# ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

**TITOLO DELL’ELABORATO**

*Elaborato in*

BASI DI DATI AVANZATE

*Relatore* *Presentata da*

Prof. Alessandra Lumini Francesco Foschini

Anno Accademico 2018 - 2020

**Indice**

[ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA CAMPUS DI CESENA 1](#_Toc57553977)

[Introduzione 3](#_Toc57553978)

[Azienda 3](#_Toc57553979)

[Problema Affrontato 3](#_Toc57553980)

[Auditing e notifiche di eventi 4](#_Toc57553981)

[Specifiche funzionali e architetturali 4](#_Toc57553982)

[Altri sistemi esistenti 5](#_Toc57553983)

[Librerie e tecnologie 5](#_Toc57553984)

[I Trigger 5](#_Toc57553985)

[Tecniche di auding 5](#_Toc57553986)

[Scelte architetturali 6](#_Toc57553987)

[Software 6](#_Toc57553988)

[Librerie 7](#_Toc57553989)

[Metodologia di Progettazione 8](#_Toc57553990)

[Implementazione Server 9](#_Toc57553991)

[Sharding e Replication 9](#_Toc57553992)

[Test e Performance 9](#_Toc57553993)

[Design 9](#_Toc57553994)

[Aspetti statici 9](#_Toc57553995)

[Base Dati 9](#_Toc57553996)

[Casi d’uso 9](#_Toc57553997)

[Design applicativo 9](#_Toc57553998)

[Aspetti dinamici 10](#_Toc57553999)

[Diagrammi di attività 10](#_Toc57554000)

[Conclusioni 10](#_Toc57554001)

[Sviluppi futuri 10](#_Toc57554002)

[Bibliografia 11](#_Toc57554003)

**Capitolo 1**

# Introduzione

## Azienda

Il tirocinio è stato svolto presso l’azienda Energy Software a Faenza. EnergySoftware s.r.l è un’azienda che nasce nel 2015 da un team di persone con esperienza decennale con l’obiettivo di fornire tutta una serie di servizi legati al mondo dell’informatica tra cui sviluppo di software per i mercati energetici, sviluppo

di applicativi per specifiche esigenze del cliente, consulenza sistematica

specializzata su server e networking.

## Problema Affrontato

Il lavoro prevede il design e la realizzazione di un sistema di notifiche che permetta di distribuire eventi generati su software di beckend e di frontend in modo da fornire informazioni costantemente aggiornate agli utenti delle applicazioni web.

Per il design è stato necessario acquisire una solida conoscenza dei sistemi distribuiti, dei problemi di concorrenza sui dati condivisi e dell’interfacciamento con database realtime.

La libreria è stata prototipata in C# su piattaforma .NET Core e il codice prodotto è stato organizzato in maniera efficiente e riutilizzabile nei prodotti della società.

Dal punto di vista del database, abbiamo studiato lo schema migliore di rappresentazione dei dati per ottenere le migliori prestazioni possibili a fronte delle necessarie caratteristiche di consistenza ed isolamento.

I risultati ottenuti sono stati oggetto di misurazione e presentazione.

## Auditing e notifiche di eventi

blablablablablablablablablablablablablablablablabla

## Specifiche funzionali e architetturali

Descrizione delle funzionalità (funzionali) + tipo di architettura richiesta (client server, interfaccia web….)

Vedi ad esempio in questo capitolato cosa si intende per requisiti architetturali (<https://www.lilt.it/sites/default/files/allegati/amm-trasparente/2018-11/capitolato-speciale-all.1-specifiche-tecniche.pdf>)

**Capitolo 2**

# Altri sistemi esistenti

## Librerie e tecnologie

blablablablablablablablablablablablablablablablablabla

## I Trigger

I sistemi relazionali blablablablablablablablablablablablablablablablablabla

## Tecniche di auding

Vedi database auding wikipedia :

(<https://www.lilt.it/sites/default/files/allegati/amm-trasparente/2018-11/capitolato-speciale-all.1-specifiche-tecniche.pdf>)

(non andare troppo nello specifico sulle altre tecnologie)

**Capitolo 3**

# Scelte architetturali

## Software

Inizialmente, ho ricercato il miglior software utilizzabile per l’applicativo.

Il progetto è stato implementato sull’ambiente di sviluppo Visual Studio.

Abbiamo pensato, per le esigenze del progetto, di utilizzare RethinkDb, un dbms non relazionale di tipo documentale.

(cosa significa db documentale -> le tabelle non seguono uno schema, i documenti di una stessa tabella possono avere campi diversi)

Per interrogare un server RethinkDb viene utilizzato il linguaggio ReQl.

Il Reql è un linguaggio di interrogazione molto simile a SQL classico.

Permette, infatti, di eseguire operazioni di “join” tra tabelle di uno stesso database.

E’ inoltre possibile generare una Pipeline di operazioni concatenandole una dopo l’altra separate da un punto.

(Mettere es Query Reql: )

Tra i software largamente utilizzati troviamo, inoltre, Git e Docker Desktop.

Git è stato fondamentale alla gestione del repository di progetto.

Docker Desktop, invece, è stato utile alla creazione e mantenimento del server Rethinkdb.

(Docker container, cosa sono ?)

## Librerie

Tramite lo store NuGet di Visual Studio è possibile scaricare diverse librerie di supporto ai propri progetti.

Ho utilizzato le Librerie bchavez/ RethinkDb.Driver, Reactive Extensions (Rx) e Simple Injector.

“bchavez/ RethinkDb.Driver” è un driver che fornisce un’api in .Net, utile per l’interfacciamento al server Rethinkdb.

Pur non essendo un driver “ufficiale”, è molto simile a quello ufficiale scritto in java.

Per implementare il manager delle notifiche è stata utilizzata la libreria Reactive Extensions (Rx).

Reactive Extensions (Rx) è una libreria per la composizione di programmi asincroni e basati su eventi utilizzando sequenze osservabili e operatori di query in stile LINQ.

Simple Injector, infine, sarà utilizzata dai sofware di frontend e beckend aziendali per gestire e richiamare le funzionalità dell’ applicativo.

Grazie ad essa, è infatti possibile registrare un’istanza di una classe e fare in modo che quella restituita sia sempre la stessa al momento della registrazione.

Attraverso questo meccanismo di Simple Injector è stato possibile implementare automaticamente il Pattern Singleton sulla classe principale della libreria (UtilityRethink).

## Metodologia di Progettazione

In seguito alla scelta del software, è stato scelto Scrum come metodologia di progettazione agile.

Esso permette la divisione del progetto in Sprint, ovvero intervalli di tempo di durata fissa generalmente da uno a quattro settimane.

Gli Sprint sono l’unità di base dello sviluppo in Scrum.

Ognuno di essi è preceduto da una riunione di pianificazione in cui vengono identificati gli obiettivi e vengono stimati i tempi.

Durante uno sprint non è permesso cambiare gli obiettivi, quindi le modifiche sono sospese fino alla successiva riunione di pianificazione, e potranno essere prese in considerazione nel successivo Sprint.

Durante lo sviluppo dell’applicazione ogni attività svolta ha assunto i seguenti stati: “ToDo”, “Doing” e “Done”.

Scrum è una metodologia di progettazione che si utilizza quando i requisiti sono ben chiari e una volta iniziato il progetto non ci sono più cose nuove che entrano nella lista “ToDo”.

A tal proposito, per la gestione del progetto ho utilizzato Jira Software attraverso credenziali datemi dall’azienda.

Quando iniziavo una nuova attività (Sprint) essa passava dallo stato “ToDo” al “Doing” per poi andare in “Done” quando veniva completato.

## 

## Implementazione Server

Per poter testare le capacità del dbms RethinkDb abbiamo pensato di affidarci alla tecnologia offerta dai container docker.

Attraverso il sito Docker-hub, sito web su cui è possibile cercare scaricare immagini docker di applicativi, abbiamo scelto di utilizzare l’immagine ufficiale “rethinkdb” mantenuta dalla società RethinkDb.

In seguito al download dell’immagine, utilizzando su terminale i comandi di Docker-compose sono riuscito a costruire tre diversi tipi di server Rethink.

Server a singolo nodo

(foto terminale e interfaccia rethink)

Server a due nodi

(foto terminale e interfaccia rethink)

Server a cinque nodi

(foto terminale e interfaccia rethink)

Ogni nodo dei server rappresenta di fatto un container.

Nei casi del server a due e cinque nodi, attraverso lo script bash “start-rethinkdb-cluster.sh” è stato possibile eseguire un join dei nodi-container in unico server.

Attraverso questa tecnica, i nodi del server collaborano tra loro per rispondere alle richieste in arrivo e anche per poter sfruttare le politiche di Replication e di Sharding.

### Sharding e Replication

Due aspetti che bisogna considerare, durante lo sviluppo di un cluster, sono le politiche di Sharding e Replication sui dati.

In particolare, le politiche di Sharding servono a distribuire i dati al meglio sul Cluster.

I dati vengono inizialmente suddivisi in piccole parti, dette shards, per essere successivamente inserite al meglio nei diversi nodi del server.

Una buona strategia di sharding è fondamentale per ottimizzare le performance.

Le diverse partizioni dei dati, inseguito ad una richiesta, potranno essere lette in parallelo dai diversi nodi.

(Strategia di sharding mio caso:

vedi paragrafo **Sharding e replica tramite ReQL:**

<https://rethinkdb.com/docs/sharding-and-replication/> li dice che usa politica di hash per lo sharding -> non ho controllo sulla partizione)

La Replication, invece, serve alla creazione e gestione di copie dei dati sui nodi del Cluster.

In caso di “node failure” è una tecnica che previene la perdita dei dati nel caso siano stati precedentemente salvati anche su un secondo nodo.

Attraverso la Replication, inoltre, un dato può essere letto in parallelo da diversi utenti siccome è presente in più nodi del Cluster.

Tramite la replicazione, è inoltre possibile, che la lettura di un dato avvenga da un nodo del server vicino geograficamente a dove è stato richiesto.

Se esistesse una sola copia di un dato ci sarebbe, infatti, altà probabilità che risieda su un nodo particolarmente lontano rispetto a dove è stato richiesto.

Ci sarebbero, di conseguenza, alti costi di rete.

(Strategia di Replication da me usata)

Gestione con Interfaccia Web (mettere livello 4)

RethinkDB consente di suddividere e replicare il cluster in base ad una tabella.

Le impostazioni possono essere controllate facilmente dalla console di amministrazione web.

Quando si utilizza l'interfaccia utente Web, è sufficiente specificare il numero di frammenti desiderati e, in base ai dati disponibili, è RethinkDB che determinerà i migliori punti di divisione per mantenere i frammenti bilanciati.

Per dividere i propri dati:

Vai alla vista tabella ( *Tabelle* → *nome tabella* ).

Fare clic sul pulsante *Riconfigura* .

Impostare il numero di frammenti e repliche che si desidera.

Fare clic sul pulsante *Applica configurazione*

(Foto Reconfigure tabella Notifications table overview)

Gestione con ReQL (mettere livello 4)

Il linguaggio ReQL consente anche un controllo più dettagliato sulla replica di un dato.

E’ possibile, quindi, oltre all’interfaccia grafica, gestire le repliche utilizzando comandi forniti dal linguaggio ReQL.

Se si avesse la necessità di ottenere configurazioni più avanzate, (es decidere su quale nodo di un Cluster inserire una replica di una tabella), sarebbe necessario rivolgersi a comandi del linguaggio ReQL.

Se si utilizza l’interfaccia grafica vengono infatti autodeterminati i migliori punti di divisione.

Per raggruppare i nodi nei cluster, RethinkDb utilizza i tag di ciascun nodo definiti dall’utente.

A tutti i nodi in un cluster RethinkDB possono essere assegnati zero o più tag:

rethinkdb --server-tag us\_west

I tag possono anche essere utilizzati nelle configurazioni di tabella per mappare le repliche ai server specificati dal tag utilizzando il comando ReQL reconfigure.

Per assegnare 3 repliche della tabella users al nodo us\_west e 2 al nodo us\_east:

r**.**table('users')**.**reconfigure(shards**=**2, replicas**=**{'us\_west':3,

'us\_east':2}, primary\_replica\_tag**=**'us\_east')**.**run(conn)

Nella libreria UtilityRethink (mettere livello 4)

Nella libreria UtilityRethink implementata, un suo sottocomponente (il DbManager) ha anch’esso una funzionalità di reconfigure che permetterà di assegnare il numero di shard e repliche dei dati di una tabella.

Non avendo al momento la necessità di dover assegnare ad uno specifico nodo una determinata replica ho lasciato a RethinkDb il compito di autodeterminarsi la migliore suddivisione dei dati.

### Comandi per la gestione del server

Aprire il terminale e dirigersi sulla cartella di progetto corrispondente all’opzione del server scelta (uno, due o cinque nodi).

In seguito, scrivere sul terminale: “docker-compose -f docker-compose.yml build”.

Questo comando è necessario eseguirlo solamente al primo utilizzo del tipo di server scelto.

In seguito all’esecuzione, infatti, viene costruita e salvata permanentemente sul proprio dispositivo l’immagine docker del server RethinkDb (se necessario è possibile successivamente cancellare l’immagine, vedi documentazione Docker).

Successivamente eseguendo, sempre su terminale, il comando “docker-compose -f docker-compose.yml up -d” il server Rethink è online e i suoi nodi sono in attesa di ricevere richieste.

Per stoppare il server digitare “docker-compose -f docker-compose.yml stop”.

E’ importante sottolineare che in seguito allo stop lo stato dell’immagine docker del server Rethink viene salvato.

Se, ad esempio, inserissi dei nuovi dati sul db e stoppassi successivamente il server, al prossimo comando di up il server mantiene i dati precedentemente caricati.

I tre comandi elencati valgono per tutti e tre i tipi di server (1, 2, 5 nodi).

## 

## Test e Performance

blablablablablablablablablablablablablablablablablabla

**Capitolo 4**

# Design

## Aspetti statici

### Base Dati

blablablablablablablablablablablablablablablablablabla

### Casi d’uso

blablablablablablablablablablablablablablablablablabla

### Design applicativo

Blablablablablablablablablablablablablablablablablabla

## Aspetti dinamici

### Diagrammi di attività

blablablablablablablablablablablablablablablablablabla

**Capitolo 5**

# Conclusioni

## Sviluppi futuri

**Capitolo 6**

# Bibliografia