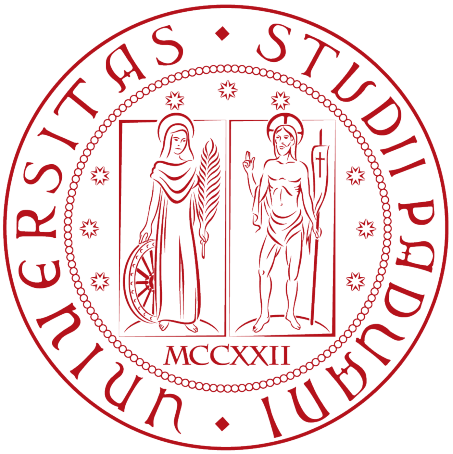
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Una soluzione Data Lake in Cloud per una piattaforma dati cliente

Tesi di laurea

Relatore

Prof.Tullio Vardanega

Laureando

Pietro Marcatti

Anno Accademico 2021-2022

Sommario

Il presente documento descrive il lavoro che ho svolto durante il periodo di *stage*, della durata di circa trecentoventi ore, presso l'azienda Patchai S.r.l. Gli obbiettivi dello *stage* erano la progettazione e la prototipazione di una soluzione *Data Lake* in *Cloud* da associare ad una piattaforma dati cliente.
In primo luogo mi è stato richiesto di condurre un'analisi esplorativa delle tecnologie utilizzabili per la realizzazione della piattaforma e del sistema di acquisizione ed elaborazione dati. In secondo luogo mi è stato richiesta l'implementazione della piattaforma, in via prototipale, con le tecnologie selezionate nella precedente fase.

Convenzioni Tipografiche

Riguardo la stesura del testo, relativamente al documento ho adottato le seguenti convenzioni tipografiche:

* Gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati, li ho definiti nel glossario, situato alla fine del presente documento;
* I termini in lingua diversa dall'italiano li ho scritti in *corsivo*
* Per riferirmi ad una sezione del documento ho anteposto il simbolo '§' al numero della stessa.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

L'Azienda Ospitante

1.1 Patchai

Patchai è una giovane *start-up* fondata nel 2018, con l'intento di rivoluzionare la ricerca clinica mettendo al centro il paziente e le tecnologie all'avanguardia. Patchai si rivolge ad aziende nel mondo della cura del paziente e i suoi rapporti sono quindi *Business to Business (B2B)*. Questo significa che la sua clientela è costituita da aziende le quali acquistano il prodotto per renderlo disponibile agli utenti finali. Patchai vende *Software as a Service (SaaS)*: la licenza *software* è l'oggetto della compravendita, questa è venduta come servizio, e il prodotto viene creato su misura per il singolo cliente. Patchai offre due categorie di SaaS: la prima a supporto di *trial* clinici e la seconda a supporto di pazienti affetti da patologie croniche in contesto slegato dallo sviluppo di un nuovo farmaco o dalla partecipazione ad un protocollo medico. Nel primo caso il ciclo di vendita corrisponde all'intera durata del *trial* clinico per cui il prodotto è utilizzato; nel secondo caso le case farmaceutiche e aziende nel campo biomedico acquistano la licenza per poter rilasciare il *software* ai loro medici e pazienti. La missione dell'azienda è di facilitare e avanzare la ricerca per migliorare l'impatto positivo dei sistemi sanitari, dei medici e delle aziende farmaceutiche sui pazienti. Come *start-up*, Patchai ha saputo aggiudicarsi finanziamenti per la sua crescita provenienti da alcuni tra i più famosi acceleratori, tra i quali Bioupper ed altri fondi di *venture capital* del settore, fino alla sua recente acquisizione da parte dell'americana Alira Health alla fine del 2021. Il *team* di sviluppo dell'azienda, che è la *business unit* nella quale sono stato inserito, è composto da una dozzina di dipendenti. Il *team* è particolarmente giovane e sapientemente coordinato da una *leadership* di esperienza. Il clima in azienda è gradevole ed invita ad un confronto sincero e amichevole. Quanto detto si riflette nella gestione autonoma del lavoro da parte dei dipendenti i quali posseggono un buon grado di libertà senza mai sentirsi trascurati dal *management*.

1.2 Processi Aziendali

1.2.1 Metodologie di Sviluppo

Lo sviluppo di un progetto in Patchai inizia con un'approfondita analisi il cui livello di dettaglio e completezza è fondamentale per la buona riuscita dello stesso. Durante questa fase vengono create quante più *User Stories* possibile a partire dagli obiettivi definiti dalla *business unit* legata al prodotto. Le *User Stories* corrispondono a spiegazioni informali e generali di una funzionalità *software* desiderata dal punto di vista dell'utente finale. Gli obiettivi nascono dai desideri del cliente nel caso di un prodotto su misura, o sono espressione di richieste che il mercato rivolge nei confronti di un prodotto come quello che sta venendo sviluppato. Lo sviluppo dei progetti in Patchai avviene secondo un modello Agile, nella fattispecie l'azienda adopera il *framework* Scrum.

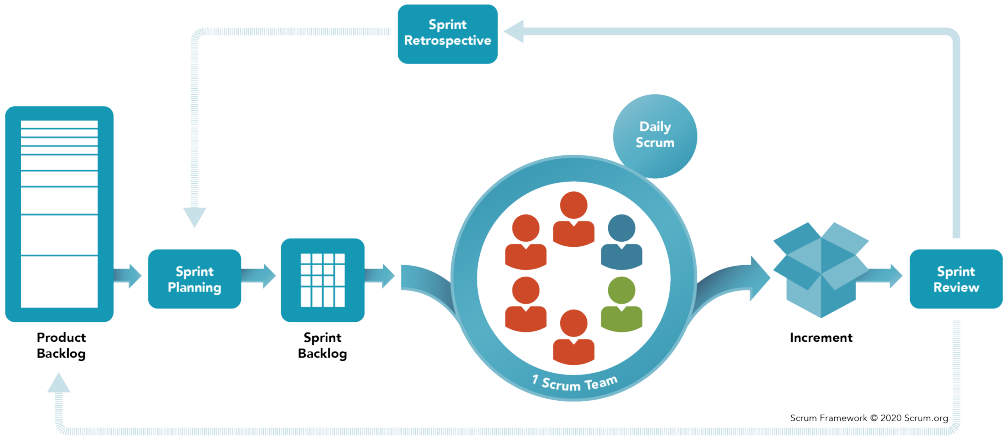


Figura 1.1: Schema di flusso di un progetto con un modello di sviluppo Scrum.

Fonte: www.scrum.org.

Scorrendo quanto riportato in figura 1.2 a partire da sinistra, dai requisiti individuati viene costruito il *Product Backlog*, ovvero una raccolta di task da svolgere per realizzare il prodotto. Quest'ultimo è alla base della pianificazione per gli *Sprint* per tutta la durata del progetto. L'organizzazione del lavoro coinvolge l'intero *team* il quale fissa un obiettivo da raggiungere e divide il lavoro tra le diverse componenti. Terminato questo rituale ha inizio lo *Sprint* che ha sempre la medesima durata di due settimane per il progetto nel quale mi sono inserito. Ogni giorno il *team* effettua la *Daily Scrum* per discutere dei recenti avanzamenti, prossimi passi ed eventuali ostacoli che devono essere rimossi. Al termine di ogni *Sprint* viene fatta una *review*, coinvolgendo tutti gli *stakeholders* del progetto, durante la quale vengono mostrati i progressi fatti sul prodotto ed è possibile ricevere il *feedback* dei presenti. Infine, al termine della *Sprint review*, segue la *Sprint retrospective* nella quale il *team*, a porte chiuse, stabilisce quali pratiche messe in atto durante lo *Sprint* abbiano realmente funzionato, e quindi meritino di venire replicate, e quali invece si siano rivelate inefficaci e vadano abbandonate.

1.2.2 Strumenti di Documentazione e *Issue Tracking*

In azienda c'è una radicata cultura per la documentazione del materiale prodotto, la quale viene costantemente aggiornata mentre il progetto procede e le necessità tecniche e funzionali diventano sempre più chiare. Detto questo, il *team* non dedica più tempo e parole del necessario alla documentazione che deve essere sempre di facile e veloce consultazione. Per riuscire in questo il team utilizza Confluence di Atlassian come punto di raccolta per la sua documentazione. Questa viene organizzata in sezioni e pagine a seconda della necessità e con l'ottica di minimizzare il tempo richiesto per reperire le informazioni necessarie.
Per quando riguarda l'*Issue Tracking*, il team adotta le *board* di Jira, utilizzandole per la gestione dei loro *ticket* e l'organizzazione delle attività durante gli *Sprint*. Il *team* ha arricchito le proprie *board* con alcuni stati aggiuntivi in cui è possibile per un *issue* trovarsi. Questo dà maggiore chiarezza sullo stato delle attività legate a quell'elemento con un aiuto visivo dato dall'incolonnamento dei *ticket*.

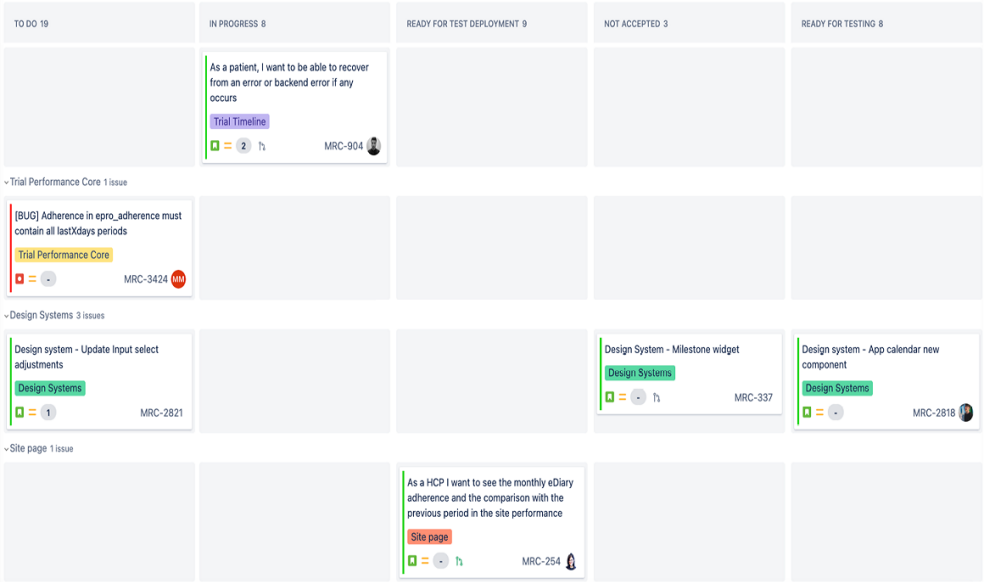


Figura 1.2: *Dashboard* per la gestione di Scrum in Jira.

1.3 Tecnologie Utilizzate

1.3.1 AWS

Le tecnologie principali utilizzate per tutta la durata del progetto sono gli Amazon Web Services (AWS). Questi compongono una piattaforma *Cloud* estremamente completa con oltre 200 servizi offerti fortemente competitivi dal punto di vista dei costi grazie all'economia di scala. L'azienda utilizza già ampiamente questa tecnologia, primariamente per offrire le API per le applicazioni, ovvero l'intermediario *software* che permette a due applicativi di comunicare tramite un protocollo comune, e gestire i *database* di produzione. I servizi specifici che ho utilizzato per la realizzazione del progetto di *stage* non sono tra quelli che l'azienda utilizzava prima dell'attivazione del

progetto. Il mio compito, nella fase esplorativa e di studio, è stato anche di valutare se i servizi disponibili fossero adatti al progetto e valesse la pena iniziare ad utilizzarli. In particolare alcuni dei servizi che ho usato sono i seguenti:

DMS

DMS è l'acronimo per il servizio di *Database Migration Service* ed è un servizio *Cloud* che consente di migrare *database* relazionali, *database* NoSQL, molti altri tipi di *datastore* e *data warehouse*. Quest'ultimi sono sistemi di *data management* pensati per il supporto al *business intelligence*. Il servizio richiede la definizione di almeno due *endpoint* associati ai capi della comunicazione e di un'istanza di replicazione. Un'istanza individua l'*hardware* che supporta il processo specificato dalla *task* di replicazione come illustrato in figura 1.3.

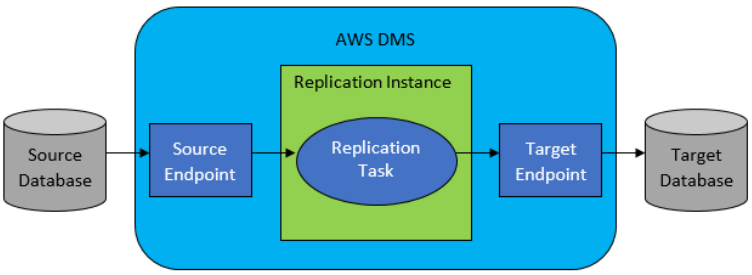


Figura 1.3: Diagramma di un sistema DMS.

Fonte: docs.aws.amazon.com/dms/latest/userguide.

Il servizio di DMS è formulato con una politica "paga quel che usi", ciò dà all'azienda la sicurezza di star pagando un servizio solo nel limite delle risorse che sta effettivamente sfruttando. Utilizzare il servizio di DMS elimina la responsabilità di dover mantenere l'infrastruttura di migrazione attiva ed efficiente, affidandosi ad AWS per questo. Nel caso specifico del progetto svolto per lo *stage*, ho utilizzato il servizio per migrare i dati di un *database* relazionale PostgresSQL ad un *bucket* S3, l'unità base stato dell'arte per l'archiviazione *Cloud* su AWS.

Lambda

Lambda è un servizio di calcolo per l'esecuzione di codice senza doversi preoccupare di predisporre o dover gestire *server*. Il codice viene eseguito dal servizio Lambda su un'infrastruttura ad altissima disponibilità che si occupa dell'amministrazione delle risorse necessarie, l'aggiornamento *software*, il calcolo della capacità richiesta e molto altro. Il servizio di Lambda accetta moltissimi tra i linguaggi di programmazione più utilizzati e permette all'utente di organizzare il suo codice in funzioni il cui costo è legato all'utilizzo. Il servizio Lambda permette inoltre di creare un'altra tipologia di risorsa ovvero i Lambda *Layer*, questi contengono delle librerie precompilate, delle *runtimes* personalizzate o altre dipendenze. I *Layer* esistono indipendentemente dalle Funzioni, che nella loro configurazione possono farsi carico di fino a cinque *Layer*. Esempi di utilizzo sono l'abilitazione di una Funzione all'utilizzo di una libreria esterna o evitare la ripetizione di codice comune a più Funzioni. È possibile specificare questa relazione in qualsiasi momento dalla pagina di configurazione della Funzione e, se non viene in altro modo specificato, persiste nel progredire di versione di entrambe

le risorse. Nel progetto di *stage*, ho utilizzato le funzioni Lambda per l'esecuzione di codice Python per la trasformazione di dati. Le ho anche associate a dei *Layers* per integrarle con le librerie di *data science* necessarie.

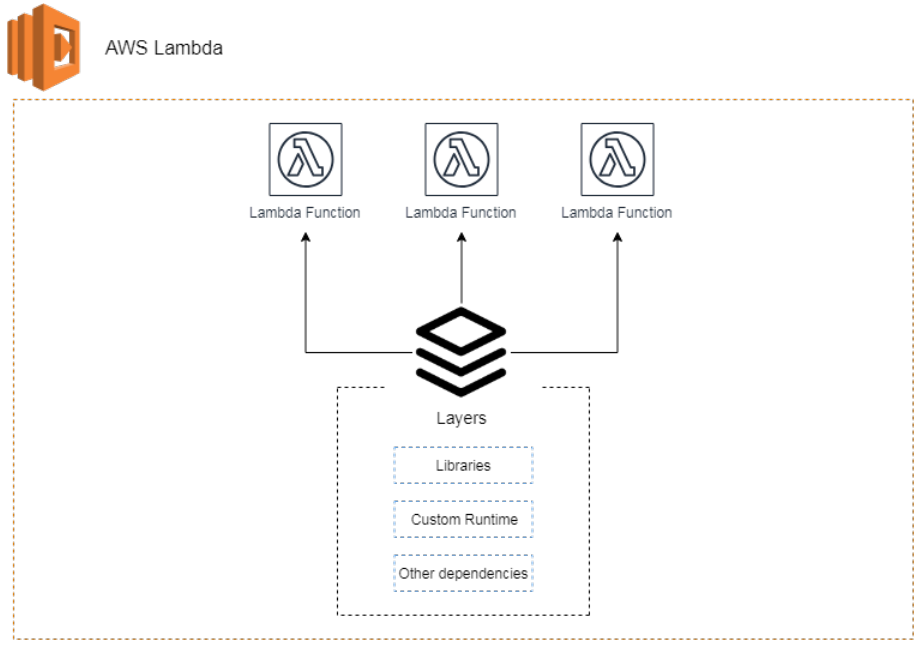


Figura 1.4: Più Funzioni possono prendere in carico e utilizzare lo stesso *Layer*.

Fonte: lumigo.io/aws-serverless-ecosystem/aws-lambda-layers.

Lake Formation

Il servizio di *Lake Formation* è il ponte di comando per la definizione di tutti gli aspetti necessari alla creazione di un *Data Lake*. Questo include la specifica di *bucket* S3 da utilizzare come contenitori per i dati, la definizione delle *policy* e ruoli per accedere agli stessi e la creazione di *data catalog*, ovvero descrizioni dei *data set* contenuti nel *Data Lake*. Ho usato il servizio per coordinare i permessi e i ruoli necessari ai servizi e agli utenti che agiscono nel *Data Lake*, infine per l'identificazione di *bucket* come sede fisica dei dati gestiti.

1.3.2 Terraform

Terraform è una tecnologia per la definizione di *Infrastructure as Code (IaC)*, quest'ultima è uno strumento per gestire un'infrastruttura tramite dei *file* di configurazione invece che un'interfaccia utente visuale. Un IaC permette di realizzare e modificare l'infrastruttura in una maniera ripetibile e sicura, definendo la configurazione delle risorse in modo che possano essere versionate, riutilizzate e condivise.
Terraform in particolare utilizza una definizione delle risorse intellegibile all'interno di *file* di configurazione dichiarativi. Gestisce l'intero ciclo di vita dell'infrastruttura dalla sua istanziazione alla sua distruzione. I vantaggi dell'utilizzo di una tecnologia come Terraform sono i seguenti:

* Terraform può gestire infrastrutture su più piattaforme *Cloud*;
* La dichiarazione intellegibile delle risorse velocizza la loro scrittura;
* Terraform mantiene uno stato dell'infrastruttura permettendo di monitorare i cambiamenti;
* Terraform può essere versionato e condiviso per collaborare all'infrastruttura.

Per comunicare con le API delle piattaforme *Cloud*, Terraform utilizza quelli che chiama *provider*. Grazie ad HashiCorp, l'azienda dietro lo sviluppo di Terraform, e la sua *community*, sono disponibili più di 1000 *provider*. Questo assicura una compabitilità molto elevata con i servizi più disparati oltre, ovviamente, a quelli più popolari come AWS.

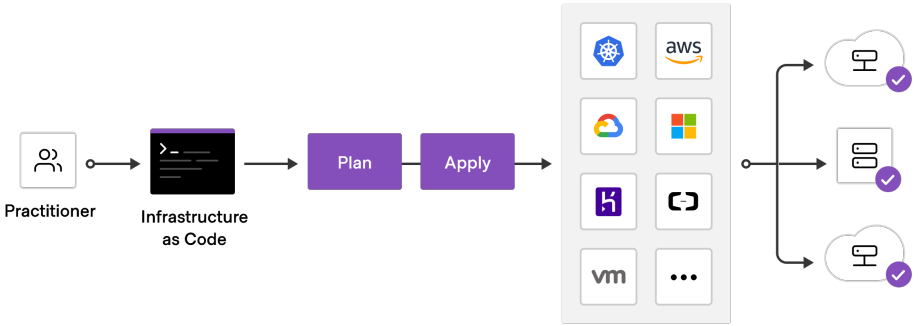


Figura 1.5: *Plan* e *Apply* sono le due direttive Terraform principali nel processo di modifica dell'infrastruttura.

Fonte: learn.hashicorp.com/tutorials/terraform/infrastructure-as-code.

Terraform è una tecnologia che l'azienda utilizza quotidianamente ed è alla base del lavoro di ingegneri *backend* e *platform*. Patchai la utilizza per aggiornare costantemente la definizione dell'infrastruttura *Cloud* affinché rifletta le modifiche, le aggiunte e le rimozioni alla collezione di risorse gestite, primariamente quelle su AWS. Personalmente ho impiegato Terraform nel progetto di *stage* per la gestione del complesso di servizi AWS che ho messo insieme per creare l'infrastruttura atta all'acquisizione, stoccaggio e trasformazione dei dati. Questo mi ha reso possibile automatizzare il processo di creazione di risorse eliminando l'elemento umano una volta dichiarata l'infrastruttura. Ciò consente di replicare i servizi in diverse zone di disponibilità a seconda della necessità dell'azienda, sia questa di natura cautelare o economica.

1.3.3 Python

Python è un linguaggio di programmazione ad alto livello, interpretato e *open-source* ed è uno dei migliori linguaggi utilizzati dai *data scientist* per i loro progetti. Python offre ottime funzionalità per lavorare con problemi matematici e statistici anche grazie alla ricca scelta di librerie disponibili. Alcune delle ragioni per cui Python è così popolare sono:

* La sua facilità di utilizzo e installazione;
* Una sintassi semplice e accessibile anche a coloro che non hanno un *background* in informatica;
* È particolarmente adatto per i prototipi veloci;
* La sua popolarità e diffusione assicurano un grande supporto e una libreria *standard* in continuo aggiornamento.

Durante il progetto di *stage* ho impiegato tre librerie Python specifiche del mondo del *data science*:

Pandas è una libreria per la manipolazione dati e l'analisi, offre funzioni utilissime per manipolare grandi quantità di dati strutturati. Tra le altre cose, Pandas offre strutture dati estremamente efficienti, ideali per fare *data wrangling*, ovvero il processo di preparazione e pulizia dei dati antecedente alla loro analisi. Tra di loro quelle che ho utilizzato nel progetto sono *Series* e *DataFrames* per contenere, rispettivamente, dati mono-dimensionali e multi-dimensionali.

Numpy è una libreria per la manipolazione di grandi *array* dimensionali ed offre molteplici operazioni da applicare su *n-arrays* e matrici. Uno dei punti di forza di Numpy è l'uso di operazioni vettorializzate per *array* che offrono *performance* migliori per molti casi applicativi. Per il progetto di *stage*, ho utilizzato Numpy per i suoi *array* e i metodi che offre su di loro, permettendo di eseguire calcoli vettorializzati efficientemente.

Scipy è una libreria composta da moduli dedicati a funzionalità specifiche come l'algebra lineare, l'integrazione, l'interpolazione, trasformate di Furier e molte altre. In particolare, durante il progetto di *stage*, ho utilizzato il modulo per l'interpolazione che ho sfruttato per effettuare delle regressioni lineari.

Questa tecnologia è utilizzata giornalmente dal *team* di *data scientist* e le librerie menzionate sono alla base del lavoro che ogni giorno producono. Questa è una tecnologia ben consolidata per il *team* ed è stata scelta, quindi, anche per la familiarità e padronanza che i dipendenti possiedono con Python.

1.3.4 Tecnologie e Architettura

Non ho scelto queste tecnologie per utilizzarle a compartimenti stagni, esse infatti collaborano per definire l'architettura del progetto. AWS è la tecnologia principale alla base di questa architettura e per questo motivo si incontra con le altre tecnologie in ogni area della soluzione creata. In particolare il servizio di migrazione DMS è stato configurato per migrare i dati del *database* relazionale di produzione in dei *bucket* S3. I *bucket* erano poi identificati come locazioni di *storage* dati per il *Data Lake* creato grazie allo strumento di *Lake Formation*. Il servizio Lambda è stato a sua volta integrato con

Lake Formation, in quanto le Funzioni Lambda sono agenti che necessitano di permessi per accedere e trasformare i dati presenti nelle locazioni apposite del *Data Lake*.
Python è il linguaggio di programmazione scelto per la specifica delle istruzioni di trasformazione e manipolazione dati tramite le Funzioni Lambda. Le due tecnologie sono interconnesse in quanto una Funzione Lambda contiene al suo interno *file* Python con le istruzioni e procedure per elaborare i dati.
Infine, Terraform è la tecnologia trasversale che permette la coordinazione di tutte le altre. Terraform è necessaria per la creazione e distruzione automatizzata delle risorse, siano queste Funzioni Lambda, il codice Python da associarvi o i ruoli e permessi relativi a *Lake Formation*. Terraform è la tecnologia collante del pacchetto utilizzato per il progetto di *stage*, proprio per la sua elevatissima compatibilità.

1.4 Mercato e Innovazione

Il mercato all'interno del quale si inserisce Patchai è quello delle aziende per la cura del paziente e dei *trial* clinici. In particolare il settore di interesse per il prodotto associato al progetto di *stage* è quello dei *Patient Reported Outcom (PRO)* nei *trial* clinici. Questi sono dei questionari dalla cadenza prefissata che i pazienti di *trial* clinici compilano per fornire i dati necessari a portare avanti la ricerca. La loro variante specifica trattata dal progetto è quella digitale: ePRO. Il mercato per questo settore è in forte crescita e Patchai vi è entrato a gamba tesa introducendo dei prodotti con un elevato contenuto tecnologico innovativo. Infatti l'azienda è pioniera nel rilascio di una piattaforma cognitiva che integri l'analisi predittiva di algoritmi di *machine learning* ai dati generati dai *trial* clinici. Lo scopo è migliorare il livello di ingaggio dei pazienti e la qualità dei dati a disposizione dei ricercatori. Patchai sta inoltre innovando il settore introducendo nei suoi prodotti un'esperienza utente conversazionale basata su *chatbot* che aiutino a monitorare la salute e il benessere dei pazienti. Per farlo introducono un elemento di gioco ed empatia al fine di migliorare l'esperienza per l'utente. Per fare innovazione, Patchai mira ad attrarre nella sua forza lavoro esperti di *machine learning* e *data analysis* ma anche esperti di *cognitive beahviour* e crede che gli *stage* in collaborazione con gli atenei siano il punto di incontro ideale per far fruttare una relazione proficua.

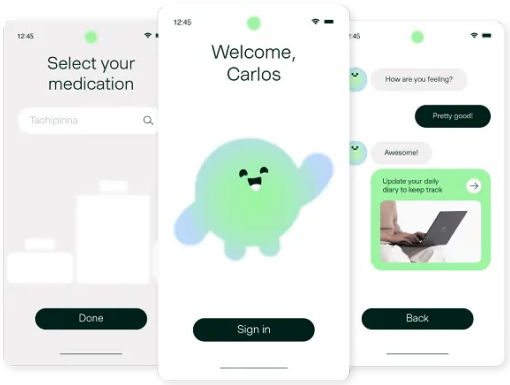


Figura 1.6: Parte dell'innovazione di Patchai è l'esperienza utente conversazionale.

Fonte: www.patchai.io/about-us.

Capitolo 2

Progetto Proposto

2.1 Rapporto tra Azienda e Stage

In conclusione del precedente capitolo ho descritto del rapporto che l'azienda ha con l'innovazione. Tale rapporto può essere riassunto come il bisogno di Patchai di attrarre personale capace di integrare il *software* prodotto con tecnologie all'avanguardia. Gli studenti al termine del proprio percorso di studi sono il frutto del sistema accademico e della ricerca, impersonano il desiderio di innovazione e progresso tecnologico. Per questo Patchai si è dimostrata entusiasta di accogliere tra i suoi dipendenti dei tirocinanti per perseguire questo desiderio di innovare. Ciò permette anche di fornire formazione pratica e un'esperienza diretta nel mondo del lavoro. Quello in cui sono stato coinvolto non è infatti il primo *stage* che Patchai organizza assieme agli atenei italiani. Non si tratta di una coincidenza, bensì di una forte convinzione, da parte di Patchai, che offrire uno *stage* ad uno studente costituisca un momento di crescita per entrambe le parti. Nella mia permanenza ho sempre ricevuto l'assistenza di cui ho necessitato e sono stato messo a mio agio. I compiti che mi sono stati assegnati erano commisurati alle mie conoscenze iniziali e allo sforzo di apprendimento che è ragionevole richiedere in un contesto simile. Questo a dimostrazione che Patchai dedica tutta l'attenzione possibile allo *stage* in corso, in prima battuta affiancando un *tutor* capace e disponibile a seguire lo studente nel suo lavoro che gli permetta contemporaneamente di dimostrarsi autonomo e proattivo. Secondariamente integrando lo studente all'interno dell'organizzazione, coinvolgendolo nei rituali aziendali, dandogli l'occasione per mettere in pratica le proprie abilità e conoscenze e fornendogli un *feedback* tecnico e professionale.

2.2 Bisogno dell'Azienda

L'utilizzo dei dati, in particolare quelli agglomerati in enormi collezioni o *Big Data*, è diventato sempre più centrale per le aziende che intendono perseguire un approccio *data driven*, ovvero guidato, motivato dai dati. Le aziende di tutto il mondo hanno iniziato da anni una corsa frenetica alla definizione di una *Data Strategy*, ovvero un protocollo strutturato per l'acquisizione, l'organizzazione, il trattamento e la comunicazione dei dati a supporto degli obiettivi aziendali. Patchai non fa eccezione ed è coinvolta in questo fenomeno. A maggior ragione anche Patchai, che è un'organizzazione giovane, desidera innovare i suoi prodotti ed è impegnata nella definizione di una *Data Strategy* per il suo prodotto *software* nel settore ePRO. Al momento, Patchai è impegnata

nella creazione di un prodotto *software* costituito da un'applicazione *mobile* pensata per i pazienti di *trial* clinici e un'applicazione *web* per medici e finanziatori degli stessi. Il *team* punta a definire una *Data Strategy* per questo progetto affinché i dati generati dagli utenti possano essere raccolti e gli possa venire data una forma capace di evidenziare tendenze e variazioni. L'implementazione di una tale *Data Strategy* è cruciale per il perseguimento della politca *data driven* che l'azienda ha abbracciato e costituisce un *focus* primario per il *team* coinvolto nello sviluppo del prodotto. In particolare, la *Data Strategy* che l'azienda ha deciso di predisporre è identificata da una *Customer Data Platform (CDP)*, o piattaforma per la gestione dei dati dei clienti. Questa risiede e si alimenta dai dati collezionati in un *Data Lake*, ovvero un punto di raccolta centralizzato, per dati strutturati e non, su qualsiasi scala. Questi due concetti sono centrali nella *Data Strategy* e nel progetto, di conseguenza lo saranno in questa relazione. Durante l'attività di *stage* mi è stato chiesto di effettuare prima uno studio sulle tecnologie disponibili per la realizzazione del *Data Lake* e della *CDP*. Nel farlo ho prodotto del materiale documentale nel quale motivavo le scelte effettuate e spiegavo per quale funzionalità ogni tecnologia era necessaria, oltre che la sua particolare configurazione utilizzata. Mi è poi stato chiesto di iniziare a creare le due componenti della *Data Strategy*. Data la natura esplorativa del mio lavoro, le caratteristiche tecniche e di *performance* attese variavano con il susseguirsi delle revisioni di avanzamento settimanali.

Figura 2.1: La definizione di una *Data Strategy* può permettere alle aziende di guadagnare dai dati.

Fonte: www.instinctools.com/blog/why-every-company-needs-a-big-data-strategy-and-how-to-build-it

2.3 Scopo del Progetto

Patchai sta sviluppando un Minimum Viable Product (MVP) per la commercializzazione del suo prodotto per il settore *ePRO* con un campo di applicazione nei *trial* clinici. L'obiettivo del progetto di *stage* è parallelo a questo sviluppo, e come già anticipato, consiste nel realizzare un'infrastruttra *Cloud*, costituita da un *Data Lake* con una *CDP* associata. La soluzione deve gestire l'approvvigionamento, lo stoccaggio, la trasfor-

mazione, l'accesso e la visualizzazione dei dati generati dagli utenti che utilizzeranno l'applicazione. La natura di questo progetto è anche esplorativa, in quanto l'azienda non ha mai sviluppato un prodotto simile in precedenza. In quest'occasione intende studiare le proprie necessità, definire una metodologia di lavoro e monitorare i costi di un'eventuale espansione delle risorse dedicate all'infrastruttura.

2.4 Obiettivi Aziendali

Gli obiettivi aziendali dello *stage* erano:

Produzione di una descrizione tecnica: Una pagina di documentazione che descrivesse l'infrastruttura di servizi AWS e la loro configurazione, una *roadmap* per la loro predisposizione e collegamento.

Studio sulle tecnologie da adottare: Lo studio delle tecnologie disponibili per la realizzazione del progetto considerando il costo di adozione, ed eventualmente la necessità di fare formazione.

Realizzazione del prodotto: Il complesso di servizi impiegati e i meccanismi di comunicazione tra di loro, i punti di accesso all'infrastruttura e i *file* di configurazione, il tutto basato su quanto descritto nella documentazione redatta. In particolare l'infrastruttura doveva gestire queste cinque operazioni fondamentali:

1. **Approvvigionamento**: il sistema deve gestire l'approvvigionamento dati da più fonti, quelle di primaria importanza sono gli inserimenti e aggiornamenti di righe del *database* relazionale di produzione. Una seconda fonte sono gli eventi generati dall'utente all'interno dell'applicazione.
2. **Stoccaggio**: i dati devono poter risiedere a lungo termine, devono essere catalogati in maniera coerente e tale da rendere immediata la consultazione. É importante tenere a mente il costo delle soluzioni nella progettazione della soluzione.
3. **Trasformazione**: il sistema deve prevedere un meccanismo automatizzato di trasformazione dei dati a partire da procedure create dai *data scientist*.
4. **Accesso ai dati**: il sistema deve fornire un meccanismo sicuro e affidabile al personale che necessita di accedere ai dati per monitorarli e valutare la creazione di nuove trasformazioni.
5. **Visualizzazione**: i dati raffinati prodotti dal sistema devono essere facilmente accessibili da parte di sistemi di visualizzazione sotto forma di grafici che compongano dei cruscotti.

2.5 Vincoli del Progetto

2.5.1 Tecnologie

AWS

Per la realizzazione del progetto doveva essere utilizzato AWS. Come spiegato in precedenza, AWS è un popolare gruppo di servizi erogati su *Cloud* che permette, integrando le sue componenti, di realizzare architetture estremamente complesse e scalabili. Patchai ha scelto AWS perché è una tecnologia già ampiamente in utilizzo dal *team*. L'integrazione con i sistemi già attivi sarebbe risultata più naturale e facilmente realizzabile da me, in quanto non possedevo precedente esperienza con altre tecnologie simili. Ulteriore motivo per la scelta di AWS è la sua scalabilità: Patchai sta per commercializzare il suo prodotto e necessita di poter adattare la potenza a disposizione della sua infrastruttura a seconda del carico generato dai clienti. Nel contesto di AWS, al concetto di scalabilità viene spesso associata la gamma di macchine virtuali da assegnare a determinati servizi affinché possano effettuare le operazioni che gli sono richieste. Queste macchine virtuali si differenziano per la quantità e qualità di risorse *hardware* a loro disposizione.

General Purpose	Compute Optimised	Memory Optimised	Accelerated Computing	Storage Optimised
A1 ARM based core and custom silicon	C4 Compute - CPU intensive apps and DBs	R4 RAM - Memory intensive apps and DB's	P2 Processing optimised- Machine Learning	H1 High Disk Throughput - Big data clusters
T2 Tiny - Web servers and small DBs		X1 Xtreme RAM - For SAP/Spark	G3 Graphics Intensive - Video and streaming	I3 IOPS - NoSQL DBs
M4 Main - App servers and general purpose		z1d High Compute and High Memory - Gaming	F1 Field Programmable - Hardware acceleration	D2 Dense Storage - Data Warehousing

Figura 2.2: AWS offre macchine virtuali con configurazioni calibrate per casi d'uso specifici.

Fonte: www.helenanderson.co.nz/which-ec2-right-for-you.

Python

La tecnologia scelta per trasformare i dati ricevuti dal sistema al fine di estrarne metriche e *trend* è Python. Python, in quanto linguaggio di programmazione ampiamente diffuso per la lavorazione dati, è tra quelli meglio conosciuti e padroneggiati dal *team* di *data scientist* che lo ha indicato come vincolo. In questo caso ho utilizzato Python, in abbinamento alle librerie menzionate in §1.3.3, per dare una forma tabellare ai dati in arrivo, effettuare calcoli vettorializzati estraendo nuove colonne e combinandone altre.

2.5.2 Tempistiche

Ho iniziato lo *stage* il 6 maggio e l'ho terminato il 1 luglio 2022, per un totale di trecentoventi ore. Il carico di lavoro giornaliero è sempre stato di otto ore, in giornate lavorative che iniziavano alle 9 del mattino e finivano alle 18 di sera, con l'ora dalle 13 alle 14 libera per la pausa pranzo. Le ore sono state preventivamente ripartite così come riportato nella seguente tabella:

Attività	**Ore di lavoro**
Onboarding	16
Studio dei concetti di *Data Lake*, CDP e *Data Architecture*	40
Comprensione del *data model* di Patchai	40
Studio dei servizi AWS e pratica *hands-on*	80
Progettazione della soluzione	40
Sviluppo della soluzione	40
Calcolo di metriche sui dati	24
Testing della soluzione sviluppata	40

Tabella 2.1: Pianificazione ore *stage*

2.6 Motivazioni Personali

Sono venuto a conoscenza di Patchai e del progetto *stage* che offriva durante l'evento STAGE-IT promosso da Assindustria Venetocentro in collaborazione con l'Università di Padova. L'evento è nato per favorire l'incontro tra aziende, con progetti innovativi in ambito IT, e studenti dei corsi di laurea in Informatica, Ingegneria Informatica e Statistica. Nella ricerca dell'attività di *stage* puntavo a un ambiente lavorativo in cui poter esplorare e sperimentare con tecnologie di rilievo nel mercato odierno. Questo perché ritengo che l'esperienza maturata con esse sia un fattore di crescita personale a livello formativo e lavorativo.
Altro fattore fondamentale che mi ha guidato nella ricerca era la possibilità di avere delle responsabilità e di conseguenza un impatto reale, in quanto credo che in queste circostanze si maturi maggiormente. Infine cercavo un ambiente giovane e innovativo che potesse mettermi a confronto con personale qualificato e proattivo. La proposta di Patchai, meglio di tutte le altre che ho scrutinato, esemplificava quanto io cercassi in un'esperienza di *stage*. Il supporto fornito dalla mia *tutor* e dal *team leader*, che ho avuto modo di conoscere prima dell'inizio del tirocinio, era del tipo e della misura che, secondo me, meglio alimenta la crescita a livello professionale. In conclusione, le motivazioni principali per cui ho scelto proprio Patchai per il mio *stage* sono:

* Durante entrambi i colloqui che ho fatto con la *tutor* aziendale, le responsabili per le risorse umane e l'acquisizione talenti e il capo dipartimento per lo sviluppo *software* ho trovato delle persone simpatiche, giovani e alla mano. Questo corrispondeva a diverse delle qualità che desideravo in uno *stage*.
* Il progetto di *stage* proposto faceva utilizzo di tecnologie *Cloud*, estremamente odierne e richieste. Questo voleva dire la possibilità di lavorare con tecnologie innovative e la cui conoscenza è spendibile nel mercato del lavoro.
* L'azienda è estremamente flessibile e consente ai suoi dipendenti il lavoro telematico. Questo aspetto mi avrebbe permesso di ritagliare più tempo, altrimenti speso per il trasporto da e per l'ufficio, da dedicare ai miei studi.

Gli obiettivi personali che mi ero prefissato prima di iniziare lo *stage* erano:

Obiettivo Personale	**Motivo**
Acquisire esperienza progettuale con tecnologie *Cloud* e *IaC*	Per imparare concretamente, facendo pratica, come valutare la scelta di tecnologie e configurazioni da utilizzare. Inoltre per riuscire a pianificare, idealmente con alto livello di dettaglio, la configurazione delle tecnologie e saperle utilizzare insieme mantenendo un buon livello di efficienza e efficacia.
Consolidare le mie conoscenze Python	Per migliorare la mia comprensione del linguaggio e riuscire ad utilizzarlo in maniera efficiente oltre che efficace, seguendo le *best practice* del caso. In particolar modo approfondire le librerie e le modalità di utilizzo del linguaggio nel contesto dell'analisi dei dati.
Acquisire esperienza nel mondo del lavoro	Per arricchire il mio bagaglio di *soft skill* interpersonali e sociali spendibili nel mondo del lavoro e contemporaneamente fare delle esperienze reali per iniziare ad espandere e caratterizzare il mio profilo di carriera.

Tabella 2.2: Obiettivi personali nello svolgimento dello *stage*

Capitolo 3

Svolgimento del Progetto

3.1 Studio Concettuale e Tecnologico

Nelle prime giornate di *stage* mi sono dedicato allo studio dei concetti e delle tecnologie coinvolte nel progetto che non conoscevo. In particolare mi sono concentrato sui concetti di *Data Lake*, *Customer Data Platform* e *Data Architecture*. Ho organizzato lo studio in maniera autonoma seppure con delle indicazioni iniziali da parte della mia *tutor* la quale mi ha segnalato alcune risorse. Le risorse primarie che ho utilizzato sono state le guide redatte da AWS. In queste guide vengono spiegati i concetti da un punto di vista teorico, spiegandone i casi d'uso e i vantaggi, continuando poi a descrivere come realizzarle concretamente con i servizi disponibili. In abbinamento a queste ho visionato dei contenuti più generali sotto forma di articoli o video divulgativi per ottenere una prima infarinatura sui concetti. Per quanto riguarda le tecnologie, invece, mi sono riferito alle risorse documentali ufficiali e seminari divulgativi ufficiali in formato video. Queste risorse ufficiali sono la documentazione delle classi, delle funzioni esposte dalle librerie per Python. Per Terraform le risorse documentali includevano anche l'uso delle istruzioni a riga di comando. Infine, per i servizi AWS le risorse documentali descrivono la funzionalità e i valori accettati per la loro configurazione. Oltre a queste ho utilizzato anche articoli di terze parti per aspetti specifici che mi erano poco chiari. In abbinamento allo studio autonomo ho condotto degli incontri di confronto e revisione giornalieri con la mia *tutor* durante i quali esponevo eventuali dubbi od ostacoli incontrati e sui quali ci soffermavamo per fare chiarezza. Il risultato dello studio delle tecnologie, e in particolare di AWS, è stato la scelta dei servizi per realizzare l'architettura e la comunicazione tra di essi. Ho preso queste decisioni basandomi sulla necessità di trovare un compromesso tra le prestazioni attese e la scelta di servizi che sono in grado di realizzare quanto desiderato in maniera efficace, tenendo a mente il punto di vista economico. Siccome ho già elencato le tecnologie utilizzate in §1.3, riassumo qui le prestazioni attese dal committente in due punti fondamentali:

* **Replicazione *near-real-time***: All'arrivo di nuovi dati generati dagli utenti nel *database* di produzione questi devono essere replicati nel *Data Lake*. Questo deve avvenire in *near-real-time* per processare ed estrapolare i nuovi valori da aggiungere al cruscotto delle metriche.
* **Accesso ai dati casuale**: La configurazione del servizio di stoccaggio dei dati deve permettere di accedere a qualsiasi *record* in maniera diretta. Questo significa con lo stesso tempo di accesso, senza dover consultare registri o indici.

Ho dedicato l'ultima parte del mio studio al *data model* dell'azienda. Il materiale che l'azienda mi ha fornito per questo compito erano le risorse documentali interne sulla struttura dei dati, di come sono organizzati nel *database*. Ho preso anche visione delle politiche aziendali di raccolta e conservazione dei dati nel rispetto dei regolamenti italiani ed europei sulla *privacy*.

3.2 Analisi dei Requisiti

Nelle prime settimane di progetto mi sono occupato dell'analisi dei requisiti, intrattenendo incontri regolari con il rappresentante del *team*. Lo scopo degli incontri era delimitare chiaramente le aspettative dell'azienda e quindi i requisiti che il progetto doveva soddisfare. Dato il carattere esplorativo del progetto, l'analisi dei requisiti non è stata svolta solamente all'inizio dello stesso. I requisiti venivano estesi, modificati o rimossi a seconda dello stato dei lavori e del grado di soddisfazione del responsabile del *team*, che in questo caso ha svolto il ruolo di committente. Per questa ragione l'azienda non mi ha richiesto una documentazione formale dei requisiti come è più comune. Invece, a partire dalle *User Stories* create durante gli incontri di discussione, abbiamo individuato cinque macro requisiti principali, ognuno di questi è poi suddiviso in molteplici sotto-requisiti. Questi cinque requisiti identificano le funzionalità cardine attese e sono le seguenti:

* **Approvvigionamento**: La soluzione deve prevedere canali *near-real-time* diversificati e specifici di approvvigionamento dati per il *database* e le applicazioni *mobile* e *web*;

* **Stoccaggio**: La soluzione riesce a stoccare tutti i dati in arrivo ed etichettarli affinché sia possibile consultarli in maniera diretta;

* **Trasformazione**: A partire da algoritmi di elaborazione definiti dal personale dell'azienda, la soluzione deve trasformare i dati in arrivo in maniera automatica;

* **Accesso ai dati**: La soluzione deve disporre di punti di accesso sicuri per il personale autorizzato dell'azienda affinché possa ispezionare e manipolare i dati.

* **Visualizzazione**: La soluzione include degli strumenti di visualizzazione dei dati trasformati e raffinati sotto forma di metriche. Queste metiche devono essere consultabili in un cruscotto aggiornato in tempo reale.

Ho quindi proseguito il lavoro di analisi per dettagliare e approfondire i cinque requisiti. Ho espanso la lista di requisiti granulari definendo requisiti più fini, spezzando i primi ed aggiungendone altri. Per fare questo approfondimento ho seguito una serie di passaggi ciclicamente fino a raggiungere una convergenza sui requisiti assieme al rappresentate del *team*:

* **Approfondimento tecnologico**: Partendo dai requisiti definiti fino a quel punto, ho progettato ad alto livello l'architettura tecnologica sottostante;

* **Colloquio con il committente**: Ho spiegato al committente la progettazione fatta, con eventuali limitazioni riscontrate e possibili punti per espanderla. In questa occasione ricevevo il *feedback* del committente il quale proponeva variazioni ai requisiti;

* **Aggiornamento requisiti**: Alla luce del *feedback* ricevuto, aggiornavo la lista di requisiti secondo la mia nuova comprensione degli stessi.

Il risultato di questa attività di approfondimento è stato l'individuazione di 29 requisiti:

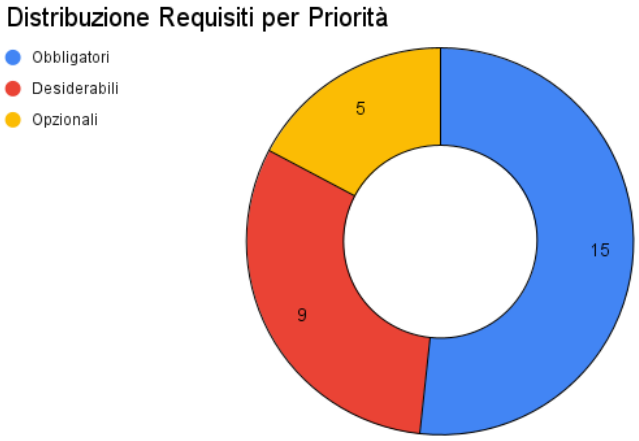


Figura 3.1: Distribuzione requisiti per priorità.

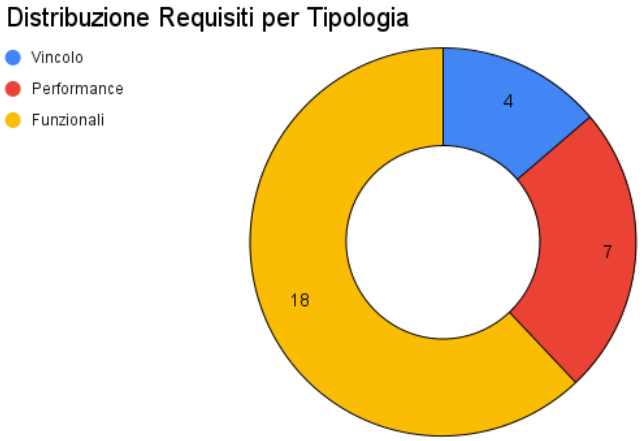


Figura 3.2: Distribuzione requisiti per tipologia.

Per tenere traccia dello stato di avanzamento del progetto svolgevo incontri settimanali con il rappresentante del *team*. In queste occasioni mostravo le funzionalità aggiunte o modificate, ricevendo un *feedback* ed un'eventuale approvazione di quest'ultime. Quando possibile valutavamo anche le *performance* del prodotto. In questa sede ci occupavamo anche di cambiare, in accordo con quanto osservato, lo stato dei *ticket* associati ai requisiti revisionati, in questo modo tracciando il soddisfacimento degli stessi.

3.3 Progettazione

Nell'attività di progettazione ho ragionato su come realizzare concretamente l'infrastruttura affinché questa rispondesse alle necessità dell'azienda. In particolare ho trovato una risposta alle cinque sfide progettuali associate ai corrispondenti requisiti cardine. Nelle sottosezioni a seguire darò una visione dall'alto delle problematiche incontrate e delle decisioni progettuali che ho preso per rispondere ai requisiti funzionali e di *performance*.

3.3.1 Architettura

Tra le possibili architetture che ho studiato e preso in considerazione, ho concluso che quella più adatta ai requisiti del progetto è un'architettura ad eventi. Questo è particolarmente evidente in quanto il committente desidera un'infrastruttura capace di attivarsi e rispondere a nuovi dati che arrivano in maniera asincrona e la cui elaborazione è anch'essa asincrona. Un evento, in questo contesto, è ogni cosa che genera un messaggio in seguito ad essere stato creato, propagato, percepito o consumato. Nella versione di architettura ad eventi che ho scelto, ovvero con una topologia *"Broker"*, vi sono tre componenti principali:

* ***Event Producer***: È la componente che genera l'evento definito come il cambiamento di stato di un elemento di un sistema;
* ***Event Broker***: È la componente che si occupa di intercettare, percepire gli eventi emessi dai *Producer* e smistarli verso i *Consumer*;
* ***Event Consumer***: È la componente che, alla ricezione di una notifica che segnala un nuovo evento, lo consuma elaborando il nuovo dato.

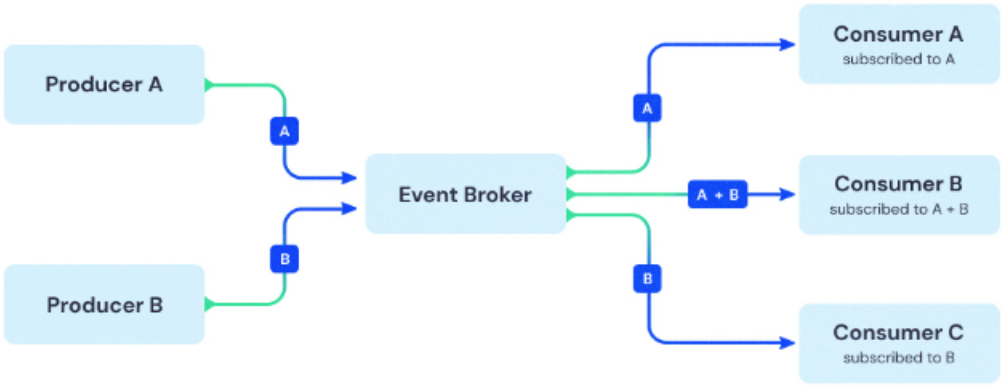


Figura 3.3: Un esempio di architettura ad eventi con topologia *"Broker"*.

Fonte: www.adservio.fr/post/event-driven-architecture

Come è possibile vedere in figura 3.3, i *Consumer* sono caratterizzati da una sottoscrizione ad un particolare *topic*. Dichiarandosi interessati ad un *topic*, i *Consumer* verranno notificati dal *Broker* ad ogni nuovo evento afferente a quel canale di comunicazione condiviso. Nel caso specifico del progetto di *stage*, la corrispondenza tra gli elementi dell'architettura e le tecnologie è la seguente:

Elemento Architetturale	Servizio AWS	Descrizione
Event Producer	**AWS DMS e AWS *Kinesis Firehose***	I servizi DMS e *Kinesis Firehose* sono quelli impiegati per l'approvvigionamento dei dati, rispettivamente per i dati dal *database* e dalle *app*. Questi fungono da *Producer* nell'architettura perché generano un evento all'arrivo di un nuovo *data point* dal rispettivo canale di comunicazione. Ciascuno di loro produce eventi afferenti a diversi *topic*.
Event Broker	**AWS S3**	S3 funge da *Broker* in quanto riceve e smista gli eventi generati dai due servizi di approvvigionamento. S3 cataloga i nuovi dati con un'etichetta a seconda della loro natura e scopo, questo identifica il *topic*. S3 Procede avvertendo i *Consumer* interessati propagando l'evento associato al nuovo dato, così segnalando che questo è disponibile.
Event Consumer	**AWS Lambda**	Ciascuna Funzione Lambda è preposta alla gestione di un particolare tipo di dati. La loro configurazione specifica a quali *topic* sono interessati e di conseguenza all'arrivo di un nuovo evento si attivano. Le Funzioni Lambda recuperano il nuovo dato e lo elaborano, consumando l'evento.

Tabella 3.1: Corrispondenza tra gli elementi architetturali e le tecnologie.

Una trattazione più approfondita sul metodo di archiviazione in S3 e sulle scelte di catalogazione degli oggetti è presente in §3.3.3. Detto questo, voglio precisare che nell'implementazione di questo modello architetturale ho scelto come metodo di distinzione dei *topic* il prefisso assegnato ad ogni oggetto. Il criterio di assegnazione dei prefissi permette di organizzare i dati nel *Data Lake* ma anche di creare dei canali per gli eventi. All'arrivo di un nuovo oggetto, S3 notifica le Funzioni Lambda che sono in ascolto di eventi legati a oggetti etichettati con un determinato prefisso. Questo meccanismo permette di avere il consumo asincrono degli eventi caratteristico di questa architettura. Come precedentemente accennato, ho scelto un'architettura ad eventi perché è tra le più adatte a gestire il flusso della soluzione che il committente ha in mente. Queste sono le sue caratteristiche che mi hanno guidato nella scelta:

* L'architettura ad eventi è pensata per adattarsi a problemi con flussi di dati complessi;
* In un'architettura ad eventi è relativamente semplice aggiungere nuove *pipeline* per i dati. Questo vuol dire aggiungere nuove tipologie di *Producer*, *Consumer* o *topic*;

* Le architetture ad eventi hanno un basso livello di accoppiamento che facilita la separazione logica tra la creazione di un evento e la sua consumazione. Un basso livello di accoppiamento rende più semplice scalare la soluzione permettendo di aggiungere o togliere *Producer* o *Consumer*;
* È *fault tolerant* perché basata sull'asincronismo: qualora un *Consumer* smetta di funzionare l'applicazione continua a lavorare. Infatti gli eventi vengono accodati nel *Broker* dove potranno essere recuperati;
* Impiegando un *Broker* non ci sarà un continuo *polling* da parte dei *Consumer*. Questo significa che la risorsa AWS verrà pagata solamente per il tempo in cui svolge effettivamente del lavoro.

3.3.2 Approvvigionamento

Ho iniziato il mio lavoro di progettazione dall'approvvigionamento dei dati. Questa funzionalità prevede che l'infrastruttura disponga di canali dedicati per il fluire dei dati dalle loro fonti verso il *Data Lake*. Per il progetto di *stage* le fonti primarie per i dati da tenere in considerazione sono due:

* *Database* relazionale PostgreSQL;
* Eventi generati dagli utenti tramite le *app*.

Per brevità, in questo capitolo, discuterò solamente di come ho progettato l'approvvigionamento dal *database*. All'interno del *database* i dati interessanti, e che vanno quindi reperiti, sono quelli corrispondenti a nuovi inserimenti o aggiornamenti in tabelle contenute in una lista che può subire modifiche. Questi dati, che si trovano nel *database* già in forma anonima, corrispondono alle risposte dai pazienti dei *trial* clinici ai questionari giornalieri digitali che devono compilare. Per questi *ePRO* l'azienda ha stabilito che le informazioni in essi contenute sono drasticamente meno significative e rilevanti se fornite con due o più giorni di ritardo rispetto alla data a cui si riferiscono. Per stabilirlo l'azienda ha studiato la letteratura accademica a riguardo. Vedremo poi in §3.3.4 come questa politica venga concretamente applicata per calcolare delle metriche che indicano il livello di coinvolgimento del paziente nel *trial*. Il fatto che questi dati abbiano una "data di scadenza" significa che la loro replicazione deve avvenire in tempi brevi per cogliere ogni variazione. Questo mi ha guidato nella scelta del servizio DMS (§1.3.1) che ho configurato per la replicazione dal *database*. Le motivazioni le principali per il suo utilizzo sono:

* Il servizio supporta la tipologia di *cluster* utilizzata per realizzare il *database* già operativo. Inoltre è compatibile con il servizio di stoccaggio S3 che ho scelto per il progetto;
* È possibile configurare quali *schema* e tabelle si vogliono replicare. Per queste ultime è possibile anche prevedere dei filtri elementari che utilizzano gli operatori aritmetici. Questi sono applicabili alle singole colonne per selezionare le righe da replicare;
* È possibile personalizzare le modalità e le tempistiche che regolano l'inizio di una trasmissione adattandole al carico di dati corrente.

L'ultimo punto è particolarmente importante per rispettare il requisito di *performance* che prevede una replicazione del dato *near-real-time*. Così facendo non si deve rinunciare

alla flessibilità di AWS che consente di pagare una risorsa performante solo quando veramente necessario. Infatti, DMS è configurabile affinché inizi una trasmissione non appena almeno una delle due condizioni seguenti risulta vera:

* La dimensione del *file* risultante dalla codifica dei dati da trasmettere nel formato selezionato è almeno X byte;
* Sono passati almeno Y secondi dall'ultima trasmissione.

Dove X e Y sono due parametri che possono essere tarati sulla base del carico da gestire e possono venire aggiornati con l'aumentare degli utenti e quindi dei dati.

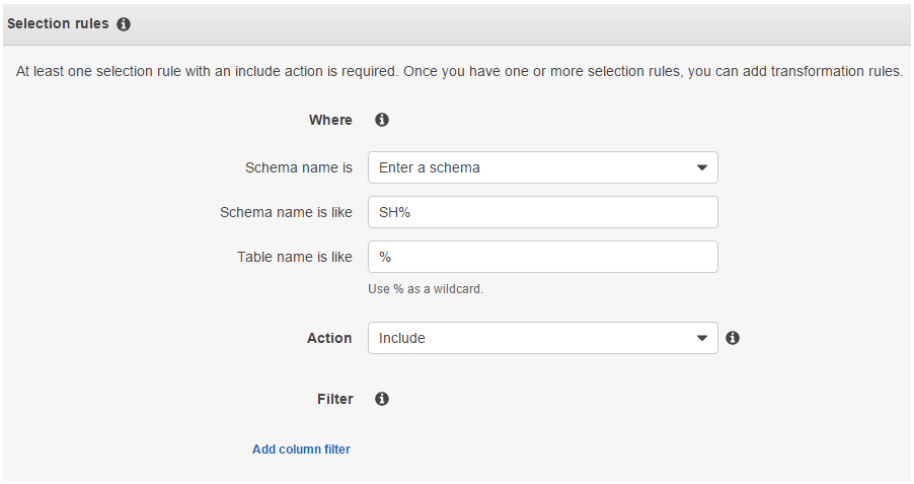


Figura 3.4: AWS DMS permette di specificare regole per la replicazione di tabelle, offrendo anche la possibilità di aggiungere dei filtri per i valori di determinate colonne

Fonte: docs.aws.amazon.com/dms/latest/sbs/

3.3.3 Stoccaggio

Contestualmente alla progettazione e la scelta delle tecnologie per l'approvvigionamento dati mi sono dedicato alla progettazione di un sistema di stoccaggio per i dati acquisiti dalle fonti del progetto. La tecnologia stato dell'arte tra quelle offerte da AWS in questo ambito applicativo, specialmente per quantitativi possibilmente enormi di dati, è S3. Quest'ultimo è un servizio di stoccaggio di dati sotto forma di oggetti, facilmente scalabile, in grado di garantire l'accesso ai dati ad altissima disponibilità e in sicurezza. S3 è tra le tecnologie più utilizzate in assoluto per l'archiviazione dei dati di *Data Lake* e *Data Warehouse*. Essendo un prodotto interno ad AWS, S3 è ricco di integrazioni con altri servizi per la *data analysis*, l'intelligenza artificiale, l'apprendimento automatico e la trasformazione di dati grezzi, privi di struttura, per carpire le correlazioni sottostanti.

L'archiviazione su S3 è strutturata in *bucket* che corrispondono all'unità fondamentale di archiviazione. All'interno dei *bucket* gli oggetti possono essere archiviati associando loro un prefisso che ricorda il percorso di archiviazione in un comune *file system* a livelli. L'oggetto, però, non viene effettivamente collocato in un albero di sotto-cartelle concrete. Le cartelle presenti dentro ad un *bucket* sono infatti astratte e servono per

organizzare gli oggetti in uno spazio piano che altrimenti non farebbe distinzione tra di loro. Sfruttando questo meccanismo per separare concettualmente gli oggetti archiviati ho predisposto i servizi affinché i dati fossero così stoccati:

* Oggetti contenenti dati grezzi, appena replicati o inviati dalle *app* sono identificati dal prefisso "*Unprocessed*";
* Oggetti risultanti dalle operazioni di trasformazione automatiche e quindi aggregati, ripuliti o altresì elaborati sono identificati dal prefisso "*Processed*";
* Ogetti costituenti *dataset* di metriche curate dai *data scientist* sono identificati dal prefisso "*Metrics*".

Ho previsto, inoltre, che il prefisso assegnato ad ogni oggetto includa tre ulteriori sotto-livelli corrispondenti. Questi corrispondono all'anno, il mese e il giorno del mese della data in cui il dato è stato acquisito o generato. I tre strati aggiuntivi servono per sfruttare il partizionamento.

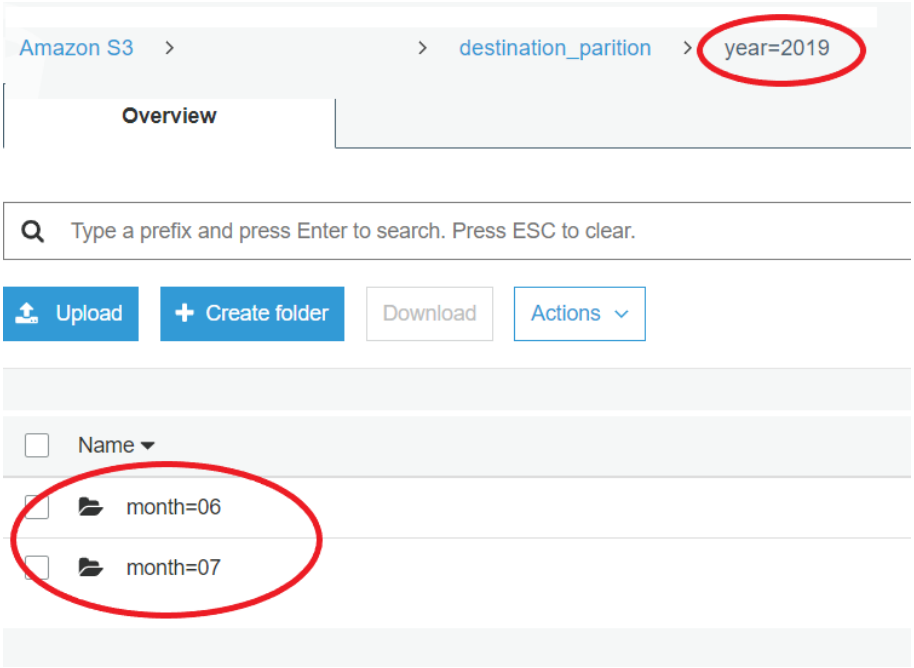


Figura 3.5: L'utilizzo di prefissi per le partizioni avviene primariamente con caratteri alfanumerici e trova larga applicazione con le date.

Fonte: shorturl.at/uwE24

Il partizionamento è una tecnica che ci permette di alleggerire le operazioni di consultazione degli oggetti consentendoci di reperire solo quelli dotati di un prefisso che soddisfa determinati requisiti. Ad esempio, sfruttando il partizionamento, è possibile accedere a tutti gli oggetti "*Unprocessed*" replicati nel primo giorno del mese, qualunque questo sia. Questo comporta dei concreti vantaggi economici in quanto non sarà necessario consultare i metadati di ogni oggetto per accertarsi della data in cui sono stati replicati o generati. Sarà invece sufficiente prelevare tutti quelli che nel

loro prefisso, alla posizione corrispondente al giorno del mese, presentano il numero desiderato. Il vantaggio economico deriva dal fatto che i servizi di archiviazione di massa, come S3, addebitano all'utente una quota fissa per ogni oggetto consultato ed è quindi fondamentale ridurre questo numero il più possibile.
Un altro aspetto importante della progettazione che coinvolge S3 è la definizione di politiche di accesso agli oggetti contenuti dentro il *bucket*. Una buona prassi che ho rispettato è configurare il *bucket* come privato e quindi con accesso consentito solamente a determinati ruoli definiti all'interno del proprio *account* AWS. I dati che vengono depositati nel *bucket* S3 vengono anche crittati contestualmente al loro arrivo. I ruoli associati alle funzioni Lambda, piuttosto che quelli associati ad altri servizi che intendono farne uso, devono essere forniti di *policy* che permettano la decrittazione dei dati. La definizione di ruoli e politiche è stato un mio compito che ho ampiamente normato nella documentazione prodotta in quanto coinvolta nella gestione della *privacy* delle persone a cui i dati si riferiscono, seppur come precedentemente menzionato, questi siano anonimi.

3.3.4 Trasformazione

Con trasformazione dei dati, in questo progetto, intendiamo il processo che prende i dati grezzi, privi di struttura, e tramite raggruppamenti, unioni, filtri, generazione ed eliminazione di colonne produce dei *dataset*. Tali *dataset* sono ripuliti da eventuali rumori, e contengono valori qualitativi sui dati di partenza sotto forma di metriche.
Per la progettazione dei meccanismi automatici di trasformazione dei dati l'azienda ha posto un vincolo tecnologico che richiedeva l'utilizzo di Python come linguaggio di programmazione. Tenendo questo a mente ho incominciato valutando le possibili opzioni tecnologiche e ho concluso che il servizio di calcolo Lambda fosse adatto, allora ho approfondito il suo studio per meglio comprendere le funzionalità disponibili e come sfruttarle. Il progetto di *stage* aveva come obiettivo per la fase di trasformazione dei dati la creazione di un *Proof of Concept* per dimostrare che la scelta tecnologica fosse in grado di produrre il risultato atteso. In tal senso ho concordato assieme al responsabile del *team* quali fossero i requisiti che il *Proof of Concept* dovesse soddisfare. Questi possono essere riassunti nei seguenti punti principali:

* Le Funzioni Lambda si attivano automaticamente all'arrivo di nuovi oggetti;
* Per ogni nuovo oggetto con dati della compilazione di ePRO deve venire aggiornata la metrica di ingaggio dei pazienti;
* I valori aggiornati della metrica di ingaggio devono essere riportati nella tabella corrispondente nel *database*;
* Il corretto funzionamento delle Funzioni Lambda deve essere monitorato e degli allarmi devono scattare superata una soglia prestabilita di fallimenti ravvicinati nel tempo.

Dal punto di vista progettuale, la sfida più difficile e significativa che ho affrontato è stata quella del calcolo dell'ingaggio dei pazienti, di conseguenza in questo capitolo mi limiterò a presentare questa. Per capire meglio la richiesta approfondiamo quanto accennato in §3.3.2 facendo maggior chiarezza sulla metrica dell'ingaggio.
L'applicazione *mobile* di Patchai propone un'esperienza utente conversazionale nella forma di un *chatbot* che aiuta i pazienti nello svolgimento delle attività giornaliere previste dal *trial* clinico. Tra le più importanti di queste attività vi è la compilazione

di un *eDiary*. Un *eDiary* è una voce giornaliera nel *record* di un paziente all'interno del *trial* e ogni *eDiary* contiene uno o più ePRO. Il valore delle informazioni inserite è cruciale per i medici, i quali usano queste informazioni per studiare gli effetti e le conseguenze dei farmaci, dei trattamenti o dei protocolli e pertanto devono essere le più accurate possibile. L'azienda ha stabilito che un paziente, mediamente, è in grado di riportare con precisione i dati richiesti con al massimo due giorni di ritardo rispetto alla data a cui questi fanno riferimento. Un ePRO correttamente compilato entro questa "data di scadenza" è considerato ingaggiato. Guardando quindi al *record* di compilazione degli *eDiary* di ogni paziente è possibile contare quanti ePRO per ciascuno *eDiary* siano stati compilati in maniera "ingaggiata". Ad ogni *eDiary* viene poi assegnato un valore nella seguente maniera:

* **Valore 0**: Se l'utente non ha compilato nessun ePRO entro la scadenza;
* **Valore 1**: Se l'utente ha compilato almeno un ePRO entro la scadenza;
* **Valore 2**: Se l'utente ha compilato tutti gli ePRO entro la scadenza;

Per raffinare ulteriormente l'interazione dell'utente con l'applicazione si vuole anche calcolare il *trend* settimanale, facente riferimento agli *eDiary* degli ultimi sette giorni. Il motivo per aggiungere questa misura è da ricercarsi nel fatto che il punteggio assegnato al singolo *eDiary* non tiene conto della costanza con cui questi vengono compilati. Il *trend* è quindi inteso come il coefficiente angolare della retta che interpola i sette valori calcolati per i corrispondenti *eDiary*. Questo permette di creare un messaggio su misura per ogni paziente che tenga in considerazione il livello di ingaggio nel *trial*. Ad esempio premiando il paziente costante e diligente e incoraggiare, rassicurandolo, quello più incostante.

3.3.5 Accesso ai Dati

La parte della progettazione dedicata all'accesso ai dati mi ha visto studiare quali tecnologie e meccanismi di autenticazione potessero dotare la soluzione del livello di sicurezza e facilità d'uso che il personale dell'azienda necessita. Lo scopo era di fornire un sistema autenticato per accedere ai dati, analizzarli e manipolarli. In questa scelta ho dovuto tenere a mente due requisiti principali per questa funzionalità:

* **Autenticazione**: Per reperire i dati del *Data Lake* e analizzarli in locale è necessario essere autenticati come personale autorizzato dell'azienda;
* **Linguaggio**: Il linguaggio scelto per la manipolazione dei dati è Python. Di conseguenza deve essere possibile reperire i *dataset* tramite del codice in questo linguaggio.

Per soddisfare questi requisiti ho ricercato se ci fossero delle librerie per accedere alle proprie risorse AWS tramite del codice Python previa autenticazione e ho optato per la libreria Boto3. Boto3 è il *Software Development Kit (SDK)* Python per AWS e permette di semplificare l'integrazione di *script* Python con i servizi di AWS. Per poter offrire l'autenticazione degli utenti è però necessario affiancare a Boto3 la *Command Line Interface (CLI)* di AWS. Questa *CLI* permette di creare un file locale sicuro dove aggiungere le proprie chiavi di accesso all'account AWS. Boto3 riesce a recuperare le credenziali tramite il nome del ruolo AWS, associato al dipendente, che viene specificato nell'instanziazione di un *client* di comunicazione nello *script*.
I *data scientist* dell'azienda lavorano prevalentemente tramite l'uso locale di *notebook*

```
import boto3

session = boto3.Session(profile_name='dev')
dev_s3_client = session.client('s3')
```

Figura 3.6: Un esempio di sessione istanziata con le credenziali associate al dipendente 'dev'.

Fonte: boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html

Jupyter. Questi sono un ambiente di sviluppo integrato per il codice e particolarmente adatto all'analisi esplorativa del dato. Sono estremamente flessibili nel senso che permettono la definizione di più blocchi di codice all'interno del singolo *notebook* ed è possibile la loro esecuzione indipendente. Nella progettazione mi sono preoccupato anche di integrare un servizio da utilizzare quando le *query* da eseguire o i *dataset* da analizzare sono troppo pesanti per le macchine in dotazione ai dipendenti. Per questo scopo ho integrato nell'architettura di servizi AWS anche AWS Athena. Athena è un potente strumento per l'esecuzione di *query* pensato per elaborare dati da enormi *dataset*. Su Athena è possibile definire la struttura di tabelle tipiche dei *database* relazionali da usare come maschere di lettura per i dati contenuti in oggetti come quelli stoccati in S3. In questo modo è possibile effettuare *query* anche su *dataset* privi di struttura disseminati in centinaia o migliaia di oggetti in un *bucket*. Athena, infine, permette di sfruttare il partizionamento dei dati per permettere di ridurre i costi in queste operazioni costose per natura, data la mole di oggetti da consultare.

3.3.6 Visualizzazione

L'obiettivo finale del cliente legato al requisito di visualizzazione è il motivo per cui le aziende si prodigano per la definizione della loro *Data Strategy*. Si vuole riuscire a supportare le decisioni di *business* con delle informazioni fattuali estratte dai dati generati dai propri utenti. Nella scelta degli strumenti di visualizzazione per i dati ho tenuto sempre a mente il pubblico che avrebbe dovuto utilizzarli. Questo pubblico potrebbe non avere le competenze per comprendere o beneficiare da una visualizzazione particolarmente tecnica o dettagliata delle informazioni. Detto questo, in accordo con il rappresentante del *team*, ho pensato di proporre due strumenti diversi di visualizzazione mirati a due categorie leggermente diverse di fruitori:

* **AWS Quicksight**: Strumento per creare visualizzazioni puramente grafiche pensato per un pubblico meno tecnico;
* **Streamlit**: Strumento che unisce i grafici ai dati sottostanti presentati in forma tabellare. È pensato per il *team* coinvolto direttamente nel prodotto e più tecnico.

Entrambi gli strumenti richiedono una prima configurazione da parte di un *data scientist* per ogni visualizzazione che si intende creare. Fatto questo, ad ogni consultazione, il bacino di dati utilizzati per creare la visualizzazione risulterà aggiornato. Una collezione di visualizzazioni così strutturata costituisce un vero e proprio cruscotto costantemente aggiornato sul valore delle metriche che per l'azienda e il *team* sono più significative per supportare e guidare le decisioni produttive.

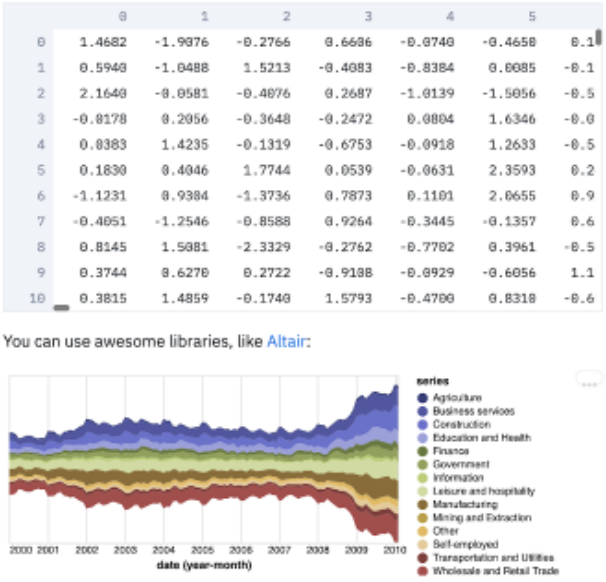


Figura 3.7: Un esempio di visualizzazione con Streamlit che unisce il grafico ai dati sottostanti.

Fonte: venturebeat.com/ai/streamlit-raises-19-million-for-a-framework-that-simplifies-ai-app-development

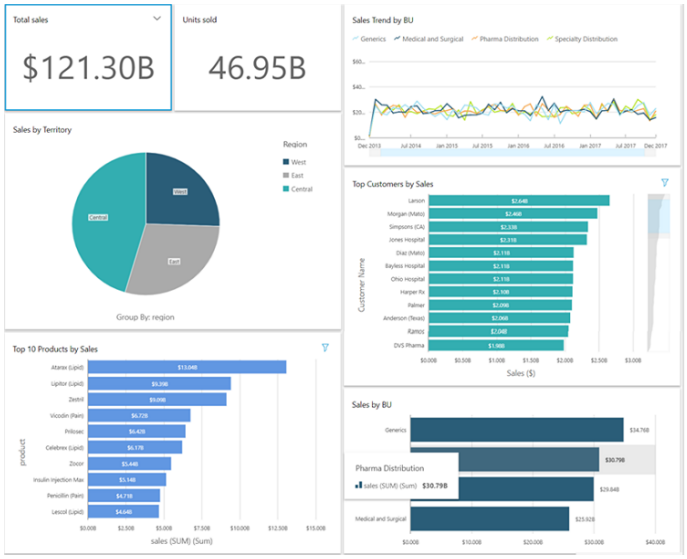


Figura 3.8: La visualizzazione con Quicksight è priva di informazioni tecniche superflue per creare una visualizzazione più pulita.

Fonte: aws.amazon.com/blogs/big-data/embed-interactive-dashboards-in-your-application-with-amazon-quicksight

3.4 Codifica

L'attività di codifica del progetto mi ha visto impegnato nella realizzazione di due componenti principali:

* **Terraform**: La codifica dei *file* Terraform comprendente la definizione delle risorse AWS e la loro configurazione. Questo permette che possano venire duplicate, eliminate o modificate in un contesto di *IaC*;
* **Python**: Gli algoritmi di trasformazione dei dati inseriti nelle Funzioni Lambda utilizzate per il *Proof of Concept* presentato in §3.3.4.

Ho iniziato a dedicarmi all'attività di codifica dopo un primo periodo di *focus* esclusivo sulla progettazione. Quando ho iniziato a implementare concretamente le soluzioni progettate, le due attività si alternavano anche per poter rispondere alle variazioni dei requisiti. A livello di tempo ho dedicato certamente più ore alla progettazione dato che le tecnologie e i concetti mi erano completamente sconosciuti e progettare una soluzione di questa complessità mi ha richiesto un grande sforzo. Sono diventato anche sempre più efficace ed efficiente nella scrittura di codice con entrambe le tecnologie nel corso del progetto. Questo è diventato più evidente man mano che facevo mie le *best practice* e meglio comprendevo quali fossero le reali aspettative dell'azienda per il codice prodotto.

Degli esempi di *best practices* che ho appreso riguardano l'utilizzo, ove possibile senza forzature, delle espressioni vettorializzate in Python. Mi riferisco in particolar modo alla libreria Numpy, per rendere più efficienti, anche di ordini di grandezza, le operazioni con *array* e matrici. Un altro esempio è la tecnica che permette di importare la configurazione di una risorsa già dichiarata su AWS sotto forma di codice nello *script* Terraform. Questa pratica permette di prendere una risorsa già definita su AWS, estrarne la configurazione completa e porla sotto il *management* di Terraform senza dover dichiarare ogni parametro manualmente nel *file*. Così facendo è possibile testare facilmente diverse configurazioni, consentendo all'umano di definire una risorsa tramite un'interfaccia grafica guidata per poi estrarre la configurazione risultate sotto forma di codice.

```
resource "aws_dms_replication_instance" "postgres-replication-instance" {
  replication_instance_class = "dms.t3.micro"
  replication_instance_id = "postgres-replication-instance"
  apply_immediately = true
  allocated_storage = 50
  publicly_accessible = true
  auto_minor_version_upgrade = true
  availability_zone = "eu-central-1a"
  multi_az = false
  replication_subnet_group_id = aws_dms_replication_subnet_group.replication-subnet-group.
  replication_subnet_group_id
  vpc_security_group_ids = [ aws_security_group.replication-security-group.id ]
}
```

Figura 3.9: Un esempio di definizione di un ruolo AWS e di una risorsa, in questo caso un'istanza di replicazione DMS.

```python
import awswrangler as wr
import boto3
import pandas as pd

boto3.setup_default_session(profile_name="marcatti")
s3_client = boto3.client("s3",region_name='eu-central-1')
s3 = boto3.resource("s3")

#Carico il nuovo file contenente epro in un dataframe
key = "unprocessed/submitted_epros/year=2022/month=06/day=15/20220615-123547181.parquet"
epros= wr.s3.read_parquet("s3://patchai-bucket/"+key)
#Rimuovo quelli eliminati e converto in formato data
epros = epros[epros.deleted_at.isnull()] #not deleted
epros['due_date'] = pd.to_datetime(epros["due_date"]).dt.date
#Filtro per epro che fanno riferimento a ediary
epro_diaries = epros[epros['parent_type']=='ediary']
#Li raggruppo per ediary sommando quelli che sono completati e quelli che sono parzialmente completati
epro_diaries_groupped=pd.pivot_table(epro_diaries,index='parent_id', columns='status', values='id',aggfunc='count').fillna(0).reset_index().rename(columns={'parent_id':'ediary_id'})
#Stampa
epro_diaries_groupped
```

Figura 3.10: Un frammento di codice Python per il calcolo dell'ingaggio del paziente.

Alcuni dati quantitativi sul mio lavoro:

* **Python**: 5372 righe di codice Python;
* **Terraform**: 563 righe di codice Terraform;

Ho prodotto codice Python per 4 funzioni Lambda effettivamente create e attivate su AWS. Le restanti righe di codice compongono 5 *notebook* Jupyter che ho predisposto, assieme alla mia *tutor*, per una dimostrazione conclusiva sul mio lavoro in azienda. Questi *notebook* rappresentano esempi d'uso reale dell'infrastruttura da parte dei *data scientist* per l'accesso e manipolazione dei dati.
Il codice Terraform è invece suddiviso in 3 *file* contenenti rispettivamente:

* La dichiarazione delle risorse dell'architettura;
* Variabili configurabili dell'architettura come la regione per il *deployment* e la lista di tabelle da replicare tramite servizio DMS;
* La configurazione dell'installazione di Terraform e i parametri di esecuzione per gli *script*.

3.5 Verifica e Validazione

Il processo di verifica del codice è stato applicato esclusivamente al lavoro prodotto in linguaggio Python, per farlo ho utilizzato la funzionalità di *testing* messa a disposizione dal servizio AWS Lambda. Per ogni funzione è possibile creare dei *test* per verificare il corretto funzionamento del codice, sia in condizioni ideali di normale funzionamento che per i casi limite. La funzionalità di test permette inoltre di simulare eventuali *trigger* che danno il via all'esecuzione della funzione come l'arrivo di un nuovo *file* in un *bucket* con delle particolari caratteristiche. Ho quindi prodotto dei *test* per ogni funzione che ho creato durante il progetto e la loro corretta esecuzione veniva valutata durante gli incontri di revisione con la mia *tutor* e con il responsabile del *team*. I *test* per le funzioni Lambda possono essere anche derivati da un *template*, per crearlo si definisce un caso da testare generico con le informazioni minime necessarie. Per ogni ulteriore *test* della stessa categoria si completa la struttura base con le specifiche del caso.
La tipologia prevalente di *test* che ho implementato è quella che simula l'inserimento di un *file* precedentemente identificato con lo stesso prefisso dei *file* che la funzione è preposta a gestire. Durante l'esecuzione, la funzione generava un *log* delle operazioni dal quale era possibile stabilire se l'esito fosse positivo o meno.

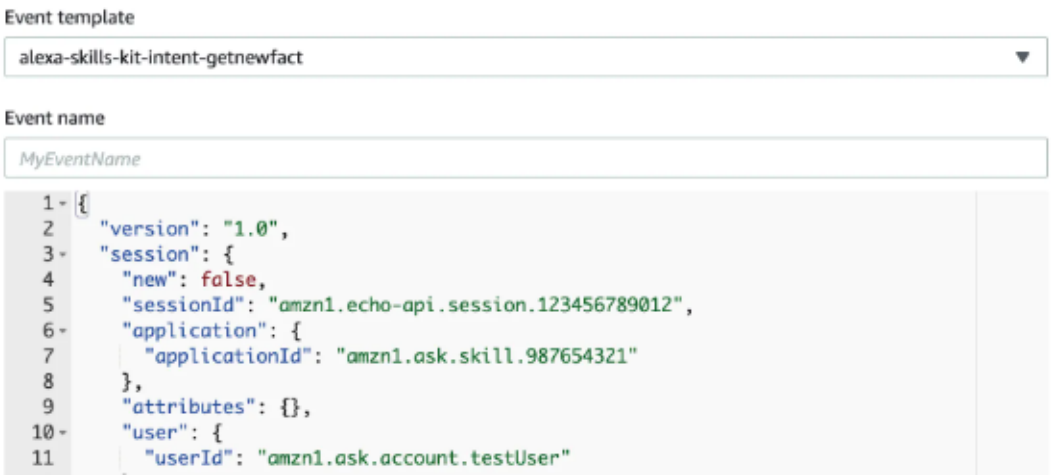


Figura 3.11: La configurazione dei test per le funzioni Lambda è dichiarata in un *file* JSON

Fonte: openupthecloud.com/how-to-test-aws-lambda

Per la verifica del codice ho usato i seguenti elementi di analisi statica:

* Controllo delle procedure e variabili realmente utilizzate nel ciclo di esecuzione;
* Controllo delle librerie importante concretamente utilizzate;
* *Inspection* e *Walkthrough* manuale;

Per l'analisi dinamica in complesso sono stati creati ed eseguiti 12 *test* sulle funzioni Lambda con la funzionalità del servizio sopracitata.
Il risultato finale della fase di verifica del codice è una copertura dello stesso pari al 100%.

3.6 Risultato Finale

Il seguente diagramma raffigura l'architettura che ho sviluppato nel progetto ad alto livello ma sono ben distinguibili i vari elementi fondamentali della progettazione:

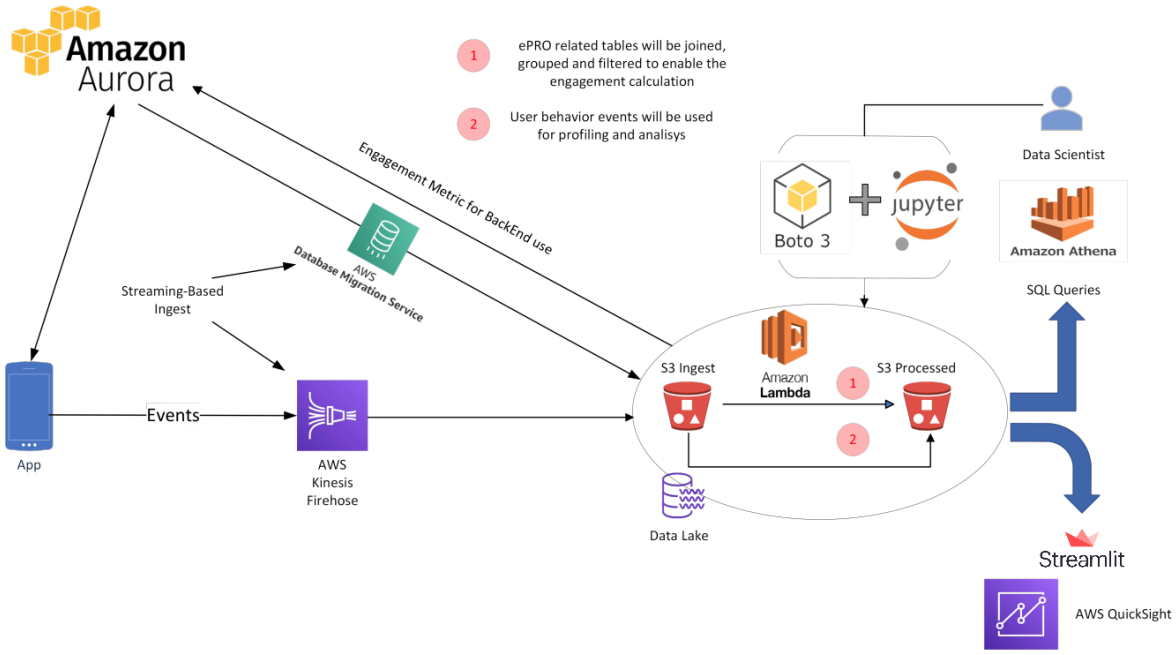


Figura 3.12: Schema dell'architettura realizzata

Il risultato del lavoro di documentazione è la creazione di 6 documenti. Ognuno di questi corrisponde ad una "pagina" *Confluence*, il prodotto Jira pensato per la documentazione.

- 5 documenti, uno per ogni macro-requisito del progetto;
- 1 documento che stima i costi del progetto a varie grandezze di *deployment*.

Il lavoro che ho svolto ha soddisfatto la totalità dei requisiti posti dall'azienda, sia per quanto riguarda i requisiti principali presentati in §3.2 che i sotto-requisiti ad essi associati. Gli obiettivi aziendali presentati in §2.4 sono tutti stati raggiunti e l'azienda si è detta estremamente soddisfatta del lavoro svolto.

Capitolo 4

Conclusioni e Retrospettiva

4.1 Obiettivi Raggiunti

4.1.1 Obiettivi Aziendali

In questa sezione riporto il soddisfacimento degli obiettivi aziendali per il progetto così come sono stati presentati in §2.4.

Obiettivo Aziendale	**Soddisfacimento**
Produzione di una descrizione tecnica	Soddisfatto
Studio sulle tecnologie da adottare	Soddisfatto
Realizzazione del prodotto: Approvvigionamento	Soddisfatto
Realizzazione del prodotto: Stoccaggio	Soddisfatto
Realizzazione del prodotto: Trasformazione	Soddisfatto
Realizzazione del prodotto: Accesso ai dati	Soddisfatto
Realizzazione del prodotto: Visualizzazione	Soddisfatto

Tabella 4.1: Soddisfacimento obiettivi aziendali del progetto.

Come è possibile vedere ho soddisfatto il 100% degli obiettivi prefissati dall'azienda per il progetto di *stage*.
Per poter affermare quanto riportato in tabella 4.1 ho sostenuto due incontri per l'accettazione del prodotto. Ho fatto questi due incontri in conclusione del progetto di *stage*, rispettivamente con la mia *tutor* e il responsabile del *team*. Per valutare la soddisfazione degli obiettivi abbiamo fatto quanto segue:

* **Produzione di una descrizione tecnica**: Ho presentato il materiale documentale prodotto descrivendo come l'ho strutturato, le convenzioni che ho seguito e le fonti alle quali mi sono rifatto. Abbiamo inoltre revisionato i puntatori a possibili sviluppi futuri che ho segnalato in continuità con quanto già fatto.

- **Studio sulle tecnologie da adottare**: Il risultato dello studio tecnologico è parte del contenuto della documentazione. Ho ricevuto l'approvazione finale dello studio tecnologico nel momento in cui abbiamo revisionato il materiale documentale prodotto. Detto questo, avevo già ricevuto un'approvazione parziale che mi ha permesso di procedere con il lavoro di codifica.

- **Realizzazione del prodotto**: In conclusione del progetto ho mostrato le *demo* preparate, testato il codice Python prodotto e visionato la configurazione Terraform della soluzione. Il corretto funzionamento dell'infrastruttura durante le *demo* e l'esito positivo dei test sono le condizioni di accettazione che abbiamo utilizzato.

Al termine di questi due incontri ho ricevuto, da entrambi i colleghi, l'accettazione del prodotto finale e contemporaneamente la conferma del soddisfacimento degli obiettivi aziendali.

4.1.2 Obiettivi Personali

Ritengo di aver raggiunto e soddisfatto tutti gli obiettivi personali che mi ero prefissato per lo *stage* in §2.6. In particolare ho approfondito molto la mia conoscenza del linguaggio Python, con il quale ora sono molto più a mio agio. Ho fatto grandi passi avanti nel suo utilizzo efficiente, ad esempio sfruttando le operazioni vettorializzate. Sono poi molto soddisfatto delle conoscenze pratiche acquisite con alcune tra le librerie più popolari per l'analisi dei dati. Queste abilità derivano da una comprensione dell'intento dietro alla loro ideazione, la scelta delle strutture dati offerte e i metodi a disposizione. Sono molto soddisfatto dell'esperienza progettuale acquisita con le tecnologie *Cloud* e quelle per l'*IaC*. Il giusto livello di sfida di questo progetto non mi ha lasciato scelta se non quella di comprendere il modo di utilizzare le tecnologie, sapendo scegliere quelle più adatte. Dovendo progettare la mia architettura ho appreso molto, ho elevato le mie conoscenze pregresse per riadattarle ed espanderle in un contesto nel quale non mi ero mai cimentato. Sono infine soddisfatto del buon livello di efficienza raggiunto dalla configurazione che ho prodotto, a riprova che ho compreso l'intento delle tecnologie e di come sfruttarle a dovere.
In conclusione sono estremamente soddisfatto dell'esperienza lavorativa acquisita. Dovendo relazionarmi ogni giorno, continuativamente, con un *team* di professionisti ho potuto apprendere molto. Ho avuto molte occasioni per lavorare sulle mie *soft skill* per l'organizzazione personale del lavoro e le relazioni con gli altri dipendenti. Ho imparato molto e fatto pratica concretamente con una metodologia di lavoro *agile* tra le più utilizzate e richieste. Ho avuto occasione di vedere e apprendere come un *team* organizza il proprio tempo, le risorse umane ed economiche. Tutte queste nuove conoscenze mi permetteranno di essere più preparato per il mio futuro ingresso nel mondo del lavoro e di avere chiaro in mente cosa cerco in un'azienda.

Ritengo di essermi posto degli obiettivi personali che mi è stato possibile raggiungere solo portando a termine il progetto con dei buoni risultati tecnici e nelle scadenze. È per questo motivo che l'aver raggiunto gli obiettivi aziendali, e il senso di appagamento e soddisfazione qui descritto relativamente agli obiettivi personali, mi danno la confidenza di dire che ho raggiunto anche quest'ultimi. Il progresso qui documentato è un'evidenza, seppur narrativa, del soddisfacimento di quest'ultimi. Nella definizione degli obiettivi personali ho trovato grande difficoltà nel identificarli affinché fosse poi possibile misurare analiticamente il loro raggiungimento. Questa sarà una lezione

importante che mi porterò via dall'esperienza di *stage* per farne tesoro nelle prossime occasioni.

4.2 Retrospettiva sulle Conoscenze

4.2.1 Conoscenze

Lo *stage* mi ha permesso di:

Approfondire la conoscenza di Python
Utilizzare le espressioni, le strutture dati e i metodi più adatti ed efficienti per produrre del codice di qualità. Migliorare le mie abilità organizzative del codice per produrne di leggibile, riutilizzabile e manutenibile. Utilizzare le librerie per l'analisi dati, un settore che mi interessa molto, usandole concretamente in un progetto complesso.

Imparare Terraform e l'*IaC*
Comprendere i vantaggi di gestire un'infrastruttura *Cloud* tramite codice che descrive la sua configurazione. Maturare esperienza pratica con Terraform sapendo gestire un complesso di servizi vasto e intrecciato.

Imparare tecnologie *Cloud* come AWS
Comprendere le motivazioni e le scelte dietro alla definizione di un complesso di servizi *Cloud*. Imparare a progettare un'architettura con i servizi di AWS e configurarli per una comunicazione efficace.

4.2.2 Abilità

Con lo *stage* ho potuto lavorare sulle seguenti abilità:

Organizzative
Ho dovuto organizzare il mio tempo per riuscire a produrre un risultato soddisfacente perseguendo: efficacia, *performance* ed economicità. La necessità di raggiungere un obiettivo in un tempo limitato, con risorse limitate, mi ha forzato a migliorare. Le mie abilità organizzative sono molto migliorate anche dall'osservazione e il confronto con i colleghi dai quali ho potuto apprendere.

Comunicative
Durante il progetto ho fatto decine di incontri con la mia *tutor* e il responsabile del *team*. Ho potuto lavorare sulle mie abilità comunicative per esporre il mio punto di vista, dare una visione completa ma concisa di un concetto o una problematica. In conclusione dello *stage* ho presentato il mio lavoro a tutta l'azienda, per farlo ho dovuto lavorare sulla comunicazione, sulla scelta contenutistica e sulla presentazione. In questa occasione ho dovuto uscire dalla mia zona di *comfort* ed elevare le mie abilità.

4.3 Considerazioni Finali

Nell'intraprendere questo percorso di *stage* ho notato un'evidente differenza tra le conoscenze fornitemi dal corso di studi e quelle richieste dal mondo del lavoro. Questa differenza è però, quasi esclusivamente, da ricercarsi nelle tecnologie che non vengono

affrontate negli insegnamenti. Il motivo non è di certo l'arretratezza o la distanza del mondo accademico da quello produttivo del lavoro, bensì è una scelta ponderata. Dato il rapido ricambio tecnologico presente nel settore informatico, che vede un susseguirsi continuo di tecnologie, l'università sceglie di spendere sapientemente le sue risorse per dare agli studenti gli strumenti per apprendere. Nel mondo del lavoro ci capiterà di dover lavorare in progetti che richiedono *suite* tecnologiche a noi sconosciute. Avendo però imparato ad organizzare il proprio lavoro, e premettendo una componente fondamentale di studio, sarà possibile trovare il bandolo della matassa e avanzare nel progetto. Il corso di studi ci ha messo nelle condizioni per fare un giro di prova, in questo senso, con i progetti e gli *assignment* previsti da alcuni degli insegnamenti. Per diversi di questi progetti o *assignment* nessuno ci ha fatto della formazione sulle tecnologie da utilizzare. Organizzando, però, lo studio autonomo delle tecnologie questi sono stati la perfetta opportunità formativa in vista dello *stage* dove ho replicato la stessa esperienza in un contesto di lavoro produttivo vero e proprio. Il divario tra le due entità è quindi presente, secondo me, ma sono fermamente convinto che venga colmato dall'abilità che si sviluppa ad apprendere in autonomia e nell'organizzare il lavoro

In conclusione ho trovato l'esperienza vissuta durante il periodo di *stage* molto utile e formativa per conoscenze e abilità apprese e ampliate.
Ritengo che un periodo di *stage* come quello descritto in questo documento sia la perfetta attività per concludere il percorso di studi triennale. In questa occasione ho potuto riflettere e colmare il divario che si forma tra il mondo del lavoro e quello universitario.
Ho potuto mettere alla prova le mie conoscenze e abilità, ma soprattutto la capacità di apprendimento. Con mia grande soddisfazione sono riuscito ad affrontare questa sfida producendo un buon risultato, portando a termine quest'esperienza come *data engineer*. Questa è stata la mia prima opportunità per lavorare in questa branca dell'informatica e dello sviluppo software. Avevo già maturato un apprezzamento e un interesse particolare per questo settore, ma l'esperienza intrapresa mi ha permesso di confermarlo. Sono estremamente soddisfatto del lavoro svolto, in particolare di come sono riuscito a costruire e gestire un'infrastruttura di servizi discretamente complessa. Ulteriore fonte di appagamento personale è per me il grado di soddisfazione espresso dalla mia *tutor* e dal responsabile del *team*, i quali hanno espresso la volontà di assumermi come dipendente. A tal proposito, sto ancora valutando se proseguire il mio percorso di studi con una laurea magistrale, ma, in ogni caso, è un onore ricevere una proposta come questa da dei professionisti per i quali nutro grande rispetto.

Acronimi e Abbreviazioni

API Application Programming Interface. 3

AWS Amazon Web Services. 3, 7

B2B Business to Business. 1

CDP Customer Data Platform. 10

CLI Command Line Interface. 21

DMS Database Migration Service. iv, 4

ePRO Electronic Patient Reported Outcome. 8, 10

IaC Infrastructure as Code. 5

MVP Minimum Viable Product. 10

PRO Patient Reported Outcome. 8

S3 Simple Storage Service. 4, 5, 7

SaaS Software as a Service. 1

SDK Software Development Kit. 21

Glossario

Big Data Con Big Data ci si riferisce a collezioni di dati di enormi dimensioni caratterizzati da una grande varietà, il cui volume è in continua crescita e che arrivano con elevata velocità. 9, 31

Data Warehouse Un data warehouse è un tipo di sistema di gestione dei dati progettato per abilitare e supportare le attività di Business Intelligence, in particolare le analitiche. 4, 31

Data Lake Con Big Data ci si riferisce a collezioni di dati di enormi dimensioni caratterizzati da una grande varietà, il cui volume è in continua crescita e che arrivano con elevata velocità. 10, 31

User stories Le User Story rappresentano una pratica Agile, utilizzata soprattutto in Scrum, per "catturare" le esigenze degli utenti esprimendo in maniera generale, non dettagliata, caratteristiche, funzioni e requisiti per il prodotto da realizzare.. 2, 31