Sistema per la gestione notifiche RealTime

Tirocinante: Francesco Foschini Tutor aziendale: Fabio Marchesi Tutor didattico: Silvia Mirri Azienda: Energy Software

Periodo di attività Tirocinio Curriculare:

- Primo periodo: 18/02/2020 - 16/03/2020.

- Interruzione causa Covid-19

- Secondo periodo: 13/07/2020 - 28/07/2020

Periodo di attività Tirocinio Per tesi:

28/10/2020 – 30/11/2020

Indice

# 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.

# Progetto: Libreria UtilityRethink

Il lavoro prevede il design e la realizzazione di un sistema di notifiche che permetta di distribuire eventi generati su software di backend 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.

# Tecnologie utilizzate

Software

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

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

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

Tra i software largamente utilizzati ci sono Git e Docker Desktop.

Git è stato fondamentale per la creazione e gestione del repository di progetto.

Docker Desktop, invece, è stato utile alla creazione e gestione del server Rethinkdb a uno/più nodi.

Librerie

Grazie a Visual Studio, nel suo store NuGet, è possibile scaricare e utilizzare librerie di supporto al progetto: bchavez/ RethinkDb.Driver, Reactive Extensions (Rx) e Simple Injector.

Il driver “bchavez/ RethinkDb.Driver” in .Net utile all’interfacciamento sul 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.

Infine, Simple Injector è una libreria che verrà utilizzata sul sofware di beckend e frontend aziendale per gestire e richiamare le funzionalità dell’applicativo.

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

E’ quindi stato possibile, sfruttando la libreria Simple Injector implementare automaticamente il Pattern Singleton sulla libreria Utility Rethink.

# Metodologia di progettazione

Una volta scelto il software e l’ambiente di sviluppo, 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.

# Attività

Il tirocinio ha previsto 5 attività principali: Ricerca, Implementazione, Test e performance, Notifiche, Refactoring per il caso d’uso aziendale.

### Ricerca

Inizialmente mi sono concentrato sulla ricerca e studio di Rethinkdb e Docker. Durante il primo periodo (18/02/2020 - 16/03/2020) non avendo ancora studiato a lezione Docker, ho studiato il significato, la gestione e la creazione di un container Docker.

Ho studiato e compreso come si utilizza Docker-compose per la creazione di un cluster a più nodi RethinkDb

Ho studiato il linguaggio di query ReQl utilizzato per interrogare RethinkDb.

Reql è molto simile a SQL classico, permette infatti di eseguire join tra tabelle ed è inoltre possibile generare una Pipeline di operazioni concatenandole una dopo l’altra separate da un punto.

Ho studiato l’api di Bchavez per connettersi, effettuare query anche complesse di select, insert, delete dall’app .Net al cluster Rethinkdb.

Ho studiato il concetto dei Task asincroni in c# e ho approfondito il pattern observable per un’utilizzo corretto della libreria Reactive Extensions (Rx) per la gestione delle notifiche.

Ho studiato e approfondito i pattern Singleton, Factory Method per poter sfruttare al meglio le funzionalità della libreria Simple Injector.

Ho studiato e approfondito meglio alcune caratteristiche sui generici in c#, in particolare sui vincoli per i parametri di input/output dei metodi di Getter e Insert di notifiche sul db RethinkDb.

Studio della Reflection per capire a RunTime il tipo di una classe (notifica).

### Implementazione server Rethink

Una volta completato l’attività di ricerca ho provveduto all’implementazione di un iniziale prototipo dell’applicativo in .Net sia dei file utili alla gestione del cluster RethinkDb.

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

Scaricata l’immagine attraverso i comandi di Docker-compose sono riuscito a costruire tre diversi tipi di server Rethink: singolo nodo, 2 nodi, 5 nodi.

Ogni nodo rappresenta di fatto un container.

Nei casi in cui il server è composto da 2 e 5 nodi grazie ad uno “script bash .sh” sono riuscito a “joinare” i nodi/container in unico server in modo che collaborino e possano essere sfruttate le politiche di Replication e di Sharding.

Andare, via terminale, sulla cartella corrispondente all’opzione del server scelta e costruire l’immagine del cluster attraverso il comando: “docker-compose -f docker-compose.yml build”.

Questo passaggio è necessario solamente al primo utilizzo.

Una volta costruita l’immagine, infatti, viene salvata permanentemente sul proprio dispositivo (se necessario è possibile cancellare l’immagine, vedi documentazione Docker).

Successivamente tramite “docker-compose -f docker-compose.yml up -d” il cluster è online e i suoi nodi sono in attesa di ricevere richieste.

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

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

Se, ad esempio, inserisco dei nuovi dati sul db e stoppo il server, al successivo up il server mantiene i dati precedentemente caricati.

Questi 3 comandi valgono per tutti e tre i casi (1, 2, 5 nodi).

### Test e Performance

Terminata l’attività di implementazione, ho iniziato quindi a verificare il funzionamento dell’applicativo e migliorare progressivamente la qualità del codice tramite rimodellazione delle classi e refactoring.

Il test sulle performance riguarda quindi uno studio del comportamento del dbms in termini di velocità di lettura e scrittura.

Attraverso l'interfaccia utente Web, è possibile gestire la replication e lo sharding dei documenti di una tabella.

E’ sufficiente, infatti, specificare il numero di repliche e il numero di frammenti desiderati e, in base ai dati disponibili, RethinkDB determinerà i migliori punti di divisione per mantenere i frammenti bilanciati.

Siccome nei due casi in cui il Cluster è costituito da più nodi è possibile applicare le politiche di sharding e replication su una singola tabella, ho testato le performance anche tenendo in considerazione questi due aspetti.

Ho eseguito ,quindi, un test che misura le performance del server Rethink in termini di velocità in read e write.

Nella attività “implementazione server Rethink” parallelamente ho progettato un piccolo applicativo finalizzato alla fase di test.

In particolare su un Db “test” nel server Rethink ho creato due tabelle: “Author” e “Post”.

Ho inserito circa 200 documenti su Author e 20 mila su Post.

Gli autori sono caratterizzati da: id, nome, età, hobby. Mentre i post da: id, author\_id, titolo, contenuto.

Da ogni post è possibile ricavare il suo autore attraverso il campo “author\_id” .

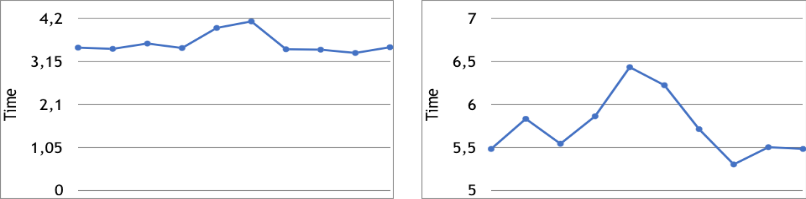
Ho costruito quindi un indice secondario sul campo “author\_id” di “Post” per ottenere le migliori performance possibili.

Nei casi a 1 e 5 nodi del server ho misurato in secondi quanto tempo impiega Rethink ad effettuare 50 inserimenti su Post e una query di selezione basata tra un join tra Author e Post per riuscire a determinare quanti post ha scritto ogni autore sfruttando quindi l’indice sul campo “author\_id” su “Post” precedentemente costruito.

Risultati test performance

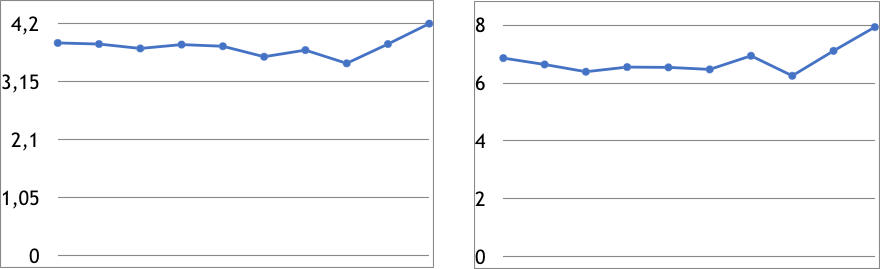
Server Rethink a singolo nodo:

Singolo nodo (non è possibile applicare politiche di sharding e replication):

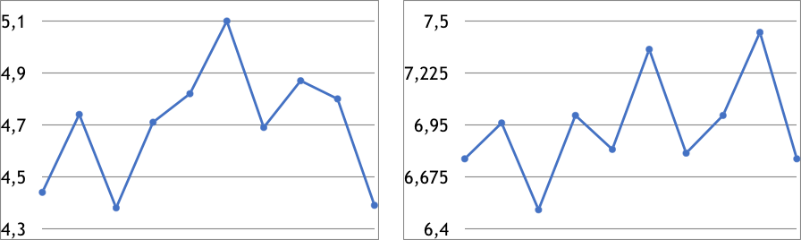
Select (media: 3,577 s) Insert (media: 5,735 s)

Server Rethink a 5 nodi:

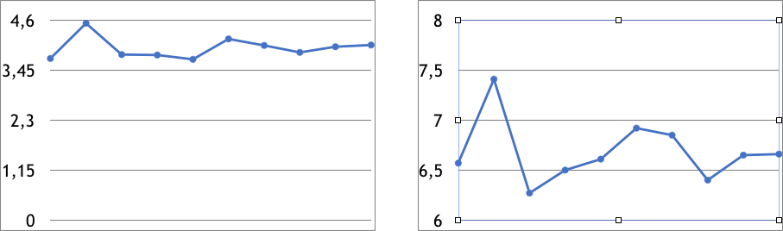
1 Shard con 5 repliche ciascuna:

Select (media: 3.787 s) Insert (media: 6,768 s)

5 Shard con 1 repliche ciascuna:

Select (media: 4,694 s) Insert (media: 6,941 s)

2 Shard con 3 repliche ciascuna:

Select (media: 3,963 s) Insert (media: 6,684 s)

### Notifiche

Una volta testato il dbms ho implementato lato app .Net un sistema in grado di ricevere notifiche quando viene effettuata una modifica ad una tabella (Insert o Update) sul cluster Rethinkdb.

Tramite una funzione chiamata “Changes” del driver di Bchavez è possibile “rimanere in ascolto” alle modifiche effettuate su una tabella.

Sempre per questo scopo, inoltre, ho utilizzato Reactive Extensions (Rx), una libreria per applicazioni in .Net.

Essa permette di generare un observable che rimane in ascolto su una variabile, nel mio caso quella che viene restituita da “Changes”.

Se l’observable trova un messaggio chiama un metodo “On next”, se riceve l’ultimo messaggio chiama “On completed” se invece trova un errore “On error”.

Per simulare gli inserimenti, ho utilizzato TPL (Task Parallel Library) ovvero una libreria di classi .Net per semplificare lo sviluppo di applicazioni concorrenti.

Ho creato quindi 3 Task per simulare 3 insert/update.

Ogni volta che una operazione insert o update viene completata con successo l’observable intercetta l’evento e chiama uno dei 3 metodi a seconda dei casi.

E’ in grado di distinguere una insert da un update, infatti, in quest’ultimo caso ti dice il vecchio e il nuovo valore del documento.

### Refactoring per il caso d’uso aziendale

# Parte finale del tirocinio -> ultime 75 ore del tirocinio per tesi:

# Refactoring del progetto per il caso d’uso aziendale e la creazione della libreria.

# Sui software di frontend e beckend aziendali serviva poter richiamare la stessa istanza della libreria UtilityRethink in diversi punti del codice.

# Ho studiato e approfondito a tal proposito i pattern Singleton e Factory method per un’utilizzo corretto della libreria Simple Injector.

# Essa, infatti, permette di registrare un’istanza di una classe e restituire sempre la stessa.

# Ho sfruttato questa caratteristica della libreria per implementare il pattern Singleton senza effettuare modifiche alla classe principale della libreria “UtilityRethink.cs”.

# SimpleInjector

# 

# Design della Struttura generale UML libreria UtilityRethink

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\StrutturaGeneraleUml.jpg

# (cambiare mettendo le interfacce?)

# UtilityRethink

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\IUtilityRethink.JPG

# E’ la classe fondamentale della libreria.

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\FotoCodice\UtilityRethinkCodice.JPG

# Una volta istanziato crea una nuova connessione con un db sul Cluster Rethink (se non esiste viene creato) e su di esso crea (sempre se non esiste gia quel db) tutte le tabelle di “Sistema”.

# UtilityRethink consente all’utente di interagire con il db richiesto sul cluster attraverso i due manager dell’applicativo: DbManager e INotificationsManager

# In questo caso, nel nostro caso d’uso, l’unica tabella di sistema è “Notifications” perché abbiamo solamente il manager delle Notifiche “INotificationsManager”.

# L’oggetto “connessione” creato viene inseguito “passato” a tutti i sotto-componenti, ovvero i manager, per poter essere utilizzato.

# DbManager

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\DbManager.JPG

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\FotoCodice\DbManagerCodice.JPG

# E’ a tutti gli effetti il manager del Db, possibile creare e eliminare tabelle indici.

# Attenzione: Non è possibile eliminare le tabelle di sistema (es “Notification”).

# NotificationsManager

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\NotificationManager.JPG

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\FotoCodice\NotificationsManager.JPG

# E’ il manager delle notifiche.

# Essendo il manager delle notifiche la sua tabella di sistema è “Notification”.

# Se in futuro si avrà la necessità di avere altri Manager per altre tabelle si potrà gestire in maniera semplice grazie a questo design andando ad aggiungerli a Utility Rethink.

# Ha due sotto componenti sempre relativi alle notifiche: IQueryNotification e IRXNotifier.

# IQueryNotification:

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\QueryNotifications.JPG

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\FotoCodice\QueryNotificationCodice.JPG

# Permette di effettuare le “interrogazioni” alla tabella di sistema “Notifications”.

# operazioni CRUD.

# Inserimento di nuove notifiche, cancellare notifiche con un certo id e richiedere Notifiche in base a determinati valori dei suoi campi.

# IRXNotifier

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\RXNotifier.JPG

# RXNotifier è un notificatore di eventi, sulla tabella di sistema di INotification, quindi su “Notifications”.

# Grazie agli studi svolti durante l’attività “Notifiche”, allo studio del pattern observer e alla combinazione delle librerie “bchavez/RethinkDb.Driver” e Reactive Extensions (Rx) è stato possibile implementare il notificatore.

# Il notificatore rimane in ascolto sulla tabella “Notifications” e, in particolare, può anche intercettare eventi che riguardano solamente notifiche che abbiano certi valori di argomento (campo “arg”).

# Gli eventi possono essere inserimento, cancellazione, modifica di una notifica che abbia come argomento uno tra quelli presi in considerazione.

# In seguito alla “richiesta di registrazione” viene restituito un NotificationSubscription, oggetto che raccoglie l’observable con un suo id.

# Ora, passaggio fondamentale, chi utilizza la libreria dovrà sottoscrivere l’observable a dei metodi implementati in base alle proprie esigenze.

# Per esempio come mostrato in figura, in caso di intercettazione corretta di un evento verrà chiamato il metodo onNext su cui è possibile manipolare la nuova Notifica oppure la vecchia (in caso di update).

# In caso di errore verrà chiamato il metodo OnError mentre al termine dell’ascolto (se verrà chiamato il metodo StopListening sull’observable) verrà invocato il metodo OnComplete.

# I tre metodi, OnNext, OnError e OnComplete verrano in questo modo implementati lato cliente ognuno in base alle proprie esigenze.

# Esempio di utilizzo notificatore sulle notifiche di esecuzione:

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\ObservableCodice.JPG

# (x ha in se lo stato dell’evento catturato: nuovo Dato, vecchio Dato, ecc …)

# Per le notifiche di nuovo dato (“NotificationNewData”) basta sostuire dentro le “<>” “NotificationNewData” al posto di “NotificationExec”.

# Esempio di OnNext, OnError e OnComplete

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\FotoCodice\OnNextOnErrorOnCompleteCodice.JPG

# Connection

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\Connection.JPG

# Oggetto Connection viene istanziato all’inizio dall’UtilityRethink per poi passarlo a tutti i sotto-componenti dell’applicativo. Tutti i sotto-componenti per interfacciarsi al Cluster Rethink utilizzano quindi lo stesso oggetto Connection.

# Nello specifico il campo conn che rappresenta la connessione vera e propria viene istanziata solamente alla prima interazione col db, successivamente non viene più istanziato ma viene sempre restituito lo stesso oggetto perché altrimenti sarebbe un’operazione pesante.

# ConnectionNodes accetta una lista di DbOptions.

# Nel nostro caso, in particolare, possono essere 1, 2 o 5, in base al numero dei nodi del Cluster su cui connettersi.

# La connessione viene eseguita tramite un’algoritmo di RoundRobin che sceglie a run-time il nodo disponibile al momento.

# Se ci si connette su indirizzi ip non raggiungibili dopo 20 secondi viene fatto scattare il Timeout per la mancata connessione.

# Le Notifiche

# C:\Users\Utente\Desktop\francesco\Università\Tesi\rethinkdb\Uml\Notification.JPG

# Nel caso d’uso aziendale esistono due tipi di notifiche: “notifiche di nuovo dato” indicate nell’applicazione come “NotificationNewData.cs” e “notifiche di esecuzione” indicate come “NotificationExec.cs”.

# Entrambe ereditano da Notification.cs , classe astratta che ha i campi base delle notifiche, ovvero: Id, Date, Arg, Text, Type.

# “NotificationNewData” possiede quindi tutti i campi di “Notification” ma ha in aggiunta un suo campo “Table”.

# “Table” servirà a identificare la tabella sul db reale aziendale in cui è stata fatta una insert/update di un nuovo dato e di conseguenza andare ad aggiornare anche la “cache del sistema aziendale”.

# Quest’ultima viene molto utilizzata dall’azienda per effettuare ricerche di Get siccome interrogare direttamente il “db vero” continuamente è costoso in termini di performance.

# Le NotificationExec serviranno invece a notificare i servizi software di frontend e beckend aziendali sullo stato di avanzamento di un Task/processo.

# Hanno in aggiunta un campo idExec che rappresenta l’id del Task.

# Dal punto di vista del db sul cluster Rethinkdb, sarà la tabella di sistema “Notifications” a raccogliere e archiviare tutte le notifiche generate dai sistemi aziendali.

# Rethinkdb essendo un db di tipo non relazionale accetta la diversità di campi tra documenti di una stessa tabella.

# Ho sfruttato quindi questa caratteristica per raccogliere nella tabella “Notifications” sia le notifiche di esecuzione (“NotificationExec.cs”) sia quelle di nuovo dato (“NotificationNewData.cs”).

# Grazie al design dell’applicazione (utilizzo dei generici), inoltre se in futuro sarà necessario aggiungere qualche altro tipo di notifica non sarà un problema.

# Le classi IQueryNotification e IRXNotifier, infatti, lavorano con ogni tipo di notifica a patto che la classe della nuova notifica erediti da “Notification” (where T: Notification).

# In particolare, per implementare questa soluzione, ho utilizzato la proprietà della Reflection per riuscire a determinare a run-time il tipo della notifica (campo “type” di “Notification).

# Problemi irrisolti / Possibili modifiche future

# -Getter di notifiche sul db Rethink per ora non sfruttano gli indici secondari.

# In caso ci fosse l’indice su un campo aggiungere la possibilità di sfruttarlo.

# -Dare la possibilità all’utente di poter impostare il timeout della connessione

# -Rendere la libreria non dipendente dalla tecnologia utilizzata.

# Esempio: se un giorno si volesse sostituire RethinkDb con RabbitMq.

# Se un giorno RethinkDb diventasse a pagamento.

# Interfaccia principale: INotificationProvider (non avrà il DbManager -> RabbitMq non è un db) viene implementata da UtilityRethink (classe specifica per Utility Rethink)

# -IQueryNotifications essendo una classe che esegue solamente operazioni CRUD rinominarla in INotificationRepository. (oggetto repository pattern di progettazione).

# -Tutto troppo “Java like”, togliere i metodi getter() e mettere i campi con “{ get; }” seguendo le convenzioni di .Net

# -Sostituire nel nome dei metodi “with” con “By” o “On”

# -L’idExec delle Notifiche di esecuzione essendo l’id di un Task/processo esterno dalla libreria non deve essere di tipo Guid.

# Scegliere tra: generico, string o object.

# Conclusioni

Questa esperienza l’ho trovata molto formativa.

Stando a stretto contatto con i dipendenti dell’azienda sono riuscito ad apprendere nuovi concetti anche pratici dell’informatica.

Ringraziamenti

Ringrazio infine l’azienda Energy Software per la sua disponibilità e accoglienza che mi ha riservato.

# Bibliografia

Https://docs.microsoft.com/it-it/dotnet/csharp/programming-guide/concepts/async/

Https:[//www.pluralsight.com/guides/build-a-scalable-fault-tolerant-system-with-asp-](http://www.pluralsight.com/guides/build-a-scalable-fault-tolerant-system-with-asp-) net-core-and-rethinkdb-on-docker-swarm-mode

Https://github.com/bchavez/RethinkDb.Driver/wiki/Extra-C%23-Driver-

Features#consuming-changefeeds https://rethinkdb.com/faq/#when-is-rethinkdb-not-a-good-choice https://rethinkdb.com/docs/rethinkdb-vs-mongodb/ https://rethinkdb.com/blog/rethinkdb-screencast https://rethinkdb.com/docs/sharding-and-replication/ https://rethinkdb.com/docs/guide/javascript/

https:[//www.ionos.it/digitalguide/server/know-how/orchestrazione-di-docker-con-](http://www.ionos.it/digitalguide/server/know-how/orchestrazione-di-docker-con-) swarm-e-compose/

<https://github.com/osirisguitar/rethinkdb-cluster-docker>

AGGIUNGERE:

-SITO SIMPLE-INJECTOR

-SITO MICROSOFT GENERICI C#

-SITO MICROSOFT REFLECTION