



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

CHEFBOOK: una piattaforma social per la condivisione di ricette

RELATORE

Prof. Carmine Gravino

Università degli Studi di Salerno

CANDIDATO

Gianmarco Riviello

Matricola: 0512113808

Anno Accademico 2024-2025

Questa tesi è stata realizzata nel



*A chi era la dimostrazione che l'intelligenza non è rappresentata da un voto
A chi mi strappava interi quaderni per farmi capire come studiare ed ambire alla perfezione
A chi, a fine allenamento, stava in chiamata con me affacciata da una finestra d'ospedale
A chi, durante la malattia, non ha mai perso il sorriso e la voglia di vivere
A chi oggi, purtroppo, è assente giustificata
A te... Mamma*

Abstract

La presente tesi illustra il processo di progettazione, sviluppo, testing e deployment di ChefBook, una piattaforma social ideata per promuovere la condivisione di ricette culinarie e facilitare l’interazione tra chef e appassionati di cucina. L’obiettivo era di creare un ambiente sicuro, scalabile e intuitivo, che permetta agli utenti di pubblicare, modificare, trasferire e visualizzare ricette, con funzionalità avanzate di controllo della privacy.

L’implementazione è stata realizzata interamente in Java utilizzando il framework Spring Boot, adottando un’architettura RESTful suddivisa in tre layer principali (Controller, Service, Repository). Il sistema utilizza MySQL come database relazionale e gestisce l’autenticazione mediante JWT e hashing delle credenziali con BCrypt.

Per assicurare la portabilità e facilitare il deployment, l’applicativo è stato containerizzato attraverso Docker, includendo sia l’applicazione che il database. È stato successivamente migrato su AWS, dove è stata progettata un’infrastruttura cloud basata su Elastic Beanstalk per l’esecuzione, RDS per il database e la suite AWS Code per la gestione del ciclo di vita del software.

Una componente fondamentale del lavoro è stata l’integrazione di pratiche DevOps. È stata implementata una pipeline CI/CD su Jenkins per l’ambiente on-premise, con stage automatizzati per build, test e deploy, integrando l’analisi statica del codice con SonarQube per garantire il rispetto di metriche qualitative. Sono stati eseguiti test unitari, test d’integrazione e validazioni via Swagger UI, sia in locale che tramite endpoint AWS.

Il progetto si conclude con un’analisi comparativa tra i tre ambienti di deployment (on-prem, Docker e cloud) mettendo in evidenza vantaggi, criticità e implicazioni in termini di costi, prestazioni e sicurezza. *ChefBook* si configura come un caso di studio completo sull’ingegneria del software moderna, dimostrando come la sinergia tra architetture software, strumenti DevOps e servizi cloud possa portare allo sviluppo di soluzioni digitali robuste, efficienti, a basso impatto economico grazie all’ottimizzazione del ciclo di sviluppo ed infine pronte per l’evoluzione futura.

Indice

Elenco delle Figure	iv
Elenco delle Tabelle	vi
1 Introduzione	1
1.1 Contesto Applicativo	1
1.2 Obiettivi	2
1.3 Risultati	4
2 Stato dell'arte	6
2.1 Piattaforme simili	6
2.1.1 Manjoo	6
2.1.2 Cookbooth	6
2.2 Tecnologie di sviluppo software all'avanguardia	7
2.2.1 Java Spring 6 e Java Spring Boot 3	7
2.2.2 Docker	9
2.2.3 SonarQube	16
2.2.4 DevOps	18
2.2.5 AWS - Amazon Web Services	23
3 Sviluppo Applicativo	29

3.1	Progettazione di Chefbook	29
3.1.1	Analisi del contesto	29
3.1.2	Requisiti funzionali	30
3.1.3	Requisiti non funzionali	30
3.1.4	Use Case	31
3.2	Scelte progettuali	35
3.2.1	Architettura	35
3.2.2	Database	35
3.2.3	Autenticazione	36
3.3	Scelte implementative	37
3.3.1	Design Pattern	37
3.4	Sviluppo Software	38
3.4.1	Entità	38
3.4.2	Controller	40
3.4.3	Service	43
3.4.4	Repository	45
3.4.5	JWT	46
3.4.6	Docker	49
3.4.7	AWS	53
4	Testing e Validazione	62
4.1	Introduzione	62
4.2	Qualità del codice	63
4.2.1	Quando è stato utilizzato	64
4.3	Test	65
4.4	Code coverage	66
4.5	Automazione CI/CD con Jenkins	66
4.6	Swagger UI	67
4.7	Test Cloud	68
5	Conclusioni	69
5.1	Costi e Conclusioni	69
5.2	Sviluppi futuri	71

5.2.1	Interfaccia Grafica	71
5.2.2	Passaggio all'architettura AWS con le Lambda Functions . . .	72
5.2.3	Gestione Avanzata della Registrazione Utenti	73
Bibliografia		74

Elenco delle figure

2.1	Confronto Fra Docker e Macchina virtuale	13
2.2	Schema architettura Docker	16
2.3	Ciclo di vita del DevOps	20
2.4	Jenkins	22
2.5	Settings di Jenkins	23
2.6	AWS - Tipi di Cloud	27
2.7	AWS Services	28
3.1	Use Case Diagram	31
3.2	Use Case Aggiunta ricetta	32
3.3	Use Case Elimina ricetta	33
3.4	Use Case Modifica ricetta	34
3.5	Architettura Spring - RESTful API	35
3.6	Autore, relativo DTO e mapper	40
3.7	Autore Controller	41
3.8	Autore Service	44
3.9	Autore Repository	45
3.10	JWT Token Generator	46
3.11	JWT Token Validator	48
3.12	Dockerfile	50

3.13 Docker-compose	51
3.14 Docker Desktop	52
3.15 Architettura AWS	54
3.16 AWS - CodeCommit	55
3.17 AWS - RDS	56
3.18 AWS - Beanstalk	57
3.19 AWS - Ambiente Beanstalk Architettura	57
3.20 AWS - CodePipeline	61
4.1 Grafico Debito tecnico	63
4.2 SonarQube in Pipeline	64
4.3 Analisi statica SonarQube	65
4.4 Interfaccia grafica di Swagger	68
5.1 Costi dei servizi AWS	70
5.2 Mock up di ChefBook	71

Elenco delle tabelle

2.1 Confronto tra Docker e Macchine virtuali	14
--	----

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Negli ultimi anni, l'evoluzione delle tecnologie digitali ha profondamente trasformato il modo in cui le persone interagiscono, condividono conoscenze e collaborano in numerosi ambiti professionali. Anche il settore culinario, storicamente legato alla manualità e alla trasmissione orale delle competenze, ha attraversato questa trasformazione. Chef professionisti, appassionati e semplici curiosi hanno trovato nel digitale un nuovo spazio di espressione e condivisione. La cucina è diventata protagonista di blog, social network, app e piattaforme tematiche in cui ricette, tecniche e consigli vengono scambiati quotidianamente, spesso accompagnati da foto, video e contenuti interattivi.

Questa digitalizzazione ha permesso di normalizzare l'accesso al sapere culinario, favorendo la sperimentazione e la contaminazione tra tradizione ed innovazione. Il sapere, un tempo custodito gelosamente o trasmesso solo attraverso lunghi percorsi formativi, è oggi a portata di click, accessibile anche da dispositivi mobili e disponibile in molteplici formati. I contenuti generati dagli utenti e la logica partecipativa delle moderne piattaforme online hanno creato comunità vivaci e collaborative, dove ogni utente può contribuire con la propria esperienza e creatività.

All'interno di questo contesto si inserisce lo sviluppo della piattaforma ChefBook, oggetto di questa proposta di tesi. ChefBook nasce come un social network verticale, pensato per offrire uno spazio digitale dedicato esclusivamente al mondo della cucina, con l'obiettivo di connettere utenti attraverso la condivisione di ricette, suggerimenti, tecniche e storie personali legate al cibo. L'intento è quello di creare una community inclusiva, dove la tecnologia non sia solo uno strumento di pubblicazione, ma anche un mezzo per apprendere, ispirarsi e crescere professionalmente.

L'approccio adottato per la realizzazione della piattaforma si fonda su principi di accessibilità, modularità e scalabilità. Particolare attenzione è stata rivolta alle modalità di distribuzione e deploy dell'applicativo, esplorando soluzioni in ambienti on-premise, tramite container Docker, e su Amazon Web Services (AWS). In questo modo si è potuto studiare in dettaglio l'impatto delle scelte infrastrutturali su costi, prestazioni e sicurezza, integrando pratiche DevOps e pipeline CI/CD per garantire un rilascio continuo, testato e affidabile.

ChefBook rappresenta dunque non solo un esercizio di sviluppo software, ma anche un progetto che incarna le dinamiche contemporanee della cultura digitale, ponendo la tecnologia al servizio della creatività gastronomica e della condivisione delle competenze.

1.2 Obiettivi

Il progetto **ChefBook** si colloca all'interno di un più ampio panorama di innovazione digitale nel settore *food-tech*, con l'obiettivo primario di sviluppare una *piattaforma social tematica dedicata al mondo culinario*, rivolta sia a chef professionisti che ad appassionati di cucina. L'intento è quello di creare un ecosistema digitale inclusivo, dinamico e orientato alla condivisione, dove gli utenti possano non solo pubblicare ricette, ma anche *collaborare, apprendere e valorizzare la propria creatività gastronomica*.

Gli obiettivi principali del progetto possono essere articolati come segue:

Obiettivi funzionali

- Fornire un ambiente strutturato per la gestione delle ricette, con strumenti intuitivi per la creazione, modifica, consultazione e cancellazione dei contenuti.
- Garantire un controllo specifico sulla privacy.
- Promuovere la collaborazione tra utenti, mediante un sistema a due fasi (*proposta* e *trasferimento*) per la condivisione delle ricette.

Obiettivi architetturali e tecnologici

- Realizzare un sistema modulare, scalabile e accessibile, capace di operare in ambienti on-premise, containerizzati (Docker) e cloud-based (AWS).
- Applicare le best practice DevOps tramite l'implementazione di pipeline CI/CD per l'automazione dei processi di build, test e deploy.
- Utilizzare tecnologie aggiornate e manutenibili, come Java Spring Boot, Docker e SonarQube, per assicurare una base solida e moderna.

Obiettivi qualitativi e strategici

- Sperimentare scenari comparativi tra diversi ambienti di deployment, valutando l'impatto su costi, performance, sicurezza e affidabilità.
- Garantire un buono standard qualitativo del codice, tramite analisi statica, copertura del codice e strumenti di controllo automatizzato.
- Progettare una piattaforma sostenibile ed estendibile, aperta a future evoluzioni sia a livello funzionale che infrastrutturale.

In sintesi, **ChefBook** rappresenta un esempio concreto di ingegneria del software moderna, dove metodologie DevOps, scalabilità cloud e attenzione alla qualità si fondono in un'unica soluzione digitale pensata per valorizzare la cultura gastronomica nel contesto tecnologico attuale.

1.3 Risultati

Lo sviluppo della piattaforma **ChefBook** ha portato alla realizzazione di un sistema funzionante, completo e coerente con gli obiettivi prefissati. La piattaforma integra in un unico ambiente digitale tutte le funzionalità progettate, offrendo un’esperienza utente accessibile, sicura e collaborativa.

In fase di sviluppo, ChefBook è stato implementato e testato in tre ambienti distinti:

1. **On-premise**: per il test iniziale dell’architettura e delle funzionalità principali.
2. **Ambiente containerizzato con Docker**: per garantire portabilità, isolamento delle dipendenze e facilità di distribuzione.
3. **Cloud su Amazon Web Services (AWS)**: per testare la scalabilità e l’integrazione con servizi cloud-native come RDS, Elastic Beanstalk e CodePipeline.

Ogni ambiente è stato dotato di una pipeline CI/CD personalizzata, che ha automatizzato i processi di build, test e deploy. Questo ha consentito di mantenere alta l’affidabilità del codice attraverso test unitari, test di integrazione e analisi statica tramite SonarQube.

Dal punto di vista funzionale, la piattaforma consente:

- la registrazione e autenticazione sicura degli utenti tramite JWT;
- la gestione completa delle ricette (creazione, modifica, cancellazione, visualizzazione);
- la definizione delle impostazioni di privacy per ogni ricetta;
- la condivisione delle ricette tra utenti attraverso un meccanismo strutturato di trasferimento collaborativo;
- la consultazione dell’intera piattaforma anche da parte di utenti non autenticati.

L’adozione di pratiche DevOps e l’utilizzo di strumenti come Jenkins e Swagger UI ha permesso di migliorare la qualità del software, riducendo il rischio di regressioni e facilitando la manutenzione.

Infine, la migrazione sul cloud ha permesso di confrontare i diversi ambienti dal punto di vista di:

- **Prestazioni:** miglioramento dei tempi di risposta e della gestione del carico;
- **Costi:** analisi dei consumi e del modello *pay-as-you-go* di AWS;
- **Sicurezza e disponibilità:** maggiore affidabilità nell'erogazione dei servizi.

Nel complesso, ChefBook rappresenta una dimostrazione concreta di come un progetto software possa essere pensato per evolvere, scalare e adattarsi a diversi scenari operativi, con un impatto positivo sia sull'utente finale che sul processo di sviluppo.

CAPITOLO 2

Stato dell'arte

2.1 Piattaforme simili

2.1.1 Manjoo

Manjoo¹ è un social network italiano dedicato agli appassionati di cucina. La piattaforma consente all'utente di caricare, condividere e stampare una propria creazione culinaria con relativi ingredienti utilizzati, dosi per ingrediente, tecniche di preparazione utilizzate e foto riguardanti la proposta. Manjoo si occupa di formattare le ricette, calcolare i relativi valori nutrizionali, diete, allergie ed intolleranze. Le ricette possono essere condivise all'interno della piattaforma o tramite link attraverso altri canali social.

2.1.2 Cookbooth

Cookbooth è un'applicazione nata nel 2013. L'app funge da social network per chef e appassionati di cucina permettendo loro di condividere ricette e tecniche culinarie attraverso foto e descrizioni dettagliate. Gli utenti hanno la facoltà di poter

¹<https://www.manjoo.it>

seguire altri cuochi, scoprire nuove ricette e interagire con la community culinaria globale.

2.2 Tecnologie di sviluppo software all'avanguardia

Per lo sviluppo del progetto proposto l'azienda *System Management S.p.A* ha deciso di utilizzare tra gli strumenti a disposizione quelli più aggiornati sul mercato in modo da avere una buona futuribilità e manutenibilità.

2.2.1 Java Spring 6 e Java Spring Boot 3

Con l'obiettivo finale di riuscire a distribuire l'applicativo in cloud si è scelto di scrivere il codice utilizzando Java Spring alla luce dei vantaggi che porta anche nel deploy in cloud.

Perchè Spring Framework è così diffuso Spring Framework offre un meccanismo avanzato per la gestione delle dipendenze, noto come *Dependency Injection (DI)*, grazie al quale ogni componente può dichiarare di cosa ha bisogno senza preoccuparsi di come ottenere tali dipendenze: sarà il container Spring a fornirle automaticamente. Questo approccio consente di sviluppare applicazioni composte da moduli indipendenti e facilmente riutilizzabili, ideali per architetture orientate ai microservizi o sistemi distribuiti.

Oltre all'iniezione delle dipendenze, Spring include numerose funzionalità pronte all'uso per semplificare la realizzazione delle applicazioni, come la gestione degli errori, la validazione dei dati, la conversione dei tipi, l'internazionalizzazione, la gestione delle risorse e l'ascolto degli eventi. Inoltre, si integra senza difficoltà con tecnologie dell'ambiente Java EE, tra cui *RMI* e i servizi web basati su Java.

In conclusione, Spring rappresenta una soluzione completa per lo sviluppo di applicazioni Java moderne e flessibili, compatibili con diversi ambienti di esecuzione e adatte a contesti scalabili e a basso accoppiamento tra componenti. [1]

Differenze fra Spring Framework e Spring Boot

- **Spring Framework** è una piattaforma open source pensata per semplificare lo sviluppo di applicazioni Java complesse. La sua architettura modulare offre strumenti per l'iniezione delle dipendenze, la programmazione orientata agli aspetti (AOP), la sicurezza, la gestione delle transazioni, l'accesso ai dati e lo sviluppo web. L'obiettivo è fornire un'infrastruttura flessibile e ordinata per costruire software scalabile e manutenibile.
- **Spring Boot**, invece, nasce per rendere ancora più rapido e semplice l'uso di Spring. Automatizza la configurazione iniziale dell'applicazione e mette a disposizione una serie di starter che preconfigurano le dipendenze più comuni. Inoltre, integra server web embedded e consente l'esecuzione dell'applicazione con un singolo comando, rendendolo particolarmente adatto per ambienti di produzione e sviluppo rapidi. [2]

Vantaggi SpringBoot Dopo aver visto cos'è Spring Boot ecco i vantaggi nell'utilizzarlo:

- **Facilità di sviluppo:** Spring Boot rende più semplice e veloce la creazione di applicazioni Java, grazie alla presenza di numerose librerie preconfigurate. Questo consente agli sviluppatori di concentrarsi sulla logica applicativa piuttosto che sulla configurazione dell'ambiente, riducendo sensibilmente i tempi di avvio dei progetti.
- **Scalabilità:** Il framework è progettato per supportare architetture a microservizi, offrendo così un'ottima base per sviluppare applicazioni che devono gestire un carico crescente o che richiedono una suddivisione modulare dei componenti.
- **Configurazione automatica:** Una delle caratteristiche distintive di Spring Boot è la sua capacità di configurare automaticamente molte impostazioni dell'applicazione in base al contesto. Ciò elimina gran parte delle configurazioni manuali e rende l'ambiente di sviluppo più snello e reattivo.
- **Sicurezza:** Spring Boot integra strumenti di sicurezza avanzati, configurabili in modo flessibile, che consentono di gestire l'autenticazione e l'autorizzazione

degli utenti. Questa protezione può essere adattata a diversi scenari applicativi, garantendo un buon livello di controllo sugli accessi.

- **Testabilità:** Il framework facilita l'adozione di pratiche di testing, supportando test automatici a vari livelli, come test unitari e di integrazione. Questo contribuisce a migliorare la qualità del codice e a individuare rapidamente eventuali malfunzionamenti.
- **Compatibilità con molteplici database:** Spring Boot supporta un'ampia gamma di sistemi di gestione dati, sia relazionali che NoSQL, come MySQL, PostgreSQL, Oracle e MongoDB. Questo permette di scegliere liberamente la tecnologia di persistenza più adatta al progetto.
- **Documentazione dettagliata:** Uno dei punti di forza di Spring Boot è la disponibilità di una documentazione estesa e aggiornata, utile sia per i principianti che per gli sviluppatori più esperti. Le guide ufficiali, i tutorial e gli esempi pratici facilitano l'apprendimento e l'utilizzo del framework. [2]

Spring Security

Spring Security gestisce le comuni vulnerabilità come CSRF, CORs. Per tutte le vulnerabilità individuate, il framework le correggerà immediatamente. Supporta varie implementazioni di autenticazione come username/password e JWT Tokens.

2.2.2 Docker

Software containerizzati

I container rappresentano una tecnologia moderna e sempre più diffusa nello sviluppo software, poiché consentono di creare ambienti isolati e portabili in cui eseguire le applicazioni. Il concetto di *containerizzazione* si basa sull'idea di racchiudere il codice dell'applicazione, insieme a tutte le librerie, framework e dipendenze necessarie, in un unico pacchetto autocontenuto.

Questo approccio garantisce che l'applicazione possa essere eseguita in modo coerente su diverse infrastrutture, a prescindere dal sistema operativo o dall'ambiente

sottostante. Ogni container fornisce un contesto di esecuzione indipendente che protegge l'applicazione da conflitti con il sistema host o con altre applicazioni.

La portabilità e l'isolamento forniti dai container rappresentano una soluzione efficace alle problematiche tradizionali di compatibilità del codice tra ambienti differenti. Infatti, grazie alla containerizzazione, è possibile evitare errori legati a configurazioni non uniformi, riducendo il rischio di bug e migliorando la produttività nello sviluppo e nel deploy.

In sintesi, confezionare un'applicazione all'interno di un container consente di trasferirla ed eseguirla ovunque sia necessario, con la certezza che il comportamento sarà sempre coerente e prevedibile. [3]

Cos'è Docker

Docker è una piattaforma open-source per lo sviluppo di applicazioni in una sandbox. I container, ovvero ambienti virtualizzati leggeri, esistono fin dagli anni '70, ma è stato Docker a renderli ampiamente accessibili. Grazie a Docker, gli sviluppatori possono creare, testare e distribuire applicazioni sia in locale che su server di produzione in modo semplice ed efficiente.

Dal lancio della versione 1.0 nel 2014, Docker ha contribuito a standardizzare l'uso dei container, diventando una tecnologia adottata sia da singoli sviluppatori che da grandi aziende. Attualmente, vanta una base utenti superiore ai 13 milioni, tra cui spiccano nomi come Netflix, Target e Adobe.

L'adozione di Docker è in costante crescita: circa un quarto delle aziende oggi lo utilizza per monitorare le proprie applicazioni. A partire dal 2015, questa percentuale è aumentata ogni anno di circa il 3%. [4]

Docker vs Virtual Machine

Sia le macchine virtuali (VM) che Docker si occupano della difficoltà di eseguire applicazioni in ambienti diversi. Tuttavia, lo fanno per motivi leggermente diversi e con approcci diversi.

Obiettivo Le macchine virtuali sono state originariamente progettate per consentire l'esecuzione di più sistemi operativi su un'unica macchina fisica. L'obiettivo è consentire agli utenti di creare un ambiente virtuale isolato dall'hardware sottostante. Le VM astraggono i dettagli hardware per semplificare l'esecuzione di applicazioni su architetture hardware diverse e utilizzare le risorse hardware in modo più efficiente. Docker, d'altra parte, è stato progettato per fornire un modo leggero e portatile per impacchettare ed eseguire applicazioni in un ambiente isolato e riproducibile. Docker analizza i dettagli del sistema operativo per affrontare la difficoltà di implementare applicazioni in ambienti diversi, come sviluppo, test e produzione. Può essere molto difficile gestire gli aggiornamenti dell'ambiente software e mantenere la coerenza dell'ambiente in ogni suo punto, in particolare nelle organizzazioni che eseguono centinaia di applicazioni o scompongono le applicazioni in centinaia di microservizi. Docker risolve il problema attraverso la containerizzazione.

Prodotto finale Una macchina virtuale è di per sé la parte utilizzabile dall'utente finale. La tecnologia non è associata a un marchio specifico. È possibile distribuire VM nei data center on-premise o accedervi mediante API come servizio cloud gestito.

Docker è il nome della piattaforma di container open source di proprietà e sotto la gestione della società Docker. Docker è sinonimo di containerizzazione. Il container è l'artefatto, la parte utilizzabile dall'utente finale.

Architettura Una macchina virtuale esegue il proprio kernel e sistema operativo host, insieme alle applicazioni e alle relative dipendenze, come librerie e altri file binari. Un hypervisor esegue il coordinamento tra l'hardware (macchina o server host) e la macchina virtuale. Assegna le risorse hardware fisiche delineate durante l'avvio di un'istanza alla macchina virtuale per il suo uso esclusivo. Su un unico potente server possono esistere più macchine virtuali, gestite da un unico hypervisor, con centinaia di applicazioni in esecuzione su ciascuna di esse.

Un container Docker contiene solo le sue dipendenze. Il software Docker Engine consente la virtualizzazione in Docker. Fornisce il coordinamento tra i container

in esecuzione e il sistema operativo sottostante, a prescindere che si tratti di una macchina fisica o virtuale.

Condivisione delle risorse Sia le macchine virtuali che i container Docker utilizzano il *multiplexing di risorse* o la *condivisione di risorse tra istanze virtualizzate*

Le macchine virtuali richiedono anticipatamente una quantità specifica di risorse dall'hardware e continuano a occupare costantemente tale quantità finché sono in esecuzione.

I container Docker, invece, utilizzano le risorse on demand. Anziché richiedere una quantità specifica di risorse hardware fisiche come nel caso delle macchine virtuali, richiedono semplicemente ciò di cui hanno bisogno dal singolo kernel del sistema operativo. Più container condividono lo stesso sistema operativo. I container Docker dirigono la condivisione delle risorse con i lead del kernel e possono utilizzare meno risorse di sistema rispetto a una VM.

Sicurezza Una VM virtuale esegue un intero sistema operativo, c'è un ulteriore livello di isolamento durante l'esecuzione delle applicazioni. Le VM offrono una maggiore sicurezza a condizione che il sistema operativo disponga di misure di sicurezza rigorose.

Poiché condividono il kernel con il sistema operativo host per ridurre il consumo di risorse, i container Docker sono a rischio se il kernel presenta vulnerabilità. Tuttavia, Docker offre anche molti controlli di sicurezza avanzati.[5]

²Fonte: <https://www.researchgate.net/figure>

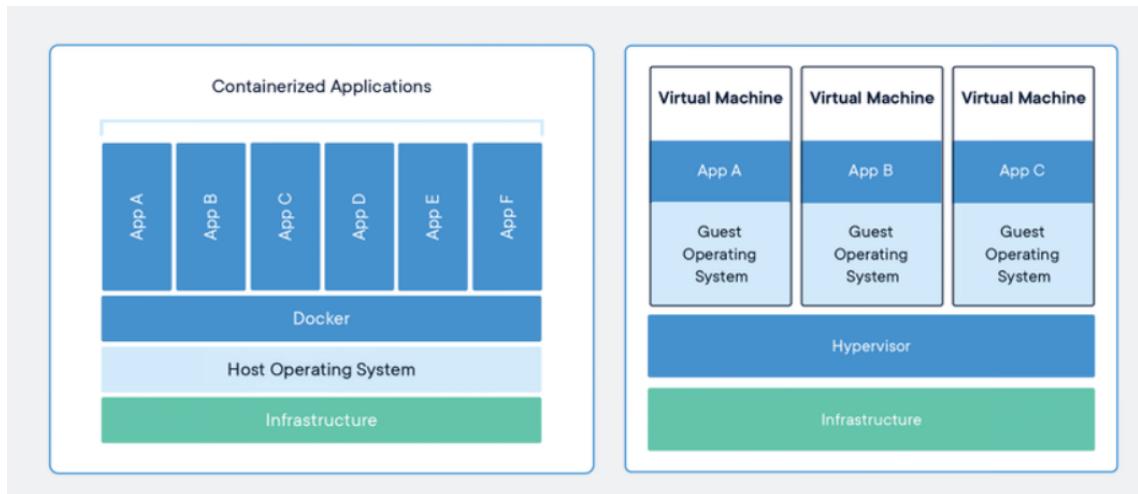


Figura 2.1: Confronto Fra Docker e Macchina virtuale²

Tabella 2.1: Confronto tra Container Docker e Macchine Virtuali³

Caratteristica	Container Docker	VM
In cosa consiste?	Docker è una piattaforma software per creare ed eseguire container Docker. Un container Docker è un'emulazione di un'istanza dello spazio utente, la parte del sistema operativo in cui vengono eseguiti i processi dell'utente.	Un'emulazione di una macchina fisica, incluso l'hardware virtualizzato, che esegue un sistema operativo.
Virtualizzazione	Il container astrae i dettagli del sistema operativo dal codice applicativo.	La VM astrae i dettagli hardware dal codice applicativo.
Obiettivo	Astrare i dettagli hardware e aumentare l'utilizzo dell'hardware.	Migliorare la gestione dell'ambiente applicativo e aumentare la coerenza tra più ambienti.
Gestito da	Docker Engine esegue il coordinamento tra sistema operativo e container Docker.	L'hypervisor esegue il coordinamento tra l'hardware fisico della macchina e le macchine virtuali.
Architettura	Condivide le risorse con il kernel host sottostante.	Esegue kernel e sistema operativo propri.
Condivisione delle risorse	On demand.	Una quantità fissa, impostata nei requisiti di configurazione di un'immagine di macchina virtuale.

Come funziona Docker

Docker Engine Docker Engine è la tecnologia client-server per la creazione e la containerizzazione di applicazioni in Docker. In sostanza, supporta tutte le attività necessarie per l'esecuzione di un'applicazione basata su container:

³Basato su:[5]

Questi sono i componenti principali del motore Docker:

- **Docker Daemon:** *Gestisce le immagini Docker, i container, le reti e i volumi. Inoltre, ascolta le richieste dell'API Docker e le elabora.*
- **Docker Engine REST API:** *Un'API sviluppata da Docker che interagisce con il daemon.*
- **Docker CLI:** *l'interfaccia a riga di comando per comunicare con il daemon Docker.[4]*

Immagini Docker *Le immagini Docker racchiudono tutto ciò che serve per eseguire un'applicazione: il suo codice sorgente, le librerie, gli strumenti e tutte le dipendenze necessarie.* Una volta che uno sviluppatore avvia un'immagine, essa viene eseguita sotto forma di container, o anche più container se necessario. In pratica, l'immagine funge da modello statico, mentre il container rappresenta la sua versione attiva e funzionante.[6]

Container Docker *I container Docker sono le istanze attive in esecuzione delle immagini Docker.* Mentre le immagini Docker sono file di sola lettura, i container sono contenuti attivi, effimeri ed eseguibili. Gli utenti possono interagire con loro e gli amministratori possono modificare le impostazioni e le condizioni utilizzando i comandi Docker.[6]

Docker Compose Docker Compose è uno strumento utilizzato per gestire applicazioni composte da più container, tutti eseguiti sul medesimo host Docker. Il suo funzionamento si basa sulla creazione di un file in formato YAML (.yml), in cui vengono elencati i vari servizi che compongono l'applicazione. Grazie a questo file, è possibile avviare e distribuire tutti i container necessari con un singolo comando.

Poiché la sintassi YAML è neutra rispetto al linguaggio di programmazione, questi file possono essere integrati in progetti sviluppati in Java, Python, Ruby e molti altri linguaggi. Inoltre, Docker Compose consente agli sviluppatori di configurare volumi persistenti per l'archiviazione dei dati, definire i nodi fondamentali dell'infrastruttura applicativa e descrivere in modo chiaro le dipendenze tra i vari servizi.[6]

Registro Docker Un registro Docker è un sistema scalabile di storage e di distribuzione open-source per immagini Docker. Consente agli sviluppatori di tenere traccia delle versioni delle immagini nei repository utilizzando l'assegnazione di tag per l'identificazione. Questo monitoraggio e identificazione vengono eseguiti utilizzando Git, uno strumento di controllo della versione.

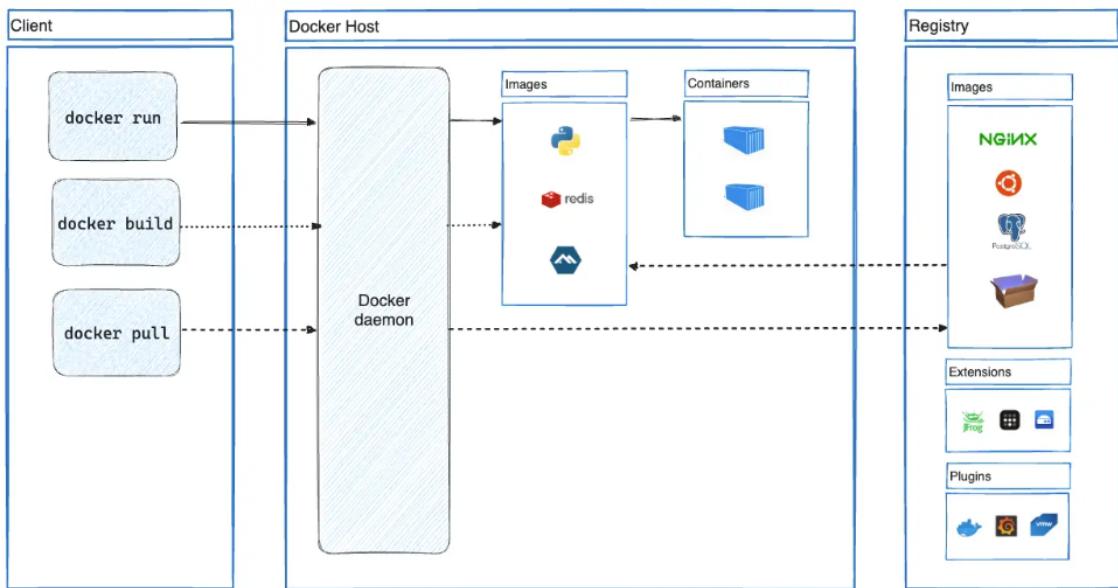


Figura 2.2: Schema architettura Docker⁴

2.2.3 SonarQube

Qualità del software

La qualità del software viene spesso messa in secondo piano o a volte del tutto ignorata ed è per questo che bisogna evidenziare quanto sia importante. Ha un ruolo così centrale da dover essere considerata durante tutta la routine dello sviluppo software; analizziamo il perchè in cinque punti:

- **Robustezza del software:** Un software robusto è in grado di gestire correttamente situazioni inattese o errori senza compromettere l'esecuzione generale del sistema. La presenza di messaggi di errore comprensibili e dettagliati rappre-

⁴Fonte: <https://docs.docker.com/get-started/docker-overview/>

senta un elemento chiave per migliorare l'interazione con l'utente e prevenire comportamenti imprevedibili.

- **Sviluppo sostenibile:** La sostenibilità nello sviluppo software consiste nella capacità dell'applicazione di evolversi nel tempo attraverso modifiche graduali e controllate. Poiché l'ambiente tecnologico è in costante mutamento, un codice ben strutturato facilita l'adattamento a nuove esigenze e tecnologie, evitando riscritture complesse.
- **Trasferibilità:** Un software di qualità è facilmente adattabile a differenti ambienti di esecuzione. Applicare pratiche di sviluppo standardizzate e utilizzare tecnologie portabili permette di distribuire il codice su diverse piattaforme con interventi minimi, aumentando la flessibilità e riducendo i costi di migrazione.
- **Affidabilità e manutenibilità:** Un codice chiaro, ben documentato e organizzato secondo principi di buona progettazione è più facile da comprendere e modificare. L'utilizzo di commenti pertinenti, una sintassi leggibile e una struttura modulare favoriscono il lavoro di squadra e migliorano la manutenibilità a lungo termine.
- **Riduzione del debito tecnico:** Investire nella qualità del codice fin dalle prime fasi di sviluppo consente di minimizzare il cosiddetto debito tecnico, ovvero il lavoro aggiuntivo generato da soluzioni provvisorie o poco curate. Una bassa qualità del codice porta a interventi continui e costosi nel tempo, mentre un codice solido garantisce maggiore stabilità e durata del sistema. [7]

Analisi statica

L'analisi statica del software è il processo di rilevamento di errori e difetti nel codice sorgente di un software. L'analisi statica può essere vista come un processo automatizzato di revisione del codice sorgente. Tra i vari tools di revisione automatica del software vi è **SonarQube**.

- **Individuazione degli errori nel codice** Il tool permette di rilevare bug e problemi potenziali nei programmi prima ancora della fase di esecuzione, migliorando l'affidabilità del software fin dalle prime fasi dello sviluppo.

- **Suggerimenti per la formattazione del codice** Alcuni analizzatori statici, come SonarQube, verificano se il codice sorgente rispetta le convenzioni di formattazione adottate dall'organizzazione. Vengono applicate regole stilistiche ampiamente riconosciute per mantenere omogeneità e leggibilità.
- **Calcolo delle metriche del software** Le metriche permettono di quantificare alcune caratteristiche del codice, come la copertura dei test (*code coverage*), la complessità o la manutenibilità. Strumenti come SonarQube offrono una panoramica numerica utile per monitorare la qualità del progetto.
- **Indipendenza dal compilatore** L'analisi statica è indipendente dal compilatore o dall'ambiente di esecuzione, e consente di individuare problemi nascosti, come comportamenti indefiniti, che potrebbero emergere solo dopo anni o con un cambio di compilatore o delle sue impostazioni di ottimizzazione.[7]

Certificazioni

Ovviamente il codice una volta analizzato può essere certificato in modo da avere anche una prova tangibile del buon sviluppo effettuato da dare al cliente, fra le varie certificazioni a disposizione ci sono quelle *ISO* che sono fra le più richieste e con più valore sia nell'ambito della sicurezza che della scrittura del codice.

2.2.4 DevOps

Continuos Delivery

Nell'ambito dello sviluppo software moderno, il paradigma DevOps rappresenta una delle metodologie più adottate per ottimizzare l'intero ciclo di vita applicativo, dalla scrittura del codice fino al rilascio in produzione. In particolare, i concetti di Continuous Integration (CI) e Continuous Delivery (CD) permettono di automatizzare i processi di integrazione e distribuzione, consentendo il rilascio frequente di nuove funzionalità senza compromettere la stabilità del sistema. Questo approccio si rivela fondamentale per garantire la qualità del software e per supportare esigenze di business in continua evoluzione.[8]

Cos'è DevOps? Il termine DevOps, una combinazione delle parole *Development* (sviluppo) e *Operations* (operazioni), riflette il percorso di integrazione di queste discipline in un unico processo continuo. In sostanza si indica come gestire in maniera integrata le fasi di sviluppo, test e distribuzione del software, con l'obiettivo finale di arrivare a rilasci che siano allo stesso tempo frequenti e stabili.[9]

Le 8 fasi di DevOps

- **Discover** I team devono organizzare workshop per esplorare, organizzare e assegnare la giusta priorità alle idee.
- **Plan** In questa fase si adotta la pratica Agile⁵ per migliorare lo sviluppo
- **Build** Si utilizza *Git* è un sistema di controllo delle versioni gratuito e open source. Offre eccellente supporto per la creazione di branch, il merge e la riscrittura della cronologia dei repository, il che ha portato a numerosi flussi di lavoro e strumenti potenti e innovativi per il processo di sviluppo.
- **Test** La continuous integration (CI) consente a più sviluppatori di contribuire a un unica repository condivisa. Eseguito il merge delle modifiche apportate al codice, vengono utilizzati test automatizzati per garantire la correttezza del codice prima dell'integrazione. Eseguendo merge e test del codice, i team di sviluppo acquisiscono fiducia nella qualità e prevedibilità del codice dopo la distribuzione.
- **Deploy** La continuous delivery (CD) consente ai team di rilasciare in produzione funzioni in modo automatico e di frequente. Si ha anche la possibilità di aggiungere flag delle funzioni, rilasciando nuovo codice agli utenti in modo costante e metodico anziché tutto insieme in una volta. Questo approccio migliora la velocity, la produttività e la sostenibilità dei team di sviluppo software.
- **Operate** Gestisce il rilascio *end-to-end* dei servizi IT ai clienti. Ciò include le pratiche coinvolte nelle attività di progettazione, implementazione, configura-

⁵è un approccio iterativo alla gestione dei progetti e allo sviluppo del software che aiuta i team a suddividere il lavoro in parti più piccole per offrire valore incrementale.

zione, distribuzione e manutenzione dell'intera l'infrastruttura IT alla base dei servizi dell'organizzazione.

- **Observe** Identifica e risolve rapidamente i task che incidono su tempi di attivit, velocit e funzionalit del prodotto. Informa automaticamente il team di modifiche, azioni ad alto rischio o errori, in modo da poter mantenere i servizi attivi.
- **Continuos Feedback** I team devono valutare ogni rilascio e creare report per migliorare i rilasci futuri. Raccogliendo feedback continui, i team possono migliorare i processi e incorporare i feedback dei clienti per migliorare i rilasci successivi.[9]

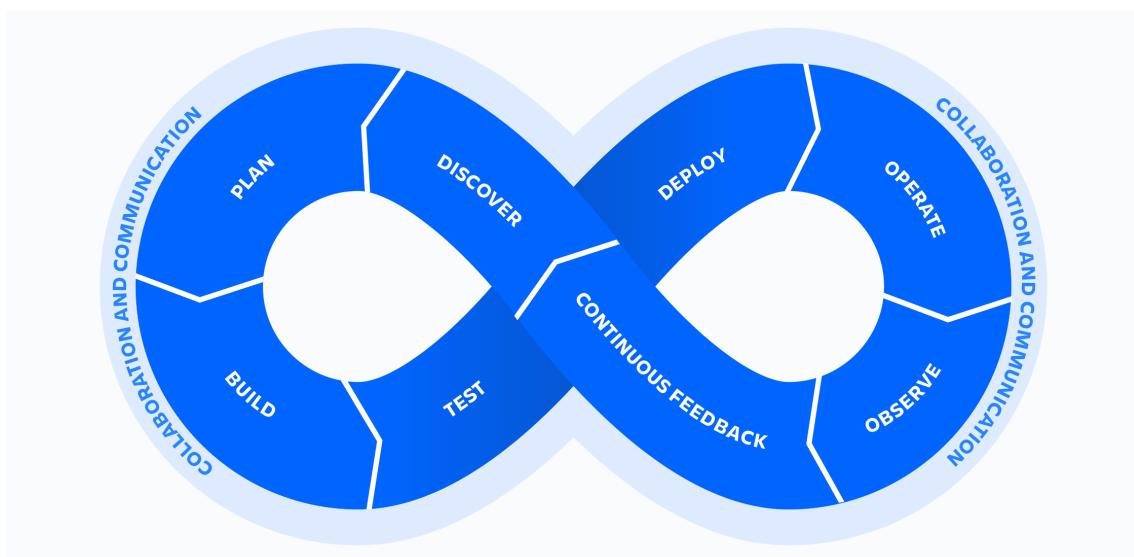


Figura 2.3: Ciclo di Vita del DevOps⁶

Vantaggi

- **Rapidit** Chi adotta il modello DevOps riesce ad effettuare rilasci pi frequenti e con maggiore affidabilit. Secondo il report "State of DevOps 2019" di DORA, i gruppi con le migliori performance rilasciano il software 208 volte pi spesso e 106 volte pi velocemente rispetto a quelli meno efficienti. La continuous delivery, basata su strumenti automatizzati, facilita la creazione, verifica e distribuzione del codice.

⁶Fonte: <https://www.atlassian.com/it/devops>

- **Maggiore collaborazione** DevOps promuove una cultura in cui sviluppatori e team operativi lavorano a stretto contatto, condividendo responsabilità. Questo approccio riduce i tempi di consegna tra le fasi e permette di scrivere codice meglio adattato all'ambiente di destinazione.
- **Rilasci più frequenti** L'incremento della velocità e della frequenza di distribuzione permette di migliorare costantemente i prodotti. Rilasciare rapidamente nuove funzionalità o correggere bug offre un vantaggio competitivo sul mercato.
- **Affidabilità e qualità** Tecniche come la continuous integration e la continuous delivery garantiscono che le modifiche siano sicure e funzionanti, contribuendo ad aumentare la qualità complessiva del software. Il monitoraggio continuo consente inoltre di valutare le prestazioni in tempo reale.
- **Integrazione della sicurezza** Con il modello DevOps, la sicurezza diventa parte integrante del ciclo di sviluppo. L'inserimento di controlli di protezione e test di sicurezza automatizzati all'interno dei *workflow* DevOps e Agile assicura che il software sia sicuro fin dalle prime fasi.[9]

Pipeline

Una pipeline di integrazione e distribuzione continua (CI/CD) è una serie di fasi predefinite che gli sviluppatori seguono per rilasciare nuove versioni del software. Queste pipeline sono progettate per ottimizzare l'intero processo di sviluppo tramite l'automazione, rendendo il rilascio del software più fluido ed efficiente.

Automatizzando le attività nelle fasi di sviluppo, test, rilascio e monitoraggio, i team possono scrivere codice di alta qualità in modo più veloce e sicuro. I test automatizzati, inoltre, aiutano a rilevare in anticipo dipendenze o problemi, evitando ritardi nelle fasi successive. Anche se ogni fase può essere svolta manualmente, l'automazione resta l'elemento chiave che distingue le pipeline CI/CD.

Queste pipeline portano benefici sia alle soluzioni basate su macchine virtuali che a quelle costruite su container e infrastrutture cloud-native. L'integrazione rapida

delle modifiche consente ai team di adattarsi prontamente ai cambiamenti aziendali e ai suggerimenti degli utenti, con impatti positivi sull'esperienza finale.

Le pipeline CI/CD possono integrare strumenti per compilare il codice, eseguire test unitari, analisi statica, controlli di sicurezza e generazione di file eseguibili. Nei contesti containerizzati, includono anche la creazione di immagini container pronte per essere distribuite in ambienti cloud ibridi.[3]

Jenkins

Per questi obiettivi vi sono molti tools a disposizione tra i quali **Jenkins** che è un server di automazione open source utilizzato per implementare processi di CI/CD all'interno di pipeline DevOps. Consente di automatizzare le fasi di build, test e deployment del software, migliorando l'efficienza e la qualità del ciclo di sviluppo.



Figura 2.4: Jenkins⁷

Pipeline implementate su Jenkins

All'interno di una pipeline DevOps, **Jenkins** svolge il ruolo di orchestratore, coordinando in modo automatico le attività principali del ciclo di sviluppo software, come la compilazione del codice, l'esecuzione dei test e il rilascio in ambienti controllati.

Attraverso l'uso della *Jenkins Pipeline*, un'estensione basata su plugin, è possibile definire l'intera sequenza di operazioni utilizzando un file chiamato *Jenkinsfile*.

⁷Fonte: <https://www.webasha.com/courses/cicd-jenkins-certification-training>

Questo approccio, noto come *Pipeline as Code*, consente di descrivere il processo di integrazione e distribuzione continua direttamente nel codice sorgente del progetto.

Tale modalità introduce numerosi vantaggi, tra cui una migliore tracciabilità delle modifiche alla pipeline, la possibilità di versionare il processo stesso insieme all'applicazione, e una maggiore coerenza tra ambienti differenti. In questo modo, ogni aggiornamento alla logica di deploy può essere sottoposto a revisione e integrato nel controllo versione, al pari del codice applicativo. [10]

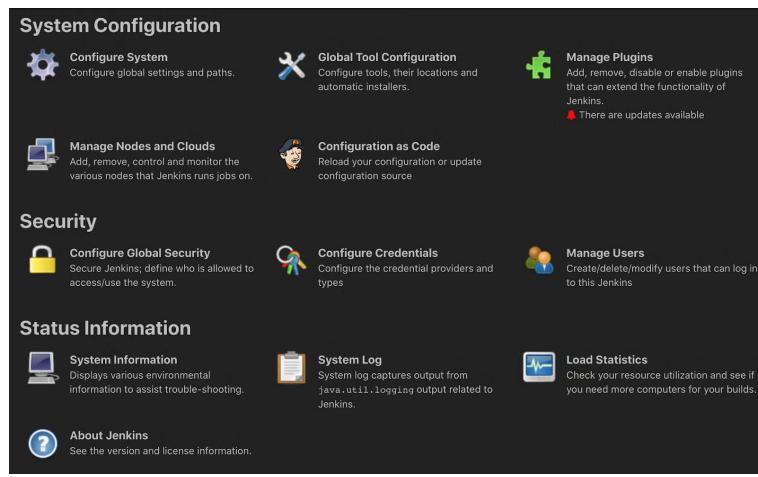


Figura 2.5: Settings di Jenkins⁸

2.2.5 AWS - Amazon Web Services

Introduzione

Con l'evoluzione digitale e la crescente domanda di potenza computazionale, archiviazione e disponibilità dei servizi IT, i data center tradizionali hanno iniziato a mostrare i loro limiti. Le aziende, nel tempo, hanno investito in server fisici, reti e sistemi di backup, espandendo continuamente le proprie infrastrutture locali. Tuttavia, questa crescita non è sostenibile all'infinito.

I principali problemi riscontrati nei data center saturi sono:

- **Spazio fisico limitato:** le sale server non possono crescere oltre certi limiti senza importanti investimenti strutturali.

⁸Fonte: <https://cd.foundation/blog/2020/08/04/introducing-the-jenkins-dark-theme>

- **Costi elevati:** manutenzione, aggiornamenti hardware, consumo energetico e personale specializzato richiedono risorse significative.
- **Scalabilità rigida:** adattarsi rapidamente a picchi di domanda è difficile; serve acquistare e installare nuovi server.
- **Rischi di downtime:** guasti o manutenzioni possono compromettere la disponibilità dei servizi.
- **Lentezza nell'innovazione:** l'infrastruttura rigida rallenta l'adozione di nuove tecnologie e soluzioni digitali.

È in questo scenario che emerge una nuova esigenza: un'infrastruttura flessibile, scalabile, accessibile on-demand e a consumo. Da qui nasce il *cloud computing*.

Cloud Computing

Il cloud computing è la fornitura su richiesta di potenza di calcolo, database, archiviazione, applicazioni e altre risorse IT tramite una piattaforma di servizi cloud via Internet. Che si tratti di gestire applicazioni per la condivisione di immagini usate da milioni di utenti o di supportare funzioni aziendali fondamentali, una piattaforma cloud consente di accedere rapidamente a risorse informatiche scalabili e convenienti. Grazie al cloud computing, non è più necessario investire in costosi hardware iniziali o dedicare tempo alla loro manutenzione. Al contrario, si possono ottenere con facilità e precisione le risorse di calcolo necessarie, sia per sviluppare una nuova idea innovativa che per gestire l'infrastruttura IT aziendale.

Le risorse sono disponibili quasi in tempo reale e si pagano solo in base al consumo effettivo. Il cloud offre un modo pratico per accedere a server, storage, database e molti altri servizi applicativi direttamente tramite Internet. Piattaforme come Amazon Web Services (AWS) si occupano della gestione e manutenzione dell'infrastruttura fisica, mentre gli sviluppatori possono configurare e usare ciò che serve attraverso un'interfaccia web intuitiva.

Il cloud computing offre un modo semplice per accedere a server, storage, database e un'ampia gamma di servizi applicativi su Internet. Una piattaforma di servizi cloud come Amazon Web Services possiede e mantiene l'hardware connesso alla

rete necessario per questi servizi applicativi, mentre gli sviluppatori fornisciono e utilizzano ciò di cui hanno bisogno tramite un'applicazione web.[11]

Infrastruttura

L'infrastruttura Cloud AWS è costruita attorno a *Regioni AWS*. Una Regione AWS è una posizione fisica nel mondo in cui abbiamo più zone di disponibilità. Le zone di disponibilità sono composte da uno o più data center singoli provvisti di alimentazione, rete e connettività ridondanti, ognuno in una struttura separata. Queste zone di disponibilità offrono la possibilità di utilizzare applicazioni e database di produzione ad alta disponibilità, tolleranti ai guasti e scalabili di quanto sarebbe possibile da un singolo data center.

Sicurezza

I clienti AWS, traggono vantaggio da un data center e da un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza. La sicurezza nel cloud è molto simile alla sicurezza nei data center locali, solo senza i costi di manutenzione di strutture e hardware. Nel cloud, non è necessario gestire server fisici o dispositivi di archiviazione. In AWS gli ambienti vengono continuamente controllati, con certificazioni da parte di organismi di accreditamento in tutte le aree geografiche. Nell'ambiente AWS, è possibile sfruttare gli strumenti automatizzati per l'inventario degli asset e il reporting degli accessi privilegiati.

Vantaggi

- **Mantenere i dati al sicuro:** l'infrastruttura AWS mette in atto solide misure di protezione per proteggere la privacy. Tutti i dati sono archiviati in AWS data center altamente sicuri.
- **Soddisfa i requisiti di conformità:** AWS gestisce dozzine di programmi di conformità nella sua infrastruttura. Ciò significa che alcuni segmenti della conformità sono già stati completati.
- **Risparmio costi:** i costi vengono ridotti utilizzando AWS i data center. Mentre gli standard più alti senza dover gestire la struttura.

- **Scalabilità rapida:** la sicurezza Cloud AWS si adatta all'utilizzo. Indipendentemente dalle dimensioni dell'azienda.[11]

Tipi di Cloud Computing

AWS mette a disposizione dei suoi clienti oltre 200 servizi che vengono divisi in 4 categorie.

Infrastructure as a Service - IaaS Questa categoria include servizi che forniscono infrastrutture virtuali, come server, reti e storage, che l'utente può configurare e controllare in autonomia. Esempi tipici di servizi IaaS sono **EC2**, **EBS** e **VPC**.

Platform as a Service - PaaS In questo modello, AWS si occupa della gestione dell'infrastruttura sottostante, permettendo agli sviluppatori di focalizzarsi esclusivamente sullo sviluppo del software e sulla logica dell'applicazione. Tra i servizi PaaS più noti troviamo **Elastic Beanstalk** e **App Runner**.

Function as a Service - FaaS Rappresenta il paradigma serverless: il codice viene eseguito automaticamente in risposta a eventi, senza che lo sviluppatore debba occuparsi di server o infrastrutture. Il principale servizio FaaS di AWS è **AWS Lambda**.

Software as a Service - SaaS Il modello prevede applicazioni software completamente gestite dal provider e accessibili via Internet. Sebbene AWS non offra molte soluzioni SaaS di sua proprietà, numerose piattaforme SaaS vengono sviluppate su infrastruttura AWS. Un esempio celebre è **Netflix**⁹, che fornisce il suo servizio di streaming appoggiandosi all'infrastruttura cloud di AWS.[12]

⁹<https://www.netflix.com/it/>

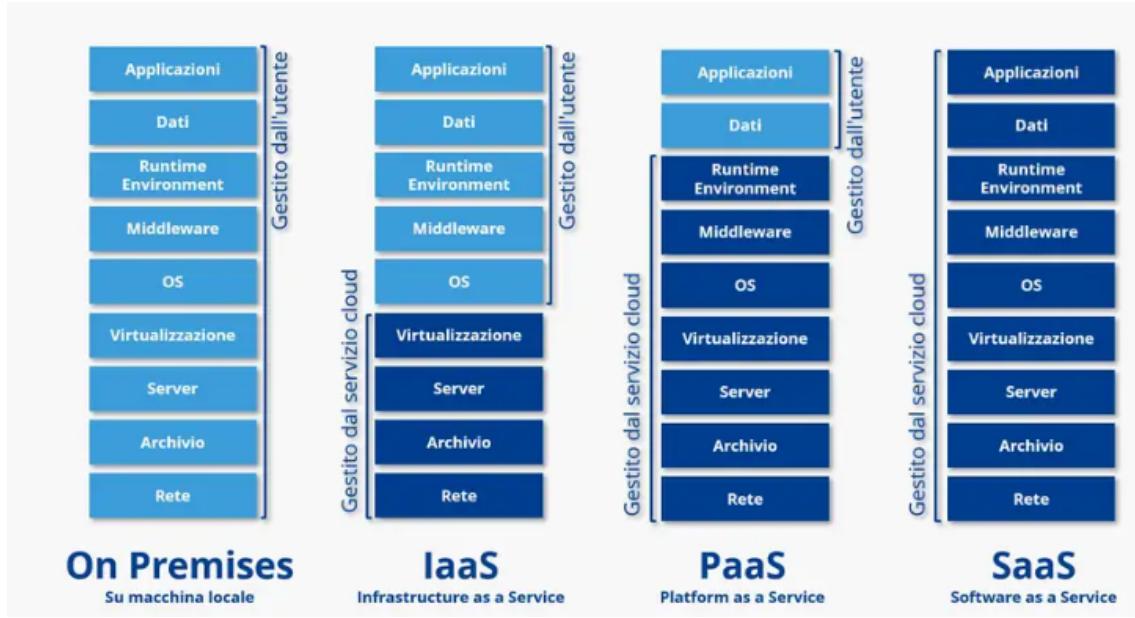


Figura 2.6: AWS - Tipi di Cloud¹⁰

Costi del Cloud in AWS

AWS adotta un modello di prezzo basato sul consumo (pay-as-you-go), che si fonda su tre principi fondamentali:

Calcolo Si paga per il tempo effettivo di utilizzo delle risorse computazionali, come le istanze EC2. La tariffazione può essere oraria o basata sul numero di richieste, a seconda del servizio utilizzato.

Archiviazione Si paga in base alla quantità di dati archiviati nel cloud. Ad esempio, Amazon S3 addebita un costo mensile per gigabyte memorizzato, con tariffe che variano in base alla classe di storage scelta.

Trasferimento Dati Trasferimento dati in entrata (IN): gratuito nella maggior parte dei casi.

Trasferimento dati in uscita (OUT): comporta costi che variano in base alla regione di origine e al volume di dati trasferiti.

¹⁰Fonte: <https://www.ionos.it/digitalguide/server/know-how/caas-i-migliori-servizi-contenitore-a-confronto/>

Questo modello di pricing risolve i problemi legati ai costi elevati dell'IT tradizionale, poiché consente di pagare solo per le risorse effettivamente utilizzate, senza necessità di investimenti iniziali significativi o impegni a lungo termine.[11]

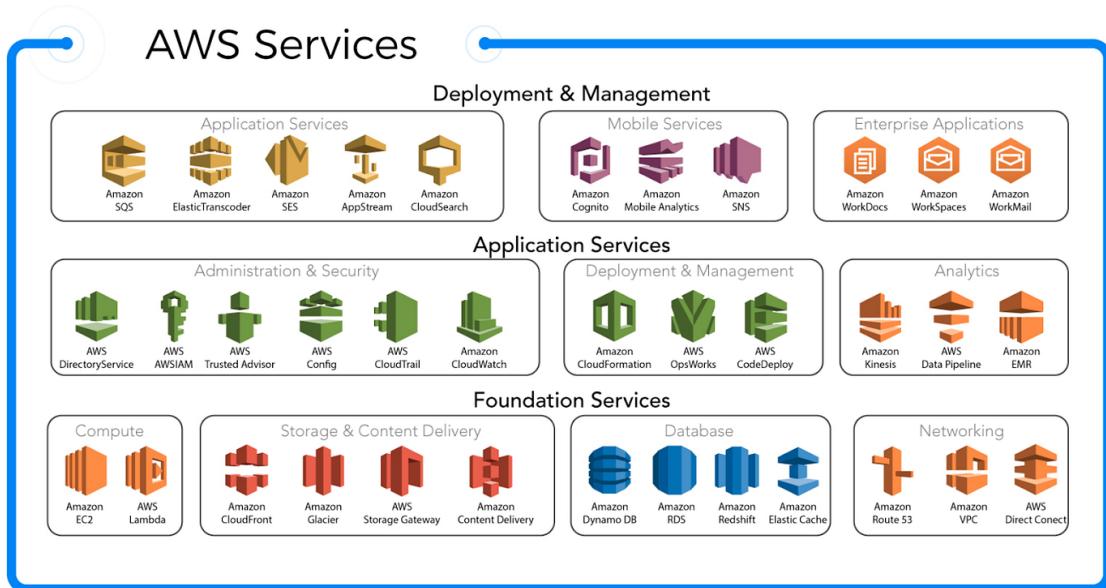


Figura 2.7: AWS services¹¹

¹¹Fonte: <https://eishatirraazia.medium.com/aws-a-superior-cloud-solution-804ef777d5e9>

CAPITOLO 3

Sviluppo Applicativo

3.1 Progettazione di Chefbook

Il progetto *di Tesi* è stato sviluppato durante il tirocinio formativo presso l'azienda *System Management S.p.A* che ha proposto di approfondire l'ambito dello sviluppo software attuale nella sfumatura del Cloud con lo sviluppo di un Social Network indirizzato agli chef in linguaggio Java. In questo capitolo si approfondiranno tutte le fasi di progettazione per lo sviluppo dell'applicativo.

3.1.1 Analisi del contesto

Il primo passo è stata un'analisi del mercato dei social network con l'obiettivo di creare uno da zero per un'esigenza di mercato da parte di utenti che non avevano uno spazio online nel quale confrontarsi. Per lo sviluppo era necessario individuare gli attori, quindi gli eventuali utenti che usufruiranno di questa piattaforma, si sono divisi in 2 utenti: autori (quindi gli chef) ed utenti visualizzatori che non hanno bisogno di alcun tipo di registrazione per visualizzare le ricette poste dagli chef.

3.1.2 Requisiti funzionali

1. **Registrazione utenti** – Gli utenti devono poter creare un account fornendo i propri dati.
2. **Gestione delle ricette** – Il sistema deve consentire la creazione, la modifica, la visualizzazione e la cancellazione delle ricette.
3. **Condivisione delle ricette** – Le ricette devono poter essere rese pubbliche o private, a discrezione dell’utente.
4. **Gestione della privacy** – Deve essere possibile controllare chi può accedere alle proprie ricette tramite impostazioni di privacy.
5. **Trasferimento ricette** – Deve essere implementato un meccanismo in due fasi per trasferire ricette da un utente ad un altro.
6. **Ambiente sicuro e strutturato** – Il sistema deve garantire protezione dei dati personali e sicurezza nell’utilizzo della piattaforma.

3.1.3 Requisiti non funzionali

1. **Scalabilità** – Il sistema deve essere progettato per gestire una crescita del numero di utenti e ricette.
2. **Sicurezza** – Protezione delle informazioni degli utenti e delle ricette con meccanismi di autenticazione e autorizzazione.
3. **Usabilità** – L’applicazione deve essere intuitiva e semplice da usare per chef professionisti e appassionati.
4. **Performance** – Tempi di risposta rapidi per la consultazione e la gestione delle ricette.
5. **Portabilità** – Prima implementazione in ambiente on-prem, con possibilità di migrazione su cloud.
6. **Manutenibilità** – Il codice deve essere organizzato per facilitare future estensioni e aggiornamenti.

3.1.4 Use Case

Negli use case si vanno ad elencare le varie casistiche che si possono creare sulle funzionalità sviluppate in relazione agli attori presi in considerazione dalla piattaforma.

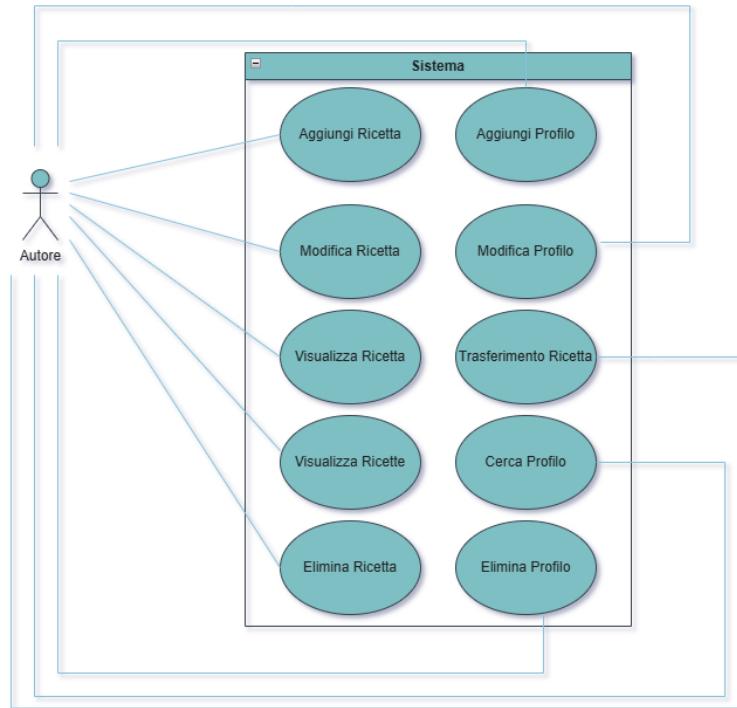


Figura 3.1: Use Case Diagram

Identificativo	<i>Aggiungi ricetta</i>	Data	30/06/2025
<i>UC_AUTORE_1</i>		Vers.	1.0
Descrizione	Il seguente UC descrive l'aggiunta di una nuova ricetta		
Attore Principale	Autore		
Attori secondari	NA		
Entry Condition	L'autore deve essere autenticato.		
Exit condition	La registrazione della ricetta è avvenuta con successo.		
On success			
Exit condition	La ricetta non è stata registrata.		
On failure	L'autore dovrà provvedere a reinserire la ricetta.		
Rilevanza/User Priority	Alta		
Frequenza stimata	1 al giorno		
FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO			
1	Autore	Entra nella sua area social privata e clicca sulla voce: "aggiungi ricetta"	
2	Sistema	Reindirizza l'utente alla pagina dedicata alla registrazione di una nuova ricetta.	
		Per la ricetta si richiede all'autore: il nome, la foto, la descrizione di essa (prodotti utili e procedimenti) e se tenerla privata oppure no	
3	Autore	Quando ha finito di compilare i campi necessari clicca su "Aggiungi ricetta".	
4	Sistema	Si accerta che i campi inseriti siano validi, quindi, non vuoti e per le stringhe non più lunghe di 255 caratteri. Nel caso sia tutto corretto mostra un messaggio di aggiunta andata a buon fine.	
Scenario/Flusso di eventi Alternativo: Autore non registrato			
2.a1	Sistema	Utente reindirizzato alla pagina di registrazione perché non autenticato	
Scenario/Flusso di eventi di Errore: Aggiunta della ricetta non riuscita			
4.a1	Sistema	Informa l'utente che l'aggiunta della ricetta non è andata a buon fine; dopodiché, lo invita a riprovare in un altro momento. Termina con insuccesso.	

Figura 3.2: Use Case Aggiunta ricetta

Identificativo	<i>Elimina ricetta</i>	Data	30/06/2025
<i>UC_AUTORE_2</i>		Vers.	1.0
Descrizione	Il seguente UC descrive l'eliminazione di una nuova ricetta	Autore	Gianmarco Riviello
Attore Principale	Autore		
Attori secondari	NA		
Entry Condition	L'autore deve essere autenticato.		
Exit condition	L'eliminazione della ricetta è avvenuta con successo.		
On success			
Exit condition	La ricetta non è stata eliminata.		
On failure	L'autore dovrà provvedere ad eliminare nuovamente la ricetta.		
Rilevanza/User Priority	Media		
Frequenza stimata	1 al mese		
FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO			
1	Autore	Entra nella sua area social privata e clicca sulla voce: "elimina ricetta"	
2	Sistema	Reindirizza l'utente alla pagina dedicata alla visualizzazione di tutte le sue ricette.	
3	Autore	Seleziona il pulsante "elimina ricetta" sulla ricetta che desidera eliminare	
4	Sistema	Mostra riepilogo della ricetta selezionata.	
5	Autore	Clicca pulsante conferma per eliminare la ricetta.	
6	Sistema	Elimina la ricetta dal Database e dalla lista di ricette dell'autore.	
Scenario/Flusso di eventi Alternativo: Autore non registrato			
2.a1	Sistema	Utente reindirizzato alla pagina di registrazione perché non autenticato	
Scenario/Flusso di eventi di Errore: Eliminazione della ricetta non riuscita			
6.a1	Sistema	Informa l'autore che l'eliminazione della ricetta non è andata a buon fine; dopodiché, lo invita a riprovare in un altro momento. Termina con insuccesso.	

Figura 3.3: Use Case Elimina ricetta

Identificativo	<i>Modifica ricetta</i>	Data	30/06/2025
<i>UC_AUTORE_3</i>		Vers.	1.0
Descrizione	Il seguente UC descrive la modifica di una ricetta esistente		
Attore Principale	Autore		
Attori secondari	NA		
Entry Condition	L'autore deve essere autenticato AND la ricetta deve esistere nel DB		
Exit condition	La modifica della ricetta è avvenuta con successo.		
On success			
Exit condition	La ricetta non è stata modificata.		
On failure	La ricetta non viene modificata.		
Rilevanza/User Priority	Media		
Frequenza stimata	1 al mese		
FLUSSO DI EVENTI PRINCIPALE/MAIN SCENARIO			
1	Autore	Entra nella sua area social privata e clicca sulla voce: "visualizza ricette"	
2	Sistema	Reindirizza l'utente alla pagina dedicata alla visualizzazione di tutte le ricette dell'autore.	
3	Autore	Clicca modifica sulla ricetta che vuole modificare.	
4	Sistema	Carica la pagina della ricetta con tutti i dati salvati precaricati ed editabili.	
5	Autore	Modifica uno o più campi. Clicca sul tasto "modifica"	
6	Sistema	Richiede la conferma delle modifiche.	
7	Autore	Clicca sul tasto "conferma"	
8	Sistema	Controlla se i dati rispettano i vincoli (per le stringhe max 255 caratteri e tutti non vuoti). Conferma la modifica	
Scenario/Flusso di eventi Alternativo: Autore non registrato			
2.a1	Sistema	Utente reindirizzato alla pagina di registrazione perché non autenticato.	
Scenario/Flusso di eventi di Errore: Modifica della ricetta non riuscita			
8.a1	Sistema	Informa l'utente che la modifica della ricetta non è andata a buon fine; dopodiché, lo invita a riprovare in un altro momento. Termina con insuccesso.	

Figura 3.4: Use Case Modifica ricetta

3.2 Scelte progettuali

Definiti i requisiti dei quali necessita la piattaforma e gli attori, c'è bisogno di definire le scelte progettuali.

3.2.1 Architettura

Il progetto è stato realizzato utilizzando un'architettura di tipo monolitico con pattern Client-Server (di cui l'applicativo ricopre il lato Server).

Lo stile architettonico adottato è il RESTful (REST API) con separazione tra componenti back-end e front-end.

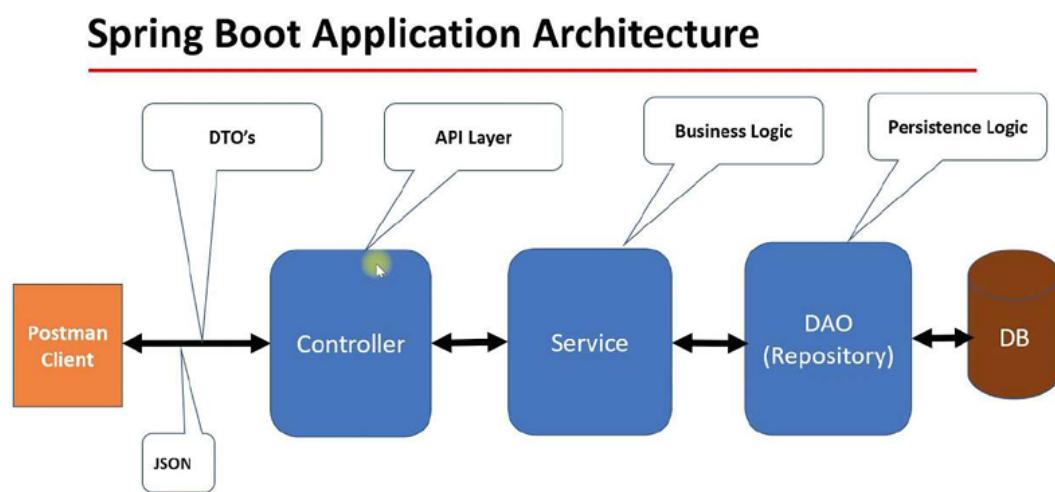


Figura 3.5: Architettura Spring - RESTful API¹

All'interno del codice sorgente, quindi, l'applicativo si presenta tripartito secondo lo schema layered di un architettura backend 3-Tier (Controller, Service, Repository).

3.2.2 Database

Per la gestione dei dati è stato adottato un database relazionale, nello specifico MySQL, una delle soluzioni più consolidate e apprezzate nel panorama dello sviluppo software. La scelta di un sistema relazionale nasce dall'esigenza di organizzare i dati in modo strutturato e coerente, seguendo un modello basato su tabelle e relazioni tra

¹Fonte: <https://www.youtube.com/watch?v=mS1L96GqwSU>

entità e favorendo l’ORM² da parte di Spring. Questo approccio consente una forte coerenza dei dati, facilita l’integrità referenziale e garantisce una maggiore chiarezza nella progettazione dello schema dati.

Quindi, l’adozione di MySQL come sistema di gestione dati relazionale rappresenta una scelta solida e pragmatica. Offre un ambiente maturo, affidabile e ben documentato, che consente di costruire applicazioni robuste, performanti e pronte a scalare nel tempo, sia dal punto di vista tecnico che funzionale.

3.2.3 Autenticazione

L’autenticazione dell’utente è stata realizzata facendo leva sulla tecnologia JSON Web Token (JWT)³, una soluzione moderna e largamente adottata nello sviluppo di applicazioni web e mobile. Il vantaggio principale di questo approccio è la possibilità di gestire l’autenticazione in maniera stateless, cioè senza la necessità di mantenere uno stato lato server per ogni utente connesso. Di fatti, una volta che l’utente effettua il login e le sue credenziali vengono verificate con successo, il server genera un token JWT.

Questo token viene poi inviato dal client nelle richieste successive, generalmente all’interno dell’header HTTP di tipo Authorization. Il server, a ogni richiesta, non fa altro che verificare la validità del token ricevuto, senza doversi preoccupare di gestire sessioni in memoria. Ciò rende l’architettura molto più semplice da scalare, soprattutto in ambienti distribuiti o basati su microservizi, dove è importante evitare colli di bottiglia legati alla gestione centralizzata dello stato.

Per quanto riguarda la protezione delle credenziali degli utenti, è stato adottato BCrypt, un algoritmo di hashing progettato proprio per questo scopo. A differenza di altri algoritmi più generici, BCrypt è sviluppato per rendere estremamente difficile un attacco di tipo brute-force.

²Object Relational Mapping

³Fonte: <https://psicografici.com/jwt-json-web-token/>

3.3 Scelte implementative

Le scelte implementative del progetto ChefBook sono state guidate dalla necessità di creare un sistema scalabile, sicuro e manutenibile, basato su Spring Boot con un’architettura monolitica RESTful.

L’implementazione segue un approccio layered, separando la logica applicativa nei livelli Controller, Service e Repository, con il supporto di DTO (Data Transfer Object) per la comunicazione tra i livelli.

L’applicazione si basa su Spring Boot 3.3.6 e Java 17 (Amazon Corretto).

Per l’esposizione delle API REST si è scelto di utilizzare la dipendenza Web Spring, che include il supporto per Spring MVC e la gestione delle richieste HTTP.

Le API sono documentate tramite SpringDoc OpenAPI⁴ ed esposte per eventuali test su endpoint Swagger.

Per la persistenza, l’applicazione utilizza Spring Data JPA con Hibernate⁵, che consente di interfacciarsi con MySQL seguendo il paradigma ORM, mentre il livello Repository sfrutta l’interfaccia JpaRepository per ridurre al minimo il codice *boilerplate*⁶ per le operazioni CRUD.

Per migliorare la gestione degli oggetti trasferiti tra frontend e backend, si è scelto di utilizzare DTO con MapStruct per la mappatura automatica tra entità e DTO, migliorando così le performance rispetto alla conversione manuale.

La sicurezza è stata gestita tramite Spring Security, con un meccanismo di autenticazione basato su JWT (JSON Web Token)

Per la registrazione degli eventi e il debug, il progetto utilizza *Log4J* con *SLF4J*, il quale consente una gestione avanzata dei log.

3.3.1 Design Pattern

La maggior parte dei design pattern utilizzati è diretta conseguenza dell’implementazione e dell’uso del framework Spring.

⁴Fonte: <https://springdoc.org/>

⁵framework open source per Java che facilita la persistenza dei dati, traducendo oggetti Java in record di database e viceversa

⁶si riferisce a porzioni di codice standard, ripetitive e poco modificabili

Di seguito un elenco di quelli utilizzati maggiormente:

1. **Layered Architecture Pattern** – Separazione in Controller, Service e Repository.
2. **Model-View-Controller (MVC)** – Struttura organizzata per le API REST.
3. **Singleton Pattern** – Gestione dei bean Spring come singleton.
4. **Factory Pattern** – Creazione e gestione automatica dei bean da parte di Spring.
5. **Dependency Injection (DI)** – Iniezione automatica delle dipendenze.
6. **Data Transfer Object (DTO) Pattern** – Separazione tra entità JPA e oggetti trasferiti via API.
7. **Repository Pattern** – Gestione dell’accesso ai dati con Spring Data JPA.
8. **Strategy Pattern** – Applicato in autenticazione e sicurezza con Spring Security.
9. **Builder Pattern** – Creazione di oggetti immutabili con Lombok.
10. **Flyweight Pattern** – Per la gestione delle entità condivise sul Database.
11. **Bridge Pattern** – Per la gestione delle relazioni tra Contratto e Implementazione.

3.4 Sviluppo Software

3.4.1 Entità

Sono state individuate 5 **entità**, ognuna di questa utilizza *Lombok* per la creazione dei metodi basici di Java: autore, ricetta, ruolo, utente e richiesta di trasferimento.

Ogni **autore** è caratterizzato da: id, email, nome, cognome, sesso e ricette da lui scritte.

La **ricetta** è caratterizzata da: id, nome, descrizione, autore proprietario e richiesta di trasferimento.

Un **ruolo** è contraddistinto da client e user.

Un **utente**, viene utilizzato per l’accesso alla piattaforma, è caratterizzato da: id, username, password e ruolo.

La **richiesta di trasferimento**, infine, è descritta da: id, ricetta in questione, mittente, destinatario, token e stato della richiesta. Ogni entità per essere trasferita all'interno dell'architettura tra controller e service utilizza il DTO con mappatura automatica di MapStruct che genera le classi mapper in java automaticamente creando metodi che convertono da DTO ad Entity e viceversa.

Esempio su Autore

```

9  @Entity
10 @Table
11 @Getter
12 @Setter
13 @NoArgsConstructor
14 @AllArgsConstructor
15 @Builder
16 public class Autore {
17     @Id
18     @GeneratedValue(strategy = GenerationType.IDENTITY)
19     private int id;
20     @Column(unique=true)
21     private String email;
22     private String nome;
23     private String cognome;
24     private String sesso;
25     @OneToMany(mappedBy = "autore")
26     private List<Ricetta> ricette;
27 }

```

Entità Autore

```

9  @Data 85 usages
10 @NoArgsConstructor
11 @AllArgsConstructor
12 @Builder
13 public class AutoreDto {
14     private int id;
15     private String nome;
16     private String cognome;
17     private String sesso;
18     @Pattern(regexp = "[a-zA-Z0-9]+@[a-zA-Z]+\.[a-zA-Z]{2,3}", message = "Pattern errato")
19     private String email;
20     private String password;
21 }

```

Autore DTO

```

@Mapper
public abstract class AutoreMapper {
    public abstract AutoreDto autoreToAutoreDto(Autore autore);
    public abstract Autore autoreDtoToAutore(AutoreDto dto);
    public abstract List<AutoreDto> autoriToAutoriDto(List<Autore> autori);
    @Mapping(target = "username", source = "dto.email")
    public abstract User autoreDtoToUser(AutoreDto dto);
}

```

Mapper di autore per DTO

Figura 3.6: Autore, relativo DTO e mapper

3.4.2 Controller

Ogni controller ha un end point esposto per essere raggiunto dalla parte front end. Per la trasmissione con il front end viene utilizzato il DTO in formato JSON. Di fatti

il controller è la prima parte dell’architettura addetta alla recezione delle richieste HTTP e alle risposte con i vari codici a seconda del pacchetto ricevuto.

```

15  @Slf4j
16  @RestController
17  @SecurityRequirement(name = "bearer-auth")
18  @RequestMapping(value = "/autori")
19  public class AutoreController {
20
21      @Autowired
22      private AutoreService autoreService;
23
24      @PostMapping(value = "/signup")
25      @ResponseStatus(code = HttpStatus.CREATED)
26      public void addAutore(@RequestBody @Valid AutoreDto dto) { autoreService.addAutore(dto); }
27
28      @DeleteMapping(value = "/{id}")
29      public ResponseEntity<Void> removeAutore(@PathVariable int id) {
30          return autoreService.removeAutore(id) ?
31              ResponseEntity.status(HttpStatus.OK).body(null) :
32              ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
33      }
34
35      @PutMapping(value = "/{id}")
36      public ResponseEntity<Void> editAutore(@RequestBody AutoreDto dto) {
37          return autoreService.editAutore(dto) ?
38              ResponseEntity.status(HttpStatus.OK).body(null) :
39              ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
40      }
41
42      @GetMapping(value = "/name")
43      public AutoreDto getAutoreByName(@RequestParam String value) { return autoreService.getAutoreByName(value); }
44
45      @GetMapping(value = "/email")
46      public AutoreDto getAutoreByEmail(@RequestParam String value) { return autoreService.getAutoreByEmail(value); }
47
48      @GetMapping(value = "/all")
49      public List<AutoreDto> getAllAutori() { return autoreService.getAllAutori(); }
50
51  }
52
53
54
55
56
57
58
59

```

Figura 3.7: Autore Controller

Nello specifico:

- `@Slf4j`

Descrizione: Aggiunge un logger log alla classe (grazie a Lombok).

Utilizzo: Permette di scrivere `log.info(...)`, `log.error(...)`, ecc. senza dover istanziare manualmente un logger.

- `@RestController`

Descrizione: Indica che questa classe è un controller REST.

Utilizzo: Combina `@Controller` e `@ResponseBody`. Restituisce automaticamente oggetti JSON/XML come risposta HTTP.

- `@SecurityRequirement(name = "bearer-auth")`

Descrizione: Specifica che questo controller richiede un’autenticazione Bearer.

Utilizzo: Usata per documentazione Swagger/OpenAPI per indicare il tipo di autenticazione richiesto.

- `@RequestMapping("/autori")`

Descrizione: Definisce il prefisso dell'URL per tutte le richieste in questo controller.

Utilizzo: Tutti i metodi gestiranno rotte che iniziano con `/autori`.

- `@PostMapping("/signup")`

Metodo associato: `addAutore`

Descrizione: Gestisce richieste POST `/autori/signup` per aggiungere un autore.

Altre annotazioni:

- `@ResponseStatus(HttpStatus.CREATED)`: ritorna HTTP 201 se tutto va a buon fine.
- `@RequestBody` e `@Valid`: mappa e valida il corpo della richiesta JSON in un oggetto `AutoreDto`.

- `@DeleteMapping("/id")`

Metodo associato: `removeAutore`

Descrizione: Gestisce richieste DELETE `/autori/{id}` per rimuovere un autore tramite ID.

Altre annotazioni:

- `@PathVariable`: estrae il parametro `id` dall'URL.

- `@PutMapping`

Metodo associato: `editAutore`

Descrizione: Gestisce richieste PUT `/autori` per modificare un autore.

Altre annotazioni:

- `@RequestBody`: riceve l'autore aggiornato nel corpo della richiesta.

- `@GetMapping("/nome")`

Metodo associato: `getAutoreByName`

Descrizione: Gestisce GET /autori/nome?value=... per ottenere un autore per nome.

Altre annotazioni:

- @RequestParam: legge il parametro value dalla query string.

- @GetMapping("/email")

Metodo associato: getAutoreByEmail

Descrizione: Come sopra, ma per l'email.

- @GetMapping("/all")

Metodo associato: getAllAutori

Descrizione: Restituisce la lista di tutti gli autori con GET /autori/all.

3.4.3 Service

Il service è il layer dove vengono effettuate le operazioni richieste (chiamate dal controller). Si potrà notare come segue i principi elencati in precedenza: *astrazione* della logica di business, *isolamento* dalla logica di persistenza (tramite repository), uso corretto di DTO e Mapper per disaccoppiare entità e API ed inoltre gestione sicura delle password e assegnazione ruoli nel contesto di un'app autenticata.

Di seguito un esempio:

```

19  @Service
20  @Transactional
21  @RequiredArgsConstructor
22  public class AutoreServiceImpl implements AutoreService{
23
24      private final AutoreRepository autoreRepository;
25
26      private final AutoreMapper autoreMapper;
27
28      private final UserRepository userRepository;
29
30      private final RoleService roleService;
31
32      private final PasswordEncoder passwordEncoder;
33
34      public void addAutore(AutoreDto dto) { 14 usages
35          Autore a = autoreMapper.autoreDtoToAutore(dto);
36          User user = autoreMapper.autoreDtoToUser(dto);
37          user.setRole(roleService.findByName("ROLE_CLIENT"));
38          user.setPassword(passwordEncoder.encode(user.getPassword()));
39          autoreRepository.save(a);
40          userRepository.save(user);
41
42
43      >     public boolean removeAutore(int id) { return invokePostControl( method: "deleteById", id, autore: null); }
44
45      public boolean editAutore(AutoreDto dto) { 4 usages
46          Autore a = autoreMapper.autoreDtoToAutore(dto);
47          return invokePostControl( method: "save", a.getId(), a);
48
49
50
51
52      public AutoreDto getAutoreByName(String nome) { 3 usages
53          Autore a = autoreRepository.findByName(nome);
54          if(a == null)
55              throw new AutoreNotFoundException("Autore non trovato");
56          return autoreMapper.autoreToAutoreDto(a);
57
58
59
60      public AutoreDto getAutoreByEmail(String email) { 11 usages
61          Autore a = autoreRepository.findByEmail(email);
62

```

Figura 3.8: Autore Service

- `@Service`

Descrizione: Segnala a Spring che questa classe è un componente del tipo `Service`, quindi viene automaticamente rilevata durante la scansione dei componenti.

- `@Transactional`

Descrizione: Tutti i metodi sono eseguiti all'interno di una transazione. Se uno fallisce, le modifiche vengono annullate.

- `@RequiredArgsConstructor` (di Lombok)

Descrizione: Genera automaticamente un costruttore con parametri per tutti i campi `final`, permettendo l'iniezione tramite costruttore (esempio: `private final AutoreRepository autoreRepository`).

3.4.4 Repository

Repository è il layer che si interfaccia direttamente col database.

Spring Data JPA consente di definire query solo scrivendo il nome del metodo seguendo una convenzione, ad esempio una semplice SELECT con clausola su un campo si può generare con il seguente paradigma: **findBy nome campo**.

Con questo tipo di implementazione uno dei vantaggi è sicuramente la mancanza di necessità di implementare manualmente le query SQL.

```

7  @Repository
8  public interface AutoreRepository extends JpaRepository<Autore, Integer> {
9      Autore findByNome(String nome);
10     Autore findByEmail(String email);
11 }
12

```

Figura 3.9: Autore Repository

- `@Repository`

Descrizione: Indica che questa interfaccia è un componente Spring di tipo `repository`. Permette a Spring di gestirla come bean e abilita la gestione delle eccezioni personalizzate.

- `public interface AutoreRepository extends JpaRepository<Autore, Integer>`

Estensione di: `JpaRepository`

Fornisce: Metodi già pronti come `save()`, `findById()`, `findAll()`, `delete()`, ecc.

Parametri:

- `Autore`: il tipo dell'entità.
- `Integer`: il tipo dell'ID primario (presumibilmente `@Id` in `Autore` è un `Integer`).

3.4.5 JWT

```

22     public class JwtTokenGeneratorFilter extends OncePerRequestFilter {
23
24     @Override
25     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
26                                     FilterChain filterChain) throws ServletException, IOException {
27         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
28         if (null != authentication) {
29             Environment env = getEnvironment();
30             if (null != env) {
31                 String secret = env.getProperty("jwt.secret");
32                 SecretKey secretKey = Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8));
33                 String jwt = Jwts.builder().issuer("Chefbook").subject("JWT Token")
34                               .claim("username", authentication.getName())
35                               .claim("authorities", authentication.getAuthorities().stream().map(
36                                   GrantedAuthority::getAuthority).collect(Collectors.joining(", ")))
37                               .issuedAt(new Date())
38                               .expiration(new Date((new Date()).getTime() + 3000000))
39                               .signWith(secretKey).compact();
40                 response.setHeader("Authorization", jwt);
41             }
42         }
43         filterChain.doFilter(request, response);
44     }
45
46     @Override
47     protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {
48         return !request.getServletPath().contains("/user/signin");
49     }
50 }

```

Figura 3.10: JWT Token Generator

1. Recupero dell'autenticazione

Il filtro ottiene le informazioni sull'utente autenticato dal contesto di sicurezza di Spring.

2. Verifica dell'autenticazione

Si assicura che ci sia un oggetto `Authentication` valido, cioè che l'utente sia effettivamente autenticato.

3. Lettura delle proprietà dell'ambiente

Viene recuperato il valore della chiave segreta per la firma del token (`jwt.secret`) da un file `application.properties`.

4. Generazione della chiave segreta

La chiave HMAC-SHA viene creata a partire dalla stringa di configurazione, ed è usata per firmare il token.

5. Costruzione del token JWT

Il filtro crea un JWT che contiene:

- L'emittente (*issuer*) del token.
- Il soggetto (*subject*), che specifica il tipo di token.
- Informazioni come il nome utente e i ruoli.
- La data di emissione e la data di scadenza.
- La firma del token, effettuata con la chiave segreta.

6. Inserimento del token nella risposta

Una volta generato, il token viene aggiunto all'header della risposta HTTP, con chiave `Authorization`.

Metodo `shouldNotFilter`

Questo metodo impedisce che il filtro venga applicato durante la richiesta di login, tipicamente sulla rotta `/user/signin`. Serve per evitare che venga generato un token mentre l'utente si sta ancora autenticando.

```

22     public class JwtTokenValidatorFilter extends OncePerRequestFilter { 2 usages
23         @Override no usages
24     @Override protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
25             throws ServletException, IOException {
26         String jwt = request.getHeader("Authorization").replace("Bearer ", "");
27         if(null != jwt) {
28             try {
29                 Environment env = getEnvironment();
30                 if (null != env) {
31                     String secret = env.getProperty("jwt.secret");
32                     SecretKey secretKey = Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8));
33                     if(null != secretKey) {
34                         Claims claims = Jwts.parser().verifyWith(secretKey)
35                             .build().parseSignedClaims(jwt).getPayload();
36                         String username = String.valueOf(claims.get("username"));
37                         String authorities = String.valueOf(claims.get("authorities"));
38                         Authentication authentication = new UsernamePasswordAuthenticationToken(username, credentials: null,
39                             AuthorityUtils.commaSeparatedStringToAuthorityList(authorities));
40                         SecurityContextHolder.getContext().setAuthentication(authentication);
41                     }
42                 }
43             } catch (Exception exception) {
44                 throw new BadCredentialsException("Invalid Token received!");
45             }
46         }
47     }
48     filterChain.doFilter(request, response);
49 }
50
51     @Override no usages
52     @Override protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {
53         return request.getServletPath().contains("/user/signin")
54             || request.getServletPath().contains("/autori/signup")
55             || request.getServletPath().contains("/v3/api-docs")
56             || request.getServletPath().contains("/swagger-ui");
57     }
58 }

```

Figura 3.11: JWT Token Validator

1. Estrazione del JWT

Recupera l'header `Authorization` dalla richiesta e rimuove il prefisso `Bearer` per ottenere il token puro.

2. Verifica delle condizioni iniziali

Controlla che il token esista e che siano disponibili:

- Le proprietà di ambiente (`Environment`)
- Il segreto per la firma (`jwt.secret`)
- La chiave derivata (`SecretKey`)

3. Parsing e verifica del token

Usa la libreria `jjwt` per:

- Verificare la firma con la chiave segreta
- Estrarre i *claims* (le informazioni) dal token

Tra i claims⁷ estratti:

- `username`: il nome dell’utente autenticato
- `authorities`: la lista dei ruoli/permessi, separati da virgola

4. Creazione del contesto di sicurezza

Costruisce un oggetto `UsernamePasswordAuthenticationToken` usando `username` e le `authorities`, e lo imposta nel contesto di sicurezza.

Così, l’utente viene considerato autenticato per il resto della richiesta.

5. Gestione degli errori

Se qualcosa fallisce (es. token malformato, firma non valida), viene lanciata una `BadCredentialsException` con il messaggio "Invalid Token received!".

Metodo `shouldNotFilter`

Questo metodo impedisce l’applicazione del filtro in alcuni casi:

- Durante la richiesta di login (`/user/signin`)
- Durante la registrazione (`/autori/signup`)
- Durante l’accesso alla documentazione OpenAPI (`/v3/api-docs`, `/swagger-ui`)

Queste eccezioni sono utili perché in quei contesti non esiste ancora un token JWT o non serve verificarlo.

3.4.6 Docker

Dopo lo sviluppo dell’applicativo lo si è trasferito su docker. Per farlo il primo passo è stato compilare il codice java per avere il file con estensione `.jar`. Una volta fatto ciò si è scritto il *Dockerfile*:

⁷I claims sono informazioni contenute nella sezione payload del JWT. Servono a descrivere l’utente o il contesto della token.

```
1 ⌘ FROM openjdk:21-jdk-slim
2 WORKDIR /app
3 COPY target/chefbook-0.0.1-SNAPSHOT.jar chefbook.jar
4 ENTRYPOINT ["java", "-jar", "chefbook.jar"]
```

Figura 3.12: Dockerfile

Nell'ordine: specifica la versione di java utilizzata, imposta la directory di lavoro, Copia il file .jar dalla directory *target* (del progetto Maven) nel container rinominandolo ed infine Esegue automaticamente il jar quando il container parte. Un'altra componente importante è il docker-compose.yaml, è un file nel quale viene scritto tutto l'ambiente da creare al lancio del comando da terminale: *docker compose*.

```

1   name: cb_suite
2 ▷ services:
3 ▷   chefbook:
4     container_name: cb
5     build: .
6     image: chefbook:latest
7     ports:
8       - 8080:8080
9     depends_on:
10    - db
11   volumes:
12     - /mnt/c/Users/amministratore/Desktop/Tirocinio/logs:/app/logs
13   networks:
14     - test_chefbook
15
16 ▷ db:
17   container_name: mysqldb
18   image: mysql:8.0
19   ports:
20     - "3307:3306"
21   environment:
22     MYSQL_ROOT_PASSWORD: root
23     MYSQL_DATABASE: chefbook
24     MYSQL_USER: admin
25     MYSQL_PASSWORD: |
26   volumes:
27     - db_data:/var/lib/mysql
28   networks:
29     - test_chefbook
30
31 volumes:
32   db_data:
33     name: dbdata
34
35 networks:
36   test_chefbook:
37     name: test_chefbook

```

Figura 3.13: Docker-compose

cb suite è il nome del progetto istanziato su docker che comprende 2 container, l'applicativo cb e il database mysql.

Viene costruita un'immagine Docker partendo dalla cartella dove si trova il Dockerfile.

Per il .jar:

Le porte 8080 del container e del computer vengono collegate, così da accedere all'app da localhost:8080.

Viene definito un volume per salvare i file di log dell'applicazione in una cartella sul computer, utile per il debug.

Il container viene messo in una rete virtuale chiamata *test chefbook*.

Si specifica che questo servizio dipende dal database, quindi il database si avvierà prima automaticamente.

Per il DB:

viene avviato un container MySQL configurato con: una password di root (root), la creazione automatica di un database chiamato chefbook, la creazione di un utente chiamato admin.

La porta 3306 del container (quella usata da MySQL) è collegata alla porta 3307 del computer, così da potersi connettere anche esternamente.

I dati del database vengono salvati in un volume chiamato *dbdata*, così non si perdono anche se il container viene eliminato.

Anche questo servizio è connesso alla rete virtuale *test chefbook*.

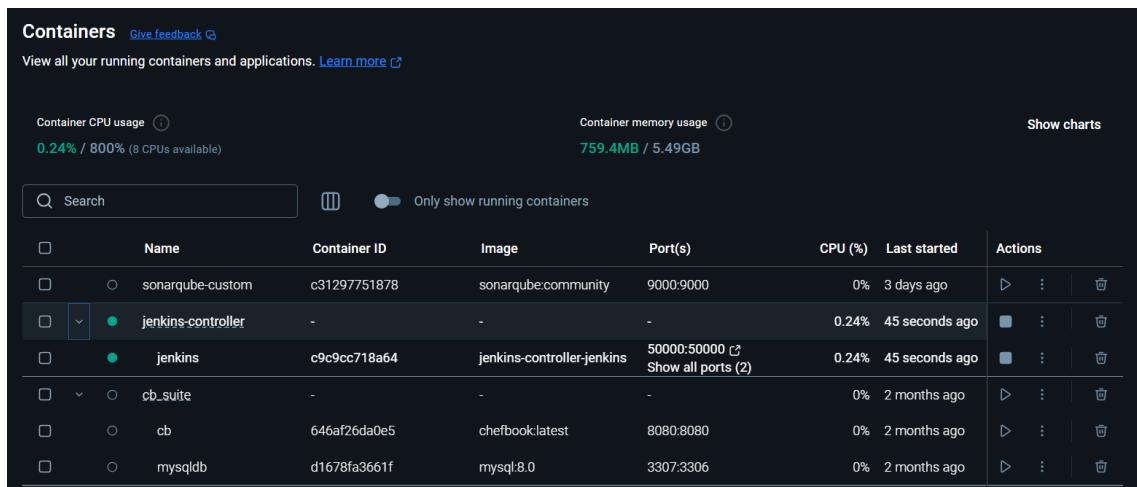


Figura 3.14: Docker Desktop

Infine docker è servito anche per i vari test (saranno poi specificati nel capitolo 4) utilizzando su di esso sonarqube e jenkins, su quest'ultimo vi è stata fatta partire la pipeline che eseguiva i test, su sonarqube invece si eseguiva l'analisi del codice.

3.4.7 AWS

Finita l'implementazione in locale è stato necessario migrare in AWS per far girare l'applicativo anche in cloud. Si sono seguiti degli specifici passaggi

- **Progettare l'architettura AWS in base alle risorse delle quali necessitava l'applicativo**
- **Migrazione del codice sorgente da GitHub a Code Commit**
- **Creazione del Database**
- **Creazione ambiente con Beanstalk**
- **Creazione Build**
- **Creazione Pipe**

Scelta dei servizi e dell'architettura

Si sono individuati i servizi necessari in base anche ad un ipotetica comunicazione con servizi front end. Per il deployment in cloud, il progetto è pensato per essere eseguito su AWS, sfruttando **AWS RDS** per il database e **AWS Elastic Beanstalk** per il backend, oltre alla suite **AWS Code** (CodeCommit, CodePipeline, CodeBuild, CodeDeploy).

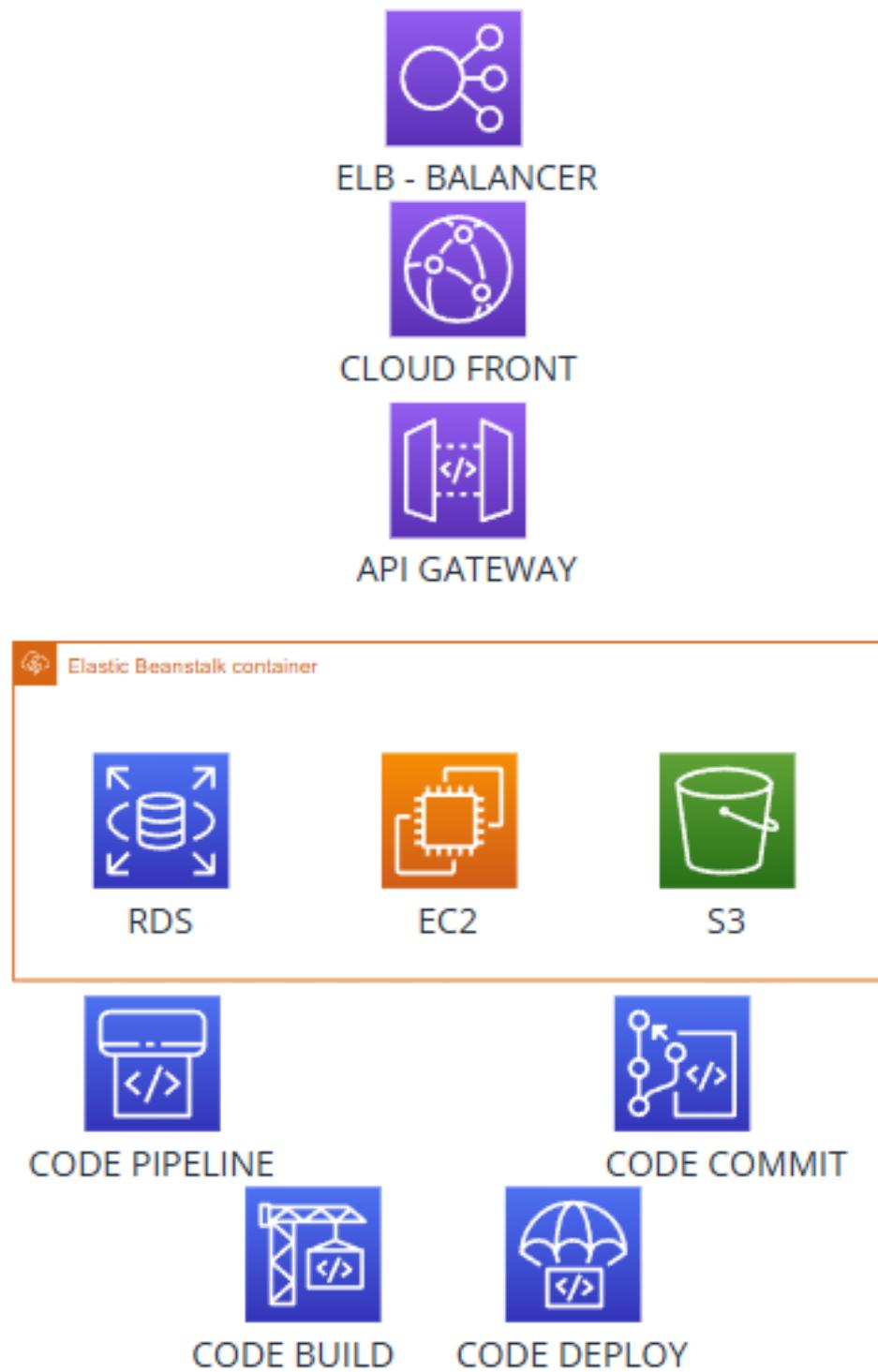


Figura 3.15: Architettura AWS - In viola i servizi FE, In azzurro i servizi per il codice, nel container i servizi utili al BE e DB

Migrazione Codice

Per la migrazione del codice vi è stato un solo passaggio dopo averlo salvato su GitHub.

Con l'ausilio della CLI AWS installata in locale e collegata all'ambiente AWS(AWS ID, Server di Dublino) è bastato creare una repository nel servizio *CodeCommit* e pushare il codice clonato da GitHub a CodeCommit

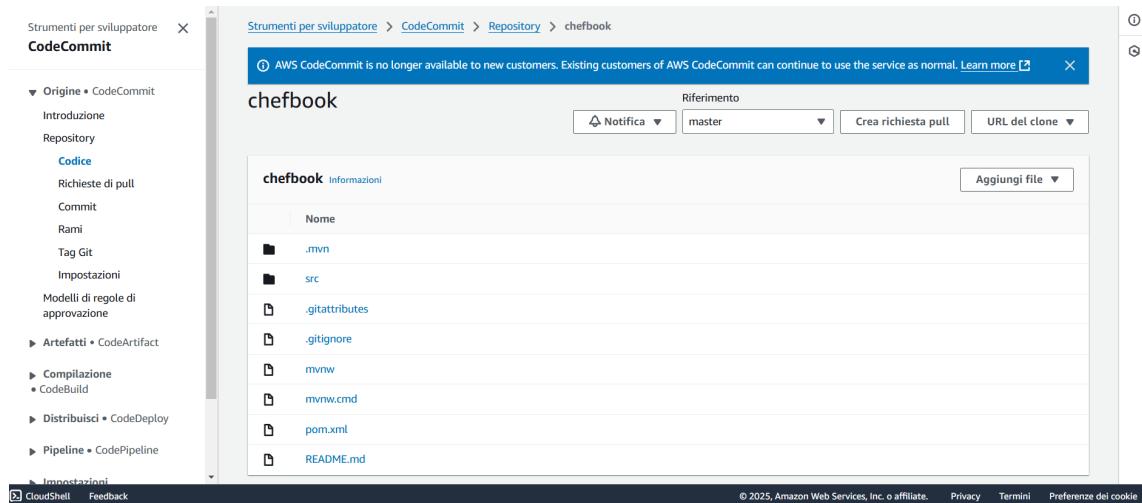


Figura 3.16: AWS - CodeCommit

Creazione Database

Per la piattaforma progettata ci si è resi conto che bastava un DB di grandezza *micro*, adatto a gestire medio/piccole mole di dati. Il Database è stato creato prima dell'ambiente stesso con Beanstalk in modo da poterlo collegare direttamente alla creazione dell'ambiente. Una volta istanziato la parte più importante da controllare era il codice con l'endpoint del DB stesso.

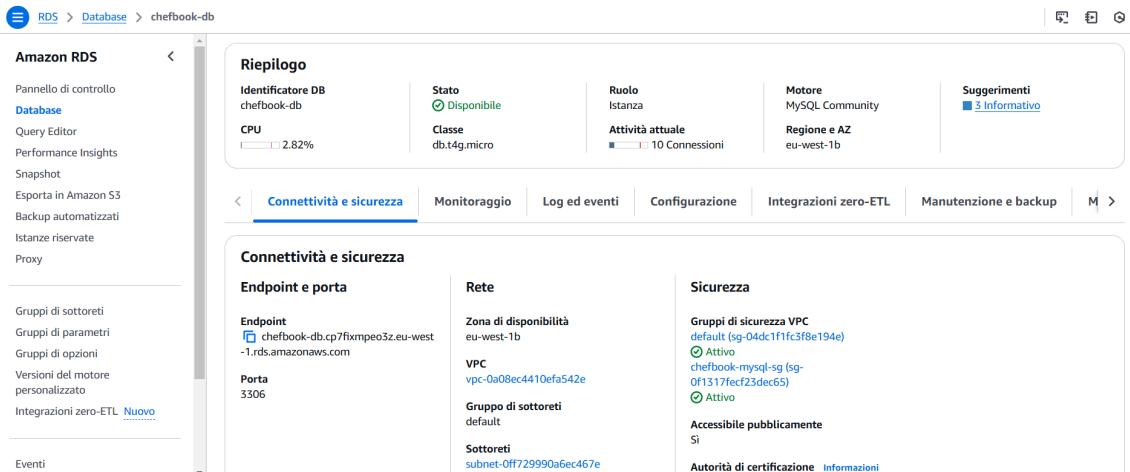


Figura 3.17: AWS - RDS

Creazione Ambiente

Quando si crea un ambiente Elastic Beanstalk è una piattaforma PaaS (Platform as a Service) che semplifica il deployment (come chefbook-app-test-env). AWS gestisce un insieme di risorse infrastrutturali, in questo caso: EC2, S3 ed RDS che viene collegato al momento della creazione in modo da avere tutto il necessario già collegato.

Perchè si è scelto EC2

Ogni *istanza* è l'equivalente di un computer/VM nel cloud.

Si sceglie il sistema operativo (Linux, Windows), la CPU, la RAM, la capacità di rete, la GPU, e il disco.

Le istanze possono essere:

- **On-Demand** - si paga a tempo
- **Reserved** - per uso a lungo termine, più economiche
- **Spot** - basate su offerte, molto economiche ma instabili

Caratteristiche:

- **Basso costo:** Le istanze `t2.micro` sono le più economiche offerte da AWS
- **Sufficiente per carichi leggeri:** Se l'applicazione ha un traffico basso o è in fase di sviluppo, una istanza micro è generalmente sufficiente.

Figura 3.18: AWS - Beanstalk

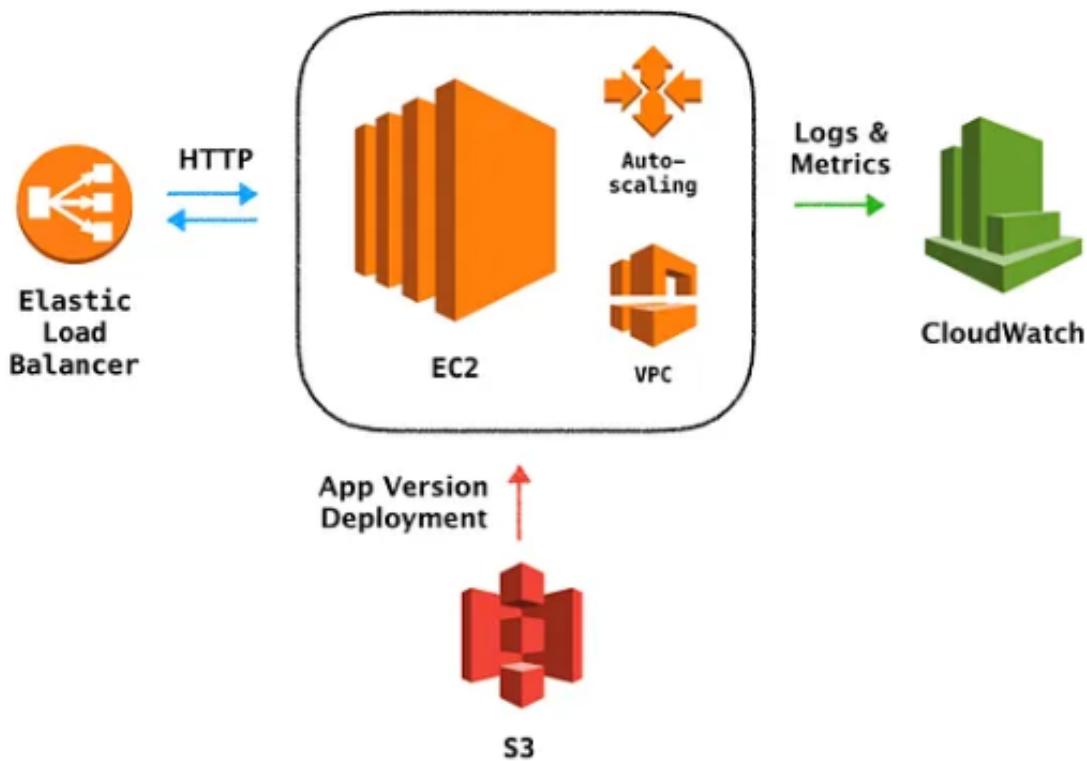


Figura 3.19: AWS - Ambiente Beanstalk Architettura⁸

Spiegazione Architettura:

⁸Fonte:<https://blog.stackademic.com/basic-aws-elastic-beanstalk-configuration-in-2024-f10d80776e6a>

1. Elastic Load Balancer (ELB)

Distribuisce automaticamente il traffico HTTP tra le istanze EC2 per garantire disponibilità e affidabilità.

Vantaggi:

- Alta disponibilità e tolleranza ai guasti.
- Scalabilità automatica con picchi di traffico.

Svantaggi:

- Costo aggiuntivo se non correttamente configurato.
- Richiede gestione accurata delle health check.

2. Amazon EC2 (Elastic Compute Cloud)

EC2 è un server virtuale che si può avviare, configurare e spegnere secondo le esigenze, pagando solo per il tempo in cui lo si utilizza.

Vantaggi:

- **Elevata configurabilità:** istanza, memoria, CPU.
- **Può essere integrato con altri servizi AWS.**
- **Deployment rapido:** L'infrastruttura viene generata automaticamente, semplificando enormemente il lavoro.
- **Risparmio economico:** Ideale per ambienti di sviluppo e test a basso costo.
- **Auto-scaling:** Anche con istanze micro, Beanstalk può scalare orizzontalmente creando nuove istanze quando il carico lo richiede.
- **Manutenzione automatica:** Gestione semplificata di aggiornamenti, logging e monitoraggio.

Svantaggi:

- **Responsabilità dell'utente nella gestione del sistema operativo.**
- **Costi variabili e dipendenti dall'utilizzo.**
- **Prestazioni limitate:** Le istanze micro sono dotate di risorse computazionali molto ridotte (CPU e RAM), quindi non adatte per applicazioni ad alta intensità.

- **Non adatte a carichi stabili o pesanti:** In questi casi è meglio passare a istanze t3.medium o superiori.[13]

3. Auto Scaling

Aggiunge o rimuove automaticamente istanze EC2 in base al carico.

Vantaggi:

- Ottimizza i costi in base alla domanda reale.
- Evita downtime causati da overload.

Svantaggi:

- Può essere complicato da configurare inizialmente.
- Richiede metriche ben definite per funzionare efficacemente.

4. Amazon VPC (Virtual Private Cloud)

Fornisce isolamento di rete per le risorse AWS.

Vantaggi:

- Maggior controllo su networking, subnet, firewall, ecc.
- Essenziale per ambienti produttivi sicuri.

Svantaggi:

- Configurazione iniziale complessa.

5. Amazon CloudWatch

Monitora e raccoglie log e metriche dalle risorse AWS.

Vantaggi:

- Monitoraggio continuo e alert configurabili.
- Integrazione con auto scaling e altre azioni automatizzate.

Svantaggi:

- Il logging approfondito può generare costi significativi.

6. Amazon S3 (Simple Storage Service)

Memorizza i file di deployment dell'applicazione.

Vantaggi:

- Alta disponibilità e durabilità dei dati.
- Facile integrazione con Beanstalk e altri servizi.

Svantaggi:

- Potenziale esposizione dei dati se i permessi non sono ben configurati.
- Non adatto a contenuti dinamici o ad accesso diretto continuo.[14]

Creazione Build

In questo stato il file viene salvato con estensione .jar e si vanno a definire la fase di build vera e propria, con i comandi da eseguire nel container di compilazione ed infine si specifica quali file output devono essere salvati come risultato della build per poi essere salvato in un bucket S3.

Creazione Pipeline

Una pipeline è un flusso automatizzato di azioni che prendono il codice sorgente e lo trasformano in una versione funzionante e distribuita dell'applicazione, senza intervento manuale. Viene definita *catena di montaggio del software*.⁹ In questo caso ha 3 step:

1. **Source** - rileva il codice sorgente nella repository
2. **Build** - crea il file .jar dopo aver eseguito i test unitari e di integrazione scritti in maven
3. **Deploy** - prende il file .jar e lo rilascia nell'ambiente Beanstalk.

⁹Fonte: <https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html>

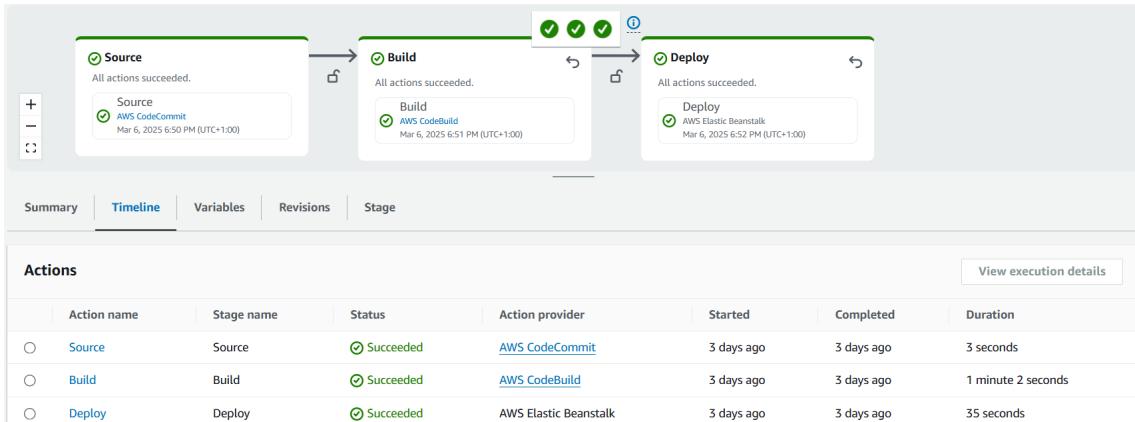


Figura 3.20: AWS - CodePipeline

CAPITOLO 4

Testing e Validazione

4.1 Introduzione

Nel ciclo di vita dello sviluppo software moderno, test e validazione non sono fasi opzionali, ma elementi essenziali per garantire qualità, affidabilità e sicurezza.

- **Evitare bug in produzione**

Errori non rilevati possono causare crash, malfunzionamenti o vulnerabilità. I test riducono drasticamente questi rischi.

- **Migliorare la qualità del codice**

Il testing aiuta gli sviluppatori a scrivere codice più modulare, leggibile e manutenibile.

- **Facilitare il refactoring e l'evoluzione del progetto**

Con una buona suite di test, si può aggiornare il codice senza paura di rompere funzionalità esistenti.

- **Automatizzare la validazione nel ciclo CI/CD**

I test possono essere eseguiti automaticamente a ogni push o pull request grazie a tool di integrazione continua.

4.2 Qualità del codice

Per la qualità del codice si è utilizzato *SonarQube*, tool che analizza il codice ed ha diverse funzionalità per migliorarlo. Più nello specifico va ad effettuare quella che si chiama analisi statica del codice¹



Figura 4.1: Grafico Debito tecnico

Il **debito tecnico** indica l'insieme del lavoro aggiuntivo che si accumula quando, per accelerare il rilascio di un software, si scelgono soluzioni provvisorie o scorciatoie. Queste scelte possono tradursi in codice poco organizzato, assenza di documentazione o test insufficienti. Sebbene possano risultare utili nell'immediato, nel lungo periodo complicano la manutenzione del software e ne ostacolano l'evoluzione, aumentando tempi e costi di sviluppo futuri. [15]

I controlli sul codice sorgente rilevano:

Bug: errori logici o implementativi.

Vulnerabilità di sicurezza: possibili falliche sfruttabili da attaccanti.

¹consiste nell'analizzare il codice sorgente senza eseguirlo, per individuare problemi strutturali, violazioni delle regole di codifica, possibili bug e rischi per la sicurezza.

Code Smell: pratiche di programmazione che, pur funzionando, compromettono la manutenibilità del software.

4.2.1 Quando è stato utilizzato

Questo tool è stato utilizzato in un preciso stage della pipeline on-prem (su jenkins), più precisamente nello stage *test* che ha delle restrizioni. Nella pipeline in locale che ha 4 stage (source, build, test e deploy) e che effettua il deploy se tutti i 4 stage vanno a buon fine, sonarQube si inserisce con delle restrizioni, più nello specifico lo stage *test* si da come *passed* se tutti i test eseguiti in java (unità ed integrazione) vanno a buon fine, la coverage dei test è almeno uguale ad una percentuale stabilita (in questo caso 80%) e soprattutto alla scansione non vi erano *blocker*²

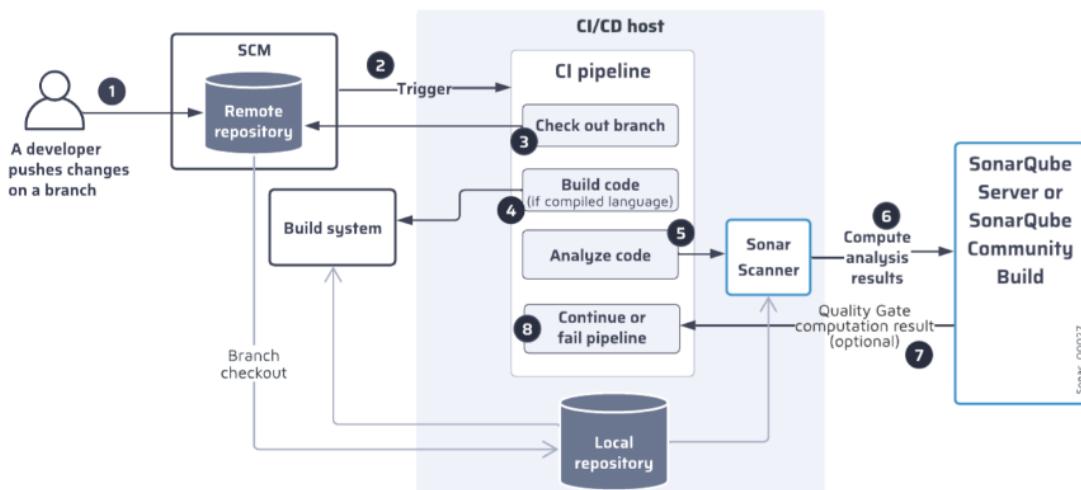


Figura 4.2: SonarQube in Pipeline³

²In SonarQube, un'issue di tipo blocker è classificata come la gravità massima tra i problemi rilevati durante l'analisi statica del codice. Queste issue rappresentano bug critici o vulnerabilità gravi che possono compromettere il corretto funzionamento o la sicurezza dell'applicazione e che devono essere risolte immediatamente.[16]

³Fonte: <https://docs.sonarsource.com/sonarqube-server/latest/analyzing-source-code/analysis-overview/>

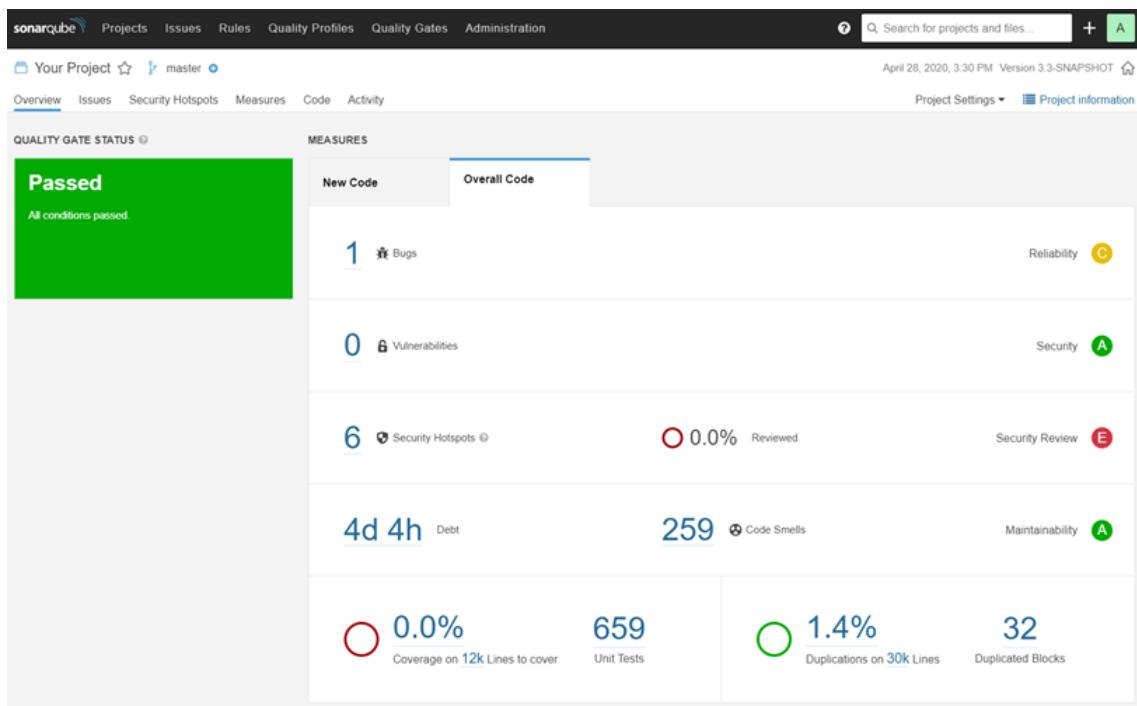


Figura 4.3: Analisi statica SonarQube ⁴

4.3 Test

Al fine di garantire la correttezza, affidabilità e manutenibilità dell'applicazione, è stata condotta una fase di testing strutturata e approfondita, che ha coinvolto diversi livelli di verifica. L'obiettivo principale è stato quello di assicurare che ogni componente del sistema rispondesse correttamente ai requisiti funzionali, che le interazioni tra i moduli avvenissero in modo coerente e che eventuali regressioni fossero facilmente identificabili.

La strategia adottata si è basata sull'impiego combinato di:

- **Test unitari** realizzati con i framework *JUnit5* e *Mockito*, volti a validare in maniera isolata ciascuna funzionalità o unità logica del sistema. Tali test hanno permesso di simulare comportamenti e dipendenze esterne (tramite mocking) e verificare il corretto funzionamento delle singole classi e metodi, con particolare attenzione alla copertura delle condizioni di errore e dei casi limite.

⁴Fonte: <https://docs.sonarsource.com/sonarqube-server/9.9/try-out-sonarqube/>

- **Test d'integrazione** sono stati eseguiti sfruttando l'annotazione `@SpringBootTest`, con l'obiettivo di testare il sistema nella sua interezza o in porzioni significative, includendo anche l'inizializzazione del contesto Spring. Questi test sono stati progettati per verificare il corretto funzionamento dei flussi applicativi principali e delle interazioni tra i vari layer dell'applicazione (controller, service, repository), simulando casi d'uso reali e scenari operativi significativi.

Grazie a questa impostazione, l'applicazione ha potuto beneficiare di un elevato livello di controllo e affidabilità del codice, minimizzando il rischio di errori in fase di esecuzione e facilitando le attività di refactoring e manutenzione evolutiva.

4.4 Code coverage

Parallelamente alla scrittura dei test unitari e di integrazione, è stata effettuata una costante attività di analisi della copertura del codice (code coverage) al fine di garantire un controllo oggettivo sul livello di verifica raggiunto. Tale misurazione è stata realizzata mediante JaCoCo⁵, i quali hanno fornito report dettagliati sulla porzione di codice effettivamente testata.

L'analisi ha riguardato sia la copertura delle linee di codice che la copertura dei rami decisionali, con l'obiettivo di:

- Identificare eventuali zone non coperte o scarsamente testate;
- Priorizzare l'aggiunta di nuovi test su parti critiche del sistema;
- Garantire la sostenibilità del codice nel lungo periodo.

In particolare, si è mirato al raggiungimento di una copertura superiore all'80% sulle componenti core dell'applicazione.

4.5 Automazione CI/CD con Jenkins

L'intero ciclo di build, test e deploy dell'applicazione è stato automatizzato tramite una pipeline di Continuous Integration e Continuous Delivery (CI/CD) basata su Jenkins, al fine di garantire qualità, velocità e affidabilità nei rilasci.

⁵Fonte: <https://www.eclemma.org/jacoco/>

La pipeline Jenkins prevede i seguenti step principali:

- Clonazione del repository da GitHub;
- Build automatica del progetto e risoluzione delle dipendenze;
- Esecuzione dei test unitari e di integrazione, con generazione automatica dei report di coverage, analisi statica del codice tramite SonarQube, con enforcement di regole di qualità e verifica del superamento del Quality Gate;
- Deployment automatizzato in ambiente di staging o produzione, condizionato al superamento di tutti i test e controlli di qualità.

Grazie all’adozione di Jenkins, si è potuto beneficiare di un feedback continuo sull’integrità del codice, rilevando tempestivamente eventuali regressioni o problemi introdotti da nuove modifiche. Inoltre, l’automazione del deploy ha ridotto significativamente il rischio di errori manuali e ha reso il ciclo di rilascio più veloce, ripetibile e sicuro.

4.6 Swagger UI

Per migliorare l’accessibilità e la comprensione delle API REST esposte dall’applicazione, è stato integrato *Swagger UI*, strumento di documentazione interattiva basato sullo standard OpenAPI.

Grazie a questa integrazione, ogni endpoint dell’applicazione è:

- Descritto in modo chiaro e leggibile, con specifica delle rotte, parametri, tipi di risposta e codici HTTP attesi;
- Navigabile tramite interfaccia web interattiva;
- Testabile direttamente dal browser.

L’applicazione è protetta tramite autenticazione con JWT, una delle soluzioni più diffuse per proteggere API REST. Il meccanismo si basa sull’emissione di un token firmato al momento dell’autenticazione, che l’utente dovrà poi fornire in ogni richiesta successiva per accedere alle risorse protette (endpoint dei vari controller).



Figura 4.4: Interfaccia grafica di Swagger⁶

4.7 Test Cloud

Tutti i test sono stati effettuati prima del deploy in cloud in modo da poter risparmiare costi e risorse una volta caricati i file su AWS. Alla luce di questa premessa comunque sono stati effettuati 2 tipi di test:

- I test scritti in locale sono stati rieseguiti anche nella pipeline citata in precedenza, scritta su *CodePipeline* con l'eccezione che non vi era il supporto di SonarQube.
- *Swagger UI* è stato raggiunto tramite endpoint fornito da AWS per poter verificare che le funzionalità messe a disposizione fossero tutte efficienti, esattamente come effettuato nella prima parte (on-prem).

⁶Fonte: <https://swagger.io/tools/swagger-ui/>

CAPITOLO 5

Conclusioni

5.1 Costi e Conclusioni

Lo sviluppo della piattaforma *ChefBook* ha rappresentato un’esperienza progettuale completa, nella quale si è riusciti a coniugare aspetti teorici, pratici e strategici in un’applicazione concreta, moderna e facilmente scalabile. Il lavoro ha permesso di affrontare tutti gli stadi del ciclo di vita di un software: dall’analisi dei requisiti alla progettazione architettonale, dall’implementazione all’integrazione in ambienti reali, fino alla validazione mediante testing strutturato.

Uno dei punti di forza più rilevanti del progetto è stato l’approccio incrementale alla distribuzione: lo sviluppo iniziale è stato eseguito in ambiente on-premise, permettendo di effettuare test approfonditi, ottimizzazioni e analisi della qualità del codice prima della migrazione in cloud. Questo ha permesso di ridurre drasticamente eventuali inefficienze o configurazioni errate prima dell’utilizzo delle risorse AWS, che come noto, seguono un modello di pricing a consumo.

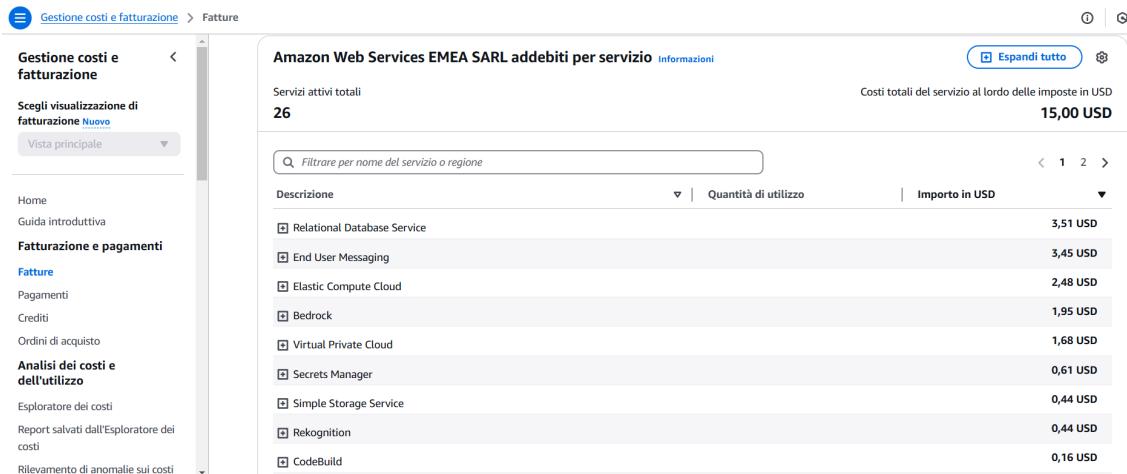


Figura 5.1: Costi dei servizi AWS

Grazie a queste scelte architetturali e operative, l'infrastruttura è risultata non solo performante, ma anche economicamente sostenibile, confermando l'importanza di un design consapevole prima della fase di deployment in cloud.

Oltre agli aspetti infrastrutturali, il progetto ha messo in pratica soluzioni tecnologiche avanzate come l'impiego di Spring Boot, Docker, CI/CD pipelines con Jenkins e SonarQube, fino alla migrazione completa su AWS con servizi come Elastic Beanstalk, RDS e CodePipeline. Le funzionalità sviluppate, tra cui la gestione delle ricette, la privacy granulare e il meccanismo di trasferimento collaborativo, hanno contribuito a creare un prodotto solido, orientato all'utente e potenzialmente pronto per essere esteso o industrializzato.

In conclusione, ChefBook non è solo un social verticale per appassionati di cucina, ma un vero caso studio sull'ingegneria del software moderna, dove metodologie DevOps, scalabilità cloud e progettazione orientata alla qualità convergono in un'unica soluzione funzionale e ben strutturata. Il lavoro svolto dimostra come **un'attenta pianificazione tecnica** possa non solo garantire qualità e sicurezza, ma anche generare vantaggi concreti in termini di sostenibilità economica.

5.2 Sviluppi futuri

5.2.1 Interfaccia Grafica

Sviluppo importante per il progetto *ChefBook* riguarda la creazione di una vera e propria interfaccia grafica (frontend), che ad oggi non è stata realizzata. Attualmente, il sistema è accessibile solo tramite API e strumenti tecnici come Swagger UI, ma per renderlo utilizzabile da tutti gli utenti, è necessario sviluppare un'interfaccia web semplice e intuitiva.

L'idea è quella di realizzare una web app moderna e responsive, accessibile sia da desktop che da dispositivi mobili. Per farlo, si potrebbe usare un framework JavaScript come **React.js**, molto diffuso e adatto a collegarsi facilmente alle API già esistenti nel progetto.

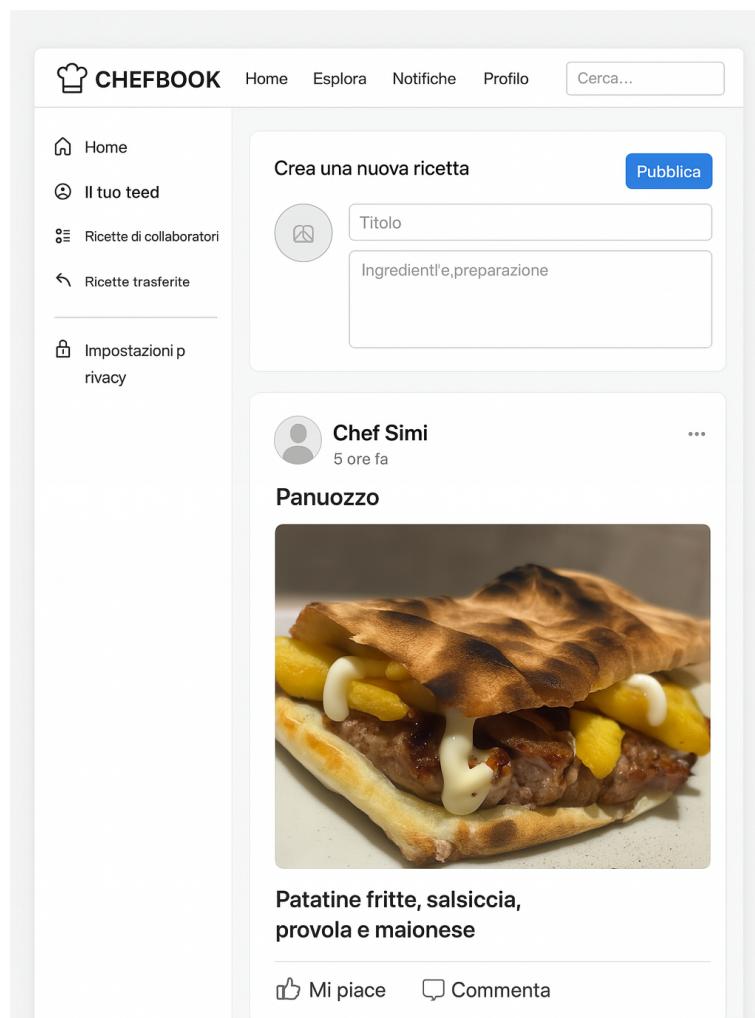


Figura 5.2: Mock up di ChefBook

5.2.2 Passaggio all'architettura AWS con le Lambda Functions

Una possibile e significativa evoluzione della piattaforma *ChefBook* riguarda la sua rifattorizzazione in ottica serverless, attraverso l'adozione delle **AWS Lambda Functions**. Questo approccio, basato sul paradigma *Function-as-a-Service* (FaaS), rappresenterebbe un cambio architettonico profondo rispetto alla struttura monolitica attualmente adottata, ma offre importanti vantaggi in termini di scalabilità, efficienza e riduzione dei costi operativi.

In particolare, le Lambda permettono di scomporre il backend in funzioni indipendenti e autonome, ciascuna eseguita in risposta a un evento specifico, ad esempio invio di una richiesta HTTP, inserimento di un record nel database o attivazione di una pipeline. Questa modularizzazione renderebbe il sistema più flessibile e facilmente manutenibile, consentendo di intervenire su singole funzionalità (ad esempio la creazione di una ricetta, l'autenticazione utente, o il trasferimento ricette) senza impattare l'intero ecosistema applicativo.

L'adozione del serverless porterebbe inoltre i seguenti vantaggi:

- **Scalabilità automatica e istantanea:** ogni funzione Lambda si adatta automaticamente al carico di lavoro, eliminando la necessità di dimensionare manualmente server o container;
- **Ottimizzazione dei costi:** il modello di pricing *pay-per-execution* consente di pagare esclusivamente per il tempo effettivo di esecuzione delle funzioni, riducendo drasticamente i costi in scenari a carico variabile o con utilizzo discontinuo;
- **Time-to-market ridotto:** grazie alla semplicità del deploy e alla natura modulare delle funzioni, è possibile introdurre nuove feature o fix in modo più rapido e isolato;
- **Maggiore resilienza:** l'indipendenza delle funzioni limita gli effetti a cascata in caso di malfunzionamenti, migliorando l'affidabilità complessiva del sistema.[17]

5.2.3 Gestione Avanzata della Registrazione Utenti

Un ulteriore sviluppo futuro per la piattaforma *ChefBook* riguarda il rafforzamento del processo di registrazione, con l'introduzione di un sistema di **verifica dell'identità professionale** degli utenti. L'obiettivo è evitare l'accesso non controllato da parte di utenti non qualificati o poco seri, garantendo che la community resti focalizzata su profili autenticamente coinvolti nel mondo culinario professionale.

Attualmente, la registrazione è aperta e automatica: ciò consente ampia accessibilità ma espone la piattaforma al rischio di creare un ambiente meno selezionato e meno credibile. Per questo motivo, si propone di introdurre una fase di controllo in grado di abilitare l'accesso soltanto a utenti che dimostrino il possesso di **titoli professionali, esperienza comprovata o appartenenza a ordini o associazioni di categoria**.

Le possibili soluzioni tecniche includono:

- **Verifica documentale:** caricamento obbligatorio, in fase di registrazione, di un attestato di formazione (es. diploma di scuola alberghiera, certificazione HAC-CP, iscrizione a enti professionali), con approvazione manuale o automatizzata da parte del sistema;
- **Integrazione con sistemi di identità digitale:** autenticazione tramite strumenti ufficiali come *SPID*, *CIE* o equivalenti europei, per garantire che l'utente corrisponda a una persona fisica identificabile e verificabile;

Questo sistema garantirebbe un maggior controllo sulla qualità della community e aumenterebbe la fiducia tra gli utenti, promuovendo uno spazio realmente professionale e orientato allo scambio qualificato di esperienze, tecniche e ricette culinarie.

Bibliografia

- [1] IBM, “Java spring boot – cos’è spring boot?” 2023. [Online]. Available: <https://www.ibm.com/it-it/topics/java-spring-boot> (Citato a pagina 7)
- [2] Nextre Engineering, “Java spring boot: cos’è, come funziona e perché usarlo,” 2023. [Online]. Available: <https://www.nextre.it/java-spring-boot/> (Citato alle pagine 8 e 9)
- [3] Red Hat, “Cos’è la containerizzazione,” 2023. [Online]. Available: <https://www.redhat.com/it/topics/cloud-native-apps/what-is-containerization> (Citato alle pagine 10 e 22)
- [4] Kinsta, “Cosa è docker: Una guida completa,” 2023. [Online]. Available: <https://kinsta.com/it/knowledgebase/cosa-e-docker/> (Citato alle pagine 10 e 15)
- [5] Amazon Web Services, “Qual è la differenza tra docker e una vm?” 2023. [Online]. Available: <https://aws.amazon.com/it/compare/the-difference-between-docker-vm/> (Citato alle pagine 12 e 14)
- [6] IBM, “Cos’è docker?” 2024. [Online]. Available: <https://www.ibm.com/it-it/think/topics/docker> (Citato a pagina 15)

- [7] D. Giampaoletti, "Sonarqube: uno strumento per l'analisi statica del software," https://amslaurea.unibo.it/id/eprint/23579/1/Davide_Giampaoletti_tesi.pdf, 2021. (Citato alle pagine 17 e 18)
- [8] R. ImpresaCity, "Cos'è jenkins, il server open source per la ci/cd," 2017. [Online]. Available: <https://www.impresacity.it/approfondimenti/18983/jenkins-ci-cd-open-source-devops.html> (Citato a pagina 18)
- [9] Atlassian, "Cos'è devops?" 2025. [Online]. Available: <https://www.atlassian.com/it/devops> (Citato alle pagine 19, 20 e 21)
- [10] J. Project, "Pipeline - jenkins user documentation," 2025. [Online]. Available: <https://www.jenkins.io/doc/book/pipeline/> (Citato a pagina 23)
- [11] A. W. Services, "Cos'è il cloud computing?" 2025. [Online]. Available: https://docs.aws.amazon.com/it_it/whitepapers/latest/aws-overview/what-is-cloud-computing.html (Citato alle pagine 25, 26 e 28)
- [12] Amazon Web Services, "Differenza tra saas, paas e iaas - tipi di cloud computing," 2025. [Online]. Available: <https://aws.amazon.com/it/types-of-cloud-computing/> (Citato a pagina 26)
- [13] AWS, "Amazon ec2 instance types," Amazon Web Services. [Online]. Available: <https://aws.amazon.com/it/ec2/instance-types/> (Citato a pagina 59)
- [14] AWS, "Cos'è un eleastic beanstalk," Amazon Web Services. [Online]. Available: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg>Welcome.html> (Citato a pagina 60)
- [15] AppMaster, "Cosa significa 'debito tecnico' nella progettazione del software?" [Online]. Available: <https://appmaster.io/it/blog/cosa-significa-debito-tecnico-nella-progettazione-del-software> (Citato a pagina 63)
- [16] SonarSource, "Sonarqube documentation." [Online]. Available: <https://docs.sonarsource.com> (Citato a pagina 64)

- [17] AWS, “Aws lambda,” Amazon Web Services, 2025. [Online]. Available: <https://aws.amazon.com/it/lambda/> (Citato a pagina 72)

Ringraziamenti

Desidero ringraziare il mio relatore, il **prof. Gravino**, per i consigli, l'impegno profuso nello stilare questa tesi, per essere stato sempre disponibile e tempestivo nel risolvere le mie incertezze.

Grazie a **System Management**, azienda che mi ha ospitato per qualche mese nella quale ho sviluppato questo progetto con l'ausilio del mio tutor, per l'ambiente nel quale mi ha accolto e per le pause nelle quali si rideva e si scherzava con i colleghi tra i vari, non posso non menzionare *Giovanni, Andrea e Carlo*.

Grazie ad **Igor**, il mio tutor in azienda, perché alle volte si dice che credere in sé stessi è alla base, ma aver avuto un tutor che ha creduto in me, simpatico, preparatissimo e sempre disponibile a risolvere ogni mio dubbio è stato un valore aggiunto non indifferente. Grazie per avermi accolto e fatto capire che essere curioso, fare domande e voler approfondire non è un difetto ma una caratteristica della quale andare fieri, grazie davvero perchè senza il tuo aiuto questo progetto non sarebbe mai esistito.