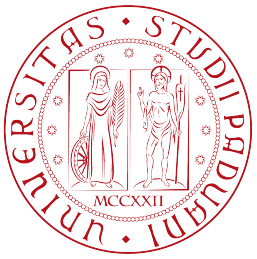

Università degli Studi di Padova

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

Elenco delle tabelle

Elenco delle figure

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*

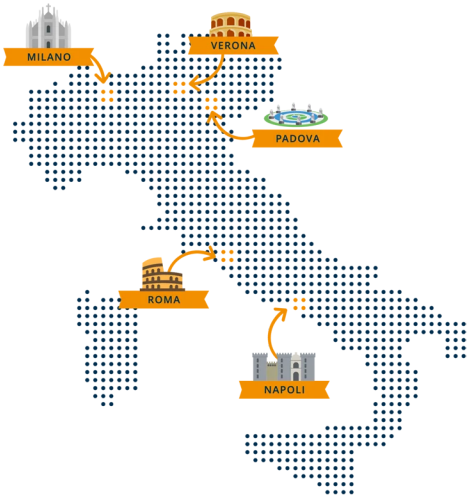

Figura 1.1: Attuali sedi di Sync Lab
Fonte: https://www.synclab.it/

1.2 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*.

1.2.1 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 per garantire libertà e flessibilità allo stagista e all'intero progetto.

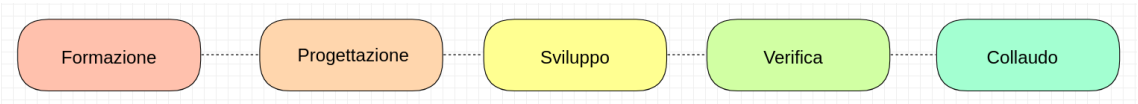

Figura 1.2: Processi interni di cui ho avuto esperienza
Fonte: elaborazione personale

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*).

Per il processo di Formazione, Sync Lab fornisce materiale sotto forma di corsi *online* tramite le

piattaforme *Coursera*[3] e *Udemy*[4] 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 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.

1.2.2 Strumenti organizzativi

L'organizzazione efficiente di un progetto è garantita dall'utilizzo dei vari strumenti a supporto, quali *Kanban Board* (come *Click Up*[5] per la gestione di progetto e *Notion*[6] per le prenotazioni della postazione di lavoro in sede), *chat* (come *Google Chat*[7]) 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.

Le attività (*task*) vengono inizialmente create nella colonna "DA FARE" dal tutor aziendale o da me, 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.

[3] *Coursera | Build Skills with Online Courses.* URL: https://www.coursera.org.
[4] *Online Courses - Learn Anything, On Your Schedule | Udemy.* URL: https://www.udemy.com.
[5] *ClickUp™ | One app to replace them all.* URL: https://clickup.com.
[6] *Notion - The all-in-one workspace for your notes, tasks, wikis, and databases.* URL: https://www.notion.os.
[7] *Google Chat.* URL: https://chat.google.com.

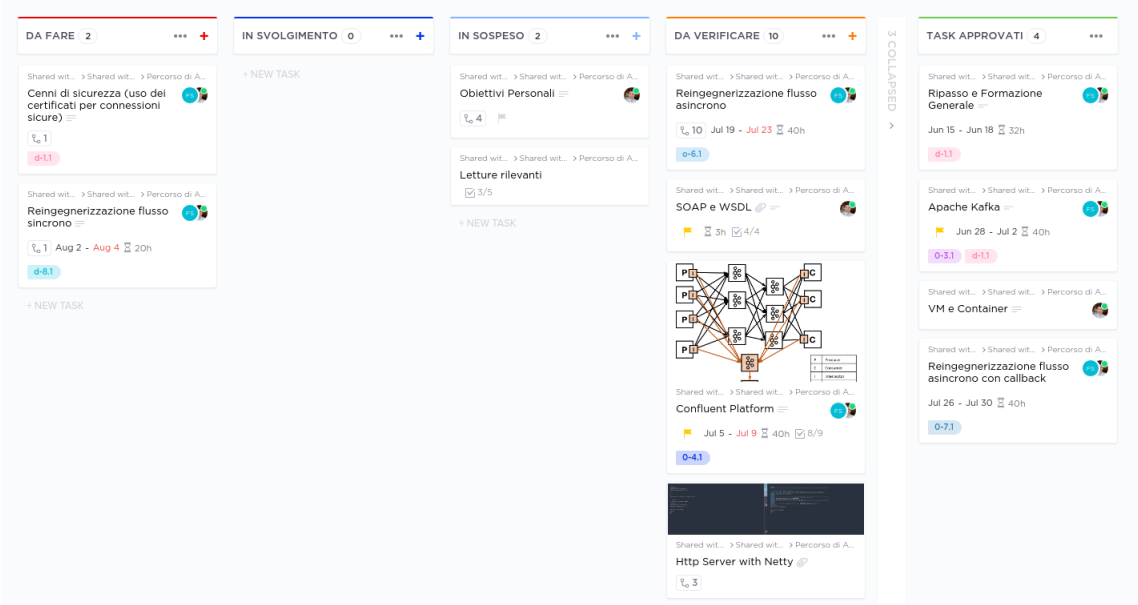

Figura 1.3: *Kanban Board* del progetto di *stage*
Fonte: elaborazione personale

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*.

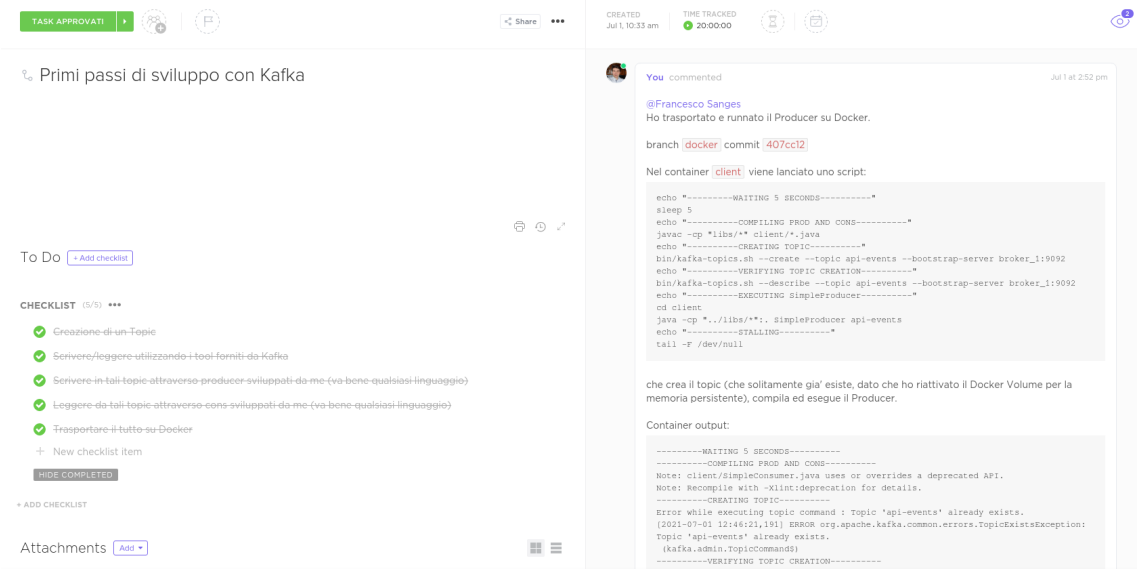

Figura 1.4: Esempio di un'attività del processo di Formazione
Fonte: elaborazione personale

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; que-

st'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.

1.3 Ambiente di lavoro

L'ambiente di lavoro di cui ho avuto esperienza risulta libero e flessibile. Lo sviluppo del prodotto nell'ambito del EAI_a dev'essere indipendente dal linguaggio di programmazione, dagli strumenti utilizzati per l'esecuzione e sviluppo, e possibilmente anche dal Sistema Operativo su cui eseguire il *software*. 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. Nonostante il percorso formativo abbia visto l'apprendimento di entrambe le tecnologie tramite l'utilizzo dei *software Virtual Box*[8] e *Docker*[9], solo quest'ultima è stata utilizzata durante il progetto.

Un altro concetto fondamentale che caratterizza l'ambiente di lavoro nel campo del EAI_a è quello del sistema distribuito, di cui l'azienda ha un forte interesse per soffisdare le necessità dei suoi clienti.

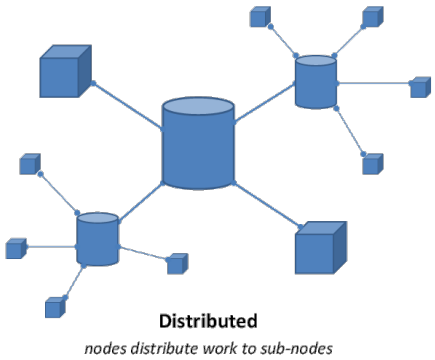

Figura 1.5: Illustrazione di un sistema distribuito
Fonte: https://www.delphitools.info/DWSH/

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

[8] *Oracle VM Virtual Box*. URL: https://www.virtualbox.org/.
[9] *Empowering App Development for Developers | Docker*. URL: https://www.docker.com/.

tutti i singoli componenti (macchine, *hardware* o *software*), cosicchè sia possibile lo scambio dei messaggi.

1.3.1 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$_a$[10], EJB$_a$[11], COBRA$_a$[12] e *Web Services*.
  - Progettazione e realizzazione di sistemi basati su MOM$_a$[13] e JMS$_a$[14].
- 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*.
  - 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*.

1.4 Dominio applicativo

1.4.1 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 1.6) ovvero dell'integrazione funzionale di applicazioni aziendali per una clientela di grandi dimensioni (come un'azienda di telecomunicazioni), tramite sistemi di integrazione e *Middleware*.

[10] *Java 2 Platform Enterprise Edition*
[11] *Enterprise Java Bean*
[12] *Common Object Request Broker Architecture*
[13] *Message Oriented Middleware*
[14] *Java Message Service*

Figura 1.6: Concetti principali del EAI_a

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

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

Dal sito di Red Hat[15]:

> *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 messaggistica enterprise, abbinate all'autenticazione SSO_a[16], aiutano gli sviluppatori a estendere tali capacità su diverse applicazioni.

[15] *Cos'è il Middleware?* URL: https://www.redhat.com/it/topics/middleware/what-is-middleware.
[16] *Single Sign On*

- **Flussi di dati:** le API_a[17] 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 1.6).

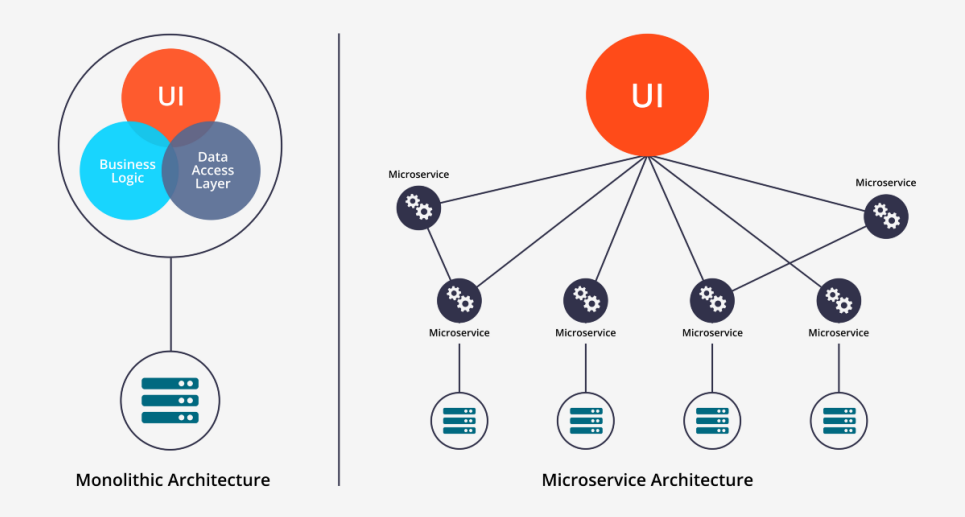

Figura 1.7: Dalle classiche soluzioni monolitiche ai moderni sistemi a microservizi
Fonte: https://aymax.fr/en/why-a-microservices-architecture/

1.4.2 *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, eliminando problemi di OS_a[18] 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

[17] *Application Programming Interface*
[18] *Operating System*

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.

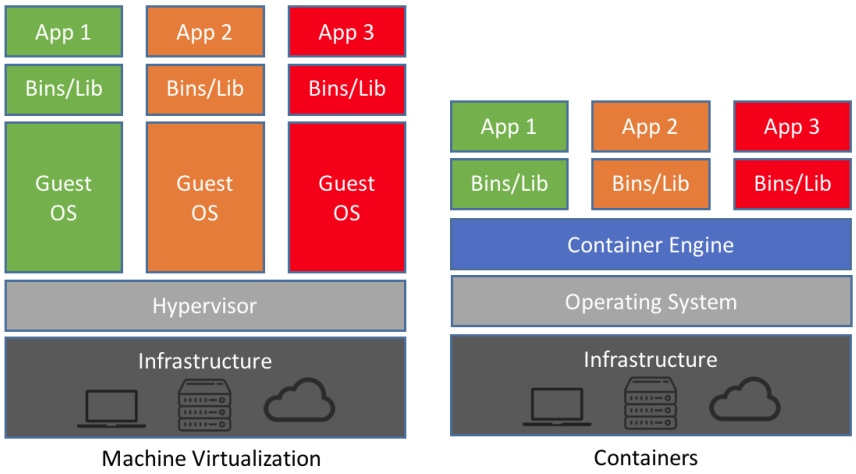

Figura 1.8: Differenti implementazioni legate alle VM_a e *container*
Fonte: https://pawseysc.github.io/container-workflows/01-docker-intro/index.html

La figura 1.8 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;
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*[19] per gestire in modo elegante la generazione e collaudo di più servizi indipententi (figura 1.8): 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. 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.

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 grado di gestire efficacemente ed in tempo reale flussi di dati in continua crescita. L'avanguardia tecnologica è uno dei principali temi dell'azienda, che garantisce che essa rimanga sempre competitiva sul mercato dei sistemi di integrazione (nel settore dell'EAI_a).

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

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*

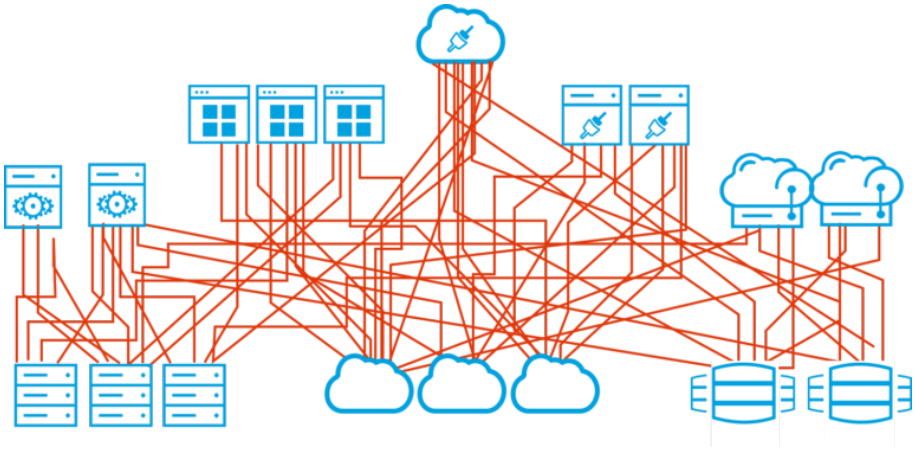

Figura 2.1: Illustrazione di un sistema basato sul P2P$_a$
Fonte: https://news.pwc.be/messaging-architecture-with-salesforce/

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.

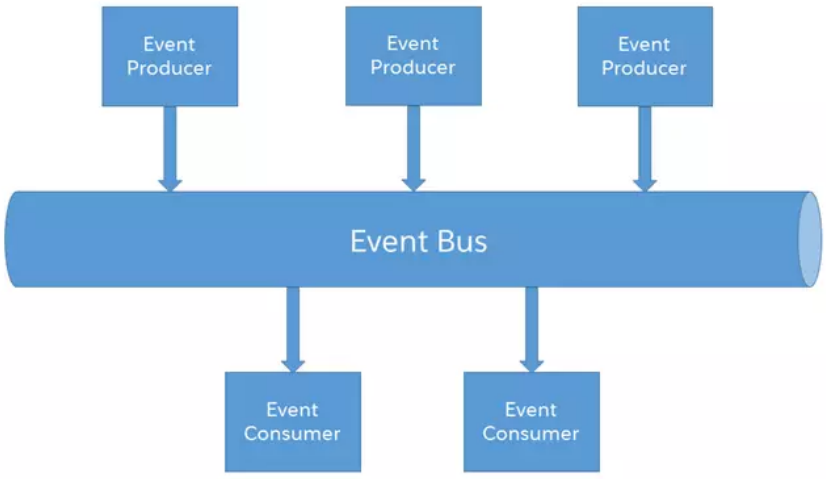

Figura 2.2: Illustrazione di un sistema basato sulla EDA$_a$
Fonte: https://news.pwc.be/messaging-architecture-with-salesforce/

Un evento è definito come un'occorrenza significativa o un cambiamento di stato del sistema; 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.

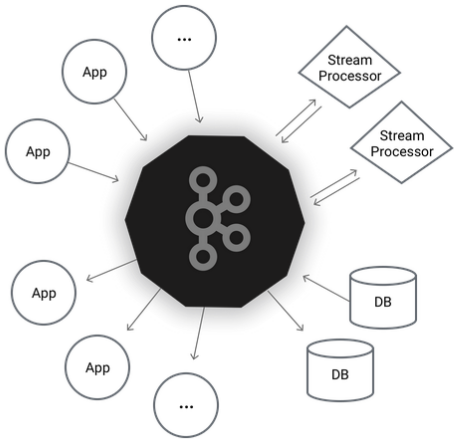

Figura 2.3: Illustrazione di Apache Kafka in un caso d'uso esemplificativo
Fonte: https://iotbyhvm.ooo/apache-kafka-a-distributed-streaming-platform/

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 $P2P_a$. Apache Kafka si integra ottimamente in molti sistemi basati sul *messaging pattern*, in cui lo scambio affidabile di dati in tempo reale è essenziale (in figura 2.3 è illustrato un caso d'uso esemplificativo di un sistema distribuito basato su Kafka).

Il *software* ha dimostrato negli anni recenti un notevole successo in diversi campi[2], 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.

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,

[2]Fonte: https://kafka.apache.org/powered-by

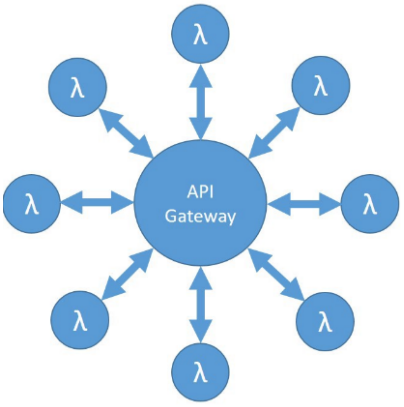

Figura 2.4: Illustrazione di un sistema a microservizi con comunicazioni tramite API_a
Fonte: https://medium.com/@riyaznet/building-serverless-microservices-on-aws-3959a93c2549

ovvero sviluppare un nuovo sistema che consenta la comunicazione tra differenti servizi con un rapido flusso di dati fra di essi (in figura 2.4, 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

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

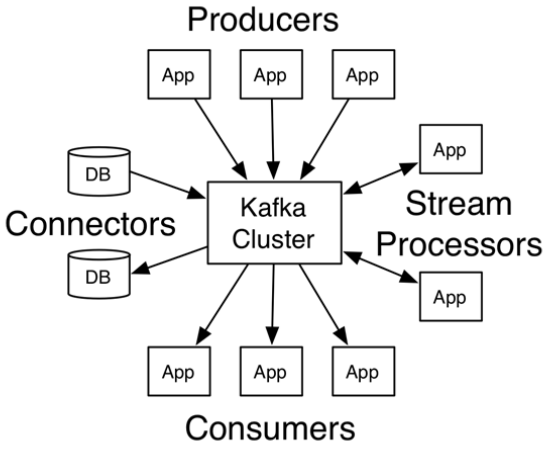

Figura 2.5: Illustrazione di un sistema a servizi con Kafka
Fonte: https://kafka.apache.org/20/documentation.html

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 2.5).

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[3]
	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*.
3	O-3.1	Apache Kafka: • Introduzione a Kafka; • Concetti fondamentali di Kafka; • Avvio e CLI_a[4]; • Programmazione in Kafka con Java.
	D-3.1	Esempi e applicazioni di Apache Kafka.

[3] *Service Oriented Architecture*$_g$
[4] *Command Line Interface*

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*

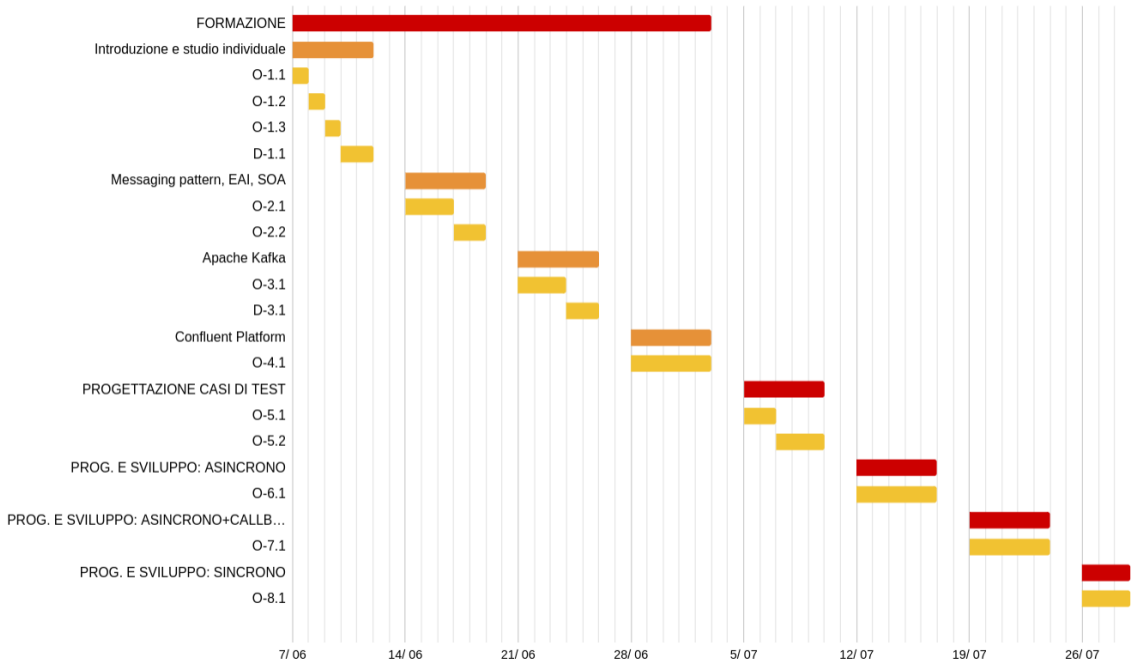

Figura 2.6: Diagramma di Gantt del piano di lavoro
Fonte: elaborazione personale

Secondo questa pianificazione, (di cui la figura 2.6 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.

Capitolo 3

Apache Kafka in caso d'uso simulato

3.1 Definizione di un *way of working*

All'inizio del percorso ho delineato un *way of working*, ovvero un metodo di lavoro da mantenere per tutta la durata dello *stage*, insieme al tutor aziendale, il responsabile del settore EAI_a e gli esperti del settore.

Per mantenere un buon livello organizzativo, quantificare l'avanzamento e rendere agevole la verifica e il supporto tecnico l'azienda ha proposto l'utilizzo di una *board* di progetto. Tra le tante opzioni disponibili, mi è stata proposta la piattaforma *ClickUp*. Rispetto alla concorrenza questa *board* è ricca di funzionalità, pulita nell'esposizione dello stato del progetto, e la maggior parte delle sue funzioni sono gratuite.

All'inizio del percorso il tutor aziendale e il responsabile del EAI_a hanno creato delle *card* contenenti le attività previste per ogni settimana (*task*) al fine di fornire una struttura generale del progetto. All'interno di questi *task* vi sono i concetti chiave, attività previste e obiettivi settimanali che lo stagista è tenuto a seguire per garantire l'efficacia del prodotto finale. Oltre a questi *task* principali, ho potuto creare di *task* ausiliari e dei *subtask* per descrivere più adeguatamente l'attività in corso.

Ciascun *task* contiene una colonna laterale dove ho mantenuto un *log* di tutto ciò che è stato eseguito relativo al *task* in questione, allo scopo di esplicitarne il progresso e rendere agevole un eventuale supporto dal tutor o l'evoluzione futura.

Ogni settimana è previsto un *online meeting* per la verifica del progresso ove necessario, la risposta ad eventuali questioni sollevate, e spiegazioni riguardo lo sviluppo della settimana successiva. Alcune di queste videoconferenze ha visto la partecipazione di altri esperti che mi hanno

aiutato a comprendere meglio il caso d'uso da reingegnerizzare, riassumendo lo stato attuale del sistema d'integrazione per uno dei clienti con relativi *file* utilizzati.

Per mantenere alto il livello di organizzazione, efficienza ed efficacia, all'inizio di ogni giornata lavorativa ho creato un breve piano giornaliero con successivo consuntivo a fine giornata. Questo ha permesso al tutor di verificare rapidamente il corretto avanzamento del processo in corso e a me di mantenere il *focus* su di esso.

3.2 Formazione

Il processo di Formazione ha avuto un importante ruolo all'interno dello *stage*, con una durata complessiva di circa quattro settimane. La causa di questo lungo periodo è data dallo studio di diversi ambiti e concetti, in particolare il settore del *Enterprise Application Integration*$_g$ e la nuova tecnologia di Kafka.

Durante questo processo Sync Lab mi ha fornito del materiale didattico per l'apprendimento, quali diapositive e appunti di origine aziendale e l'accesso a dei corsi riguardo *Software architecture*, *Service Oriented Architecture*$_g$ (tramite i corsi online su *Coursera*) e Apache Kafka (tramite i corsi online su *Udemy*).

Il tutor aziendale e responsabile EAI$_a$ hanno fornito durante i *meeting* settimanali ulteriori chiarimenti e approfondimenti sul come Sync Lab applica questi concetti nello sviluppo di architetture *software*.

Per i corsi online a maggiore contenuto nozionistico ho redatto degli appunti riassuntivi, con lo scopo di consolidare l'apprendimento e velocizzare la verifica del tutor aziendale.

3.3 Progettazione di un caso d'uso

Ad alcune videoconferenze ha partecipato anche un altro esperto *senior* aziendale, esterno al progetto di *stage* in questione, per illustrarmi un caso d'uso in cui l'azienda ha esposto un prototipo di sistema di integrazione utilizzando i concetti di *Web Service*, SOAP$_a$[1] e *request/response*, permettendomi la visualizzazione dei file WSDL$_a$[2], XML$_a$[3] e XSD$_a$[4] associati. Ho pertanto generato un caso d'uso adatto agli scopi dello *stage* ispirandomi al caso d'uso reale illustrato.

Il caso d'uso modellato tratta una richiesta di credito telefonico da parte di un cliente ad un'azienda di telecomunicazioni tramite *Web Service*, per soddisfare il requisito dello sviluppo

[1] *Simple Object Access Protocol*
[2] *Web Service Description Language*
[3] *eXtensible Markup Language*
[4] *XML Schema Definition*

della reingegnerizzazione del flusso di dati asincrono. La *request* avviene tramite flusso di un file $JSON_a$[5] che viene trasmesso attraverso i vari servizi che compongono il sistema di integrazione, basatosul *Design Pattern* di tipo *publish/subscribe*.

Va precisato che il contenuto di tale $JSON_a$ non è strettamente rilevante allo sviluppo e funzionamento del *Middleware*, ma aiuta a stabilire il contesto di utilizzo.

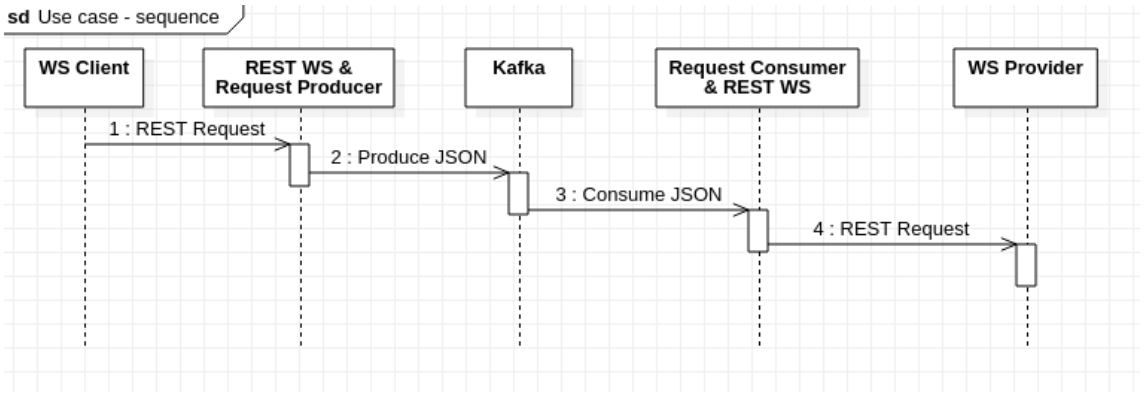

Figura 3.1: Diagramma di sequenza UML_a per il prototipo di caso d'uso iniziale
Fonte: elaborazione personale

Il caso d'uso (figura 3.1) è composto dai seguenti passaggi:

1. il cliente (*WS_a*[6] *Client*) effettua una richiesta di credito tramite invio di un file $JSON_a$ al successivo Servizio Web in ascolto.

2. il servizio composto da $REST_a$[7] WS_a e *Request Producer* riceve il $JSON_a$ e lo inserisce in Kafka tramite l'apposito *Kafka Producer*, assumendo la funzione di *publisher*.

3. il servizio di *Request Consumer*, sottoscritto al *topic* in questione riceve il $JSON_a$ e lo invia al WS_a finale tramite una $REST_a$ request.

4. il servizio in coda chiamato WS_a *Provider* riceve il $JSON_a$; grazie ai dati ricevuti è in grado di fornire il servizio richiesto dal *Client* nello *step* 1.

La modellazione dell'architetture e struttura del sistema da sviluppare seguirà questo prototipo di UC_a[8]. Il modello associato al caso asincrono con *callback* seguirà la stessa struttura e *step* dello UC_a illustrato qui sopra, con l'aggiunta speculare del messaggio di ritorno.

[5] *JavaScript Object Notation*
[6] *Web Service*
[7] *REpresentational State Transfer*
[8] *Use Case*

3.4 Progettazione architetturale

La progettazione architetturale ha portato alla produzione di diversi diagrammi UML$_a$ per rappresentare efficacemente l'architettura del prodotto e fornire un modello da seguire durante il processo di sviluppo. Il processo ha richiesto frequenti *meeting* e confronti per raggiungere un risultato finale soddisfacente al fine della sperimentazione. I diagrammi rappresentano i componenti *color coded*, notazione utilizzata per dare continuità e chiarezza attraverso le diverse tipologie di UML$_a$ *diagrams*.

UML$_a$ *sequence diagrams*

A partire dallo UC$_a$ descritto nela sezione precedente, ho prodotto un UML$_a$ *sequence diagram* più approfondito per rappresentare il flusso del JSON$_a$ tra i vari componenti (figura seguente).

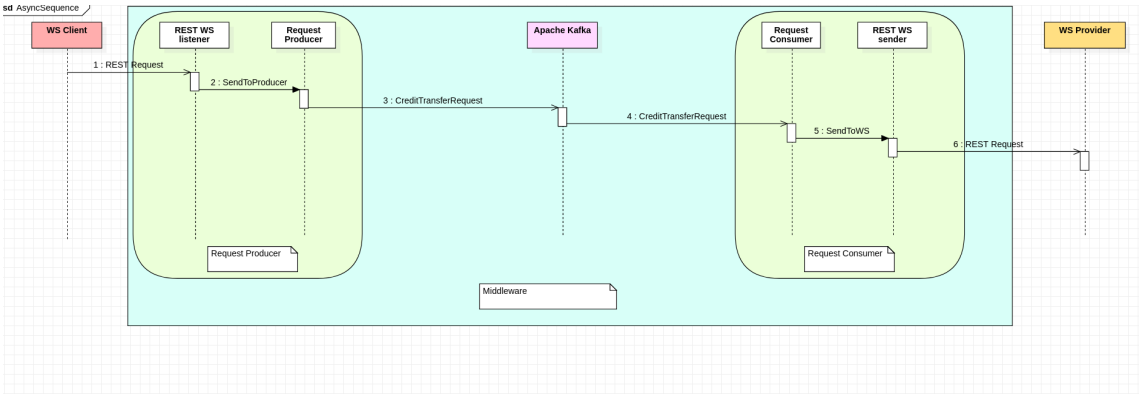

Figura 3.2: Diagramma UML$_a$ di sequenza per la reingegnerizzazione del flusso asincrono
Fonte: elaborazione personale

La progettazione del sistema associato al caso asincrono con *callback* comprende il flusso descritto qui sopra, con aggiunta del flusso di ritorno descritto nella figura seguente.

La reingegnerizzazione del flusso sincrono è stata scartata in favore dello studio di funzionalità aggiuntive tramite l'utilizzo della piattaforma di *event streaming*. La progettazione di un sistema basato su questo flusso era inizialmente prevista (come requisito desiderabile) nel piano di lavoro iniziale poiché associata ad un caso d'uso reale (di cui si è parlato nelle sezioni precedenti), ma infine è stata giudicata poco opportuna e fuori dagli scopi di Apache Kafka, un sistema basato sull'asincronicità

Il tempo associato a tale requisito è stato pertanto riproposto per testare un'altra funzione utile in un *Middleware* quali la trasformazione di alcuni dati presenti nel JSON$_a$. Più precisamente, è

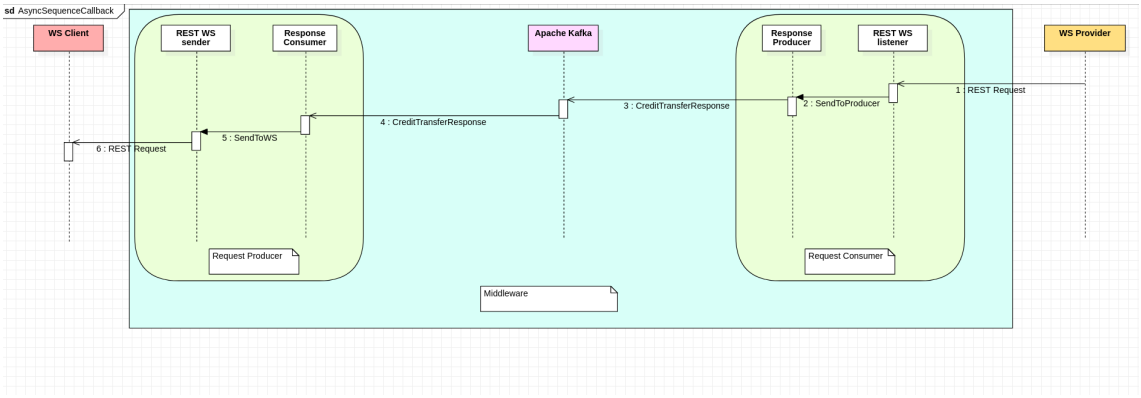

Figura 3.3: Diagramma di sequenza UML$_a$ per la reingegnerizzazione del flusso asincrono con *callback*
Fonte: elaborazione personale

stato aggiunto un dato sensibile che viene nascosto e sostituito con asterischi "*" dopo la produzione del *topic* in Kafka grazie all'utilizzo di Kafka Streams.

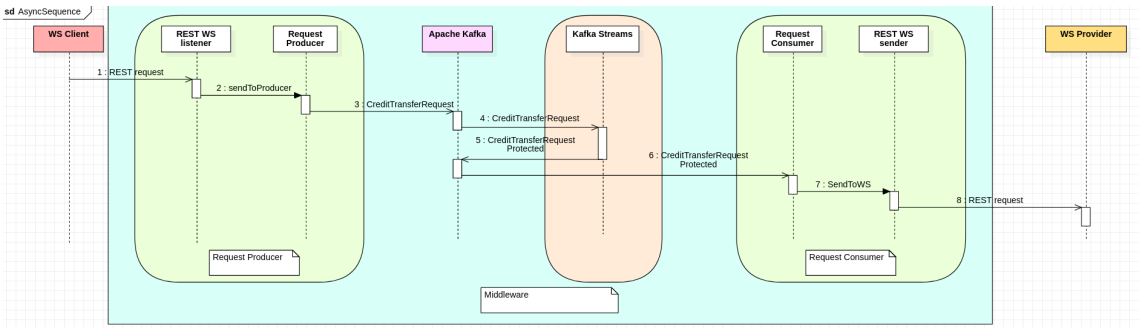

Figura 3.4: Diagramma di sequenza UML$_a$ per la reingegnerizzazione del flusso asincrono con protezione dei dati sensibili.
Fonte: elaborazione personale

In figura 3.4 possiamo vedere il nuovo flusso asincrono con protezione (mascheramento) del dato sensibile. La versione con *callback* del flusso qui sopra prevede anche il flusso di ritorno (*callback*), non illustrato in figura ma facilmente intuibile grazie al grafico che la precede.

UML$_a$ *deployment diagrams*

A supporto di questi UML$_a$ *sequence diagram* che rappresentano efficacemente il flusso di dati, punto focale dell'intero sistema di integrazione (asincrono con la protezione del dato sensibile), ho prodotto ulteriori diagrammi, tra i quali il *deployment diagram*. Questo diagramma ha lo scopo di rappresentare la configurazione dei processi *run time*, modellando la struttura di base in cui eseguono i diversi servizi; il diagramma esprime l'ambiente in cui i vari componenti risiedono e ove essi comunicano tra di loro.

Con l'approvazione degli esperti aziendali, ho deciso di appoggiare il sistema di integrazione sulla piattaforma Docker.

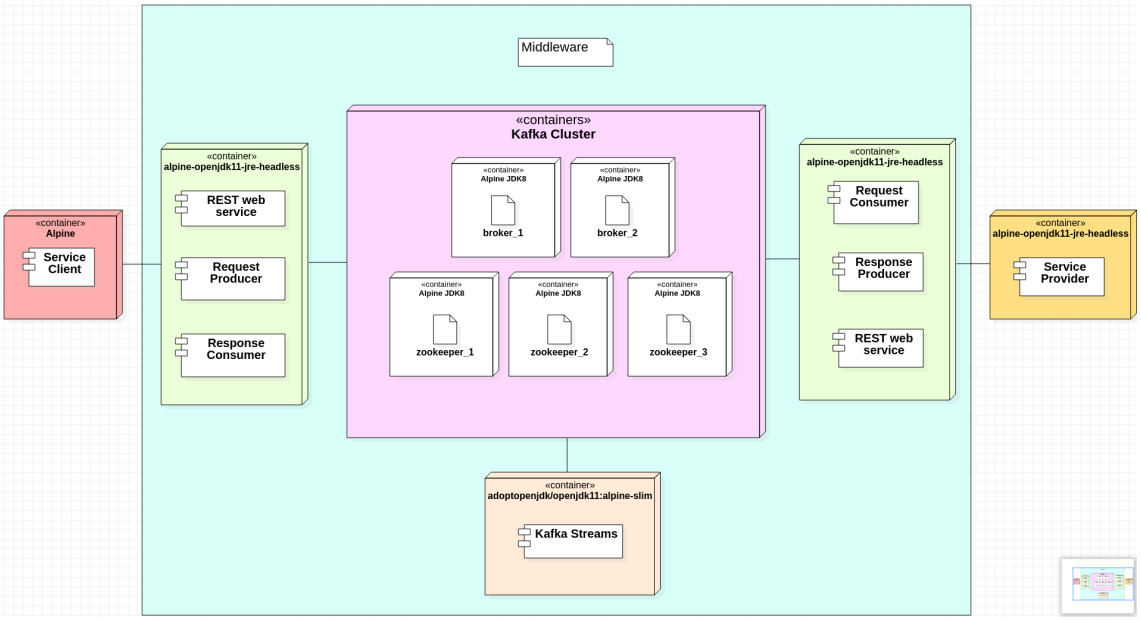

Figura 3.5: UML$_a$ *deployment diagram* per la reingegnerizzazione del flusso asincrono (protetto)
Fonte: elaborazione personale

Il diagramma di *deployment* (figura 3.5) vede pertanto l'utilizzo di numerosi *container* indipendenti che dialogano attraverso una rete locale all'interno di Docker. Questi *container* sono raffigurati dai vari nodi (rappresentati dai cubi in rilievo in figura). A questa notazione fa eccezione il nodo virtuale intitolato "Kafka Cluster", che ha solamente lo scopo di raggruppare i vari nodi legati al *environment* di Kafka con funzione comune, ma che in realtà non compone un container reale a se stante. All'interno di questi nodi sono rappresentati gli artefatti che eseguono nel relativo *container*, per esplicitare la presenza dei componenti. Si può inoltre notare che l'ambiente di Apache Kafka è composto da un *cluster* composto da due servizi *Broker* e tre servizi *Zookeeper*, allo scopo di simulare un caso d'uso reale in cui i diversi componenti sono distribuiti in sistemi indipendenti e garantiscono l'affidabilità dello *streaming* di eventi.

UML$_a$ *component diagram*

Allo scopo di riassumere elegantemente i vari componenti del sistema ho elaborato un UML$_a$ *component diagram*.

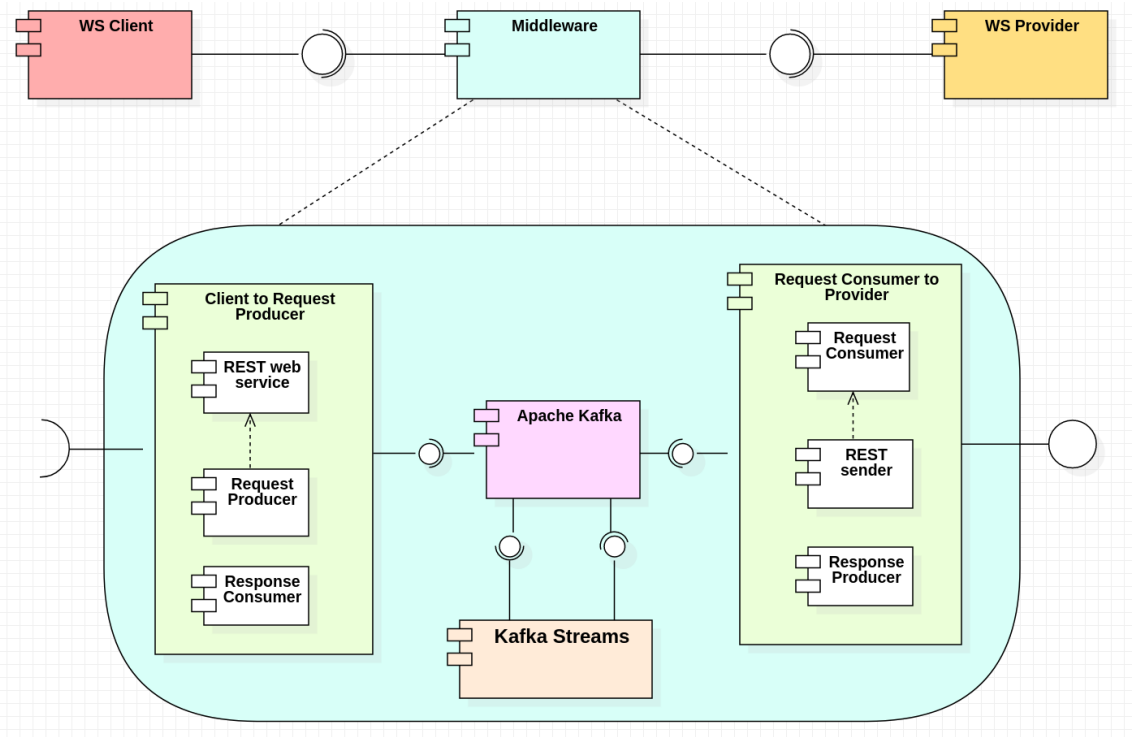

Figura 3.6: UML$_a$ *component diagram* per la reingegnerizzazione del flusso asincrono (protetto)
Fonte: elaborazione personale

3.5 *Setup* dell'ambiente di lavoro

3.6 Sviluppo

3.7 Collaudo

Confronto con i requisiti posti nel piano di lavoro, valutazione riguardo i risultati raggiunti, quelli non raggiunti e i possibili sviluppi futuri

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. 4, 12, 18

CLI Command Line Interface. 22

COBRA Common Object Request Broker Architecture. 10

EAI *Enterprise Application Integration*$_g$. 1, 4, 7, 9–11, 14, 15, 17, 19, 21, 22, 25, 26

EDA Event Driven Architecture. 4, 16, 20

EJB Enterprise Java Bean. 10

ICT Information and Communication Technologies. 5, 19

J2EE Java 2 Platform Enterprise Edition. 10

JMS Java Message Service. 10

JSON JavaScript Object Notation. 27, 28

MOM Message Oriented Middleware. 10

O-O Object Oriented. 10

OS Operating System. 12

P2P Point To Point. 4, 15–17

REST REpresentational State Transfer. 23, 27

SOA *Service Oriented Architecture*$_g$. 22, 26

SOAP Simple Object Access Protocol. 26

SSO Single Sign On. 11

UC Use Case. 27, 28

UML Unified Modeling Language. 10, 27–31

VM Virtual Machine. 4, 13, 14

WS Web Service. 27

WSDL Web Service Description Language. 26

XML eXtensible Markup Language. 26

XSD XML Schema Definition. 26

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 . 10, 15, 20, 26, 33

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 . 22, 26, 33

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/.