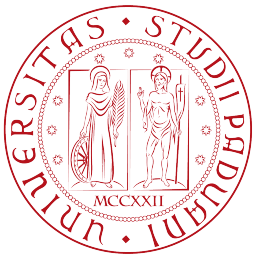

Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea in
Informatica

Tesi di Laurea

Sperimentazione di Apache Kafka per l'integrazione funzionale di un'applicazione aziendale

Experimenting with Apache Kafka for the Integration of an Enterprise Application

Relatore:
Prof. Tullio Vardanega

Laureando:
Andrea Dorigo
1170610

Anno Accademico 2020/2021

Indice

1 Contesto aziendale **5**

1.1 Azienda ospitante 5

1.2 Servizi offerti dall'azienda 6

1.3 Processi interni e strumenti organizzativi 7

1.3.1 Processi interni 7

1.3.2 Strumenti organizzativi 8

1.4 Ambiente di lavoro 9

1.4.1 Sistemi distribuiti 10

1.4.2 *Service Oriented Architecture*$_g$ 11

1.4.3 *Messaging pattern* 12

1.4.4 *Enterprise Service Bus* 13

1.5 Dominio applicativo 13

1.5.1 EAI$_a$ e *Middleware* 13

1.5.2 *Container* e *Virtual Machine* 15

1.5.3 L'introduzione di Apache Kafka 18

1.6 L'innovazione all'interno dell'azienda 19

2 Apache Kafka nell'Integrazione Aziendale **20**

2.1 Obiettivi aziendali 20

2.1.1 Kafka come Middleware 22

2.2 Motivazioni e obiettivi personali 23

2.2.1 Scelta del percorso 23

2.2.2 Obiettivi personali 23

2.3 Il percorso di Stage 23

2.3.1 Obiettivi dello *stage* 23

2.3.2 Prodotti attesi 24

2.3.3 Contenuti formativi previsti 24
2.3.4 Interazione tra studente e referenti aziendali 25
2.3.5 Pianificazione del lavoro 25

3 Apache Kafka in caso d'uso simulato **29**

4 Obiettivi soddisfatti e bilancio formativo **30**
4.1 Obiettivi soddisfatti dallo stage 30
4.2 Maturazione professionale acquisita 30
4.3 Distanza tra le competenze necessarie e quelle acquisite nel corso di studi 30

Elenco delle tabelle

2.1 Pianificazione settimanale dello *stage* 27

Elenco delle figure

1.1 Attuali sedi di Sync Lab 6
1.2 Processi interni di cui ho avuto esperienza 7
1.3 *Kanban Board* del progetto di *stage* 8
1.4 Esempio di un'attività del processo di Formazione 9
1.5 Illustrazione di un sistema distribuito 10
1.6 SOA_a applicata con un ESB_a 11
1.7 Schema di un *messaging design pattern* 12
1.8 Illustrazione esemplificativa di un ESB_a 13
1.9 Concetti principali del EAI_a 14
1.10 Dalle classiche soluzioni monolitiche ai moderni sistemi a microservizi 15
1.11 Differenti implementazioni legate alle VM_a e *container* 16
1.12 Frammento di codice che espone un esempio di ambiente *multi-container* con docker-compose 17
1.13 Schema di un topic contenente diversi eventi, diviso in partizioni (P), con molteplici *producer* 18

2.1 Illustrazione di un sistema basato sul $P2P_a$ 21
2.2 Illustrazione di un sistema basato sulla EDA_a 21
2.3 Illustrazione di Apache Kafka in un caso d'uso esemplificativo 22
2.4 Illustrazione di un sistema a servizi con Kafka 24
2.5 Diagramma di Gantt del piano di lavoro 28

Capitolo 1

Contesto aziendale

1.1 Azienda ospitante

Sync Lab s.r.l.[1] è un'azienda di produzione software, ICT_a[2] e consulenze informatiche nata nel 2002 a Napoli. L'azienda al suo stato attuale presenta un organico aziendale composto da più di 200 risorse, con un fatturato annuo di 12 milioni, una solida base finanziaria e una diffusione sul territorio a livello nazionale. Sync Lab possiede delle significative fette di mercato riguardanti lo sviluppo di prodotti nel settore mobile, videosorveglianza e sicurezza delle strutture informatiche aziendali.

L'azienda ha acquisito numerose certificazioni ISO LL-C per attestare la qualità dei servizi forniti. La certificazione ISO-9001 attesta la gestione della qualità, ISO-14001 la gestione dell'ambiente, ISO-27001 la sicurezza dei sistemi di gestione dati e ISO-45001 la sicurezza nel luogo di lavoro.

Tra i clienti di Sync Lab vi sono ditte a livello nazionale di grandi dimensioni e ampio organico, come Intesa San Paolo, TIM, Vodafone, Enel e Trenitalia che necessitano prodotti di un'elevata sicurezza e adatti al considerevole flusso di dati aziendale.

Sync Lab ha fornito prodotti e consulenze a più di 150 clienti, distribuiti tra clienti diretti e finali, e attualmente possiede cinque sedi (figura 1.1): Napoli, Roma, Milano, Padova e Verona.

L'azienda è suddivisa in molteplici settori dislocati nelle diverse sedi; l'esperienza personale mi ha portato a conoscere il settore dell'*Enterprise Architecture Integration* e del *Tecnical Professional Services Padova*.

[1] *Sync Lab.* URL: https://www.synclab.it/.
[2] *Information and Communication Technologies*

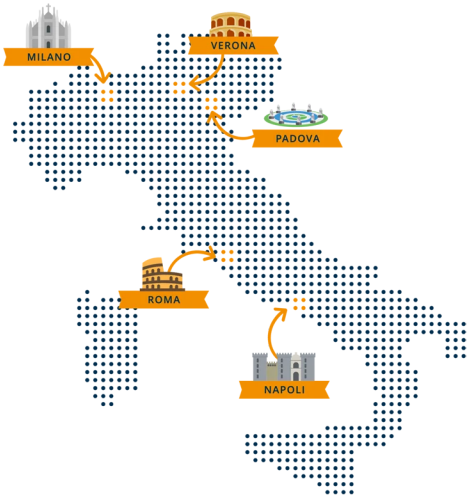

Figura 1.1: Attuali sedi di Sync Lab
Fonte: https://www.synclab.it/

1.2 Servizi offerti dall'azienda

Per comprendere appropriatamente il contesto che ha portato alla nascita del progetto di *stage* è bene conoscere la tipologia di servizi e prodotti che l'azienda offre ai propri clienti. Sync Lab offre per i propri clienti numerosi servizi, tra cui:

- Valutazione e controllo progetti
  - *Planning e project management*; definizione di *Milestone* e *team* di progetto.
  - Valutazione di impatto e *risk analysis*; monitoraggio e *benchmarking*.
  - Valutazione e controllo di progetti *software* attraverso l'utilizzo di metriche e modelli economici di stima e previsione.
- Sistemi distribuiti di *Enterprise*
  - Progettazione e realizzazione di sistemi distribuiti *Enterprise* in architettura J2EE$_a$[3], EJB$_a$[4], COBRA$_a$[5] e *Web Services*.
  - Progettazione e realizzazione di sistemi basati su MOM$_a$[6] e JMS$_a$[7].
- Tecnologie *Object Oriented*
  - Applicazione delle tecnologie O-O$_a$ all'analisi e progettazione di *software* applicativo e di sistema e nella definizione di architetture distribuite *enterprise*.

[3] *Java 2 Platform Enterprise Edition*
[4] *Enterprise Java Bean*
[5] *Common Object Request Broker Architecture*
[6] *Message Oriented Middleware*
[7] *Java Message Service*

– Utilizzo di metodologie O-O$_a$ per progettazione di applicazioni e processi e UML$_a$, con supporto di strumenti di *modeling*, applicazione e definizione di *Design Pattern*.

1.3 Processi interni e strumenti organizzativi

L'azienda adotta dei processi interni per delineare l'avanzamento di un progetto. Sync Lab utilizza una strategia *Agile* per gestire i progetti in modo da consentire un'evoluzione e adattabilità in base alle richieste del cliente e fornire soluzioni *ad-hoc*.

1.3.1 Processi interni

Durante il percorso di *stage* sono stato coinvolto nei processi di Formazione, Progettazione architetturale, Sviluppo, Verifica e Collaudo; i processi di Manutenzione ed Evoluzione sono stati solamente accennati in quanto al di fuori dello scopo del percorso. Questi processi, nella mia esperienza personale, non sono stati delineati rigorosamente: ciò ha lo scopo di garantire libertà e flessibilità allo stagista e di conseguenza all'intero progetto.

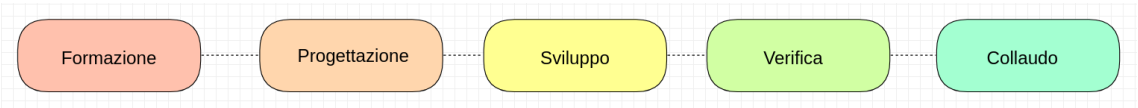

Figura 1.2: Processi interni di cui ho avuto esperienza
Fonte: elaborazione personale

Ogni processo è suddiviso in attività modulari, per rendere l'avanzamento efficace e quantificabile (in figura 1.2 sono illustrati i processi relativi allo *stage* in ordine temporale da sinistra verso destra).

Per il processo di Formazione, Sync Lab fornisce materiale sotto forma di corsi *online* tramite le piattaforme *Coursera*[8] e *Udemy*[9] e diapositive aziendali che illustrano i concetti chiave del settore EAI$_a$.

Il processo di Sviluppo della mia esperienza personale è risultato abbastanza libero per quanto riguarda le tecnologie e i *software* utilizzati, purché le scelte fossero adeguatamente motivate e adeguate.

Il processo di Progettazione architetturale è uno dei più complessi, che necessita di una buona dose di esperienza nell'ambito. Per affrontare questo processo, oltre ad approfondire le mie conoscenze riguardo i diversi *design pattern* e *software architecture style*, l'azienda mi ha accompagnato

[8] *Coursera | Build Skills with Online Courses.* URL: https://www.coursera.org.
[9] *Online Courses - Learn Anything, On Your Schedule | Udemy.* URL: https://www.udemy.com.

e supportato nella progettazione stessa, con relative motivazioni. Il tutor aziendale Francesco Sanges e il responsabile del settore EAI$_a$ Salvatore Dore sono stati di fondamentale aiuto in questo processo.

Il processo di Verifica è stato eseguito dal tutor aziendale e dal Responsabile del settore EAI$_a$ a scadenza settimanale, tramite colloqui *online* o resoconti sulla *online board* di riferimento.

Il processo di Collaudo è avvenuto tramite una presentazione *online* e dimostrazione *live* del prodotto sviluppato all'intera azienda.

1.3.2 Strumenti organizzativi

L'organizzazione efficiente di un progetto è garantita dall'utilizzo dei vari strumenti a supporto, quali *Kanban Board* (come *Click Up*[10] per la gestione di progetto e *Notion*[11] per le prenotazioni della postazione di lavoro in sede), *chat* (come *Google Chat*[12]) per i confronti rapidi con gli altri membri interni al progetto ed e-mail per le comunicazioni con componenti esterni al progetto.

Lo strumento più utilizzato in ambito organizzativo durante il mio percorso è la *Kanban Board* di *Click Up*, che ha permesso la gestione, il confronto, la quantificazione e la verifica del progresso. La figura seguente illustra uno *screenshot* che raffigura lo stato dell'avanzamento.

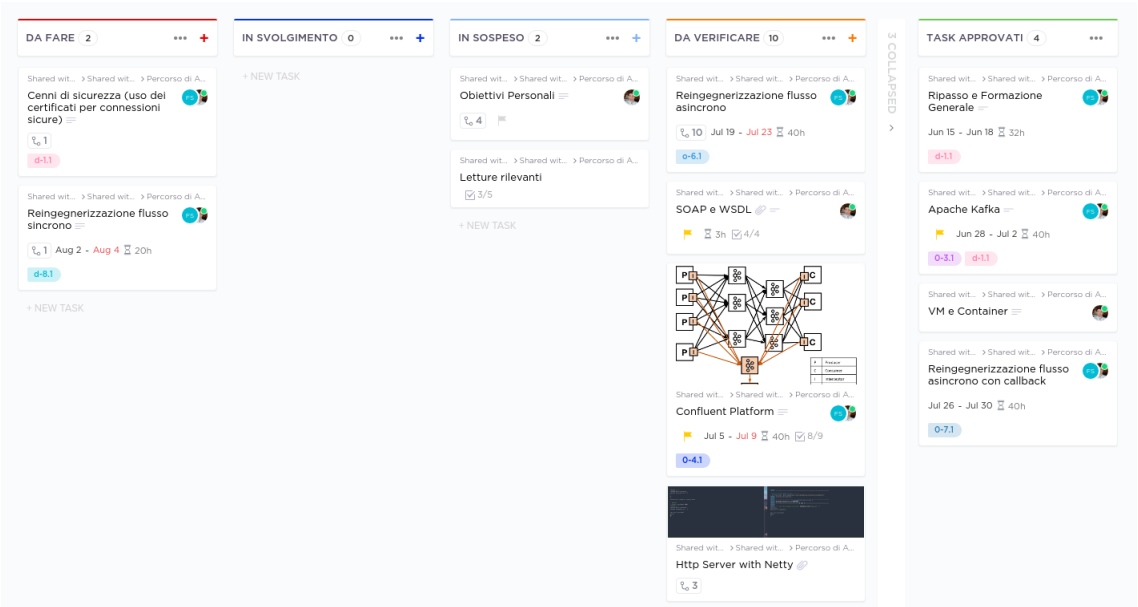

Figura 1.3: *Kanban Board* del progetto di *stage*
Fonte: elaborazione personale

[10] *ClickUp™ | One app to replace them all.* URL: https://clickup.com.
[11] *Notion - The all-in-one workspace for your notes, tasks, wikis, and databases.* URL: https://www.notion.os.
[12] *Google Chat.* URL: https://chat.google.com.

Le attività (*task*) vengono inizialmente create nella colonna "DA FARE" dal tutor aziendale o dal sottoscritto, ove ritenuto opportuno. Per dimostrare l'avanzamento il *task* si sposta verso destra a seconda dello stato raggiunto; lo stagista ha la responsabilità del cambiamento di stato fino alla colonna "DA VERIFICARE", dopodichè è compito del tutor aziendale la verifica e lo spostamento del *task* in "TASK APPROVATI", che comporta l'approvazione finale e conclusione dell'attività.

Per tenere traccia del lavoro svolto riguardante una specifica attività ho utilizzato le *card* messe a disposizione dalla piattaforma, che mi hanno consentito di delineare precisamente la pianificazione e descrizione dell'avanzamento in dettaglio del singolo *task*.

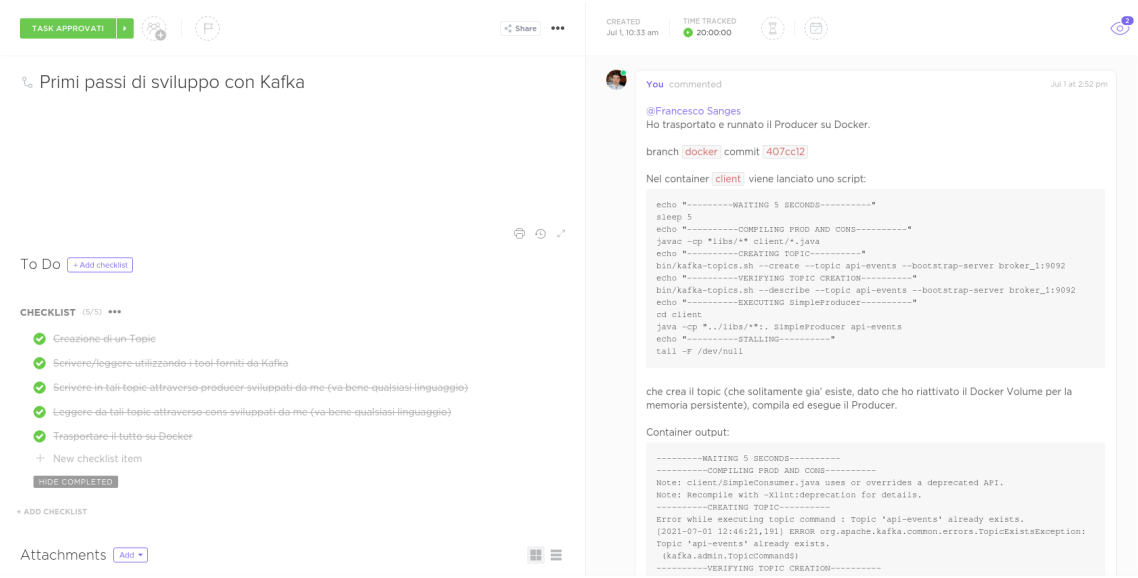

Figura 1.4: Esempio di un'attività del processo di Formazione
Fonte: elaborazione personale

Questa *card* contiene una casella di testo per inserire una descrizione e appunti utili ove sia richiesto, una *checklist* approfondita, e una colonna che mantiene uno storico dei commenti; quest'ultima colonna non solo permette a me di mantenere un'importante resoconto sul lavoro svolto, ma consente anche al tutor aziendale e esperti del settore di quantificare il progresso e di fornire un aiuto rapido e contestuale.

1.4 Ambiente di lavoro

L'ambiente di lavoro di cui ho avuto esperienza risulta libero e flessibile, tanto nel campo organizzativo quanto in quello tecnologico.

Lo sviluppo del prodotto *software* nell'ambito del EAI$_a$ è fortemente consigliato essere indipendente dal linguaggio di programmazione, dagli strumenti utilizzati per l'esecuzione e sviluppo, e ove possibile eanche dal Sistema Operativo su cui esso esegue. A tal scopo si utilizzano strumenti quali *Virtual Machine* e *Container*: essi non solo garantiscono l'indipendenza dal Sistema Operativo in uso, ma simulano efficacemente il caso d'uso reale in cui gli eseguibili sono dislocati in più dispositivi come spesso accade per il cliente finale. Il percorso formativo ha visto l'apprendimento di entrambe le tecnologie tramite l'utilizzo dei *software Virtual Box*[13] e *Docker*[14], ma infine solo quest'ultima è stata utilizzata durante il progetto.

Nonostante vi sia questa libertà tecnologica riguardo la scelta dei software e linguaggi, è necessario tenere in considerazioni i concetti principali vincolanti del settore, come quelli di sistema distribuito, SOA$_a$[15], *messaging pattern* e ESB$_a$[16].

1.4.1 Sistemi distribuiti

Un punto fondamentale dunque che caratterizza l'ambiente di lavoro, è quello del sistema distribuito, di cui l'azienda ha un forte interesse per soffisfare le necessità di integrazione dei suoi clienti.

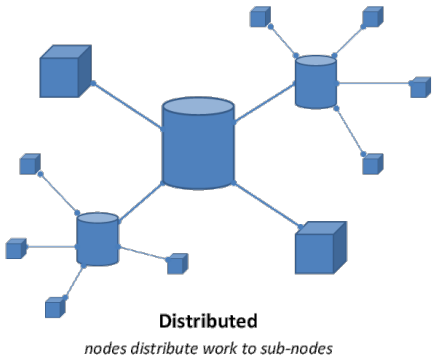

Figura 1.5: Illustrazione di un sistema distribuito
Fonte: https://www.delphitools.info/DWSH/

Un sistema distribuito (figura 1.5) **è una collezione di componenti indipendenti** (spesso collocati in macchine differenti) **che condividono dei messaggi fra di loro per raggiungere un obiettivo comune.** Una'architettura software basata su di un sistema distribuito necessita di una rete che connette tutti i singoli componenti (macchine, *hardware* o *software*), cosicchè sia possibile lo scambio dei messaggi.

[13] *Oracle VM Virtual Box*. URL: https://www.virtualbox.org/.
[14] *Empowering App Development for Developers | Docker*. URL: https://www.docker.com/.
[15] *Service Oriented Architecture*$_g$
[16] *Enterprise Service Bus*

Il vantaggio principale di questo tipo di sistemi è la relativamente economica scalabilità orizzontale dei grandi sistemi: per migliorare la *performance* del sistema è sufficiente aggiungere delle nuove macchine, meno costoso che richiedere *hardware* sempre più potente.

Un secondo vantaggio di fondamentale importanza è la tolleranza ai guasti (*fault tolerance*); rispetto ad un sistema centralizzato, ove nel caso di guasto nella macchina centrale si interrompe l'intero sistema, nel caso di un sistema distribuito esso continua a funzionare e fornire il servizio ove vi sia un guasto in un numero (limitato) di macchine.

Un ulteriore punto a favore dei sistemi distribuiti è la bassa latenza: il *service client* si connette al nodo a se più geograficamente vicino, riducendo il tempo di risposta dei sistemi che coprono vasti territori.

Questo tipo di sistemi risulta molto favorevole per i clienti di grande dimensione di Sync Lab, per cui i vantaggi superano lo svantaggio del costo iniziale più alto dato dall'installazione del sistema.

1.4.2 *Service Oriented Architecture*$_g$

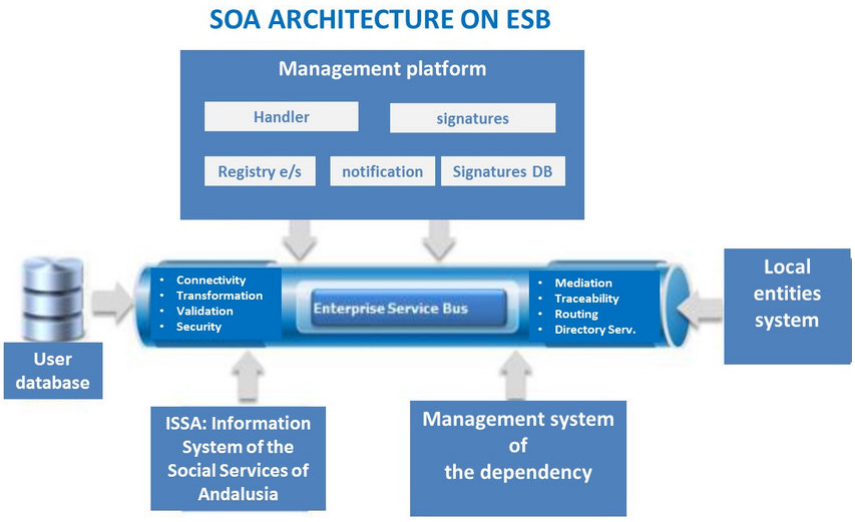

Figura 1.6: SOA$_a$ applicata con un ESB$_a$

Fonte: https://www.researchgate.net/figure/Fig-9-Service-Oriented-Architecture-SOA-Service-Oriented-Architecture-SOA-Service_fig7_330599363

La *Service Oriented Architecture*$_g$ è una tipologia di *software architecture* spesso utilizzata in ambito EAI$_a$. Essa definisce un modo per rendere i componenti di un architettura *software* riutilizzabili, tramite una decomposizione di un sistema in parti più piccole che comunicano tramite interfacce di servizio che possono essere classificate come sotto-sistemi. **Ogni servizio in una SOA$_a$ contiene il codice e le integrazioni dei dati necessari per eseguire una funzione**

aziendale completa e discreta. Le interfacce di servizio comportano un *loose coupling*, il che significa che possono essere richiamate con poca o nessuna conoscenza della sottostante modalità di implementazione dell'integrazione. I servizi sono esposti utilizzando protocolli di rete standard, come SOAP_a[17]/HTTP_a o JSON_a[18]/HTTP_a, per inviare richieste di lettura o modifica dei dati. I servizi sono pubblicati per consentire agli sviluppatori di trovarli rapidamente e riutilizzarli per assemblare nuove applicazioni in modo modulare.

Questo tipo di architettura consente a Sync Lab di creare sistemi basati sui microservizi, derivati dalla SOA_a, per dare modularità, scalabilità, flessibilità e facilità di manutenzione o aggiornamento ai propri prodotti.

1.4.3 *Messaging pattern*

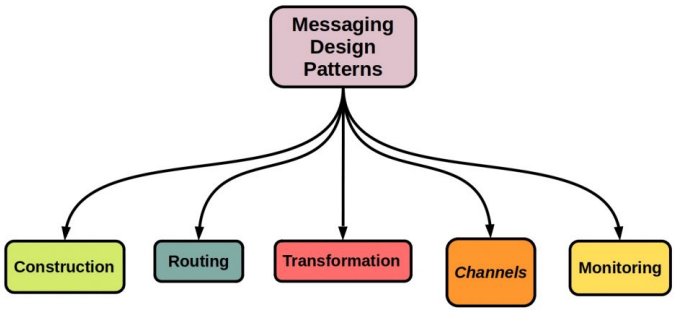

Figura 1.7: Schema di un *messaging design pattern*
Fonte: https://starship-knowledge.com/messaging-patterns

I *messaging pattern* sono alla base di ogni sistema di integrazione. Il *design pattern* si occupa dello scambio di messaggi, come si può intuire dal nome. Un messaggio è un pacchetto di dati atomico che può essere trasmesso attraverso un canale in modalità asincrona. Un canale è un condotto virtuale che connette un servizio che invia i dati ad uno che li riceve.

Nella maggior parte dei sistemi di integrazione, il dato potrebbe necessitare diverse elaborazioni e non conoscere direttamente il destinatario del messaggio; è possibile applicare i concetti relativi alla *pipes and filters architecture* inserendo un ricettore comune, un *message router* che si occupa del *routing* di tutti i messaggi e del passaggio di essi attraverso i vari filtri e trasformazioni. Un

[17] *Simple Object Access Protocol*
[18] *JavaScript Object Notation*

message broker è un modulo *software* spesso posto all'interno di una MOM_a; si occupa, oltre al *routing*, della validazione, memorizzazione ed invio dei messaggi alle destinazioni appropriate

1.4.4 *Enterprise Service Bus*

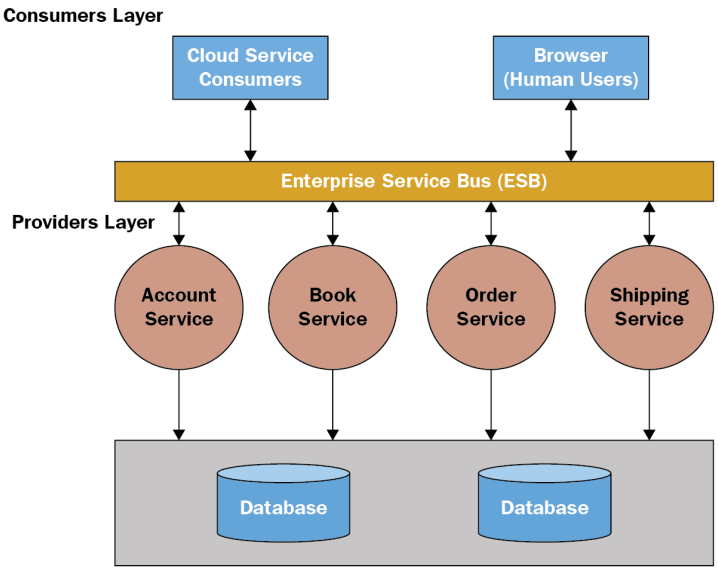

Figura 1.8: Illustrazione esemplificativa di un ESB_a

Fonte: https://subscription.packtpub.com/book/application_development/9781789133608/1/ch01lvl1sec12/service-oriented-architecture-soa

Un ESB_a è un *pattern* architetturale in cui un *software* centrale consente l'integrazione tra diverse applicazioni (figura 1.8). Esso si occupa della comunicazione, trasformazione e conversione dei dati all'interno di una SOA_a; è una tipologia più evoluta di *message broker*. Senza strumenti di questo tipo, la *Service Oriented Architecture*$_g$ porterebbe ad un sistema composto semplicemente da un gruppo di servizi. Ogni servizio dovrebbe occuparsi dello scambio di messaggi con tutti gli altri ($P2P_a$[19]) creando, nei sistemi più grandi, problemi per quanto riguarda l'estensione e mantenibilità dei servizi.

1.5 Dominio applicativo

1.5.1 EAI_a e *Middleware*

Il percorso di *stage* intrapreso è associato al settore del *Enterprise Architecture Integration*, che si occupa principalmente del EAI_a (*Enterprise Application Integration*$_g$, figura 1.9) ovvero del-

[19] *Point To Point*

Figura 1.9: Concetti principali del EAI_a

Fonte: https://commons.wikimedia.org/wiki/File:KrisangelChap2-EAI.png

l'integrazione funzionale di applicazioni aziendali per una clientela di grandi dimensioni (come un'azienda di telecomunicazioni), tramite sistemi di integrazione e *Middleware*.

I *Middleware* e sistemi di integrazione prodotti comprendono l'utilizzo di molteplici linguaggi e tecnologie in continua evoluzione.

Dal sito di Red Hat[20]:

> *Il middleware è un software che fornisce alle applicazioni servizi e capacità frequentemente utilizzati, tra cui gestione dei dati e delle API, servizi per le applicazioni, messaggistica e autenticazione.*
>
> *Aiuta gli sviluppatori a creare le applicazioni in modo più efficiente e agisce come un tessuto connettivo tra applicazioni, dati e utenti.*
>
> *Può rendere conveniente lo sviluppo, l'esecuzione e la scalabilità di applicazioni alle organizzazioni con ambienti multicloud e containerizzati.*

I *Middleware* pertanto vedono due importanti utilizzi nel settore EAI_a:

- **Integrazione su più livelli:** i *Middleware* connettono i principali sistemi aziendali interni ed esterni. Capacità di integrazione quali trasformazione, connettività, componibilità e mes-

[20] *Cos'è il Middleware?* URL: https://www.redhat.com/it/topics/middleware/what-is-middleware.

saggistica enterprise, abbinate all'autenticazione SSO_a[21], aiutano gli sviluppatori a estendere tali capacità su diverse applicazioni.

- **Flussi di dati:** le API_a[22] rappresentano una modalità per condividere i dati tra le applicazioni. Un altro approccio è quello del flusso di dati asincrono, che consiste nella replica di un set di dati in un livello intermedio, da cui i dati possono essere condivisi con più applicazioni.
- **Ottimizzazione di applicazioni esistenti:** con l'adozione del *Middleware*, gli sviluppatori possono trasformare le applicazioni monolitiche esistenti in applicazioni *cloud native* o a microservizi, mantenendo i validi strumenti già in uso ma migliorandone prestazioni e portabilità (figura 1.9).

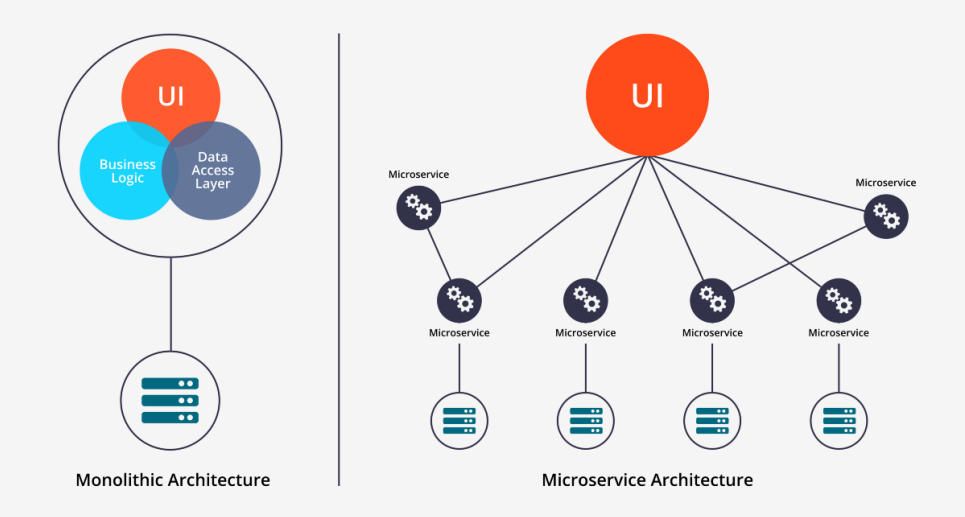

Figura 1.10: Dalle classiche soluzioni monolitiche ai moderni sistemi a microservizi
Fonte: https://aymax.fr/en/why-a-microservices-architecture/

1.5.2 *Container* e *Virtual Machine*

Come anticipato nella sezione precedente, tra le tecnologie più utilizzate in questo settore aziendale vi sono molte piattaforme che permettono di simulare ambienti distribuiti su più macchine fisiche, ove possibile anche indipendenti dal Sistema Operativo su cui viene eseguito il prodotto software.

La simulazione di questi *distributed environment* avviene grazie a sistemi basati sul concetto di *container* (come Docker e Kubernetes) oppure interi Sistemi Operativi che vengono eseguiti all'interno di una *Virtual Machine*. Il vantaggio principale di queste tecnologie è che rendono l'esecuzione del software al loro interno completamente indipendente dall'ambiente circostante,

[21] *Single Sign On*
[22] *Application Programming Interface*

eliminando problemi di OS_a[23] differenti tra i componenti del team o tra l'azienda e i clienti o divergenze nelle dipendenze con relative versioni. Un *container* o una *Virtual Machine* contengono tutto il necessario affinchè sia possibile eseguire il software al suo interno su diverse macchine fisiche (o anch'esse virtuali).

Queste piattaforme non solo rendono agevole l'esecuzione del software prodotto, ma anche lo sviluppo: la condivisione, *debugging* e manutenzione risultano più agevoli grazie alla condivisione dell'intero *container* o VM_a con gli altri membri del team.

Inoltre, si adattano particolarmente bene a simulare l'ambiente distribuito, un concetto fondamentale nel settore del *eai*; infatti è sufficiente generare molteplici *container* o VM_a sulla stessa macchina fisica per simulare un sistema composto da più macchine fisiche distinte, minimizzando l'utilizzo di risorse senza compromettere il risultato del prodotto finale. È così possibile per l'azienda riprodurre un sistema complesso che si avvicina alle risorse ed esigenze effettive del cliente, che usualmente possiede molti computer e server dislocati.

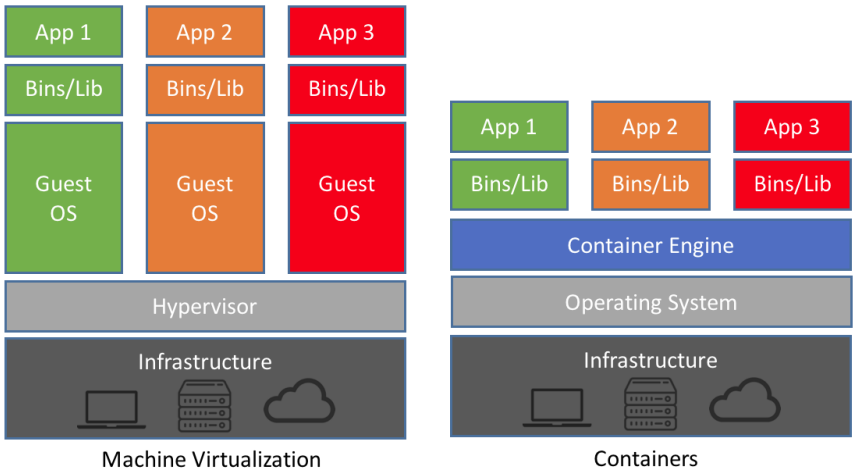

Figura 1.11: Differenti implementazioni legate alle VM_a e *container*
Fonte: https://pawseysc.github.io/container-workflows/01-docker-intro/index.html

La figura 1.11 rappresenta graficamente le due diverse implementazioni delle due tecnologie. Vi sono dunque delle notevoli differenze, vantaggi e svantaggi tra l'utilizzo dell'una e dell'altra, tra cui:

1. i *container* sono più rapidi delle VM_a nell'esecuzione;

2. i *container* sono più leggeri delle VM_a in termini di memoria;

[23] *Operating System*

3. i *container* sono più adatti a simulare un'architettura a microservizi, dato la relativa semplicità ed efficienza rispetto ad una VM_a;
4. le VM_a sono considerate tendenzialmente più sicure dei *container*
5. le infrastrutture e strumenti di gestione di grandi quantità di VM_a sono più consolidate dei corrispettivi strumenti associati ai *container*.

Durante il percorso di stage ho approfondito le mie conoscenze riguardo entrambe le tecnologie, optando di utilizzare i *container* all'interno del mio progetto di *stage*, poichè più efficiente considerate le risorse a mia disposizione. La scelta della piattaforma di *container* è stata quella di Docker. Più precisamente, ho utilizzato l'estensione *Docker-compose*[24] per gestire in modo elegante la generazione e collaudo di più servizi indipententi (figura 1.11): non solo questo *software* consente di creare velocemente una rete di *container* comunicanti su di una rete isolata, ma rende anche rapido ed efficiente lo sviluppo grazie alla possibilità di modificare e riavviare rapidamente un singolo servizio.

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
```

Figura 1.12: Frammento di codice che espone un esempio di ambiente *multi-container* con docker-compose

Come si può vedere in figura 1.12, è relativamente semplice e veloce creare un ambiente composto da due container (il servizio intitolato "web", composto a partire dalla cartella corrente e accessibile alla porta 5000, e il servizio "redis", generato a partire da un'immagine predefinita, entrambi connessi alla stessa rete locale di default di Docker.

Il caso d'uso realizzato nel mio percorso ha simulato grazie ai *container* un sistema a microservizi che simula le risorse di un grande cliente gestore di telecomunicazioni, secondo una visione coerente con il tipo di clientela reale dell'azienda.

[24] *Overview of Docker Compose | Docker Documentation.* URL: https://docs.docker.com/compose/.

1.5.3 L'introduzione di Apache Kafka

In questo periodo Sync Lab sta iniziando dei percorsi per implementare nuove tecnologie nei prodotti *Middleware*. Uno di questi prodotti è Apache Kafka.

Kafka è una piattaforma di *event streaming*, un sistema distribuito e moderno basato sugli eventi anzichè su di una soluzione più classica come può essere quella del *request/response* e $P2P_a$.

L'*event streaming* è una pratica focalizzata sul catturare dati in *real-time* da diverse fonti come *database*, sensori e dispositivi mobili sotto la forma di un flusso continuo di eventi; garantisce uno scambio continuo di informazioni e la loro interpretazione.

Apache Kafka è un *software* fondato sul *design pattern* del *publish/subscribe*. Contiene infatti delle API_a intitolate *Producer* e *Consumer*, che un utente della piattaforma può utilizzare rispettivamente per pubblicare degli eventi o riceverli istantaneamente. Permette inoltre di memorizzare questo flusso di dati in modo affidabile, *fault tolerant*, e durabile, o di processare il flusso per processarlo e trasformarlo.

Un evento è definito come un'occorrenza significativa o un cambiamento di stato del sistema, e nell'ottica del *messaging design pattern* occuperebbe il ruolo di un messaggio atomico.

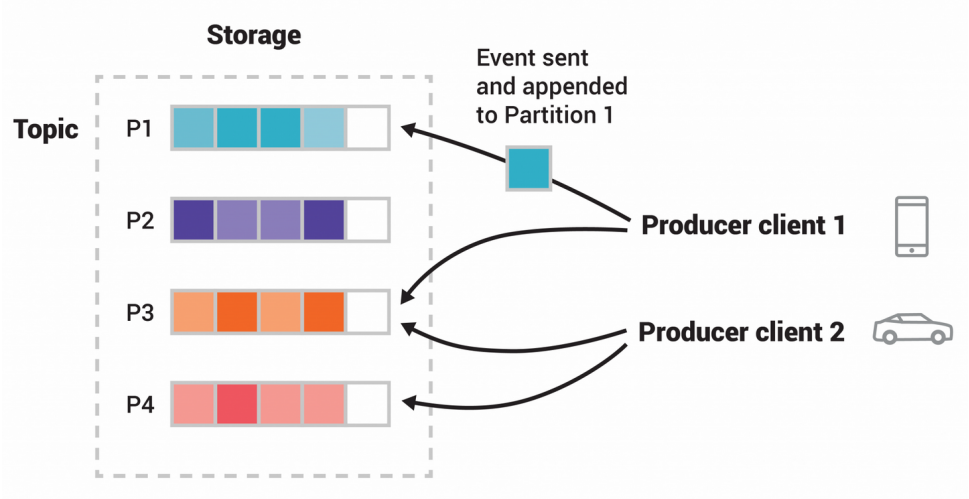

Figura 1.13: Schema di un topic contenente diversi eventi, diviso in partizioni (P), con molteplici *producer*
Fonte: https://kafka.apache.org/intro

Essi sono organizzati all'interno di *topics*, delle liste ordinate di eventi in cui molteplici applicazioni possono scrivere o leggere tali eventi simultaneamente. In figura si può vedere un esempio di *topic*, diviso in diverse partizioni, in cui due *producer* inseriscono dati simultaneamente.

Kafka viene eseguito come un gruppo (*cluster*) di *server* distribuiti. Alcuni di questi *server* sono chiamati *broker*, e insieme compongono il livello di memorizzazione (*storage layer*). Altri *server* sono dedicati all'esecuzione di Kafka Connect, un componente essenziale per l'importazione dei

dati da altre fonti; esso è molto rilevante nel campo dei *Middleware*, poichè permette di integrare Kafka all'interno di sistemi pre-esistenti in modo graduale, con un investimento iniziale parziale.

I *client* invece permettono di sviluppare applicazioni distribuite a microservizi che leggono, scrivono e processano il flusso di dati; questi *client* mettono a disposizione delle interfacce per molti linguaggi e *software* differenti, tra cui Java, Scala, Python, C++/C oltre a delle REST_a API_a.

Il sistema è molto adattabile, dato che può essere installato anche su *acrlongvm*, *container*, ambienti *cloud* e addirittura *bare-metal hardware*. La piattaforma è costituita da un sistema distribuito di *server* e *client* che comunicano attraverso un protocollo *network* TCP_a[25] ad alta *performance*.

Il *software* ha dimostrato negli anni recenti un notevole successo in diversi campi[26], come quello del flusso di *Big Data*, del monitoraggio e dell'elaborazione dati in tempo reale. L'adozione del *software* nell'ambito del EAI_a è in crescita dato le dimostrate qualità nel gestire grandi moli di dati: la sua performance, sicurezza e scalabilità sono i punti che hanno portato il software al suo attuale successo.

1.6 L'innovazione all'interno dell'azienda

Questo contesto dell'integrazione aziendale porta dunque l'impresa ad avere un'importante propensione all'innovazione, talvolta esplicitamente richiesta dai clienti.

Una direzione dell'evoluzione attuale nel settore EAI_a riguarda la migrazione verso sistemi sempre più distribuiti, in accordo con l'ambiente di lavoro descritto nelle sezioni precedenti, in grado di gestire efficacemente ed in tempo reale flussi di dati in continua crescita e appartenenti al mondo del *Big Data*.

L'avanguardia tecnologica è pertanto uno dei principali temi dell'azienda, che garantisce che essa rimanga sempre competitiva sul mercato dei sistemi di integrazione e nel settore dell'EAI_a. L'implementazioni di nuove tecnologie, come può essere una *Event Driven Architecture* basata su Kafka, permetterebbe a Sync Lab di offrire soluzioni sempre più moderne, scalabili e affidabili ai propri clienti.

[25] *Transmission Control Protocol*
[26] Fonte: `https://kafka.apache.org/powered-by`

Capitolo 2

Apache Kafka nell'Integrazione Aziendale

2.1 Obiettivi aziendali

Per soddisfare le richieste dei clienti ed essere sempre competitiva e all'avanguardia, una priorità di Sync Lab sono le esplorazioni tecnologiche e di prodotto anche tramite l'utilizzo di percorsi di *stage* insieme ai laureandi, come quanto accaduto nella mia esperienza. Questi percorsi consentono all'azienda non solo di testare l'utilizzo di nuovi *software* ma anche di conoscere e mettere alla prova le capacità del laureando in vista di una potenziale assunzione al termine dello *stage*.

Nel settore dell'*Enterprise Application Integration*$_g$, l'evoluzione tecnologica è diretta verso soluzioni sempre più distribuite e con un flusso di dati in continuo aumento. Uno degli obiettivi specifici nell'area EAI$_a$ di Sync Lab è pertanto quello di trovare un *software* o tecnologia in grado di soddisfare i bisogni dei clienti di gestire un flusso di dati di dimensioni molto maggiori a quelle attuali, tramite architetture a messaggio che utilizzano servizi distribuiti.

Nell'ambito dei *Middleware* per i sistemi di integrazione, l'aumento del flusso di dati e lo spostamento verso strutture distribuite provoca una difficoltà nella comunicazione e utilizzo di tali dati tra i diversi servizi. Nell'ambito dell'integrazione per un cliente di piccole dimensioni, in cui i dati circolano tra un numero di componenti limitato, può essere sufficiente un'architetture basata sul P2P$_a$[1].

[1] *Point To Point*

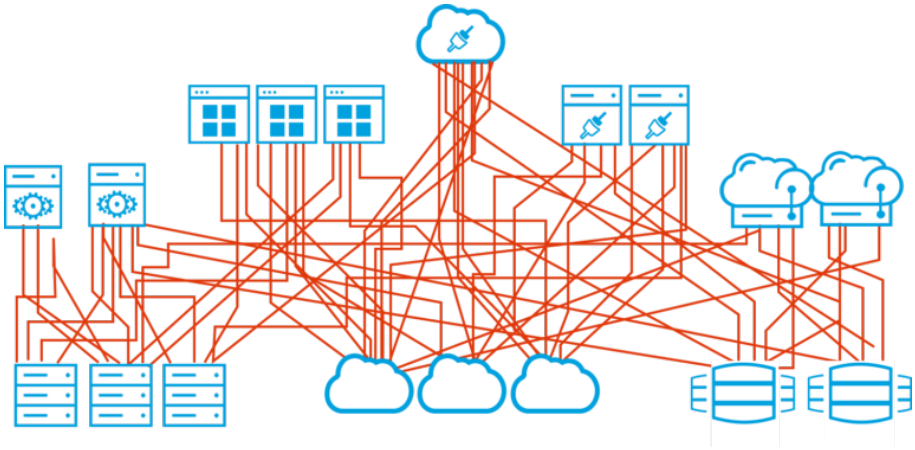

Figura 2.1: Illustrazione di un sistema basato sul $P2P_a$
Fonte: https://news.pwc.be/messaging-architecture-with-salesforce/

Nel caso di un cliente di maggiori dimensioni tuttavia questo approccio rende la manutenzione e gestione del flusso di dati molto difficoltoso e costoso in termini di risorse (figura 2.1), dato che il grande numero di collegamenti tra i vari punti.

Una delle soluzioni che viene maggiormente implementata per risolvere questo problema è la migrazione verso una EDA_a (*Event Driven Architecture*), un'architettura basata sugli eventi in grado di scambiare dati tra punti multipli.

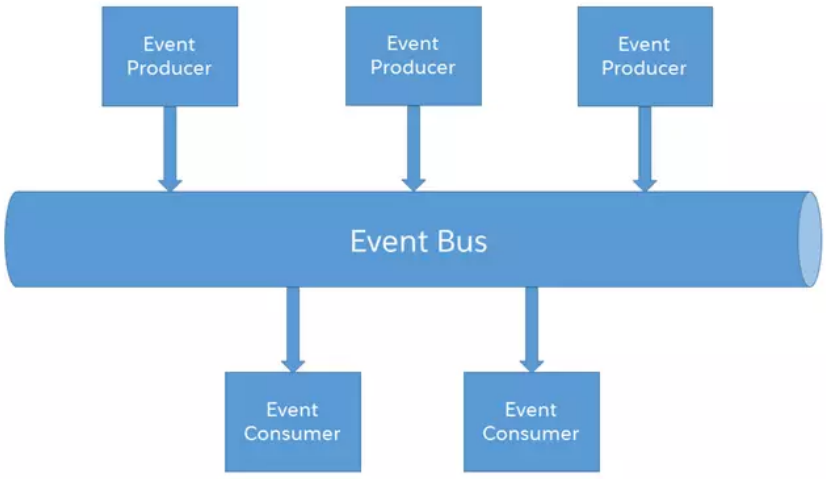

Figura 2.2: Illustrazione di un sistema basato sulla EDA_a
Fonte: https://news.pwc.be/messaging-architecture-with-salesforce/

Questo tipo di architettura è pertanto definita per gestire la produzione, il rilevamento e la reazione agli eventi (figura 2.2) grazie ad un *Design Pattern* di tipo *Publish/Subscribe*, eliminando i problemi visti precedentemente causati dal sistema $P2P_a$. Questa architettura prevede l'utilizzo

di servizi chiamati *Producer*, il cui scopo è fornire dati al *event bus* centrale. Una volta che i dati sono inseriti all'interno del *bus* centrale, ogni servizio in ascolto (*Subscriber*) li riceverà idealmente in tempo reale.

2.1.1 Kafka come Middleware

Per soddisfare le esigenze di innovazione l'azienda ha avviato un percorso per indagare le capacità del software Apache Kafka nell'ambito dell'integrazione aziendale.

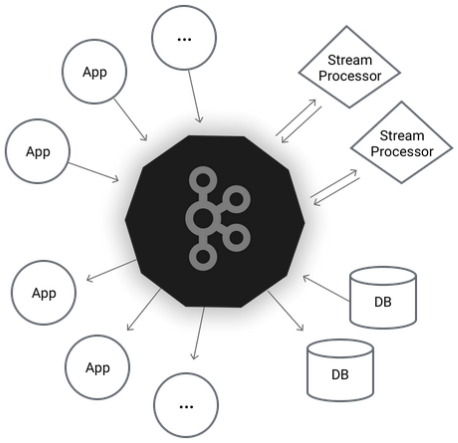

Figura 2.3: Illustrazione di Apache Kafka in un caso d'uso esemplificativo
Fonte: https://iotbyhvm.ooo/apache-kafka-a-distributed-streaming-platform/

Apache Kafka si integra ottimamente in molti sistemi basati sul *messaging pattern* e una EDA_a, in cui lo scambio affidabile di dati tra numerosi servizi in tempo reale è essenziale (in figura 2.3 è illustrato un caso d'uso esemplificativo di un sistema distribuito basato su Kafka).

L'interesse di Sync Lab nel *software* risiede dunque nell'utilizzo di Kafka come un *Middleware* per soddisfare i problemi di integrazione aziendale e reingegnerizzare i flussi di dati preesistenti, ovvero sviluppare un nuovo sistema che consenta la comunicazione tra differenti servizi con un rapido flusso di dati fra di essi (in figura 2.3, un semplice sistema a microservizi indipendenti).

L'azienda ha avviato un percorso per testare le capacità di Kafka rispetto agli attuali strumenti utilizzati nel settore, per valutare i vantaggi e svantaggi che l'adozione di tale *software* può fornire al cliente.

Sono numerosi i vantaggi che Kafka può portare nel settore, fra cui:

- gestione rapida e performante di un enorme flusso di dati;
- scalabilità;

- sicurezza riguardo la persistenza dei dati;
- semplice integrazione e affiancamento a sistemi già esistenti;
- l'essere una piattaforma *open source*;
- processazione dei dati in tempo reale integrata.

2.2 Motivazioni e obiettivi personali

2.2.1 Scelta del percorso

Una delle ragioni che mi ha portato a scegliere questo percorso di *stage* è l'interesse verso Apache Kafka. L'utilizzo della piattaforma di *event streaming* è sempre più in crescita, come l'evoluzione verso sistemi sempre più distribuiti e a microservizi.

Un altro fattore fondamentale alla scelta del percorso sono stati la famigliarità con l'azienda, il personale giudizio positivo che ho avuto riguardo il loro metodo di lavoro e la libertà di sviluppo concessa: ho ritenuto importante la possibilità di elaborare personalmente un'architettura del caso d'uso con una visione ad alto livello, anzichè il semplice sviluppo di un software predeterminato e dal percorso strettamente imposto.

2.2.2 Obiettivi personali

L'obiettivo fondamentale dello *stage* è colmare il divario tra il mondo accademico e quello lavorativo. Grazie al percorso di *stage* in una ditta esterna ho avuto l'opportunità di conoscere l'ambiente di lavoro di un'azienda nel campo ICT_a, facilitandomi l'inserimento nel mondo del lavoro.

Un altro obiettivo è ottenere una formazione riguardo la tecnologia di Kafka, che ritengo possa arricchire fortemente le mie capacità e *skill* professionali. Sono pertanto interessato a sviluppare la mia formazione riguardo l'utilizzo e le implicazioni di questa tecnologia in rapida espansione, la cui formazione potrà essermi utile in molti campi anche al di fuori degli obiettivi dell'azienda ospitante lo *stage*.

2.3 Il percorso di Stage

2.3.1 Obiettivi dello *stage*

Il percorso di *stage* offerto dall'azienda si inserisce all'interno della strategia aziendale più ampia descritta sopra. Al fine di esplorare la tecnologia di Apache Kafka nell'ambito di un *Middleware*

per l'integrazione aziendale, l'azienda ha proposto un percorso di ***stage* il cui obiettivo è la reingegnerizzazione di un flusso di dati asincrono, utilizzando un'architettura basata su Kafka all'interno di un caso d'uso simulato tramite servizi indipendenti.**

Lo stagista ha il compito di osservare, testare e verificare che il *software* possa svolgere alcuni compiti inerenti all'area del EAI_a, analizzando alcuni casi d'uso presenti in un *Middleware* aziendale in ambito *telco*. Il percorso di prevede una durata di 300 ore lavorative.

2.3.2 Prodotti attesi

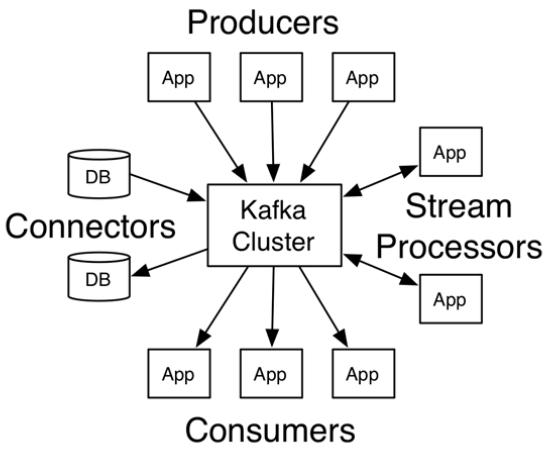

Figura 2.4: Illustrazione di un sistema a servizi con Kafka
Fonte: https://kafka.apache.org/20/documentation.html

I prodotti attesi al termine dello *stage* sono dunque associati alla realizzazione di tre flussi di integrazione, basati su dei casi d'uso reali, per la gestione dei paradigmi di integrazione asincrono e asincrono con *callback* (due requisiti obbligatori), e sincrono ove fosse disponibile del tempo aggiuntivo e se ritenuto opportuno; durante il percorso considerato il contesto e le opportunità offerte dal *software* questo obiettivo verrà sostituito per testare delle funzionalità aggiuntive di Kafka.

Il prodotto *software* finale sarà un sistema basato su servizi indipendenti costruito con un'architettura di tipo EDA_a tramite l'utilizzo di Kafka (figura 2.4).

2.3.3 Contenuti formativi previsti

La realizzazione di questi prodotti necessita una sostanziale formazione dello stagista riguardo i principali concetti del settore del *Enterprise Application Integration* e l'utilizzo della piattaforma

di *event streaming* Kafka. Più precisamente, i contenuti formativi previsti durante questo percorso di *stage* sono i seguenti:

- Concetti chiave del *Enterprise Application Integration*$_g$;
- *Design architettetturali*;
- Cenni di *Networking* applicato alle architetture distribuite;
- Architetture di Integrazione e *Middleware*;
- Apache Kafka.

2.3.4 Interazione tra studente e referenti aziendali

Regolarmente, (almeno una volta la settimana) ci saranno incontri online (tramite la piattaforma Google Meet) con il tutor aziendale Francesco Giovanni Sanges, il responsabile dell'area EAI$_a$ Salvatore Dore e gli esperti delle tecnologie affrontate. I meeting saranno necessariamente online, dato il dislocamento dei vari membri in diverse città.

Lo scopo di questi incontri sarà quello di verificare lo stato di avanzamento, chiarire gli obiettivi ove necessario, affinare la ricerca e aggiornare la pianificazione iniziale.

2.3.5 Pianificazione del lavoro

Ad ogni incremento è associato un requisito obbligatorio, desiderabile o facoltativo. A questi requisiti vi è associato un codice identificativo per favorirne il tracciamento futuro, in che precede la voce descrittiva dell'incremento. Ogni codice è composto da una lettera seguita da dei numeri interi, secondo il seguente modello:

A-X.Y.Z

ove, da sinistra verso destra:

- **A** rappresenta la lettera che qualifica il requisito come obbligatorio, desiderabile o facoltativo, secondo la seguente notazione:
  - *O* per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
  - *D* per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;

  - F per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

- **X** rappresenta la settimana in cui viene inizialmente pianificato l'incremento (identificata da un numero incrementale e intero, partendo da 1). Questo consente allo studente, al tutor interno e al tutor interno una rapida quantificazione dell'avanzamento corrente dello stage rispetto a quanto inizialmente pianificato.

- **Y** rappresenta la posizione sequenziale prevista dell'incremento all'interno della settimana (incrementale e intero, partendo da 1). Esso è strettamente associato alla lettera.

Di seguito viene presentata la pianificazione settimanale delle ore lavorative previste. Ad ogni settimana sono assegnate le voci contenenti gli incrementi previsti in essa, ove i codici utilizzano la notazione descritta precedentemente.
Tutte le settimane prevedono 40 ore lavorative, fatta eccezione per l'ultima che ne prevede 20.

Settimana	**Codice**	**Task associati**
1	O-1.1	Incontro con le persone coinvolte nel progetto per discutere i requisiti e le richieste relative al sistema da sviluppare
	O-1.2	Verifica credenziali e strumenti di lavoro assegnati
	O-1.3	Presa visione dell'infrastruttura esistente
	D-1.1	Ripasso approfondito riguardo i seguenti argomenti: • Ingegneria del *software*; • Sistemi di versionamento; • Architetture *software*; • Cenni di *Networking*.
2	O-2.1	Nozioni fondamentali riguardo EAI_a e SOA_a[2]
	O-2.2	Approfondimenti riguardo le Architetture a Messaggio, in particolare: • *Integration Styles*; • *Channel Patterns*; • *Message Construction Patterns*; • *Routing Patterns*; • *Transformation Patterns*; • *System Management Patterns*.

[2] *Service Oriented Architecture*$_g$

3	O-3.1	Apache Kafka: • Introduzione a Kafka; • Concetti fondamentali di Kafka; • Avvio e CLI$_a$[3]; • Programmazione in Kafka con Java.
	D-3.1	Esempi e applicazioni di Apache Kafka.
4	O-4.1	Confluent Platform: • *Service registry*; • REST$_a$ *proxy*; • kSQL; • Confluent *connectors*; • *Control center*.
5	O-5.1	Analisi dei casi d'uso reali
	O-5.2	Realizzazione dei componenti per l'esecuzione dei casi di test
6	O-6.1	Analisi reingegnerizzazione e collaudo del flusso di integrazione asincrono
7	O-7.1	Analisi e reingegnerizzazione e collaudo del flusso di integrazione asincrono con callback
8	O-8.1	Analisi e reingegnerizzazione e collaudo del flusso di integrazione sincrono.

Tabella 2.1: Pianificazione settimanale dello *stage*

Secondo questa pianificazione, (di cui la figura 2.5 rappresenta il diagramma di Gantt) le 300 ore di *stage* previste sono approsimativamente divise in:

- 160 ore di Formazione sulle tecnologie;
- 60 ore di Progettazione dei componenti e dei test;
- 60 ore di Sviluppo dei componenti e dei test;
- 20 ore di Valutazioni finali, Collaudo e Presentazione della Demo.

[3] *Command Line Interface*

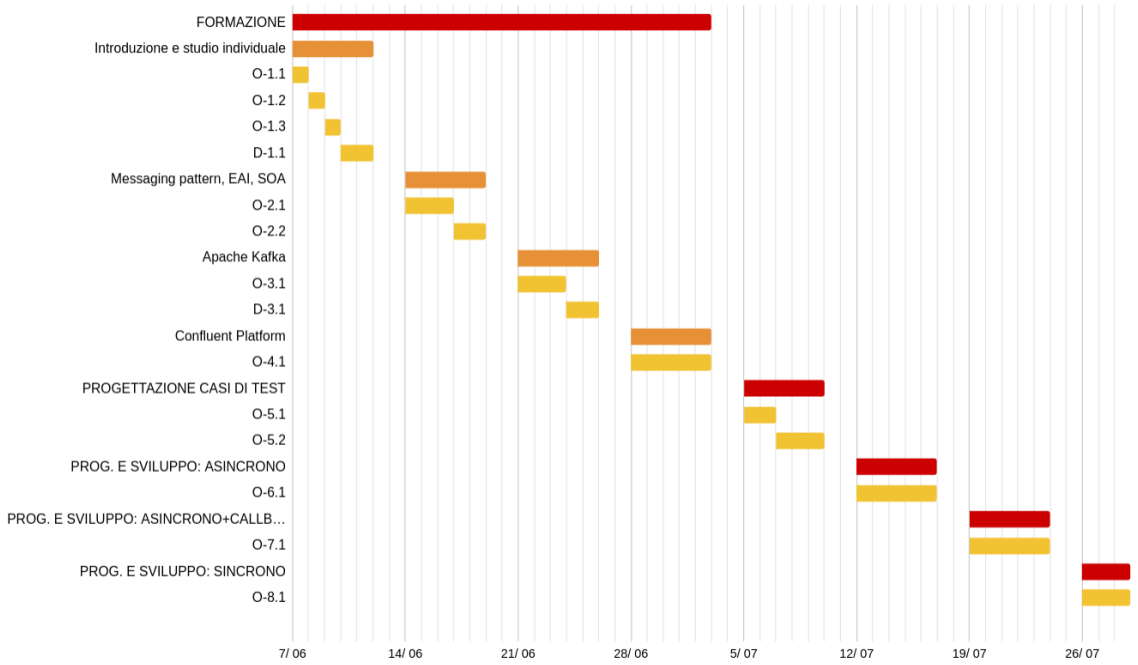

Figura 2.5: Diagramma di Gantt del piano di lavoro

Fonte: elaborazione personale

Capitolo 3

Apache Kafka in caso d'uso simulato

Capitolo 4

Obiettivi soddisfatti e bilancio formativo

4.1 Obiettivi soddisfatti dallo stage

Valutazione oggettiva riguardo il percorso e i risultati raggiunti da esso.

4.2 Maturazione professionale acquisita

Descrizione delle conoscenze e abilità professionali acquisite grazie al percorso di *stage*. Valutazione del miglioramento personale portato avanti durante il percorso di stage.

4.3 Distanza tra le competenze necessarie e quelle acquisite nel corso di studi

Breve valutazione riguardo le difficoltà riscontrate, e considerazioni riguardo le competenze ottenute durante il corso di laurea che più mi hanno aiutato durante il percorso.

Acronimi

API Application Programming Interface. 15, 18, 19

CLI Command Line Interface. 27

COBRA Common Object Request Broker Architecture. 6

EAI *Enterprise Application Integration*$_g$. 1, 4, 7, 8, 10, 11, 13, 14, 19, 20, 24–26

EDA Event Driven Architecture. 4, 19, 21, 22, 24

EJB Enterprise Java Bean. 6

ESB Enterprise Service Bus. 1, 4, 10, 11, 13

HTTP HyperText Transfer Protocol. 12

ICT Information and Communication Technologies. 5, 23

J2EE Java 2 Platform Enterprise Edition. 6

JMS Java Message Service. 6

JSON JavaScript Object Notation. 12

MOM Message Oriented Middleware. 6, 13

O-O Object Oriented. 6, 7

OS Operating System. 16

P2P Point To Point. 4, 13, 18, 20, 21

REST REpresentational State Transfer. 19, 27

SOA *Service Oriented Architecture*$_g$. 1, 4, 10–13, 26

SOAP Simple Object Access Protocol. 12

SSO Single Sign On. 15

TCP Transmission Control Protocol. 19

UML Unified Modeling Language. 7

VM Virtual Machine. 4, 16, 17

Glossario

Enterprise Application Integration

Il termine si riferisce al processo d'integrazione tra diversi tipi di sistemi informatici di un'azienda attraverso l'utilizzo di software e soluzioni architetturali.

Fonte: https://it.wikipedia.org/wiki/Enterprise_Application_Integration . 13, 20, 25, 31

Service Oriented Architecture

La Service Oriented Architecture definisce un modo per rendere i componenti software riutilizzabili tramite interfacce di servizio. Queste interfacce utilizzano standard di comunicazione comuni in modo da poter essere rapidamente integrate in nuove applicazioni senza dover eseguire ogni volta una profonda integrazione.

Ogni servizio in una SOA incorpora il codice e le integrazioni dei dati necessari per eseguire una funzione aziendale completa e discreta (ad esempio, il controllo del credito del cliente, il calcolo di un pagamento di un prestito mensile o l'elaborazione di un'applicazione ipotecaria). Le interfacce di servizio forniscono un accoppiamento libero, il che significa che possono essere richiamate con poca o nessuna conoscenza della sottostante modalità di implementazione dell'integrazione.

I servizi sono esposti utilizzando protocolli di rete standard - come SOAP (simple object access protocol)/HTTP o JSON/HTTP - per inviare richieste di lettura o modifica dei dati. I servizi sono pubblicati per consentire agli sviluppatori di trovarli rapidamente e riutilizzarli per assemblare nuove applicazioni.

Fonte: https://www.ibm.com/it-it/cloud/learn/soa . 1, 10, 11, 13, 26, 32

Bibliografia

[1] *ClickUp™ | One app to replace them all.* URL: https://clickup.com.

[2] *Cos'è il Middleware?* URL: https://www.redhat.com/it/topics/middleware/what-is-middleware.

[3] *Coursera | Build Skills with Online Courses.* URL: https://www.coursera.org.

[4] *Empowering App Development for Developers | Docker.* URL: https://www.docker.com/.

[5] *Google Chat.* URL: https://chat.google.com.

[6] *Notion - The all-in-one workspace for your notes, tasks, wikis, and databases.* URL: https://www.notion.os.

[7] *Online Courses - Learn Anything, On Your Schedule | Udemy.* URL: https://www.udemy.com.

[8] *Oracle VM Virtual Box.* URL: https://www.virtualbox.org/.

[9] *Overview of Docker Compose | Docker Documentation.* URL: https://docs.docker.com/compose/.

[10] *Sync Lab.* URL: https://www.synclab.it/.