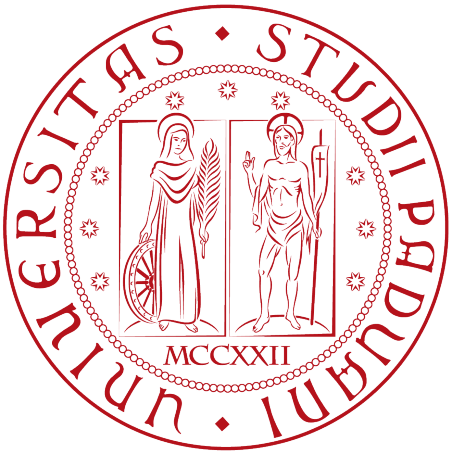
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Realizzazione e analisi prestazionale di un database NoSQL per la possibile migrazione da un database relazionale

Tesi di laurea

Relatore

Prof.Luigi De Giovanni

Laureando

Nicholas Sertori

Anno Accademico 2021-2022

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

— Oscar Wilde

Dedicato a ...

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage svolto presso l'azienda Ifin Sistemi s.r.l. Lo stage è stato svolto alla conclusione del percorso di studi della laurea triennale in Informatica, occupando circa trecentoventi ore divise in otto settimane. Lo scopo del progetto svolto è stato di effettuare uno studio di fattibilità per l'integrazione di una soluzione di database NoSQL nei prodotti dell'azienda. Lo studio di fattibilità ha comportato una fase di analisi delle varie soluzioni NoSQL esistenti sul mercato, una fase di analisi delle soluzioni attualmente adottate all'interno dei prodotti Ifin, ed infine una fase di valutazione pratica delle soluzioni individuate, con relativi benchmark per il confronto delle prestazioni ed un approfondimento sulle differenze di progettazione tra database relazionali classici e database NoSQL.

"Life is really simple, but we insist on making it complicated"

— Confucius

Ringraziamenti

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

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

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Dicembre 2022 Nicholas Sertori

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

L'azienda proponente è Ifin Sistemi s.r.l., un'azienda di prodotto che si occupa principalmente di informatica finanziaria. Il suo core business è incentrato su piattaforme che facilitano l'archiviazione di pratiche e documenti legali in modo sicuro e affidabile, e mediano l'invio di fatture elettroniche tra aziende e Sistema di Interscambio, un sistema informatico gestito dall'Agenzia delle Entrate.

Figura 1.1: Logo di Ifin Sistemi s.r.l.

1.2 Situazione Attuale

I due prodotti di punta dell'azienda, che compongono il core business sopra accennato, sono LegalArchive e InvoiceChannel. All'interno di Ifin coesistono vari team che mantengono la codebase di questi software, occupandosi della loro manutenzione e della modellazione di nuove funzionalità richieste dai clienti.
A livello pratico, il codice per i software di Ifin è scritto in Java, appoggiato quando serve al framework Spring.
All'interno di quello che è lo stack tecnologico aziendale, data la scelta del linguaggio di programmazione, troviamo quello che può essere considerato uno standard per lo sviluppo di applicativi che fanno largo uso di database. Tecnologie come JPA, JDBC, Tomcat, Hybernate ed Eclipselink risultano essere mattoni fondamentali alla base dei

software di Ifin.
Infine, per quanto riguarda la scelta dei database veri e propri, anche in questo caso l'azienda fa riferimento a quelli che sono gli standard dell'industria. Si parla quindi di database relazionali, e più nello specifico di Oracle Database e Microsoft SQL Server.

1.3 Esigenze da cui nasce l'idea del progetto

Sebbene attualmente, all'interno dell'azienda, siano implementate varie soluzioni intelligenti per garantire il funzionamento delle piattaforme anche in situazioni di stress dei sistemi (come per esempio il partizionamento delle tabelle più grandi), i database relazionali possono soffrire di problemi di scalabilità quando la mole di dati che devono gestire raggiunge determinate dimensioni.
L'utilizzo di una soluzione NoSQL è pertanto un'allettante alternativa, proprio perchè spesso scalabilità e affidabilità sono caratteristiche centrali di queste tecnologie. Occorre tuttavia effettuare uno studio più completo per determinare se l'utilizzo di questo tipo di database si presta realmente alle necessità e alle complessità dei sistemi di Ifin, per poter giustificare un investimento non indifferente di risorse nella conversione e migrazione che ne conseguirebbe. Il progetto di stage si inserisce in questo contesto, unendo le necessità dell'azienda alla possibilità di effettuare una ricerca consocitiva dei database NoSQL.

1.4 Organizzazione del testo

Il secondo capitolo descrive il progetto di stage e presenta una iniziale pianificazione delle attività.

Il terzo capitolo descrive il contesto del progetto, presentando nel dettaglio le tecnologie studiate e quelle già implementate dall'azienda.

Il quarto capitolo approfondisce la fase di preparazione alla sperimentazione, con il settaggio degli strumenti necessari e l'organizzazione dell'ambiente di test.

Il quinto capitolo descrive la fase di sperimentazione e raccolta dei dati utili al confronto che sta al centro del progetto.

Il sesto capitolo contiene le conclusioni elaborate al termine del tirocinio.

Capitolo 2

Descrizione dello Stage

2.1 Introduzione al Progetto

L'obiettivo dello stage è quello di effettuare uno studio preliminare delle tecnologie che gravitano attorno al concetto di NoSQL, per evidenziarne vantaggi e svantaggi, in modo da poter valutare in maniera concreta eventuali possibilità di integrazione nello stack aziendale.
Una volta portato a termine questo studio, si vuole portare a confronto le soluzioni attualmente adottate con quelle analizzate, per valutare in modo concreto in che modo queste ultime possono portare ad un miglioramento nella gestione e nell'utilizzo degli applicativi aziendali.

2.2 Obiettivi Formativi

Gli obiettivi formativi dell'attività di stage sono i seguenti:

* Approfondire conoscenze in ambito NoSQL;
* Apprendere come effettuare attività di test di carico per la valutazione prestazionale di un Database;
* Apprendere metodologie ed approcci propri dell'ambiente lavorativo, diversi da quelli universitari.

2.3 Attività preventivate

La durata complessiva dello stage è stata di 8 settimane di lavoro a tempo pieno per un totale di circa trecentoventi ore.
Secondo il piano di lavoro iniziale definito con l'azienda, le attività sono distribuite come riportato in Tabella 2.1.

Durata in ore	Settimana	Descrizione

40	1	∗ Formazione sullo stack operativo e di sviluppo aziendale; ∗ Formazione stack Java EE
80	2, 3	∗ Studio NoSQL in generale; ∗ Verifica soluzioni NoSQL specifiche; ∗ Identificazione di soluzioni NoSQL enterprise da analizzare e KPI aziendali.
80	4, 5	∗ Analisi di dettaglio delle soluzioni precedentemente identificate.
80	6, 7	∗ Creazione e codifica di test per lo stuio delle performance di carico e aderenza ai KPI individuati.
40	8	∗ Revisione test e stesura documentazione finale.
Totale ore:		**320**

Tabella 2.1: Pianificazione delle attività

Durante il periodo di stage in azienda il piano di lavoro ha subito modifiche e di conseguenza il consuntivo delle attività svolte riportato nel capitolo conclusivo diverge dalla pianificazione qui presentata. Tali modifiche sono state effettuate in risposta al naturale evolversi del progetto di fronte ad imprevisti ed esigenze nate durante il percorso.
Durante tutta la durata del tirocinio si è lavorato a contatto con il relatore preposto all'interno dell'azienda e con varie altre figure di riferimento più esperte nei vari ambiti toccati dal progetto.

Capitolo 3

Contesto

Questo capitolo è dedicato all'introduzione delle nozioni emerse dalla fase di apprendimento svolta durante il tirocinio, riguardanti nello specifico le varie di tipologie di DB NoSQL esistenti.

3.1 Cosa significa NoSQL

Quando si parla di database NoSQL si intendono tutte quelle tecnologie di persistenza di dati che non prevedono strettamente l'utilizzo del paradigma SQL. L'acronimo significa infatti Not-only Structured Query language.
Tuttavia, mentre le tecnologie classiche che rientrano sotto l'ombrello dei RDBMS (relational database management systems) sono in qualche modo uniformate dalla "lingua franca" che condividono (SQL), le implementazioni che ricadono nel paradigma NoSQL sono estremamente varie nell'effettivo modo in cui gestiscono i dati, e di conseguenza nei linguaggi che adottano.
Questo può rendere il processo di selezione più complesso, ma fornisce anche un'ampia gamma di opzioni che, se scelte con cognizione di causa, possono portare a soluzioni estremamente specializzate ed efficaci.

3.2 Idee e Implementazioni

Vengono di seguito elencati i vari sottogruppi che possiamo individuare all'interno del panorama NoSQL.

3.2.1 Key-Value Store

Questa categoria di DB rappresenta in realtà un modo per immagazzinare informazioni, basato sulla strutturazione dei dati in coppie "chiave-valore".
Tutte le operazioni effettuate su un DB di questo tipo si basano quindi sulla chiave per recuperare il suo valore associato. Nella sua forma più semplcie, un key-value store funziona esattamente come un dizionario.

Come esempio abbiamo il caso di Redis.
Nasce inizialmente come DB appartenente al paradigma "key-value", composto quindi da un set di chiavi a cui sono legati dei valori, contenuto per intero nella memoria RAM.
L'idea era di avere una sorta di cache in cui salvare dei dati frequentemente utilizzati per poterli recuperare molto velocemente.
Se inizialmente veniva utilizzato a supporto di altri DB, nel tempo Redis è stato ampliato fino a diventare una soluzione unica (Redis Stack), in grado grazie ai suoi moduli di effettuare ricerche fulltext, visualizzare le informazioni in grafi ed implementare il salvataggio di documenti in formato JSON su memoria persistente in un database distribuito.
Il potere di questa soluzione sta nell'avere tutte queste funzionalità all'interno dello stesso prodotto, garantendo una semplificazione non indifferente del processo di integrazione.

Figura 3.1: Logo di Redis

3.2.2 Wide Column Store

Questo tipo di database NoSQL estende il concetto di "key-value", dove le informazioni sono raccolte in colonne, le colonne in righe, le righe in tabelle e quest'ultime sono raccolte sotto un cosiddetto "keyspace". A differenza dei database relazionali classici, questo tipo di soluzione non necessita di uno schema, la struttura che definisce il contenuto di una tabella nei RDBMS. Questo permette ai Wide Column DB di accettare dati non strutturati, diventando quindi molto più flessibile.
Un'altra importante differenza è che questo tipo di DB è decentralizzato e può essere scalato a dimensioni considerevoli senza troppi problemi, grazie al modo in cui è strutturato.

Prendiamo come esempio cardine il DB Cassandra, che è un database NoSQL distribuito, decentralizzato, scalabile e ad "alta disponibilità". Questo significa che può essere fatto operare su macchine diverse per spezzare il carico, ma soprattutto che non segue il paradigma master-slave. In questo modo tutti i nodi (client) sono omogenei e hanno gli stessi privilegi. La decentralizzazione permette di garantire la disponibilità del sistema, ad esempio quando uno o più nodi dovessero essere irraggiungibili o non responsivi.
Si parla poi di scalabilità per gli stessi motivi, poiché aggiungere nodi alla rete e ridistribuire il carico è estremamente facile.
Cassandra sfrutta inoltre un linguaggio proprietario simile a SQL, denominato CQL, che risulta quindi facilmente comprensibile se si è abituati a lavorare con DB relazionali.

Questo tipo di DB viene utilizzato quando la disponibilità dei dati 24/7 è una priorità, specialmente se tali dati sono in costante crescita, e la loro struttura tende a cambiare.

Il costo da pagare come tradeoff per le potenzialità elencate è una minore consistency dei dati, ma l'argomento verrà approfondito nella sezione 3.3.

Figura 3.2: Logo di Cassandra

3.2.3 Document Store

Questo paradigma racchiude probabilmente la famiglia più ampia e utilizzata di database NoSQL, presentando comunque soluzioni diverse al suo interno.
L'idea principale è sempre quella di non fare uso di uno schema per strutturare i propri dati. Al posto di avere righe in una tabella, in questi DB si usano documents raggruppati in collections. I documenti non seguono una struttura uniforme all'interno della stessa collezione, e sono quasi sempre salvati in formato JSON o simili.
In questo tipo di struttura la lettura dei dati è più rapida, a discapito di scrittura e update.
Generalmente sono di facile utilizzo e sono tra i DB NoSQL più versatili.

Prenderemo come esempio MongoDB e CouchDB.
MongoDB è uno dei DB NoSQL più popolari e diffusi, sfrutta documenti e collezioni, salva i propri dati in formato BSON (Binary-JSON), e adotta il paradigma master-slave.
Funziona molto bene quando la funzione principale di cui si ha bisogno è il salvataggio di grosse moli di dati, mentre è meno performante se questi dati devono essere prelevati e rielaborati, specialmente quando è necessario unire dati provenienti da documenti o collezioni diverse.
CouchDB è piuttosto simile, sebbene sia un progetto meno popolare. Differisce da Mongo per come salva i documenti (direttamente in formato JSON) e per l'architettura di base, che si distanzia dal paradigma master-slave e sfrutta nodi multipli per mantenere le informazioni sempre disponibili, a discapito della loro consistenza.

Figura 3.3: Logo di MongoDB a sinistra, CouchDB a destra

3.2.4 Graph Database

Esistono alcune categorie di DB all'interno del panorama NoSQL che sono state sviluppate soddisfare necessità specifiche. Una di queste categorie è quella dei DB basati

sui grafi.
Se nei RDBMS le relazioni sono sfruttate per collegare le tabelle, nel caso dei grafi le relazioni diventano vere e proprie entità, al pari dei nodi che esse collegano.
Questo consente di recuperare i dati con richieste (query) più concise e leggibili, oltre che ridurre i tempi di attesa.
Questo tipo di DB funziona al meglio in situazioni dove le query si basano molto sulle relazioni tra i dati, dove un DB relazionale dovrebbe operare molte operazioni di Join che aumentano notevolmente il tempo di elaborazione.

Un esempio di DB in questa categoria è Neo4j.
Come i DB relazionali, Neo4j è "ACID compliant", ovvero rispetta i parametri di atomicità, consistenza, isolazione e durabilità. Non è tuttavia un database distribuito e soffre quindi quando si parla di scalabilità.
Sfrutta un linguaggio molto intuitivo e simile a SQL per le query, chiamato cypher.

Come già detto, questo tipo di soluzione è utile in determinati campi e risulta molto interessante, ma può risultare inutile, o addirittura un ostacolo, se utilizzata in casi in cui non è necessaria.

Figura 3.4: Logo di neo4j

3.2.5 Search Engine

Un'altra soluzione interessante, ma altrettanto specifica, è quella dei cosiddetti "full-text search engine". Questo tipo di database è piuttosto simile, in superficie, a quelli basati sui documenti, con la differenza che il search engine analizza i contenuti del database e ne fornisce un indice. In questo modo la ricerca di dati, che ovviamente sfrutta tale indice, risulta estremamente rapida anche su dataset molto grandi, con la possibilità di implementare anche vari filtri per migliorare la user experience.

L'esempio più classico di implementazione di questo tipo di tecnologia è Elasticsearch, che è inizialmente nato come search engine e si è successivamente espanso per fornire le funzionalità classiche di un normale database.

Figura 3.5: Logo di Elasticsearch

3.3 CAP Theorem, Facilità di Integrazione

Nel tentativo di categorizzare i DB distribuiti, siano essi relazionali o meno, si sfrutta spesso il cosiddetto CAP Theorem, secondo cui un database distribuito può fornire soltanto due delle seguenti tre garanzie:

* **Consistency**: i dati sono sempre coerenti all'interno dei vari nodi distribuiti;
* **Availability**: i dati sono sempre disponibili in ogni momento;
* **Partition Tolerance**: i dati sono disponibili anche se partizionati, ovvero separati "orizzontalmente" su nodi diversi.

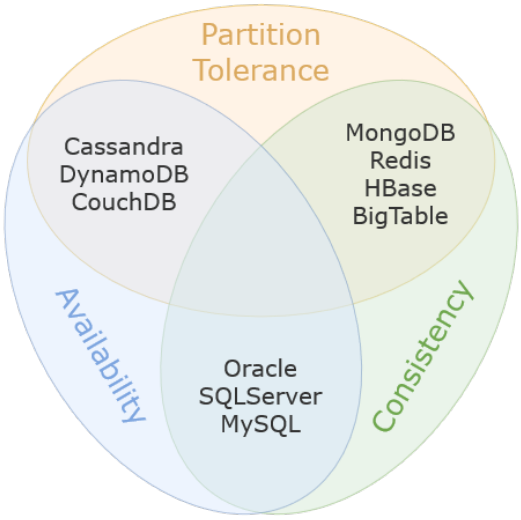


Figura 3.6: Rappresentazione del CAP Theorem, con posizionamento di alcuni database al suo interno in base alle loro caratteristiche

Mentre le soluzioni classiche (DB relazionali) garantiscono Consistenza e Disponibilità, ma hanno grossi problemi con la scalabilità orizzontale e la separazione dei dati in cluster diversi, le soluzioni NoSQL tendenzialmente garantiscono il Partizionamento, e possono quindi soddisfare soltanto una delle altre due caratteristiche.

In base quindi alle necessità di Ifin Sistemi, sarà necessario valutare quale tipo di database adottare anche in base al CAP theorem.

3.3.1 Integrazione

Tenendo in considerazione che lo scopo della ricerca che si è condotta rimane quello di studiare la possibilità di integrazione nello stack aziendale, è necessario valutare

quanto i database individuati si prestino a tale operazione.

Per ognuna delle implementazioni viste precedentemente è necessario seguire regole specifiche per mettere in comunicazione il proprio applicativo con il database.
A volte alcune soluzioni sembrano più funzionali al proprio caso d'uso, ma non hanno un sistema ben documentato/conosciuto/sviluppato per interfacciarsi con l'applicativo che stiamo sviluppando, solitamente per motivi di compatibilità dei linguaggi e diffusione di librerie e driver utili.
A tale scopo elenchiamo i software individuati e le loro opzioni di integrazione, tenendo a mente che lo stack aziendale si basa sull'utilizzo di java come linguaggio di programmazione.

Redis Stack

Per quanto riguarda Redis, Redis Stack e tutti i moduli ad esso collegati, il modo più semplice e "ad alto livello" per interagire con il database è sfruttare Redis OM, costruito sulla base di Spring Data Redis. Si tratta di un modulo del progetto "Spring Data", che racchiude librerie specifiche per interagire con diversi database usando il framework Spring.

Cassandra

Cassandra può essere utilizzato in modo facile tramite Astra DB, un multi-cloud DBaaS costruito su Cassandra.
Per interagire con Astra si possono sfruttare le soluzioni più disparate, dall'utilizzo delle REST API all'implementazione di driver specifici per Java che ne facilitano l'utilizzo.

MongoDB

Mongo è una delle implementazioni NoSQL più diffuse e versatili. Si può utilizzare Atlas come "MongoDB as a Service", e anche in questo caso viene fornita la Query API per effettuare operazioni sui dati. Mongo fornisce driver per l'integrazione con Java, ma si può direttamente usare Spring Data, espressamente creato per fornire un modello familiare per datastores basato su Spring.

CouchDB

CouchDB non possiede librerie specifiche per l'integrazione e la comunicazione con applicativi scritti in Java. Quelle che esistono spesso non sono ufficiali e quindi utilizzabili "a proprio rischio e pericolo". Risulta possibile implementare la comunicazione direttamente tramite REST API, ma è dispendioso.
Esiste tuttavia una versione di CouchDB sviluppata da IBM, denominata Cloudant, che propone il servizio cloud e l'integrazione con Java.

Neo4j

Anche Neo4j ha recentemente fornito una propria piattaforma, lanciando il progetto Aura per fornire DBaaS, con supporto per integrazione con Spring Boot e Spring Data.

Elasticsearch

Elasticsearch sfrutta Elastic Cloud come soluzione distribuita.
Possiede varie funzionalità interessanti tra cui la possibilità di migrare i propri dati da un cloud all'altro, a prescindere dal provider.

3.4 Panoramica sui prodotti Ifin e sulle soluzioni RDBMS adottate in azienda

La seguente sezione è il risultato del dialogo avuto con i team leader di tre diversi gruppi di sviluppo interni all'azienda, necessario per comprendere più nel dettaglio il funzionamento dei software che essa produce e mantiene.
Nello specifico, si è ritenuto importante capire quali fossero le tecnologie coinvolte all'interno di questi prodotti, come essi comunichino con i database che sono al centro delle dissertazioni presenti in questa tesi, ed infine quali sono i colli di bottiglia che essi affrontano, per cominciare a farsi un'idea sui punti critici che una migrazione verso un sistema diverso potrebbe migliorare o risolvere.

3.4.1 LegalArchive

Il primo prodotto analizzato è Legal Archive, un applicativo che si occupa di archiviazione di documenti con importanza legale.
Questo software si occupa di prendere in carico i documenti che un'ente desidera conservare, e tradurli in informazioni che possano essere inserite in un database.
Questo è soltanto uno dei suoi scopi, ma è quello che più da vicino riguarda gli argomenti affrontati in tirocinio.
Per comunicare con i DB, che in questo caso sono distribuzioni enterprise di Microsoft SQL Server e Oracle Server, LegalArchive si serve di uno stack di strumenti che semplificano tale dialogo.
Il database viene montato su un server Tomcat, e per fare uso delle informazioni presenti all'interno di esso si sfruttano il connettore JDBC ed EclipseLink come implementazione della specifica JPA.
L'architettura si basa sul pattern Model-View-ViewModel, sfruttando DTO e DAO all'interno del modello per costruire oggetti java derivanti dagli elementi del database.
Più semplicemente, ogni elemento di una tabella nel database ha una diretta corrispondenza con un oggetto di una classe all'interno del codice, in modo da poter manipolare tali elementi in base alle necessità.
I problemi più grossi che creano rallentamenti nel servizio sono legati alle query utilizzate per richiedere i dati al database.
Può infatti capitare che queste vengano testate su campioni troppo piccoli, rischiando poi di dare problemi o "rompersi" quando lavorano sulle grosse moli di dati (fino a circa 200.000 inserimenti di nuove righe ogni ora) con cui il software ha a che fare ogni giorno.
Un altro punto critico di questo sistema è il modo in cui si affronta l'aumentare dei dati di cui esso si deve occupare. Finora questo problema è stato affrontato implementando tre diversi approcci di partizionamento:

* Partizionamento lato Documenti, dividendo le tabelle più grosse in varie tabelle ordinate con un indice, per facilitarne la gestione;

* Partizionamento lato Application, operato sui database da parte dell'azienda;

* Partizionamento su diversi database paralleli.

Dalle informazioni raccolte emerge quindi come i problemi principali per questo prodotto siano la scalabilità (finora soddisfatta sfruttando vari sistemi di partizionamento delle tabelle) e la rapidità di scrittura e persistenza di nuovi dati.

Figura 3.7: Logo di LegalArchive

3.4.2 InvoiceChannel

InvoiceChannel è il secondo prodotto per importanza all'interno dell'azienda, ed è strettamente legato a LegalArchive. Si tratta di un software che fa da ponte tra le aziende ed il Sistema di Interscambio, un servizio dell'Agenzia delle Entrate che gestisce la fatturazione elettronica.
Si presenta come una piattaforma tramite cui inserire dati e documenti perché questi vengano inoltrati al SdI ed elaborati come necessario. All'interno di InvoiceChannel è presente anche l'opzione di conservare le fatture inserite assieme con eventuali allegati all'interno di LegalArchive.
Dal lato tecnino, anche per quanto riguarda Invoice Channel vengono sfruttati sia database Oracle che Microsoft.
Sia per quanto riguarda l'ambiente di produzione che negli ambienti dei clienti è presente un'unica istanza, non distribuita nel cloud.
Anche IC si basa su Tomcat e JDBC per la comunicazione tra applicativo e database.
Come per Legal Archive, anche per questo applicativo le preoccupazioni maggiori ricadono su scalabilità e performance in condizioni di utilizzo massivo, date le grosse moli di dati in continuo aumento.
Anche l'ottimizzazione dei tempi di attesa per determinate operazioni time-sensitive è un tema caldo, su cui si sta ancora lavorando.

Figura 3.8: Logo di InvoiceChannel

3.4.3 Dialogo con il Team di Test

Dopo aver consultato i due team che si occupano in modo diretto dello sviluppo e del mantenimento del software, si è ritenuto opportuno avere un confronto ulteriore con il team che all'interno dell'azienda si occupa di effettuare test su tutte le piattaforme, in modo da avere un'idea più concreta sugli strumenti e sulle metodologie da adottare.
L'approccio proposto consisterà nel costruire due database, uno relazionale e uno secondo il paradigma NoSQL, per simulare l'ambiente dei prodotti di Ifin ed effettuare una serie di query volte al raccoglimento di dati che rappresentino le prestazioni di tali database.
In questo modo sarà possibile determinare, dati alla mano, quali possono essere vantaggi e svantaggi delle tecnologie coinvolte.

3.5 Conclusione della fase di Indagine

Si evince dall'analisi eseguita e dai dialoghi avuti con i vari team leader che lo stack tecnologico utilizzato in Ifin è ben consolidato e strettamente legato ai processi. Per questo motivo, l'idea di sostituire parti di esso rappresenta una sfida non indifferente e necessita di una valutazione attenta di vantaggi e svantaggi, poiché apportare modifiche di questa taglia e importanza richiederebbe uno sforzo immane.
Per quanto riguarda i problemi che le tecnologie in questione sono costrette ad affrontare quotidianamente, emerge una comune preoccupazione per scalabilità dei sistemi e performance delle operazioni in rapidità ed affidabilità.
Il fatto che questi siano i punti forti di molte soluzioni NoSQL giustifica lo studio di fattibilità che si sta eseguendo, nonostante l'effettiva problematicità di una eventuale ristrutturazione dei principali prodotti di Ifin.

Dato quanto visto nei capitoli precedenti, si è scelto MongoDB come database NoSQL da adottare per effettuare il confronto con i database relazionali. Oltre a proporre soluzioni alternative ai problemi principali in cui tali database incorrono, Mongo è estremamente popolare, ben documentato, ed è già stato approcciato all'interno di Ifin per altri progetti al di fuori dei suoi prodotti principali. Per questi motivi è sembrato una buona scelta ed un ottimo rappresentante del mondo NoSQL.

Capitolo 4

Metodologie

4.1 Strumenti e Tecnologie utilizzate

Per poter eseguire il confronto tra le prestazioni di database relazionali e NoSQL è stato necessario preparare tutta una serie di strumenti utili a far funzionare i database stessi e al monitoraggio dei dati contenuti al loro interno.

4.1.1 Docker

Per poter effettuare test consistenti, evitare problemi di installazione e compartimentalizzare l'ambiente di lavoro, si è deciso di sfruttare Docker. Questo software permette di virtualizzare l'esecuzione di altri applicativi in ambienti chiusi e controllati, facilitando determinati processi di sviluppo. L'utilizzo di Docker in questo progetto lo rende più maneggevole e lineare.
Per sfruttare questo strumento è necessario installare l'engine di Docker che ci permette di gestire i vari container che andremo a creare.
Nello specifico, un container è una istanziazione di un'immagine, che a sua volta è uno snapshot del software che vogliamo "dockerizzare".
Docker ci permette di creare più container che funzionano parallelamente e indipendentemente gli uni dagli altri. Possono tuttavia comunicare utilizzando delle porte specificate nella fase di creazione dell'immagine.

Nel caso di questo progetto, docker risulta utile innanzitutto per gestire il funzionamento dei due DB (che saranno PostgreSQL e MongoDB), inserendoli entrambi in un rispettivo container.
Per facilitare poi i processi di comunicazione con i DB verranno utilizzati altri due container contenenti delle interfacce grafiche (rfispettivamente pgAdmin e MongoExpress).

Figura 4.1: Logo di Docker

4.1.2 PostgreSQL e tecnologie annesse

Dopo aver scelto Docker come ambiente in cui far girare i database, alcune altre scelte legate alle tecnologie coinvolte in questa ricerca sono state prese come conseguenza. Sebbene infatti sia possibile "dockerizzare" moltissimi software, agli scopi di questa tesi è risultato più comodo partire da immagini pre-esistenti ed affidabili, disponibili sulla piattaforma ufficiale del software (hub-docker).
Questo ha permesso di non investre troppo tempo nella preparazione degli strumenti e di concentrare le risorse disponibili nell'effettivo confronto tra database.

Per tutti questi motivi si è quindi scelto di usare PostgreSQL come database relazionale da portare a confronto con MongoDB.
PostreSQL è un database relazionale open source molto robusto che si presta bene per rappresentare le prestazioni di un generico database di questa categoria. Esso è inoltre presente nell'hub di docker con un'immagine ufficiale, ed è quindi facile da far funzionare all'interno di un container a differenza di altri prodotti simili.
Sebbene PostgreSQL non sia utilizzato massivamente all'interno dell'azienda, risulta comunque molto simile alle sue controparti non open source, MSSQL e Oracle Server, che sono invece largamente uaste all'interno di Ifin e sarebbero oggetto di una potenziale migrazione qualora i risultati di questa tesi si rivelassero convincenti.

Vale la pena di evidenziare che a differenza delle tecnologie che ricadono sotto la sigla "NoSQL", quelle legate ai database relazionali sono molto più simili tra loro, poichè condividono appunto il paradigma relazionale che sta alla base di tutte le variazioni esistenti proposte da aziende diverse in contesti diversi.
In questo senso, usare PostgreSQL non è una scelta incoerente con i motivi che hanno dato alla luce questo progetto.

Figura 4.2: Logo di PostgreSQL

4.1.3 MongoDB e tecnologie annesse

4.2 Selezione di un ambito di confronto

4.3 Modellazione del database in MongoDB

Capitolo 5

Progettazione e codifica

Breve introduzione al capitolo

5.1 Tecnologie e strumenti

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

Tecnologia 1

Descrizione Tecnologia 1.

Tecnologia 2

Descrizione Tecnologia 2

5.2 Ciclo di vita del software

5.3 Progettazione

Namespace 1

Descrizione namespace 1.

Classe 1: Descrizione classe 1

Classe 2: Descrizione classe 2

5.4 Design Pattern utilizzati

5.5 Codifica

Capitolo 6

Verifica e validazione

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia

Riferimenti bibliografici

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

Siti web consultati

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