UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Magistrale in

Scienze e Tecnologie dell'Informazione

Mole.io

un sistema per la gestione centralizzata dei log applicativi

RELATORE

Prof. Ernesto DAMIANI

TESI DI LAUREA DI

Federico GANDELLINI

CORRELATORE

Matr. 123456

Prof. Nome COGNOME

Anno Accademico 2013/2014

Ringraziamenti

Un grazie a ...

Indice

Introduzione

In questa tesi descriveremo Mole.io: un sistema centralizzato per la raccolta e l'aggregazione di messaggi provenienti da applicazioni remote.

Durante il loro ciclo di lavoro o *processing*, le applicazioni software eseguono operazioni significative o entrano in situazioni di errore, in questi casi è importante che le persone cha hanno in carico la gestioni di questi sistemi, siano informate dell'accaduto in modo da operare scelte opportune o applicare le dovute correzioni (*bugfix*).

Gli sviluppatori spesso utilizzano messaggi di tracciamento (*log*) per stampare a video o salvare in *files* stati significativi delle applicazioni. Gli stessi *log* sono utilizzati più spesso per riportare situazioni di errore (*Exception* e *Stack Trace*).

Il problema principale di questo approccio è la *località* dei *log*, solitamente questi *files* vengono salvati, nella stessa macchina sulla quale sta operando l'applicazione.

All'aumentare del numero di applicazioni da gestire e del numero di macchine in produzione, capita spesso che i server siano in luoghi geograficamente distanti tra loro. Questa situazione rende evidente la difficoltà di ottenere un feedback veloce dello stato di ogni software e delle eventuali situazioni di errore in cui le applicazioni si trovano.

Mole.io cerca di risolvere il problema facendo in modo che i software

che lo utilizzano, siano in grado di inviare le informazioni che ritengono significative ad un server centrale, che le raccoglie, le cataloga e le aggrega per essere facilmente supervisionate da parte degli sviluppatori.

Nel primo capitolo tratteremo approfonditamente il problema dei *log*, i contesti nei quali essi vengono utilizzati e le problematiche legate alla gestione di questo tipo di soluzione di tracciamento. Vedremo anche come utilizzare i *log* per ottenere informazioni di supporto alla *business intelligence*.

Il secondo capitolo riporterà un elenco dei principali *software* per la gestione centralizzata dei *log* presenti sul mercato e delle soluzioni *Open Source* che sono state prese a modello per la realizzazione di Mole.io. Descriveremo ogni applicazione e mostreremo come Mole.io possa essere una soluzione innovativa sotto svariati punti di vista.

I due capitoli seguenti permetteranno di approfondire i dettagli tecnici delle metodologie di sviluppo applicate durante il *design* del software e alcune tra le principali tecnologie utilizzate per la realizzazione del sistema.

Il quinto capitolo descriverà la struttura di Mole.io e le varie componenti software che rendono l'applicazione scalabile, sicura e garantiscono l'alta affidabilità della soluzione. Uno spazio particolare sarà inoltre riservato alle problematiche incontrate durante lo sviluppo.

Nel sesto capitolo vedremo in modo oggettivo, con *benchmark* e *stress test* il comportamento di Mole.io all'aumentare del carico di lavoro e dimostreremo come le soluzioni di design applicate garantiscano buone *performance* anche in condizioni critiche.

Infine discuteremo i risultati ottenuti e proporremo alcune interessanti funzionalità che trasformeranno Mole.io dall'attuale *proof of concept* ad un vero e proprio servizio.

Capitolo 1

Log:
contesti e problematiche

Iniziamo riportando alcune definizioni che utilizzeremo spesso nel seguito della tesi. Diciamo *processo* un programma in esecuzione e *log* l'insieme dei messaggi prodotti, a fini informativi, da tale processo.

La decisione di quali e quante informazioni salvare, spetta tipicamente allo sviluppatore o al personale addetto alla gestione dell'applicazione.

I messaggi di log posso essere di vario tipo, è usanza comune caratterizzare ogni messaggio con un livello di gravità (*severity*) permettendo così una identificazione più rapida degli errori più gravi, rendendo repentino l'intervento di riparazione dell'applicazione.

Informazioni spesso salvate all'interno dei log sono dati specifici del sistema nel quale l'applicazione è in esecuzione, dati relativi all'utente che la sta utilizzando, oppure relativi allo stato del sistema in un preciso istante temporale. Ogni log è infine corredato da un messaggio significativo che lo rende immediatamente identificabile tra altri.

1.1 Trattare gli errori applicativi

Il salvataggio dei log, come abbiamo anticipato, è una operazione molto comune nei software, ma diventa fondamentale quando si vuole monitorare lo stato interno di una applicazione in esecuzione e si vuole essere informati riguardo alle situazioni di errore nelle quali quest'ultima incorre.

Salvare le informazioni relative alle situazioni di errore è importante per gli sviluppatori, questo permette, infatti di velocizzare l'individuazione di errori (*bug*) nel flusso di lavoro del programma e, di conseguenza la loro risoluzione (*bugfix*).

Il salvataggio dei log avviene tipicamente su uno o più *files* presenti nella stessa macchina nella quale sta funzionando l'applicazione. A seconda del tempo di esecuzione di una applicazione e della frequenza con la quale essa produce messaggi di log, i files sui quali vengono salvate le informazioni possono diventare molto grandi. Un file di questo tipo è molto complesso da gestire da parte degli addetti ai lavori, infatti diventa lungo e complesso trovare informazioni significative all'interno di esso e soprattutto diventa complesso correlare le situazioni di errore e capire con quale frequenza o in quali condizioni si presenta un particolare malfunzionamento.

problema località: se ho molte applicazioni e molti clienti devo tener monitorata (da remoto) la situazione dei log per ogni cliente/app/macchina, verificare che non esplodano i files e quando diventano troppo grandi, archiviare parte di questi.

problema spazio, località difficili da leggere e difficile tirarci fuori delle info significative

1.2 La centralizzazione

tante applicazioni (anche tipi diversi) che loggano tanti clienti da gestire dislocati sul territorio

veremo che mole usa un db documentale perhcé si presta meglio a salvare dati non fortemente strutturati

1.3 Business intelligence

capire come gli utenti usano il sistema statistiche sul sistema decidere come indirizzare lo sviluppo

Capitolo 2

Log: software e applicazioni

ci sono tante soluzioni sul mercato, di seguito alcune ma non ci piacciono

2.1 Prodotti e soluzioni sul mercato

overview di alcuni sistemi di logging con le relative funzioni specifiche i competitor airbreak - logga solo rollbar - aggrega papertrail - live log

2.2 Una nuova applicazione: Mole.io

perché le soluzioni sul mercato non ci piacciono le peculiarità di mole.io

Capitolo 3

Metodologie di sviluppo

3.1 User stories

3.2 Test e behavior driven development

Capitolo 4

Tecnologie utilizzate

In questo capitolo approfondiremo i dettagli delle tecnologie utilizzate per realizzare Mole.io. Illustreremo, per ognuna di esse, le motivazioni che ci hanno spinto alla scelta di particolari soluzioni software e le problematiche incontrate durante il loro utilizzo.

Mole.io è stato interamente sviluppato utilizzando il linguaggio JavaScript. L'utilizzo di questa tecnologia è abbastanza comune all'interno delle pagine web e si presta bene all'utilizzo *client side*, meno comune è invece la sua applicazione nella parte *server*. La piattaforma Node.js permette di utilizzare JavaScript per sviluppare la parte *backend* delle applicazioni. Rendere uniforme il linguaggio utilizzato permette facilitare lo sviluppo e le fruibilità del progetto da parte degli sviluppatori. Per poter lavorare al progetto è infatti richesta solo la conoscenza di JavaScript e non di altri linguaggi, questo è un ottimo requisito quando si pensa ad un team di sviluppo in espansione.

Quella del paragrafo precedente è solo una delle motivazioni che ci hanno spinto a scegliere Node.js come tecnologia di sviluppo, iniziamo quindi a vedere in dettaglio questa tecnologia.

4.1 Node.js

La *homepage* del sito ufficiale [1] di Node.js fornisce una sintetica ma precisa descrizione di questa tecnologia, partiremo proprio da essa per illustrarne le peculiarità.

La descrizione ufficiale recita:

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Scopriamo immediatamente che Node.js è una *piattaforma*. L'utilizzo di questo termine mette l'accento su un aspetto fondamentale di questa tecnologia: fornire un ambiente nel quale le applicazioni sviluppate possano funzionare con il supporto di librerie di sistema fornite da Node.js stesso.

La piattaforma Node.js utilizza JavaScript come linguaggio di sviluppo. Per farlo si avvale del potente interprete *V8* presente all'interno del browser *Chrome* di *Google*. L'utilizzo di un linguaggio altamente popolare e di una base solida come quella fornita dal popolare *browser* permettono di costruire applicazioni in modo semplice e veloce.

L'ultimo importante concetto, che leggiamo dalla prima frase della descrizione, è che Node.js è principalmente orientato allo sviluppo di applicazioni che lavorano con la rete. Per sua natura, la piattaforma ci aiuta a fare in modo che esse siano scalabili.

La seconda parte della descrizione spiega sinteticamente alcune caratteristiche peculiari di Node.js e ne definisce meglio il contesto applicativo.

L'intera piattaforma è centrata sul concetto di *evento*. Si dice evento un messaggio che viene scatenato in un determinato istante dell'elaborazione e che successivamente è catturato e gestito dalle componenti del sistema che sono preposte alla gestione di quell'evento specifico.

Il sistema ad eventi viene utilizzato da Node.js congiuntamente ad una gestione non bloccante delle operazioni di *Input/Output*. Questo significa che una operazione potenzialmente lunga, come ad esempio la comunicazione con il *filesystem* o con i dispositivi di *rete* non bloccano il flusso di esecuzione del programma principale. Dopo aver richiesto ad altri attori del sistema il dato di cui necessita, il programma continua il suo normale flusso di esecuzione e verrà informato, utilizzando un evento, quando il dato richiesto sarà disponibile. Questa gestione non bloccante delle operazioni di *I/O* è chiamata *I/O* asincrono.

Questa modalità di gestione delle operazioni non strettamente legate alla logica applicativa, permette a Node.js di essere molto efficiente se utilizzato per la realizzazione di applicazioni che elaborano grandi quantità di dati ma devono rimanere *reattive* nei confronti di nuove richieste di elaborazione.

4.1.1 L'I/O asincrono

Abbiamo introdotto il concetto di I/O asincrono, di seguito illustreremo come Node.js riesca a gestirlo utilizzando una quantità limitata di risorse di sistema.

La gestione delle operazioni di scrittura su filesystem è

Questa gestione non bloccante delle operazioni di *I/O* è chiamata *I/O* asincrono, perché il flusso di esecuzione del programma non è lineare, ma è soggetto a continue biforcazioni e ricongiunzioni dovute alla gestione delle operazioni costose in termini di tempo.

Il vantaggio di una gestione di questo tipo è che la logica del programma non viene influenzata dalle operazioni esterne.

Questo è un aspetto molto controverso e discusso, infatti se è abituati a pensare tale linguaggio come un giocattolo, troppo instabile e imprevedibile per

4.1.2 La storia

4.1.3 NPM e moduli

4.2 RabbitMQ

4.3 MongoDB

4.3.1 Fronteggiare le richieste

4.4 AngularJS e altre tecnologie di frontend

4.4.1 Gestione delle dipendenze

4.5 Strumenti per il deploy

Capitolo 5

Mole.io

5.1 Architettura del sistema

5.1.1 CQRS ed estensibilità

5.1.2 mole

I denormalizzatori

5.1.3 mole-suit

I plugin e gli widget

5.2 Autenticazione degli utenti

5.3 Scalabilità e affidabilità

5.4 Problematiche di sviluppo

Capitolo 6

Configurazioni e benchmark

problemi con i benchmark - dipendi dalla rete su cui sei - nostro client fatto con node - perché non l'abbiamo usato - come sono stati fatti i benchmark - specifiche del sistema VM, ram, hdd, ... - risultati ottenuti

Conclusioni e sviluppi futuri

Bibliografia

[1] Inc Joyent. node.js, 2014.