimento inziale di due utenti "Mario Rossi" e "Luca Bianchi" con password già hashate.

In modo analogo, il microservizio Course Service inizializza il proprio schema, **mysql-coursedb**, creando due cartelle (se non esistenti).
La tabella **courses** presenta i campi:

- *course_id*: di tipo intero, Primary key ed auto-increment;
- *title*: di tipo stringa;
- *description*: di tipo stringa;
- *duration*: di tipo stringa.

La tabella **enrollments** indica le iscrizioni degli utenti ai corsi e presenta i campi:

- *id:* di tipo intero, Primary key ed auto-increment;
- *course_id:* di tipo intero, con Foreign Key al campo course_id della tabella courses;
- *user_email:* di tipo stringa.

Questo script permette l'inserimento dei corsi "Python base", "Data Science", "Machine Learning", "Lingua inglese", "Lingua francese" e delle iscrizioni dei due utenti presenti nella tabella users a due corsi diversi.

L’uso di database separati garantisce **indipendenza** tra i microservizi e rispetta i principi dell’architettura a microservizi, consentendo di evolvere lo schema dati di un servizio senza impattare sugli altri.

11. Prometheus e cAdvisor per il monitoraggio delle metriche

Il sistema EduConnect integra un’infrastruttura di **monitoraggio** basata su **Prometheus** e **cAdvisor**, con l’obiettivo di raccogliere e analizzare metriche a livello sia applicativo che di container.

In ambiente **Docker Compose**, Prometheus è configurato per eseguire lo scraping direttamente dai microservizi, che espongono metriche white-box come il numero di richieste e i tempi di risposta, e da **cAdvisor**, che fornisce metriche black-box relative all’utilizzo delle risorse da parte dei container (memoria, CPU, errori di rete, tempo di avvio). In questo contesto, cAdvisor è eseguito come un container dedicato e Prometheus lo interroga tramite endpoint HTTP standard, etichettando i dati con il nome logico dei container definiti in docker-compose.yml.

In **Kubernetes**, invece, la raccolta avviene in modo più nativo: Prometheus sfrutta la service discovery dell’orchestratore e interroga direttamente i kubelet all’endpoint /metrics/cadvisor. In questo modo, le metriche vengono arricchite con label aggiuntive come namespace, pod e container, consentendo una visibilità più granulare e dettagliata sul comportamento dei microservizi in esecuzione nel cluster. Grazie a questa modalità, è possibile distinguere facilmente i consumi delle risorse per singolo Pod o ReplicaSet, anche quando i nomi dei container non sono più statici come in Docker Compose.

La combinazione tra **Prometheus** e **cAdvisor** garantisce quindi una copertura completa: da un lato, metriche white-box fornite direttamente dalle applicazioni, dall’altro lato metriche black-box che permettono di valutare l’uso delle risorse e la stabilità dell’infrastruttura. Il **Predictor Service** utilizza queste metriche per costruire serie temporali e applicare modelli predittivi (ad esempio ARIMA), fornendo uno strumento proattivo per anticipare possibili anomalie o degradazioni delle prestazioni.

12. Distribuzione dell’applicazione

La distribuzione di EduConnect è stata progettata per essere flessibile e portabile, supportando sia l’esecuzione in ambiente **Docker Compose** (per sviluppo e test locali) sia in ambiente **Kubernetes** (per scenari di orchestrazione avanzata e produzione).

In fase di sviluppo, l’applicazione viene eseguita tramite **Docker Compose**, che sfrutta un file docker-compose.yaml nella root del progetto per coordinare l’avvio di tutti i microservizi. Ogni servizio dispone di un proprio **Dockerfile**, che descrive i passaggi necessari alla containerizzazione (installazione delle dipendenze, configurazione dell’ambiente ed esecuzione dell’applicazione). Docker Compose garantisce che tutti i container vengano eseguiti in un’unica rete virtuale, facilitando la comunicazione tra i microservizi e semplificando le operazioni di build e avvio

tramite un singolo comando. In questo scenario, l'**API Gateway** svolge la funzione di entry point centralizzato, ricevendo e instradando le richieste verso i vari componenti.

In produzione o in ambienti di laboratorio più complessi, l'applicazione viene distribuita in **Kubernetes**. Per ogni microservizio è stata predisposta una cartella dedicata (/k8s/<microservizio>/) contenente i manifest YAML necessari, che includono tipicamente:

- **Deployment** per la gestione dei Pod e delle repliche,
- **Service** per l'esposizione e il bilanciamento del carico interno,
- **Secret** e **ConfigMap** per la gestione sicura delle configurazioni,
- eventuali **PersistentVolumeClaim** per i microservizi che necessitano di persistenza.

Il traffico esterno è gestito da un **Ingress Controller**, che sostituisce l'API Gateway utilizzato in Docker Compose e fornisce un accesso nativo e scalabile ai microservizi del cluster. Tutte le risorse Kubernetes sono raggruppate all'interno di un **namespace dedicato (educonnect)**, che consente di isolare l'applicazione e semplificarne la gestione.

La distribuzione su Kubernetes avviene in modo semplificato ma efficace. Dopo aver avviato **Minikube** ed eseguito la creazione del namespace dedicato educonnect, l'intero sistema viene deployato tramite un unico comando: "*kubectl apply -f k8s/ --recursive*".
Grazie a questa modalità, vengono letti tutti i manifest YAML presenti nelle sottocartelle, che definiscono Deployment, Service, Secret e ConfigMap per i vari microservizi. Questa procedura consente di ricreare l'ambiente completo in maniera riproducibile e coerente, rendendo semplice l'avvio o il ripristino dell'applicazione. In futuro, il processo potrebbe essere esteso con pipeline di Continuous Integration e Continuous Deployment (CI/CD), automatizzando le fasi di build, test e rilascio.

Uno degli obiettivi principali dell'adozione di Kubernetes è garantire **scalabilità** e **resilienza** all'applicazione. Grazie ai manifest di tipo **Deployment**, ogni microservizio viene eseguito in Pod che possono essere facilmente replicati per bilanciare il carico di lavoro. In caso di malfunzionamento o crash, Kubernetes rileva l'errore tramite le probe di liveness/readiness e ricrea automaticamente i Pod, assicurando la continuità del servizio.
Anche Kafka contribuisce alla scalabilità: la suddivisione dei topic in partizioni permette di distribuire i messaggi tra più consumer, mentre i **consumer group** consentono di bilanciare il carico su più istanze del Subscriber Service. In questo modo, EduConnect è progettato per adattarsi a un numero crescente di utenti e corsi senza perdere affidabilità.

Questa duplice strategia di distribuzione rende EduConnect facilmente eseguibile in ambienti di sviluppo leggeri, ma al tempo stesso pronto per l'orchestrazione su larga scala tipica dei cluster Kubernetes, mantenendo coerenza tra le due modalità e riducendo i tempi di setup.

13. Sicurezza dell'architettura

La sicurezza di EduConnect si basa principalmente sull'uso di **JWT (JSON Web Token)**, generati dall'Auth Service e validati dagli altri microservizi prima di eseguire operazioni sensibili. Questo approccio consente di mantenere la piattaforma **stateless**, evitando la necessità di sessioni

centralizzate e semplificando la gestione dell'autenticazione. Le informazioni contenute nel token permettono di identificare l'utente in modo sicuro e di controllare l'accesso alle API.
A livello infrastrutturale, in Kubernetes le credenziali e le chiavi segrete sono gestite tramite **Secret**, che vengono montati nei container in fase di avvio. Ciò garantisce una separazione tra il codice applicativo e i dati sensibili, riducendo i rischi di esposizione accidentale. In prospettiva, la sicurezza potrebbe essere rafforzata introducendo ruoli differenziati per gli utenti (studenti, docenti, amministratori), proteggendo le API tramite **Ingress TLS/HTTPS** e integrando strumenti di sicurezza nativi dell'orchestratore.