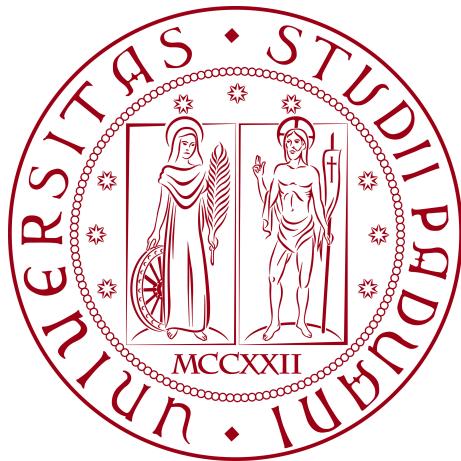


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Studio di Operatori Kubernetes per la gestione
di applicazioni persistenti.**

Tesi di Laurea Triennale

Relatore

Prof. Vardanega Tullio

Laureando

Rosson Lorenzo

Matricola 2042349

© Rosson Lorenzo, Corso di Laurea in Informatica, Agosto 2024. Tesi di Laurea Triennale: “*Studio di Operatori Kubernetes per la gestione di applicazioni persistenti.*”, Università degli Studi di Padova, Dipartimento di Matematica “Tullio Levi-Civita”, Corso di Laurea in Informatica. Tutti i diritti riservati.

Sommario

Il presente documento descrive ed esplora il lavoro svolto durante il periodo di *stage*, della durata di trecentoventi ore, dal laureando Lorenzo Rosson presso l’azienda Zucchetti Spa. Lo scopo del tirocinio è quello di apprendere ciò che comprende l’uso di *Kubernetes* nella gestione di applicazioni persistenti, “*Cloud Native*”, tramite Operatori realizzati appositamente, in modo da poterne valutare potenzialità e limitazioni in ottica di produzione. Per iniziare mi sono dedicato all’acquisizione delle competenze tecniche richieste per poter riuscire nello sviluppo del progetto, attraverso lo studio di documentazione, la lettura approfondita di articoli inerenti e la visione di video esempio. Inizialmente relative all’architettura e funzionamento di *Kubernetes*, in secondo luogo, relative alla sviluppo di operatori e quindi necessariamente, al linguaggio di programmazione individuato per tale attività, *Go (Golang)*. Successivamente all’apprendimento delle conoscenze richieste, ho innanzitutto creato un ambiente basilare con *Kubernetes* testandone le funzionalità native al fine di consolidare l’apprendimento teorico con l’esperienza pratica. Sono poi passato alla progettazione dell’operatore andando a valutare e confrontare principalmente varie politiche di natura gestionale con il fine di migliorare le prestazioni dell’applicazione che controlla. Infine ho proceduto con lo sviluppo vero e proprio dell’operatore andando a coprire le funzionalità previste, soffermandomi con maggiore attenzione su quelle relative alla componente adibita alla persistenza. A supporto, da buona prassi, ho svolto in modo incrementale, un’attività di documentazione di tutti i periodi e relativi prodotti del progetto, comprendente progettazione, codifica e *testing*.

Il documento è suddiviso nei seguenti 4 capitoli:

- **L’azienda:** capitolo che introduce e presenta il contesto ed ambiente aziendale nel quale ho avuto l’opportunità di svolgere il tirocinio. Comprendente un approfondimento sull’azienda, suoi prodotti e clienti, processi interni e di sviluppo, e propensione all’innovazione;
- **Lo stage:** capitolo che illustra lo *stage*, il suo scopo e il suo ruolo nel contesto aziendale. Comprendente elenco e descrizione di vincoli e obiettivi del progetto e rapporto dell’azienda con le attività di *stage*;
- **Il progetto:** capitolo che descrive quanto svolto durante il tirocinio per il progetto di *stage* suddividendo le attività affrontate:
 - **Analisi dei Requisiti:** descrizione di quanto fatto in relazione all’attività di analisi dei requisiti, suddividendo applicazione di test e operatore *kubernetes*;
 - **Progettazione:** descrizione di quanto fatto in relazione all’attività di progettazione, suddividendo applicazione di test e operatore *kubernetes*;
 - **Codifica:** descrizione di quanto fatto in relazione all’attività di programmazione, suddividendo applicazione di test e operatore *kubernetes*;
 - **Verifica:** descrizione di quanto fatto in relazione all’attività di verifica, suddividendo analisi statica e dinamica;
 - **Validazione e collaudo:** descrizione di quanto fatto in relazione all’attività di validazione, e successivo collaudo, con riepilogo finale degli obiettivi raggiunti.
- **Valutazioni retrospettive:** capitolo che conclude il documento elencando e descrivendo gli obiettivi raggiunti. Comprendente un’autovalutazione della mia crescita e del mio percorso di *stage*, ed alcune riflessioni finali sul mio percorso di studi.

Convenzioni tipografiche

Il documento adotta le seguenti convenzioni tipografiche:

- I termini in lingua straniera, compresi acronimi e sigle, sono evidenziati dall'uso del *corsivo*;
- I sottotitoli non presenti nell'indice, e i nomi di elementi di alcuni elenchi, possono venire evidenziati in **grassetto**. Questo per facilitare lettura ed analisi del documento;
- Tutti gli elementi degli elenchi puntati sono separati da punto e virgola (";"), mentre l'ultimo elemento termina con un punto (".");
- Solamente l'iniziale della prima parola di ogni titolo è riportata in maiuscolo, questo, a meno che nel titolo non siano presenti nomi propri.
- Tabelle, immagini e frammenti di codice, riportano un numero progressivo, una descrizione e, se di proprietà di terzi, la fonte.

Indice

Bibliografia	xii
Sitografia	xiii
1 L’azienda	1
1.1 Struttura organizzativa	1
1.2 Principali prodotti e clienti	2
1.3 Processi interni	4
1.3.1 Metodologie in adozione	6
1.3.2 Processi di sviluppo	7
1.4 Tecnologie in utilizzo	9
1.5 Propensione all’innovazione	14
2 Lo stage	16
2.1 La visione dell’azienda	16
2.2 Il ruolo degli <i>stage</i> nel contesto aziendale	17
2.3 Scopo	18
2.4 Vincoli e obiettivi	21
2.5 Motivazione della scelta	31
3 Il progetto	35
3.1 Analisi dei requisiti	35
3.1.1 Scenari e casi d’uso	36
3.1.2 Requisiti	42
3.2 Progettazione	45
3.3 Codifica	50

INDICE

3.4 Verifica	56
3.4.1 Analisi Statica	57
3.4.2 Analisi Dinamica	58
3.5 Validazione e collaudo	60
4 Valutazioni retrospettive	67
4.1 Soddisfacimento degli obiettivi	67
4.2 Crescita personale	70
4.3 Riflessioni finali	71

Elenco delle figure

1.1	Torre Zucchetti, sede principale e simbolo dell'azienda.	2
1.2	Interfaccia Zucchetti <i>inrecruiting</i> , <i>software</i> per la gestione del reclutamento di nuovo personale.	3
1.3	Processo <i>Scrum</i> , suddiviso in fasi.	7
1.4	Architettura d'esempio di un'applicazione Zucchetti di base.	10
1.5	<i>Podman Desktop</i> , interfaccia grafica dello strumento.	12
1.6	<i>Team Zucchetti</i> ai mondiali di calcio 2022.	15
2.1	Conferenze <i>KubeCon & CloudNativeCon, Hong Kong</i> Agosto 2024.	17
2.2	Astrazione funzionamento di base degli operatori.	19
2.3	Metodologia di pianificazione.	22
2.4	Astrazione infrastruttura per la gestione delle cartelle condivise.	25
2.5	<i>Gantt view</i> del progetto.	28
2.6	Crescita professionale.	33
3.1	Casi d'uso visualizzazione della cartella condivisa.	37
3.2	Casi d'uso scalabilità dell'applicazione.	39
3.3	Astrazione schedulazione dell'applicazione.	40
3.4	Astrazione processo di gestione degli aggiornamenti, iterazione 1.	41
3.5	Astrazione processo di gestione degli aggiornamenti, iterazione 2.	42
3.6	Astrazione architettura dell'applicazione di <i>test</i>	45
3.7	Astrazione architettura del <i>cluster Kueernetes</i>	47
3.8	Astrazione processo di aggiornamento, iterazione 1.	48
3.9	Astrazione processo di aggiornamento, iterazione 2.	49

ELENCO DELLE FIGURE

3.10 Fondatori di <i>Go</i> , da sinistra a destra: <i>Robert Griesemer, Rob Pike</i>	52
e <i>Ken Thompson</i>	
3.11 Astrazione dell' <i>Ingress</i> e suo funzionamento.	59

Elenco delle tavole

2.1	Tabella riassuntiva degli obiettivi individuati durante la stesura del piano di lavoro iniziale.	30
3.1	Tabella riassuntiva dei requisiti individuati durante il corso del progetto.	44
3.2	Tabella riassuntiva del numero di linee di codice prodotto. . . .	56
3.3	Tabella riassuntiva dei requisiti coperti durante il corso del progetto.	65
4.1	Tabella retrospettiva, degli obiettivi e loro stato finale.	69

Elenco dei codici sorgenti

3.1	Sezione di codice relativo al <i>manifest</i> dell'applicazione.	51
3.2	Sezione di codice relativo all'applicazione della strategia <i>Canary</i>	54
3.3	Sezione di codice relativo alla funzionalità di <i>Scaling</i>	55
3.4	<i>Output</i> dell'inizio del ciclo di aggiornamento.	61
3.5	<i>Output</i> dei processi di <i>scaling</i> e <i>scheduling</i> dei <i>pod</i>	62
3.6	<i>Output</i> della conclusione della strategia <i>Canary</i>	63
3.7	<i>Output</i> della conclusione della strategia <i>Blue-Green</i>	64

Bibliografia

Sitografia

Acquisizione dell'azienda Ranocchi. URL: <https://www.zucchetti.it/it/cms/news-e-eventi/comunicati/zucchetti-acquisisce-il-gruppo-ranocchi.html> (cit. a p. 1).

Amazon S3. URL: <https://aws.amazon.com/it/pm/serv-s3/> (cit. a p. 24).

Apache Tomacat. URL: <http://tomcat.apache.org/> (cit. a p. 10).

Atlas. URL: <https://atlasgo.io/integrations/kubernetes/operator> (cit. a p. 65).

Cloc. URL: <https://github.com/AlDanial/cloc> (cit. a p. 56).

CloudNativePG. URL: <https://cloudnative-pg.io/> (cit. a p. 65).

CSS. URL: <https://www.w3.org/Style/CSS/Overview.en.html> (cit. a p. 10).

Datashim. URL: <https://github.com/datashim-io/datashim> (cit. a p. 24).

EsLint. URL: <http://eslint.org/> (cit. a p. 11).

Express. URL: <https://expressjs.com/> (cit. a p. 26).

GitHub. URL: <http://github.com/> (cit. a p. 13).

GitLab. URL: <http://about.gitlab.com/> (cit. a p. 13).

Go (Golang). URL: <https://go.dev/> (cit. a p. 20).

HTML. URL: <https://html.spec.whatwg.org/multipage/> (cit. a p. 10).

Java. URL: <http://www.java.com/> (cit. a p. 9).

JavaScript. URL: <http://www.javascript.com/> (cit. a p. 9).

Jenkins. URL: <http://www.jenkins.io/> (cit. a p. 11).

Kubectl. URL: <https://kubernetes.io/docs/reference/kubectl/> (cit. a p. 24).

Kubernetes. URL: <http://kubernetes.io/> (cit. a p. 12).

Linux Foundation, CloudNative. URL: <https://www.cncf.io/> (cit. a p. 16).

Manifesto Agile. URL: <http://agilemanifesto.org/> (cit. a p. 6).

Markdown. URL: <https://www.markdownguide.org/> (cit. a p. 26).

Microsoft Office. URL: <http://www.microsoft.com/microsoft-365/> (cit. a p. 13).

Microsoft Teams. URL: <http://www.microsoft.com/microsoft-teams/> (cit. a p. 13).

Minikube. URL: <https://minikube.sigs.k8s.io/> (cit. a p. 24).

Minio. URL: <https://min.io/> (cit. a p. 24).

Node. URL: <https://nodejs.org/en> (cit. a p. 26).

Object Oriented Programming (OOP). URL: https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti (cit. a p. 52).

Operator SDK. URL: <https://sdk.operatorframework.io/> (cit. a p. 20).

Podman. URL: <http://podman.io/> (cit. a p. 12).

Postgresql. URL: <http://www.postgresql.org/> (cit. a p. 9).

Protocollo HTTP, sito di riferimento. URL: <https://www.w3.org/Protocols/> (cit. a p. 11).

Python. URL: <http://www.python.org/> (cit. a p. 11).

React. URL: <https://react.dev/> (cit. a p. 26).

Scrum. URL: <http://scrum.org/> (cit. a p. 6).

Visual Studio Code. URL: <https://code.visualstudio.com/> (cit. a p. 58).

YAML. URL: <https://yaml.org/> (cit. a p. 49).

Capitolo 1

L’azienda

1.1 Struttura organizzativa

Zucchetti venne fondata da Domenico Zucchetti nel 1978 con sede a Lodi, tuttora centro dell’azienda (Figura 1.1). Tra gli anni Ottanta e Duemila, la società ha ampliato progressivamente la propria presenza in diversi settori, tra cui l’*enterprise resource planning (ERP)*, sistemi *software* integrati che consentono alle aziende di gestire e automatizzare molte attività loro quotidiane, come la contabilità, la gestione delle risorse umane, la fornitura, le vendite, la produzione e molte altre, ma anche, l’automazione industriale, l’uso di tecnologie, sistemi di controllo e *software* per gestire e automatizzare processi industriali, e la robotica. Questo processo di espansione ha incluso anche l’ingresso in alcuni mercati esteri attraverso l’acquisizione di svariate società informatiche. Dal 2008, il Gruppo Zucchetti è guidato dai figli del fondatore, Alessandro e Cristina, la cui *leadership* ha portato l’azienda a crescere significativamente, acquisendo numerose aziende, anche concorrenti. Basti pensare come, la loro ultima operazione, o almeno la sua ufficializzazione¹, risalga a poco meno di due mesi fa, ovvero contemporaneamente al mio tirocinio. Oggi, Zucchetti può vantare una presenza internazionale che conta svariate sedi, oltre 9.000 dipendenti e uno dei fatturati annui più rilevanti nel settore a livello nazionale.

¹ Acquisizione dell’azienda Ranocchi. URL: <https://www.zucchetti.it/it/cms/news-e-eventi/comunicati/zucchetti-acquisisce-il-gruppo-ranocchi.html>.

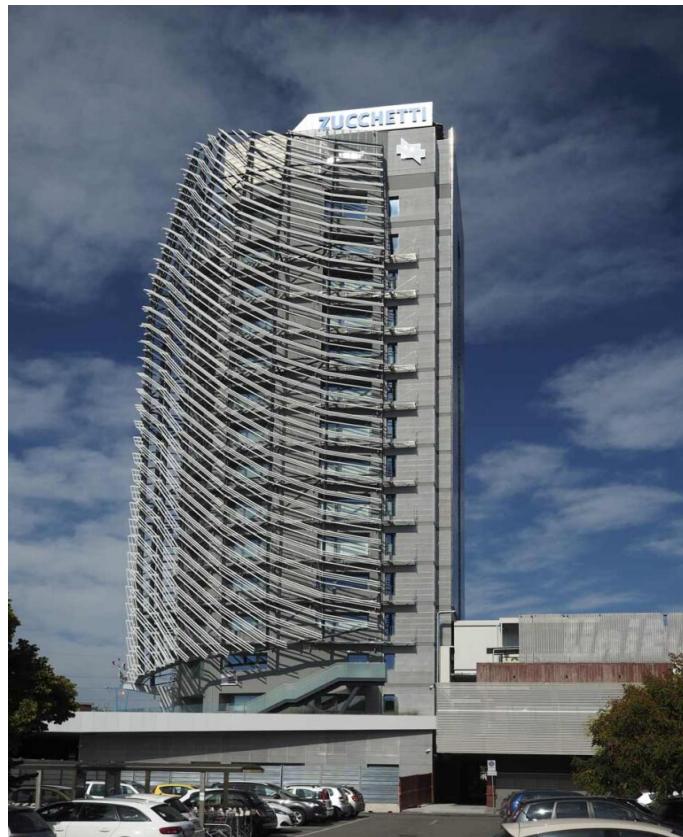


Figura 1.1: Torre Zucchetti, sede principale e simbolo dell'azienda.

Fonte: <https://www.promozioneacciaio.it>

Sono inoltre numerose le certificazioni ed i riconoscimenti ottenuti nel corso degli anni, tra cui: la certificazione *ISO 9001:2015* per il sistema di gestione della qualità, la *ISO 22301* per la gestione della continuità operativa, la *ISO/IEC 27018* per la protezione delle informazioni personali nei servizi *cloud* pubblici e moltissime altre.

1.2 Principali prodotti e clienti

Il Gruppo Zucchetti è *leader* nel settore delle soluzioni *software* e dei servizi tecnologici, offrendo una gamma completa di prodotti per soddisfare le esigenze di diverse tipologie di aziende.

Tra i principali prodotti, possiamo trovare:

- **Zucchetti ERP:** un *software* integrato per la gestione delle risorse aziendali che copre la contabilità, la gestione delle risorse umane, il controllo di gestione e la logistica;
- **HR Infinity:** una *suite* completa per la gestione del personale, che include strumenti per la pianificazione delle presenze, il calcolo delle retribuzioni e l'amministrazione del personale;



Figura 1.2: Interfaccia Zucchetti *inrecruiting*, *software* per la gestione del reclutamento di nuovo personale.

Fonte: <https://hr.zucchetti.it>

- **Infinity Safety:** *software* per la gestione della salute e sicurezza sul lavoro, che permette la valutazione dei rischi e la gestione degli incidenti;
- **Zucchetti e-Fattura:** una soluzione per la creazione, l'invio e la conservazione delle fatture elettroniche, conforme alle normative italiane ed europee;
- **Zucchetti Automation:** sistemi per l'automazione dei processi produttivi, comprendendo soluzioni di robotica e *Internet of Things (IoT)*, con-

cetto tecnologico che si riferisce al dominio di dispositivi fisici connessi a *Internet* e in grado di raccogliere, scambiare e analizzare dati.

- **Zucchetti Cloud:** servizi *cloud*, ovvero risiedenti sull'*Internet*, e *data center*, strutture fisiche utilizzate per ospitare sistemi informatici critici e componenti correlati, come *server*, *storage*, reti e dispositivi di sicurezza, per l'*hosting*, la gestione e la sicurezza dei dati aziendali;
- **Infinity SitePainter:** una piattaforma per la creazione di portali *web* aziendali e siti *internet*, permettendo una gestione semplice e intuitiva dei contenuti *online*.

L'azienda serve un'ampia varietà di clienti, che spaziano dalle piccole e medie imprese a grandi enti e organizzazioni, operando in diversi settori. Tra i principali clienti di Zucchetti si trovano alcune delle più grandi catene di supermercati italiane come Esselunga, che utilizza le loro soluzioni per la gestione dei punti vendita e il controllo degli *stock*; ma anche multinazionali come Ferrero, nel settore dolciario, e il Gruppo Calzedonia, nel settore moda, che beneficiano delle soluzioni della casa di sviluppo per l'automazione e la gestione delle risorse umane. L'azienda collabora inoltre con *leader* nel settore automobilistico come *Fiat Chrysler Automobiles (FCA)*, che implementa le soluzioni *ERP* e di automazione industriale, e con Ferrovie dello Stato per la gestione delle risorse e la sicurezza. Infine, tra i clienti di spicco figurano anche l'Ospedale San Raffaele, per la gestione delle risorse sanitarie, e Banca Mediolanum, per la gestione dei servizi finanziari e la sicurezza dei dati. Zucchetti fornisce soluzioni personalizzate e supporto dedicato, caratteristiche che gli hanno permesso di costruire un portfolio di clienti prestigiosi e diversificati, consolidando la propria posizione di *leadership* nel mercato.

1.3 Processi interni

Le informazioni che riporto, derivano dalla mia esperienza di *stage* in una delle sedi del reparto di ricerca e sviluppo (*RD*) dell'azienda. Tali potrebbero quindi non valere o valere solo in parte per gli altri reparti e sedi.

L'azienda presenta una struttura organizzativa ben definita e processi interni progettati per ottimizzare l'efficienza e garantire elevati *standard* di qualità in tutte le sue operazioni. I principali processi interni dell'azienda comprendono:

- **Consulenza:** per la quale l'azienda predispone un servizio clienti progettato per offrire assistenza tempestiva e professionale attraverso supporto tecnico dedicato, che comprende *help desk*, assistenza telefonica e *online*, e servizi di consulenza personalizzata;
- **Fornitura:** processo che gestisce l'approvvigionamento e la distribuzione dei prodotti, assicurando che le soluzioni siano consegnate ai clienti in modo efficiente e conforme agli *standard* di qualità aziendali. Zucchetti da molta importanza alla trasparenza con il cliente, all'interno dell'azienda questo processo è infatti strettamente legato a quello di consulenza e controllo della qualità.
- **Controllo della qualità:** per la quale l'azienda adotta rigorosi *standard* di controllo per assicurare che tutti i prodotti e servizi soddisfino le aspettative dei clienti. Il processo include *test* approfonditi e controlli degli *standard* interni che seguono certificazioni secondo le normative internazionali come la già citata *ISO 9001:2015*;
- **Sviluppo delle competenze del personale:** processo focalizzato sul reclutamento e sulla crescita professionale dei dipendenti. Zucchetti promuove un ambiente di lavoro stimolante e collaborativo, offrendo programmi di formazione continua per lo sviluppo delle competenze.
- **Gestione delle infrastrutture IT:** comprende la gestione delle infrastrutture tecnologiche, la manutenzione dei sistemi e la gestione dei dati. Zucchetti utilizza soluzioni avanzate di *cloud computing* e *data center* per garantire sicurezza ed affidabilità operativa.
- **Manutenzione:** il processo di manutenzione include la gestione continua delle soluzioni *software* e *hardware* fornite ai clienti. L'azienda assicura che i suoi prodotti siano sempre aggiornati e funzionanti, fornendo supporto

per eventuali problemi tecnici e implementando miglioramenti periodici per rispondere alle esigenze del mercato.

Questi processi interni sono supportati da una cultura aziendale orientata all'innovazione, alla qualità e al servizio eccellente, che puntano a rispondere in modo efficace alle esigenze in continua evoluzione dei suoi clienti.

Pilastro di ogni *software house*, vengono poi i processi di sviluppo e gestione del progetto.

1.3.1 Metodologie in adozione

Per la gestione dei progetti e lo sviluppo *software*, l'azienda, fa uso di una metodologia di stampo *Agile*², approccio pensato per affrontare le sfide ricorrenti nei progetti *software*, come la necessità di flessibilità, la rapida evoluzione dei requisiti e l'importanza della collaborazione e trasparenza continua tra *team* e clienti.

Il modello *Agile* si basa su una serie di principi e pratiche che mirano a migliorare la reattività e l'efficacia dei *team* di sviluppo. Essi stati formalizzati nel Manifesto *Agile*, pubblicato nel 2001, che enfatizza valori fondamentali come:

- **Valorizzare gli individui e le interazioni** rispetto ai processi e agli strumenti;
- **Preferire il *software* funzionante** alla documentazione dettagliata;
- **Promuovere la collaborazione con il cliente** piuttosto che focalizzarsi sulla negoziazione dei contratti;
- **Adattarsi al cambiamento** anziché seguire rigidamente un piano.

Nello specifico, come ho potuto constatare nella sede nella quale ho svolto il tirocinio, l'azienda adotta un modello che risulta avvicinarsi molto a quello rappresentato dal *framework Scrum*³.

²Manifesto Agile. URL: <http://agilemanifesto.org/>.

³Scrum. URL: <http://scrum.org/>.

1.3.2 Processi di sviluppo

Lo sviluppo di ogni progetto, compreso quello relativo al mio *stage*, attraversa varie fasi che si rifanno a quanto previsto da questa metodica, osservabili in figura 1.3, seguente.

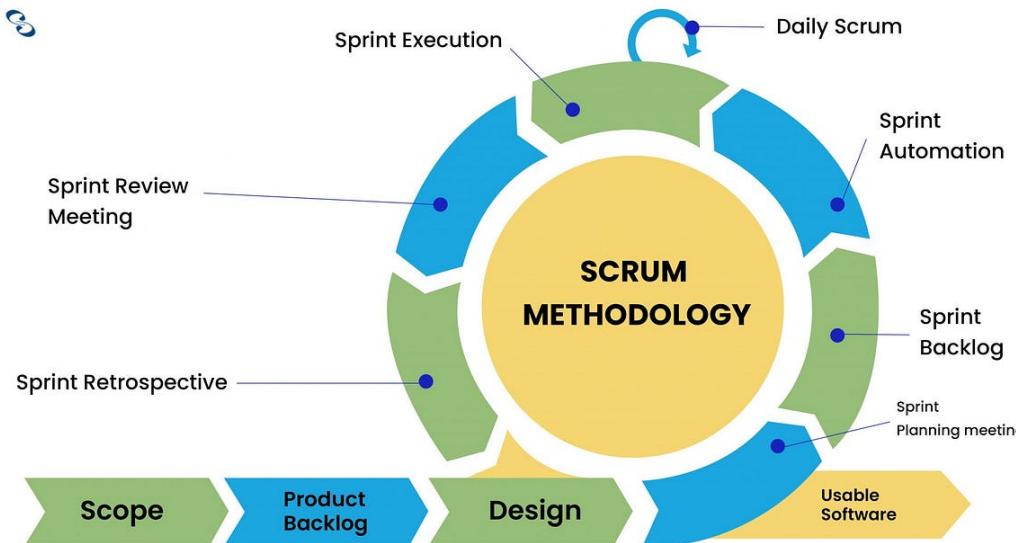


Figura 1.3: Processo *Scrum*, suddiviso in fasi.

Fonte: <https://blog.stackademic.com>

Inizialmente, il progetto viene suddiviso in periodi, detti *sprint*, nel mio caso di durata settimanale, ai quali corrispondono determinati obiettivi riconducibili solitamente a specifiche *feature* del prodotto finale. Tali obiettivi vengono raccolti in un documento denominato ***Product Backlog***, una lista prioritaria di funzionalità, *bug* e miglioramenti che dovranno essere implementati.

Nel caso concreto del mio *stage*, ciò si rispecchia nella redazione dell'iniziale piano di lavoro.

Similmente, l'azienda ne produce uno a partire dalle esigenze e richieste del cliente o dall'idea e obiettivi del progetto, nel caso quest'ultimo sia di ricerca. Ogni *sprint* inizia quindi con una fase detta, di ***Sprint Planning***, durante la quale il *team* di sviluppo decide quali elementi del *backlog*, devono essere affrontati durante l'iterazione. Durante questa fase, il *team* discute e definisce

chiaramente gli obiettivi e i criteri di accettazione per ogni *ticket*, attività.

Come ho avuto modo di osservare l'azienda definisce questi ultimi solitamente durante un breve *meeting* che comprende principalmente gli sviluppatori, per poi suddividersi le varie attività, assegnando i compiti e gestendo il progetto attraverso vari strumenti di supporto che descriverò successivamente.

Corrispondentemente a quanto io abbia fatto ad ogni settimana, aggiornando in modo incrementale il preventivato piano di lavoro.

Durante lo *sprint* poi, il *team* si incontra quotidianamente in brevi riunioni chiamate **Daily Stand-up** o **Daily Scrum** con lo scopo di sincronizzare le attività del *team*, risolvere eventuali impedimenti e pianificare il lavoro giornaliero.

Per quanto riguarda la mia esperienza essi sono associabili alle interazioni quotidiane, anche se non calendarizzate ma in base alle necessità di ambe le parti, con il mio *tutor* aziendale. L'azienda predilige invece, quando possibile, la loro calendarizzazione. Inoltre i *meeting* avvengono sia tra gli sviluppatori, che talvolta, con clienti e collaboratori.

Successivamente, al termine di ogni *sprint*, si tiene una **Sprint Review**, durante la quale il *team* dimostra le funzionalità completate agli *stakeholder* e raccoglie *feedback*. Questo momento è cruciale per assicurarsi che il prodotto soddisfi le aspettative dei clienti e per apportare eventuali correzioni o miglioramenti. Anche questa penultima fase è riconducibile a quanto ho vissuto durante il tirocinio. Alla fine di ogni settimana di lavoro infatti, attraverso un *meeting* scadenzato, illustravo al mio *tutor* quanto fatto e raggiunto, raccogliendo *feedback* e consigli. Tale *meeting* andava poi ad impattare sul successivo *sprint planning*, rivedendo il piano di lavoro in base alle nuove priorità.

A questo proposito, l'azienda sembra tenere questi incontri più raramente, o perlomeno con una minore frequenza. Infine, ogni *sprint* si conclude con una **Sprint Retrospective**, una riunione interna del *team*, essenziale per il miglioramento costante delle pratiche e delle dinamiche di squadra, volta a riflettere su ciò che è andato bene, ciò che potrebbe essere migliorato e come apportare miglioramenti concreti al processo di sviluppo. Nel corso della mia esperienza ho avuto modo di confrontarmi settimanalmente, oltre che con il mio *tutor* aziendale, e con me stesso, con il mio relatore. Andando sostanzialmente in

parte, a replicare questa fase, ripercorrendo quanto fatto e affrontato. All'interno dell'azienda invece, come mi è stato possibile osservare, solitamente queste retrospettive coinvolgono gli sviluppatori incaricati e i loro punti di riferimento, come altri sviluppatori *senior*, ovvero a loro superiori in esperienza, o *project manager*.

1.4 Tecnologie in utilizzo

La sua vasta espansione e la numerosa acquisizione di altri enti nel corso della sua storia, ha portato l'azienda a diversificarsi in moltissimi settori con un'ampia varietà di prodotti e di servizi offerti. Ciò si riflette nel utilizzo di una vasta gamma di linguaggi di programmazione, strumenti di supporto e *framework*. Inoltre, per mantenere il suo vantaggio competitivo e rispondere alle evoluzioni del mercato, Zucchetti investe significativamente nella formazione continua dei propri dipendenti per, e nell'adozione di moderne e nascenti tecnologie.

Durante il mio *stage* ho avuto modo di osservare da vicino alcune delle tecnologie e strumenti, da loro, più comunemente utilizzati. Tra questi, spiccano i seguenti:

- **Java**⁴: linguaggio di programmazione ad oggetti ampiamente utilizzato per la sua robustezza, portabilità e scalabilità. In Zucchetti, impiegato principalmente per lo sviluppo di applicazioni *enterprise*, soprattutto come linguaggio di programmazione *server-side*, ovvero che si occupa di tutto ciò che avviene e viene gestito su un *server* nelle applicazioni *web*;
- **JavaScript**⁵: linguaggio di *scripting* versatile, utilizzato sia lato *client* che lato *server*. Per l'azienda fondamentale per lo sviluppo di interfacce *web* interattive e dinamiche;
- **Postgresql**⁶: sistema di gestione di *database* relazionale *open-source* conosciuto per la sua affidabilità e le sue avanzate capacità di gestione dei

⁴ Java. URL: <http://www.java.com/>.

⁵ JavaScript. URL: <http://www.javascript.com/>.

⁶ Postgresql. URL: <http://www.postgresql.org/>.

dati. Zucchetti utilizza *PostgreSQL* per archiviare e gestire dati nelle sue applicazioni, sfruttando le sue potenti funzionalità di *query* e la sua compatibilità con altri strumenti di analisi dati;

- **Apache Tomcat**⁷: *server* applicativo *open-source* per l'esecuzione di applicazioni *Java*, in particolare quelle basate su *servlet* e *JSP*. I *servlet* sono classi *Java* utilizzate per estendere le funzionalità di un *server web*, mentre *JSP*, è una tecnologia che consente di creare pagine *web* dinamiche utilizzando *Java*. Viene utilizzato dall'azienda per il *deployment* di applicazioni *web*;

Durante il tirocinio, ho avuto la possibilità di usare un'applicazione base dell'azienda per alcuni *test* utili al mio progetto, che approfondirò in seguito. Allo stesso tempo, ciò mi ha permesso di capire come le tecnologie da loro utilizzate siano integrate le une con le altre. Nell'immagine 1.4 sottostante, è possibile vederne un'approssimazione.

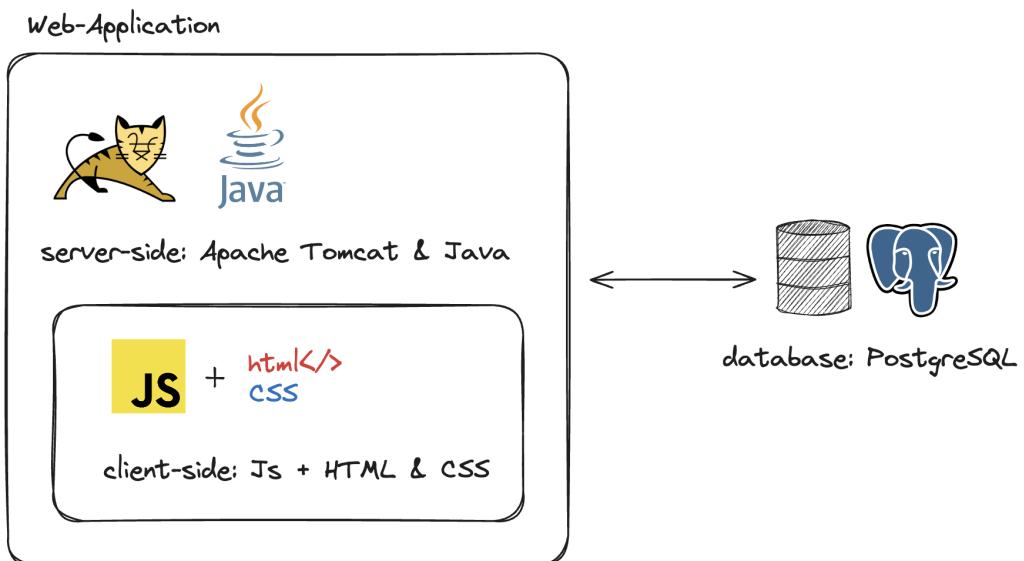


Figura 1.4: Architettura d'esempio di un'applicazione Zucchetti di base.

Nello specifico nella *web-application* dell'azienda: *HTML*⁸ e *CSS*⁹ gesti-

⁷Apache Tomacat. URL: <http://tomcat.apache.org/>.

⁸HTML. URL: <https://html.spec.whatwg.org/multipage/>.

⁹CSS. URL: <https://www.w3.org/Style/CSS/Overview.en.html>.

scono la struttura e lo stile delle pagine *web*. *JavaScript* è utilizzato per la logica *client-side*, come la gestione degli eventi e le richieste al *server*. Mentre *Java* è utilizzato per la logica di *business*. I *servlet* gestiscono le richieste *HTTP*¹⁰, interagiscono con il *database*, basato su *PostgreSQL*, e generano risposte, spesso sotto forma di pagine *JSP* o dati *JSON*, formato di testo "leggero" per lo scambio di dati.

- ***Python***¹¹: linguaggio di programmazione flessibile e dinamico, noto per la sua applicazione in vari campi come il *machine learning*, l'automazione e lo *scripting*, che l'azienda utilizza principalmente in progetti innovativi e di ricerca;
- ***Jenkins***¹²: *tool* di integrazione continua *open-source* che automatizza il processo di *build*, *test* e *deployment* delle applicazioni. In Zucchetti utilizzato per assicurare che il *software* venga continuamente integrato e verificato, preservandone la qualità;
- ***EsLint***¹³: Uno strumento di *linting* per *JavaScript* che aiuta a identificare e risolvere problemi nel codice. L'azienda lo integra nel processo di sviluppo per garantire che il codice *JavaScript* sia conforme agli *standard* interni e sia privo di errori potenziali, migliorando la manutenzione e la leggibilità del codice;

¹⁰Protocollo *HTTP*, sito di riferimento. URL: <https://www.w3.org/Protocols/>.

¹¹*Python*. URL: <http://www.python.org/>.

¹²*Jenkins*. URL: <http://www.jenkins.io/>.

¹³*EsLint*. URL: <http://eslint.org/>.

- **Podman**¹⁴: Tecnologia di containerizzazione che permette di creare, distribuire e gestire applicazioni in ambienti isolati e portabili, detti *container*. Zucchetti ne fa uso per garantire che le applicazioni siano facilmente scalabili e distribuibili su diverse piattaforme, migliorando l'efficienza operativa e riducendo le incompatibilità tra ambienti di sviluppo e produzione;

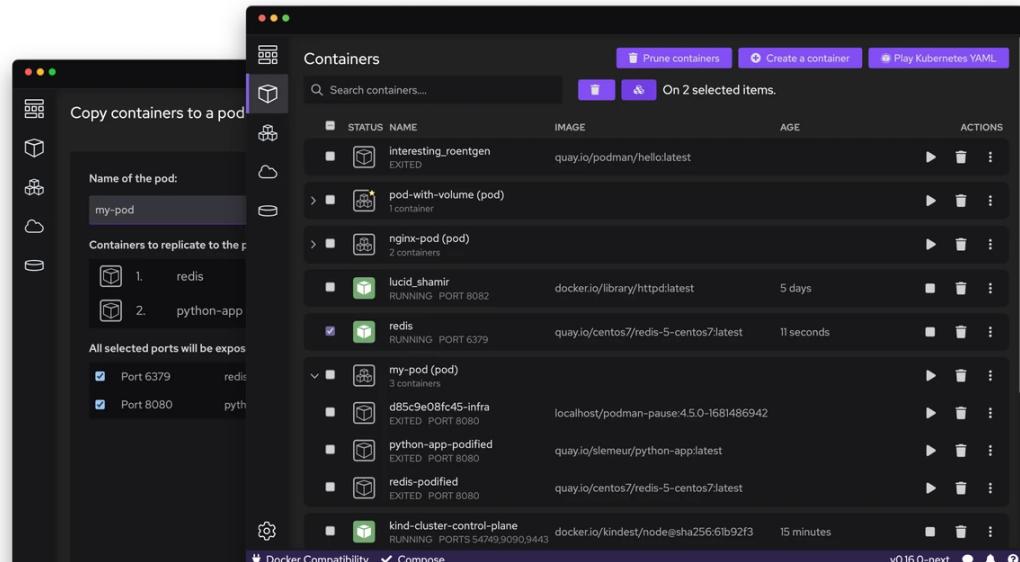


Figura 1.5: *Podman Desktop*, interfaccia grafica dello strumento.

Fonte: <https://podman.io>

- **Kubernetes**¹⁵: Una piattaforma *open-source* per l'orchestrazione di *container* che automatizza la distribuzione, la gestione e lo *scaling* delle applicazioni containerizzate. Viene utilizzato dall'azienda per gestire in modo efficace le applicazioni distribuite su larga scala, in progetti di ricerca e sviluppo.

Per quanto riguarda la gestione dei progetti e la collaborazione tra *team*, Zucchetti utilizza una serie di strumenti avanzati che facilitano il lavoro di squadra, la comunicazione e il versionamento del *software*. Questi strumenti sono fonda-

¹⁴Podman. URL: <http://podman.io/>.

¹⁵Kubernetes. URL: <http://kubernetes.io/>.

mentali per mantenere l'efficienza operativa e garantire che tutti i membri del *team* siano allineati sugli obiettivi del progetto.

- ***GitHub***¹⁶: Una piattaforma di *hosting* per il controllo di versione e la gestione del codice sorgente basata su *Git*. In Zucchetti, *GitHub* è utilizzato per la gestione collaborativa del codice, consentendo ai *team* di sviluppo di lavorare insieme in modo efficiente;
- ***GitLab***¹⁷: Simile a *GitHub*, *GitLab* è un'altra piattaforma basata su *Git*, ma con un'attenzione particolare all'integrazione continua e alla distribuzione continua (*CI/CD*);
- **La *suite Office* di Microsoft**¹⁸: Un insieme di applicazioni di produttività che includono *Word*, *Excel*, *PowerPoint* e *Outlook*. In Azienda, la *suite* è utilizzata per una vasta gamma di attività amministrative e operative, come la redazione di documenti, l'analisi dei dati, la preparazione di presentazioni e la gestione della corrispondenza elettronica;
- ***Microsoft Teams***¹⁹: Una piattaforma di collaborazione che integra *chat*, video conferenze, archiviazione di *file* e applicazioni di lavoro. Utilizzato dall'azienda per facilitare la comunicazione e la collaborazione sia tra i *team*, soprattutto in un contesto di lavoro remoto o distribuito, sia con esterni, quali clienti o collaboratori.

Degno di nota è anche l'adozione di applicazioni sviluppate internamente da Zucchetti, progettate per ottimizzare la visualizzazione e la gestione dei progetti. Questi strumenti personalizzati si integrano perfettamente con le piattaforme già in uso, come *GitHub*, permettendo all'azienda di sfruttare al meglio le proprie tecnologie e risorse, garantendo al contempo una gestione più efficiente e su misura delle attività di sviluppo.

¹⁶ *GitHub*. URL: <http://github.com/>.

¹⁷ *GitLab*. URL: <http://about.gitlab.com/>.

¹⁸ *Microsoft Office*. URL: <http://www.microsoft.com/microsoft-365/>.

¹⁹ *Microsoft Teams*. URL: <http://www.microsoft.com/microsoft-teams/>.

1.5 Propensione all'innovazione

Zucchetti, forte della sua affermazione e solidità finanziaria derivanti da una storia di numerosi successi, pone grande attenzione e risorse nell'innovazione, sia nell'ambito tecnologico che industriale. Questa propensione all'innovazione si riflette in diversi aspetti della sua organizzazione e delle sue operazioni.

In particolare, secondo i dati aziendali, dei 9000 dipendenti di Zucchetti, quasi un terzo è impiegato nel settore di ricerca e sviluppo (*RD*). Questo impegno significativo dimostra la volontà dell'azienda di investire costantemente in nuove tecnologie e soluzioni innovative. La sede di Padova, dove ho avuto l'occasione di svolgere il mio tirocinio, è un esempio tangibile di questo impegno, essendo essa stessa un centro dedicato alla ricerca e sviluppo. Questa esperienza mi ha permesso di osservare da vicino l'importanza che l'azienda attribuisce ai progetti e processi orientati all'innovazione.

Durante il mio *stage*, ho potuto constatare che molti dei progetti in corso erano focalizzati su aree all'avanguardia come l'automazione industriale, l'intelligenza artificiale e il *cloud computing*.

Zucchetti inoltre collabora attivamente con numerose entità esterne, tra cui università, centri di ricerca e aziende private a livello internazionale. Queste collaborazioni non solo ampliano le competenze e le risorse a disposizione dell'azienda, ma favoriscono anche uno scambio continuo di conoscenze e idee. Alcune delle collaborazioni degne di nota includono partnership con il Politecnico di Milano, l'Università di Bologna e l'Istituto Italiano di Tecnologia.

L'azienda partecipa regolarmente a progetti di ricerca europei e internazionali, ottenendo finanziamenti e riconoscimenti per le sue soluzioni innovative. Questi progetti coprono una vasta gamma di settori, dall'*ICT* (*Information and Communication Technology*) alla *smart manufacturing*, confermando la capacità di Zucchetti di adattarsi e innovare in vari contesti.

Uno dei più recenti e importanti, quello per la sicurezza degli stadi ai mondiali di calcio 2022. Durante i quali Zucchetti ha avuto la responsabilità di gestire gli accessi a 6 degli 8 stadi per la durata di tutta la competizione. Un *team* di esperti, visibili in figura 1.6, si è occupato dell'installazione e monitoraggio dei tornelli automatici per l'affluenza del pubblico.



Figura 1.6: Team Zucchetti ai mondiali di calcio 2022.

Fonte: <https://zucchetti.it>

Inoltre, l'azienda, promuove e partecipa a conferenze, *workshop* e *hackathon*, eventi che incentivano la creatività e l'innovazione tra i suoi dipendenti e nella comunità tecnologica più ampia. Questi eventi offrono opportunità per esplorare nuove idee, sviluppare prototipi e collaborare con esperti del settore.

Grazie a questa strategia integrata di investimenti in *RD*, collaborazioni esterne e coinvolgimento attivo nella comunità tecnologica, Zucchetti riesce a mantenere un alto livello di innovazione, rispondendo efficacemente alle esigenze in continua evoluzione dei suoi clienti e del mercato globale.

Capitolo 2

Lo stage

2.1 La visione dell’azienda

L’azienda, da sempre orientata all’innovazione tecnologica, ha riconosciuto da tempo l’importanza strategica del mondo *Cloud Native*¹. Questa filosofia di sviluppo, che promuove la creazione di applicazioni resilienti, scalabili e facilmente gestibili nel *cloud*, rappresenta per Zucchetti un passo fondamentale verso il futuro dell’informatica aziendale. Zucchetti, come già descritto, fornisce ai suoi clienti una vasta gamma di servizi che richiedono aggiornamenti continui, operazioni di manutenzione e, talvolta, personalizzazioni su richiesta. Questi interventi non solo necessitano di risorse significative, ma possono anche avere tempi di applicazione variabili. L’obiettivo dell’azienda è garantire non solo l’eccellenza del *software*, ma anche la massima continuità ed affidabilità del servizio, mantenendo elevata l’efficienza e riducendo i consumi. Questo approccio non solo consente di beneficiare di un miglioramento economico, ma promuove anche la sostenibilità, un valore a cui l’azienda si è dedicata con impegno negli anni. Come illustrerò a breve, in questo contesto, il progetto che mi è stato affidato assume un ruolo centrale.

La visione dell’azienda è chiara: creare strumenti che permettano ai clienti di sfruttare appieno le potenzialità del *cloud*, senza doversi preoccupare di complessità tecniche o di gestione.

¹Linux Foundation, CloudNative. URL: <https://www.cncf.io/>.

Attraverso questo progetto, Zucchetti punta ad innovarsi ulteriormente. Nello specifico la mia esperienza rappresenterà uno dei primi passi di ricerca verso questo mondo, quello del *Cloud Native*. Attualmente infatti, l'azienda è ancora in fase di esplorazione di queste tecnologie emergenti, che si ritrovano nel pieno del loro sviluppo. Basti pensare come, una delle più importanti conferenze 2.1 al riguardo, si terrà a breve, rispetto alla stesura di questo stesso documento.

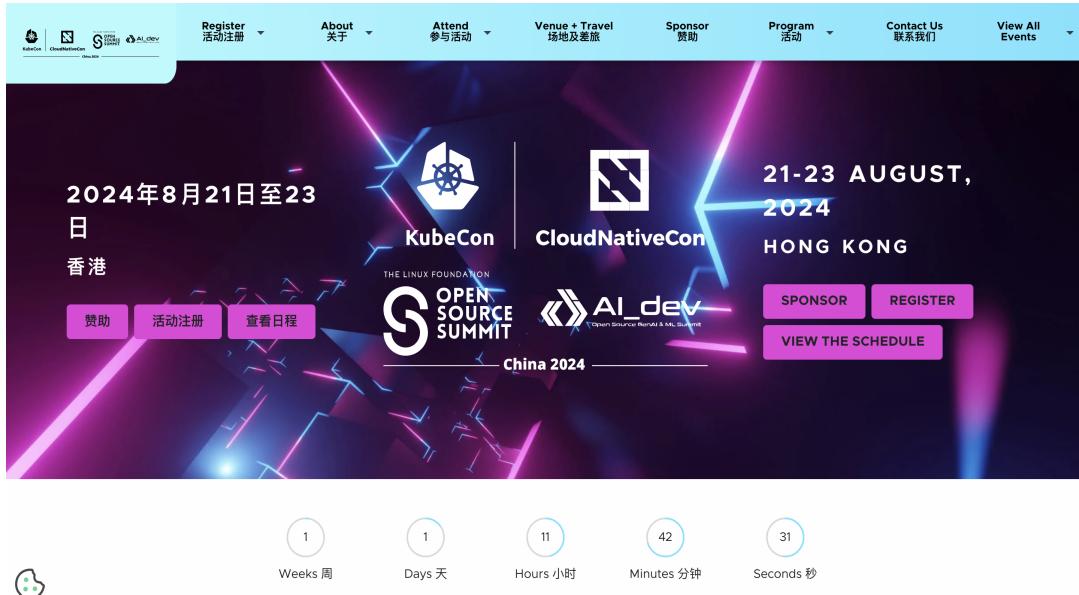


Figura 2.1: Conferenze *KubeCon & CloudNativeCon, Hong Kong Agosto 2024*.

Fonte: <https://events.linuxfoundation.org>

Riassumendo, lo *stage* mira a fornire esempi concreti, *test* e risultati che aiuteranno Zucchetti ad ottenere una visione più chiara e informata sul potenziale e sull'applicabilità di tali soluzioni nel loro contesto.

2.2 Il ruolo degli *stage* nel contesto aziendale

Gli *stage* rappresentano per l'azienda non solo una risorsa strategica per esplorare e innovare in ambiti tecnologici avanzati, ma anche un'opportunità per lo *scouting*, reclutamento, e formazione di nuovi giovani. Zucchetti utilizza gli *stage* come un canale per la sperimentazione di nuove tecnologie, approfittando di progetti che richiedono tempo e risorse i quali non sempre, le aziende

in generale, sono disposte o hanno la possibilità di investire.

L’azienda dimostra una visione lungimirante nel trattare gli *stage* come vere e proprie opportunità di sviluppo tecnologico. I progetti assegnati agli stagisti sono frequentemente orientati verso le ultime innovazioni, come l’intelligenza artificiale, il *cloud computing*, e l’*Internet delle Cose (IoT)*, che rappresentano le frontiere più avanzate del settore tecnologico. Questi progetti non solo contribuiscono al progresso dell’azienda, ma permettono anche agli stagisti di lavorare su tecnologie all’avanguardia e di acquisire competenze preziose.

Durante il mio tirocinio, ho avuto l’opportunità di osservare da vicino come Zucchetti gestisce questi processi. Ho visto come l’azienda valorizza le capacità e le idee dei giovani, integrandoli in progetti reali e offrendo loro un ambiente di apprendimento stimolante. Allo stesso modo, ho potuto constatare il valore che gli altri stagisti apportano, contribuendo a progetti paralleli e acquisendo esperienza pratica nelle stesse aree innovative.

In alcuni casi, Zucchetti riconosce il potenziale degli stagisti e offre loro opportunità di lavoro al termine dello *stage*. Questo approccio consente all’azienda di accogliere talenti già formati e familiarizzati con i progetti e la cultura aziendale, garantendo una continuità di competenze e contribuendo alla crescita e all’innovazione continua.

In questo contesto, gli *stage* non sono solo un’opportunità per la crescita professionale degli stagisti, ma anche una strategia per l’azienda per rimanere all’avanguardia e alimentare una cultura di innovazione continua.

2.3 Scopo

Il progetto previsto dal mio *stage* ha l’obiettivo di compiere uno studio di ricerca sugli operatori per *Kubernetes*, andandone a sviluppare uno che possieda alcune *feature* specificate dall’azienda. Gli operatori non sono altro che programmi, ciclici, che ne estendono le capacità native, per gestire applicazioni e servizi complessi attraverso la definizione e l’automazione di logiche di business specifiche e personalizzate.

Nello specifico questi ultimi, hanno il ruolo di automatizzare alcuni processi che

soltamente necessiterebbero l'intervento di un lavoratore addetto.

Essi sono composti da due componenti principali:

- **Custom Resource Definitions (CRD)**: che permettono di estendere l'API di *Kubernetes* per creare e gestire nuove risorse personalizzate.
- **Custom Controllers**: responsabili della gestione e del controllo dello stato delle risorse personalizzate definite attraverso i CRD. Un controller è un programma che osserva lo stato delle risorse personalizzate e applica le azioni necessarie per mantenerle nello stato desiderato.

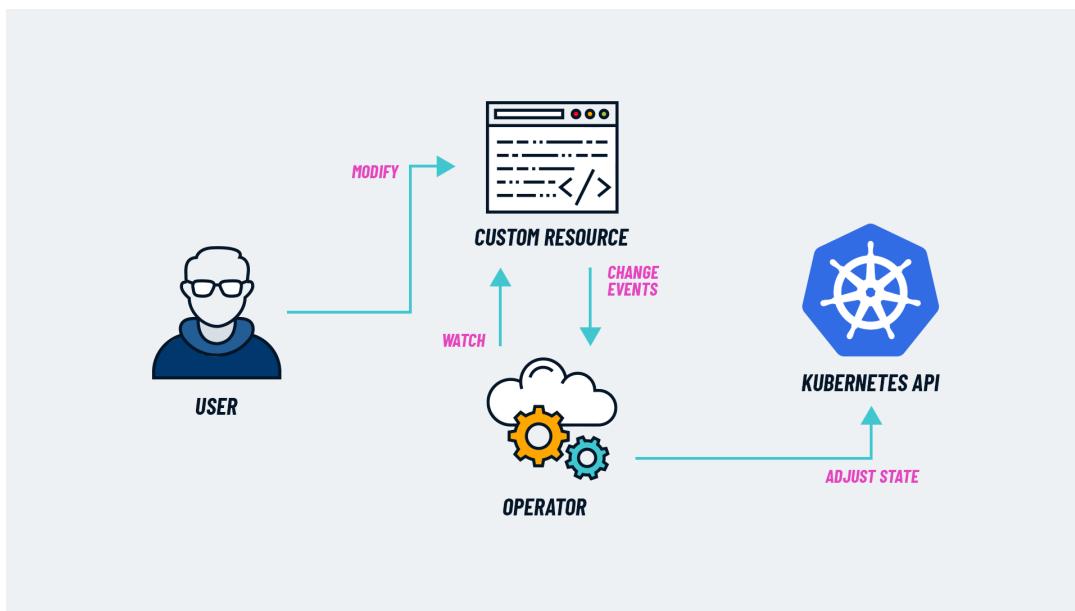


Figura 2.2: Astrazione funzionamento di base degli operatori.

Fonte: <https://www.cncf.io>

L'operatore pensato dall'azienda, doveva avere il compito di facilitare la gestione di applicazioni persistenti, all'interno dell'ambiente *Kubernetes*, garantendo non solo l'efficienza operativa, ma anche la continuità del servizio e la sicurezza dei dati.

Nello specifico l'azienda ha voluto porre l'attenzione su alcuni macro argomenti:

- **Scalabilità**: rappresenta la capacità di un sistema di gestire un aumento della richiesta di risorse o di carico di lavoro senza compromettere le sue

prestazioni o stabilità. L'operatore doveva supportare la crescita dinamica e la distribuzione equa delle risorse per rispondere efficacemente a variazioni nel carico di lavoro.

- **Gestione degli aggiornamenti:** fondamentale per mantenere le applicazioni e l'infrastruttura al passo con le ultime versioni e *patch*. L'operatore mirava a facilitare e automatizzare l'aggiornamento delle applicazioni senza causare interruzioni del servizio, garantendo che i cambiamenti fossero applicati in modo sicuro e senza impatti negativi sull'operatività.
- **Persistenza:** capacità di mantenere i dati e lo stato delle applicazioni anche in caso di riavvii o guasti dei nodi (*server*). L'operatore doveva assicurare che i dati fossero memorizzati in modo duraturo e che l'accesso ai dati persistenti fosse sempre affidabile e sicuro, anche in scenari di *failover* o recupero di emergenza.

Esso doveva essere sviluppato in *Go (Golang)*², linguaggio di programmazione *open-source* sviluppato da *Google*, progettato per essere semplice, efficiente e altamente performante, attraverso l'uso del *Operator SDK*³, strumento che fornisce una serie di *tools* e *framework* per semplificare la creazione, il *test* e la distribuzione degli operatori.

²*Go (Golang)*. URL: <https://go.dev/>.

³*Operator SDK*. URL: <https://sdk.operatorframework.io/>.

2.4 Vincoli e obiettivi

Il primo *step* per avviare il tirocinio, prevede la stesura di un piano di lavoro che definisca vincoli e obiettivi dello *stage* e relativo progetto. Esso viene redatto dallo studente insieme al proprio *tutor* aziendale in modo preventivo, per poi venire approvato da *tutor* aziendale, *tutor* interno (relatore della tesi) e dall'amministratore degli *stage* universitari. In quanto tale, lo stesso, subisce solitamente, come anche nel mio caso, modifiche che seguono e si adattano all'avanzare del progetto.

L'attività di pianificazione ha perciò seguito tutto lo sviluppo del progetto in modo incrementale. Al fine di mantenere un ritmo di lavoro produttivo ed efficace è stata posta molta importanza alla definizione di obiettivi che risultassero concreti e raggiungibili. Tali venivano infatti rivisti al termine di ogni settimana di lavoro, valutando in base ai progressi ottenuti, se fosse necessario o meno, apportare modifiche al piano di lavoro così da allinearsi alle esigenze emergenti e alle sfide incontrate.

La pianificazione flessibile ha consentito di adattare rapidamente il piano di lavoro alle reali condizioni del progetto, garantendo che ogni fase fosse realizzata con precisione e che il progetto rimanesse sempre in linea con gli obiettivi prefissati. Questa modalità di lavoro ha facilitato l'identificazione e la risoluzione tempestiva di eventuali problemi, migliorando l'efficienza e l'efficacia complessiva.



Figura 2.3: Metodologia di pianificazione.

Il principale strumento di gestione utilizzato è stato *GitHub*, che ha svolto un ruolo cruciale nella gestione del codice, nella documentazione e nel tracciamento delle modifiche. *GitHub* ha permesso una gestione centralizzata delle versioni, facilitando la collaborazione e la revisione. Nello specifico ogni progresso degno di nota è stato reso identificabile da specifici e consoni *commit*.

Vincoli temporali e organizzativi:

Il tirocinio prevedeva una durata di 8 settimane per un totale di 320 ore, 40 l'una, a partire dal 10 Giugno 2024 fino al 2 Agosto 2024. L'orario di lavoro si teneva dalle 09:00 alle 18:00 con un'ora di pausa pranzo tra le 13:00 e le 14:00 e alcune pause ordinarie di breve durata, previste per tutti i dipendenti. Nello specifico tutti i giorni di lavoro sono stati svolti in loco ad eccezione di 2, svolti in modalità *smart* per necessità personali.

Rapporti con *tutor* e azienda:

CAPITOLO 2. LO STAGE

Durante il periodo dello *stage*, i rapporti con il *tutor*, e talvolta, con gli altri dipendenti dell’azienda, hanno giocato un ruolo importante per il progresso e successo del progetto. Il mio *tutor* aziendale mi ha fornito un supporto costante e strategico, che è risultato essenziale per orientare e guidare il progetto.

Ho avuto la fortuna di lavorare a stretto contatto con lui, poiché la mia postazione di lavoro era adiacente alla sua. Questo ha facilitato un’interazione continua e immediata, permettendoci di affrontare questioni e discutere idee con facilità. Inoltre, abbiamo stabilito fin da subito *meeting* regolari, programmati su base settimanale, che ci hanno consentito di monitorare i progressi, risolvere problematiche e pianificare le attività future. La comunicazione è stata caratterizzata da un approccio collaborativo e costruttivo, con il *tutor* sempre disponibile a fornire chiarimenti e suggerimenti anche al di fuori degli incontri formali.

Oltre che con il mio *tutor* aziendale, ho avuto l’occasione di confrontarmi con altri dipendenti e supervisori, discutendo, raccogliendo pareri e consigli differenti, che hanno tutti contribuito in parte, al completamento del progetto. Infine, come già menzionato, anche i rapporti con il mio *tutor* interno, e relatore, che ho trovato sempre disponibile e interessato, hanno avuto peso nel compimento dello *stage*.

Revisioni:

Le revisioni congiunte del progetto sono state un elemento cruciale per monitorare e migliorare continuamente il lavoro svolto. Settimanalmente, grazie al mio *tutor* interno, ho avuto modo di ripercorrere tutte le attività da me svolte, stilando dei brevi resoconti della settimana che potessero in un certo senso fotografare lo stato del progetto. Parallelamente, insieme al mio *tutor* aziendale, abbiamo discusso e valutato quanto completato pianificando poi, in base ai risultati ottenuti, il periodo successivo. Inoltre mensilmente, quindi complessivamente 2 volte, ho svolto delle revisioni più approfondite le quali prevedevano una presentazione e dimostrazione del lavoro svolto.

Vincoli tecnologici:

L’azienda ha inoltre voluto definire il dominio tecnologico che avrei dovuto innanzitutto studiare, e successivamente, utilizzare per la realizzazione del proget-

to ma anche sua gestione. Sebbene fosse necessario aderire a certe tecnologie, l’azienda mi ha anche concesso un’ampia libertà su altre scelte, specialmente per le tecnologie complementari. Tra le tecnologie utilizzate vi sono:

- **GitHub**: per il versionamento, salvataggio e condivisione di codice e documentazione prodotta;
- **Docker/Podman**: strumenti essenziali per la containerizzazione delle applicazioni. Sono stati utilizzati per creare, gestire e distribuire i container, risultando necessari per garantire l’integrazione con *Kubernetes*;
- **Kubernetes**: piattaforma di orchestrazione container utilizzata per gestire e scalare le applicazioni containerizzate. Centrale per la gestione dei servizi e dell’applicazione distribuita all’interno del progetto.
- **Minikube e Kubectl**: strumenti proposti dal sottoscritto e successivamente approvati dall’azienda, per la gestione e la simulazione di un *cluster Kubernetes* in ambiente di sviluppo. *Minikube*⁴ consente di eseguire un *cluster Kubernetes* locale, mentre *Kubectl*⁵ è utilizzato per interagire con il *cluster* e gestire le risorse.
- **Go (Golang)**: per lo *scripting* dell’operatore.
- **MinIO**⁶: *software open-source* progettato per fornire una soluzione di *storage* compatibile con l’*API* di *Amazon S3*⁷. È utilizzato principalmente come *server* di *storage* per dati non strutturati, come *file*, oggetti e immagini. Usato nel progetto per la gestione di cartelle condivise, utilizzate nell’applicazione per gestire una parte della persistenza.
- **Dashshim**⁸: *framework*, individuato dal sottoscritto, e poi utilizzato per mappare i *bucket*, contenitori utilizzati per memorizzare e organizzare og-

⁴ *Minikube*. URL: <https://minikube.sigs.k8s.io/>.

⁵ *Kubectl*. URL: <https://kubernetes.io/docs/reference/kubectl/>.

⁶ *Minio*. URL: <https://min.io/>.

⁷ *Amazon S3*. URL: <https://aws.amazon.com/it/pm/serv-s3/>.

⁸ *Dashshim*. URL: <https://github.com/dashshim-io/dashshim>.

getti in servizi di *storage*, con *MinIO* a dei *Persistent Volume Claim* utilizzabili successivamente dai *pod* dell'applicazione. I *Persistent Volume Claim* sono risorse *Kubernetes* che consentono alle applicazioni di richiedere e utilizzare spazio di archiviazione persistente all'interno di un *cluster*, rappresentato dai *Persistent Volumes*. Mentre i *pod* sono una raccolta di uno o più *container* che condividono lo stesso spazio di rete e le stesse risorse di *storage*. Rappresentano l'unità più piccola e basilare che può essere distribuita, replicata e gestita in un *cluster*.

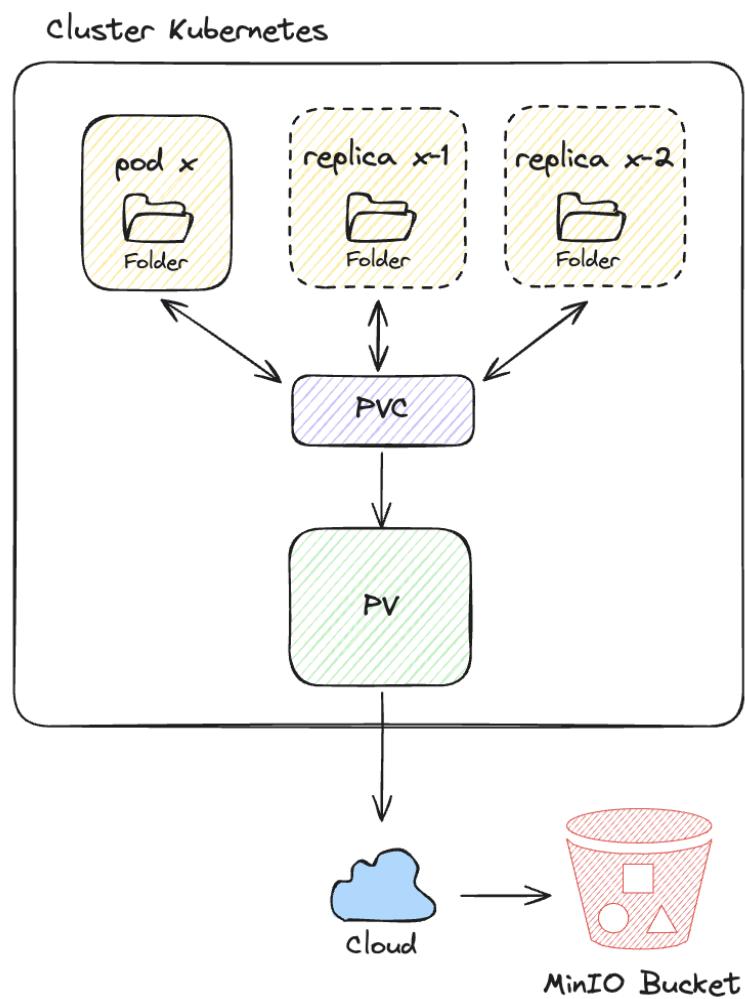


Figura 2.4: Astrazione infrastruttura per la gestione delle cartelle condivise.

Ed altre, per lo sviluppo di un applicazione di *test* a libera scelta. Per la quale, nello specifico, ho utilizzato:

- **React**⁹: una libreria *JavaScript* per costruire interfacce utente interattive e dinamiche, utilizzata per sviluppare il *front-end* dell'applicazione di *test*;
- **Express**¹⁰: *framework* minimalista per *Node.js*¹¹, usato per gestire le richieste *HTTP* e costruire l'*API* del *back-end* dell'applicazione di *test*;
- **Postgresql**: *DBMS* per *database* relazionali avanzato e *open-source*, utilizzato per memorizzare e gestire i dati dell'applicazione di *test*.

Per la stesura della documentazione invece, non avendo alcuna specifica, ho adotta principalmente il *Markdown*¹², un linguaggio leggero e semplice che si integra perfettamente con *Github*.

Piano di lavoro:

Il documento prevede poi l'organizzazione dei periodi di progetto in attività, che riporto per com'è stata pensata inizialmente:

- **Prima settimana - introduzione e studio:**
 - Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare;
 - Verifica credenziali e strumenti di lavoro assegnati;
 - Presa visione dell'infrastruttura esistente;
 - Formazione sulle tecnologie adottate.
- **Seconda settimana - analisi e comprensione:**
 - Individuazione dominio di applicazioni *stateful*;
 - Individuazione requisiti e vincoli in relazione alle suddette applicazioni.
- **Terza settimana - primo approccio e scalabilità verticale:**

⁹React. URL: <https://react.dev/>.

¹⁰Express. URL: <https://expressjs.com/>.

¹¹Node. URL: <https://nodejs.org/en>.

¹²Markdown. URL: <https://www.markdownguide.org/>.

- Primo approccio pratico al progetto;
- Creazione e clonazione di *pod* con persistenza dei dati;
- *Scheduling* e scalabilità verticale dei *pod* con persistenza dei dati;
- Sviluppo della prima versione dell'operatore.

- **Quarta settimana - *test* e documentazione:**

- *Test* sulla clonazione di *pod* con persistenza dei dati;
- *Test* sulla scalabilità verticale dei *pod* con persistenza dei dati;
- Stesura di documentazione relativa ai progressi ottenuti.

- **Quinta settimana - aggiornamenti a caldo con Blue/Green:**

- Implementazione dello "switch" tra *pod*, a caldo;
- Applicazione della strategia *blue/green*;
- Gestione della persistenza a caldo.

- **Sesta settimana - aggiornamenti tra versioni in conflitto:**

- Implementazione degli aggiornamenti, a caldo;
- Gestione delle differenze tra versioni delle immagini *Docker* contenute nei *pod*;
- Valutazione e implementazione di strategie per la gestione delle conversioni tra versioni differenti.

- **Settima settimana - applicazione di *test* con strategia *Canary*:**

- Applicazione della strategia *Canary* per gli aggiornamenti;
- Identificazione ed implementazione di *test* dei *pod* temporanei.

- **Ottava settimana - conclusione:**

- *Test* e verifica delle nuove funzionalità;
- Stesura di documentazione relativa ai progressi ottenuti.

CAPITOLO 2. LO STAGE

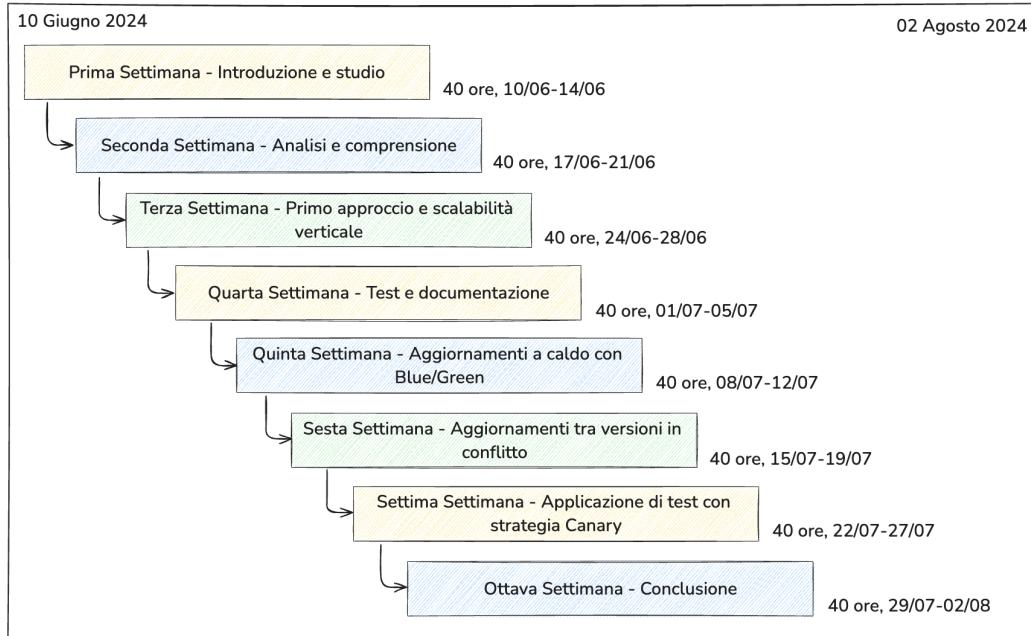


Figura 2.5: *Gantt view* del progetto.

Come già accennato, il piano di lavoro ha subito alcune modifiche durante lo *stage* adattandosi allo sviluppo del progetto. Nel complesso le attività delineate preventivamente non hanno comunque subito degli stravolgimenti. Riporto le settimane e relative attività che hanno subito dei cambiamenti:

- **Seconda settimana - analisi e comprensione:** alla quale è stata aggiunta l'attività: "Sviluppo di una basilare applicazione di *test*, che seguia l'architettura specificata dall'azienda.;"
- **Terza settimana - primo approccio e scalabilità orizzontale:** la cui attività: "*Scheduling* e scalabilità verticale dei *pod* con persistenza dei dati." è stata modificata in: "*Scheduling* e scalabilità orizzontale dei *pod* con persistenza dei dati.;"
- **Quarta settimana - test e documentazione:** la cui attività: "*Test* sulla scalabilità verticale dei *pod* con persistenza dei dati." è stata modificata in: "*Test* sulla scalabilità orizzontale dei *pod* con persistenza dei dati.". alla quale, inoltre, è stata aggiunta l'attività: "Realizzazione di

una presentazione, a scopo informativo, sul linguaggio di programmazione "Go", poi esposta in azienda;

- **Ottava settimana - conclusione:** alla quale è stata aggiunta l'attività: "Collaudo dell'operatore per la gestione di un'applicazione dell'azienda.".

Obiettivi:

Tali attività hanno portato all'individuazione di alcuni obiettivi, stilati con la collaborazione del *tutor* aziendale, e approvati dal mio *tutor* interno e relatore, che riporto:

Si farà riferimento agli obiettivi secondo le seguenti notazioni:

- **O** per gli obiettivi obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- **D** per gli obiettivi desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- **F** per gli obiettivi facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da un codice sequenziale di numeri, identificativo del obiettivo.

Codice	Descrizione	Tipologia
O-O1	Dominio e analisi dei requisiti di applicazioni <i>stateful</i> utili	Documentale
O-O2	Gestione dei <i>pod</i> con operazioni di base quali creazione e eliminazione	Funzionale
O-O3	Sviluppo di un operatore per <i>kubernetes</i> per automatizzare la gestione dei <i>pod</i>	Funzionale
O-O4	Scalabilità verticale e <i>scheduling</i> dei <i>pod</i>	Funzionale
Continua nella prossima pagina...		

Tabella 2.1 – Continuo della tabella

Codice	Descrizione	Tipologia
O-O5	Aggiornamento a caldo con strategia <i>Blue/Green</i>	Funzionale
O-O6	Gestione di versioni incompatibili con attenzione alla persistenza dei dati	Funzionale
O-O7	Applicazione della strategia <i>Canary</i>	Funzionale
O-O8	Ideazione e implementazione di <i>test</i> su nuovi <i>pod</i> temporanei	Funzionale
O-O9	Documentazione del codice e dei progressi ottenuti	Documentale
O-D1	<i>Test</i> su <i>pod</i> e immagini con architetture differenti	Funzionale

Tabella 2.1: Tabella riassuntiva degli obiettivi individuati durante la stesura del piano di lavoro iniziale.

Gli obiettivi individuati preventivamente durante la stesura del piano di lavoro, sono stati sottoposti durante lo svolgimento del progetto, ad un adattamento e raffinamento, in risposta al cambiamento di alcune attività.

Nello specifico:

- **O-O4** è stato modificato, la nuova descrizione è divenuta: "Scalabilità orizzontale e *scheduling* dei *pod*.";
- **O-O10** nuovo obiettivo funzionale: "Sviluppo di un'applicazione di *test*.";
- **O-O11** nuovo obiettivo funzionale: "Gestione della persistenza dei dati, tramite cartelle condivise.>";
- **O-O12** nuovo obiettivo funzionale: "Gestione della persistenza dei dati, tramite *database*.";
- **O-O6** è stato suddiviso in:

- **O-O6A:** "Gestione di versioni incompatibili con attenzione alla persistenza dei dati - Cartelle condivise.";
- **O-O6B:** "Gestione di versioni incompatibili con attenzione alla persistenza dei dati - *Database*.".

Prodotti attesi:

Infine il documento elenca i prodotti attesi, riassumibili in:

- Documentazione relativa al codice sviluppato complessivamente;
- Documentazione relativa allo studio iniziale delle tecnologie;
- *Environment Kubernetes* per il *testing* dell'operatore;
- Operatore realizzato in *Go*, munito di tutte le *feature* previste.

Alcuni di essi sono stati rivisti durante il corso dello *stage*. Sono state quindi apportate le seguenti modifiche:

- Aggiunto: "Presentazione a scopo informativo sul linguaggio di programmazione *Go*".

2.5 Motivazione della scelta

Durante *Stage-IT*, evento organizzato dall'università che offre a noi studenti l'opportunità di incontrare e conoscere diverse aziende, tra le quali Zucchetti, ho avuto la possibilità di interagire direttamente con alcuni rappresentanti dell'azienda e di approfondire le loro proposte di *stage*. In particolare, Zucchetti presentava tre progetti distinti: il primo focalizzato sull'intelligenza artificiale, il secondo dedicato allo studio di tecnologie per la creazione di infografiche e il terzo, che ho scelto, relativo al mondo del *CloudNative*.

La prima cosa che mi colpì positivamente fu l'approccio dell'azienda nel presentare i progetti come linee guida piuttosto che come vincoli rigidi. Zucchetti ha incentivato fin da subito la proposta di idee innovative e la personalizzazione dei progetti, incoraggiando noi candidati a esplorare soluzioni originali e a contribuire con il nostro punto di vista. Questo approccio aperto e flessibile ha

reso l'opportunità di *stage* ancora più stimolante e allettante.

Inizialmente, ero maggiormente attratto dai progetti che trattavano l'intelligenza artificiale, tema molto discusso e ricorrente tra le varie aziende, i quali però tendevano a presentare molte somiglianze ed a risultare quindi, perlomeno per il sottoscritto, meno interessanti. Ciò unito alla mia curiosità per un ambito a me sconosciuto e meno esplorato di altri in campo universitario, anche in ottica magistrale, unita all'opportunità di confrontarmi con una tecnologia emergente e molto richiesta come quella del *cloud native*, mi ha spinto a scegliere il progetto relativo a *Kubernetes*. Questa scelta, a mio parere, si è rivelata estremamente gratificante. Oltre infatti ad avere appreso molto dal punto di vista tecnologico, ho avuto anche il modo di approcciarmi ad un ambiente lavorativo, che ho trovato, professionale, accogliente e soprattutto stimolante a contatto con persone disponibili, interessate, e di istruttiva esperienza, in un'azienda grande e rinomata come Zucchetti.

Ho sempre pensato allo *stage* come ad un'opportunità di crescita sia professionale che personale di fondamentale importanza. Ha rappresentato il mio primo approccio al mondo del lavoro in un contesto aziendale, un ambiente che si discosta notevolmente dalla realtà accademica alla quale noi studenti siamo abituati. Sebbene non avessi la certezza esatta di cosa avrei affrontato, il mio obiettivo principale era massimizzare l'apprendimento e trarre il massimo vantaggio da questa esperienza.



Figura 2.6: Crescita professionale.

Fonte: <https://it.vecteezy.com>

Obiettivi professionali:

- **Acquisizione di competenze tecniche:** uno dei miei principali obiettivi era acquisire competenze pratiche in ambiti tecnologici di mio interesse e soprattutto, a me sconosciuti, così da ampliare il portfolio delle mie conoscenze;
- **Esperienza pratica:** essendo la prima volta che lavoravo in un contesto aziendale, volevo familiarizzare con le dinamiche di lavoro di questo ambiente. L'obiettivo era comprendere come le teorie apprese durante il percorso accademico si traducono in attività quotidiane e sfide concrete all'interno di un'azienda;
- **Conoscenza delle pratiche aziendali:** ero inoltre curioso di apprendere come un'azienda gestisce i suoi progetti, coordina le attività tra i membri del *team*, e implementa soluzioni tecnologiche.

Obiettivi personali:

- **Sviluppo delle competenze trasversali:** oltre alle competenze tecniche, puntavo a migliorare abilità trasversali come la comunicazione, il *problem-solving* e la gestione del tempo. Penso che questi aspetti che ho avuto modo di scoprire appena, in ambito universitario, principalmente tramite il corso di Ingegneria del *Software*, siano essenziali nel mondo del lavoro;
- **Adattamento a nuovi ambienti:** volevo verificare la mia capacità di adattarmi a un nuovo ambiente lavorativo e alle sue dinamiche. Questo includeva l'integrazione in gruppo già formato, l'approccio alle sfide e la capacità di lavorare sotto scadenze e direttive;
- **Crescita professionale:** inoltre volevo sfruttare l'ambiente di lavoro e l'opportunità di interagire con professionisti esperti per ricevere *feedback* costruttivi, consigli preziosi e ispirazioni che potessero orientare e arricchire la mia futura carriera.

Sono convinto che lo *stage* abbia rappresentato una fase cruciale per fare il ponte tra la formazione teorica e l'applicazione pratica, e sono grato per l'opportunità di averlo potuto vivere in un contesto così stimolante e innovativo.

Capitolo 3

Il progetto

3.1 Analisi dei requisiti

L’analisi dei requisiti ha costituito una fase cruciale del progetto di *stage*, seguita dalla fase di formazione, che, come possibile immaginare, ha previsto la visione e studio di varia documentazione, in combinazione con la visione di video corsi, e *tutorial*.

Questa attività ha coinvolto una valutazione approfondita delle necessità e delle aspettative per due componenti principali del progetto: l’applicazione di *test* e l’operatore *Kubernetes*. Ogni componente è stato analizzato separatamente per garantire che tutte le specifiche e le esigenze venissero adeguatamente soddisfatte.

Per quanto riguarda l’applicazione di *test*, l’analisi dei requisiti ha avuto un carattere meno complesso rispetto all’operatore *Kubernetes*, ma comunque fondamentale per garantire che il progetto di *test* fosse funzionale e rispondesse agli obiettivi prefissati. La fase di analisi ha incluso:

- **Definizione degli obiettivi:** identificare le funzionalità e le caratteristiche che l’applicazione di *test* doveva possedere;
- **Identificazione dei requisiti:** raccolta di requisiti specifici relativi alle funzionalità dell’applicazione e le tecnologie da utilizzare.

L'operatore *Kubernetes* ha rappresentato una parte più complessa e dettagliata del progetto, data la sua importanza centrale nella gestione delle applicazioni persistenti. La fase di analisi dei requisiti relativa ha comportato:

- **Definizione dei requisiti funzionali:** comprendere e documentare le funzionalità richieste per l'operatore, inclusa la gestione automatizzata delle applicazioni persistenti, la scalabilità, la gestione degli aggiornamenti e la persistenza dei dati.
- **Analisi delle esigenze di configurazione:** identificare le configurazioni necessarie per l'integrazione dell'operatore con *Kubernetes*, come i requisiti per i *Custom Resource Definitions (CRD)*, i *Persistent Volume Claims (PVC)* e le configurazioni di *storage*.
- **Studio delle tecnologie:** valutazione delle tecnologie e degli strumenti necessari per sviluppare l'operatore, come il *Kubernetes Operator SDK, Go* (Golang) per lo *scripting*, e altre tecnologie di supporto.
- **Valutazione dei requisiti di sicurezza e prestazioni:** assicurarsi che l'operatore soddisfacesse gli *standard* di sicurezza e di prestazioni necessari per garantire l'affidabilità e la sicurezza dei dati e delle applicazioni gestite previste dall'azienda.

3.1.1 Scenari e casi d'uso

In questa sezione, presento alcuni esempi di scenari e casi d'uso principali che ritengo siano tra i più significativi e interessanti, e che penso riescano nel riassumere efficacemente le attività svolte e le sfide affrontate durante lo *stage*.

L'applicazione di *test*, in quanto tale, non ha richiesto lo sviluppo di un ampio numero di funzionalità. Tuttavia, sono state identificate ed analizzate con particolare attenzione alcune caratteristiche fondamentali che l'applicazione doveva necessariamente possedere in base alle richieste dell'azienda.

Come ad esempio, la funzionalità di *login*, essenziale per qualsiasi applicazione che gestisca utenti, è stata cruciale non solo per la sua importanza intrinseca

ma anche come *test* dell'integrazione tra applicazione e *database*.

Attori:

- **Utente:** rappresenta l'utente fruitore dell'applicazione in uno stato specifico, quello di: autenticato. Un'utente è autenticato se ha effettuato l'operazione di *login* con successo.

Visualizzazione e uso della cartella condivisa:

L'applicazione doveva inoltre fornire, su richiesta dell'azienda, l'accesso e gestione ad una cartella condivisa tra tutti gli utenti per la persistenza di alcuni dati specifici.

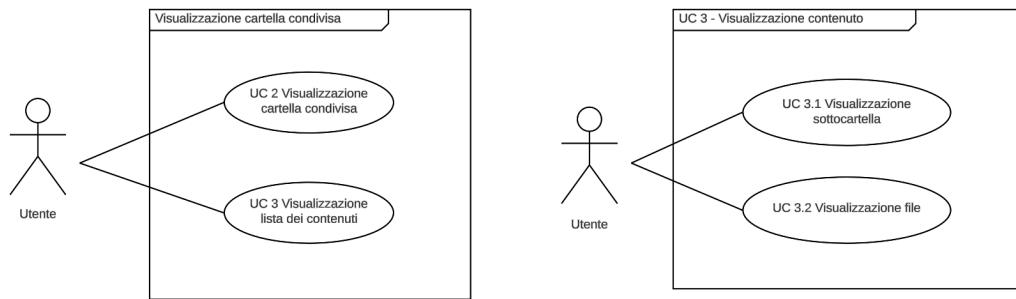


Figura 3.1: Casi d'uso visualizzazione della cartella condivisa.

- **Attori:** Utente.

- **Precondizioni:**

- L'utente è autenticato;
- L'utente è nella pagina di visualizzazione della cartella condivisa.

- **Postcondizioni:**

- L'utente visualizza i contenuti della cartella condivisa;
- L'utente visualizza un elemento della cartella:
 - * L'utente visualizza una sottocartella (*UC 3.1*);
 - * L'utente visualizza un *file* (*UC 3.2*).

- **Scenario principale:** L'Utente autenticato è attualmente nella pagina adibita alla visualizzazione della cartella condivisa. L'utente visualizza i contenuti della cartella. L'utente visualizza un contenuto della cartella, sottocartella (*UC 3.1*) o *file* (*UC 3.2*) che sia. Il Sistema mostra all'utente i contenuti previsti.

Ogni utente, poi, doveva poter gestire i contenuti della cartella, eseguendo in pratica due operazioni: il caricamento di un nuovo contenuto e la rimozione di un qualsiasi contenuto, ad eccezione della cartella radice stessa.

L'operatore, d'altra parte, per com'è stato pensato dall'azienda, ha previsto un insieme più ampio di funzionalità, con anche requisiti specifici in termini di sicurezza e prestazioni. I casi d'uso che seguono rappresentano un'approssimazione che vuole riassumere quanto analizzato durante il tirocinio.

Attori:

- **Operatore:** Rappresenta l'operatore, che in modo autonomo interagisce con l'ambiente *Kubernetes* (il Sistema). Come infatti spiegherò successivamente, una volta configurato, l'operatore adopera automaticamente senza praticamente nessun altro intervento "umano".

Scalabilità

L'operatore doveva poter scalare, orizzontalmente, l'applicazione in base ad alcune metriche quali: percentuale di *CPU* e memoria utilizzata ed il numero di connessioni. Ciò ha reso necessario il monitoraggio di tali metriche, a carico dell'operatore stesso. L'operatore doveva poi interpretare queste metriche, utilizzando dei valori di soglia per decidere come scalare l'applicazione: "verso l'alto," aggiungendo una replica, o "verso il basso", eliminandone una.

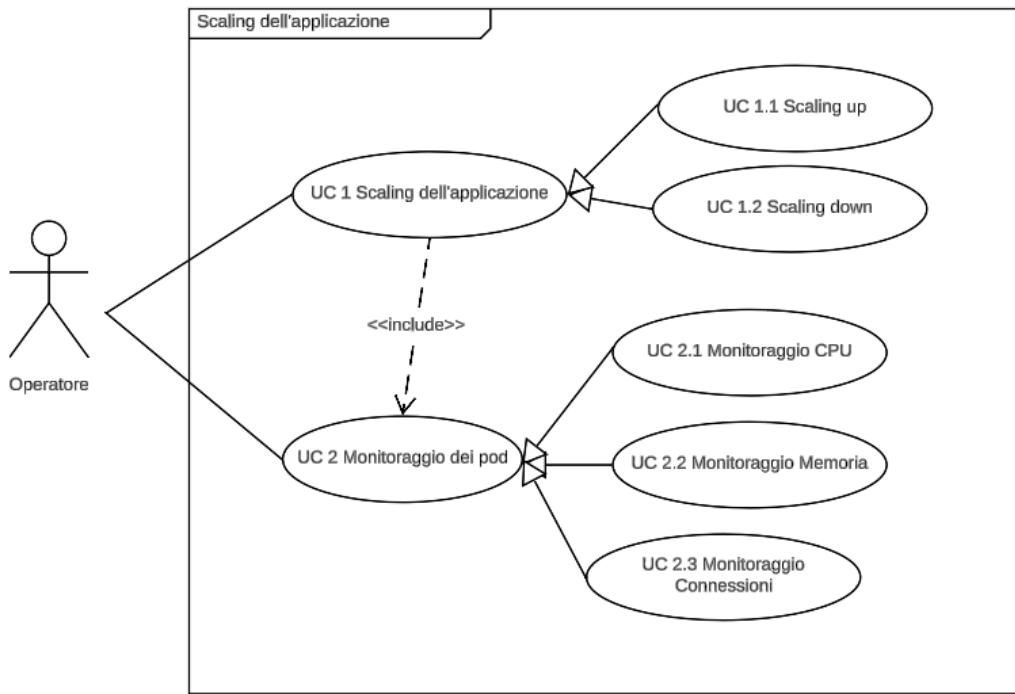


Figura 3.2: Casi d'uso scalabilità dell'applicazione.

- **Attori:** Operatore.
- **Precondizioni:**
 - Il sistema è correttamente configurato;
 - L'operatore sta monitorando il sistema (*UC 2*):
 - * L'operatore sta monitorando la *CPU* dei *pod* (*UC 2.1*);
 - * L'operatore sta monitorando la memoria dei *pod* (*UC 2.2*);
 - * L'operatore sta monitorando le connessioni ai *pod* (*UC 2.3*).
 - L'operatore ha richiesto lo *scaling* dei *pod* (*UC 1*):
 - * L'operatore ha richiesto la creazione di una nuova replica (*UC 1.1*);
 - * L'operatore ha richiesto l'eliminazione di una vecchia replica (*UC 1.2*).
- **Postcondizioni:**
 - Il Sistema ha eseguito la richiesta dell'operatore.

- **Scenario principale:** L'Operatore sta monitorando i *pod* dell'applicazione (*UC 2*). In base alle varie metriche: uso di *CPU*, memoria e numero di connessioni attive, l'Operatore scala l'applicazione richiedendo la creazione di una nuova replica (*UC 1.1*), oppure richiedendo l'eliminazione di una già presente, vecchia (*UC 1.2*). Il Sistema esegue le richieste dell'operatore, aggiornandosi.

Scheduling

L'operatore doveva essere in grado di pianificare la scalabilità dei *pod* dell'applicazione in base all'orario corrente e a una configurazione di intervalli temporali, ciascuno associato a un numero massimo di repliche. Ad esempio, tra le 00:00 e le 06:00, il numero massimo di repliche sarà limitato a 2, sulla base dell'ipotesi che durante questo intervallo il traffico verso l'applicazione risulti generalmente basso.

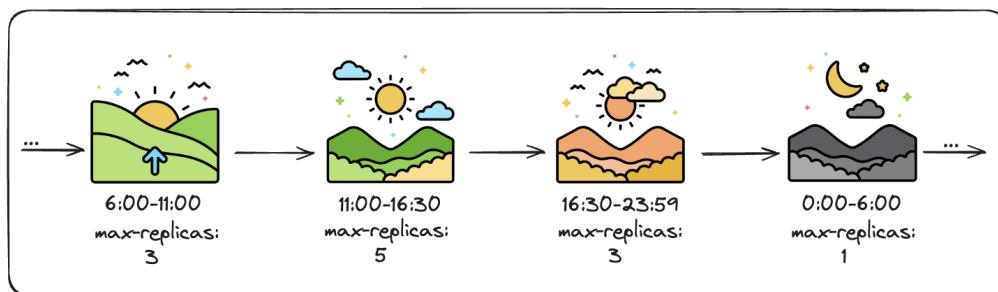


Figura 3.3: Astrazione schedulazione dell'applicazione.

Gestione degli aggiornamenti

L'operatore doveva poter gestire gli aggiornamenti di versione dell'applicazione in esecuzione all'interno del *cluster Kubernetes*. L'azienda aveva già individuato alcune tecniche da utilizzare per valutarne le potenzialità: *Blue-Green* e *Canary*.

La *Blue-Green Deployment* è una strategia di distribuzione che prevede l'uso di due ambienti di produzione separati, chiamati "*Blue*" e "*Green*", per garantire un aggiornamento fluido e senza interruzioni delle applicazioni.

Mentre la *Canary Deployment* è una strategia di distribuzione che prevede l'in-

troduzione graduale di una nuova versione dell'applicazione, rilasciandola inizialmente a una piccola parte degli utenti per monitorare il comportamento prima di un'implementazione completa.

Spesso, queste due strategie vengono combinate per ottenere i benefici di entrambe e mitigare i loro rispettivi svantaggi. La loro combinazione permette di ottenere un rilascio sicuro e fluido con un controllo dettagliato e una gestione dei rischi migliorata. Ecco perché, insieme al mio *tutor*, abbiamo deciso di adottare questa combinazione. Il processo di gestione degli aggiornamenti è stato quindi diviso in 2 scenari principali, i quali condividono il comportamento iniziale.

Aggiornamento di versione - IT1

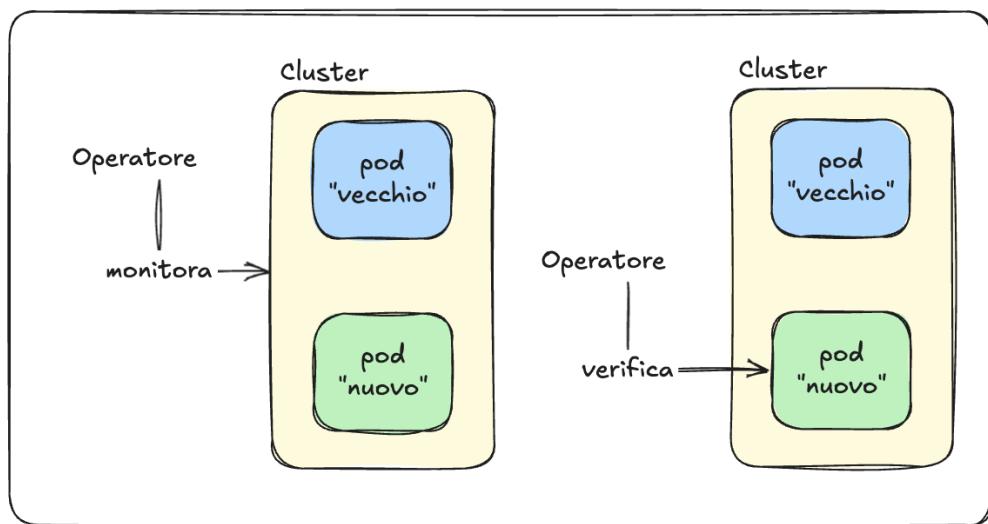


Figura 3.4: Astrazione processo di gestione degli aggiornamenti, iterazione 1.

Com'è possibile vedere in Figura 3.4, all'iterazione 1 del processo, l'operatore si occuperà di monitorare il *cluster* e di verificare il "nuovo" *pod*, rappresentante la nuova versione dell'applicazione. Il processo di verifica prevede poi, alcuni *test* riconducibili alla strategia *Canary*. In questo momento entrambe le applicazioni dovranno essere raggiungibili, anche se, quella "nuova", solo da un bacino di utenti controllato.

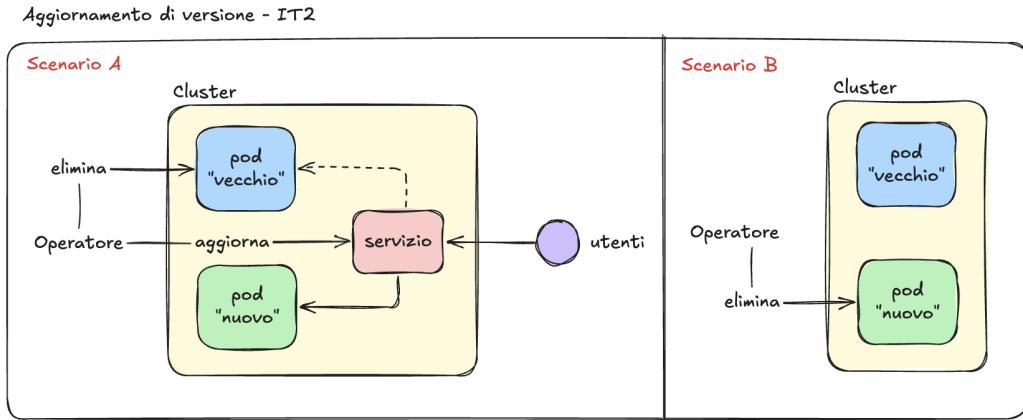


Figura 3.5: Astrazione processo di gestione degli aggiornamenti, iterazione 2.

Vi sarà poi una biforcazione. Com'è possibile vedere nell'immagine 3.5, l'operatore assumerà, a seconda dell'esito dei *test*, comportamenti differenti. Nello scenario "A", ovvero se tutti i *test* saranno passati, l'operatore procederà eliminando il "vecchio" *environment* quando tutte le sessioni su esso presenti saranno terminate, e reindirizzando gli utenti a quello "nuovo", ormai stabile, così da non avere interruzioni del servizio. Nello scenario "B", ovvero se almeno uno dei *test* non dovesse passare, l'operatore procederà eliminando il "nuovo" ambiente comunicando verso l'esterno l'esito dell'operazione.

3.1.2 Requisiti

Definiti scenari e casi d'uso ho proceduto, delineando i requisiti del progetto suddividendoli in categoria, e priorità. Ogni requisito doveva essere identificabile e tracciabile. Ad ognuno di essi è stato quindi affibbiato un codice riconoscitivo, così composto:

$$R[\text{Categoria}][\text{Priorità}]-[\text{Indice}]$$

dove:

- **Categoria** può assumere valori:
 - **F**: requisito funzionale;
 - **Q**: requisito qualitativo (e/o prestazionale);

- **V**: requisito di vincolo (relativo ad esempio alle tecnologie da utilizzare);
- **D**: requisito documentale.

- **Priorità** può assumere valore:

- **O**: requisito obbligatorio;
- **D**: requisito desiderabile;
- **OP**: requisito opzionale.

- **Indice** è un numero incrementale che identifica il requisito.

Come per quanto fatto per i casi d'uso, conseguentemente, anche in questo caso ho preferito suddividere i requisiti dell'applicazione di *test* e quelli dell'operatore in quanto due elementi separati.

Per quanto riguarda l'applicazione, mi sono limitato a definire i requisiti funzionali obbligatori derivati dalle richieste espresse dall'azienda.

Per quanto riguarda i requisiti dell'operatore invece, ho inizialmente definito i requisiti funzionali, suddividendoli in categorie specifiche. Durante lo sviluppo, ho poi esteso questa definizione per includere requisiti qualitativi e vincoli, che erano strettamente legati alle funzionalità implementate progressivamente. Questo approccio si è reso necessario perché lo sviluppo e l'osservazione dell'ambiente *Kubernetes* hanno rivelato aspetti secondari che, insieme all'azienda, non avevamo inizialmente considerato o ai quali non avevamo attribuito la giusta importanza.

Molti di questi aspetti riguardano le prestazioni dell'operatore e del *cluster Kubernetes* nel suo complesso. Ad esempio, sono stati presi in considerazione: il tempo di risposta di determinate richieste, la sicurezza delle operazioni, con l'implementazione di strategie di crittografia per proteggere alcuni dati di configurazione, e la persistenza e il tempo di aggiornamento della cartella condivisa. Altri requisiti funzionali poi, sono stati stabiliti per identificare le operazioni di interazione tra l'operatore e le risorse del *cluster*. Le risorse sono oggetti che rappresentano vari aspetti della gestione e del funzionamento delle applicazioni e

CAPITOLO 3. IL PROGETTO

dei servizi all'interno di un *cluster*, dei quali illustrerò alcuni esempi, successivamente. Infine, i requisiti documentali sono stati stilati per il tracciamento delle attività relative alla documentazione. Riporto qui sotto una tabella riassuntiva dei requisiti individuati complessivamente:

	Obbligatori	Desiderabili	Opzionali	Totali
Funzionali	30	2	x	32
Qualitativi	4	2	x	6
Di vincolo	6	2	x	8
Documentali	4	1	x	5
Totali	44	7	x	51

Tabella 3.1: Tabella riassuntiva dei requisiti individuati durante il corso del progetto.

3.2 Progettazione

L'attività di progettazione ha susseguito quelle di studio delle tecnologie e di analisi dei requisiti. Come quest'ultima, anch'essa è stata suddivisa in due sezioni: rispettivamente, la prima per la progettazione dell'applicazione di *test*, mentre la seconda per l'operatore e il *cluster Kubernetes*. Per quanto riguarda l'applicazione mi sono limitato a seguire le direttive dell'azienda, la quale aveva espresso delle caratteristiche principali che l'applicazione doveva possedere.

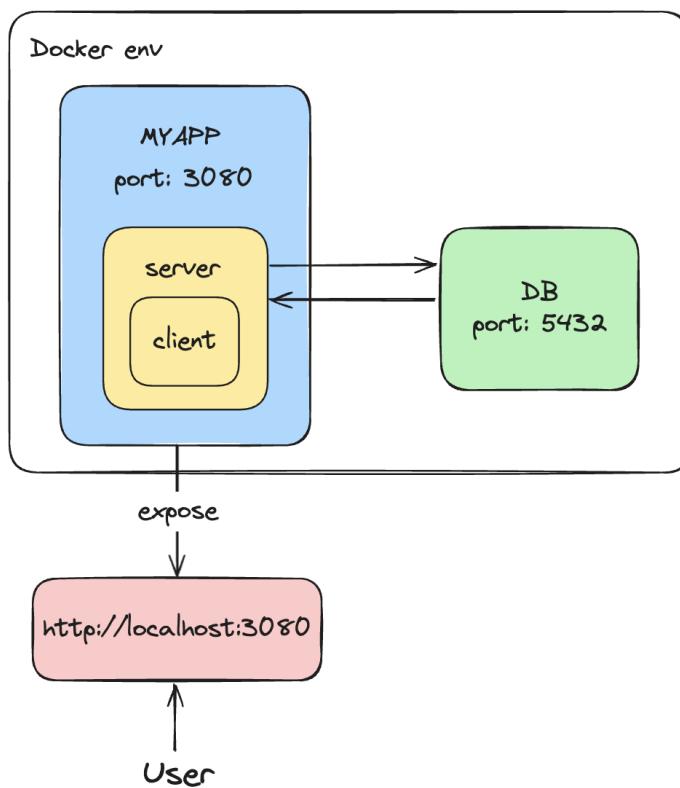


Figura 3.6: Astrazione architettura dell'applicazione di *test*.

Innanzitutto, l'applicazione doveva essere realizzata con il fine di containerizzarla utilizzando *Docker/Podman*. La struttura dell'applicazione, poi, doveva soddisfare i seguenti requisiti fondamentali:

- **DB:** l'applicazione doveva possedere un *database* per la gestione dei dati e la loro persistenza;

- **Server:** L'applicazione doveva possedere un *server* che fungesse da nodo centrale per la gestione delle richieste degli utenti. Inoltre esso doveva disporre di una cartella che sarebbe poi stata condivisa da tutti gli utenti
- **Client:** Il *client* doveva essere integrato nel *server* e rappresentare l'unico punto di accesso per gli utenti. Questo componente è responsabile dell'interfaccia utente e delle interazioni con il *server*.

Per quanto invece concerne l'operatore, l'attività di progettazione è stata largamente più complessa. Non tanto per il *design* del codice dell'operatore stesso, il quale non segue particolari *pattern*, e per il quale, la già citata *Operator SDK* fornisce numerose agevolazioni. Ma più per la progettazione e configurazione del *cluster Kubernetes*. Come ho già accennato, un *cluster*, contiene delle risorse. Queste vengono configurate, solitamente una sola volta, manualmente. Esse rappresentano vari oggetti, e ve ne sono di tipologie diverse, come ad esempio:

- **Deployment:** è una risorsa che gestisce il ciclo di vita dei *pod*, assicurandosi che un numero specifico di repliche di un'applicazione sia sempre in esecuzione. I *Deployment* permettono di dichiarare lo stato desiderato delle applicazioni e di gestire l'aggiornamento e il *rollback*, ripristino, dei *pod*;
- **Service:** risorsa che fornisce un'astrazione per l'accesso ai *pod* di un'applicazione. I *Service* stabiliscono un punto di accesso stabile per i *pod* e possono esporre i servizi su una porta specifica;
- **Ingress:** è una risorsa che gestisce l'accesso esterno ai servizi all'interno del *cluster*. Si occupa di instradare le richieste *HTTP* e *HTTPS* ai servizi appropriati basandosi su regole di *routing* definite.

Una volta compreso cosa sono le risorse e capite le loro peculiarità, mi sono cimentato nella progettazione di un'ambiente che potesse fare fronte alle richieste dell'azienda riguardanti le funzionalità dell'operatore.

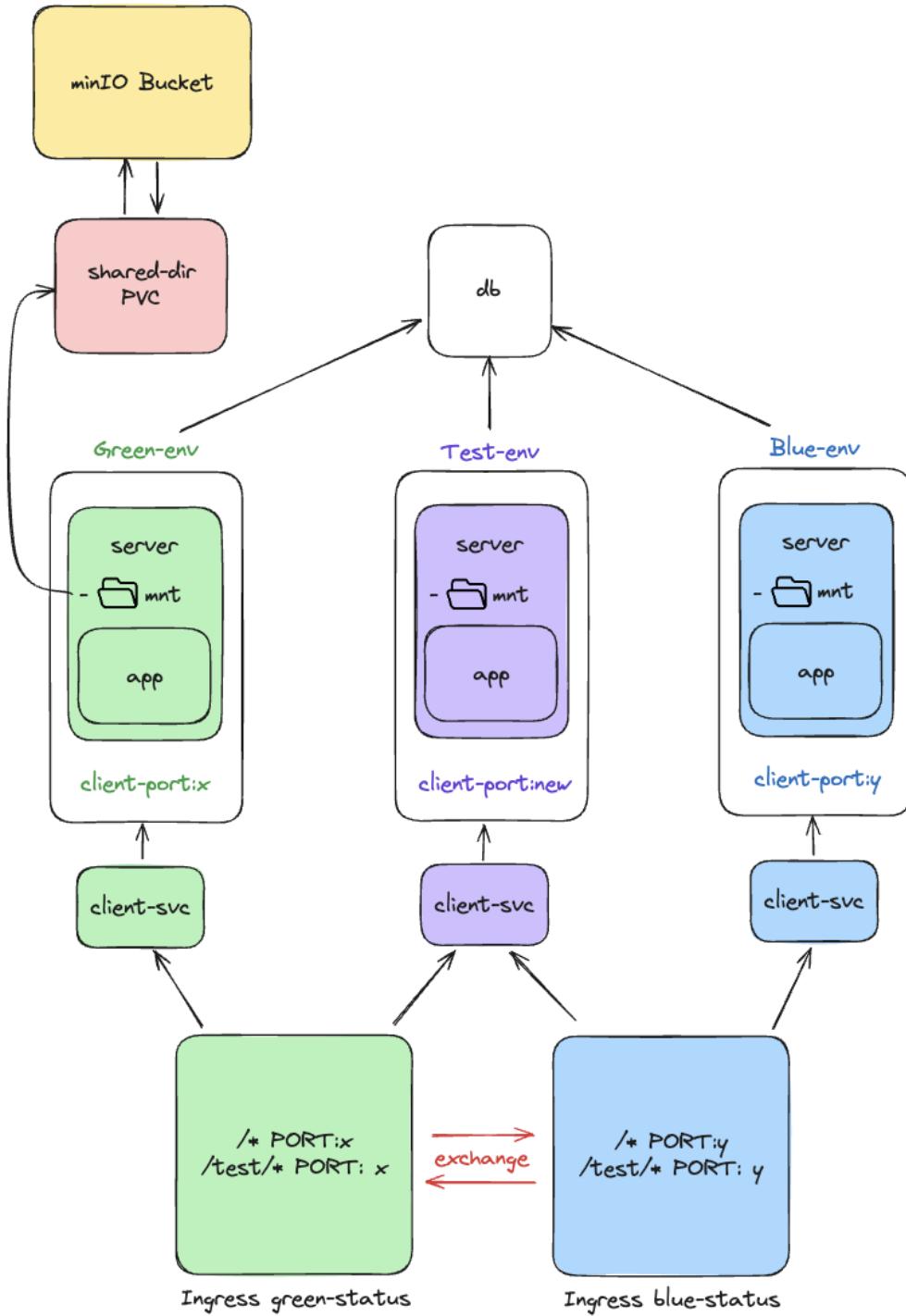


Figura 3.7: Astrazione architettura del *cluster Kubernetes*.

Com’è possibile intuire dall’immagine 3.7, la maggiore difficoltà è stata quella di riuscire ad ideare un’ambiente, che potesse permettere l’adozione delle strategie *Blue-Green* e *Canary* presentate in precedenza. Allo scopo, sono stati previsti tre *environment* differenti. L’idea di base era quella di avere un’am-

biente isolato, ed equivalente, per ogni "versione" dell'applicazione. L'ambiente, doveva essere progettato in funzione dell'operatore e viceversa. Mi è stato quindi utile immaginare come l'operatore avrebbe poi eseguito determinate operazioni.

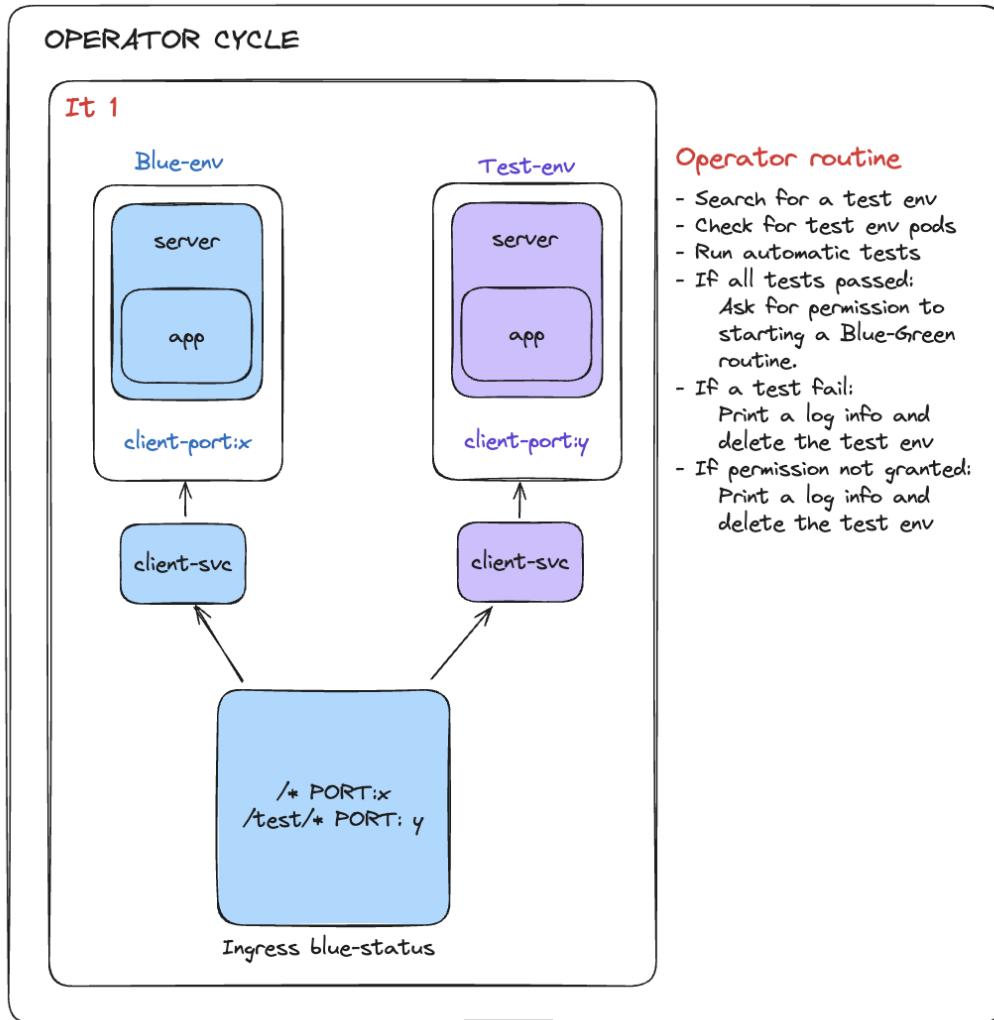


Figura 3.8: Astrazione processo di aggiornamento, iterazione 1.

Nell'immagine sopra (Figura 3.8) è possibile vedere la prima iterazione del processo di aggiornamento di versione dell'applicazione, e la relative *routine* dell'operatore. In questo istante, ci sono solo due *environment*: quello "*blue*" e quello di *test*. Quest'ultimo dovrà contenere ed eseguire la nuova versione dell'applicazione, che dopo la fase di *test*, verrà ospitata dall'ambiente "*green*". Gli utenti verranno indirizzati solo all'applicazione "*blue*", mentre chi autorizzato potrà accedere a quella di *test*, che dovrà quindi essere raggiungibile.

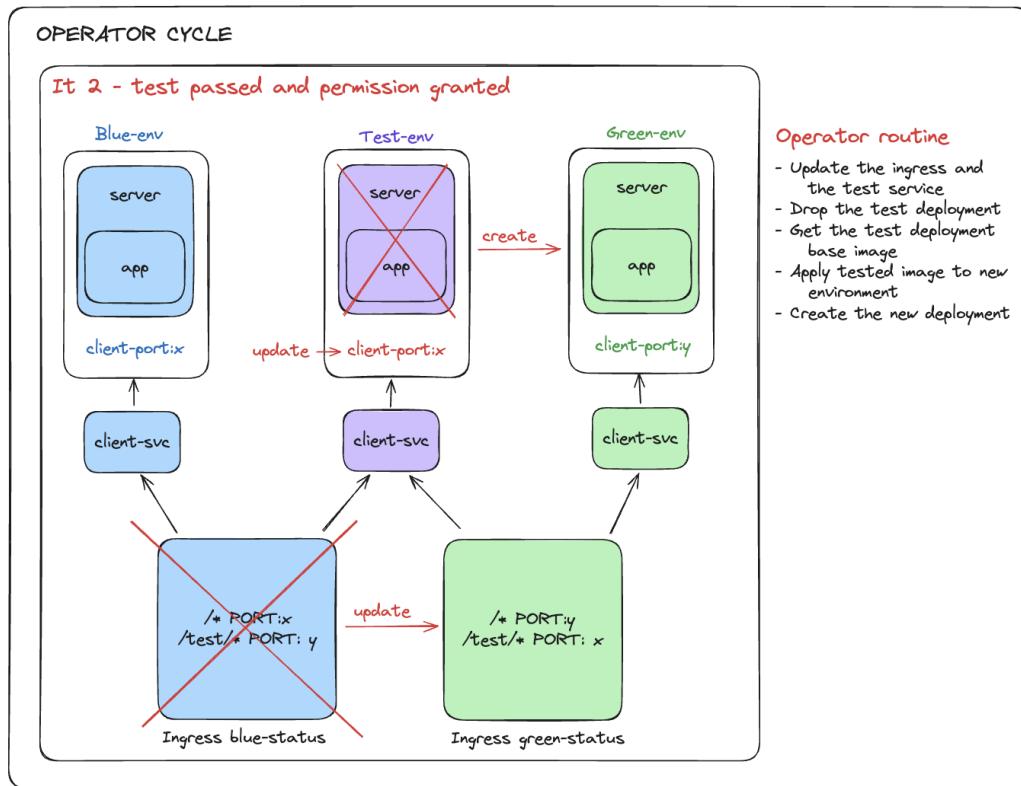


Figura 3.9: Astrazione processo di aggiornamento, iterazione 2.

All’iterazione successiva, che rappresenta lo scenario nel quale i *test* del nuovo ambiente sono stati superati, l’operatore dovrà creare un nuovo *environment* "green", eliminare quello di *test*, ed instradare correttamente l’utenza. Gli utenti che si connetteranno al servizio da questo momento in poi, dovranno essere indirizzati al "nuovo" ambiente. Mentre quelli che erano attivi su quello ormai "vecchio", dovranno poter continuare ad usufruirne. Solamente quando tutte le sessioni presenti sul "vecchio" ambiente saranno terminate, l’operatore potrà procedere alla sua eliminazione.

Infine la progettazione ha inoltre compreso la definizione dei *manifests* di configurazione delle risorse *Kubernetes*. Essi sono *file* di configurazione scritti in *YAML*¹, linguaggio di serializzazione di dati, (o *JSON*) che definiscono le risorse che *Kubernetes* deve creare e gestire.

¹ *YAML*. URL: <https://yaml.org/>.

3.3 Codifica

L'attività di codifica, strettamente interconnessa a quella di progettazione, è stata articolata in diverse fasi. Inizialmente, mi sono concentrato sulla realizzazione dell'applicazione di *test*, in quanto era il componente più semplice e rapido da programmare oltre ad essere necessario per verificare e validare il corretto funzionamento dell'operatore che avrei successivamente implementato. Sebbene l'applicazione di *test* abbia rappresentato una parte fondamentale del progetto, ho deciso di non approfondirne ulteriormente i dettagli, in quanto ritengo che l'operatore ne rappresenti l'aspetto più interessante e innovativo.

Una volta completata l'applicazione di *test*, mi sono dedicato alla scrittura dei *manifests* delle risorse *Kubernetes*, come *Deployment*, *Service*, e *Ingress*. Questa fase è stata cruciale per configurare e verificare l'ambiente *Kubernetes* in pratica, consentendomi di acquisire una comprensione più profonda di come orchestrare e gestire le risorse all'interno del *cluster*. Ne segue un esempio semplificato:

```
# Deployment of the client

apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-blue # This format is required for the operator
  ↳ to work properly (name-env)
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels: # These labels are required for the operator to
    ↳ work properly
    app: client
    environment: blue
    version: v1.0.0
  #...
  spec:
    #...
    volumeMounts:
      - name: shared-folder
        mountPath: /express/mnt
    #...
    volumes:
      - name: shared-folder
    persistentVolumeClaim:
      claimName: shared-folder
```

Codice 3.1: Sezione di codice relativo al *manifest* dell'applicazione.

Com'è possibile notare, ho voluto includere le parti che riprendono concetti già illustrati come: *Volumes*, *PVCS* e *Deployments*, e che quindi possano risul-

tare più comprensibili e significative.

Solo successivamente ho potuto dedicarmi alla codifica dell'operatore vero e proprio. Come menzionato in precedenza, l'operatore è stato sviluppato utilizzando il linguaggio *Go* e l'*Operator SDK*, strumenti che hanno richiesto una solida fase di formazione preliminare.

Durante la sua implementazione, ho dovuto familiarizzare con alcuni paradigmi peculiari di *Go*, che si discostano significativamente dai linguaggi di programmazione con cui avevo già esperienza. Sebbene *Go* sia sintatticamente semplice e relativamente facile da scrivere, presenta delle scelte progettuali uniche che ne esaltano la velocità di compilazione e l'efficienza operativa, ma al tempo stesso eliminano intenzionalmente molte delle "comodità" offerte da altri linguaggi, come la programmazione orientata agli oggetti (*OOP*)² tradizionale. Nonostante infatti sia nato dopo l'avvento dell'*OOP*, *Go* non supporta direttamente la creazione di oggetti come avviene in altri linguaggi orientati agli oggetti, preferendo invece un approccio più minimalista e orientato alla composizione.



Figura 3.10: Fondatori di *Go*, da sinistra a destra: *Robert Griesemer*, *Rob Pike* e *Ken Thompson*.

Fonte: <https://www.jesuisundev.com>

²Object Oriented Programming (*OOP*). URL: https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti.

Forse il più importante di tutti, sono le *Goroutine*, un elemento chiave di *Go* che consente di gestire la concorrenza in modo estremamente efficiente. Le *Goroutine* permettono di eseguire funzioni in modo asincrono e parallelo, con un *overhead* di risorse molto ridotto rispetto ai *thread* tradizionali. Nel contesto dello sviluppo dell'operatore, sono state fondamentali per gestire operazioni come il monitoraggio continuo delle risorse *Kubernetes* e l'applicazione delle modifiche necessarie in modo tempestivo. Ad esempio, l'operatore doveva essere in grado di ascoltare eventi specifici all'interno del *cluster*, come la creazione, l'aggiornamento o l'eliminazione di risorse, e reagire in modo appropriato, spesso eseguendo più operazioni simultaneamente. Come già introdotto in precedenza gli operatori non sono altro che programmi ciclici che periodicamente eseguono operazioni sul *cluster*. Il corrispettivo di un *main* di un programma la cui unica funzione è quella di stampare la scritta "*Hello Word!*", negli operatori non è altro che una funzione detta: "*Reconcile loop*", che stampa la stessa scritta ma ciclicamente. Questa funzione rappresenta il cuore dell'operatore, ed esso si sviluppa a partire da essa. Altre funzioni, con responsabilità specifiche, vengono tutte richiamate all'interno di questo *loop*, ed eseguite a seconda della loro definizione, in *background*, utilizzando le *Goroutine* o sequenzialmente.

Le più complesse da realizzare sono state quelle che implementavano le varie strategie di gestione degli aggiornamenti dell'applicazione. Ne riporto qui sotto una semplificazione:

```
func (r *ScalerReconciler) canaryRoutine(ctx context.Context,
    deployment apiv1alpha1.DeploymentInfos, testDeployment
    *appsv1.Deployment, ingress *networkingv1.Ingress) {
    setCanaryRunning(true)

    defer setCanaryRunning(false) // set the routine "not
        → running" when this function returns
    fmt.Println("Test environment is up and ready to be tested!")

    // Run the tests

    if r.runTests(ctx) {
        //...
    }
}

func (r *ScalerReconciler) Reconcile(ctx context.Context, req
    → ctrl.Request) (ctrl.Result, error) {
    log := log.FromContext(ctx).WithValues("Request.Namespace",
        → req.Namespace, "Request.Name", req.Name)
    log.Info("Reconcile called")

    //...
    go r.canaryRoutine(ctx, deployment, testDeployment, ingress)
    //...
    return ctrl.Result{RequeueAfter: time.Duration(30 *
        → time.Second)}, nil
}
```

Codice 3.2: Sezione di codice relativo all'applicazione della strategia *Canary*.

Lo *scope* di codice mostra un *Reconcile loop* che chiama una funzione definita come *canaryRoutine* utilizzando la primitiva *go*, utilizzata per chiamare la funzione come *Goroutine*. Questa funzione, semplificata, mostra solo una parte della logica della strategia *Canary* e sua implementazione.

```
func (r *ScalerReconciler) isCPUThresholdExceeded(ctx
    ↳ context.Context, deployment *apiv1alpha1.DeploymentInfos,
    ↳ threshold int32) (bool, error) {
    fmt.Printf("*** Checking CPU usage: ***\n")
    //...
}

//... Reconcile loop

for _, deployment := range deploymentsInfos {
    fmt.Printf("\n*** Starting Scaling routine for deployment
        ↳ with name: %s : ***\n", deployment.Name)
    // Check thresholds and add replicas if exceeded
    cpuExceeded, err := r.isCPUThresholdExceeded(ctx,
        ↳ &deployment, scaler.Spec.CPUMThreshold)
    if err != nil {
        log.Error(err, "failed to check CPU threshold")
        return ctrl.Result{}, err
    }
    if cpuExceeded {
        fmt.Printf("Warning: CPU threshold exceeded \n")
        if err := r.addReplica(ctx, &deployment,
            ↳ interval.Replicas); err != nil {
            log.Error(err, "failed to add replica - CPU
                ↳ threshold")
            return ctrl.Result{}, err
        }
    }
}
//...
```

Codice 3.3: Sezione di codice relativo alla funzionalità di *Scaling*.

Quest'altra sezione di codice, invece, mostra una parte della logica del *Reconcile loop* responsabile dello *scaling* dei *pod* dell'applicazione in base alla percentuale di *CPU* in utilizzo. La funzione chiamata *CPUThresholdExceeded* si occupa invece di monitorare effettivamente la *CPU* dei *pod*.

Riassumendo durante il processo di codifica ho sviluppato non solo l'applicazione di *test*, e la logica dell'operatore, ma anche tutti i *file* di configurazione necessari alla creazione dell'ambiente di rilascio e *test* di entrambi. Riporto in formato tabellare delle stime del numero linee di codice scritte effettivamente (*Single line of code*), suddivise per linguaggi, per ogni componente del progetto:

Si noti che i dati sono stati ricavati utilizzando *Cloc*³, uno strumento apposito.

Prodotto	Linee di codice	Numero di <i>file</i>	Linguaggi
Applicazione di <i>test</i>	526	9	<i>Javascript</i>
Immagini <i>Docker</i>	45	4	<i>Dockerfile</i>
Operatore	3484	13	<i>Go</i>
<i>Manifests</i>	1423	51	<i>YAML</i>
Commenti	1101	nd	Misti
Documentazione	568	4	<i>Markdown</i>
Totale	>7141	>81	Misti

Tabella 3.2: Tabella riassuntiva del numero di linee di codice prodotto.

3.4 Verifica

L'attività di verifica è stata condotta esclusivamente sul codice relativo all'operatore e a corredo della strategia di rilascio *Canary*. Questa decisione è stata presa in considerazione del fatto che le applicazioni che l'operatore gestirà

³ *Cloc*. URL: <https://github.com/AlDanial/cloc>.

saranno già, presumibilmente, sottoposte a *test* unitari durante il loro sviluppo. Questa tipologia di *test* non è quindi stata prevista per l'applicazione di *test*, in quanto ritenuti di secondaria importanza.

Concentrarsi sulla verifica dell'operatore e sull'implementazione dei *test* per la strategia *Canary* ha permesso di garantire la stabilità e l'affidabilità del processo di gestione delle applicazioni all'interno dell'ambiente *Kubernetes*, assicurando che l'operatore potesse eseguire correttamente le sue funzioni chiave, come il controllo dello *scaling*, la distribuzione delle nuove versioni e il monitoraggio dello stato dei *pod*.

Il processo di verifica è stato suddiviso in due macro sezioni: analisi statica e dinamica.

3.4.1 Analisi Statica

L'analisi statica del codice dell'operatore ha previsto l'adozione di 2 modalità differenti: manuale e automatica.

Manualmente ho fatto uso dei metodi già esplorati durante il corso di Ingegneria del *Software* entrambi i quali prevedono, appunto, la revisione manuale del codice:

- **Walkthrough:** viene utilizzato nel momento in cui non si sappia dove viene riscontrata la problematica e consiste in una lettura più ampia scorrendo nella sua interezza il documento/codice per trovare l'errore. Questo metodo è sicuramente molto efficace ma anche molto dispendioso in termini di risorse;
- **Inspection:** a differenza del *Walkthrough*, questo metodo viene utilizzato quando si è a conoscenza di dove potrebbe essere la problematica. Risulta quindi essere un approccio più mirato per l'eliminazione dell'errore e molto meno dispendioso.

Per quanto invece riguarda procedimenti automatici di analisi statica del codice, mi sono affidato agli strumenti dell'*editor* utilizzato per la stesura di

praticamente tutti i *file*, *Visual Studio Code*⁴, e ai controlli eseguiti dal *compiler* di *Go*. Quest’ultimo non solo esegue la compilazione del codice, ma effettua anche una serie di controlli che vanno dalla verifica della corretta tipizzazione delle variabili, alla rilevazione di importazioni non utilizzate, fino alla gestione delle condizioni di errore. Questi controlli sono essenziali per garantire che il codice rispetti le convenzioni e le buone pratiche del linguaggio, contribuendo a ridurre il rischio di *bug* e a migliorare la qualità complessiva del progetto.

3.4.2 Analisi Dinamica

In questa sottosezione, oltre ad illustrare quanto fatto per i *test*, relativi all’analisi dinamica, del codice dell’operatore, descriverò anche quanto ideato e sviluppato per quanto riguarda i *test* relativi alla strategia di *Canary deployment*. Come già accennato in precedenza, la strategia *Canary*, prevede il rilascio graduale di una nuova versione dell’applicazione a un sottogruppo limitato di utenti, mentre il resto degli utenti continua a utilizzare la versione precedente. Questo approccio permette di verificare il nuovo *endpoint* in un ambiente di produzione con un impatto minimo, identificando eventuali *bug* o problemi di *performance* in modo rapido e controllato. Durante questa fase, il comportamento e le metriche della nuova versione vengono attentamente monitorati e confrontati con quelli della versione precedente per verificare che tutto funzioni come previsto. Se il nuovo *endpoint* supera con successo questi *test*, la distribuzione può essere estesa a un numero maggiore di utenti, fino a diventare la versione principale per tutti, stabile. In caso contrario, è possibile ritirare rapidamente la nuova versione e ripristinare quella precedente, minimizzando i rischi e garantendo la continuità del servizio.

Insieme al mio *tutor* aziendale, abbiamo deciso di rivederne in parte la logica per meglio allinearla alle esigenze dell’azienda. Invece di distribuire casualmente una parte degli utenti alla nuova versione in fase di *test*, come avviene solitamente, abbiamo optato per un approccio più controllato. Utilizzando l’*Ingress* precedentemente menzionato, abbiamo creato un nuovo *endpoint* accessibile esclusi-

⁴ *Visual Studio Code*. URL: <https://code.visualstudio.com/>.

vamente a coloro che possiedono i permessi necessari, come sviluppatori e *tester*. Questi ultimi avranno il compito di verificare manualmente il funzionamento e l'usabilità dell'applicazione, consentendo un *feedback* più mirato e un controllo più preciso durante la fase di *test*.

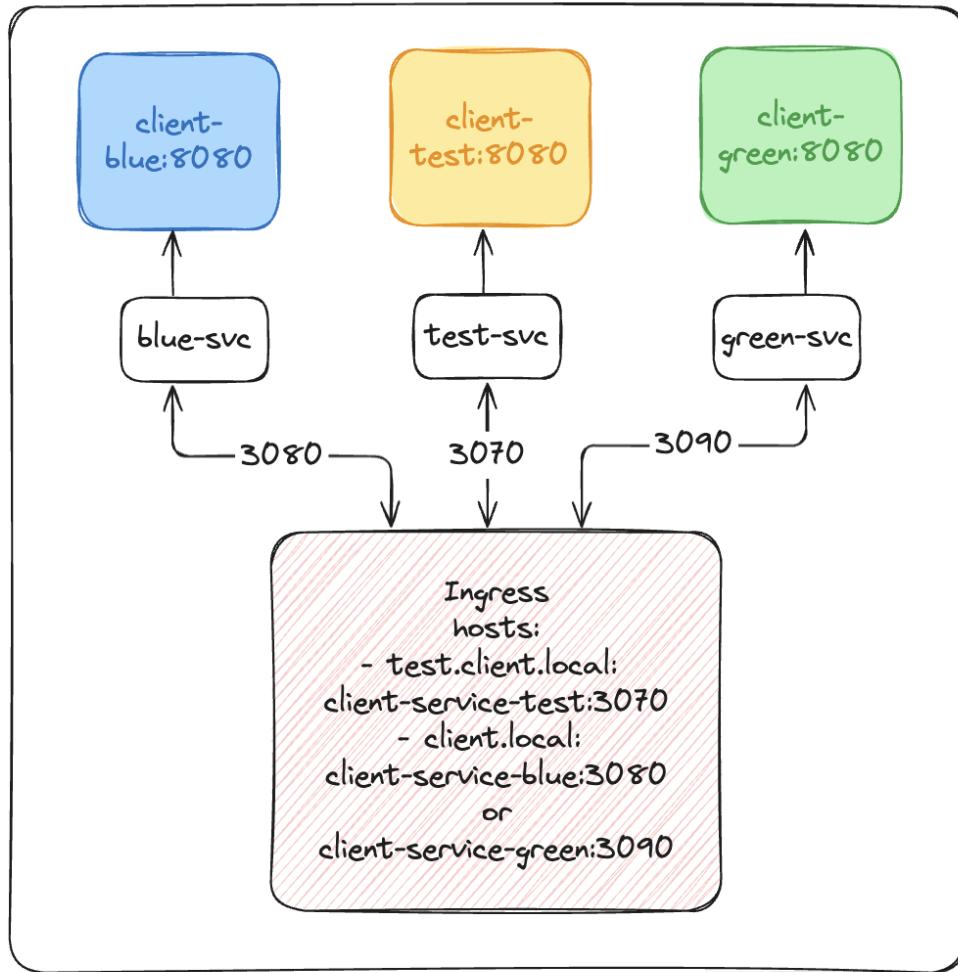


Figura 3.11: Astrazione dell'*Ingress* e suo funzionamento.

A completare il *testing* della "nuova" applicazione, ho sviluppato una *suite* di *test* automatici, gestiti dall'operatore stesso, che andassero a verificare non tanto le singole unità del codice dell'applicazione, poiché si presume che siano già state verificate precedentemente, ma piuttosto l'integrazione tra le diverse componenti dell'applicazione e il suo corretto funzionamento all'interno del *cluster Kubernetes*. Questi *test* miravano a simulare scenari realistici per garantire che le diverse parti dell'applicazione lavorassero insieme senza problemi e che

fossero correttamente configurate per interagire con le risorse del *cluster*, come i volumi persistenti, i servizi e gli ingressi. Ad esempio, verificando che l'applicazione fosse accessibile agli utenti finali tramite gli *endpoint* configurati, e che le operazioni comuni potessero essere eseguite senza errori, garantendo un'esperienza utente senza intoppi.

Ritornando invece a parlare del codice dell'operatore, per quanto riguarda i *test* di unità relativi, ho utilizzato gli strumenti che fornisce l'*Operator SDK*. Questi strumenti includono il *test framework* integrato che consente di simulare eventi e stati del *cluster Kubernetes* all'interno di un ambiente controllato.

Grazie all'*Operator SDK*, ho potuto scrivere *test* di unità per verificare singolarmente le funzioni del *controller*, assicurandomi che ogni parte del codice reagisse correttamente ai cambiamenti nello stato delle risorse gestite. Per esempio, ho verificato come l'operatore crea, aggiorna o elimina le risorse in risposta a modifiche nelle *Custom Resource Definitions (CRD)* associate.

3.5 Validazione e collaudo

Durante l'ultima settimana di tirocinio, ho avuto l'opportunità di collaudare l'operatore che ho sviluppato, utilizzandolo per gestire una delle applicazioni di base dell'azienda. Di seguito, presento alcuni frammenti dell'*output* generato dall'operatore, che illustrano l'intero ciclo di aggiornamento dell'applicazione utilizzando le strategie di *deployment Blue-Green* e *Canary*. Ho suddiviso questi frammenti in modo da migliorarne la comprensibilità e facilitare l'analisi dei singoli passaggi eseguiti dall'operatore durante il processo di aggiornamento.

```
*** Fetching all the running deployments in Namespace: default
↪ ***
Found these deployments:
- Name: client-blue | to be managed ...
- Name: client-test

*** Starting Canary routine : ***
Searching for new (test) deployment environment pods: Namespace:
↪ default, DeploymentName: client, targetEnv: test
Number of new (test) pods: 2

*** (Goroutine running) Canary routine: ***
Test enviroment is up and ready to be tested!
*** Running Test: Client Availability ***
*** Testing client Availability: ***
Test Client Availability passed.

...
All the jobs have been run. Press [Y] to make the test env the
↪ new stable env and starting the Blue-Green routine, or press
↪ [N] to rollback:
Y
You entered: Y
```

Codice 3.4: Output dell'inizio del ciclo di aggiornamento.

Questo primo frammento mostra l'inizio del ciclo di aggiornamento dell'applicazione all'interno del *cluster*. In esso è anche possibile vedere l'applicazione della strategia *Canary* e la gestione dei relativi *test*.

```
*** Setting current schedule : ***  
Current time is within interval 16:30-23:59, max pod setted to: 3  
  
*** Starting Scaling routine for deployment with name:  
→ client-blue : ***  
  
*** Checking CPU usage: ***  
DEBUG: searching for pods labeled as: client-blue  
Number of pods found: 2  
CPU % usage for pod named client-blue-5dfbd68bc8-chrrh : 10%  
CPU % usage for pod named client-blue-5dfbd68bc8-vbhpm : 10%  
Average CPU usage for deployment client-blue: 10%  
  
*** Checking Memory usage: ***  
DEBUG: searching for pods labeled as: client-blue  
Number of pods found: 2  
Memory % usage for pod named client-blue-5dfbd68bc8-vbhpm : 12%  
Memory % usage for pod named client-blue-5dfbd68bc8-chrrh : 12%  
Average memory usage: 12%  
  
*** Checking ActiveUsers: ***  
Using port: 3080  
Active users: 36
```

Codice 3.5: *Output* dei processi di *scaling* e *scheduling* dei *pod*.

In quest'altro, è invece possibile visionare l'*output* relativo ai processi di schedulazione e *scaling* dei *pod*. Si può notare inoltre il monitoraggio di questi ultimi e delle loro principali risorse.

```
New served (test) service: client-service-test
New served (test) service port is: 3080
...
Old environment pods scaled to 0 - Completed
Deleting the deployment ...
Deployment deleted - Completed
Test deployment image: ross7/myapp-green:latest
Creating new env based on the tested image ...
New env ready!
Switched test enviroment to new stable release, starting
→ Blue-Green routine ...

!!!!!!!!!!!!!! Response: Hi from a BLUE deployment,
→ someone_there? true !!!!!!!!!

Active users: 38
!!!!!!!!!!!!!! Response: Hi from a BLUE deployment,
→ someone_there? true !!!!!!!!!

!!!!!!!!!!!!!! Response: Hi from a BLUE deployment,
→ someone_there? true !!!!!!!!!
```

Codice 3.6: *Output* della conclusione della strategia *Canary*.

Mentre qui sopra, è possibile analizzare l'*output* relativo alla fine della prima iterazione del ciclo di aggiornamento, comprendente inoltre il controllo degli utenti attivi nella vecchia versione dell'applicazione.

```
...
!!!!!!!!!!!!!!!!!!!!!! Response: Hi from a BLUE deployment,
→ someone_there? false !!!!!!!!!!!!!!!!!

*** Shutting down old deployment: ***
Searching for old deployment environment pods: Namespace:
→ default, DeploymentName: client, targetEnv: blue
Waiting for the pods to actually being scaled down ...

...
Old environment pods scaled to 0 - Completed
Deleting the deployment ...
Deployment deleted - Completed
Migration to new environment - Successfull
```

Codice 3.7: *Output della conclusione della strategia Blue-Green.*

Infine, in quest'ultimo frammento, è possibile vedere la fine del ciclo di aggiornamento, che si conclude con l'applicazione della strategia *Blue-Green*. Com'è possibile constatare complessivamente, vengono eseguite tutte le operazioni menzionate in precedenza, come il monitoraggio delle risorse, lo *scheduling* dei *pod*, e l'aggiornamento dell'applicazione.

Successivamente, insieme al mio *tutor* aziendale, abbiamo analizzato quanto svolto e raggiunto, in funzione del soddisfacimento dei requisiti definiti. Nello specifico ho coperto la maggior parte dei requisiti che avevamo previsto. Riporto in formato tabellare gli esiti del tracciamento:

	Obbligatori	Desiderabili	Totali	Soddisfatti
Funzionali	30	2	32	31
Qualitativi	4	2	6	5
Di vincolo	6	2	8	8
Documentali	4	1	5	4
Totali	44	7	51	48

Tabella 3.3: Tabella riassuntiva dei requisiti coperti durante il corso del progetto.

Com’è possibile consultare in tabella 3.3, gli unici requisiti non soddisfatti risultano essere 3, nello specifico 2 affibbiati al gruppo dei desiderabili e uno obbligatorio.

Il requisito funzionale obbligatorio, non soddisfatto, riguarda la gestione di versioni incompatibili del *database*. Per tale casistica ho infatti svolto numerose ricerche, ma senza riuscire a completamente sopperire al problema. Riporto alcune fonti correlate che penso possano essere interessanti:

- ***CloudNativePG***⁵: è un operatore *Kubernetes open-source* progettato per gestire e orchestrare *database PostgreSQL* su *cluster*. L’obiettivo principale di *CloudNativePG* è fornire una soluzione scalabile, sicura e resiliente per il *deployment*, la gestione e il monitoraggio di *PostgreSQL* in ambienti *cloud-native*;
- ***Atlas Operator***⁶: altro operatore per la gestione e il controllo dei *database*. È progettato per integrare *Atlas*, una piattaforma che consente la gestione degli schemi dei *database* tramite un approccio “*Database-as-Code*”.

⁵ *CloudNativePG*. URL: <https://cloudnative-pg.io/>.

⁶ *Atlas*. URL: <https://atlasgo.io/integrations/kubernetes/operator>.

CAPITOLO 3. IL PROGETTO

L'*Atlas Operator* automatizza e facilita la gestione degli schemi del *database* all'interno di un *cluster*, rendendo il processo di sviluppo, migrazione e gestione dei *database* più fluido e conforme alle pratiche più moderne.

La funzionalità si è rivelata essere troppo complessa, e spendere altro tempo su di essa avrebbe compromesso altri aspetti del progetto, i quali, in accordo con l'azienda, abbiamo deciso di prioritizzare. Ritengo comunque che il lavoro svolto complessivamente sia più che apprezzabile, considerando che il tutto è stato sviluppato da zero. Il mio progetto ha gettato le basi per futuri sviluppi e miglioramenti. Sono convinto che questo lavoro possa servire da solida base per progetti successivi, facilitando ulteriori esplorazioni e perfezionamenti nelle aree trattate.

Capitolo 4

Valutazioni retrospettive

4.1 Soddisfacimento degli obiettivi

A conclusione del processo di sviluppo del progetto, insieme al mio *tutor* aziendale, abbiamo svolto un'attività di retrospettiva per ripercorrere l'intero periodo di *stage* e valutarne i risultati, analizzandone la completezza dei prodotti. Presento quindi un resoconto in formato tabellare che mostra lo stato di completamento di tutti gli obiettivi previsti dal piano di lavoro, aggiornato in modo progressivo.

Codice	Stato	Descrizione
O-O1	Soddisfatto	A coprire l'obiettivo, è la documentazione prodotta a tal proposito. Ho redatto un documento specifico che include inoltre lo studio sulle tecnologie inerenti.
O-O2	Soddisfatto	Ho completamente compreso la logica dietro al ciclo di vita dei <i>pod Kubernetes</i> . Gestendone creazione e eliminazione.
Continua nella prossima pagina...		

Tabella 4.1 – Continuo della tabella

Codice	Stato	Descrizione
O-O3	Soddisfatto	A coprire l'obiettivo, è l'operatore prodotto e poi aggiornato. Ho sviluppato l'operatore partendo dalle funzionalità di base, apprese durante la precedente formazione.
O-O4	Soddisfatto	Ho compreso e gestito il meccanismo di <i>scaling</i> dei <i>pod</i> riuscendo anche a misurare i diversi approcci (verticale, orizzontale e misto)
O-O5	Soddisfatto	Ho prima compreso, e poi integrato nell'operatore la strategia <i>Blue-Green</i> per la gestione degli aggiornamenti.
O-O6A	Soddisfatto	L'ambiente che ho prodotto utilizza e gestisce la cartella condivisa riuscendo correttamente a garantire la persistenza dei dati all'interno dell' <i>cluster</i> .
O-O6B	Non soddisfatto	Come già menzionato, purtroppo, non sono riuscito a gestire gli aggiornamenti di versione dell'applicazione tra versioni aventi <i>database</i> incompatibili
O-O7	Soddisfatto	L'operatore che ho sviluppato adotta con efficacia la strategia <i>Canary</i> per il <i>test</i> delle nuove versioni e loro successivo rilascio in combinazione con la <i>Blue-Green</i> .
O-O8	Soddisfatto	Ho correttamente sviluppato una <i>suite</i> di <i>test</i> a completamento della strategia <i>Canary</i> , integrandoli nell'operatore.
Continua nella prossima pagina...		

Tabella 4.1 – Continuo della tabella

Codice	Stato	Descrizione
O-O9	Soddisfatto	A coprire l'obiettivo è la documentazione prodotta a tal proposito. Ho infatti corredato il codice di relativa documentazione che ne illustrasse il funzionamento.
O-10	Soddisfatto	Ho sviluppato un'applicazione utile al <i>test</i> dell'operatore, comprendendo inoltre come gestirne i componenti all'interno del <i>cluster</i> .
O-11	Soddisfatto	Ho gestito correttamente la persistenza di alcuni dati, tramite l'uso di cartelle condivise e <i>storage</i> esterni.
O-12	Soddisfatto	L'applicazione dispone di un <i>database</i> per la gestione di alcuni dati. Il quale sono riuscito a gestire con successo all'interno del <i>cluster Kubernetes</i> .
O-D1	Soddisfatto	Ho avuto modo di verificare il funzionamento del <i>cluster</i> prodotto con immagini basate su architetture differenti documentandone i risultati.

Tabella 4.1: Tabella retrospettiva, degli obiettivi e loro stato finale.

A verificare lo stato degli obiettivi sopra elencati, sono i prodotti consegnati e/o presentanti all'azienda, quali :

- Documentazione relativa al codice sviluppato complessivamente;
- Documentazione relativa allo studio iniziale delle tecnologie;
- *Environment Kubernetes* per il *testing* dell'operatore;
- Operatore realizzato in *Go*, munito di tutte le *feature* previste;

- Presentazione a scopo informativo sul linguaggio di programmazione *Go*.

Gli obiettivi completati coprono oltre il 90% di quelli previsti, considerando in particolare quelli obbligatori. Questo dato evidenzia che sono riuscito a raggiungere la maggior parte dei traguardi prefissati all'inizio dello *stage*. Di conseguenza, posso ritenermi complessivamente soddisfatto dell'esperienza. Inoltre, avendo consegnato all'azienda tutti i prodotti previsti e completato con successo le attività principali, ritengo che il mio *stage* sia stato un successo. Soprattutto considerando la complessità delle tecnologie coinvolte e la mia inesperienza iniziale nell'utilizzarle. Come ho già sostenuto, sono convinto che il lavoro svolto possa costituire una solida base per progetti futuri e contribuire positivamente ai piani dell'azienda.

Per quanto riguarda gli obiettivi personali, posso esprimere un giudizio altrettanto positivo. Non solo ho avuto l'opportunità di apprendere numerose nozioni su tecnologie di mio interesse, in primis *Kubernetes*, ma ho anche potuto sperimentare per la prima volta un contesto lavorativo aziendale. Questo mi ha permesso di confrontarmi con nuove dinamiche, imparare a lavorare in *team* e gestire responsabilità concrete, migliorando le mie capacità di adattamento e comunicazione. Ho acquisito un'esperienza pratica che ha arricchito il mio bagaglio professionale, rendendomi più consapevole delle sfide e delle opportunità che il mondo del lavoro può offrire. Penso che questo percorso mi abbia permesso di crescere non solo dal punto di vista tecnico, ma anche da quello personale, sviluppando una maggiore fiducia in me stesso e nelle mie capacità di affrontare progetti complessi in un ambiente strutturato.

4.2 Crescita personale

Lo *stage* mi ha permesso di acquisire competenze significative sia sul fronte tecnologico che su quello professionale. Prima del tirocinio, infatti, non avevo mai lavorato direttamente con le tecnologie che il progetto richiedeva. In particolare, l'unica tecnologia con cui avevo avuto un'esperienza minima era *Docker*, durante il corso di Ingegneria del *Software*. Le altre tecnologie, come *Kubernetes*, erano per me conosciute solo per nome, senza aver mai avuto l'occasione di

approfondirle o di utilizzarle in un progetto pratico e concreto.

L'approccio al linguaggio di programmazione *Go* rappresentava per me un'ulteriore sfida, dato che si tratta di un linguaggio relativamente giovane rispetto ai classici come *Java* o *C++*, con paradigmi e caratteristiche proprie che lo differenziano notevolmente dagli altri linguaggi con cui avevo familiarità. La necessità di imparare e applicare rapidamente queste nuove tecnologie ha stimolato la mia capacità di adattamento e di apprendimento autonomo, portandomi a sviluppare un approccio più critico e consapevole nei confronti delle nuove sfide tecniche. Dal punto di vista professionale, lo *stage* mi ha offerto l'opportunità di immergermi in un ambiente aziendale strutturato e di osservare come un'azienda affermata come Zucchetti gestisce i suoi processi operativi e organizzativi. Ho avuto modo di imparare a lavorare con rigore e metodo, seguendo procedure standardizzate e rispettando le tempistiche assegnate. Inoltre, ho potuto collaborare con colleghi esperti, osservando le loro metodologie di lavoro, comprendendo l'importanza della comunicazione efficace e del lavoro di squadra, e imparando a gestire le relazioni professionali in un contesto di collaborazione.

Questa esperienza mi ha aiutato a sviluppare una maggiore consapevolezza delle dinamiche aziendali e a comprendere l'importanza del ruolo di ciascun individuo all'interno di un progetto complesso. Mi ha insegnato non solo le competenze tecniche necessarie, ma anche l'importanza della flessibilità, della capacità di lavorare sotto pressione e della gestione del tempo e delle risorse in modo efficiente. Penso che questi aspetti abbiano contribuito in modo significativo alla mia crescita personale e professionale, fornendomi una base solida per affrontare future sfide lavorative.

4.3 Riflessioni finali

In conclusione, vorrei riflettere su come il mio percorso di studi mi abbia preparato a questa esperienza conclusiva di *stage*. Tengo a precisare che quanto segue è solo una mia opinione personale, basata sulle mie sensazioni e osservazioni. Non mi considero in possesso delle competenze necessarie per valutare in modo approfondito l'intero percorso di studi, ma desidero comunque condividere-

re il mio punto di vista.

Ritengo che il corso che meglio rispecchia ciò che ogni studente si trova ad affrontare durante il tirocinio sia quello di Ingegneria del *Software*, che non a caso ho menzionato più volte nel documento. Questo corso si distingue infatti per l'accento posto sulla componente pratica dello sviluppo *software*, un aspetto che è spesso meno enfatizzato in altri corsi del nostro programma di studi. Durante il corso di Ingegneria del *Software*, ho avuto la possibilità di confrontarmi con situazioni che richiedevano un approccio pratico e concreto, avvicinandosi così a quello che è stato il lavoro svolto durante il mio *stage*.

Il nostro corso di laurea è strutturato in modo da fornire una solida base teorica in vari ambiti dell'informatica, cosa che considero estremamente preziosa. Tuttavia, ho notato che l'aspetto pratico dello sviluppo *software* viene affrontato solo nelle fasi finali del percorso di studi. Questo approccio mi ha portato a sentirmi inizialmente un po' disorientato quando mi sono trovato a dover applicare le mie conoscenze in un contesto pratico, come quello del corso di Ingegneria del *Software*. Se non avessi avuto l'opportunità di seguire questo corso, soprattutto per quanto riguarda la parte progettuale, probabilmente avrei affrontato il tirocinio con un senso di spaesamento ancora maggiore.

L'aver avuto un corso come Ingegneria del *Software* prima del tirocinio mi ha permesso di sviluppare una base pratica che si è rivelata cruciale per affrontare le sfide del tirocinio stesso. Sono convinto che l'esperienza del progetto di questo corso abbia svolto un ruolo fondamentale nel prepararmi a confrontarmi con problemi reali e nel darmi una certa sicurezza nell'approcciare il mondo del lavoro. In definitiva, mentre la solida formazione teorica fornita dal corso di laurea è essenziale, penso che una maggiore integrazione della componente pratica durante tutto il percorso potrebbe ulteriormente arricchire l'esperienza degli studenti, preparandoli in modo ancora più efficace per il loro ingresso nel mondo del lavoro.