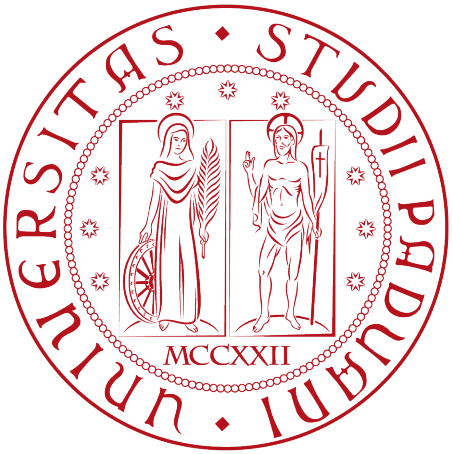
**Università degli Studi di Padova**

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica




# Implementazione di librerie infrastrutturali per la gestione di risorse Cloud aziendali

*Tesi di laurea triennale*

*Relatore*

Prof. Tullio Vardanega

*Laureando*

Francesco Bugno

Anno Accademico 2021-2022

*Dedicato alla mia famiglia e alle persone a me più care*

# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 312 ore, dal laureando Francesco Bugno presso l'azienda THRON S.p.A. L'obiettivo dello stage era implementare un set di librerie per la gestione di risorse infrastrutturali all'interno dei progetti aziendali. Le librerie sviluppate dovevano poi essere integrate in maniera semplice ed autonoma con la filiera di CI/CD aziendale. Queste librerie inoltre dovevano essere parametrizzabili in modo da fornire allo sviluppatore abbastanza libertà per soddisfare le sue esigenze, senza però uscire dalle convenzioni aziendali.

## Organizzazione del testo

Il presente documento si divide in quattro capitoli:

1. **Presentazione dell'azienda:** informazioni generali, descrizione dei prodotti offerti, organizzazione e tecnologie utilizzate, tipo di clientela e propensione all'innovazione;
2. **Collocazione del progetto nella realtà aziendale:** perché lo hanno proposto, aspettative dell'azienda e aspettative personali, obiettivi preventivati e vincoli da rispettare;
3. **Descrizione delle attività di stage:** cosa ho prodotto e come l'ho fatto, quali scelte progettuali e tecnologiche ho effettuato, quali problematiche ho riscontrato;
4. **Valutazione retrospettiva dello stage:** obiettivi raggiunti e competenze acquisite.

## Convenzioni tipografiche

All'interno del documento ho adottato le seguenti convenzioni tipografiche:

* Gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* I termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

# Ringraziamenti

*Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Vardanega, relatore della mia tesi, sia per l'aiuto ed il sostegno fornitomi durante la stesura del lavoro ma anche per la professionalità con cui gestisce e coordina il corso di studi.*

*Desidero ringraziare con affetto i miei genitori per il sostegno, per essermi stati vicini durante gli anni di studio e per non aver mai smesso di credere in me.*

*Ringrazio di cuore nonna Cesarina per essersi presa cura di me durante tutta la mia crescita e per essere stata sempre di conforto nei momemnti più difficili.*
*Questo traguardo lo dedico a te.*

*Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme, per le avventure vissute e per i pianti condivisi. Siete indispensabili come l'aria che respiro.*

*Infine, ci tengo a ringraziare l'azienda THRON, nello specifico il mio tutor Luca Tiozzo per l'infinita disponibilità e tutti i colleghi con cui ho avuto la fortuna di lavorare nell'ultimo periodo per la compagnia, le risate e gli aperitivi passati insieme.*

*Padova, Luglio 2022* Francesco Bugno

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Capitolo 1

# Introduzione

## 1.1 Profilo aziendale

THRON S.p.A. è un'azienda italiana, attiva da oltre vent'anni, che opera nell'ambito dello sviluppo di SaaS (*Software as a Service*) con servizi di *marketing* e *business intelligence* che la collocano nel settore dei DAM (*Digital Asset Management*), ossia sistemi integrati di gestione dei contenuti aziendali (1.2 per maggiori informazioni). I fondatori sono l'attuale CEO Nicola Meneghello ed CTO Dario De Agostini che nel 2000 decidono di avviare la loro attività con sede a Piazzola sul Brenta, di fronte alla rinomata villa Contarini.

Negli anni THRON riesce a ritagliarsi uno spazio proprio in questo mercato portando il *made in Italy* in un campo del tutto inesplorato. Agli albori, per vincere questa sfida, i fondatori hanno puntato sulle persone e sono riusciti a riportare in Italia diversi "cervelli" fuggiti all'estero, creando un *team* competitivo. I principali valori che caratterizzano e rendono unico il *team* di THRON sono infatti i seguenti:

* **Antifragilità**: ovvero la capacità di affrontare le difficoltà, accogliere i tentativi e gli errori come opportunità per migliorarsi;
* **Innovazione**: intesa come adattamento ad un contesto in continuo cambiamento, per offrire ai clienti soluzioni tecnologiche sempre aggiornate e realmente utili;
* **Ispirazione:** verso i clienti per aiutarli a raggiungere i loro obiettivi.

## 1.2 Dominio applicativo

Il prodotto offerto da THRON è la *Thron DAM Platform*, una piattaforma per la gestione dei contenuti digitali.
Questo prodotto è nato per raggruppare in un unico posto tutti gli strumenti per poter gestire contenuti digitali. L'idea alla base per la realizzazione di questo prodotto è considerare i contenuti come il vero valore aggiunto, slegando quindi l'organizzazione dalla piattaforma di distribuzione. Le piattaforme di distribuzione infatti sono in continua evoluzione, ma il valore del contenuto rimarrà sempre invariato nel tempo.

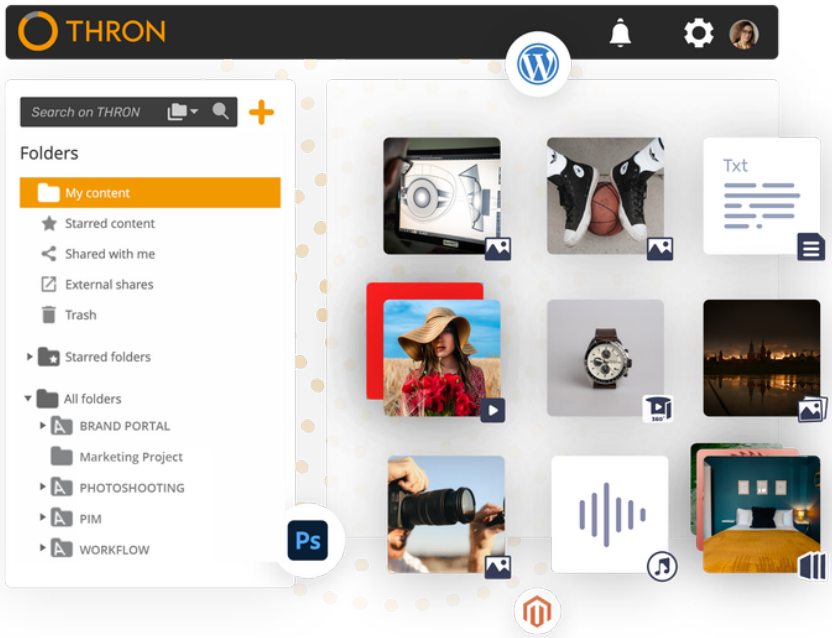



**Figura 1.1:** Thron DAM Platform

Grazie alla condivisione degli *asset* digitali su qualsiasi canale di distribuzione si riescono ad evitare duplicazioni, facilitando inoltre la ricerca dei contenuti.
Il prodotto, fruibile attraverso un'applicazione *web*, è in grado di gestire contenuti di vario genere (documenti, video, immagini, audio e molto altro).

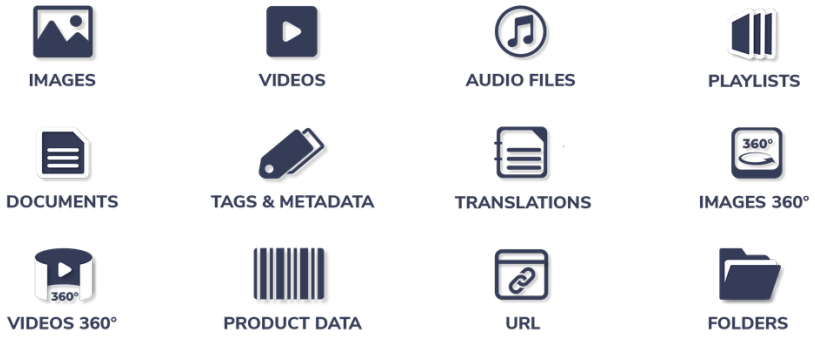



**Figura 1.2:** Contenuti supportati dalla Thron DAM Platform

La piattaforma raggruppa tutti questi contenuti e dopo una prima fase di analisi, li arricchisce con *tag* e metadati. Questo avviene tramite un potente motore semantico e analitico che aggiunge le informazioni durante tutto il ciclo di vita, aggiornandole nel tempo. In questo modo i contenuti vengono ordinati secondo *tag* ed è più semplice organizzarli e gestirli, soprattutto quando la mole di dati è elevata.

Inoltre la *Thron DAM Platform* permette di aggiungere permessi differenti, in base alle esigenze, sulla fruizione dei contenuti.
Infine, tramite l'utilizzo di algoritmi di Machine Learning, viene creato uno storico in modo tale che qualsiasi azione eseguita sia tracciata, permettendo così di:

* fornire una vista degli interessi di ciascun utente, in base ai contenuti di cui ha fruito su qualsiasi canale;
* suggerire ad ogni utente i contenuti più rilevanti, a seconda delle sue preferenze e massimizzando la soddisfazione.

## 1.3 Organizzazione aziendale

Essendo l'azienda molto articolata e suddivisa in diverse sedi non ho potuto approfondire la conoscienza dell'intera organizzazione, bensì solo la parte *Executive*, ed in particolare il reparto tecnico.
La parte *Executive* dell'azienda si suddivide in 4 reparti:

* Reparto tecnico;
* Reparto marketing;
* Reparto commerciale;
* Reparto amministrativo.

Il reparto tecnico, ossia l'area alla quale sono stato assegnato, ha subito nel tempo diversi cambiamenti. Uno di questi cambiamenti nell'organizzazione dell'organico aziendale l'ho vissuto in prima persona durante il periodo di *stage*. Al momento del mio arrivo infatti il reparto tecnico era suddiviso nei seguenti *team*:

* ***Team* Prodotti**: gestisce un'espansione del prodotto principale, ovvero il PIM (*Product Information Management*) che permette di gestire e arricchire le schede prodotto del catalogo;
* ***Team* Contenuti**: segue principalmente le tematiche legate alla *Thron DAM Platform* e a tutte le funzionalità che ne derivano;
* ***Team* Infrastruttura&Fruibilità**: si occupa del mantenimento dell'infrastruttura di rete e della fruizione dei contenuti;
* ***Team* Design**: responsabile della grafica dei servizi offerti.

Inizialmente ero stato inserito nel *team* di Infrastruttura&Fruibilità.
Dopo due settimane dal mio arrivo però l'azienda ha tenuto una riunione in cui spiegava a tutto il reparto tecnico come sarebbe cambiata l'organizzazione interna, a causa delle problematiche riscontrate dalla forte dipendenza che si veniva a creare tra i vari *team*.
Da quello che ho potuto capire per l'implementazione di un nuovo sevizio o di una nuova funzionalità, il *team* Prodotti e il *team* Contenuti dovevano necessariamente attendere un DevOps per predisporre l'infrastruttura necessaria per la loro "attivazione". Questo creava da una parte notevoli rallentamenti per il *team* Infrastruttura&Fruibilità poiché la poche figure di *DevOps* erano tutte concentrate all'interno di questo *team* e dall'altra bloccava completamente il normale ciclo di vita dei servizi sviluppati dagli altri *team*.

La soluzione adottata ha portato alla fusione del *team* Contenuti con il *team* Infrastruttura&Fruibilità per cercare di diminuire le dipendenze ed ha introdotto due nuove figure all'interno dell'organico aziendale.
Le figure professionali introdotte sono le seguenti:

* Solution Architect: progetta o modifica un'architettura di sistema con lo scopo di migliorare le funzionalità di un certo *business*;
* POM: responsabile dell'insieme dei processi aziendali. Si occupa di gestire i vari *team* all'interno del reparto tecnico.

Oltre a queste nuove figure introdotte, ogni *team* si avvale di un PO che si occupa di gestire al meglio le attività da portare a termine nel rispetto delle tempistiche preventivate per il loro rilascio.
All'interno di ogni *team* ci sono poi gli sviluppatori, che a seconda della propria competenza si suddividono in: *Frontend*, *Backend* e DevOps. Per dare un quadro generale, la Figura 1.3 rappresenta la struttura aziendale attuale.

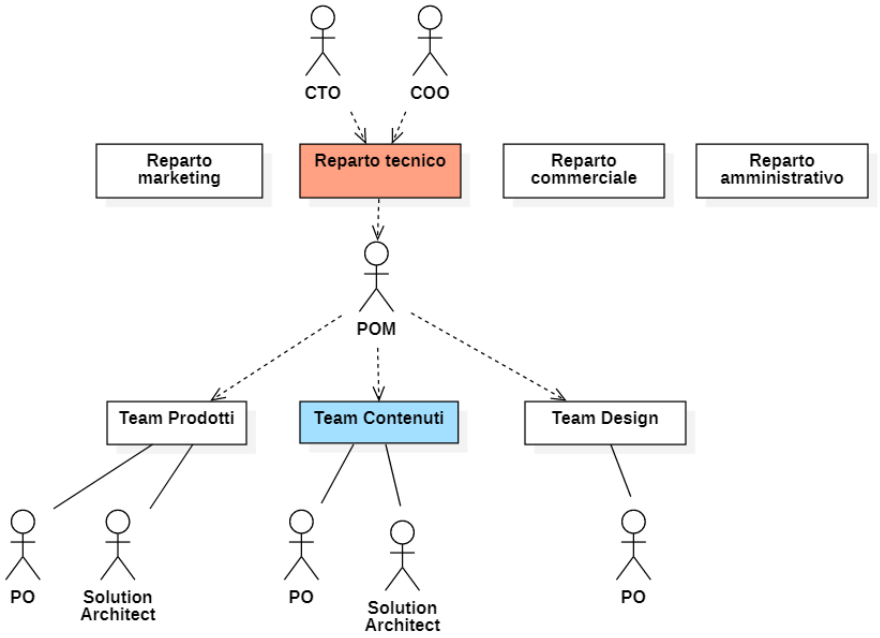



**Figura 1.3:** Struttura aziendale

## 1.4 Processi e tecnologie

### 1.4.1 Processi

THRON non usa un modello di ciclo di vita standard per lo sviluppo del suo prodotto, ma i suoi metodi si possono paragonare a quello di un modello agile di tipo Scrum.
Il progetto è ormai avviato e consolidato, quindi la sua evoluzione procede con piccoli passi incrementali, che permettono di migliorare a poco a poco il prodotto, anche

grazie ad un importante coinvolgimento del cliente, che viene spesso interpellato per suggerire modifiche e miglioramenti.

Durante gli incontri con i clienti vengono mostrate le evoluzioni della piattaforma, con dei prototipi delle migliorie, in modo da avere un riscontro da parte dei clienti sull'efficacia della soluzione raggiunta o su eventuali suggerimenti dal cliente stesso. Sempre in questi incontri non è raro che il cliente richieda nuove funzionalità alla piattaforma che vorrebbe implementate o modificate.

Dopo gli incontri con il cliente, segue un incontro interno per valutarne l'andamento, discutendo del grado di soddisfacimento del cliente e sulle sue eventuali proposte. In questi incontri interni vengono spesso presentate nuove idee per l'evoluzione della piattaforma che vengono discusse e analizzate. Le nuove idee, miglioramenti, adattamenti ed errori vengono scremati dal Project Management che le gestisce con un sistema di ticket interno all'azienda.

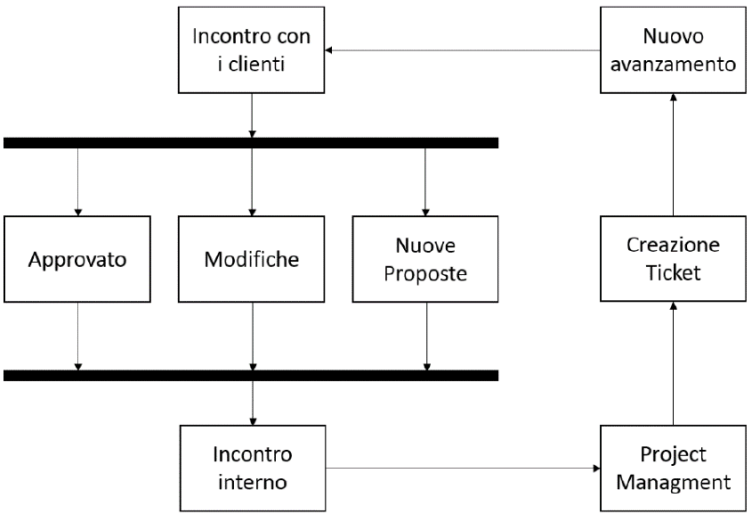



**Figura 1.4:** Diagramma di flusso delle attività interne

Utilizzando un modello agile le attività da svolgere sono distribuite all'interno di Sprint della durata di tre settimane.
Per quanto riguarda il coordinamento interno dei vari *team* del reparto tecnico si svolgono giornalmente dei *daily meeting* della durata massima di mezz'ora in cui ogni partecipante espone quali attività è riuscito a concludere il giorno precedente, quali attività si è preso in carico per la giornata e se ha riscontrato qualche problema nello svolgere il proprio lavoro.
A conclusione del *meeting* è possibile effettuare domande e sollevare perplessità riguardo alle tematiche trattate.
Oltre a ciò, un giorno alla settimana c'è una riunione di *competence* della durata di un'ora in cui gli sviluppatori, divisi per competenze appunto, discutono delle novità introdotte sul mercato, si scambiano consigli e condividono le conoscenze maturate durante la settimana.

### 1.4.2 Tecnologie di supporto

THRON occupandosi di molti settori dell'informatica, non possiede una suite di strumenti comune per lo sviluppo, perciò ogni *team* decide autonomamente gli strumenti che ritiene più opportuni per gli obiettivi e le attività che gli vengono assegnati.
In comune però troviamo gli strumenti di versionamento, comunicazione e documentazione.

Come strumento di versionamento del codice, l'azienda utilizza Git. In particolare, per poter raccogliere tutto il codice derivante dai vari progetti delle varie aree sviluppo, viene utilizzato un servizio di AWS (*Amazon Web Services*) chiamato CodeCommit. Questi strumenti possono essere utilizzati mediante CLI (*Command Line Interface*) oppure tramite la console messa a disposizione da AWS.

Per quanto riguarda la comunicazione, l'azienda utilizza Microsoft Outlook per la posta elettronica, questo perché è un software che funziona su entrambi i sistemi operativi utilizzati per lo sviluppo (Windows e OS X) e poiché consente una facile gestione di eventi e riunioni su calendario. Ogni membro dell'azienda ha un proprio indirizzo *email* con la forma: *nome.cognome@thron.com*. Oltre alle *email* si dispone di Microsoft Teams, un servizio di messaggistica istantanea per le comunicazioni di minor importanza o che hanno bisogno di una risposta rapida.

Per la documentazione, viene utilizzata la suite Office, che rappresenta lo standard per i documenti di testo. In particolare Word è usato per la stesura di documenti provvisori, informali o che necessitano di modifiche veloci. Per i documenti finali, si converte il documento in formato PDF. Excel viene usato per lo scambio di dati tabellari o qualora ci fosse il bisogno di tabelle e grafici. Power Point è usato per fare presentazioni, in modo da spiegare i concetti con più efficacia ai propri dipendenti durante le riunioni, o durante gli incontri con i clienti.
Per la condivisione della documentazione aziendale THRON utilizza una piattaforma collaborativa chiamata Confluence, mentre per il tracciamento delle issue e la gestione dei progetti ha iniziato da poco ad utilizzare Jira.

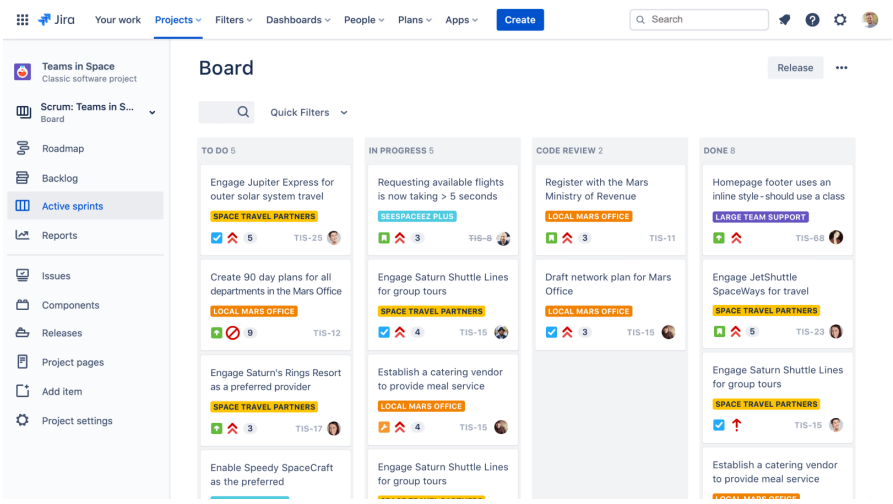



**Figura 1.5:** Atlassian Jira - Sistema di gestione dei progetti *agile*

L'ambiente di lavoro più comune è Windows, data la quantità di software a lui dedicati, ma si utilizza anche OS X soprattutto nel reparto di Design.
Ubuntu è invece lo standard sui server aziendali, per la sua stabilità, semplicità e convenienza in ambito server. Per poter testare accuratamente il comportamento dei progetti e della piattaforma sui vari sistemi operativi e browser, c'è una stanza con computer configurati a tale scopo.

## 1.5 Tipologia di clienti

THRON ha la capacità di proporre un unico prodotto per aziende che hanno differenti esigenze tra loro. Questo prodotto è un DAM intelligente che permette di migliorare l'organizzazione, la ricerca e la distribuzione dei contenuti e ottenere al tempo stesso dati strategici relativi al loro utilizzo. Il valore aggiunto che Thron fornisce è infatti la capacità di ottenere dati sugli interessi degli utenti a partire dalle informazioni sulla fruizione dei contenuti, permettendo di inviare messaggi personalizzati ad ogni singolo utente.

L'azienda mira ad una vasta ed eterogenea clientela, per questo motivo ha creato un prodotto ampio e facilmente configurabile. In questo modo è il prodotto che si adatta alle esigenze dei diversi clienti e non viceversa. Questo è un punto molto forte, infatti inizialmente il prodotto viene venduto solo con il modulo base che offre le funzionalità standard utili a tutti, poi i clienti possono, attraverso la sezione *Marketplace*, comprare o aggiungere moduli gratuiti che offrono diverse funzionalità rendendo molto flessibile la piattaforma.

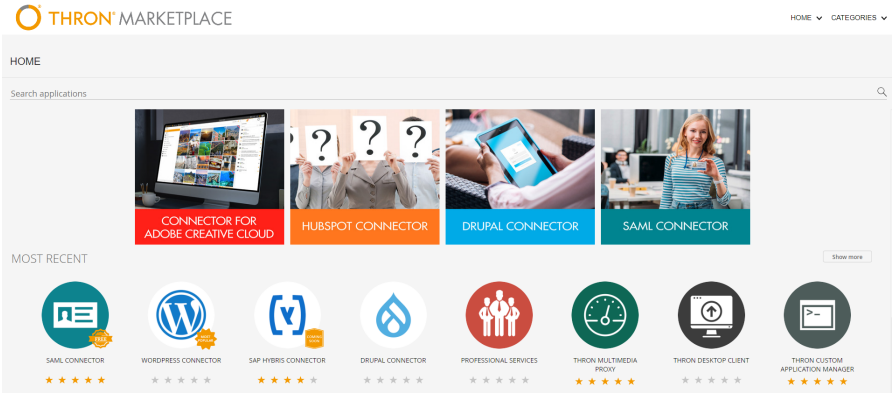



**Figura 1.6:** Marketplace di THRON - https://marketplace.thron.com/

Nel caso non sia sufficiente, l'azienda si propone di confezionare soluzioni personalizzate per il caso d'uso specifico del cliente, in modo da soddisfare sempre appieno le loro richieste e necessità. Lo stretto contatto con clienti, che hanno esigenze molto diverse tra loro, porta ad una naturale evoluzione della piattaforma, che deve innovarsi continuamente per soddisfare le richieste dei clienti. Ogni evoluzione della piattaforma, conferisce a Thron una sempre più crescente completezza delle funzionalità offerte. La quantità di clienti, uniti alla quantità di contenuti elaborati, contribuisce alla stabilità della piattaforma.

## 1.6 Propensione all'innovazione

L'azienda è alla continua ricerca di nuove tecnologie e strumenti che possano migliorare l'attuale prodotto offrendo la migliore esperienza possibile all'utilizzatore finale di Thron. Proprio per questo THRON, coltiva questo aspetto cercando personale che abbia attitudine al cambiamento e contemporaneamente esprima le proprie soluzioni ai problemi incontrati dando libero sfogo alla propria creatività.

Personalmente credo che la propensione all'innovazione che ho percepito all'interno dell'azienda sia dovuta principalmente dalle persone che ci lavorano ogni giorno. Molti membri di THRON infatti arrivano dalla mia stessa realtà universitaria. Loro come me hanno avuto la possibilità di mettersi alla prova con progetti reali, che una volta ultimati hanno portato reale valore aggiunto all'azienda.
Questo crea inevitabilmente una reale soddisfazione per il lavoro svolto e spinge le persone a rimanere a lavorare per THRON.

La maggior parte dell'innovazione a mio parere deriva proprio dai progetti di *stage* proposti poiché permettono ogni anno all'azienda di esplorare nuove tecnologie, di pensare a soluzioni differenti, di tenersi al passo con i tempi.

Questa peculiarità permette a THRON di evolversi costantemente. Altri aspetti fondamentali per il successo aziendale sono il pensiero critico e la collaborazione che contraddistinguono i vari *team*. Utlime ma non per importanza ci sono la complicità ed il senso di appartenenza che legano le persone a THRON. Tutti questi elementi uniti tra di loro portano serenità nell'ambiente di lavoro permettendo la crescita personale di ogni dipendente che indirettamente influisce sulla crescita aziendale.

# Capitolo 2

# Strategia aziendale

## 2.1 Motivazioni dello *stage*

THRON da anni crede nella collaborazione di *stage* con gli studenti dell'Università di Padova. Ogni anno infatti permette agli studenti, conosciuti tramite l'evento StageIT promosso da Assindustria Venetocentro in collaborazione con l'Università di Padova, di effettuare progetti di *stage* presso la loro sede centrale.

Lo *stage* è visto dall'azienda come un percorso preferenziale per potersi conoscere reciprocamente, in modo che possa valutare lo studente e che lo studente possa approcciarsi al mondo del lavoro e alle metodologie aziendali.
Da quello che ho potuto notare, molti dei progetti proposti dall'azienda negli anni passati altro non erano che la traduzione alle richieste dei propri clienti. In questo modo THRON ha la possibilità di anticipare l'analisi delle possibili soluzioni e sfruttare i progetti di *stage* per ottimizare i tempi e nel migliore dei casi proporre una dimostrazione del servizio implementato ai propri clienti.
Un valido esempio a supporto delle mie affermazioni è il progetto di *stage* effettuato dal mio tutor, anch'esso *ex* studente dell'Università di Padova, che nel 2016 ha implementato una soluzione per la gestione dinamica delle *thumbnail* con riconoscimento automatico del soggetto. Ad oggi questa soluzione prende il nome di RTIE e risulta essere una delle funzionalità più apprezzate ed utilizzate dai clienti di THRON.

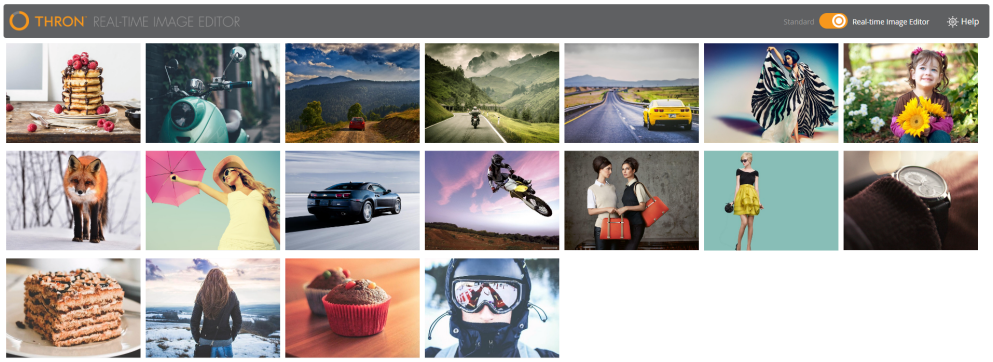



**Figura 2.1:** Real-Time Image Editor (RTIE)

Oltre a ciò THRON sfrutta gli *stage* per incrementare il proprio organico aziendale, andando alla ricerca di studenti volenterosi da poter assumere.
Per l'azienda infatti lo *stage* è un modo per testare e provare nuove soluzioni, di cui non è sicura della loro riuscita o della loro efficacia, rispetto alle soluzioni già esistenti.

Per quanto riguarda i colloqui, in THRON si dà più importanza alle *soft skill* degli studenti rispetto alle competenze tecniche perché l'azienda è consapevole che le tematiche affrontate per i progetti di *stage* non sono materia d'esame durante il percorso di studi universitario, grazie soprattutto alle precedenti esperienze avute sempre con studenti di informatica dell'Università di Padova.

Nel progetto in questione non c'era la certezza che potesse soddisfare tutti i punti richiesti, ma volevano comunque provare a vedere se era possibile realizzare quanto idealizzato, per potersi basare su dati concreti ed eventualmente prendere decisioni sul possibile utilizzo della strada sperimenta.
Nel mio caso specifico l'azienda era interessata a formare una nuova persona riguardo le tematiche di *DevOps*, un settore in espansione dove vorrebbe continuare ad investire, dato che ricopre un ruolo sempre più importante per la direzione di crescita ed evoluzione dell'azienda.

## 2.2 Introduzione al progetto

Come descritto nel capitolo precedente, nell'ultimo periodo THRON ha subito dei cambiamenti per quanto riguarda l'organizzazione aziendale. Insieme a questi cambiamenti, l'azienda, attraverso questo progetto di *stage* mira a diminuire le dipendenze che si creano tra i vari *team* del reparto tecnico all'implementazione di nuovi servizi e funzionalità da aggiungere al prodotto Thron.

Da anni ormai THRON ha dismesso l'utilizzo dei server fisici, basando tutta l'erogazione dei propri servizi tramite Cloud computing, passando quindi attraverso la rete *Internet*. Attualmente il Cloud provider a cui si appoggia THRON per l'erogazione dei propri servizi è AWS.
Per questo motivo nel tempo la figura del *DevOps* è diventata indispensabile per l'erogazione ed il mantenimento delle nuove funzionalità implementate dai vari *team*. Attualmente infatti, in azienda, la maggior parte dell'infrastruttura e delle risorse AWS utilizzate dai servizi implementati è scritta dai *DevOps* utilizzando *template* AWS CloudFormation. Questi *template* permettono la scrittura dell'infrastruttura di rete tramite file di testo in formato *JSON* o *YAML*.


**Figura 2.2:** Funzionamento AWS CloudFormation

Con il tempo però i *DevOps* hanno iniziato a riscontrare delle problematiche legate all'utilizzo di questi *template*.
Il primo è il problema del mantenimento poichè con la costante aggiunta di ulteriori integrazioni infrastrutturali da parte di AWS diventa sempre più complicato lavorare con file *JSON* o *YAML* di grandi dimensioni.
Oltre a ciò i modelli *CloudFormation* mancano di astrazione, obbligando gli sviluppatori a scrivere numerose righe di testo per indicare dettagli di livello inferiore. Infine esiste anche il problema della duplicazione del codice poichè il riuso e la successiva modifica manuale di *template* già esistenti porta ad errori difficilmente individuabili oltre che ad uno spreco di tempo.

Un esempio concreto delle dinamiche aziendali attuali all'implementazione di nuovi servizi e funzionalità da aggiungere al prodotto Thron viene visualizzato in Figura 2.3.

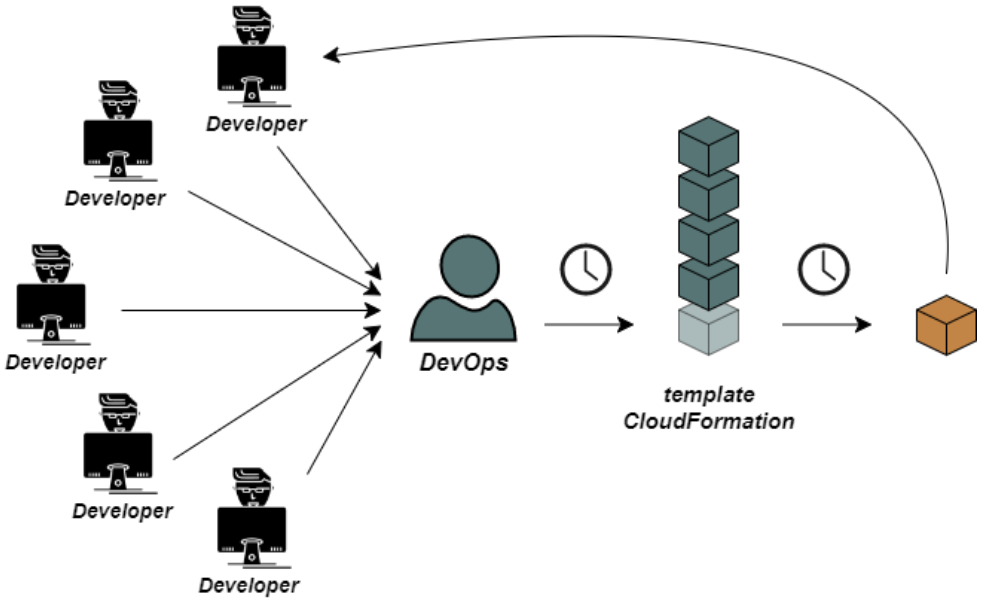



**Figura 2.3:** Dinamiche aziendali attuali per l'implementazione di un nuovo servizio

Gli sviluppatori che non fanno parte della *competence* dei *DevOps* infatti non hanno le conoscenze necessarie per predisporre l'infrastruttura adatta all'erogazione dei servizi da loro implementati. Questo porta gli sviluppatori a dover richiedere il supporto diretto di un *DevOps* ogni qual volta ci sia la necessità di erogare un nuovo servizio. Essendo attualmente i *DevOps* in numero molto inferiore rispetto agli sviluppatori di *competence* differenti si va a creare un "collo di bottiglia" che implica un rallentamento forzato nello sviluppo della maggior parte dei nuovi progetti aziendali.

Le soluzione pensata per risolvere questo problema è rendere più indipendenti gli sviluppatori dando loro la possibilità di utilizzare delle librerie per la creazione automatica delle risorse AWS utili all'erogazione dei servizi sviluppati. Così facendo si diminuisce sia il carico di lavoro dei *DevOps* che le tempistiche per l'erogazione di un nuovo servizio.

La figura 2.4 presenta la situazione appena descritta.

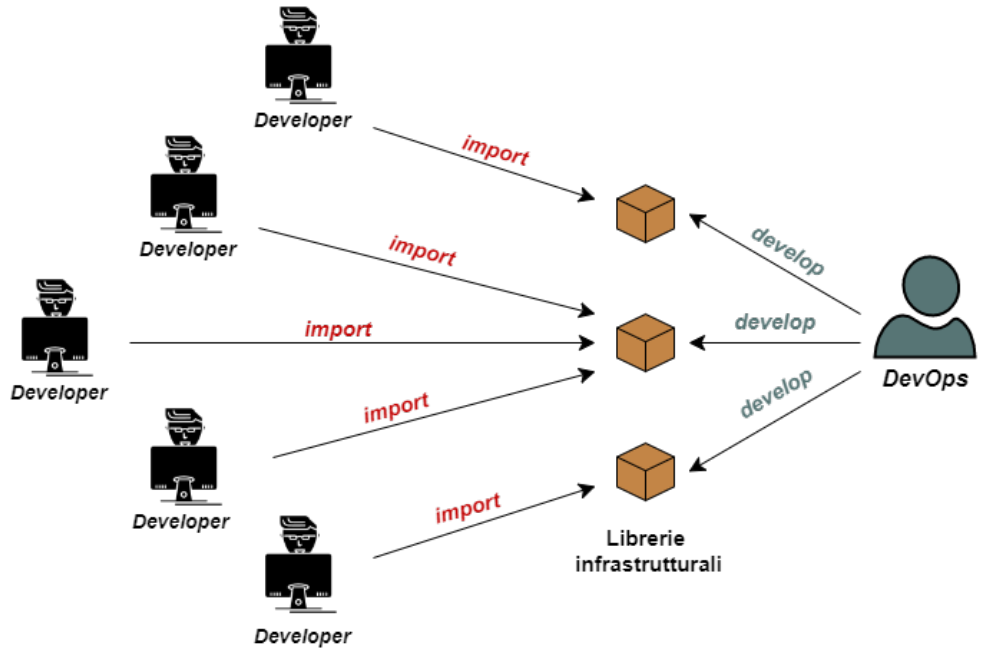



**Figura 2.4:** Dinamiche aziendali dopo lo sviluppo delle librerie infrastrutturali

Le motivazioni che hanno spinto THRON a proporre questo progetto di *stage* sono quindi il voler velocizzare la creazione di nuovi progetti, riducendo al minimo la duplicazione di codice e permettendo allo sviluppatore che inizializza un nuovo progetto abbastanza libertà per soddisfare le sue esigienze senza però uscire dalle convenzioni aziendali.

## 2.3 Obiettivi dello *stage*

L'obiettivo principale di questo *stage* era sviluppare delle librerie per la creazione di insiemi di risorse infrastrutturali riutilizzabili dai vari progetti aziendali. Queste librerie dovevano poter essere parametrizzabili in modo da fornire allo sviluppatore che inizializza un nuovo progetto abbastaza libertà per soddifare le proprie esigienze. Per fare ciò, dopo aver familiarizzato con l'architettura AWS utilizzata in azienda, il tutor ha richiesto di fare uno *scouting* sulle possibili soluzioni IaC (*Infrastructure as Code*) da utilizzare per lo sviluppo delle librerie. In secondo luogo, insieme al tutor aziendale, abbiamo analizzato il caso d'uso più ricorrente alla creazione di nuovi progetti aziendali.

Per svolgere questo progetto di *stage* sono stato inserito all'interno del *team* Contenuti, in particolare nell'*openspace* dove si trovavano tutti i *DevOps*. Questo mi ha permesso da una parte di ricevere supporto dai colleghi e dall'altra di partecipare in modo attivo alle dinamiche aziendali.

Prima dell'inizio dello *stage*, con l'aiuto del mio tutor, abbiamo individuato i seguenti obiettivi obbligatori, desiderabili e opzionali.

### 2.3.1 Obiettivi obbligatori

* Sviluppo di una o più librerie (a seconda dei casi d'uso presi in considerazione) che, grazie allo strumento di *IaC* scelto, semplifichi la definizione infrastrutturale ai membri non appartenenti alla competenza *DevOps*;
* Documentazione del processo di scelta dello strumento di *IaC* per lo svilupppo delle librerie;
* Documentazione per l'utilizzo completo della libreria, fornendo parametrizzazioni, snippet di esempio e limiti tecnici.

### 2.3.2 Obiettivi desiderabili

* Integrazione delle librerie negli *scaffold* di inizializzazione dei progetti gestiti con CI/CD;
* Implementazione dei *test* di unità da eseguire all'interno delle *pipeline*;
* Implementazione dei *test* di integrazione infrastrutturale da eseguire all'interno delle *pipeline*.

### 2.3.3 Obiettivi opzionali

* Miglioramento dell'integrazione con i sistemi di monitoraggio aziendali;
* Implementazione di un meccanismo di rollback automatico del *deploy* in caso una versione presenti dei problemi;
* Calcolo dei costi fissi e variabili per la creazione delle risorse AWS.

## 2.4 Vincoli dello *stage*

### 2.4.1 Vincoli metodologici

Durante il periodo di *stage*, l'azienda ha richiesto di stilare un documento in cui riportare lo *scouting* effettuato per le possibili soluzioni *IaC*. Mi ha richiesto infatti di analizzare con precisione i pro e contro di ogni possibile soluzione, per avere un quadro generico valido consultabile dai *DevOps*.

Per tutta la durata dello *stage*, ho sempre lavorato a stretto contatto con il tutor aziendale. Le postazioni di lavoro erano una accanto all'altra, in modo tale da poter verificare quotidianamente e con facilità l'andamento delle attività ed eventualmente ricevere supporto in caso di problemi.

Durante l'ultima settimana di *stage*, l'azienda ha richiesto la realizzazione di una presentazione che avrei dovuto esporre a tutto l'organico aziendale nel mio ultimo giorno lavorativo.
Tale presentazione, esposta in forma verbale con il sostegno di diapositive, è servita per illustrare il lavoro svolto durante il progetto esponendo i risultati raggiunti e le criticità emerse.

### 2.4.2 Vincoli temporali

Lo svolgimento dello *stage* ha previsto una durata di 312 ore complessive di lavoro. Queste ore sono state effettuate nell'arco di due mesi: 8 settimane lavorative da 40 ore ciascuna.
L'orario lavorativo effettuato coincide con l'orario di lavoro aziendale, ossia 8 ore complessive che vanno dalle ore 8:30 alle ore 13:00 e dalle ore 14:00 alle ore 18:00.

Prima dell'inizio dello *stage* ho redatto, insieme al tutor aziendale, il piano di lavoro nel quale abbiamo riportato la scansione temporale delle attività da svolgere.
La distribuzione pianificata nel documento viene riportata nel capitolo 3.1.

Il tutor non ha comunque richiesto di seguire in maniera ferrea le tempistiche riportate nel piano di lavoro in quanto per alcune attività non era facilmente identificabile il tempo necessario per completarle, ad esempio non si sapeva quanti casi d'uso si sarebbero potuti analizzare.
Proprio per questo in base alla priorità delle attività istanziate il piano poteva essere dilatato a piacimento, previa consultazione del tutor.

### 2.4.3 Vincoli tecnologici

Per il progetto di *stage*, l'azienda mi ha consentito massima libertà sulla configurazione dell'ambiente di lavoro. Mi hanno dato invece dei vincoli allo *scouting* da effettuare per le possibili soluzioni *IaC*. La soluzione da adottare per lo sviluppo delle librerie doveva necessariamente avere le seguenti caratteristiche:

* Compatibilità con l'attuale *Cloud provider* aziendale (AWS);
* Possibilità di *rollback* automatico in caso di guasti;
* Implementabile tramite un linguaggio di programmazione.

Per quanto riguarda l'implementazione della libreria, il tutor ha solamente consigliato un linguaggio di programmazione a discapito di altri ma effettivamente non mi ha imposto alcun vincolo.
Naturalmente dovendo replicare il comportamento dei casi d'uso analizzati per la creazione di risorse AWS, i vincoli indirettamente imposti erano di seguire il funzionamento di questi ultimi.

## 2.5 Motivazioni personali

Durante l'evento StageIT, ho sfruttato l'opportunità concessami per parlare con il maggior numero possibile di aziende. Molti dei progetti proposti li ho trovati interessanti e ho quindi voluto approfondire la mia comprensione andando successivamente ad effettuare colloqui individuali presso le singole aziende. Questo mi ha permesso di pormi dei obiettivi personali sulle le tematiche e finalità dello *stage*.

Per questo, ho deciso di affrontare uno *stage* che da una parte mi permettesse di ampliare le possibilità di trovare lavoro all'interno dell'azienda ospitante, e dall'altra potesse servirmi come bagaglio personale. Il mio obiettivo principale però era trovare uno *stage* in grado di formarmi su argomenti e tematiche utili, difficilmente affrontate durante il mio corso di studi ma ampliamente richieste dal mercato del lavoro.

Per questo motivo la mia scelta è ricaduta su THRON, poiché mi ha fatto da subito una buona impressione. La loro volontà di instaurare un legame forte con i propri dipendenti, in modo da offrire un ambiente di lavoro sereno, ma soprattutto le tematiche innovative trattate quotidianamente mi hanno spinto a prendere la mia decisione. Nello specifico, mi hanno incuriosito le tematiche di *Cloud computing* affrontate per il progetto di *stage* e la mia personale scoperta della figura del *DevOps*, di cui fino a quel momento ignoravo completamente l'esistenza.

Lo *stage* con THRON mi permetteva quindi di soddisfare i miei requisiti di formazione, e mi permetteva di mettermi in gioco affrontando sfide e problemi mai affrontati.

# Capitolo 3

# Svolgimento dello *stage*

## 3.1 Pianificazione delle attività

Prima dell'inizio dello *stage*, con il tutor aziendale, abbiamo redatto un piano di lavoro di 312 ore. La Tabella 3.1 presenta le attività preventivate inizialmente da tale piano.

| Settimana | Descrizione attività | Ore |
|---|---|---|
| **1ª** | - Orientamento iniziale in THRON con configurazione dell'ambiente di lavoro | 4 |
| | - Formazione sull'architettura AWS aziendale | 20 |
| | - Analisi dei requisiti funzionali e di vincolo | 16 |
| **2ª** | - Analisi dei *pattern* aziendali più comuni per l'implementazione di un nuovo servizio | 8 |
| | - Studio del caso d'uso selezionato | 8 |
| | - *Scouting* delle possibili soluzioni di *IaC* per lo sviluppo della libreria | 24 |
| **3ª** e **4ª** | - Sviluppo della libreria per la gestione delle risorse infrastrutturali | 80 |
| **5ª** | - Implementazione dei *test* di unità e di integrazione | 40 |
| **6ª** | - Integrazione della libreria sviluppata nella *pipeline CI/CD* aziendale | 24 |
| | - Implementazione del sistema di monitoraggio | 16 |
| **7ª** | - Implementazione del *rollback* automatico in caso di problemi | 16 |
| | - Calcolo dei costi fissi e variabili per l'utilizzo delle risorse AWS | 16 |

*Continua nella pagina successiva*

| Settimana | Descrizione attività | Ore |
|---|---|---|
| **8ª** | - Stesura della documentazione di progetto | 24 |
| | - Stesura delle diapositive per la presentazione interna del lavoro svolto | 16 |

**Tabella 3.1:** Piano di lavoro inizialmente preventivato

Tuttavia, verso la fine della seconda settimana di lavoro, il piano inizialmente preventivato risultava essere fin troppo ottimistico. Per prevenire l'insorgere di ritardi e possibili problematiche, a seguito di un confronto con il tutor aziendale, abbiamo concordato una modifica alla pianificazione. La Tabella 3.2 presenta le modifiche apportate al piano di lavoro.

| Settimana | Descrizione attività | Ore |
|---|---|---|
| **2ª** e **3ª** | - Analisi dei *pattern* aziendali più comuni per l'implementazione di un nuovo servizio | 8 |
| | - Studio del caso d'uso selezionato | 16 |
| | - *Scouting* delle soluzioni di *IaC* per lo sviluppo della libreria | 30 |
| | - Studio delle risorse AWS da implementare | 26 |
| **4ª**, **5ª** e **6ª** | - Familiarizzazione con la soluzione di *IaC* selezionata per lo sviluppo della libreria | 24 |
| | - Sviluppo della libreria per la gestione delle risorse infrastrutturali | 56 |
| | - Sviluppo di un'applicazione per simulare la chiamata alla libreria | 32 |
| **7ª** | - Implementazione dei *test* di unità e di integrazione | 20 |
| | - Integrazione della libreria sviluppata nella *pipeline CI/CD* | 20 |
| **8ª** | - Stesura della documentazione di progetto | 24 |
| | - Stesura delle diapositive per la presentazione interna del lavoro svolto | 16 |

**Tabella 3.2:** Piano di lavoro aggiornato in seguito alle modifiche

Già al mio primo giorno di lavoro in THRON, dopo aver fatto un *tour* aziendale insieme al mio tutor, mi hanno assegnato la postazione di lavoro all'interno dell'*openspace* del *team* Contenuti.
Questo mi ha permesso di stare a stretto contatto con i colleghi di lavoro.
Dopo qualche giorno dal mio arrivo, una volta configurata la postazione di lavoro e familiarizzato con l'ambiente circostante, il CTO mi ha invitato a partecipare ai *daily meeting* aziendali. La mia presenza a queste riunioni giornaliere è servita a farmi

sentire parte integrante dell'organico aziendale ed ha aiutato a capire meglio alcune dinamiche aziendali.
Oltre ai *daily meeting*, una volta alla settimana partecipavo alla riunione di *competence* dei *DevOps*, in cui si discuteva per un'ora delle novità nell'ambito di interesse e si condividevano le conoscenze acquisite durante la settimana con i colleghi.
Entrambe le riunioni, sia quella giornaliera che quella settimanale hanno contribuito ad accrescere il mio interesse per l'azienda.
Grazie a ciò sono rimasto sempre in contatto con il mio tutor permettendo la condivisione continua delle attività svolte.

Durante l'attività di *scouting*, ho reddatto due documenti distinti, uno contente un riassunto delle considerazioni e dei dettagli rilevanti analizzati durante lo studio, e l'altro contenente una tabella riassuntiva che confrontava le possibili soluzioni analizzate.
Al completamento dello stage invece, ho preparato una piccola presentazione, supportata da diapositive, da mostrare al mio tutor e al resto del *team* di sviluppo. All'interno di questa presentazione ho spiegato tutto il lavoro svolto, gli accorgimenti, i risultati ottenuti e le difficoltà incontrate durante il mio percorso di *stage*.

## 3.2 Analisi preventiva dei rischi

Nei primi giorni di lavoro il tutor mi ha spiegato l'architettura AWS aziendale. THRON ha un'architettura estremamente complessa ed articolata. Inizialmente infatti ho riscontrato parecchie difficoltà poiché il continuo flusso di nozioni trasmesse dal mio tutor risultava difficile da elaborare correttamente.

Per questo, dopo essermi ambientato all'interno dell'azienda, verso la fine della prima settimana di lavoro, ho iniziato a maturare un pensiero critico su quello che era il progetto di *stage* proposto dall'azienda.
Parlando con il tutor infatti ho notato che il progetto proposto non aveva una linea ben definita. Inizialmente infatti lo scopo principale era quello di analizzare un paio di casi d'uso di *pattern* architetturali comuni alla creazione di un nuovo servizio o funzionalità.
Dopo che il tutor mi ha spiegato a grandi linee il primo caso d'uso da replicare, ho pensato che data la difficoltà delle tematiche affrontate sarebbe stato il caso di individuare i possibili rischi e le rispettive contromisure da adottare per mitigarli. Tali rischi sono raggruppati in tabella 3.3, la quale contiene la descrizione del rischio, la sua probabilità di occorrenza, il suo grado di pericolosità e le contromisure da adottare nel caso esso si presenti.

| Rischio | Descrizione | Grado | Contromisure |
|---|---|---|---|
| Difficoltà tecnologiche | Il tempo richiesto per l'apprendimento delle nuove tecnologie potrebbe causare ritardi nello sviluppo | **Occorrenza:** Media **Pericolosità:** Alta | Chiedere chiarimenti al tutor aziendale sulle nuove tecnologie che stanno rallentando l'avanzamento del progetto |
| Difficoltà di realizzazione | Le risorse AWS create nell'ambiente di prova potrebbero non avere lo stesso comportamenento nell'ambiente di sviluppo | **Occorrenza:** Media **Pericolosità:** Alta | Segnalare e analizzare il problema con il tutor aziendale in modo da trovare una nuova soluzione alternativa |
| Difficoltà di realizzazione | La libreria sviluppata potrebbe non essere integrabile con la *pipeline CI/CD* aziendale poichè quest'ultima risulta ancora in fase di sviluppo | **Occorrenza:** Alta **Pericolosità:** Bassa | Rivedere ed eventualmente sostituire gli obiettivi che non possono essere soddisfatti |
| Difficoltà nelle tempistiche | Lo sviluppo del progetto di *stage* potrebbe non finire nel periodo pianificato a causa di ritardi | **Occorrenza:** Media **Pericolosità:** Alta | Avvisare il tutor nel caso in cui avvengano rallentamenti rispetto al piano di lavoro prefissato e aggiornare la pianificazione di conseguenza |
| Impegni personali | Durante il periodo di *stage* è possibile che avvengano assenze dovute a problemi di salute o impegni personali | **Occorrenza:** Bassa **Pericolosità:** Bassa | Pianificare il lavoro in modo da avere giorni per recuperare le assenze |

**Tabella 3.3:** Analisi dei rischi

## 3.3 Analisi dei requisiti

### 3.3.1 Caso d'uso analizzato

Come descritto in precedenza, dopo aver familiarizzato con l'architettura AWS aziendale, insieme al mio tutor siamo andati ad analizzare quelli che erano i *pattern* infrastrutturali più comuni alla creazione di un nuovo progetto aziendale.

Il caso d'uso più comune in assoluto consisteva nella creazione di un *service* ECS (*Elastic Container Service*), ossia un servizio di gestione dei containers altamente scalabile e veloce, agganciato ad un ALB (*Application Load Balancer*), il quale si occupa di distribuire le richieste in ingresso dei *client*.
(Per maggiori informazioni visitare il capitolo 3.6).

Per analizzare il caso d'uso il tutor mi ha consegnato il *template* CloudFormation di un progetto aziendale complesso, di cui dovevo analizzare la parte relativa al *delivery* del *service* ECS.
Come primo approccio l'ho trovato abbastanza complicato poiché non avendo nessuna conoscenza in questo ambito mi ritrovavo a leggere *file* `.yaml` che configuravano risorse AWS di cui ignoravo completamente l'esistenza.
Per questo motivo ho speso molto tempo a studiarmi la documentazione di AWS e a chiedere delucidazioni al tutor che, per mia fortuna, era sempre molto disponibile e comprensivo.

Essendo il progetto di *stage* finalizzato allo sviluppo di una libreria per la creazione dell'infrastruttura base per l'erogazione di nuovi progetti aziendali, le interazioni da parte dell'utilizzatore (che nel nostro caso è lo sviluppatore che importa la libreria), dovevano essere ovviamente ridotte allo stretto necessario.

Per rappresentare quanto descritto ho utilizzato i diagrammi dei casi d'uso (in inglese *Use Case Diagram*), ossia diagrammi di tipo UML dedicati alla descrizione delle funzioni o servizi offerti da un sistema, secondo percezione ed utilizzo da parte degli attori che interagiscono col sistema stesso.

Inizialmente l'idea era quella di analizzare un caso d'uso, replicarne il comportamento e poi passare al caso d'uso successivo.
Date le difficoltà riscontrate nella familiarizzazione con tutte le tecnologie, risorse e servizi offerti da AWS mi sono limitato ad analizzare il caso d'uso precedentemente descritto.
Avendo quindi analizzato un solo caso d'uso durante lo *stage* il diagramma risulta semplice da descrivere.


**Figura 3.1:** *Use Case* - UC1: Creazione dell'infrastruttura base

## UC1: Creazione dell'infrastruttura base

**Attori Principali:** Sviluppatore applicativi.

**Precondizioni:** Lo sviluppatore deve aver installato correttamente la libreria all'interno del proprio progetto.

**Descrizione:** Tramite la parametrizzazione del costruttore lo sviluppatore può configurare, entro le limitazioni previste dalla libreria, l'infrastruttura necessaria all'erogazione del proprio progetto.

**Postcondizioni:** Viene creata l'infrastruttura base per poter utilizzare correttamente il progetto sviluppato.

### 3.3.2 Requisiti

Durante lo *stage* ho eseguito due volte l'analisi dei requisiti. La prima è servita per lo *scouting* della soluzione di *IaC* più consona da utilizzare per l'implementazione della libreria infrastrutturale.
La seconda si è resa necessaria per determinare quali risorse AWS la libreria sviluppata dovesse implementare e collegare tra loro.

Il codice dei requisiti è così strutturato:

R[*fase*][*priorità*][*tipologia*][*codice*]

Dove la *fase* si riferisce a una delle seguenti:

* **1** : Riguarda la fase di raccolta dei requisiti necessaria per determinare la soluzione *IaC* più consona;
* **2** : Riguarda la fase di raccolta dei requisiti necessaria per lo sviluppo della libreria e dell'applicazione che la richiama.

Dove la *priorità* si riferisce a uno dei seguenti livelli:

* **O** : Obbligatorio;
* **D** : Desiderabile;
* **F** : Facoltativo.

Dove la *tipologia* si riferisce a uno dei seguenti tipi di requisito:

* **F** : Funzionale, ovvero funzione o servizio offerto dal sistema;
* **P** : Prestazionale, ovvero specifica prestazionale richiesta;
* **Q** : Qualitativo, ovvero condizione di validità per una funzione o servizio;
* **V** : Di vincolo, ovvero definiscono i requisiti che descrivono i vincoli imposti dall'azienda.

Nella fase riguardante lo *scouting*, i requisiti elencati sono stati tutti dedotti in seguito a delle decisioni prese internamente durante un *meeting* di *competence* insieme a tutti i *DevOps*.

Un esempio di come ho specificato nel documento dell'analisi dei requisiti lo *scouting* delle possibili soluzioni *IaC* è il seguente:

* **R1OV1**: la soluzione *IaC* deve essere compatibile con l'attuale *Cloud provider* aziendale. (Fonte: interna)

La Figura 3.2 presenta i requisiti di vincolo principali individuati dai *DevOps* per la possibile soluzione *IaC* da utilizzare.

I requisiti individuati per lo sviluppo della libreria e dell'applicazione che la richiama invece sono emersi dallo studio di alcuni *template CloudFormation* messi a disposizione dal mio tutor durante l'analisi dei *pattern* infrastrutturali più comuni alla creazione di progetti aziendali.

La tabella 3.4 presenta un riepilogo dei requisiti emersi durante l'analisi efettuata.


**Figura 3.2:** Requisiti di vincolo per la soluzione di *IaC* da utilizzare

| Tipologia | Obbligatorio | Desiderabile | Facoltativo | Totale |
|---|---|---|---|---|
| **Funzionale** | 4 | 1 | 1 | 6 |
| **Prestazionale** | 0 | 0 | 0 | 0 |
| **Qualitativo** | 1 | 2 | 0 | 3 |
| **Di vincolo** | 5 | 3 | 0 | 8 |
| **Totale** | 10 | 6 | 1 | 17 |

**Tabella 3.4:** Riepilogo dei requisiti

## 3.4 *Scouting* delle soluzioni di *IaC*

Dopo un'attenta analisi dei requisiti ho iniziato lo *scouting* delle possibili soluzioni *IaC* per poter sviluppare la libreria infrastrutturale.
Prima di iniziare a parlare dello *scouting* però è utile capire per quale motivo le soluzioni *IaC* nel tempo stiano riscontrando sempre più successo tra la *community* dei *DevOps*.
Le soluzione di *IaC* consentono agli sviluppatori di codificare l'infrastruttura in un modo da rendere il *provisioning* automatizzato, più veloce e ripetibile.
Le componenti chiave infatti per i *DevOps* sono: il controllo delle versioni, l'integrazione continua e l'implementazione continua.

Le varie soluzioni *IaC* sono sempre più utilizzate poiché contribuiscono a:

* **Migliorare la velocità**: l'automazione è più veloce rispetto allo spostarsi manualmente in un'interfaccia quando hai bisogno di eseguire l'implementazione e/o la connessione di risorse;

* **Aumentare l'affidabilità**: se l'infrastruttura è grande infatti, diventa facile sbagliare la configurazione di una risorsa o eseguire il *provisioning* dei servizi

nell'ordine errato. Con le *IaC*, il *provisioning* e la configurazione delle risorse viene sempre eseguito esattamente come dichiarato;

* **Prevenire una differenza della configurazione**: una differenza della configurazione si verifica quando la configurazione che ha eseguito il *provisioning* del tuo ambiente non corrisponde più all'ambiente effettivo;
* **Supportare la sperimentazione, l'esecuzione di *test* e l'ottimizzazione**: poiché la soluzione *IaC* accelera e semplifica in misura così considerevole l'esecuzione del *provisioning* di una nuova infrastruttura, si possono apportare delle modifiche sperimentali e testarle senza investire molto tempo e tante risorse permettendo di espandere rapidamente la nuova infrastruttura per la produzione.

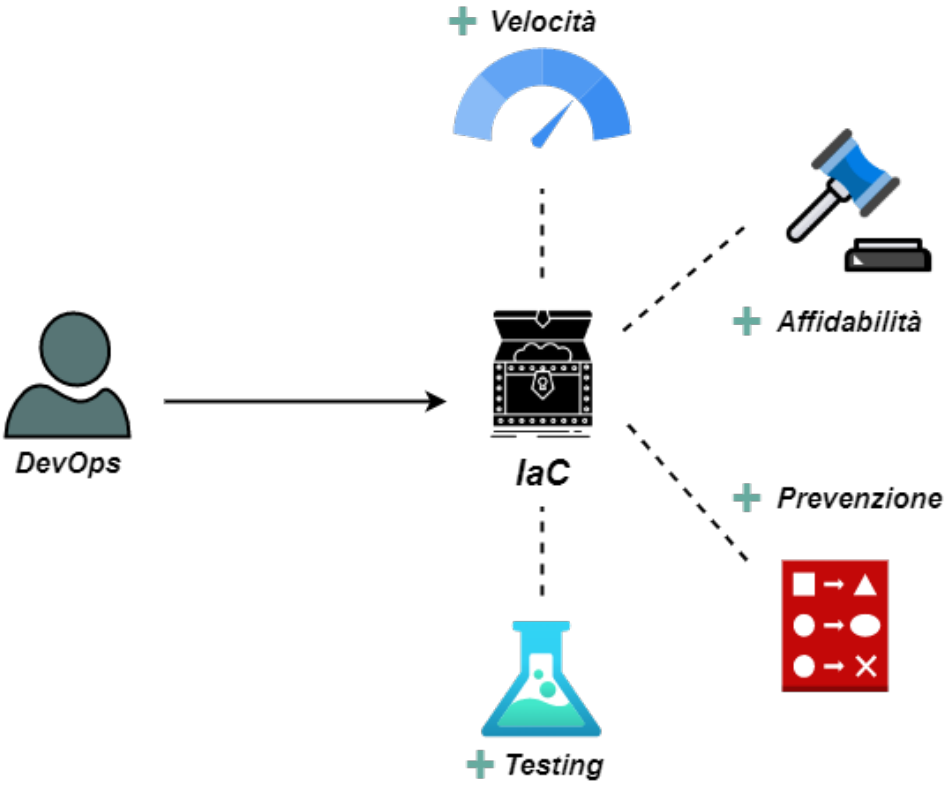



**Figura 3.3:** Vantaggi utilizzo *Infrastructure as Code*

Le soluzioni esaminate sono state molte ma dovendo fare un paragone sensato mi sono limitato ad esaminare nel dettaglio quelle che rispettavano maggiormente i requisiti di vincolo descritti in precedenza.

Le soluzioni *IaC* che ho confrontato sono le seguenti:

* **AWS CDK** (descritto nella sezione 3.4.1);
* **Terraform** (descritto nella sezione 3.4.2);
* **Pulumi** (descritto nella sezione 3.4.3).

### 3.4.1 AWS CDK

Come prima soluzione *IaC* ho analizzato AWS CDK, la quale consente di definire le risorse *Cloud* di AWS tramite i più comuni linguaggi di programmazione.
Questa soluzione, sviluppata e successivamente rilasciata da AWS nel 2019, permette di facilitare il *provisioning* di applicazioni *Cloud* utilizzando la potenza espressiva e la familiarità dei linguaggi di programmazione.
Il CDK offre componenti di alto livello, chiamati costrutti, che preconfigurano le risorse *Cloud* con impostazioni predefinite.
AWS CDK altro non è che un *transpiler* che produce AWS *Cloud Assembly*, un formato intermedio composto da modelli CloudFormation e altri *file* che carica sul servizio CloudFormation, il quale agisce da motore di distribuzione per il *provisioning* sicuro e ripetibile delle risorse (Per maggiori informazioni su AWS CDK vedere il capitolo 3.5).
Di seguito analizzo quelli che sono i pro e i contro di questa soluzione:

**Pro:**

* Utilizzo dei linguaggi di programmazione più comuni;
* Possibilità di *rollback* automatico in caso di guasti;
* Aggiornamento immediato in caso di nuove funzionalità rilasciate da AWS;
* Offre un'ottima documentazione ed ha un'ampia *community* di sviluppatori;

**Contro:**

* Vincolato all'utilizzo di AWS come unico *Cloud provider*;
* Soffre di alcune limitazioni date da AWS CloudFormation;

### 3.4.2 Terraform

Come seconda soluzione *IaC* ho analizzato Terraform[1]. Questa soluzione *IaC* nasce da HashiCorp nel 2014. Attualmente considerata *leader* nel mercato delle soluzioni *IaC* data la sua longevità, la sua ottima documentazione e la *community* da cui è supportata.
Terraform utilizza un linguaggio proprietario dichiarativo chiamato HCL (*HashiCorp Configuration Language*) per descrivere l'infrastruttura sul *Cloud*. Poiché utilizza una sintassi semplice, può eseguire il *provisioning* dell'infrastruttura in più *Cloud provider*. Terraform crea e gestisce risorse su piattaforme *Cloud* attraverso le loro API. Infatti i *provider* consentono a Terraform di lavorare praticamente con qualsiasi loro piattaforma o servizio dotato di un'API accessibile. Il *workflow* principale di Terraform consiste in tre fasi, rappresente di seguito tramite un'immagine esplicativa (Figura 3.4).
Oltre a questa soluzione tengo a sottolineare che ho esaminato anche il *Cloud Development Kit*, attualmente in fase di sviluppo, di Terraform (CDKTF[2]). Con questa nuova implementazione anche Terraform consentirà agli sviluppatori di implementare l'infrastruttura di rete tramite linguaggi di programmazione, restando di fatto aggiornata nel mercato delle *IaC*. Guardando la documentazione del CDKTF, gli stessi sviluppatori del servizio ne sconsigliano l'utilizzo per l'ambiente di produzione poichè essendo ancora in una versione *alpha* è ancora piena di *bug* e potrebbe portare a degli

[1] *Terraform*. URL: https://www.terraform.io/.
[2] *Cloud Development Kit for Terraform (CDKTF)*. URL: https://www.terraform.io/cdktf.

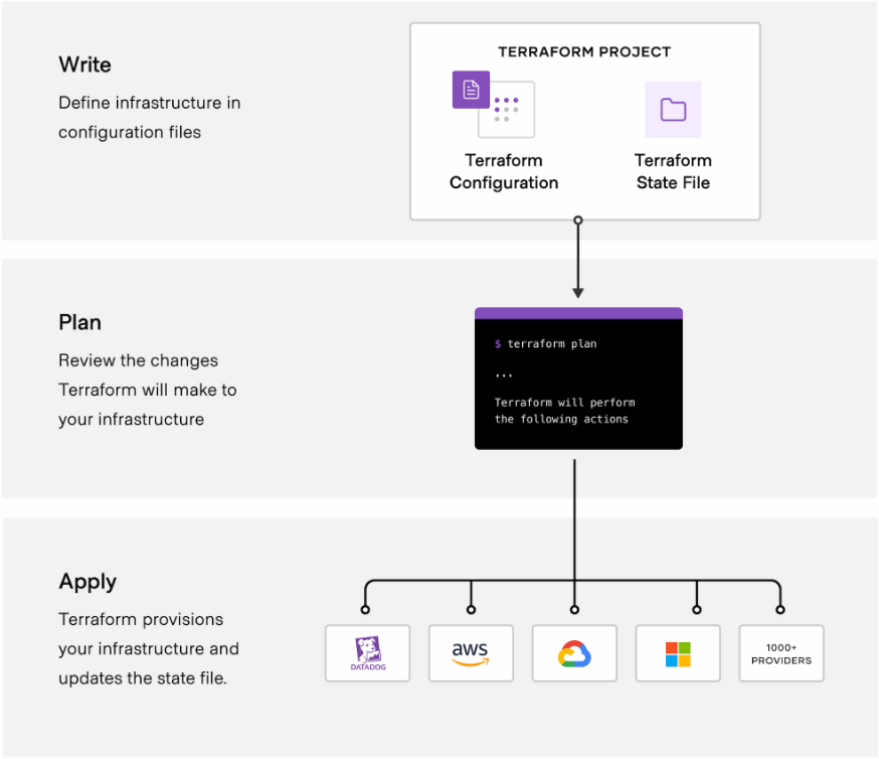



**Figura 3.4:** *Workflow* principale di Terraform

spiacevoli inconvenienti. Per questa ragione l'ho esclusa dal confronto finale. Di seguito analizzo quelli che sono i pro e i contro di questa soluzione *IaC*:

**Pro:**

- Possibilità di astrazione dal *Cloud provider*;
- *Leader* nel mercato delle *Infrastructure as Code*;
- Soluzione più utilizzata in giro per il mondo;
- Offre un'ottima documentazione ed ha un'ampia *community* di sviluppatori;

**Contro:**

- Utilizzo di un linguaggio dichiarativo proprietario, che ha un'alta curva di apprendimento;
- Non offre la possibilità di effettuare un *rollback* automatico (anche se ho visto esserci soluzioni alternative per aggirare il problema);
- Piano aziendale costoso;

### 3.4.3 Pulumi

Come ultima soluzione *IaC* ho infine analizzato Pulumi[3], un progetto *open source* nato nel 2019 con l'intento di risolvere le limitazioni delle soluzioni analizzate in precedenza.

[3] *Pulumi*. URL: https://www.pulumi.com/.

Similmente ad AWS CDK permette di scrivere infrastrutture tramite l'ausilio dei classi linguaggi di programmazione (JavaScript, TypeScript, Python, Java, .NET e Go) e condivide concetti simili anche per quanto riguarda la modularità e astrazione di alto livello per l'incapsulamento di più risorse.
Anche questa soluzione, come Terraform, supporta numerosi *Cloud provider*.
La differenza sostanziale rispetto ad AWS CDK è nel modo in cui le due soluzioni interpretano i linguaggi imperativi e sul come distribuiscono le risorse.
In Pulumi infatti, c'è un *open-source engine* che comprende i linguaggi imperativi e li comunica direttamente con i vari *Cloud provider* per distribuire l'infrastruttura. Il *workflow* principale di Pulumi è rappresento di seguito tramite un'immagine esplicativa (Figura 3.5).

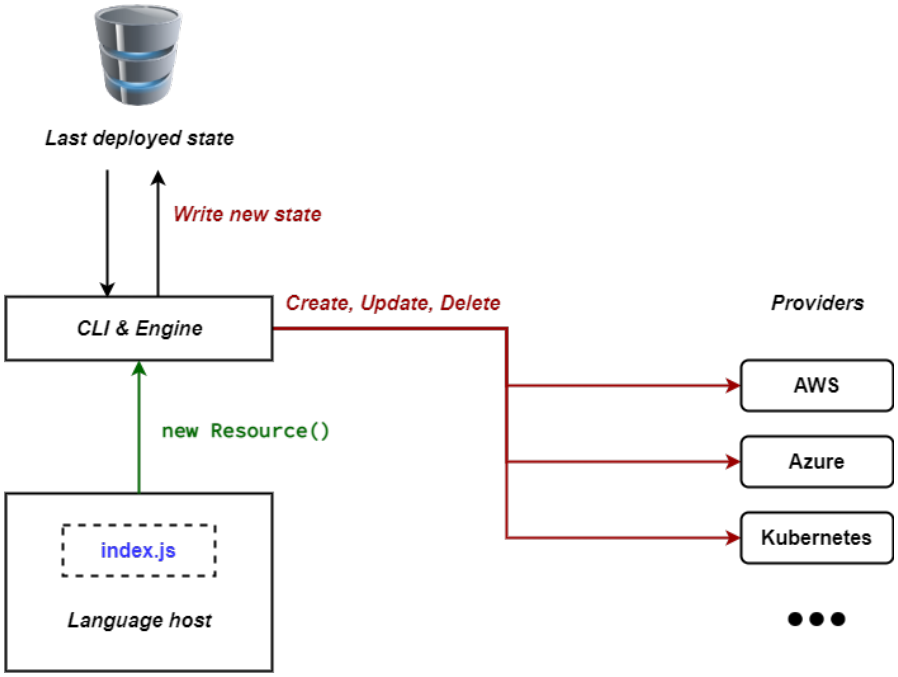



**Figura 3.5:** *Workflow* principale di Pulumi

Di seguito analizzo quelli che sono i pro e i contro di questa soluzione *IaC*:

**Pro:**

* Possibilità di astrazione dal *Cloud provider*;
* Utilizzo dei linguaggi di programmazione più comuni;
* Possibilità di *rollback* automatico in caso di guasti;
* Permette la traduzione e l'utilizzo di *template* scritti con altre *IaC*.

**Contro:**

* Offre una documentazione meno dettagliata a causa della novità del progetto;
* Supportato da una piccola *community*.

### 3.4.4 Confronto delle soluzioni *IaC* analizzate

Una volta studiati tutti i pro e i contro delle tre soluzioni analizzate, ho creato un documento che riassumeva le principali differenze.
In questo documento infatti, ho rappresentato tramite una tabella il confronto delle soluzioni *IaC* così da consentire ai *DevOps* di avere una visione completa senza dover andare troppo nel dettaglio.
La Figura 3.6 rappresenta l'idea alla base del documento redatto e serve per visualizzare la motivazioni che mi hanno spinto a scegliere una soluzione a discapito delle altre.

| | AWS CDK | Terraform | Pulumi |
|---|---|---|---|
| Compatibilità con AWS | ✓ | ✓ | ✓ |
| Linguaggio imperativo | ✓ | ✗ | ✓ |
| Astrazione dal Cloud Provider | ✗ | ✓ | ✓ |
| Rollback Automatico | ✓ | ✗ | ✓ |
| Ampia Documentazione | ✓ | ✓ | ✗ |
| Solida Community | ✓ | ✓ | ✗ |

**Figura 3.6:** Confronto delle *IaC* analizzate

### 3.4.5 Motivazioni soluzione adottata

Dopo aver analizzato con cura le possibili soluzioni di *IaC* confrontate, ho scelto di utilizzare AWS CDK per i seguenti motivi:

1. Essendo AWS l'attuale *Cloud provider* utilizzato in azienda, qualsiasi nuova funzionalità aggiunta ad AWS sarebbe subito disponibile per AWS CDK, cosa non valida per quanto riguarda le altre soluzioni;

2. La documentazione rilasciata risulta essere curata nei minimi dettagli. Fornisce *snippet* di codice d'esempio per qualsiasi linguaggio supportato;

3. AWS ha una *community* ampia e consolidata;

4. Infine ho tenuto in considerazione anche la questione dei costi. Utilizzare AWS CDK infatti non porterebbe alcun costo aggiuntivo all'azienda poichè paga già per i servizi offerti da AWS. L'utilizzo invece delle altre due soluzioni, comporterebbe dei costi aggiuntivi per THRON.

## 3.5 Familiarizzazione con AWS CDK

Inizialmente, per prendere confidenza con la soluzione *IaC* selezionata per lo sviluppo della libreria, ho realizzato dei progetti di prova seguendo le linee guida offerte dalla documentazione di AWS.
Questo mi ha permesso di avere un primo approccio pratico e concreto di quello che avrei dovuto implementare successivamente.

Sfortunatamente però AWS CDK si è rivelato estremamente complesso da utilizzare poiché a mio parere necessità di una buona conoscenza pregressa di AWS CloudFormation e delle risorse AWS.
Per questo motivo ho dovuto passare altro tempo a studiarmi per bene il funzionamento che sta alla base del CDK.

La Figura 3.7 presenta la struttura di base di un'applicazione AWS CDK.

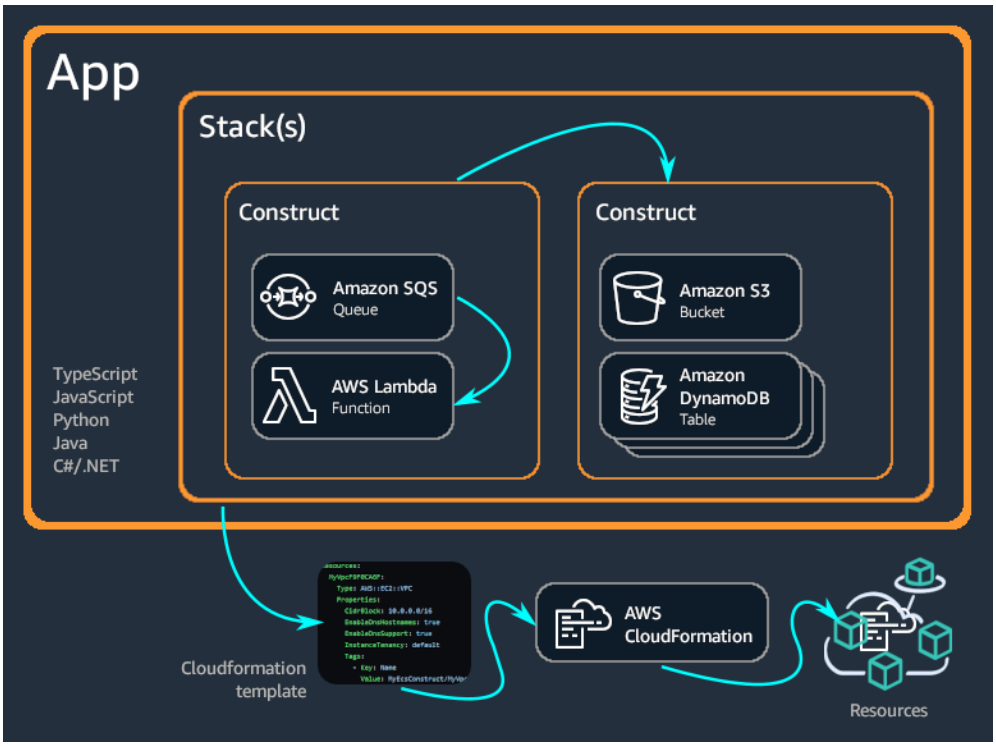



**Figura 3.7:** Struttura applicazione AWS CDK

Vado ora a descrivere velocemente le componenti illustrate nella figura precedente:

***Construct***: i costrutti sono gli elementi di base delle applicazioni AWS CDK. Un costrutto rappresenta un "componente *Cloud*" e incapsula tutto ciò di cui AWS CloudFormation ha bisogno per creare il componente. Un costrutto può rappresentare una singola risorsa AWS, oppure può essere un'astrazione di livello superiore costituita da più risorse correlate tra loro. Ci sono tre diversi livelli di costrutti:

* **Costrutti L1**: questi costrutti (di livello più basso) rappresentano direttamente tutte le risorse disponibili in AWS CloudFormation;

```
bucket = s3.CfnBucket(self, "MyBucket", bucket_name="MyBucket")
```

**Figura 3.8:** Esempio di costrutto L1

* **Costrutti L2**: rappresentano anch'essi risorse AWS, ma con un'API di livello superiore. I costrutti AWS offrono comode impostazioni predefinite e riducono la necessità di conoscere tutti i dettagli delle risorse AWS che rappresentano, fornendo metodi pratici che semplificano il lavoro con la risorsa;

```
s3.Bucket(self, "MyFirstBucket", versioned=True)
```

**Figura 3.9:** Esempio di costrutto L2

* **Costrutti L3**: questi costrutti vengono chiamati *pattern*. Sono progettati per aiutare l'utente (che nel nostro caso è lo sviluppatore) a completare operazioni comuni in AWS, che spesso coinvolgono più tipi di risorse.

```
class NotifyingBucket(Construct):

    def __init__(self, scope: Construct, id: str, *, prefix=None):
        super().__init__(scope, id)
        bucket = s3.Bucket(self, "bucket")
        topic = sns.Topic(self, "topic")
        bucket.add_object_created_notification(s3notify.SnsDestination(topic),
            s3.NotificationKeyFilter(prefix=prefix))
```

**Figura 3.10:** Esempio di costrutto L3

***Stack***: unità di distribuzione in AWS CDK. Tutte le risorse AWS definite all'interno di uno *stack*, direttamente o indirettamente, vengono fornite come un'unica unità.

```
class MyFirstStack(Stack):

    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        s3.Bucket(self, "MyFirstBucket")
```

**Figura 3.11:** Esempio di uno Stack

***App***: infine una volta dichiarato uno *stack*, è necessario istanziarlo da qualche parte per poter fare il *deploy*. Per fare ciò si utilizza il costrutto *App*. Il costrutto *App* non richiede alcun argomento di inizializzazione, perché è l'unico costrutto che può essere usato come radice dell'albero dei costrutti.
Così si può usare l'istanza di *App* per definire una singola istanza del proprio *stack*.

```
app = App()
MyFirstStack(app, "hello-cdk")
app.synth()
```

**Figura 3.12:** Esempio di una App

## 3.6 Studio delle risorse AWS coinvolte

Per tutta la durata dello *stage* ho passato la maggior parte del tempo a studiare nella documentazione di AWS le risorse utili all'implementazione del caso d'uso analizzato inizialmente.
Se visto in modo superficiale infatti, il caso d'uso preso in considerazione potrebbe sembrare di semplice implementazione.
In realtà lo sviluppo di un "semplice" *service* ECS agganciato ad un *Application Load Balancer* nasconde al suo interno tutta la complessità data dalla numerosità delle risorse richiamate, dalle loro relazioni e dalle loro dipendenze.

Per avere una visione dall'alto del caso d'uso implementato si veda la Figura 3.13.

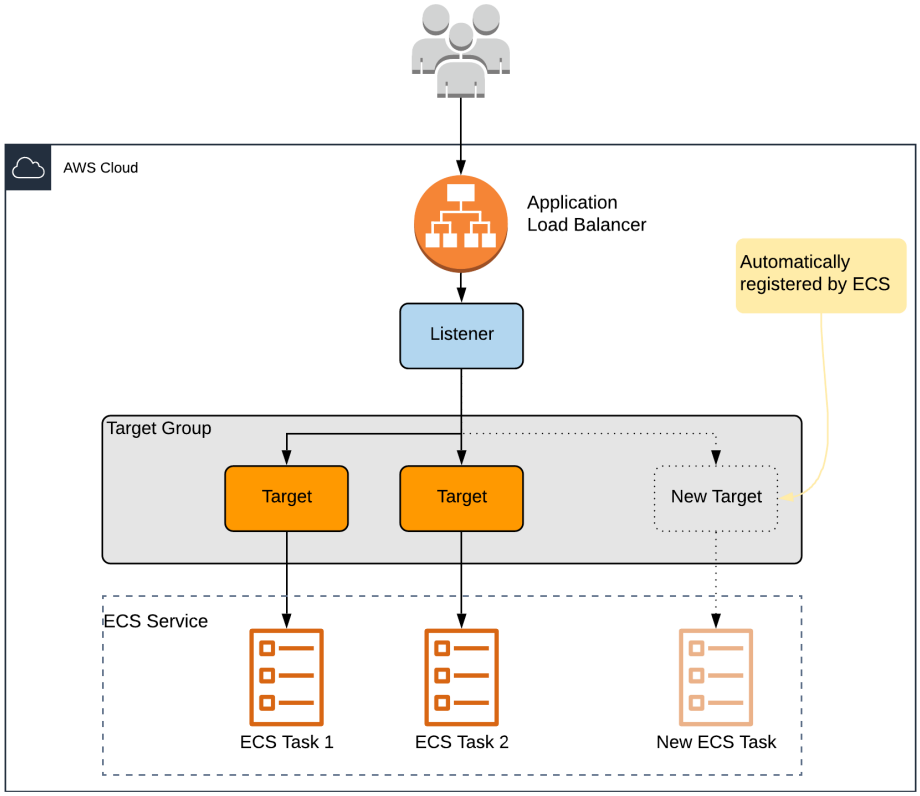



**Figura 3.13:** Caso d'uso: *service* ECS agganciato ad un ALB

Per poter capire il caso d'uso prima è necessario dare una definizione di alcune delle risorse coinvolte e degli elementi che le vanno a costituire.

**AWS ECS**: *Amazon Elastic Container Service*, è un servizio di gestione dei *container* altamente scalabile e veloce. Può essere utilizzato per eseguire, arrestare e gestire i *container* su un *cluster* (raggruppamento logico di processi o servizi). Con Amazon ECS, i *container* sono definiti attraverso una definizione di processi utilizzata per eseguire singoli processi o i processi all'interno di un *service*. In questo contesto, un *service* è una configurazione che può essere usata per eseguire e gestire simultaneamente un numero specificato di processi in un *cluster*. AWS ECS offre la possibiltà di decidere con che opzione eseguire i *container*:

* **AWS Fargate**: opzione *serverless* che permette di pagare solamente per l'effettivo utilizzo. Con questa soluzione non ti devi preoccupare di gestire l'infrastruttura poiché la gestisce AWS Fargate per te in automatico;
* **AWS EC2**: in alternativa, per un avere un maggiore controllo sull'infrastruttura, puoi eseguire i tuoi processi e i tuoi servizi su un *cluster* di istanze Amazon EC2 gestite da te.

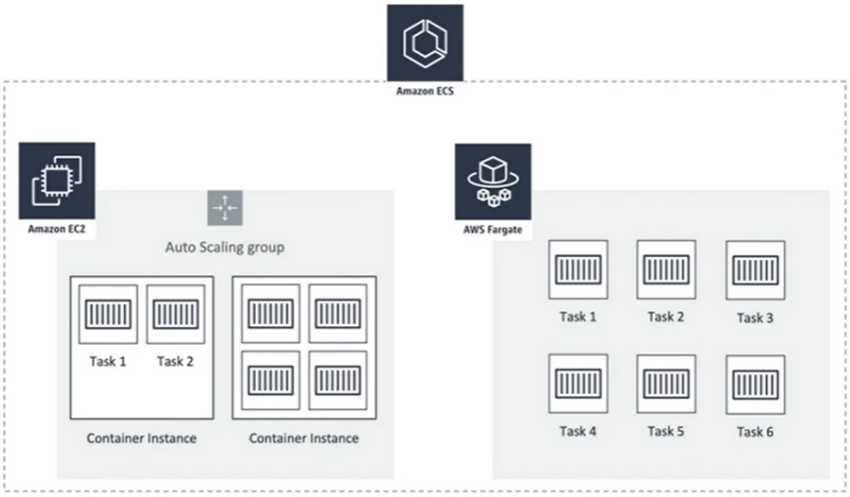



**Figura 3.14:** Tipologie di esecuzione per un *service* ECS

**AWS IAM**: *Amazon Identity and Access Management* è un servizio di gestione degli accessi che consente di controllare in modo sicuro l'accesso alle risorse AWS.
AWS IAM può essere utilizzato per controllare chi si è autenticato e chi è autorizzato a visualizzare o eseguire azioni specifiche sulle risorse.
In un service ECS, un AWS IAM può essere usato per controllare l'accesso a livello di istanza dei *container* tramite le IAM Role, ovvero possono essere assegnati dei ruoli specifici per alcuni utenti con permessi differenti.

**AWS Auto Scaling**: è un servizio che imposta il dimensionamento automatico per i tuoi processi. Il dimensionamento si basa su *policy* definite dall'utente e controlli dello stato di integrità. AWS Auto Scaling può essere utilizzato con un AWS EC2 per dimensionare le istanze di *container* all'interno del *cluster*.

**AWS ELB**: *Amazon Elastic Load Balancing* è un servizio che distribuisce automaticamente il traffico delle applicazioni in entrata tra i processi del *service* ECS.
Può essere usato per ottenere livelli più elevati di tolleranza ai guasti nelle applicazioni. Allo stesso tempo, può fornire la quantità necessaria di capacità di bilanciamento del carico per distribuire il traffico delle applicazioni.
Viene utilizzato per creare un *endpoint* che consenta di bilanciare il traffico tra servizi in un *cluster*.

**AWS ECR**: *Amazon Elastic Container Registry* è un registro di *container* Docker completamente gestito, che offre sicurezza, scalabilità e affidabilità. Amazon ECR supporta *repository* Docker privati con autorizzazioni basate su risorse tramite IAM, per consentire a utenti specifici di accedere a repository e immagini. Gli sviluppatori possono utilizzare la CLI di Docker per l'invio, l'estrazione e la gestione delle immagini.

## 3.7 Tecnologie e strumenti utilizzati

Un punto focale dello *stage* era riuscire ad apprendere velocemente tutte le tecnologie e gli strumenti necessari per lo sviluppo della libreria infrastrutturale.
In questo capitolo espongo dunque le varie tecnologie e i vari strumenti con cui ho familiarizzato durante il periodo di *stage*.

Prima di iniziare a descrivere le varie tecnologie, la Figura 3.15 presenta un quadro generale d'insieme, descrivendo tramite un diagramma architetturale le loro relazioni.

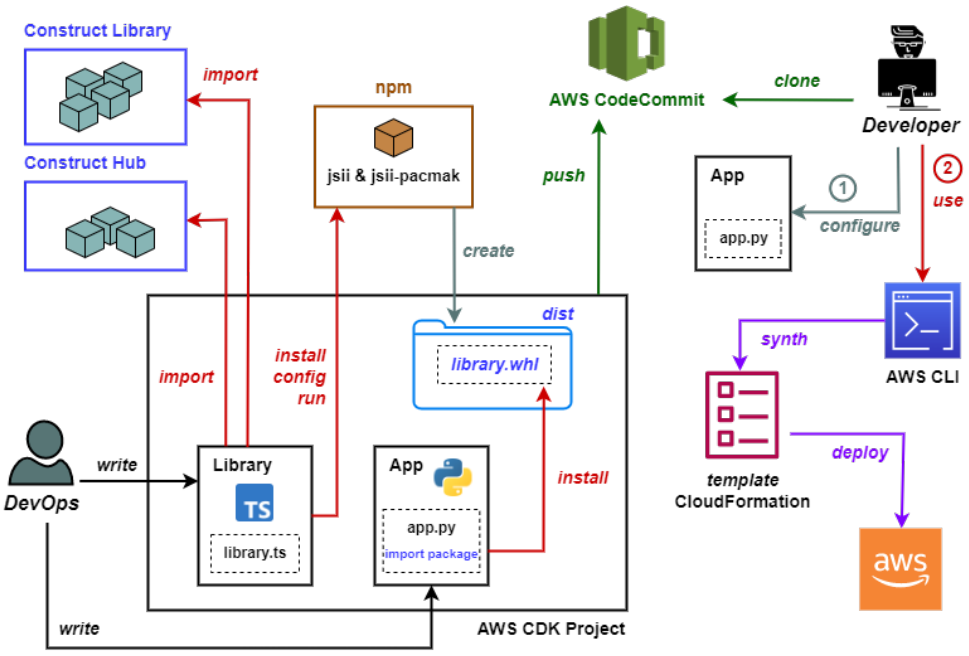



**Figura 3.15:** Diagramma architetturale delle tecnologie utilizzate

### 3.7.1 VSCode

Visual Studio Code[4] è un *editor* di codice sorgente leggero ma potente sviluppato da Microsoft. Si tratta di un *editor cross-platform* compatibile con Windows, Linux e macOS, che permette di evidenziare la sintassi di ciascun linguaggio di programmazione, integrare il supporto per il *debugging*, controllo *Git*, completamento automatico delle istruzioni, *snippet* e *refactoring* del codice. Il punto di forza di Visual Studio Code sono le estensioni grazie alle quali è possibile ampliare notevolmente le sue funzionalità. L'ho scelto come *editor* per lo sviluppo della libreria poiché già utilizzato durante il corso di studi all'Università.

### 3.7.2 Python

Python[5] è un linguaggio di programmazione ad alto livello, orientato agli oggetti e adatto per molteplici usi. Esso offre una grande flessibilità, semplicità e dinamicità che lo rendono adatto per sviluppare applicazioni distribuite, scripting, computazione numerica e *testing*. Queste caratteristiche, assieme all'enorme comunità di sviluppatori che Python ha attualmente sulle spalle, sono state determinanti nella scelta di tale linguaggio per implementare l'applicazione che simula la chiamata alla libreria sviluppata. In THRON viene ampliamente utilizzato dai *DevOps* per lo *scripting*, per questo motivo infatti il mio tutor me lo ha suggerito come linguaggio.

### 3.7.3 TypeScript

TypeScript[6] è un linguaggio di programmazione *open source* sviluppato da Microsoft. Estende il linguaggio di programmazione JavaScript fornendo il supporto della tipizzazione e delle interfacce durante la fase di codifica. Durante la fase di compilazione il linguaggio viene prima tradotto e poi compilato in *JavaScript*, permettendo così di avere un maggiore controllo e comprensione sulla codifica. Questo linguaggio l'ho utilizzato per lo sviluppo della libreria infrastrutturale. Le motivazioni che mi hanno spinto a scegliere questo linguaggio le descrivo più avanti, precisamente nella sezione 3.8.

### 3.7.4 AWS CloudFormation

AWS CloudFormation[7] è un servizio che aiuta a modellare e impostare le risorse AWS, in modo da poter dedicare meno tempo alla gestione di tali risorse e più tempo a concentrarsi sulle applicazioni eseguite in AWS. Lo sviluppatore crea un *template* che descrive tutte le risorse AWS desiderate e *CloudFormation* si occupa del *provisioning* e della configurazione di tali risorse. Questi *template* permettono la scrittura dell'infrastruttura di rete tramite file di testo in formato JSON o YAML. Non è quindi più necessario creare e configurare manualmente le risorse AWS tramite *console*; se ne occupa *CloudFormation*.

Attualmente in THRON la maggior parte dell'infrastruttura e delle risorse AWS utilizzate dai servizi implementati è scritta utilizzando template *CloudFormation*.

[4] *Visual Studio Code*. URL: https://code.visualstudio.com/.
[5] *Python*. URL: https://www.python.org/.
[6] *TypeScript*. URL: https://www.typescriptlang.org/.
[7] *AWS CloudFormation*. URL: https://docs.aws.amazon.com/cloudformation/index.html.

Durante lo *stage* ho analizzato molti di questi template per capire quali fossero le risorse AWS più comuni all'istanziazione di nuovi progetti aziendali.

### 3.7.5 AWS CDK

AWS Cloud Development Kit[8] (AWS CDK) è un *framework* di sviluppo *software open-source* che consente di definire l'infrastruttura *cloud* in codice e di effettuare il *provisioning* attraverso *AWS CloudFormation*. Offre infatti un'astrazione di alto livello, orientata agli oggetti, per definire le risorse AWS in modo imperativo utilizzando la potenza dei moderni linguaggi di programmazione. Attualmente il CDK supporta i seguenti linguaggi di programmazione: JavaScript, TypeScript, Python, Java, .NET e Go. Utilizzando la Construct Library[9] offerta da AWS CDK, è possibile incapsulare facilmente le *best practice* di AWS nella definizione dell'infrastruttura e condividerla. AWS CDK è la soluzione *IaC* da me selezionata per lo sviluppo della libreria infrastrutturale.

### 3.7.6 AWS CodeCommit

AWS CodeCommit[10] è un servizio di controllo di versione completamente gestito da AWS che ospita *repository* sicuri basati su *Git*. Consente ai team di sviluppo di collaborare facilmente al codice in un ecosistema sicuro e altamente scalabile. CodeCommit elimina la necessità di gestire un proprio sistema di controllo di versione o di preoccuparsi di scalare la sua infrastruttura. CodeCommit può essere utilizzato per archiviare in modo sicuro qualsiasi cosa, dal codice sorgente ai *file* binari, e funziona perfettamente con gli strumenti *Git* esistenti. In azienda questo servizio è utilizzato dal team di sviluppo per la condivisione di *repository*.

### 3.7.7 AWS CLI

AWS Command Line Interface[11] (AWS CLI) è uno strumento *open source* che consente di interagire con i servizi AWS utilizzando un'interfaccia a riga di comando. Con una configurazione minima, AWS CLI ti consente di iniziare a eseguire comandi che implementano funzionalità equivalenti a quelle fornite dalla *console* di gestione di AWS, dal *prompt* dei comandi nel tuo terminale. Questo strumento è essenziale per i *DevOps* all'interno di THRON perché velocizza molto qualsiasi azione utile alla configurazione, al *deploy*, al *testing* e alla gestione in generale delle risorse AWS.

La seguente tabella contiene i comandi principali digitabili da riga di comando con le rispettive funzioni associate:

| Comando | Descrizione |
|---|---|
| `cdk ls` | Elenca gli *stack* presenti nell'*app* |

*Continua nella pagina successiva*

[8] *AWS Cloud Development Kit.* URL: https://docs.aws.amazon.com/cdk/index.html.
[9] *AWS Construct Library.* URL: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-construct-library.html.
[10] *AWS CodeCommit.* URL: https://aws.amazon.com/it/codecommit/.
[11] *AWS Command Line Interface.* URL: https://docs.aws.amazon.com/cli/index.html.

| Comando | Descrizione |
|---|---|
| `cdk synth` | Sintetizza il codice in un *template* CloudFormation e lo stampa nel terminale |
| `cdk deploy` | Esegue il *deploy* dello/degli *stack* |
| `cdk destroy` | Esegue la distruzione dello/degli *stack* |
| `cdk diff` | Confronta lo *stack* attuale con quello caricato prima |
| `cdk init` | Crea un nuovo progetto CDK nella cartella attuale |
| `cdk doc` | Apre nel *browser* la CDK API *reference* |
| `cdk doctor` | Controlla il progetto CDK per individuare possibili problemi |

**Tabella 3.5:** Comandi principali dell'AWS CDK Toolkit

### 3.7.8 AWS Construct Library

AWS Construct Library è un ampio insieme di moduli, sviluppati da AWS, che espongono le API utili alla definizione di risorse AWS per le applicazioni sviluppate tramite AWS CDK. Durante lo *stage* l'ho utilizzata come fonte di documentazione ed è servita a scrivere svariati costrutti per la libreria infrastrutturale implementata.

### 3.7.9 Construct Hub

Construct Hub[12] è una piattaforma per trovare, riutilizzare e condividere costrutti creati non solo da AWS ma anche dai *partner* AWS, da terze parti e dalla *community* di sviluppatori. Anche questa risorsa l'ho utilizzata durante lo *stage* per documentarmi e prendere spunto dalla *community DevOps* per la scrittura dei costrutti.

### 3.7.10 Jsii

jsii[13] è un *framework open source* sviluppato da AWS che consente di riutilizzare le classi JavaScript (create in TypeScript) da una serie di altri linguaggi di programmazione (ad esempio, .NET, Java, Python). Il compilatore jsii è un compilatore TypeScript modificato, che estrae le API delle librerie di classi dai moduli compilati e produce una rappresentazione delle API indipendente dal linguaggio. È la tecnologia che consente all'AWS CDK di fornire librerie poliglotte da un'unica *codebase*. Ho scelto di utilizzare questo *framework* in autonomia poiché ho pensato potesse risultare utile creare un ulteriore livello di astrazione permettendo all'azienda di slegarsi dall'utilizzo di un singolo linguaggio di programmazione per la creazione di applicazioni tramite AWS CDK.

[12] *Construct Hub*. URL: https://constructs.dev/.
[13] *jsii*. URL: https://aws.github.io/jsii/.

### 3.7.11 Jsii-pacmak

jsii-pacmak[14] fa parte del progetto jsii ed è un *framework* che genera pacchetti pronti per la pubblicazione per i moduli jsii.
Esso si occupa di generare i *binding* delle librerie nei linguaggi di programmazione supportati. Prende quindi in ingresso la traduzione effettuata da jsii e crea dei pacchetti nei linguaggi specificati. Durante lo *stage* è servito per la traduzione della libreria sviluppata da TypeScript a Python.

## 3.8 Codifica della libreria infrastrutturale

Inizialmente la libreria infrastrutturale la stavo scrivendo in Python poiché è il linguaggio più utilizzato dai *DevOps* e quello consigliatomi dal mio *tutor*.
Dopo aver analizzato però le possibili soluzioni, sono arrivato alla conclusione che sarebbe stato più opportuno svilupparla in TypeScript poiché tramite l'utilizzo di `jsii` viene consentito al codice scritto in qualsiasi linguaggio di interagire naturalmente con le classi JavaScript.

Ho quindi proposto questa soluzione ad una delle riunioni di *competence* dei *DevOps*, ricevendo *feedback* positivi e passando quindi ad utilizzare TypeScript come linguaggio di programmazione per la scrittura della libreria.

`jsii` è infatti la stessa tecnologia che permette as AWS CDK di fornire librerie poliglotte da un'unica *codebase*; consente quindi ad una libreria scritta in TypeScript di essere utilizzata in progetti realizzati utilizzando diversi linguaggi (oltre a TypeScript e JavaScript anche in Python, Java, C#).
Oltre a questo strumento ho utilizzato anche `jsii-pacmak` che consente la generazione di pacchetti pronti alla pubblicazione per i moduli `jsii`.
Grazie all'ausilio di questi strumenti *open source*, si ha quindi la possibilità di pubblicare la libreria pacchettizzata in diversi linguaggi all'interno della propria organizzazione partendo dalla stessa *codebase*.

Per dare un'idea della semplicità di utilizzo degli strumenti appena menzionati ne mostro il funzionamento qui di seguito con qualche esempio:

Per installare `jsii` e `jsii-pacmak` basterà digitare il seguente comando sul terminale: (Prerequisito: aver installato Node.js ed npm)

```
npm install --save-dev jsii jsii-pacmak
```

Una volta fatto ciò basterà andare nel proprio progetto CDK, aprire il file `package.json` e aggiungere le seguenti righe:

```
"scripts": {
    "build": "jsii",
    "watch": "jsii -w",
    "package": "jsii-pacmak",
    "test": "jest"
}
```

Successivamente, sempre nello stesso file `package.json` aggiungere anche:

[14] *jsii-pacmak*. URL: https://github.com/aws/jsii/tree/main/packages/jsii-pacmak.

```
"jsii": {
    "outdir": "dist",
    "target": {
      "python": {
        "distName": "mycorp.mypackage",
        "module": "mycorp.mypackage",
      }
    }
}
```

Una volta fatto ciò basterà tornare sul terminale ed eseguire i seguenti comandi:

```
npm run build && npm run package
```

Questo permetterà di creare una cartella "*dist*" con all'interno tutti i *package* tradotti nei linguaggi di destinazione specificati.

Per installare la libreria lo sviluppatore dovrà semplicemente eseguire un comando simile a questo, sostituendo il *path*:

```
pip install ../project-folder/dist/python/mycorp.mypackage-0.1.0.whl
```

Per testare poi il corretto funzionamento della libreria ho poi creato una applicazione CDK scritta in Python.
L'applicazione CDK semplicemente andrà ad importare il *package* creato precedentemente e tramite la chiamata al costruttore della libreria consentirà la parametrizzazione da parte dello sviluppatore.

Aprire quindi l'applicazione CDK (nel nostro caso sviluppata in Python) e importare il *package*:

```
from mycorp import mypackage
```

Per poi andare ad aggiungere ed istanziare il costruttore allo *stack*:

```
myConstruct: mypackage.NameStack = mypackage.NameStack(
    self,
    "myConstruct",
    props: {}
)
```

Per la parametrizzazione del costruttore si è pensato di richiedere allo sviluppatore di inserire solamente i campi obbligatori per la creazione del servizio, assegnando dei valori di *default* agli attributi delle risorse automaticamente generate.
Oltre a questo si è pensato di lasciare libero lo sviluppatore di inserire manualmente anche configurazioni più specifiche.

La Figura 3.16 e la Figura 3.17 rappresentano rispettivamente degli *snippet* d'esempio della libreria infrastrutturale e dell'applicazione che la richiama.

```
const targetGroup = new elbv2.ApplicationTargetGroup(this, 'TargetGroupDelivery', {
  targetType: elbv2.TargetType.IP,
  protocol: props.protocol ?? elbv2.ApplicationProtocol.HTTP,
  port: props.targetGroupPort ?? 8080,
  slowStart: cdk.Duration.seconds(props.slowStart ?? 30),
  deregistrationDelay: cdk.Duration.seconds(props.deregistrationDelay ?? 60),
  vpc: vpc,
  healthCheck: {
    path: props.healthCheckPath ?? '/health',
    interval: cdk.Duration.seconds(props.healthCheckInterval ?? 10),
    timeout: cdk.Duration.seconds(props.healthCheckTimeout ?? 3),
    healthyThresholdCount: props.healthyThresholdCount ?? 2,
    unhealthyThresholdCount: props.unhealthyThresholdCount ?? 4,
    healthyHttpCodes: props.healthyHttpCodes ?? '204',
  },
});
cdk.Tags.of(targetGroup).add('Name', props.siteName+'-'+props.templateName+'delivery');

const fargateTaskDefinition = new ecs.FargateTaskDefinition(this, 'FargateTaskDef', {
  memoryLimitMiB: props.ram ?? 512,
  cpu: props.cpu ?? 256,
});
fargateTaskDefinition.node.addDependency(taskRoleDelivery, logGroup);

fargateTaskDefinition.applyRemovalPolicy(RemovalPolicy.DESTROY);

const container = fargateTaskDefinition.addContainer("WebContainer", {
  image: ecs.ContainerImage.fromRegistry(props.imageName ?? 'amazon/amazon-ecs-sample'),
  logging: ecs.LogDriver.awsLogs({
    streamPrefix: props.templateName+'-delivery',
    logGroup: logGroup,
  }),
  environment: props.envVar,
});
```

**Figura 3.16:** *Snippet* di esempio - Libreria infrastrutturale

```
myECS: ecs_http.EcsDelivery = ecs_http.EcsDelivery(
# REQUIRED PARAMETERS
#-----------------------------------------------
    self,
    # Name of the project
    "MyEcsDelivery",
    # Vpc
    vpc=self.__get_vpc(),
    # Cluster
    cluster=self.__get_ecs_cluster(),
    # Name of the template
    template_name="myTemplate",
    # Name of the role
    role="myRole",
    # Name of the site
    site_name=self.__get_current_sitename(),
    # Type of the environment
    environment=self.__get_current_environment(),
    # Launch type (EC2 or FARGATE)
    launch_type=ecs_http.LaunchType.EC2,
    # Listener
    add_app_listener=self.__get_listener(),
    # The number of cpu units used by the task
    cpu=256,
    # The amount (in MiB) of memory used by the task
    ram=512,
    # The port number on the container
    container_port=8080,
    # The image used to start a container
    image_name='image/name',
    # The maximum number of tasks in %  that can run in a service during a deployment
    max_healthy_p=200,
    # The minimum number of tasks in % that must continue to run and remain healthy during a deployment
    min_healthy_p=50,

# OPTIONAL PARAMETERS
#-----------------------------------------------
```

**Figura 3.17:** *Snippet* di esempio - Applicazione CDK che richiama la libreria

### 3.8.1 Evoluzioni future

Per quanto riguarda invece l'effettivo utilizzo del costrutto da parte dello sviluppatore si è solamente pensata una possibile soluzione.
Quando si andrà a creare un nuovo progetto su *genesis* (generatore automatico di progetti aziendali) si specificherà che si tratta di un progetto CDK, il quale provvederà a fornire in automatico tutti i *file* di configurazione e l'applicazione qui definita.
Lo sviluppatore quindi dovrà solamente fare l'`import` della libreria e configurare il costruttore con la parametrizzazione desiderata. Non sarà inoltre necessario per lo sviluppatore installarsi la CDK poiché sarà integrata con la *pipeline* aziendale.

## 3.9 Copertura dei requisiti

Durante il mio percorso di *stage*, sono emerse alcune problematiche che hanno rallentato il normale flusso di lavoro.
Prima tra tutte l'utilizzo di AWS CDK, perché pur permettendo un elevato livello di astrazione ha comunque bisogno della conoscenza del funzionamento delle risorse alla base dei costrutti. Per questo ho dovuto dedicare molto del mio tempo a disposizione per lo studio delle risorse e le loro dipendenze.
Altro problema riscontrata è il complicato sviluppo della libreria per due ambienti completamente diversi (quello di *testing* messo a disposizione dell'azienda e quello di sviluppo aziendale).

Purtroppo, non ho avuto tempo di implementare i *test* di unità e di integrazione, ma ho comunque provveduto con il mio *tutor* a verificare il funzionamento della libreria tramite la *console* di AWS.
Per quanto riguarda l'integrazione della libreria nella *pipeline* aziendale, non ho avuto modo di farlo poiché l'azienda non ha ancora finito di implementarla.

Di seguito riporto la tabella della copertura dei requisiti:

| Tipologia | Numero requisiti | Soddisfatti |
|---|---|---|
| **Funzionale** | 6 | 5 |
| **Prestazionale** | 0 | 0 |
| **Qualitativo** | 3 | 2 |
| **Di vincolo** | 8 | 7 |
| **Totale** | 17 | 14 |

**Tabella 3.6:** Copertura dei requisiti

In conclusione, il prodotto che sono riuscito a realizzare è un *pattern* infrastrutturale che permette in modo semplice ed intuitivo il suo riutilizzo all'interno della propria organizzazione.
Con questo gli sviluppatori di *competence* diverse da quella *DevOps* riusciranno a configurare in modo autonomo l'infrastruttura necessaria per l'erogazione dei servizi da loro sviluppati, senza dover richiedere il supporto da parte di un *DevOps*.

# Capitolo 4

# Valutazione retrospettiva

## 4.1 Obiettivi raggiunti

Come illustrato precedentemente nella sezione 2.3, prima dell'inizio dello stage, ho redatto un piano di lavoro che descriveva le attività che avrei svolto. Ho svolto quest'operazione in collaborazione con il tutor aziendale.
Gli obiettivi inizialmente preventivati si concentravano sull'analisi delle possibili soluzioni *IaC*, sullo sviluppo di una libreria infrastrutturale per la gestione delle risorse *Cloud* necessarie alla creazione di un nuovo progetto aziendale, sulla successiva integrazione della libreria nella *pipeline* aziendale ed infine sull'aggiunta di qualche funzionalità utile come il miglioramento dei sistemi di monitoraggio aziendale oppure all'implementazione di un meccanismo di *rollback*.

Degli obiettivi aziendali appena descritti sono riuscito a soddisfare solamente quelli obbligatori poiché il tempo a mia disposizione era molto limitato per la complessità del progetto proposto.
Il mio tutor ha comunque espresso pareri positivi sul prodotto sviluppato poiché si è reso conto, in corso d'opera, che le attività inizialmente preventivate risultavano essere troppo ottimistiche. Il prodotto da me sviluppato l'ho poi testato insieme al tutor in un ambiente AWS di "prova" messo a disposizione dall'azienda. Per il passaggio nell'ambiente di sviluppo aziendale mancherebbero alcune integrazioni date dalla diversità degli ambienti.

Dal punto di vista funzionale la libreria sviluppata e l'applicazione che la richiama tramite la chiamata al costruttore, permettono la parametrizzazzione da parte dello sviluppatore che può scegliere in autonomia quali informazioni inserire manualmente e per quali invece utilizzare la costruzione di *default*. Così facendo uno sviluppatore esterno alla *competence DevOps* riuscirà in autonomia a creare l'infrastruttura necessaria a creare le risorse AWS utili al *deploy* del proprio servizio sviluppato.

Per quanto riguarda l'integrazione della libreria nella *pipeline* aziendale non ho potuto soddisfare alcun obiettivo poiché la *pipeline* risultava essere ancora incompleta.
Nel complesso il tutor aziendale e le figure con la quale ho comunicato all'interno dell'azienda, sono state soddisfatte del lavoro svolto.

Personalmente invece mi è dispiaciuto non riuscire a soddisfare tutti gli obiettivi aziendali poiché verso la fine dello *stage* ho notato come il progetto propostomi servisse

effettivamente a soddifare un bisogno aziendale concreto che nel breve periodo avrebbe avuto bisogno di una soluzione funzionante. La Figura 4.1 visualizza gli obiettivi principali raggiunti e quelli che invece mancherebbero da sviluppare.


**Figura 4.1:** Copertura attuale degli obiettivi inizialmente preventivati

## 4.2 Competenze acquisite

L'esperienza di *stage* è stata un'ottima opportunità di confermare le competenze acquisite durante il mio corso di studi e di apprendere cose nuove. Oltre all'acquisizione di competenze tecniche, mi ha permesso di imparare cosa significhi lavorare a contatto con altre persone in un contesto aziendale.

Le conoscenze infatti che posso dire di aver aggiunto o ampliato rispetto al bagaglio formativo in mio possesso prima dell'inizio dello *stage* sono diverse.

Sicuramente ho acquisito la capacità di lavorare in gruppo. Spesso venivo coinvolto in discussioni riguardanti il mio progetto di *stage* e ciò mi ha permesso di migliorare le mie competenze sotto il profilo della comunicazione.
Tale esperienza mi ha fatto capire inoltre quanto importante siano i pareri di altre persone, in quanto permettono di analizzare il problema sotto altri punti di vista e di portare alla luce eventuali casi limite inizialmente non presi in considerazione.

Riguardo le competenze tecniche acquisite, durante lo *stage* ho potuto apprendere nuovi linguaggi di programmazione come Python e TypeScript.
Ho imparato come eseguire uno *scouting* completo che comprendesse tutti dettagli utili ma che alla fine risultasse di facile comprensione per poter far arrivare i vantaggi o svantaggi delle soluzioni analizzate anche a persone estranee alle tematiche affrontate.
Ho aumentato le mie conoscenze del sistema operativo Ubuntu e ho ottenuto maggiore familiarità con la scrittura da linea di comando.

In particolare però, sono rimasto colpito dalle infinite opportunità che offre Amazon AWS.
Durante lo *stage* infatti ho familiarizzato con numerosi servizi offerti da AWS. In particolare ho avuto la possibilità sia di configurare i vari servizi dalla *console* di AWS, così da riceve *tutorial* esplicativi del servizio analizzato, ma soprattuto la possibilità di configurare tali servizi tramite soluzioni di *IaC*.
In particolare ho potuto rendermi conto delle effettive potenzialità di una piattaforma

*serverless* rispetto ad una che utilizza *server* tradizionali.

Altra conoscenza acquisita di cui prima ignoravo totalmente l'esistenza è il ruolo ricoperto dalla figura del *DevOps*.
Durante il periodo di *stage* infatti ho potuto osservare quali mansioni ricopre questa figura all'interno di un'azienda, a quali problematiche deve saper rispondere con prontezza e quanto sia fondamentale all'interno di un *team* di sviluppo di notevoli dimensioni.

Lo *stage* svolto infatti mi ha permesso di poter valutare il ruolo ricoperto da questa figura professionale. Questo mi ha aiutato a far chiarezza sul percorso da seguire dopo la laurea.
Anche se ad un primo approccio sono rimasto spaventato dalla quantità di conoscenze richieste ad un *DevOps*, con il tempo mi sono sempre più incuriosito da questa figura professionale poiché ricopre veramente molte competenze e riesce ad avere un quadro generale d'insieme.
Per questi motivi attualmente sto valutando la possibilità di sceglierla come occupazione nell'immediato futuro.

## 4.3 Distanza tra mondo universitario e lavorativo

L'università indubbiamente ha il difficile compito di formare persone che, una volta terminato il percorso di studi, possano entrare facilmente nel mondo del lavoro. Compito reso ancora più difficile dall'ampia gamma di strade differenti che l'indirizzo di informatica offre e dalla continua evoluzione del settore.

In generale, la formazione data dal corso di studi in informatica permette di preparare a dovere uno studente che si interfaccia per la prima volta al mondo del lavoro.
Le nozioni di base trattate durante gli anni accademici infatti permettono agli studenti di avere una base solida da cui partire. Inoltre, l'università è utile per creare rapporti e relazionarsi con persone che condividono la tua stessa passione, favorendo il confronto costruttivo e la collaborazione.
A mio parere, uno studente che esce dall'università, ha acquisito un'ottima capacità di adattamento rispetto allo studio di nuove tecnologie e di nuovi strumenti di lavoro e riesce a svolgere in autonomia il lavoro assegnatogli. Oltre a ciò è preparato per affrontare l'insorgere di situazioni impreviste.

Lo scopo della preparazione offerta dal corso di informatica da me affrontato, è fornire una base di conoscenze teoriche, che poi possano essere utilizzate in molti contesti, ambiti e con strumenti anche molto differenti tra loro.
Le nozioni base alla quale mi riferisco sono fornite principalmente dai corsi di Programmazione ad oggetti, Basi di dati, Tecnologie web ed Ingegneria del software.
Questi corsi, sono stati utili soprattutto per la presenza di progetti ad essi correlati. Infatti, se durante il corso svolto in aula, viene affrontata solamente la parte teorica della disciplina, i progetti ne completano l'apprendimento con un esercizio pratico.
In particolar modo, il progetto di Ingegneria del software si è rivelato fondamentale per la quantità di tecnologie e discipline affrontate, per l'utilizzo di una metodologia di lavoro e per la collaborazione all'interno di un gruppo numeroso di persone.

Quanto riportato fin'ora descrive il caso generico e la mia personale visione dell'indiscussa utilità di uno studio universitario.

Nel mio caso specifico però, la preparazione universitaria non è risultata sufficiente per affrontare tematiche tanto nuove quanto complesse.
A mio parere nel corso di studi manca una preparazione più specifica riguardante l'architettura di rete, soprattutto per quanto riguarda le tecnologie *Cloud*. D'altra parte però capisco anche che in un percorso di studi con crediti formativi fissati sia difficile andare a toccare qualsiasi aspetto dell'informatica.

Questo mi porta a pensare che forse il progetto di *stage* inizialmente proposto da THRON fosse volutamente ambizioso per poter suscitare l'interesse degli studenti.
Le tematiche affrontate durante lo *stage* infatti si sono rivelate molto attuali e mi hanno permesso di approfondire, tramite esempi pratici, alcune nozioni teoriche fornite dal corso universitario di Reti.
Tutte queste nozioni però inverosimilmente avrebbero richiesto qualche giorno per il loro apprendimento. Solamente dopo aver effettuato la nuova pianificazione delle attività infatti si è dato il giusto spazio allo studio delle tematiche affrontate.

Concludo questa riflessione ammettendo che le motivazioni principali che mi hanno portato a selezionare questo progetto erano appunto la sua complessità e la mia voglia di mettermi in gioco.
Solo a posteriori posso affermare che le tematiche DevOps siano molto vaste e richiedano una conoscenza dettagliata di molti aspetti legati all'informatica.

In generale però, ritengo che questa esperienza di stage sia stata altamente formativa non solo per quanto riguarda le nozioni apprese, ma soprattutto per una crescita personale. La disponibilità del tutor aziendale e di tutti i *team* con cui ho avuto occasione di relazionarmi, mi ha permesso di avere una visione più ampia del lavoro affidatomi e delle dinamiche aziendali.

# Glossario

**API** Application Programming Interface, in informatica indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 36

**AWS** Amazon Web Services, è un'azienda statunitense di proprietà del gruppo Amazon, che fornisce servizi di *Cloud computing* su un'omonima piattaforma *on demand*. 6

**AWS CloudFormation** è un servizio che aiuta a modellare e configurare le risorse AWS così da poter dedicare meno tempo alla gestione di tali risorse e concentrarsi invece sulle applicazioni eseguite in AWS. 10

**CI/CD** è un approccio per lo sviluppo di software, focalizzato sull'automazione delle procedure che portano il codice dallo sviluppo all'integrazione, dal *test* alla distribuzione e deployment finale. Infatti CI fa riferimento alla metodologia di Integrazione Continua, mentre CD a quella di Distribuzione Continua e/o Deployment Continuo, che pur indicando livelli di automazioni diversi vengono spesso usati in modo intercambiabile. 13

**CLI** Command Line Interface, è un'interfaccia a riga di comando, ovvero un'interfaccia basata su comandi testuali per eseguire azioni su un sistema informatico. 6

**Cloud computing** indica, in informatica, un paradigma di erogazione di servizi offerti su richiesta da un fornitore a un cliente finale attraverso la rete internet, a partire da un insieme di risorse preesistenti, configurabili e disponibili in remoto sotto forma di architettura distribuita. 10

**Cloud provider** è una società di terze parti che fornisce servizi di archiviazione, applicazioni, infrastruttura o piattaforma basati sul *Cloud*. 10

**CodeCommit** servizio di controllo delle versioni ospitato da *Amazon Web Services*, utile per archiviare e gestire privatamente asset quali documenti, codice sorgente e file binari nel *Cloud*. Questo servizio ospita *repository* Git privati. 6

**Confluence** è una piattaforma collaborativa sviluppata da Atlassian. 6

**DAM** Digital Asset Management, categoria per prodotti software definita da Forrester, il DAM è uno strumento che permette alle società che investono nei contenuti di fare efficienza nei processi e accorciare il time-to-market. 1, 7

**DevOps** figura professionale che fa da collante tra il team di sviluppo, che si occupa di introdurre nuove funzionalità ad un'applicazione, ed il team di infrastruttura che si occupa di preservare la stabilità di un'applicazione una volta rilasciata. Negli ultimi anni questa figura ha acquisito sempre più rilevanza poichè introduce processi, strumenti e metodologie per bilanciare le esigenze durante l'intero ciclo di vita dello sviluppo del software, dalla codifica alla distribuzione, fino alla manutenzione e agli aggiornamenti. 3, 4

**Docker** è un insieme di prodotti *Platform as a service* che utilizzano la virtualizzazione a livello di sistema operativo per fornire *software* in pacchetti chiamati *container*. 33

**Git** software per il controllo di versione distribuito utilizzabile da interfaccia a riga di comando, creato con lo scopo di essere un semplice strumento per facilitare lo sviluppo del kernel Linux. 6

**IaC** *Infrastructure as Code*, è un paradigma IT che definisce in linguaggio informatico non soltanto il software stesso, ma anche l'infrastruttura necessaria per il suo funzionamento, come lo spazio di archiviazione, la potenza di calcolo o le risorse di rete. Prevede quindi la descrizione di un hardware in un codice eseguibile, che può essere facilmente adattato, duplicato, cancellato e "versionato" in qualsiasi momento. 12

**JavaScript** è un linguaggio di programmazione multi paradigma orientato agli eventi, comunemente utilizzato nella programmazione Web lato client (esteso poi anche al lato server) per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso. 34

**Jira** è una suite di software proprietari per il tracciamento delle issue sviluppato da Atlassian, che consente il bug tracking e la gestione dei progetti sviluppati con metodologie agili. 6

**Machine Learning** è una branca dell'intelligenza artificiale che utilizza tecniche statistiche per dare ad un sistema informatico l'abilità di imparare, ovvero migliorare le proprie performance su task specifici, attraverso dati. 3

**PIM** Product Information Management, è l'espansione del prodotto Thron per gestire ed arricchire le schede prodotto del catalogo. Permette di organizzare le schede prodotto da un unico lato, approvarle in modo veloce e automatico ed arricchirle con qualsiasi contenuto presente in Thron. 3

**PO** Product Owner, è la persona responsabile di massimizzare il valore del prodotto e del lavoro scolto dal team di sviluppo. E' colui che decide cosa deve essere fatto, quali funzionalità deve avere un prodotto o servizio e quando rilasciarle sul mercato. Il Product Owner rappresenta la voce del cliente in azienda. 4

**POM** Product Operations Manager, è il responsabile dell'insieme dei processi aziendali tramite cui un'impresa realizza e consegna un prodotto/servizio al cliente. 4

**Project Management** Insieme di attività svolte dal project manager volte all'analisi, progettazione, pianificazione e realizzazione degli obiettivi di un progetto. 5

**rollback** è un'operazione che consente il ritorno ad uno stato precedente di versione, lasciando in questo modo l'infrastruttura sempre disponibile. 13

**RTIE** Real-Time Image Editor, *image server* che permetta di creare in tempo reale le thumbnail richieste, evitando l'elaborazione in fase di caricamento del contenuto. L'RTIE restituisce precisamente l'immagine alla giusta risoluzione, permette di eliminare il carico di lavoro richiesto ad adattare le thumbnails create in precedenza all'effettiva risoluzione finale. Consente inoltre, tramite un sistema automatico, di tagliare e ridimensionare le foto in modo intelligente, tenendo conto del soggetto, eliminando il contorno superfluo. 9

**SaaS** Software as a Service, è un modello di distribuzione del software applicativo dove un produttore di software sviluppa, opera e gestisce un'applicazione che mette a disposizione dei propri clienti via Internet previo abbonamento. 1

**Scrum** Framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo. 4

**Solution Architect** colui che progetta o modifica un'architettura di sistema con lo scopo di migliorare la funzionalità di un certo business. I Solution Architects testano, integrano e programmano sistemi software per assicurarsi che specifici problemi siano risolti. 4

**Sprint** periodo di tempo prestabilito durante il quale un lavoro specifico deve essere completato e reso pronto per la revisione. 5

**StageIT** evento promosso da Assindustria Venetocentro in collaborazione con l'Università di Padova per favorire l'incontro tra aziende con progetti innovativi in ambito IT e studenti dei corsi di laurea in Informatica, Ingegneria informatica e Statistica. 9

**UML** *Unified Modeling Language*, è un linguaggio che permette, tramite l'utilizzo di modelli visuali, di analizzare, descrivere, specificare e documentare un sistema software anche complesso. 21

# Bibliografia

## Siti web consultati

*AWS Cloud Development Kit*. URL: https://docs.aws.amazon.com/cdk/index.html (cit. a p. 35).

*AWS CloudFormation*. URL: https://docs.aws.amazon.com/cloudformation/index.html (cit. a p. 34).

*AWS CodeCommit*. URL: https://aws.amazon.com/it/codecommit/ (cit. a p. 35).

*AWS Command Line Interface*. URL: https://docs.aws.amazon.com/cli/index.html (cit. a p. 35).

*AWS Construct Library*. URL: https://docs.aws.amazon.com/cdk/api/v2/docs/aws-construct-library.html (cit. a p. 35).

*Cloud Development Kit for Terraform (CDKTF)*. URL: https://www.terraform.io/cdktf (cit. a p. 25).

*Construct Hub*. URL: https://constructs.dev/ (cit. a p. 36).

*jsii*. URL: https://aws.github.io/jsii/ (cit. a p. 36).

*jsii-pacmak*. URL: https://github.com/aws/jsii/tree/main/packages/jsii-pacmak (cit. a p. 37).

*Pulumi*. URL: https://www.pulumi.com/ (cit. a p. 26).

*Python*. URL: https://www.python.org/ (cit. a p. 34).

*Terraform*. URL: https://www.terraform.io/ (cit. a p. 25).

*TypeScript*. URL: https://www.typescriptlang.org/ (cit. a p. 34).

*Visual Studio Code*. URL: https://code.visualstudio.com/ (cit. a p. 34).