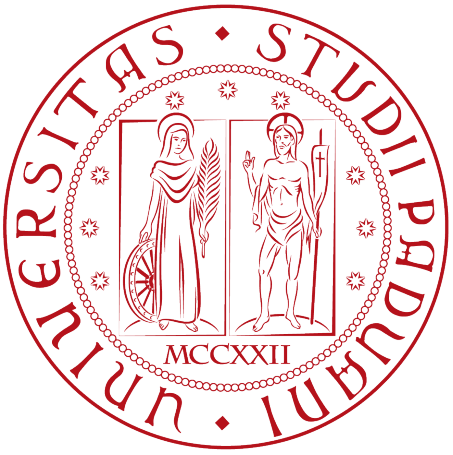
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo di una piattaforma di monitoraggio del traffico utente mediante APM e IA

Tesi di laurea

Relatore
Prof. Marco Zanella

Laureando
Marco Cola
Matricola 2079237

Anno Accademico 2024-2025

Temet Nosce

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di *stage*, della durata di 320 ore, dal laureando Marco Cola presso l'azienda Kirey S.r.l.
L'obiettivo principale dello *stage* è stato lo sviluppo di una piattaforma per il monitoraggio e l'analisi dei dati di navigazione relativi ad una *web application*, con particolare attenzione alla raccolta, indicizzazione ed elaborazione dei *log*.
Il progetto si è articolato in quattro fasi principali:

- Una prima fase di preparazione dell'ambiente di lavoro, comprendente l'individuazione, l'installazione e la configurazione delle componenti *software* necessarie, con successiva verifica del corretto funzionamento e della connettività tra i moduli;
- Una seconda fase di implementazione dell'estrazione dei dati, realizzata attraverso la configurazione di agenti di raccolta, la creazione di *pipeline* di *log* tramite *Logstash* e l'invio dei dati a *Elasticsearch*, con validazione del processo di acquisizione e indicizzazione;
- Una terza fase di elaborazione e rappresentazione grafica dei dati, che ha previsto lo sviluppo di *script* per la generazione di traffico utente e monitoraggio, l'analisi e aggregazione dei dati su *Elasticsearch* e la realizzazione di *dashboard* avanzate in *Kibana* con metriche di *performance*, accesso e flussi utente. Sono state inoltre configurate regole di *alerting* e notifiche per la rilevazione in tempo reale di anomalie mediante *Machine Learning*;
- Una fase finale di documentazione tecnica del progetto, contenente la descrizione delle tecnologie e dei prodotti utilizzati, la rappresentazione dei flussi logici dell'applicazione, nonché un'analisi dei pro e contro di ciascuna componente e delle principali criticità riscontrate.

Lo *stage* ha permesso di acquisire competenze trasversali nell'ambito dell'osservabilità delle applicazioni *web*, approfondendo l'utilizzo dello *stack Elastic* e l'integrazione con strumenti di monitoraggio automatizzato, con un approccio volto alla creazione di soluzioni scalabili, robuste e orientate al miglioramento continuo delle prestazioni.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Marco Zanella, relatore della mia tesi, per l'aiuto costante e il sostegno puntuale fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori e mia sorella Sara per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Un grazie anche a Benedetta, per esserci sempre stata, con pazienza e amore.

Desidero poi ringraziare tutta la mia famiglia per il supporto incondizionato e l'amore che mi hanno sempre dimostrato.

Infine, un ringraziamento speciale va agli amici incontrati durante il percorso e quelli che già c'erano, per aver reso questa esperienza unica e indimenticabile.

Padova, Dicembre 2025 Marco Cola

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

Kirey Group è uno dei *system integrator* europei più dinamici e in crescita, specializzato nell'accompagnare le imprese nei percorsi di trasformazione digitale e di adozione di modelli *data-driven*.
Con sede principale in Italia e una presenza consolidata in diversi paesi europei ed extraeuropei, il gruppo conta oltre 1500 dipendenti ed opera in dieci paesi.
La missione di Kirey è rendere l'innovazione accessibile, trasformando il potenziale tecnologico in valore economico e in nuovi modelli di *business*.
L'azienda si distingue per un approccio che unisce affidabilità tecnica, innovazione, competenza centrata sul lavoro delle persone e sinergia cross-funzionale, elementi che costituiscono i valori fondanti del marchio.
Il manifesto del gruppo sintetizza questa filosofia nel concetto "*Data Made Human*", ovvero la volontà di tradurre la complessità dei dati in soluzioni comprensibili, intuitive e ad alto impatto, mettendo sempre la persona al centro della tecnologia.

Figura 1.1: Logo di Kirey S.r.l.

1.1.1 Storia e servizi

La storia del gruppo affonda le radici negli anni settanta e, attraverso fusioni, acquisizioni e nuove fondazioni, ha portato alla nascita di Kirey Group nel 2016.
Negli anni successivi l'azienda ha accelerato la propria espansione internazionale integrando nuove realtà, consolidando così competenze e capacità operative in diversi

settori e mercati.
Il portafoglio di servizi è ampio e integrato, con *Data & AI* come filo conduttore e aree principali che comprendono:

- *Cloud & Infrastructure*, con soluzioni ibride e *on-premise*, sicurezza in ambienti *cloud*, migrazione e monitoraggio;
- *Software Development*, che spazia dallo sviluppo *agile* e *mobile* alla *system integration*, con particolare attenzione alla qualità e all'automazione dei *test*;
- *Cybersecurity*, con servizi di consulenza, *audit*, architetture sicure, *managed services* e sistemi antifrode;
- *Data & AI*, che include *data integration*, *data governance*, *analytics*, *machine learning*, *synthetic data*, *forecasting* e soluzioni ESG[g].

Kirey Group pone grande attenzione alla sostenibilità, alla trasparenza e all'integrità, adottando pratiche responsabili nei confronti di clienti, partner, dipendenti e *stakeholder*.
L'azienda è inoltre attivamente impegnata in progetti sociali, promuove la diversità e l'inclusione, e investe nello sviluppo delle competenze tecnologiche e professionali dei propri collaboratori.
Oggi il gruppo conta oltre 1370 casi di *business* realizzati, 10 *Innovation Center* attivi, un fatturato di circa 126 milioni di euro e più di 1500 collaboratori distribuiti in 10 paesi.

1.2 L'idea

L'idea alla base dello stage consiste nello sviluppo di una piattaforma per il monitoraggio intelligente del traffico utente di una *web application*. L'obiettivo principale è quello di sfruttare algoritmi di intelligenza artificiale e di *Machine Learning* per individuare e segnalare automaticamente eventuali anomalie nei dati raccolti.
La piattaforma sarà in grado di analizzare i flussi in tempo reale, rilevando accessi sospetti, rallentamenti e potenziali minacce, così da consentire interventi tempestivi e garantire sia la sicurezza sia le prestazioni ottimali del sistema.
Il progetto è stato organizzato in quattro fasi principali:

- Una prima fase di formazione e preparazione dell'ambiente di lavoro, utile a familiarizzare con le tecnologie e i prodotti utilizzati, che comprende l'individuazione, l'installazione e la configurazione delle componenti necessarie, con successiva verifica della connettività tra i moduli;
- Una fase di analisi e progettazione, in cui saranno definite le specifiche funzionali e la soluzione tecnica tramite la configurazione degli agenti di raccolta, la realizzazione di *pipeline* di *log* e la loro indicizzazione in *Elasticsearch*;
- Una fase di realizzazione e *test* della piattaforma, con sviluppo di *script* per il monitoraggio sintetico, analisi su *Elasticsearch*, creazione di *dashboard* in *Kibana* e configurazione di regole di *alerting*;
- Infine la stesura della documentazione tecnica e funzionale, contenente la descrizione delle tecnologie adottate, dei flussi logici implementati e delle criticità riscontrate.

Per lo sviluppo saranno impiegati linguaggi come *Python* e *Java*, sistemi *Linux* e i prodotti della suite *Elastic Stack*, strumenti particolarmente adatti per l'elaborazione e il monitoraggio di grandi volumi di dati in tempo reale.

1.3 Organizzazione del testo

Il secondo capitolo approfondisce il progetto di stage, descrivendo gli obiettivi e le attività svolte, la metodologia di lavoro adottata e un'analisi preventiva dei principali rischi e relative strategie di mitigazione;

Il terzo capitolo è dedicato all'analisi dei requisiti della piattaforma e alle motivazioni che ne hanno guidato le scelte progettuali, fornendo al contempo una mappatura completa e strutturata dei requisiti individuati per il sistema;

Il quarto capitolo approfondisce le tecnologie e i principi teorici alla base della soluzione proposta, analizzando i principali strumenti e approcci per il monitoraggio delle prestazioni applicative;

Il quinto capitolo approfondisce lo sviluppo del prodotto, descrivendo l'architettura complessiva del sistema, le fasi di implementazione e i risultati ottenuti;

Nel sesto capitolo viene descritta l'esperienza complessiva dello stage, con una riflessione sui risultati raggiunti e le competenze acquisite, oltre a suggerimenti per possibili sviluppi futuri del progetto.

1.3.1 Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
- i comandi, le query, i percorsi di file e il codice inline sono rappresentati tramite il carattere `monospaziato`.

Capitolo 2

Descrizione dello stage

Il capitolo approfondisce il progetto di stage, descrivendo gli obiettivi e le attività svolte, la metodologia di lavoro adottata e un'analisi preventiva dei principali rischi e relative strategie di mitigazione.

2.1 Introduzione al progetto

Lo *stage* è stato svolto presso l'azienda Kirey Group S.r.l., realtà consolidata nell'ambito della fornitura di prodotti e servizi informatici, con clienti internazionali e una forte specializzazione nei settori *Banking*, *Insurance*, *Oil&Gas* e Pubblica Amministrazione. L'attività si è inserita nel contesto del team APM[g]e ha avuto come obiettivo principale la realizzazione e il collaudo di una piattaforma per il monitoraggio delle performance di una *web application*.
Il progetto è stato sviluppato interamente in ambiente *Linux* mediante WSL[g], utilizzando la *suite Elastic Stack* e i suoi principali componenti:

- *Elasticsearch* per la gestione e indicizzazione dei dati;
- *Kibana* per la visualizzazione;
- *Logstash* per l'ingestione e la trasformazione dei log;
- *Fleet Server* per la raccolta distribuita delle metriche e la gestione degli agenti;
- *APM Server/Agent* per il tracciamento delle *performance* applicative.

Sono stati realizzati sviluppi in *Python* e *Java* per l'estensione delle funzionalità e l'integrazione di algoritmi di AI[g]e *Machine Learning*, con l'obiettivo di rilevare automaticamente anomalie e problematiche di prestazione.
La finalità complessiva è quella di fornire un sistema scalabile, proattivo e ben documentato, capace di garantire prestazioni ottimali e un monitoraggio continuo della *web application*.

2.2 Pianificazione

Tutte le attività sono state condotte in affiancamento ad un *tutor* aziendale che ha curato sia la parte di formazione che di indirizzamento delle attività.
A tal fine sono stati svolti dei momenti di confronto settimanali per la valutazione dello stato di avanzamento delle attività e momenti quotidiani di confronto sulle problematiche riscontrate.
Le attività proposte sono state collocate all'interno di un progetto più ampio portato avanti in Kirey da un *team* di persone eterogeneo.
Al termine dello *stage* sono stati presentati i risultati ottenuti a tutto il *team*.
L'infrastruttura tecnologica e le piattaforme su cui girerà l'applicazione sono state messe a disposizione da Kirey.

2.3 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Inesperienza nella suite Elastic

Descrizione: La limitata esperienza iniziale nell'utilizzo della *Elastic Stack* (*Elasticsearch*, *Kibana*, *Logstash*, *APM*) potrebbe comportare difficoltà nella configurazione e nell'integrazione delle componenti, rallentando lo sviluppo del progetto.

Impatto: Alto.

Probabilità: Alta.

Soluzione: Organizzazione di momenti di confronto con il *tutor* aziendale e studio personale della documentazione, al fine di acquisire le competenze necessarie.

2. Integrazione tra componenti Elastic

Descrizione: La comunicazione tra i diversi moduli della *suite Elastic* (*Elasticsearch*, *Kibana*, *Logstash*, *APM*) potrebbe presentare problemi di configurazione, causando ritardi o malfunzionamenti nell'acquisizione dei dati.

Impatto: Medio.

Probabilità: Media.

Soluzione: Esecuzione di test di connettività e validazione progressiva delle *pipeline*, con il supporto del *tutor* aziendale per la risoluzione dei problemi.

3. Qualità e coerenza dei dati raccolti

Descrizione: I dati acquisiti dagli agenti potrebbero risultare incompleti, duplicati o non coerenti, compromettendo le analisi e le *dashboard*.

Impatto: Alto.

Probabilità: Media.

Soluzione: Definizione di regole di filtraggio e validazione all'interno delle *pipeline Logstash* ed esecuzione di test di integrità sugli indici *Elasticsearch*.

4. Scalabilità e carico del sistema

Descrizione: L'aumento del volume dei dati e delle richieste potrebbe impattare sulle prestazioni della piattaforma, riducendo l'efficienza del monitoraggio.

Impatto: Medio.

Probabilità: Media.

Soluzione: Implementazione di strategie di *scaling* orizzontale e utilizzo di metriche di *Kibana* per monitorare l'impatto del carico in tempo reale.

5. Implementazione di algoritmi di Machine Learning

Descrizione: L'integrazione di modelli di *Machine Learning* per il rilevamento delle anomalie potrebbe richiedere competenze specifiche e tempi di sviluppo più lunghi del previsto.

Impatto: Alto.

Probabilità: Media.

Soluzione: Formazione preliminare su tecniche di *Machine Learning* e utilizzo di librerie e *framework* consolidati per accelerare lo sviluppo.

6. Problemi di configurazione delle pipeline Logstash

Descrizione: Errori di configurazione nelle *pipeline* di *Logstash* potrebbero causare la perdita, la duplicazione o la trasformazione errata dei dati raccolti, compromettendo l'affidabilità delle analisi.

Impatto: Alto.

Probabilità: Media.

Soluzione: Adozione di un approccio con *test* di validazione a ogni modifica delle *pipeline* e utilizzo di ambienti di prova per verificare la correttezza del flusso dei dati prima della messa in produzione.

7. Accesso limitato a funzionalità premium di Elastic

Descrizione: Durante le prime settimane di lavoro in locale, la mancata possibilità di operare su *Elastic Cloud* e di accedere a funzionalità *premium* come il *Machine Learning* potrebbe limitare l'analisi dei dati e rallentare la validazione di alcune funzionalità previste dal progetto.

Impatto: Medio.

Probabilità: Bassa.

Soluzione: Esecuzione preventiva delle attività in locale con dati di *test*, studio della documentazione sulle funzionalità *premium* e pianificazione di un passaggio successivo a *Elastic Cloud* non appena disponibile, con supporto del *tutor* aziendale.

2.4 Requisiti e obiettivi

Gli obiettivi del progetto sono stati classificati in base alla loro priorità secondo le seguenti notazioni:

- **Ob** (Requisiti Obbligatori) - requisiti essenziali e imprescindibili per il successo del progetto, vincolanti in quanto obiettivo primario richiesto dal committente;
- **D** (Requisiti Desiderabili) - requisiti importanti ma non critici, la cui assenza non compromette il progetto, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- **Op** (Requisiti Opzionali) - requisiti desiderabili ma non essenziali, rappresentanti valore aggiunto non strettamente competitivo.

Durante il periodo di stage si prevede il raggiungimento dei seguenti obiettivi:

- **Preparazione dell'ambiente per l'implementazione della soluzione:**
  - **Ob1.1:** Individuazione delle componenti ed eventuali librerie da utilizzare;
  - **Ob1.2:** Installazione e configurazione delle componenti;
  - **Ob1.3:** Verifica del corretto funzionamento dell'ambiente e *test* di connettività tra componenti.
- **Implementazione estrazione dati dalla web application:**
  - **Ob2.1:** Configurazione *agent* (*Beats/APM*) per la raccolta dati della navigazione;
  - **Ob2.2:** Implementazione *pipeline* di *log* tramite *Logstash* per filtraggio e inoltro dati in *Elasticsearch*;
  - **Ob2.3:** Verifica della corretta acquisizione dei dati e loro indicizzazione in *Elasticsearch*.
- **Implementazione elaborazione dati e rappresentazione grafica dei dati:**
  - **Ob3.1:** Sviluppo *script Python/Java* per il monitoraggio sintetico (*Selenium*) e generazione di traffico *log*;
  - **Ob3.2:** Analisi e aggregazione dati in *Elasticsearch*, con *query* e visualizzazioni preliminari;
  - **Ob3.3:** Creazione *dashboard* avanzate su *Kibana* con metriche di *performance*, accesso e flussi utente;
  - **Op3.4:** Configurazione regole di *alerting* e notifiche per anomalie rilevate in tempo reale.
- **Documentazione dettagliata:**
  - **Ob4.1:** Descrizione delle tecnologie e prodotti utilizzati;
  - **Ob4.2:** Descrizione dei flussi logici del progetto e delle funzionalità dell'applicazione;
  - **D4.3:** Pro/contro di ogni componente e criticità nell'applicazione.

Nel capitolo successivo verrà approfondita l'analisi dei requisiti identificati in questa sezione, al fine di definirne in modo più strutturato la natura e le caratteristiche.

Capitolo 3

Analisi dei requisiti

Il capitolo è dedicato all'analisi dei requisiti della piattaforma, con l'obiettivo di fornire una visione completa e dettagliata delle funzionalità e delle caratteristiche attese dal sistema. Verranno illustrate le esigenze degli utenti e del contesto operativo, evidenziando le specifiche tecniche e le funzionalità che hanno guidato le scelte progettuali.

3.1 Requisiti del sistema

A seguito di un'attenta attività di analisi del progetto e degli obiettivi tecnici e funzionali prefissati, sono state redatte le tabelle di tracciamento che riassumono in modo strutturato i requisiti individuati.
Durante questa fase sono state identificate differenti tipologie di requisiti, distinte sia in base alla loro categoria (funzionale, non funzionale, qualitativo o di vincolo), sia in base alla loro priorità di implementazione (obbligatorio, desiderabile o opzionale).
Per garantire una tracciabilità chiara e univoca, a ciascun requisito è stato assegnato un codice identificativo composto da lettere che ne descrivono la tipologia e l'importanza, secondo la seguente convenzione:

R = Requisito

F = Funzionale

N = Non funzionale

Q = Qualitativo

V = Vincolo

O = Obbligatorio

D = Desiderabile

Z = Opzionale

Ogni requisito analizzato sarà identificato univocamente dalla seguente notazione:

R[Priorità][Categoria]-[Numero]

Dove:

- **Priorità** indica la priorità di implementazione, che può essere O = Obbligatorio, D = Desiderabile, Z = Opzionale;
- **Categoria** la categoria di appartenenza, che può essere F = Funzionale, N = Non funzionale, Q = Qualitativo, V = Vincolo;
- **Numero** un numero progressivo che identifica in modo univoco il requisito all'interno della sua categoria.

Nelle tabelle 3.1, 3.2, 3.3 e 3.4 sono riportati in modo sintetico tutti i requisiti emersi dall'analisi, classificati in base alla loro priorità e accompagnati da una breve descrizione della relativa funzionalità o vincolo tecnico.

3.1.1 Requisiti funzionali

I requisiti funzionali descrivono cosa deve fare il sistema. Sono le funzionalità concrete che la soluzione deve offrire per raggiungere gli obiettivi del progetto.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Codice	Descrizione	Classificazione
ROF-1	Il sistema deve permettere la raccolta automatica di metriche e *log* relativi alla *web application* tramite agenti *OpenTelemetry* o *Elastic APM*.	Obbligatorio
ROF-2	Il sistema deve inviare i dati raccolti agli *endpoint Elasticsearch* per l'analisi e l'indicizzazione.	Obbligatorio
ROF-3	Il sistema deve consentire la creazione di *pipeline* di *log* tramite *Logstash* per filtraggio, trasformazione e inoltro dei dati in *Elasticsearch*.	Obbligatorio
ROF-4	Il sistema deve prevedere la configurazione di *Elastic Agents* (*Beats*/*APM*) per la raccolta dati della navigazione.	Obbligatorio
ROF-5	Il sistema deve generare *dashboard* avanzate e visualizzazioni in *Kibana*, con metriche di *performance*, accesso e flussi utente.	Obbligatorio
ROF-6	Il sistema deve permettere la verifica della corretta acquisizione dei dati e la loro indicizzazione in *Elasticsearch*.	Obbligatorio
ROF-7	Il sistema deve prevedere lo sviluppo e l'esecuzione di *script* automatizzati in *Python* o *Java* per la simulazione del traffico utente (*Synthetic Monitoring*).	Obbligatorio
ROF-8	Deve essere possibile filtrare e ricercare i *log* per *host*, servizio, livello di severità o periodo temporale.	Obbligatorio
RDF-9	Il sistema dovrebbe prevedere la configurazione di regole di *alerting* e notifiche in tempo reale per anomalie rilevate.	Desiderabile
RDF-10	Il sistema dovrebbe integrare algoritmi di *Machine Learning* per l'individuazione automatica di anomalie.	Desiderabile
RDF-11	Il sistema dovrebbe consentire l'esportazione delle *dashboard* o dei risultati delle *query* in formato PDF[g]o CSV[g].	Desiderabile
RZF-12	Il sistema può prevedere un modulo aggiuntivo per la generazione automatica di *report* periodici delle metriche raccolte.	Opzionale
RZF-13	Il sistema può consentire l'importazione automatica delle configurazioni APM da ambienti di *test* o *staging*.	Opzionale

3.1.2 Requisiti non funzionali

I requisiti non funzionali definiscono come il sistema deve comportarsi, cioè le sue proprietà di qualità interna.
Non aggiungono nuove funzioni, ma impongono vincoli di prestazioni, sicurezza, disponibilità, scalabilità, affidabilità e manutenibilità.

Tabella 3.2: Tabella del tracciamento dei requisiti non funzionali

Codice	Descrizione	Classificazione
RON-1	Il sistema deve essere scalabile e consentire l'aggiunta di nuove fonti di dati o agenti senza compromettere la stabilità.	Obbligatorio
RON-2	Il sistema deve garantire l'affidabilità nella trasmissione e nella conservazione dei dati raccolti.	Obbligatorio
RON-3	La piattaforma deve assicurare un tempo di latenza accettabile nella visualizzazione dei dati ($<$ 5 secondi per l'aggiornamento delle *dashboard*).	Obbligatorio
RDN-4	Il sistema dovrebbe garantire la possibilità di eseguire *test* di carico e stress per valutare la stabilità dell'ambiente.	Desiderabile
RDN-5	Il sistema dovrebbe supportare l'autenticazione per la gestione degli accessi a *Kibana*.	Desiderabile

3.1.3 Requisiti qualitativi

I requisiti qualitativi specificano le proprietà qualitative che influenzano l'esperienza d'uso e la manutenibilità.
Si concentrano su aspetti percepibili, come semplicità, chiarezza, flessibilità o estendibilità.

Tabella 3.3: Tabella del tracciamento dei requisiti qualitativi

Codice	Descrizione	Classificazione
ROQ-1	L'interfaccia di *Kibana* deve offrire una rappresentazione chiara e intuitiva delle metriche principali.	Obbligatorio
ROQ-2	I dati devono essere visualizzabili in forma aggregata e filtrabile in base a intervalli temporali e categorie di evento.	Obbligatorio
ROQ-3	Le *dashboard* devono presentare una chiara distinzione cromatica tra metriche positive, neutre e anomale.	Obbligatorio
RDQ-4	Le *dashboard* dovrebbero essere personalizzabili dall'utente secondo criteri di interesse (*performance*, accessi, flussi).	Desiderabile
RZQ-5	Il sistema può includere un *layout dark/light mode* o temi grafici personalizzati per una migliore leggibilità.	Opzionale

3.1.4 Requisiti di vincolo

Impongono limitazioni o condizioni esterne al progetto: ambienti, tecnologie, compatibilità, strumenti, *standard* aziendali o legali.

Tabella 3.4: Tabella del tracciamento dei requisiti di vincolo

Codice	Descrizione	Use Case
ROV-1	Il sistema deve utilizzare i prodotti della suite *Elastic Stack* (*Elasticsearch, Logstash, Kibana, Beats, APM Server/Agent*).	Obbligatorio
ROV-2	L'ambiente operativo deve essere *Linux* (*Red Hat* o distribuzioni equivalenti).	Obbligatorio
ROV-3	Tutti i componenti *software* devono essere compatibili con la versione di *Linux* installata (es. *Ubuntu 22.04 LTS* o *Red Hat 9*).	Obbligatorio
ROV-4	Le componenti devono rispettare le seguenti versioni minime: • *Python* >= 3.10 • *Java* >= 16 • *Node.js* >= 17 • *Logstash* >= 8.10 • *Kibana* >= 8.10 • *Beats* >= 8.10 • *APM Server* >= 8.10 • *Elasticsearch* >= 8.10 • *OpenTelemetry Java Agent* >= 1.26.0	Obbligatorio
ROV-5	La *web application* deve essere compatibile con i principali *browser* (*Chrome* >= 120, *Firefox* >= 115, *Edge* >= 120, *Safari* >= 15).	Obbligatorio
ROV-6	Le configurazioni devono essere eseguite in ambiente *Docker* o su infrastruttura aziendale.	Obbligatorio
RDV-7	La documentazione tecnica deve essere redatta in *Markdown*, *LaTeX* o PDF.	Desiderabile
RDV-8	Il sistema dovrebbe supportare la distribuzione tramite *Docker Compose*.	Desiderabile

3.2 Riepilogo dei requisiti

Tabella 3.5: Riepilogo dei requisiti

Tipologia	Obbligatorio	Desiderabile	Opzionale	Totale
Funzionali	8	3	2	13
Non funzionali	3	2	0	5
Qualitativi	3	1	1	5
Di Vincolo	6	2	0	8
Totale	20	8	3	31

3.3 Rappresentazione architetturale preliminare

La rappresentazione architetturale descrive la struttura della soluzione proposta, evidenziando le principali componenti coinvolte nel sistema di APM e le loro relazioni. Essa costituisce il collegamento tra la fase di analisi dei requisiti e la successiva implementazione pratica. I dettagli tecnici e i diagrammi della soluzione verranno approfonditi nel capitolo dedicato allo sviluppo della piattaforma.

3.3.1 Struttura generale della soluzione di monitoraggio

La soluzione di APM proposta si fonda su un'architettura ibrida che combina le funzionalità di *OpenTelemetry* con la potenza analitica dello *Stack Elastic*, fornendo un sistema di osservabilità completo.
L'applicazione di riferimento, *PetClinic*, viene monitorata sul lato *backend* tramite l'*OpenTelemetry Java Agent*, avviato insieme all'applicazione mediante parametri `-Dotel`. L'agente intercetta automaticamente chiamate HTTP[g], operazioni sul *database* e transazioni applicative, generando le relative tracce e metriche.
A differenza di un'integrazione diretta, i dati raccolti non vengono inviati immediatamente all'*APM Server*, ma passano prima attraverso l'*OpenTelemetry Collector*. Il *Collector* funge da livello intermedio dedicato all'elaborazione e arricchimento dei dati di telemetria, oltre a gestire la traduzione del formato in modo efficiente.
Una volta processati, i dati vengono inoltrati all'*APM Server* (integrato nell'*Elastic Agent* gestito tramite *Fleet*) utilizzando il protocollo OTLP[g].
L'*APM Server* riceve i dati normalizzati dal *Collector*, li converte nel formato interno di *Elastic* e li invia a *Elasticsearch* per l'indicizzazione.
Parallelamente, il monitoraggio del *frontend* è gestito dal *RUM Agent JavaScript*, che traccia interazioni utente, tempi di caricamento, prestazioni percepite nel *browser* ed eventuali errori *JavaScript*. Il *RUM Agent* comunica direttamente con l'*APM Server*, senza passare dal *Collector*.
Tutti i dati prodotti dai vari agenti convergono infine in *Elasticsearch*, dove vengono memorizzati e resi disponibili alla consultazione tramite *Kibana*.

3.3.2 Integrazione dei componenti principali

L'architettura della soluzione di monitoraggio si compone dei seguenti moduli principali:

- **Elasticsearch:** motore di ricerca e analisi distribuito che gestisce la memorizzazione, l'indicizzazione e l'interrogazione dei dati provenienti dagli agenti di monitoraggio in tempo reale;

- **Kibana:** interfaccia di visualizzazione e analisi integrata con *Elasticsearch*, consente di visualizzare metriche e tracce applicative, creare *dashboard* personalizzate e configurare regole di *alerting* per il monitoraggio del sistema;
- **Fleet:** componente centralizzato per la gestione e la configurazione degli agenti *Elastic*. Attraverso il *Fleet Server*, coordina la comunicazione con gli *Elastic Agents*;
- **Elastic Agent:** agente unificato che integra funzionalità di raccolta dati, sicurezza e monitoraggio. Gestito tramite *Fleet*, può essere configurato per includere moduli specifici come *APM Server* e *Beats* per la raccolta di metriche di sistema e *log*;
- **APM Server:** incluso all'interno dell'*Elastic Agent*, riceve i dati di telemetria provenienti dagli agenti applicativi. Supporta lo standard *OpenTelemetry*, fungendo da *endpoint* compatibile per la ricezione dei dati raccolti dal *Java Agent*;
- **OpenTelemetry Collector:** componente intermedio che riceve i dati di telemetria dall'*OpenTelemetry Java Agent*, li elabora e li inoltra all'*APM Server* tramite OTLP;
- **OpenTelemetry Java Agent:** agente di strumentazione automatica avviato con l'applicazione *PetClinic*, utilizzato per raccogliere metriche e tracce delle transazioni lato *backend* (chiamate HTTP, query al *database*, eccezioni, tempi di risposta). I dati generati vengono inviati all'*APM Server* attraverso il protocollo OTLP;
- **RUM Agent JavaScript:** agente di monitoraggio RUM eseguito nel *frontend* dell'applicazione, che raccoglie dati sulle interazioni utente, sui tempi di caricamento delle pagine e sulle prestazioni percepite nel *browser*;

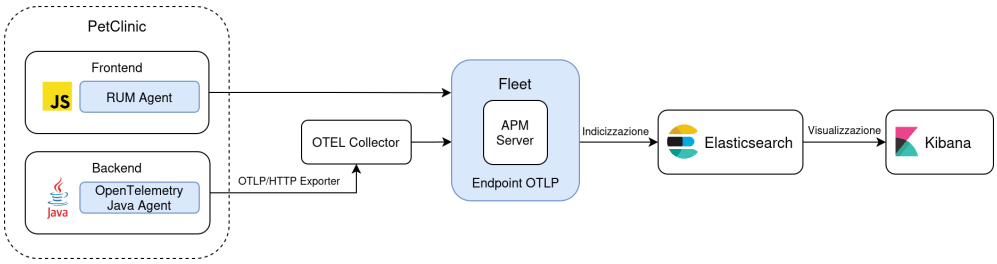


Figura 3.1: Integrazione tra componenti del sistema di monitoraggio

3.3.3 Applicazione di riferimento: Spring PetClinic

Per la sperimentazione del sistema di Application Performance Monitoring (APM) è stata utilizzata come applicazione di riferimento *Spring PetClinic*, una *web application open source* sviluppata in *Java* e basata sul *framework Spring Boot*.
La scelta di *PetClinic* è motivata dalla sua architettura chiara e ben documentata, composta da un livello di presentazione (*controller* e *template*), un livello logico (servizi) e un livello di persistenza (*repository* e *database* MySQL[g]).
Tale struttura consente di osservare comportamenti applicativi realistici, come chiamate HTTP, interazioni con il *database* e logiche di *business*, fornendo un contesto ideale per testare le funzionalità di monitoraggio offerte da *Elastic APM* e *OpenTelemetry*.

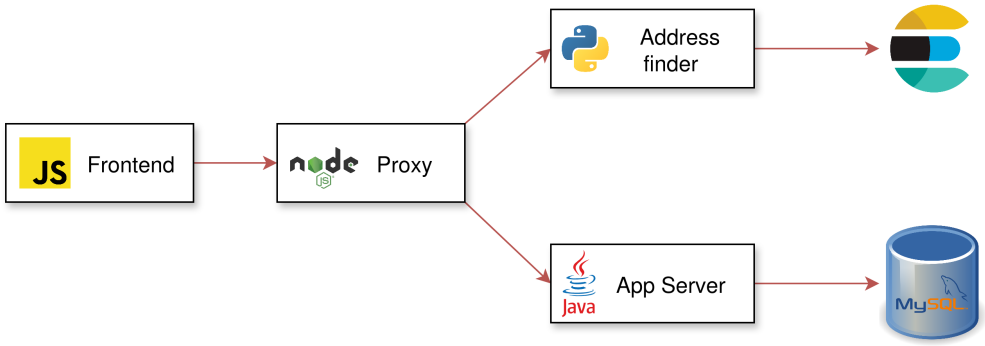


Figura 3.2: Architettura di Spring PetClinic

L'applicazione *PetClinic* verrà quindi utilizzata come caso di studio per la validazione della soluzione di monitoraggio proposta, analizzando successivamente il modo in cui i diversi moduli architetturali interagiscono con essa.

3.4 User Stories e scenari di utilizzo

Le seguenti *user stories* descrivono brevemente i principali scenari di utilizzo del sistema di APM, evidenziando gli obiettivi e le esigenze degli utenti tecnici aziendali che operano sulla piattaforma di monitoraggio.
L'unico attore considerato è l'*Observability Engineer*, figura responsabile della configurazione, analisi e manutenzione del sistema all'interno dell'azienda.

Tabella 3.6: User Stories

ID	User Story	Obiettivo
US1	Come *Observability Engineer*, voglio analizzare i tempi medi di risposta degli *endpoint* dell'applicazione per individuare colli di bottiglia o degradi di *performance*.	Ottimizzare le prestazioni applicative.
US2	Come *Observability Engineer*, voglio monitorare l'utilizzo di CPU[g], memoria e connessioni per valutare la stabilità del sistema.	Garantire la corretta gestione delle risorse.

ID	User Story	Obiettivo
US3	Come *Observability Engineer*, voglio rilevare e classificare errori e anomalie dai *log* per identificare rapidamente le cause dei malfunzionamenti.	Migliorare l'affidabilità del sistema.
US4	Come *Observability Engineer*, voglio configurare regole di *alerting* in *Kibana* per essere notificato in caso di anomalie.	Ridurre i tempi di risposta agli incidenti.
US5	Come *Observability Engineer*, voglio visualizzare le metriche aggregate in *dashboard* personalizzate.	Facilitare l'analisi e la comunicazione dei risultati.
US6	Come *Observability Engineer*, voglio visualizzare in *Kibana* le metriche di prestazione del *backend* (tempi medi di risposta, *throughput*, tasso di errore, durata media delle transazioni) per valutare lo stato di salute dell'applicazione.	Monitorare le prestazioni e la stabilità del servizio.
US7	Come *Observability Engineer*, voglio visualizzare metriche di utilizzo delle risorse di sistema (CPU, memoria, disco, rete) per individuare potenziali colli di bottiglia.	Ottimizzare l'efficienza del sistema.
US8	Come *Observability Engineer*, voglio analizzare in *Kibana* le metriche relative al comportamento degli utenti (*page load time*, errori *JavaScript*, interazioni lente) raccolte dal *RUM Agent*.	Migliorare l'esperienza utente.
US9	Come *Observability Engineer*, voglio correlare in un'unica *dashboard* le metriche di infrastruttura, *application performance* e *user experience*, per ottenere una visione completa del sistema.	Favorire un'analisi integrata e diretta delle anomalie.
US10	Come *Observability Engineer*, voglio integrare l'*OpenTelemetry Java Agent* e il *RUM Agent JavaScript* per raccogliere rispettivamente metriche di *backend* e *frontend*.	Avere una visione completa del comportamento dell'applicazione.
US11	Come *Observability Engineer*, voglio configurare e gestire gli agenti tramite *Fleet* per garantire una distribuzione e un aggiornamento centralizzati.	Semplificare la manutenzione e ridurre gli errori di configurazione.
US12	Come *Observability Engineer*, voglio eseguire *test* sintetici periodici per monitorare la disponibilità e i tempi di risposta delle pagine.	Garantire la continuità operativa.
US13	Come *Observability Engineer*, voglio esportare le *dashboard* in formato PDF o CSV per la condivisione all'interno del *team*.	Documentare e condividere le analisi in modo efficace.
US14	Come *Observability Engineer*, voglio ricevere notifiche automatiche via *email* quando vengono superate determinate soglie di prestazione.	Automatizzare la gestione degli incidenti e migliorare il tempo di reazione.

ID	User Story	Obiettivo
US15	Come *Observability Engineer*, voglio configurare in *Kibana job* di *Machine Learning* per l'individuazione automatica di anomalie.	Rilevare comportamenti anomali senza intervento manuale.
US16	Come *Observability Engineer*, voglio visualizzare in *Kibana* i risultati dei *job* di *Machine Learning* (anomalie e previsioni) integrati nelle *dashboard*.	Integrare l'analisi automatica con la visualizzazione interattiva dei dati.

Capitolo 4

Tecnologie e principi teorici

Il capitolo presenta il quadro teorico e tecnologico di riferimento del progetto, descrivendo i principali approcci e strumenti per il monitoraggio delle prestazioni applicative. Vengono illustrate le tecnologie analizzate e le motivazioni che hanno guidato la scelta della soluzione, in relazione ai requisiti e agli obiettivi del sistema di osservabilità sviluppato.

4.1 APM e Observability

L'osservabilità di un sistema si basa sull'analisi congiunta di tre pilastri fondamentali: metriche, *log* e tracce.

- **Logs**: un *log* è un *record* testuale di un evento ad un orario specifico, come un tentativo di accesso, un errore di sistema o una transazione completata. I *log* forniscono dettagli di contesto che aiutano a diagnosticare problemi specifici;
- **Metriche**: una metrica è una misurazione numerica di un aspetto specifico delle prestazioni del sistema, come l'utilizzo della CPU, la latenza delle richieste o il numero di utenti attivi;
- **Tracce**: una traccia rappresenta il percorso di una singola richiesta attraverso i vari componenti di un sistema distribuito, consentendo di visualizzare il flusso delle operazioni e identificare i colli di bottiglia.

Un sistema APM moderno integra questi aspetti per fornire una visione completa dello stato e del comportamento di un applicazione. Nel contesto del monitoraggio di applicazioni *web* come *PetClinic*, l'osservabilità consente di analizzare i dati di *performance* raccolti dal sistema di APM per identificare e risolvere problemi, ottimizzare le prestazioni e migliorare l'esperienza utente complessiva.

4.2 Approcci e tecnologie per il monitoraggio

Nel mondo dell'*Application Performance Monitoring* esistono diversi approcci e tecnologie per raccogliere, analizzare e visualizzare i dati di telemetria.
Il monitoraggio delle prestazioni può essere realizzato mediante diverse strategie, che si distinguono per tipo di dati raccolti, modalità di raccolta e grado di integrazione con l'applicazione.

4.2.1 Approcci principali

Gli approcci al monitoraggio delle applicazioni possono essere classificati in base alla modalità di raccolta dei dati e al livello di osservabilità fornito.
Tra i principali si distinguono:

- **Monitoraggio basato sugli agenti (Agent-based monitoring)**[1]**:** prevede l'integrazione di componenti *software*, detti *agent*, all'interno dell'applicazione o dell'infrastruttura. Questi raccolgono metriche, *log* e tracce in tempo reale, offrendo una visione complessiva del sistema. È il modello adottato da strumenti come *Elastic APM*;

- **Monitoraggio senza agenti (Agentless monitoring)**[2]**:** in questo caso la raccolta dei dati avviene tramite l'analisi di *log* o metriche esposte da servizi esterni, senza modificare il codice dell'applicazione. Sebbene riduca l'invasività, questo approccio offre un livello di dettaglio inferiore rispetto ai sistemi *agent-based*;

- **Distributed tracing**[3]**:** metodo che consente di tracciare l'intero ciclo di vita di una richiesta distribuita tra più servizi o microservizi, associando a ciascun evento un identificativo univoco;

- **Synthetic monitoring**[4]**:** utilizza richieste simulate e *test* automatizzati per verificare la disponibilità e le prestazioni delle applicazioni da diverse località geografiche. È utile per individuare problemi prima che impattino sugli utenti reali dell'applicazione;

- **Real User Monitoring (RUM)**[5]**:** misura le prestazioni dal punto di vista dell'utente reale, analizzando tempi di caricamento, interazioni e metriche di esperienza. Combinato con il monitoraggio sintetico, fornisce una visione completa della *user experience* indicata con il termine End User Monitoring (EUM)[g].

Nel corso del monitoraggio di *PetClinic*, l'approccio adottato integra principalmente l'*Agent-based monitoring*, il *distributed tracing*, il *Synthetic Monitoring* e il *Real User Monitoring*, al fine di ottenere una visione dettagliata delle prestazioni dell'applicazione.

4.2.2 Tecnologie e piattaforme note

Negli ultimi anni si sono affermate diverse piattaforme e *framework* dedicati al monitoraggio e all'osservabilità, che adottano architetture e modelli di raccolta dati differenti.
Tra le più rilevanti si trovano:

- **Elastic Stack:** una delle soluzioni *open source* più diffuse, integra *Elasticsearch*, *Logstash* e *Kibana*, estesa con *Elastic APM* per il tracciamento delle prestazioni. Offre un ecosistema unificato per metriche, *log* e tracce;

[1] *Agent-Based Monitoring.* URL: `https://www.motadata.com/it-glossary/agent-based-monitoring/`.
[2] *Agentless Monitoring.* URL: `https://www.solarwinds.com/resources/it-glossary/agentless-monitoring`.
[3] *Distributed Tracing.* URL: `https://aws.amazon.com/what-is/distributed-tracing/`.
[4] *Synthetic Monitoring.* URL: `https://en.wikipedia.org/wiki/Synthetic_monitoring`.
[5] *Real User Monitoring.* URL: `https://en.wikipedia.org/wiki/Real_user_monitoring`.

- **OpenTelemetry:** standard *open source* promosso dalla *Cloud Native Computing Foundation (CNCF)* per la raccolta e l'esportazione di dati di telemetria. Fornisce SDK[g]e agenti per numerosi linguaggi di programmazione, garantendo interoperabilità tra sistemi di osservabilità differenti;

- **Prometheus e Grafana:** soluzione *open source* focalizzata sulle metriche. *Prometheus* raccoglie e memorizza dati temporali, mentre *Grafana* li visualizza in *dashboard* personalizzabili. È molto usata in ambienti *cloud-native* e *Kubernetes*;

- **Datadog, New Relic, Dynatrace:** piattaforme commerciali che offrono funzionalità avanzate di APM, monitoraggio dell'infrastruttura e analisi basata su *machine learning*.

Le soluzioni *open source*, come *Elastic Stack* e *OpenTelemetry*, offrono maggiore flessibilità e possibilità di personalizzazione, rendendole particolarmente adatte per ambienti di sviluppo e sperimentazione.
Nel contesto del progetto di monitoraggio *PetClinic*, l'attenzione si è concentrata sull'integrazione tra l'*Elastic Stack* e *OpenTelemetry*, con l'obiettivo di realizzare una soluzione di monitoraggio estendibile e compatibile con l'infrastruttura aziendale.

4.3 Tecnologie e strumenti utilizzati

Di seguito viene data una panoramica delle tecnologie e degli strumenti utilizzati.

4.3.1 Elastic Stack

Elasticsearch

Elasticsearch[6] è un motore di ricerca e analisi distribuito basato su *Apache Lucene*, progettato per gestire grandi volumi di dati in tempo reale. Fornisce funzionalità avanzate di ricerca *full-text*, analisi dei dati e aggregazioni, rendendolo ideale per applicazioni di monitoraggio, analisi dei *log* e *business intelligence*.
Elasticsearch supporta un'architettura scalabile e flessibile, consentendo di distribuire i dati su più nodi e *cluster* per garantire alta disponibilità e prestazioni elevate.

Figura 4.1: Logo Elasticsearch

[6] *Elasticsearch*. URL: `https://www.elastic.co/elasticsearch/`.

Kibana

Kibana[7] è uno strumento di visualizzazione e analisi dei dati *open source*, progettato per lavorare in stretta integrazione con *Elasticsearch*. Fornisce un'interfaccia utente intuitiva che consente agli utenti di creare *dashboard* interattive, visualizzare grafici, mappe e tabelle, e analizzare i dati in tempo reale.
Kibana supporta una vasta gamma di visualizzazioni personalizzabili e offre funzionalità avanzate come il filtraggio dei dati, la ricerca *full-text* e l'esplorazione delle relazioni tra i dati, rendendolo uno strumento potente per l'analisi dei *log*, il monitoraggio delle prestazioni e la *business intelligence*.

Figura 4.2: Logo Kibana

Elastic Agent e Fleet

Elastic Agent[8] è un agente unificato sviluppato da *Elastic* che consente di raccogliere, monitorare e proteggere i dati da diverse fonti in modo semplice ed efficace.
Integrando funzionalità di raccolta dati, sicurezza e monitoraggio, *Elastic Agent* semplifica la gestione degli agenti e riduce la complessità operativa.
Fleet è una funzionalità di gestione centralizzata all'interno di *Kibana* che consente di distribuire, configurare e monitorare gli *Elastic Agent* in modo scalabile e automatizzato. Attraverso *Fleet*, gli utenti possono gestire facilmente le *policy* di raccolta dati, aggiornare gli agenti e monitorare lo stato della loro infrastruttura da un'unica interfaccia.

[7] *Kibana*. URL: `https://www.elastic.co/kibana/`.
[8] *Elastic Agent*. URL: `https://www.elastic.co/elastic-agent`.

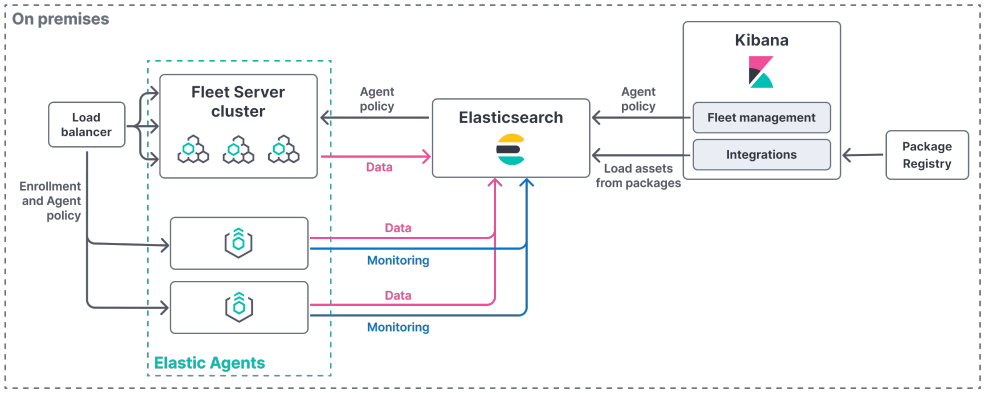


Figura 4.3: Funzionamento Elastic Agents e Fleet

La figura 4.3 mostra l'architettura generale di gestione degli agenti tramite *Fleet*. Gli *Elastic Agents* installati sull'applicazione comunicano con il *Fleet Server* per ricevere le *policy* di configurazione e inviare i dati raccolti a *Elasticsearch* per la visualizzazione in *Kibana*.

OpenTelemetry

OpenTelemetry[9] è un progetto *open source* che fornisce un insieme di API[g], librerie e strumenti per la raccolta di dati di telemetria da applicazioni e servizi.
L'obiettivo di *OpenTelemetry* è standardizzare la raccolta e l'esportazione di dati di telemetria mediante l'OTLP, facilitando l'integrazione con diversi sistemi di monitoraggio e analisi, tra cui l'*Elastic Stack*.
OpenTelemetry supporta vari linguaggi di programmazione e offre un'architettura che consente agli sviluppatori di personalizzare la raccolta dei dati in base alle esigenze specifiche delle loro applicazioni.

Figura 4.4: Logo OpenTelemetry

Elastic APM e APM Server

Elastic APM[10] è una soluzione di monitoraggio delle prestazioni applicative sviluppata da *Elastic*, progettata per tracciare e analizzare le prestazioni delle applicazioni in tempo reale.

[9] *OpenTelemetry*. URL: `https://opentelemetry.io/`.
[10] *Elastic APM*. URL: `https://www.elastic.co/apm/`.

Elastic APM consente di identificare colli di bottiglia, errori e problemi di latenza, fornendo una visione dettagliata del comportamento dell'applicazione attraverso tracce distribuite, metriche e *log*.
L'*APM Server* è un componente dell'*Elastic stack* che funge da punto di raccolta per i dati di telemetria inviati dagli agenti integrati nelle applicazioni. L'*APM Server* elabora questi dati e li invia a *Elasticsearch* per l'indicizzazione, consentendo agli utenti di visualizzare le informazioni tramite *Kibana*.

Figura 4.5: Logo Elastic APM

RUM Agent e Synthetic Monitoring

Il *RUM Agent (Real User Monitoring Agent)*[11], nel contesto del progetto di monitoraggio di *PetClinic* è una libreria *JavaScript* che consente di monitorare le prestazioni dell'applicazione *web* dal punto di vista dell'utente finale.
Integrando il *RUM Agent* nelle pagine *web*, è possibile raccogliere dati sulle interazioni degli utenti, i tempi di caricamento delle pagine e altri eventi significativi. Questi dati vengono inviati all'*APM Server* per l'analisi e la visualizzazione tramite *Kibana*.
Il *Synthetic Monitoring*[12], invece, utilizza *script* automatizzati per simulare le interazioni degli utenti con l'applicazione *web*, consentendo di testare la disponibilità e le prestazioni da diverse località geografiche.
Nel progetto, tale monitoraggio è stato implementato tramite *Kibana Synthetics*, che utilizza *Playwright* come motore di esecuzione di *script* basati su *browser*. Sono stati quindi sviluppati *script Playwright* dedicati, eseguiti da *Kibana* per riprodurre scenari di utilizzo reali e verificare la corretta operatività dell'applicazione.

Figura 4.6: Logo Playwright

[11] *RUM Agent JS*. URL: `https://www.elastic.co/docs/reference/apm/agents/rum-js`.
[12] *Synthetic Monitoring*. URL: `https://www.elastic.co/what-is/synthetic-monitoring`.

4.3.2 Linguaggi di programmazione

Python

Python[13] è un linguaggio di programmazione ad alto livello, ampiamente utilizzato in vari ambiti, tra cui lo sviluppo *web*, l'analisi dei dati, l'intelligenza artificiale e l'automazione.
Python è dotato di una vasta libreria standard e di un ecosistema ricco di pacchetti e *framework* che ne estendono le funzionalità e lo rendono un linguaggio versatile e potente.
In questo progetto *Python* è stato utilizzato principalmente per la creazione di *script* di automazione tramite *Selenium*, al fine di eseguire test automatizzati sull'applicazione *web PetClinic*.

Figura 4.7: Logo Python

JavaScript

JavaScript[14] è un linguaggio di programmazione interpretato, utilizzato per lo sviluppo di applicazioni *web* lato *client* che consente di creare interfacce utente interattive e dinamiche.
Nel contesto del progetto di monitoraggio di *PetClinic*, *JavaScript* è stato utilizzato per integrare il *RUM Agent*, permettendo la raccolta di dati sulle prestazioni dell'applicazione dal punto di vista dell'utente finale.

Figura 4.8: Logo JavaScript

[13] *Python Programming Language*. URL: `https://www.python.org/`.
[14] *JavaScript Programming Language*. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`.

Java

Java[15] è un linguaggio di programmazione ad alto livello, orientato agli oggetti, ampiamente utilizzato nel mondo dello sviluppo *software*.
OpenTelemetry fornisce un agente *APM* specifico per *Java*, che consente di monitorare le prestazioni di *backend* dell'applicazione *PetClinic* in modo dettagliato integrando l'agente nel codice dell'applicazione.

Figura 4.9: Logo Java

4.3.3 Framework e librerie

Node.js

Node.js[16] è utilizzato in *PetClinic* lato *server* per eseguire codice *JavaScript*. Funge da intermediario per le richieste tra il *client* e il *server*, gestendo operazioni asincrone e migliorando le prestazioni complessive dell'applicazione *web*.

Figura 4.10: Logo Node.js

Spring Boot

Spring Boot[17] è il *framework* principale utilizzato per sviluppare l'applicazione *PetClinic*. Fornisce un ambiente di sviluppo semplificato per la creazione di applicazioni *Java* basate su *Spring*, offrendo funzionalità integrate per la gestione delle dipendenze, la configurazione automatica e il supporto per vari moduli come *Spring MVC*, *Spring Data* e *Spring Security*.

[15] *Java Programming Language*. URL: `https://www.java.com/en/`.
[16] *Node.js*. URL: `https://nodejs.org/`.
[17] *Spring Framework*. URL: `https://spring.io/`.

Figura 4.11: Logo Spring

Selenium

Selenium[18] è un *framework open source* utilizzato per l'automazione dei *test* delle applicazioni *web*. Consente di simulare le interazioni degli utenti con il *browser*, eseguendo *test* funzionali e di regressione in modo automatizzato.
Selenium supporta diversi linguaggi di programmazione, tra cui *Java*, *Python* e *JavaScript* e può essere integrato con vari strumenti di *testing* e *framework* di sviluppo.
Durante il progetto è stata utilizzata la libreria *Selenium* per *Python* per creare *script* di *test* automatizzati che simulano le azioni degli utenti sull'applicazione *PetClinic*.

Figura 4.12: Logo Selenium

4.3.4 Strumenti di sviluppo

VSCode

Visual Studio Code (VSCode)[19] è un *editor* di codice sorgente sviluppato da *Microsoft*. Supporta una vasta gamma di linguaggi di programmazione e offre diverse funzionalità avanzate come il *debugging* integrato, il controllo della versione tramite *Git* e un *marketplace* ricco di estensioni.

[18] *Selenium*. URL: `https://www.selenium.dev/`.
[19] *Visual Studio Code*. URL: `https://code.visualstudio.com/`.

Figura 4.13: Logo VSCode

4.3.5 Database

MySQL

MySQL[20] è un sistema di gestione di *database* relazionali *open source*, ampiamente utilizzato per la memorizzazione e la gestione dei dati in applicazioni *web* e aziendali. *MySQL* supporta il linguaggio SQL[g]per l'interrogazione e la manipolazione dei dati, offrendo funzionalità avanzate come transazioni, indicizzazione e replicazione.
Nel contesto dell'applicazione *PetClinic*, *MySQL* viene utilizzato come *database* per archiviare le informazioni relative agli utenti, agli animali domestici, alle visite veterinarie e ad altri dati pertinenti all'applicazione.

Figura 4.14: Logo MySQL

4.3.6 Containerizzazione

Docker

Docker[21] è una piattaforma di containerizzazione che consente di automatizzare il *deployment* di applicazioni all'interno di *container* leggeri e portabili. I *container* isolano le applicazioni e le loro dipendenze, garantendo coerenza tra gli ambienti di sviluppo, *test* e produzione.

[20] *MySQL*. URL: `https://www.mysql.com/`.
[21] *Docker*. URL: `https://www.docker.com/`.

Nel contesto dell'applicazione *PetClinic*, *Docker* è stato utilizzato per creare un ambiente di sviluppo replicabile e per semplificare il *deployment* dell'applicazione e dei suoi componenti.

Figura 4.15: Logo Docker

Docker Compose

Docker Compose[22] è uno strumento che consente di definire e gestire applicazioni *multi-container* tramite un file di configurazione `docker-compose.yml`.
Attraverso *Compose* è possibile avviare, fermare e orchestrare più servizi correlati come un'unica applicazione, semplificando la gestione degli ambienti di sviluppo e test.
Nel contesto dell'applicazione *PetClinic*, *Docker Compose* è stato utilizzato per avviare simultaneamente i componenti dell'applicazione, come il servizio principale, il *database* e gli strumenti di monitoraggio.

Figura 4.16: Logo Docker Compose

4.3.7 Tecnologie analizzate

Model Context Protocol (MCP)

Durante lo *stage* è stato analizzato il *Model Context Protocol*[23] (MCP), uno standard progettato per integrare modelli di intelligenza artificiale con strumenti esterni.
Il protocollo permette di esporre funzionalità tramite un *server MCP* che può essere utilizzato da applicazioni AI (come agenti conversazionali o strumenti di automazione).
Sebbene le funzionalità di AI dello *stack Elastic* come l'*AI Assistant* o gli *Elastic AI*

[22] *Docker Compose*. URL: `https://docs.docker.com/compose/`.
[23] *Model Context Protocol*. URL: `https://modelcontextprotocol.io/docs/getting-started/intro`.

Agents siano pienamente disponibili solo a partire dalla versione 9.2, è stata comunque analizzata la configurazione del *server MCP* con la versione 8.15.3, preparando una procedura per il suo avvio e integrazione preliminare.

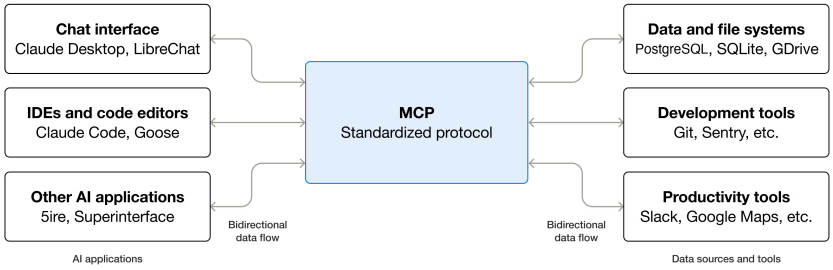


Figura 4.17: Protocollo MCP

4.4 Criteri di scelta della soluzione

Le principali motivazioni che hanno guidato la scelta della soluzione di monitoraggio basata su *Elastic Stack* e *OpenTelemetry* sono:

- **Open source e flessibilità:** entrambe le tecnologie sono *open source*, consentendo una maggiore personalizzazione e adattabilità alle esigenze specifiche del progetto;
- **Scalabilità:** la soluzione è progettata per gestire grandi volumi di dati in ambienti distribuiti, garantendo prestazioni elevate anche con l'aumento del carico di lavoro;
- **Ecosistema completo:** *Elastic Stack* offre un ecosistema completo per la raccolta, l'analisi e la visualizzazione dei dati, semplificando la gestione del sistema di monitoraggio;
- **Requisiti aziendali:** la scelta è stata influenzata dalla necessità di integrare il sistema di monitoraggio con l'infrastruttura esistente dell'azienda, che già utilizza componenti dell'*Elastic Stack* e *Docker*;
- **Comunità attiva:** entrambe le tecnologie vantano una comunità di sviluppatori attiva e in crescita, che contribuisce al miglioramento continuo delle piattaforme e fornisce supporto agli utenti.

4.5 Integrazione con l'ambiente esistente

L'integrazione della soluzione di monitoraggio è stata progettata tenendo conto delle specifiche dell'ambiente tecnico aziendale, basato su infrastrutture *Linux* e containerizzazione tramite *Docker*.
Tutti i componenti dell'*Elastic Stack* sono stati distribuiti in un ambiente isolato, gestito tramite *Docker*, mantenendo la compatibilità con le versioni approvate dall'azienda.

La comunicazione tra l'applicazione *web PetClinic* e il sistema di osservabilità avviene tramite l'agente *OpenTelemetry Java*, che esporta i dati di telemetria verso l'*Elastic APM Server* gestito da *Fleet* dopo essere passati per un *Collector OpenTelemetry*. L'approccio adottato permette un'integrazione trasparente con l'ambiente esistente, garantendo la scalabilità del sistema e la possibilità di estendere la soluzione ad altri servizi o applicazioni monitorate nel futuro.

Capitolo 5

Sviluppo della piattaforma

In questo capitolo vengono approfondite le fasi di sviluppo del sistema, descrivendo l'architettura complessiva, le fasi di implementazione e i risultati ottenuti

5.1 Architettura complessiva del sistema

L'architettura realizzata durante il tirocinio ha l'obiettivo di fornire un sistema completo di osservabilità per la *web app PetClinic*, integrando in un ambiente unico la raccolta dei *log*, metriche, tracce e monitoraggio sintetico.
Per raggiungere questo obiettivo è stato utilizzato l'ecosistema *Elastic Stack*, *Docker* e *OpenTelemetry*.
L'applicazione *PetClinic* è strumentata tramite l'*OpenTelemetry Java Agent*, che esporta metriche e tracce verso un *OpenTelemetry Collector*. Quest'ultimo funge da punto di aggregazione e inoltra i dati al sistema APM gestito da *Elastic Agent* tramite *Fleet*.
In parallelo, i *log* dell'applicazione vengono scritti su un file locale dal *container PetClinic* e successivamente raccolti da *Logstash*, che li elabora e li inoltra verso *Elasticsearch* seguendo una *pipeline* personalizzata.
Per quanto riguarda la visualizzazione dei dati, è stato utilizzato *Kibana*, tramite cui è possibile monitorare l'andamento dell'applicazione, creare *dashboard*, analizzare metriche di *performance*, effettuare ricerche sui *log* ed eseguire attività di *anomaly detection*.
Accanto ai dati reali provenienti dall'applicazione, è stato integrato un sistema di *Synthetic Monitoring* basato su *Kibana Synthetics*, che utilizza *script Playwright* per simulare il comportamento degli utenti e verificare la disponibilità e il corretto funzionamento dei principali flussi di navigazione. Trattandosi di un applicazione di prova infatti l'utilizzo di monitoraggio sintetico permette di lavorare con una quantità più significativa di dati.
Nel complesso, l'architettura si presenta come una *pipeline* altamente modulare, in cui ogni componente è orchestrato tramite *Docker Compose*. Questa struttura rende l'ambiente facilmente replicabile, facilitando sia le attività di sviluppo che quelle di *troubleshooting*.
Nella figura si riporta l'architettura complessiva del sistema e il flusso dei dati tra le varie componenti:

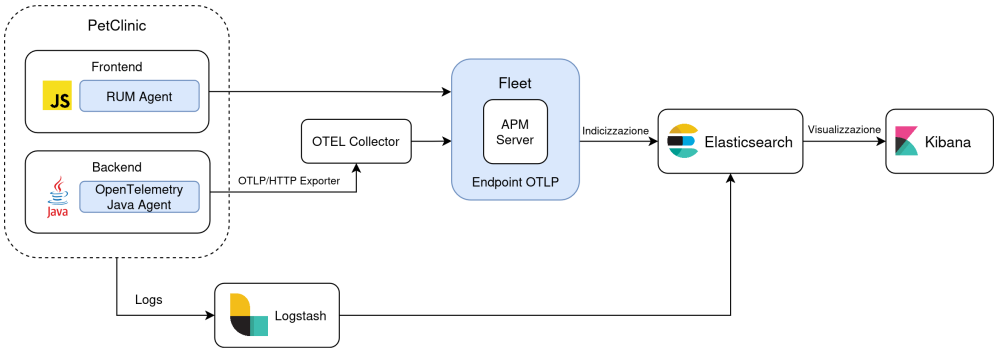


Figura 5.1: Architettura complessiva

5.2 Struttura del progetto

La realizzazione del sistema di osservabilità è stata accompagnata dalla definizione di una struttura del progetto che ne facilitasse lo sviluppo e le attività di manutenzione. Il *repository* principale, chiamato *elastic-project*, contiene tutte le componenti necessarie all'avvio tramite *Docker Compose* e alla configurazione delle *pipelines* di *log*, metriche e tracce, nonché all'avvio delle istanze del *Fleet Server* e delle *policy* per il *Synthetic monitoring*.
La struttura complessiva è riportata di seguito:

```
elastic-project/
|-- docker-compose.yml
|-- .env
|-- logstash.conf
|-- logs/
|-- collector/
|   \-- config.yaml
|-- mcp-server-elasticsearch/
|   \-- Dockerfile
\-- spring-petclinic/
    \-- Dockerfile
    \-- opentelemetry-javaagent.jar
```

Questa organizzazione è stata costruita con l'obiettivo di isolare le responsabilità dei vari componenti:

- **docker-compose.yml** rappresenta il *file* principale per l'orchestrazione dei *container*, in cui vengono definiti i servizi fondamentali. La scelta di accorpare tutti i servizi in un unico file semplifica la fase di avvio e garantisce un ambiente riproducibile;

- **spring-petclinic**/ contiene il codice dell'applicazione *PetClinic* e il relativo *Dockerfile* per la creazione dell'immagine personalizzata con l'*OpenTelemetry Java Agent* integrato;

- **collector/** include la configurazione dell'*OpenTelemetry Collector* nel *file config.yaml*, responsabile della ricezione dei dati OTLP provenienti da *PetClinic* e del loro inoltro al sistema APM gestito da *Fleet*;
- **logstash.conf** definisce la *pipeline* di *Logstash* che legge e filtra i *log* dell'applicazione, applicando un primo livello di arricchimento e inviando i dati a *Elasticsearch*;
- **logs/** funge da volume locale in cui l'applicazione *PetClinic* rende disponibili i *file* di *log*;
- **mcp-server-elasticsearch/** contiene i *file* necessari per l'esecuzione del *server* MCP[g]dedicato a *Elasticsearch*, che abilita l'interazione con strumenti di AI generativa come *Claude Code*;
- **.env** contiene le variabili d'ambiente necessarie per la configurazione dei servizi.

Questa struttura ha permesso di lavorare in modo indipendente sulle singole componenti del sistema senza introdurre interferenze tra servizi, e ha contribuito a semplificare la fase di diagnosi dei problemi.

5.3 Implementazione del logging

L'implementazione della *pipeline* di raccolta e indicizzazione dei *log* applicativi rappresenta un passaggio fondamentale del sistema di osservabilità sviluppato.
L'obiettivo principale era quello di acquisire i *log* generati da *Spring PetClinic*, arricchirli con metadati utili, e renderli disponibili per la consultazione e l'analisi tramite *Kibana*.
Per ottenere questo risultato è stata realizzata una *pipeline* composta da tre elementi principali:

1. Scrittura dei *log* nel *container PetClinic*, in un percorso montato come volume esterno;
2. Raccolta ed elaborazione dei *log* tramite *Logstash*, configurato con un *file* di *pipeline* dedicato;
3. Indicizzazione dei *log* in *Elasticsearch* tramite una struttura di indici gestita con ILM[g]e un *template* personalizzato.

5.3.1 Scrittura dei log applicativi

Il primo passo ha riguardato la configurazione di *PetClinic* affinché producesse *log* su *file*, in modo da poterli acquisire tramite *Logstash*. Il *Dockerfile* dell'applicazione è stato modificato aggiungendo il parametro `-Dlogging.file.name=/var/log/petclinic/app.log` alla riga di comando di avvio.
Questo approccio ha permesso di centralizzare i *log* in un unico *file*, esporlo come volume condiviso e separare la fase di generazione dei *log* dalla loro elaborazione.

5.3.2 Pipeline Logstash

La raccolta e trasformazione dei *log* è stata implementata tramite *Logstash*, attraverso un *file* di configurazione dedicato `logstash.conf`:

```
input {
  file {
    path => "/var/log/petclinic/app.log"
    start_position => "beginning"
    sincedb_path   => "/usr/share/logstash/data/sincedb-petclinic
        "
  }
}

filter {
  mutate {
    add_field => {
      "service.name"      => "petclinic"
      "event.dataset"     => "petclinic.app"
      "data_stream.type"  => "logs"
      "data_stream.dataset" => "petclinic"
      "data_stream.namespace" => "default"
    }
  }
}

output {
  stdout { codec => rubydebug }

  elasticsearch {
    hosts    => [ "${ELASTIC_HOSTS}" ]
    user     => "${ELASTIC_USER}"
    password => "${ELASTIC_PASSWORD}"
    ssl      => true
    cacert   => "/usr/share/logstash/certs/ca/ca.crt"
    index    => "petclinic-logs"
  }
}
```

La *pipeline* si articola in tre fasi: input, filtro e output.

Input: Viene utilizzato il plugin `file` per monitorare il *file* `app.log` generato dall'applicazione. Il *file* viene letto dall'inizio e l'avanzamento è tracciato tramite un *file* `sincedb`, così da evitare duplicazioni in caso di riavvio.

Filter: I *log* vengono arricchiti con metadati aggiuntivi, tra cui:

- `service.name`;
- `event.dataset`;
- campi relativi ai data stream di *Elasticsearch*.

Queste informazioni migliorano la struttura dei documenti, permettendo un'analisi più efficace e una migliore organizzazione dei dati nei *data view*.

Output: L'output della pipeline invia i documenti verso *Elasticsearch* tramite l'*endpoint* HTTPS[g], utilizzando credenziali e certificati definiti nel *file* `.env`.
I *log* vengono scritti sull'alias `petclinic-logs`, gestito successivamente tramite ILM.

5.3.3 Struttura degli indici in Elasticsearch

Prima dell'invio dei *log* è stata predisposta l'infrastruttura necessaria in *Elasticsearch*.

Policy ILM: È stata definita una *policy* di gestione del ciclo di vita dell'indice (*Index Lifecycle Management*) in grado di:

- effettuare *rollover* ogni 1 GB[g]o ogni 24 ore;
- eliminare automaticamente i dati più vecchi dopo 7 giorni.

strutturata come segue:

```
PUT _ilm/policy/petclinic-logs-ilm
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_size": "1gb",
            "max_age": "1d"
          }
        }
      },
      "delete": {
        "min_age": "7d",
        "actions": {
          "delete": {}
        }
      }
    }
  }
}
```

Index Template: È stato creato un *template* denominato `petclinic-logs-template` per definire:

- il numero di *shard* e repliche;
- il limite massimo dei campi mappati;
- il *mapping* dei principali campi del *log*;
- regole dinamiche per trattare le stringhe come *keyword*.

```
PUT _index_template/petclinic-logs-template
{
  "index_patterns": ["petclinic-logs-*"],
  "template": {
```

```
"settings": {
  "index.lifecycle.name": "petclinic-logs-ilm",
  "index.lifecycle.rollover_alias": "petclinic-logs",
  "number_of_shards": 1,
  "number_of_replicas": 0,
  "index.mapping.total_fields.limit": 2000,
},
"mappings": {
  "dynamic": true,
  "properties": {
    "@timestamp": {
      "type": "date"
    },

    "message": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword", "ignore_above": 256 }
      }
    },

    "log.level":       { "type": "keyword" },
    "log.logger":      { "type": "keyword" },
    "log.thread":      { "type": "keyword" },
    "log.original":    { "type": "text" },

    "service.name":    { "type": "keyword" },
    "event.dataset":   { "type": "keyword" },

    "trace.id": {
      "type": "keyword",
      "ignore_above": 256
    },
    "span.id": {
      "type": "keyword",
      "ignore_above": 256
    },

    "error": {
      "properties": {
        "type":     { "type": "keyword" },
        "message":  { "type": "text" },
        "stack":    { "type": "text" }
      }
    }
  },

  "dynamic_templates": [
    {
      "strings_as_keywords": {
        "match_mapping_type": "string",
        "mapping": { "type": "keyword", "ignore_above": 256 }
      }
    }
```

```
      ]
    }
  },
  "_meta": {
    "description": "Template index per petclinic"
  }
}
```

Alias e primo indice: È stato infine creato il primo indice gestito dalla *policy* ILM, associato all'*alias* `petclinic-logs` con `is_write_index = true`, in questo modo *Logstash* scrive sempre nell'*alias*, mentre *Elasticsearch* gestisce la creazione dei nuovi indici tramite ILM:

```
PUT petclinic-logs-000001
{
  "aliases": {
    "petclinic-logs": {
      "is_write_index": true
    }
  }
}
```

5.3.4 Risultato della pipeline

La *pipeline* finale dei *log* può essere sintetizzata nel seguente flusso:

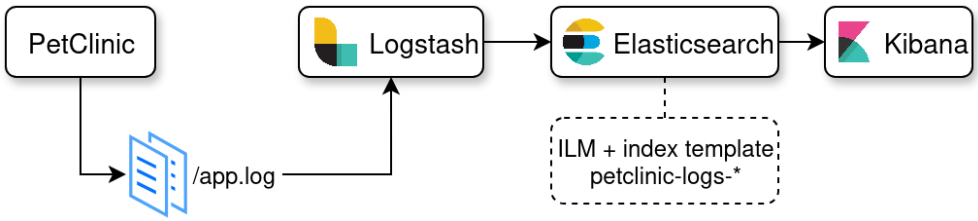


Figura 5.2: Pipeline Logs

Grazie a questa configurazione viene garantita un'acquisizione affidabile dei *log*, un arricchimento consistente delle informazioni e una consultazione efficace tramite *Kibana*.

5.4 Implementazione di traces e metrics

Parallelamente alla *pipeline* dei *log*, è stata realizzata una *pipeline* dedicata alla raccolta di metriche e tracce distribuite generate dalla *web app PetClinic*.
L'obiettivo è stato quello di ottenere una visione approfondita sui tempi di risposta, sui flussi di esecuzione delle richieste e sulle interazioni tra i componenti del sistema.
Per raggiungere questo risultato è stato adottato *OpenTelemetry*, integrato con l'*Elastic APM* tramite *Fleet*.
La *pipeline* completa è composta dai seguenti elementi:

1. Strumentazione automatica dell'applicazione tramite *OpenTelemetry Java Agent*;
2. Raccolta dei dati OTLP (*OpenTelemetry Protocol*) da parte dell'*OpenTelemetry Collector*;
3. Inoltro dei dati verso l'*APM Server* gestito dall'*Elastic Agent*;
4. Indicizzazione delle metriche e delle tracce in *Elasticsearch* e visualizzazione in *Kibana*.

5.4.1 Strumentazione dell'applicazione

L'applicazione *PetClinic* è stata strumentata utilizzando l'*OpenTelemetry Java Agent*, un agente *Java* esterno che permette di raccogliere automaticamente metriche e tracce senza modifiche al codice sorgente.
Nel `Dockerfile` e nella cartella di *Petclinic* rispettivamente sono stati aggiunti:

- i parametri di avvio per abilitarne il caricamento e configurarne l'esportazione;
- il file `opentelemetry-javaagent.jar`.

Il comando di avvio che viene eseguito dal `Dockerfile` è il seguente:

```
CMD ["java", \
    "-javaagent:/app/opentelemetry-javaagent.jar", \
    "-Dotel.service.name=petclinic", \
    "-Dotel.exporter.otlp.endpoint=http://collector:4318", \
    "-Dotel.exporter.otlp.protocol=http/protobuf", \
    "-Dotel.exporter.otlp.insecure=true", \
    "-Dlogging.file.name=/var/log/petclinic/app.log", \
    "-jar", "/app/app.jar"]
```

Grazie a questa configurazione, l'applicazione esporta automaticamente:

- metriche sulle performance (CPU, *heap*, ecc);
- tracce delle richieste HTTP in ingresso;
- eventi relativi agli errori e alle eccezioni.

5.4.2 RUM Agent frontend

Oltre ai dati provenienti dal *backend* tramite l'*OpenTelemetry Java Agent*, la *pipeline* raccoglie anche metriche, errori e tracce generate dal *frontend* dell'applicazione attraverso il *RUM Agent* di *Elastic APM*.
Il *RUM Agent*, integrato direttamente nelle pagine *web*, invia i dati di *performance* del *browser*, la durata delle transazioni utente, gli errori *JavaScript* e le informazioni sulla navigazione direttamente all'*APM Server*.
Il *RUM Agent*, a differenza dell'*OpenTelemetry Collector*, comunica nativamente con il protocollo *Elastic APM* e invia i dati direttamente all'*endpoint* APM, senza transitare attraverso il *Collector*.
Entrambe le sorgenti, confluiscono nei *data stream* `traces-apm*` e `metrics-apm*`, permettendo una visione unificata delle *performance* sia del *backend* sia del *frontend*.

5.4.3 OpenTelemetry Collector

Il *Collector* funge da punto di raccolta per i dati OTLP provenienti dalla *web app*. In questo progetto è stato configurato in modalità *gateway*, aggregando i dati e inoltrandoli al sistema APM gestito da *Elastic*.
Il file `config.yaml` definisce i seguenti componenti principali:

- un `receiver` OTLP (HTTP e gRPC[g]) per ricevere i dati dall'applicazione;
- un processore `memory_limiter` per evitare sovraccarichi;
- un processore `batch` per inviare i dati in blocchi ottimizzati;
- un `exporter` `otlphttp/elastic` diretto verso il container `apm-agent`.

Il *collector* è eseguito come servizio dedicato nel `docker-compose`, con volume in sola lettura.
Ecco la configurazione completa:

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318
        cors:
          allowed_origins: ["http://localhost:*", "http
              ://127.0.0.1:*", "*"]
          allowed_headers: ["*"]

processors:
  memory_limiter:
    check_interval: 2s
    limit_percentage: 80
    spike_limit_percentage: 25
  batch:
    timeout: 2s
    send_batch_size: 1024


exporters:
  otlphttp/elastic:
    endpoint: "http://apm-agent:8200"
    tls:
      insecure: true

service:
  pipelines:
    traces:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/elastic]
    metrics:
      receivers: [otlp]
      processors: [memory_limiter, batch]
```

```
      exporters: [otlphttp/elastic]
    logs:
      receivers: [otlp]
      processors: [memory_limiter, batch]
      exporters: [otlphttp/elastic]
```

5.4.4 Elastic Agent e APM Server

Il sistema APM di *Elastic* è gestito tramite l'*Elastic Agent* registrato in *Fleet*. Nel progetto è stato utilizzato un *container* dedicato (`apm-agent`) configurato per:

- registrarsi automaticamente presso il *Fleet Server* tramite *enrollement token*;
- esporre l'*endpoint* APM sulla porta 8200;
- ricevere dati OTLP dal *Collector*.

L'*APM Server* è responsabile di:

- validare metriche e tracce in arrivo;
- convertirle nei documenti *Elasticsearch*;
- indicizzarle automaticamente in base ai moduli di ingesione dello *stack Elastic*.

La corretta registrazione del *container* in *Fleet* è un prerequisito per il funzionamento della *pipeline*, a tal fine, è stata dedicata un'apposita *policy APM*.

5.4.5 Indicizzazione e visualizzazione dei dati in Kibana

Una volta ricevuti dal sistema APM, i dati vengono indicizzati in *Elasticsearch* nei *data stream*:

- `traces-apm*` per le tracce distribuite;
- `metrics-apm*` per le metriche;
- `logs-apm*` per eventuali eventi generati dall'agente.

Questi indici vengono raccolti automaticamente nella *data view* `APM`, fornita nativamente dalla piattaforma.

5.4.6 Risultato della pipeline

Il flusso risultante può essere sintetizzato come segue:

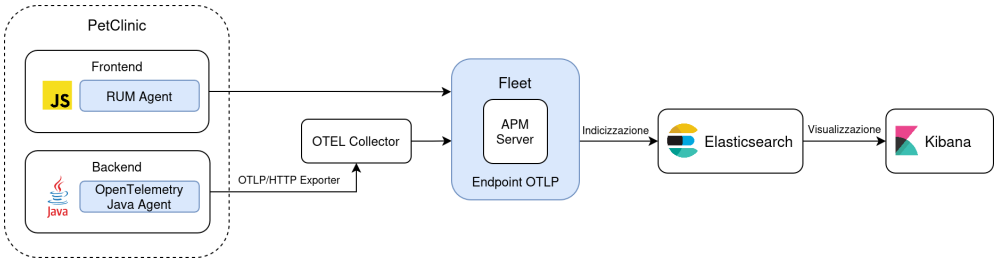


Figura 5.3: Pipeline Traces e Metrics

5.5 Containerizzazione con Docker Compose

L'intera infrastruttura di osservabilità è stata containerizzata utilizzando *Docker Compose*, con lo scopo di ottenere un ambiente completamente riproducibile, isolato e facilmente avviabile.
Il file `docker-compose.yml` rappresenta il punto centrale dell'orchestrazione e descrive un insieme eterogeneo di servizi che lavorano assieme, tra cui:

- `es01`: nodo *Elasticsearch* responsabile della memorizzazione dei *log*, delle metriche e delle tracce;
- `kibana`: interfaccia grafica per la visualizzazione dei dati;
- `fleet-server`: servizio dedicato alla gestione centralizzata degli agenti *Elastic*;
- `apm-agent`: componente che espone l'*endpoint APM* per ricevere metriche e tracce dal *Collector* e dal *RUM Agent*;
- `collector`: *OpenTelemetry Collector* configurato in modalità *gateway* sulla porta 4318;
- `logstash01`: responsabile della *pipeline* di elaborazione dei *log* applicativi;
- `petclinic`: applicazione oggetto del monitoraggio;
- `mysql`: *database* a supporto dell'applicazione;
- `synthetics-agent`: componente dedicato al monitoraggio sintetico;
- `mcp-server-elasticsearch`: server MCP utilizzato per l'integrazione con strumenti di AI generativa;
- `setup`: servizio dedicato alla generazione e distribuzione dei certificati TLS[g].

5.5.1 Gestione dei certificati e della sicurezza

Una parte rilevante dell'orchestrazione riguarda la gestione della sicurezza.
Il servizio `setup` genera automaticamente certificati autofirmati e li distribuisce agli altri *container* attraverso un volume condiviso. Questo passaggio è essenziale poiché i servizi dello *stack Elastic* richiedono comunicazioni cifrate e autenticazione abilitata.
Il file `.env`, incluso nel progetto, contiene inoltre credenziali, porte e parametri di configurazione necessari all'avvio dei *container*.

5.5.2 Coordinamento dell'avvio dei servizi

La corretta gestione dell'ordine di avvio è stata un aspetto importante della configurazione. Alcuni servizi, come `fleet-server` e `apm-agent`, richiedono che *Elasticsearch* sia completamente operativo prima di potersi registrare ed esporre i propri *endpoint*. Questo comportamento è stato gestito tramite direttive `depends_on`, *healthcheck* e l'utilizzo di servizi come `setup`, che garantiscono la generazione dei certificati necessari al *cluster*.
L'uso esplicito delle dipendenze e dei controlli di stato ha reso possibile evitare errori di avvio legati alla sincronizzazione tra i servizi, un problema frequente nelle architetture *multi-container*.

5.5.3 Risultato dell'orchestrazione

L'orchestrazione basata su *Docker Compose* consente di controllare e avviare l'intero ambiente di osservabilità tramite il comando:

```
docker compose up -d
```

Questo approccio ha permesso di ottenere un sistema facilmente estendibile e semplice da mantenere.

5.6 Visualizzazione dei dati in Kibana

Al termine della configurazione delle *pipeline* di *log*, metriche, tracce e monitoraggio, è stato necessario predisporre in *Kibana* gli strumenti per la visualizzazione e l'analisi dei dati raccolti.
Sono stati individuati diversi *data view*, alcuni già presenti di *default* in *Kibana*, altri appositamente creati per facilitare l'accesso ai vari tipi di dati:

- `traces-apm*`, fornita nativamente da *Kibana*, contenente le tracce distribuite provenienti dal *backend* e dal *frontend*;
- `metrics-apm*`, fornita nativamente da *Kibana*, relativa alle metriche di sistema raccolte dall'*OpenTelemetry Java Agent*;
- `logs-apm*`, fornita nativamente da *Kibana*, include eventuali eventi generati dagli agenti *Elastic*;
- `petclinic-logs-*`, *data view* dedicato ai *log* applicativi raccolti tramite *Logstash* e indicizzati in *Elasticsearch*.

Per facilitare l'analisi, sono state create due viste dati principali:

- **APM**: fornita nativamente da *Kibana*, che aggrega metriche, tracce ed errori provenienti dal *backend*, dal *frontend* e dal monitoraggio sintetico;
- **petclinic-logs**: vista personalizzata per l'esplorazione dei *log* applicativi indicizzati tramite la *pipeline Logstash*.

5.6.1 Verifica dell'indicizzazione tramite Dev Tools

Dopo l'avvio dei servizi, è stata effettuata una verifica manuale dell'indicizzazione tramite la *console Dev Tools* di *Kibana*.
Questo controllo è stato utile per confermare il corretto funzionamento della *pipeline* di *log* e verificare che i documenti fossero instradati verso il *data stream* appropriato.
Alcuni esempi di *query* utilizzate sono:

- `GET petclinic-logs-*/_search?size=5` per verificare che i documenti applicativi arrivino all'indice corretto;
- `GET petclinic-logs-*/_mapping` per verificare che il *mapping* dei campi sia in linea con il template di indicizzazione.

La risposta di Elasticsearch ha confermato che:

- *Logstash* stava correttamente inviando i *log* verso l'alias `petclinic-logs`;
- il *Collector* e il *RUM Agent* inviavano correttamente le tracce all'*APM Server*;
- tutti i documenti venivano mappati in base al *template* configurato.

5.7 Creazione delle dashboard

All'interno della sezione *"Dashboards"* di *Kibana* sono state realizzate due *dashboard* dedicate all'analisi e alla visualizzazione dei dati raccolti dal *data view* `APM`.
Le *dashboard* hanno lo scopo di fornire una panoramica del comportamento dell'applicazione sia dal punto di vista dell'utente finale sia dal punto di vista delle *performance* e della stabilità lato *server*.

5.7.1 Dashboard Frontend - User Journey

La *dashboard* dedicata al *frontend* analizza il comportamento degli utenti e le *performance* percepite dal *browser* tramite i dati raccolti dal *RUM Agent*.
Essa comprende cinque visualizzazioni principali:

- **Funnel di navigazione**: con l'asse X impostato su `transaction.name` e l'asse Y sul `countof(records)`, per rappresentare i percorsi più frequenti;
- **Tempo di caricamento per pagina**: con `url.full` sull'asse X e `avg(transaction.duration.us)` sull'asse Y;
- **Errori JavaScript per browser**: con `countof(error.id)` sull'asse Y e `user_agent.name` sull'asse X;
- **Interazioni per tipo**: basata su `span.name` sull'asse X e `countof(records)` sull'asse Y;
- **Distribuzione dei browser per indirizzo IP**: utilizzata per analizzare la provenienza e l'ambiente *client* delle sessioni.

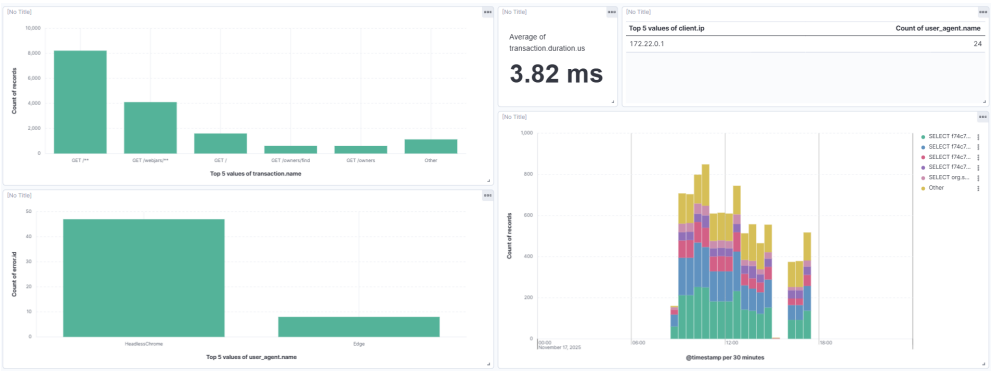


Figura 5.4: Dashboard Frontend - User Journey

5.7.2 Dashboard Backend - Health & Stability

La *dashboard* relativa al *backend* fornisce invece una panoramica dello stato di salute della *web app* e delle sue *performance* lato *server*.
Anche in questo caso sono presenti cinque visualizzazioni:

- **Tempo medio di risposta per endpoint**: con `url.path` in X e `avg(transaction.duration.us)` in Y;
- **Errori per tipo di eccezione**: con `error.exception.type` rappresentato in un grafico a torta;
- **Chiamate lente** (oltre 2 secondi): identificate tramite un filtro su `transaction.duration.us >= 2000000` e rappresentate in formato tabellare;
- **Distribuzione delle richieste per metodo HTTP**: basata su `http.request.method` in un grafico a torta;
- **Conteggio totale delle richieste**: che fornisce una vista aggregata dei volumi di traffico applicativo.

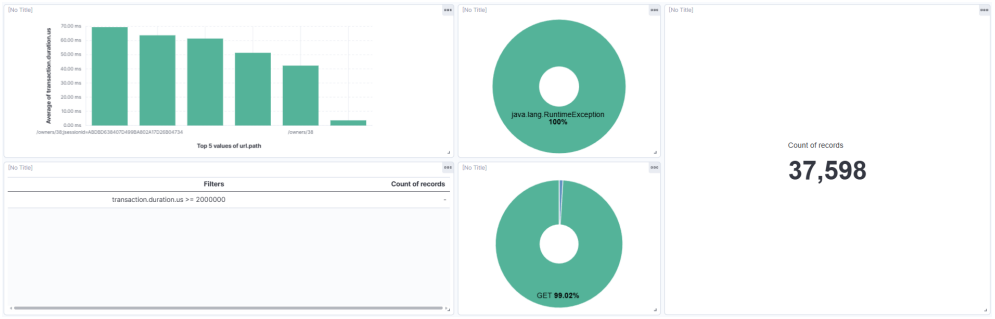


Figura 5.5: Dashboard Backend - Health & Stability

Le due *dashboard* costituiscono un insieme di strumenti utili per comprendere il comportamento dell'applicazione in condizioni reali, individuare potenziali anomalie e monitorare l'esperienza utente e la stabilità del sistema.

5.8 Machine Learning e Anomaly Detection

All'interno dell'infrastruttura è stato integrato il modulo di *Machine Learning* di *Kibana* con lo scopo di identificare automaticamente comportamenti anomali nei dati raccolti.
Questa funzionalità fa parte della versione premium dell'*Elastic stack* messa a disposizione dall'azienda e consente di rafforzare la capacità di monitoraggio della piattaforma, introducendo un livello di analisi predittiva che va oltre le normali visualizzazioni fornite in *Kibana*.

5.8.1 Anomaly Detection Jobs

Sono stati configurati due *job* distinti, ognuno progettato per analizzare un diverso aspetto del comportamento dell'applicazione:

- **anomaly_transactions**: *job* di tipo *multimetric* con partizionamento per `transaction.name`. Analizza la durata mediana delle transazioni con l'obiettivo di rilevare rallentamenti anomali per singolo *endpoint*. Gli *influencer* configurati (`transaction.name` e `service.name`) permettono di identificare rapidamente quale operazione contribuisce maggiormente alla deviazione osservata.
- **error_rate_detection**: *job* di tipo *single metric* orientato alla rilevazione di anomalie nel tasso di errori delle transazioni. Il job analizza l'andamento storico delle occorrenze di errore e segnala incrementi inaspettati rispetto al comportamento atteso.

Entrambi i *job* operano in modalità continua, aggiornando costantemente il modello statistico in funzione dei dati più recenti e assegnando un *anomaly score* agli eventi fuori norma.

5.8.2 Regole di alerting

Per ciascun *job* è stata definita una regola di *alerting* dedicata basata sugli *anomaly records*. La regola si attiva quando lo *anomaly score* supera una soglia configurata, generando una notifica all'interno di *Kibana* e via *email*. Ogni *alert* include informazioni su:

- livello di anomalia;
- metrica o transazione coinvolta;
- valore osservato e valore previsto;
- link diretto alla sezione di dettaglio del *job*.

5.8.3 Benefici per il monitoraggio

L'integrazione del modulo ML[g]fornisce alla piattaforma di osservabilità uno strato di analisi avanzato capace di:

- individuare anomalie graduali o intermittenti difficili da rilevare manualmente;
- evidenziare rallentamenti e incrementi del tasso di errore prima che impattino direttamente l'utente finale;
- fornire una spiegazione del problema tramite gli *influencer* e i grafici di contributo.

5.9 Synthetic Monitoring

Il *Synthetic Monitoring* è stato utilizzato per simulare in modo automatico le interazioni degli utenti con *PetClinic*, per monitorarne la disponibilità, i tempi di risposta e il corretto funzionamento.
La configurazione è stata effettuata all'interno della sezione *Observability –> Synthetics –> Monitors* di *Kibana*, dove sono stati creati sette *monitor* dedicati ai percorsi di navigazione più rilevanti.

5.9.1 Creazione dei monitor

Kibana consente di definire *monitor* sintetici caricando uno *script JavaScript* o scrivendolo direttamente nell'interfaccia.
Per la generazione degli *script* è stato utilizzato *Synthetics Recorder*, strumento nativo che registra le azioni dell'utente nel *browser* ed esporta automaticamente codice *Playwright* pronto per l'esecuzione.
Questa soluzione si è dimostrata più efficiente rispetto ad alternative come *Katalon*, che forniva esportazioni non compatibili (es. *Python2*) oppure *Selenium*.
Ogni *monitor* esegue uno *script Playwright* che riproduce una sequenza di azioni definite: apertura della *homepage*, navigazione nei menu, ricerca di proprietari, inserimento di nuovi dati e visualizzazione delle sezioni dedicate ai veterinari.

5.9.2 Flussi monitorati

Sono stati implementati sette *monitor*, ciascuno focalizzato su un flusso specifico:

- **owners**: navigazione nella sezione *Find Owners* e consultazione dei profili;
- **add_owner**: inserimento di un nuovo proprietario tramite il *form* dell'applicazione;
- **add_owner_pet**: creazione di un proprietario e aggiunta di un nuovo animale;
- **find**: ricerca di un proprietario tramite il campo *lastName*;
- **home**: verifica del caricamento della *homepage*;
- **navigation**: navigazione sequenziale tra *Home, Find Owners, Veterinarians, Error* e ritorno alle sezioni principali;
- **veterinarians**: esplorazione della pagina dei veterinari e delle sue sottosezioni.

Ciascun monitor esegue ad intervalli di tempo regolari il relativo *script*, registrando attività nell'applicazione, che a sua volta invia le metriche e i risultati prodotti a *Elasticsearch* secondo le *pipeline* previste, consentendo di analizzare la disponibilità dell'applicazione, i tempi di risposta e l'eventuale presenza di errori direttamente nella sezione *Synthetics* di *Kibana*.
Un esempio di *script Playwright* utilizzato per il *monitor* **add_owner_pet** è il seguente:

```
step('Go to petclinic and add owner + pet', async () => {
  await page.goto('http://petclinic:8080/');
  await page.getByRole('link', { name: 'Find Owners' }).click();
  await page.getByRole('link', { name: 'Add Owner' }).click();
  await page.getByLabel('First Name').click();
  await page.getByLabel('First Name').fill('Mark');
  await page.getByLabel('Last Name').click();
  await page.getByLabel('Last Name').fill('Cole');
  await page.getByLabel('Address').click();
  await page.getByLabel('Address').fill('Via Roma 1');
  await page.getByLabel('City').click();
  await page.getByLabel('City').fill('Roma');
  await page.getByLabel('Telephone').click();
  await page.getByLabel('Telephone').fill('0000000000');
```

```
  await page.getByRole('button', { name: 'Add Owner' }).click();
  await page.getByRole('link', { name: 'Add New Pet' }).click();
  await page.getByLabel('Name').click();
  await page.getByLabel('Name').fill('Bob');
  await page.getByLabel('Birth Date').fill('2025-10-10');
  await page.getByLabel('Type').selectOption('hamster');
  await page.getByRole('button', { name: 'Add Pet' }).click();
});
```

Lo stesso script in *Python*, presentato di seguito, è stato testato con *Selenium* ma ha richiesto maggiori passaggi per essere eseguito correttamente, dimostrando l'efficienza dell'approccio basato su *Playwright* e *Synthetics Recorder*:

```
import sys
import time
from selenium import webdriver
from selenium.webdriver.firefox.options import Options
from selenium.common.exceptions import WebDriverException
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import Select

def main(show_gui=True):
    opts = Options()
    if not show_gui:
        opts.add_argument("--headless")

    try:
        driver = webdriver.Firefox(options=opts)
    except WebDriverException as e:
        print("Errore avviando webdriver:", e)
        sys.exit(2)

    try:
        url = "http://petclinic:8080"
        print("apro", url)
        driver.get(url)
        time.sleep(1)

        driver.get(url +"/owners/find")
        time.sleep(1)

        add_owner_link = driver.find_element(By.LINK_TEXT, "Add
            Owner")
        add_owner_link.click()
        time.sleep(1)

        driver.find_element(By.ID, "firstName").send_keys("Mark")
        driver.find_element(By.ID, "lastName").send_keys("Cole")
        driver.find_element(By.ID, "address").send_keys("Via Roma
             1")
        driver.find_element(By.ID, "city").send_keys("Roma")
        driver.find_element(By.ID, "telephone").send_keys
            ("0000000000")
```

```
        driver.find_element(By.CSS_SELECTOR, "button[type='submit
            ']").click()
        time.sleep(2)

        add_pet_link = driver.find_element(By.LINK_TEXT, "Add New
             Pet")
        add_pet_link.click()
        time.sleep(1)

        driver.find_element(By.ID, "name").send_keys("Bob")
        driver.find_element(By.ID, "birthDate").send_keys
            ("10/10/2025")
        Select(driver.find_element(By.ID, "type")).select_by_
            visible_text("hamster")

        driver.find_element(By.CSS_SELECTOR, "button[type='submit
            ']").click()
        time.sleep(2)

        print("Proprietario e animale aggiunti con successo.")
    finally:
        try:
            driver.quit()
        except Exception:
            pass
        print("Test completato")

if __name__ == "__main__":
    main(show_gui=True)
```

5.9.3 Benefici per il monitoraggio

L'integrazione del *Synthetic Monitoring* garantisce un ulteriore livello di controllo, complementare a metriche, tracce e *log*. Grazie ai *monitor* sintetici è possibile infatti rilevare:

- rallentamenti o interruzioni nei percorsi chiave dell'applicazione;
- problemi lato UI[g]o di caricamento delle pagine;
- differenze di comportamento tra esecuzioni consecutive.

Questo approccio consente di simulare l'esperienza dell'utente finale e di individuare rapidamente anomalie che potrebbero impattare sulla disponibilità complessiva del servizio, oltre ad aumentare il numero di dati di telemetria disponibili.

5.10 Integrazione con MCP Server e strumenti AI

Durante il periodo di *stage* è stata analizzata la possibilità di integrare la piattaforma di osservabilità con strumenti di intelligenza artificiale generativa basati sul protocollo MCP (Model Context Protocol). Tale integrazione sarebbe utile per abilitare funzionalità avanzate come il riassunto dei *log* e l'interazione con i dati tramite agenti conversazionali.

5.10.1 Limitazioni della versione 8.15 dello stack

In fase di analisi è emerso che le funzionalità AI più recenti come gli *Elastic AI Agents*, l'*AI Assistant* integrato in *Kibana* e le automazioni basate su agenti sono disponibili solo a partire dalla versione **9.2** dello *stack Elastic*.
Tuttavia, questa versione non è stata adottata nel progetto per due motivi principali:

- non tutte le immagini *Docker* necessarie risultano ancora disponibili o stabili per un ambiente completamente containerizzato;
- la clientela dell'azienda presso cui è stato svolto lo *stage* utilizza stabilmente versioni **8.x**, rendendo necessaria la compatibilità con tale ecosistema.

Per questi motivi l'integrazione con gli strumenti AI nativi dello *stack* non è stata implementata, pur essendo stata valutata e studiata.

5.10.2 Avvio e configurazione del MCP server

Nonostante l'impossibilità di utilizzare le funzionalità AI avanzate, è stata preparata una procedura per l'inizializzazione corretta dell'*MCP Server* con *Elastic 8.15.3*, con l'obiettivo di predisporre l'infrastruttura per futuri sviluppi.
La procedura include:

- l'avvio del *container* `mcp-server-elasticsearch` tramite *docker compose*;
- la configurazione delle variabili d'ambiente per autenticazione ed *endpoint*;
- la registrazione del *server MCP* con un *client* compatibile.

Questa fase ha permesso di verificare la piena compatibilità del *server MCP* con *Elasticsearch 8.15* e di documentare i prerequisiti per una futura attivazione delle funzionalità AI.

5.10.3 Risultati e prospettive

La configurazione del *server MCP* costituisce una base solida per eventuali evoluzioni del progetto verso l'impiego di agenti di intelligenza artificiale direttamente all'interno di *Kibana*, non appena tali strumenti saranno pienamente supportati in versioni stabili dello *stack Elastic*.
L'infrastruttura realizzata è quindi predisposta per integrare funzionalità di analisi e automazione, mantenendo allo stesso tempo la compatibilità con gli ambienti utilizzati in produzione dall'azienda.

5.11 Sintesi dei flussi end-to-end

La Figura 5.6 fornisce una visione complessiva del funzionamento della piattaforma di osservabilità realizzata.
Il diagramma integra tutte le *pipeline* sviluppate durante il progetto, ovvero la raccolta di metriche, tracce e *log*, il monitoraggio sintetico e l'analisi finale, evidenziando il ruolo di ciascun componente e il flusso dei dati dalla sorgente fino alla visualizzazione in *Kibana*.

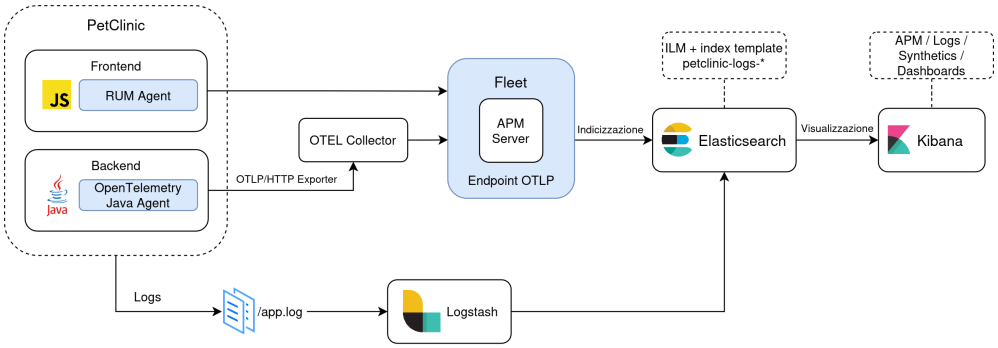


Figura 5.6: Sintesi dei flussi end-to-end

Capitolo 6

Conclusioni

6.1 Consuntivo finale

Il progetto ha portato alla realizzazione di una piattaforma completa di osservabilità per la *web application PetClinic*, comprendente la raccolta e l'analisi di *log*, metriche, tracce distribuite e *test* sintetici. È stata inoltre configurata un'infrastruttura basata su *Docker Compose*, integrata con *Fleet*, *Logstash* e l'*OpenTelemetry Collector*, affiancata da strumenti di analisi avanzati quali *Machine Learning* e *Anomaly Detection*.
Al termine dello *stage*, l'intera soluzione risulta funzionante, validata ed avviabile tramite un unico comando, garantendo riproducibilità e coerenza dell'ambiente.
L'intera piattaforma inoltre è compatibile con le versioni utilizzate in azienda dello *stack Elastic*, facilitando un eventuale integrazione futura.
Durante le ultime settimane di *stage* sono stati condotti più *test* sulla piattaforma alla presenza dei membri del gruppo, al fine di allineare tutti i presenti sulle funzionalità implementate e raccogliere *feedback* utili.
Al termine di questi *test*, la piattaforma è stata replicata in un ambiente di laboratorio aziendale, per valutarne il funzionamento in un contesto reale.
Alla fine del percorso di *stage* è stata inoltre resa disponibile una documentazione completa, ad illustrare il funzionamento di tutte le componenti della piattaforma, i pro e i contro delle tecnologie utilizzate e delle scelte effettuate, assieme alle possibili evoluzioni future.
In conclusione, il progetto ha raggiunto gli obiettivi prefissati, fornendo una soluzione efficace e ben documentata per l'osservabilità della *web app PetClinic*.

6.2 Valutazione dei rischi individuati

A conclusione del progetto è stata effettuata una revisione dei rischi identificati nella fase iniziale (Sezione 2.3) al fine di valutare quali si siano effettivamente presentati, quali siano stati mitigati e quali non abbiano avuto impatto sullo svolgimento delle attività.
In particolare:

- **Inesperienza nella suite Elastic:** si è verificato in modo parziale nelle prime settimane, ma è stato mitigato tramite studio autonomo e supporto del *tutor* aziendale.

- **Integrazione tra componenti Elastic:** il rischio si è verificato, nello specifico sono stati riscontrati alcuni problemi di comunicazione tra *Logstash*, *APM Server* e *Elasticsearch*, ma sono stati risolti tramite *debug* e lettura dei *log*.
- **Qualità e coerenza dei dati raccolti:** il rischio non si è manifestato.
- **Scalabilità e carico del sistema:** il rischio si è manifestato solamente all'interno dell'ambiente locale WSL[g] *Linux* a causa di una impostazione errata di memoria al container di *Elasticsearch*, ma è stato risolto aumentando la memoria allocata.
- **Implementazione di algoritmi di Machine Learning:** alcuni limiti di licenza hanno inizialmente rallentato l'adozione del modulo ML, successivamente superati tramite l'attivazione del periodo di prova per i *test* locali.
- **Problemi di configurazione delle pipeline Logstash:** il rischio non si è manifestato.
- **Accesso limitato a funzionalità premium di Elastic:** il rischio si è verificato nelle prime settimane; è stato poi mitigato con l'attivazione di un periodo di prova gratuito in locale.

Nel complesso, i rischi previsti si sono dimostrati utili a guidare il lavoro: quelli più rilevanti si sono presentati nella fase iniziale ma sono stati risolti, mentre altri non hanno avuto impatto significativo sul progetto.

6.3 Raggiungimento degli obiettivi

Per completezza, si riporta la validazione dei requisiti individuati durante l'analisi descritta al capitolo 3, divisi in funzionali, non funzionali, qualitativi e di vincolo.
Per ciascun requisito è stato indicato l'esito finale ottenuto al termine del progetto: soddisfatto, parzialmente soddisfatto oppure non applicabile.

6.3.1 Raggiungimento dei requisiti funzionali

I requisiti funzionali descrivono cosa deve fare il sistema. Sono le funzionalità concrete che la soluzione deve offrire per raggiungere gli obiettivi del progetto.

Tabella 6.1: Tabella del tracciamento dei requisiti funzionali con esito

Codice	Descrizione	Classificazione	Esito
ROF-1	Il sistema deve permettere la raccolta automatica di metriche e *log* relativi alla *web application* tramite agenti *OpenTelemetry* o *Elastic APM*.	Obbligatorio	Soddisfatto
ROF-2	Il sistema deve inviare i dati raccolti agli *endpoint Elasticsearch* per l'analisi e l'indicizzazione.	Obbligatorio	Soddisfatto
ROF-3	Il sistema deve consentire la creazione di *pipeline* di *log* tramite *Logstash* per filtraggio, trasformazione e inoltro dei dati in *Elasticsearch*.	Obbligatorio	Soddisfatto
ROF-4	Il sistema deve prevedere la configurazione di *Elastic Agents* (*Beats*/*APM*) per la raccolta dati della navigazione.	Obbligatorio	Soddisfatto
ROF-5	Il sistema deve generare *dashboard* avanzate e visualizzazioni in *Kibana*, con metriche di *performance*, accesso e flussi utente.	Obbligatorio	Soddisfatto
ROF-6	Il sistema deve permettere la verifica della corretta acquisizione dei dati e la loro indicizzazione in *Elasticsearch*.	Obbligatorio	Soddisfatto
ROF-7	Il sistema deve prevedere lo sviluppo e l'esecuzione di *script* automatizzati in *Python* o *Java* per la simulazione del traffico utente (*Synthetic Monitoring*).	Obbligatorio	Soddisfatto
ROF-8	Deve essere possibile filtrare e ricercare i *log* per *host*, servizio, livello di severità o periodo temporale.	Obbligatorio	Soddisfatto
RDF-9	Il sistema dovrebbe prevedere la configurazione di regole di *alerting* e notifiche in tempo reale per anomalie rilevate.	Desiderabile	Soddisfatto
RDF-10	Il sistema dovrebbe integrare algoritmi di *Machine Learning* per l'individuazione automatica di anomalie.	Desiderabile	Soddisfatto

Codice	Descrizione	Classificazione	Esito
RDF-11	Il sistema dovrebbe consentire l'esportazione delle *dashboard* o dei risultati delle *query* in formato PDF o CSV.	Desiderabile	Soddisfatto
RZF-12	Il sistema può prevedere un modulo aggiuntivo per la generazione automatica di report periodici delle metriche raccolte.	Opzionale	Parzialmente soddisfatto
RZF-13	Il sistema può consentire l'importazione automatica delle configurazioni APM da ambienti di *test* o *staging*.	Opzionale	Soddisfatto

6.3.2 Raggiungimento dei requisiti non funzionali

I requisiti non funzionali definiscono come il sistema deve comportarsi, cioè le sue proprietà di qualità interna. Non aggiungono nuove funzioni, ma impongono vincoli di prestazioni, sicurezza, disponibilità, scalabilità, affidabilità e manutenibilità.

Tabella 6.2: Tabella del tracciamento dei requisiti non funzionali con esito

Codice	Descrizione	Classificazione	Esito
RON-1	Il sistema deve essere scalabile e consentire l'aggiunta di nuove fonti di dati o agenti senza compromettere la stabilità.	Obbligatorio	Soddisfatto
RON-2	Il sistema deve garantire l'affidabilità nella trasmissione e nella conservazione dei dati raccolti.	Obbligatorio	Soddisfatto
RON-3	La piattaforma deve assicurare un tempo di latenza accettabile nella visualizzazione dei dati ($<$ 5 secondi per l'aggiornamento delle *dashboard*).	Obbligatorio	Soddisfatto
RDN-4	Il sistema dovrebbe garantire la possibilità di eseguire *test* di carico e stress per valutare la stabilità dell'ambiente.	Desiderabile	Non applicabile
RDN-5	Il sistema dovrebbe supportare l'autenticazione per la gestione degli accessi a *Kibana*.	Desiderabile	Soddisfatto

6.3.3 Raggiungimento dei requisiti qualitativi

I requisiti qualitativi specificano le proprietà qualitative che influenzano l'esperienza d'uso e la manutenibilità. Si concentrano su aspetti percepibili, come semplicità, chiarezza, flessibilità o estendibilità.

Tabella 6.3: Tabella del tracciamento dei requisiti qualitativi con esito

Codice	Descrizione	Classificazione	Esito
ROQ-1	L'interfaccia di *Kibana* deve offrire una rappresentazione chiara e intuitiva delle metriche principali.	Obbligatorio	Soddisfatto
ROQ-2	I dati devono essere visualizzabili in forma aggregata e filtrabile in base a intervalli temporali e categorie di evento.	Obbligatorio	Soddisfatto
ROQ-3	Le *dashboard* devono presentare una chiara distinzione cromatica tra metriche positive, neutre e anomale.	Obbligatorio	Soddisfatto
RDQ-4	Le *dashboard* dovrebbero essere personalizzabili dall'utente secondo criteri di interesse (*performance*, accessi, flussi).	Desiderabile	Soddisfatto
RZQ-5	Il sistema può includere un *layout dark/light mode* o temi grafici personalizzati per una migliore leggibilità.	Opzionale	Soddisfatto

6.3.4 Raggiungimento dei requisiti di vincolo

Impongono limitazioni o condizioni esterne al progetto: ambienti, tecnologie, compatibilità, strumenti, standard aziendali o legali.

Tabella 6.4: Tabella del tracciamento dei requisiti di vincolo con esito

Codice	Descrizione	Classificazione	Esito
ROV-1	Il sistema deve utilizzare i prodotti della suite *Elastic Stack* (*Elasticsearch, Logstash, Kibana, Beats, APM Server/Agent*).	Obbligatorio	Soddisfatto
ROV-2	L'ambiente operativo deve essere *Linux* (*Redphat* o distribuzioni equivalenti).	Obbligatorio	Soddisfatto
ROV-3	Tutti i componenti *software* devono essere compatibili con la versione di *Linux* installata (es. *Ubuntu 22.04 LTS* o *Red Hat 9*).	Obbligatorio	Soddisfatto
ROV-4	Le componenti devono rispettare le seguenti versioni minime: • *Python* >= 3.10 • *Java* >= 16 • *Node.js* >= 17 • *Logstash* >= 8.10 • *Kibana* >= 8.10 • *Beats* >= 8.10 • *APM Server* >= 8.10 • *Elasticsearch* >= 8.10 • *OpenTelemetry Java Agent* >= 1.26.0	Obbligatorio	Soddisfatto
ROV-5	La *web application* deve essere compatibile con i principali *browser* (*Chrome* >= 120, *Firefox* >= 115, *Edge* >= 120, *Safari* >= 15).	Obbligatorio	Soddisfatto
ROV-6	Le configurazioni devono essere eseguite in ambiente *Docker* o su infrastruttura aziendale.	Obbligatorio	Soddisfatto
RDV-7	La documentazione tecnica deve essere redatta in *Markdown*, *LaTeX* o PDF.	Desiderabile	Soddisfatto
RDV-8	Il sistema dovrebbe supportare la distribuzione tramite *Docker Compose*.	Desiderabile	Soddisfatto

6.4 Riepilogo dei requisiti raggiunti

Tabella 6.5: Riepilogo dei requisiti raggiunti

Tipologia	Obbligatorio	Desiderabile	Opzionale	Totale
Funzionali	8	3	2	13
Non funzionali	3	2	0	5
Qualitativi	3	1	1	5
Di vincolo	6	2	0	8
Totale	20	8	3	31

Dall'analisi conclusiva emerge che il progetto ha soddisfatto la totalità dei requisiti classificati come *obbligatori* e *desiderabili*, per un totale di 28 requisiti su 31. I tre requisiti classificati come *opzionali* sono stati valutati come non applicabili nell'ambito dello stage, in quanto non rientravano negli obiettivi tecnici concordati con l'azienda. Nel dettaglio, sono stati soddisfatti:

- 8 requisiti funzionali obbligatori, 3 desiderabili e 1 opzionali;
- 3 requisiti non funzionali obbligatori e 1 desiderabile;
- 3 requisiti qualitativi obbligatori, 1 desiderabile e 1 opzionale;
- 6 requisiti di vincolo obbligatori e 2 desiderabili.

Complessivamente, il sistema realizzato rispetta pienamente le richieste principali definite nella fase di analisi dei requisiti.

6.5 Conoscenze acquisite

Lo sviluppo del progetto mi ha permesso di sviluppare competenze avanzate nell'ambito dell'osservabilità delle applicazioni web, comprendendo l'utilizzo di tecnologie quali *Elasticsearch*, *Kibana*, *Fleet*, *Logstash* e l'*OpenTelemetry Collector*.
Ho acquisito familiarità con la progettazione di *pipeline* di dati, l'analisi delle metriche applicative, la creazione di *dashboard*, l'indicizzazione tramite *Elasticsearch* e la gestione del monitoraggio sintetico.
Sul piano operativo ho migliorato la capacità di lavorare in ambienti containerizzati, diagnosticare problemi distribuiti e strutturare documentazione tecnica chiara e replicabile.
L'opportunità di sviluppare in autonomia una soluzione completa mi ha permesso di affinare le competenze di *problem solving*, gestione del tempo e comunicazione tecnica.
In sintesi, il progetto ha rappresentato un'importante occasione di crescita professionale, fornendomi importanti competenze pratiche e teoriche nel campo dell'osservabilità delle applicazioni moderne.

6.6 Valutazione personale

Il progetto di stage ha rappresentato per me un'importante opportunità di crescita. Sono molto grato al team di Kirey per avermi supportato in ogni fase del percorso e per avermi seguito con attenzione e competenza.
Lavorare su un progetto come questo è stato stimolante in quanto mi è stata data carta

bianca su come strutturare la piattaforma, permettendomi di sperimentare diverse soluzioni prima di arrivare alla soluzione definitiva con *docker compose*.
Il lavoro svolto su questo progetto mi ha portato sicuramente a migliorare le mie competenze tecniche, in particolare nell'ambito dell'osservabilità delle applicazioni *web* e della containerizzazione.
Inoltre, ho avuto l'opportunità di sviluppare competenze trasversali come la gestione del tempo, la comunicazione tecnica e il *problem solving*, grazie alle riunioni con il *team* e ai momenti di allineamento.
In conclusione, ritengo che questo progetto di *stage* sia stato un'esperienza molto positiva e formativa, che mi ha permesso di acquisire nuove competenze e di crescere professionalmente.

Acronimi e abbreviazioni

AI Artificial Intelligence. 67

API Application Program Interface. 67

APM Application Performance Monitoring. 17, 35, 57, 58, 67

CPU Central Processing Unit. 67

CSV Comma-Separated Values. 67

ESG Environmental, Social and Governance. 68

EUM End User Monitoring. 22, 68

GB Gigabyte. 68

gRPC gRPC Remote Procedure Calls. 68

HTTP HyperText Transfer Protocol. 68

HTTPS HyperText Transfer Protocol Secure. 68

ILM Index Lifecycle Management. 68

MCP Model Context Protocol. 69

ML Machine Learning. 69

MySQL My Structured Query Language. 69

OTLP OpenTelemetry Protocol. 69

PDF Portable Document Format. 69

RUM Real User Monitoring. 69

SDK Software Development Kit. 69

SQL Structured Query Language. 70

TLS Transport Layer Security. 70

UI User Interface. 70

UML Unified Modeling Language. 70

WSL Windows Subsystem for Linux. 70

Glossario

AI *Artificial Intelligence* (ing. Intelligenza Artificiale) è un ramo dell'informatica che si occupa della creazione di sistemi in grado di svolgere compiti che normalmente richiederebbero l'intelligenza umana, come il riconoscimento vocale, la visione artificiale, l'apprendimento automatico e la risoluzione di problemi complessi. L'obiettivo principale dell'AI è sviluppare algoritmi e modelli che permettano alle macchine di apprendere dai dati, adattarsi a nuove situazioni e prendere decisioni autonome, migliorando così l'efficienza e l'efficacia in vari settori, tra cui la medicina, la finanza, l'industria e i servizi. 5, 31, 37, 45, 53, 65

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 25, 65

APM *Application Performance Monitoring* (ing. Monitoraggio delle Prestazioni delle Applicazioni) è un insieme di pratiche e strumenti utilizzati per monitorare, misurare e gestire le prestazioni e la disponibilità delle applicazioni software. L'obiettivo principale dell'APM è garantire che le applicazioni funzionino in modo ottimale, offrendo un'esperienza utente fluida e senza interruzioni. Ciò include il monitoraggio di vari parametri come tempi di risposta, tassi di errore, utilizzo delle risorse e throughput, nonché l'identificazione e la risoluzione di problemi che potrebbero influire sulle prestazioni dell'applicazione. 5, 11, 15, 17, 21, 23, 37, 42–44, 65

CPU *Central Processing Unit* (ing. Unità Centrale di Elaborazione), comunemente nota come processore, è il componente principale di un computer responsabile dell'esecuzione delle istruzioni dei programmi e del controllo delle operazioni del sistema. La CPU interpreta e processa i dati, eseguendo operazioni aritmetiche, logiche e di controllo, fungendo da cervello del computer. Le prestazioni della CPU sono influenzate da vari fattori, tra cui la velocità di clock, il numero di core e l'architettura del processore. 17, 18, 21, 42, 65

CSV *Comma-Separated Values* (ing. Valori Separati da Virgola) è un formato di file di testo utilizzato per rappresentare dati tabulari, in cui ogni riga del file corrisponde a un record e i valori all'interno di ogni record sono separati da virgole. Il formato CSV è ampiamente utilizzato per l'importazione e l'esportazione di dati tra

diversi programmi, come fogli di calcolo, database e applicazioni di analisi dei dati, grazie alla sua semplicità e compatibilità con molti software. 11, 18, 58, 65

ESG *Environmental, Social, Governance*, (ing. Ambientale, Sociale e di Governance) è un acronimo che indica i criteri utilizzati per valutare la sostenibilità e la responsabilità di un'azienda. L'aspetto ambientale riguarda pratiche come riduzione delle emissioni, uso delle risorse e tutela del clima; quello sociale include rapporti con dipendenti, clienti e comunità, promuovendo inclusione e condizioni di lavoro eque; infine, la governance si riferisce ai meccanismi di gestione, trasparenza, etica e correttezza nei processi decisionali. 2, 65

EUM *End User Monitoring* (ing. Monitoraggio dell'Utente Finale) è una tecnica di monitoraggio delle prestazioni delle applicazioni che si concentra sull'analisi dell'esperienza degli utenti finali mentre interagiscono con un'applicazione o un sistema. L'EUM raccoglie dati in tempo reale sulle prestazioni percepite dagli utenti, come i tempi di risposta, la disponibilità del servizio e gli errori riscontrati, fornendo informazioni preziose per migliorare l'usabilità e l'efficienza dell'applicazione. 65

GB *Gigabyte* (ing. Gigabyte) è un'unità di misura della capacità di memorizzazione dei dati nei sistemi informatici, equivalente a 1.073.741.824 byte nel sistema binario comunemente utilizzato in informatica, o a 1.000.000.000 byte nel sistema decimale.. 39, 65

gRPC *gRPC Remote Procedure Calls* (ing. Chiamate di Procedura Remota gRPC) è un framework open source sviluppato da Google che consente la comunicazione tra applicazioni distribuite attraverso chiamate di procedura remota (RPC). gRPC utilizza il protocollo HTTP/2 per il trasporto dei dati e supporta vari linguaggi di programmazione. 43, 65

HTTP *HyperText Transfer Protocol* (ing. Protocollo di Trasferimento Ipertestuale) è un protocollo di comunicazione utilizzato per la trasmissione di dati su Internet, in particolare per il trasferimento di pagine web tra server e client (come i browser web). HTTP definisce le regole e le convenzioni per la richiesta e la risposta di risorse, come documenti HTML, immagini, video e altri contenuti multimediali. Il protocollo è basato su un modello client-server, in cui il client invia una richiesta al server, che elabora la richiesta e restituisce una risposta contenente i dati richiesti. 15–17, 42, 43, 65

HTTPS *HyperText Transfer Protocol Secure* (ing. Protocollo di Trasferimento Ipertestuale Sicuro) è una versione sicura del protocollo HTTP che utilizza la crittografia SSL/TLS per proteggere i dati trasmessi tra il client e il server, garantendo l'integrità e l'autenticità delle comunicazioni su Internet. 39, 65

ILM *Index Lifecycle Management* (ing. Gestione del Ciclo di Vita degli Indici) è una funzionalità di Elasticsearch che consente di automatizzare la gestione del ciclo di vita degli indici, definendo politiche per la creazione, l'archiviazione, la rotazione e la cancellazione degli indici in base a criteri specifici come l'età, le dimensioni o lo stato di salute. 37, 39, 41, 65

MCP *Model Context Protocol* (ing. Protocollo di Contesto del Modello) è un protocollo di comunicazione progettato per facilitare l'interazione tra modelli di intelligenza artificiale generativa e sistemi esterni, consentendo lo scambio di informazioni contestuali e migliorando la capacità dei modelli di fornire risposte pertinenti e accurate. 31, 37, 45, 52, 65

ML *Machine Learning* (ing. Apprendimento Automatico) è un ramo dell'intelligenza artificiale che si concentra sullo sviluppo di algoritmi e modelli statistici che consentono ai computer di apprendere dai dati e migliorare le proprie prestazioni nel tempo senza essere esplicitamente programmati per ogni compito specifico. 49, 56, 65

MySQL *My Structured Query Language* è un sistema di gestione di database relazionali (RDBMS) open source basato sul linguaggio SQL (Structured Query Language). MySQL è ampiamente utilizzato per la gestione e l'organizzazione di grandi quantità di dati in applicazioni web, software aziendali e sistemi di gestione dei contenuti. Offre funzionalità avanzate come transazioni, replica, partizionamento e supporto per vari motori di archiviazione, rendendolo una scelta popolare per sviluppatori e amministratori di database in tutto il mondo. 17, 65

OTLP *OpenTelemetry Protocol* (ing. Protocollo OpenTelemetry) è un protocollo di comunicazione standardizzato utilizzato per la trasmissione di dati di telemetria, come tracce, metriche e log, tra componenti di sistemi di osservabilità basati su OpenTelemetry. Il protocollo supporta vari formati di trasporto, facilitando l'integrazione con una vasta gamma di tecnologie e infrastrutture. 15, 16, 25, 37, 42–44, 65

PDF *Portable Document Format* (ing. Formato di Documento Portatile) è un formato di file sviluppato da Adobe Systems per rappresentare documenti in modo indipendente dall'hardware, dal software e dal sistema operativo utilizzati per crearli o visualizzarli. I file PDF possono contenere testo, immagini, grafica vettoriale e persino elementi interattivi come moduli compilabili e collegamenti ipertestuali. Il formato PDF è ampiamente utilizzato per la condivisione di documenti, in quanto preserva il layout e la formattazione originale, garantendo che il documento venga visualizzato correttamente su qualsiasi dispositivo o piattaforma. 11, 14, 18, 58, 61, 65

RUM *Real User Monitoring* (ing. Monitoraggio degli Utenti Reali) è una tecnica di monitoraggio delle prestazioni delle applicazioni web che si concentra sull'analisi del comportamento e dell'esperienza degli utenti reali mentre interagiscono con un sito web o un'applicazione. A differenza delle soluzioni di monitoraggio sintetico, che simulano il traffico utente attraverso script predefiniti, il RUM raccoglie dati direttamente dai browser degli utenti, fornendo informazioni dettagliate sulle prestazioni percepite, i tempi di caricamento delle pagine, gli errori riscontrati e altri aspetti critici dell'esperienza utente. 16, 65

SDK *Software Development Kit* (ing. Kit di Sviluppo Software) è un insieme di strumenti, librerie, documentazione e esempi di codice forniti agli sviluppatori per facilitare la creazione di applicazioni software specifiche per una piattaforma, un framework o un sistema operativo. 23, 65

SQL *Structured Query Language* (ing. Linguaggio di Query Strutturato) è un linguaggio standardizzato per l'interrogazione e la manipolazione di database relazionali. SQL consente di eseguire operazioni come la selezione, l'inserimento, l'aggiornamento e la cancellazione di dati, nonché la creazione e la modifica di strutture di database. 30, 65

TLS *Transport Layer Security* (ing. Sicurezza del Livello di Trasporto) è un protocollo crittografico progettato per fornire comunicazioni sicure su reti di computer, come Internet. TLS garantisce la riservatezza, l'integrità e l'autenticità dei dati trasmessi tra client e server, proteggendo le informazioni sensibili da intercettazioni e manomissioni. 45, 65

UI *User Interface* (ing. Interfaccia Utente) è il punto di interazione tra un utente e un sistema informatico, che può includere elementi visivi come finestre, icone, menu e pulsanti, nonché componenti tattili, vocali o gestuali.. 52, 66

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 66

WSL *Windows Subsystem for Linux* (ing. Sottosistema Windows per Linux) è una funzionalità di Windows 10 e versioni successive che consente agli utenti di eseguire un ambiente Linux completo direttamente su Windows, senza la necessità di una macchina virtuale o di un dual boot. WSL permette agli sviluppatori di utilizzare strumenti, librerie e applicazioni Linux nativi all'interno di Windows. 5, 56, 66

Bibliografia

Siti web consultati

Agent-Based Monitoring. URL: https://www.motadata.com/it-glossary/agent-based-monitoring/ (cit. a p. 22).

Agentless Monitoring. URL: https://www.solarwinds.com/resources/it-glossary/agentless-monitoring (cit. a p. 22).

Distributed Tracing. URL: https://aws.amazon.com/what-is/distributed-tracing/ (cit. a p. 22).

Docker. URL: https://www.docker.com/ (cit. a p. 30).

Docker Compose. URL: https://docs.docker.com/compose/ (cit. a p. 31).

Elastic Agent. URL: https://www.elastic.co/elastic-agent (cit. a p. 24).

Elastic APM. URL: https://www.elastic.co/apm/ (cit. a p. 25).

Elasticsearch. URL: https://www.elastic.co/elasticsearch/ (cit. a p. 23).

Java Programming Language. URL: https://www.java.com/en/ (cit. a p. 28).

JavaScript Programming Language. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (cit. a p. 27).

Kibana. URL: https://www.elastic.co/kibana/ (cit. a p. 24).

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.

Model Context Protocol. URL: https://modelcontextprotocol.io/docs/getting-started/intro (cit. a p. 31).

MySQL. URL: https://www.mysql.com/ (cit. a p. 30).

Node.js. URL: https://nodejs.org/ (cit. a p. 28).

OpenTelemetry. URL: https://opentelemetry.io/ (cit. a p. 25).

Python Programming Language. URL: https://www.python.org/ (cit. a p. 27).

Real User Monitoring. URL: https://en.wikipedia.org/wiki/Real_user_monitoring (cit. a p. 22).

RUM Agent JS. URL: https://www.elastic.co/docs/reference/apm/agents/rum-js (cit. a p. 26).

Selenium. URL: https://www.selenium.dev/ (cit. a p. 29).

Spring Framework. URL: https://spring.io/ (cit. a p. 28).

Synthetic Monitoring. URL: https://en.wikipedia.org/wiki/Synthetic_monitoring (cit. a p. 22).

Synthetic Monitoring. URL: https://www.elastic.co/what-is/synthetic-monitoring (cit. a p. 26).

Visual Studio Code. URL: https://code.visualstudio.com/ (cit. a p. 29).