UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Ingegneria Elettrica, Elettronica e Informatica

Corso di Laurea Magistrale in Ingegneria Informatica

Progettazione di Sistemi Distribuiti (Ing. Antonella Di Stefano)

Alessandro Sangiorgi-Salvatore Mulè

Homework 3

Anno Accademico 2017/2018

Home Work 3

SCANNER#1: programma Java che analizza ogni nuova riga di uno specifico log file alla ricerca di specifiche keywords o espressioni regolari (da file di conf/properties).
Per ogni match, ID macchina, timestamp e riga sono pubblicati sul P/S Queue.

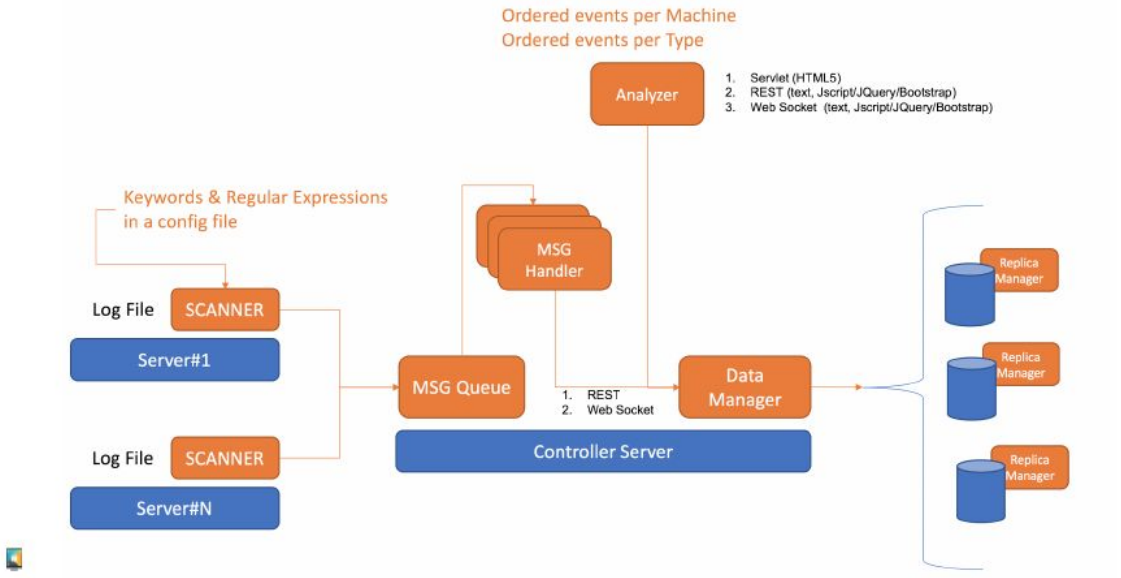
MSG QUEUE: riceve i dati (asynch) dagli SCANNER e li invia ai vari MSG Handler
● MSG Handler Topic-based: salva sul db usando i servizi del data manager (synch).

IL DATA MANAGER, sviluppato a componenti EJB, rappresenta l'interfaccia Read (per Analyser) / Write (MSG Handler) per i database.
● Il Data Manager espone i propri servizi stateless tramite interfaccia WebSocket

DB: Mysql/MongoDB

5 repliche
● Replicazione passiva (primary-Backup con garanzia di sequential consistency: Scrittura commited da primary, Lettura da Backup) con Paxos come algoritmo di leader election e fault detector basato time-out
● A ciascun replica corrisponde un ejb istanziato sul DataManager che funge da connettore verso il database remoto

ANALYSER "acts as CONTROLLER".
Mette a disposizione all'utente finale query relative ai dati memorizzati.
Esempio
- Eventi ordinati nel tempo per tutte le macchine
- Eventi specifici ordinati nel tempo per tutte le macchine



Schema di riferimento.

Introduzione.

Questo documento è da ritenersi una documentazione non specifica dell'implementazione del progetto in questione.

Verrà di seguito spiegata l'architettura del sistema distribuito con annesse scelte progettuali.

Architettura del sistema distribuito.

Per la simulazione in locale del sistema distribuito, si è preferito l'uso di Docker per la sua versatilità e la facilità di utilizzo.

Docker.

Per lo sviluppo del progetto è stato utilizzato Docker, di preciso sono stati costruiti diversi container raggruppati tramite docker-compose:

- 1 container per ogni nodo database composto da Glassfish server, java, mysql(5 totali);
- 1 container per DataManager, nel quale è integrata anche la View, composto da Glassfish server;
- 1 container per ogni MessageHandler composto da python, pika lib per rabbitmq e websocket-client module (3 totali);
- 1 container per ogni scanner composto da python e pika lib per rabbitmq (tramite l'opzione -scale di docket-compose è possibile variare il numero di essi) ;
- 1 container per RabbitMQ

L'utilizzo di docker semplifica l'integrazione con diverse piattaforme (e.g. AWS).

Di seguito, sono spiegate le varie entità del progetto.

SCANNER.

Lo scanner è stato implementato in python, esegue una lettura dal file ERROR_WARN.log e diversifica l'invio della riga letta, tramite una politica TOPIC-Based (in questo caso diversificata per ip) verso la coda.

Gli IP (topic) interessati, come da consegna, sono:

10.18.122.23, 10.18.122.24, 10.18.122.30.

RABBITMQ.

Un container docker integra una coda RabbitMQ raggiungibile all'ip statico 10.123.123.253. Ogni entry del file di log viene spedita dallo scanner verso l'exchange di rabbitmq, la quale raggruppa i messaggi in 3 code, una per ogni topic.

MESSAGE-HANDLER.

Ad ogni topic corrisponde un message-handler (nel nostro caso 3 container), il quale ha il compito di inoltrare i dati presenti nella coda verso DataManager tramite web socket ("ws://10.123.123.100:8080/Homework3-war/write"), col fine di salvarli, tramite DataManager, nel Primary-DB.

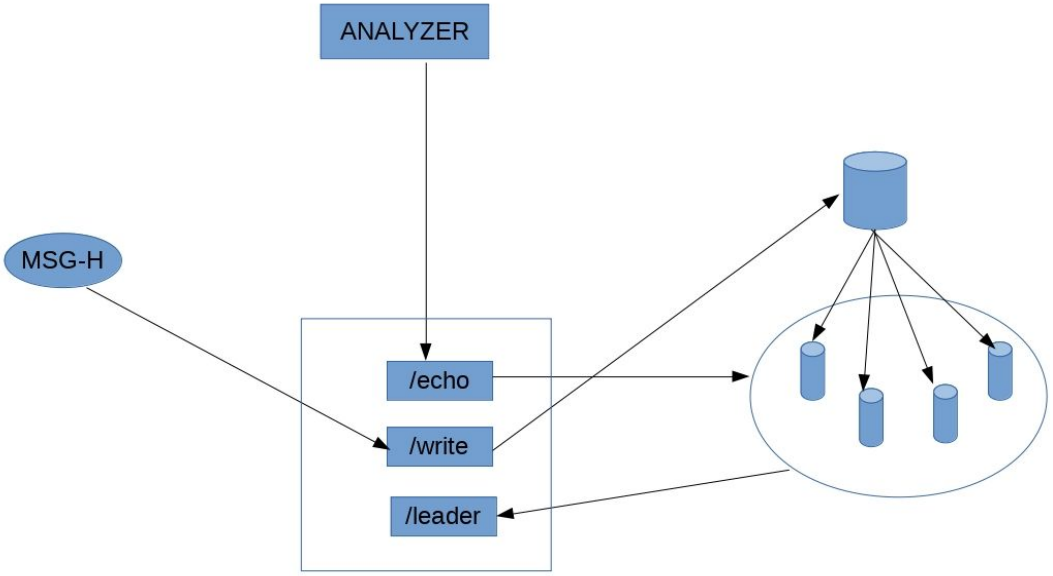
Ad occuparsi del trasferimento dati tra MSG-Handler e il DataManager è lo script in python *receiver.py* che si serve del modulo websocket-client.

DATAMANAGER.

Il DataManager ha lo scopo di:

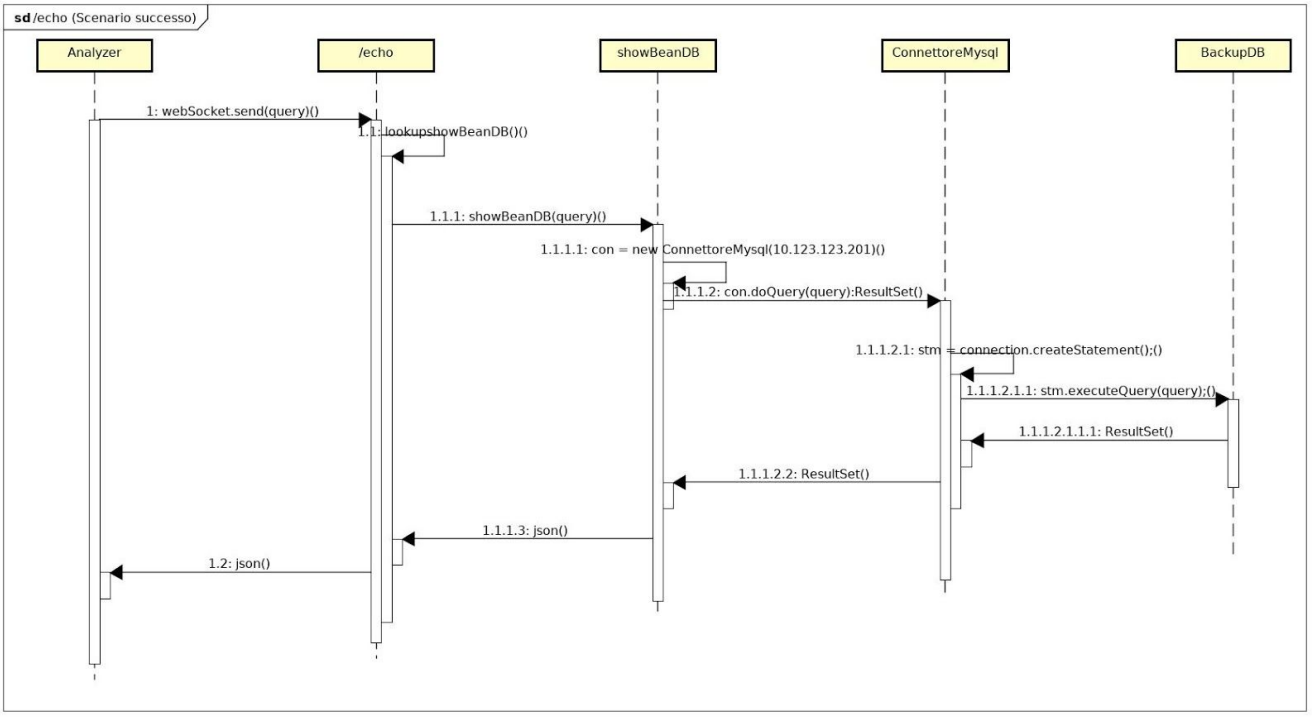
- Mettere in comunicazione MSG-Handler con il Primary per eseguire la scrittura;
- Mettere in comunicazione Analyzer (view che nel nostro caso è deployata nello stesso nodo del DM) con Replica per la lettura da DB;
- Mantenere informazioni sul leader/repliche attive.

La comunicazione avviene tramite web socket, nello specifico sono stati definiti 3 endPoint che identificano gli obiettivi appena elencati.



- **/echo**: ws tra Analyzer e DM che permette di fare delle query tramite EJB ad una replica.
- **/write**: ws tra MSG-H e DM. Attraverso di essa il DataManager prende gli elementi ricevuti dal MSG-H e li scrive sul Primary tramite EJB.
- **/leader**: Endpoint al quale le repliche, dopo la fase di leader election, comunicano l'esito dell'elezione.

/echo.



La query viene effettuata tramite un bottone presente nella view. In quell'istante viene creata una websocket con il datamanager, che alla ricezione del messaggio (OnMessage) applica le procedure necessarie alla visualizzazione dei dati richiesti.
Attualmente sono state predisposte 2 query:

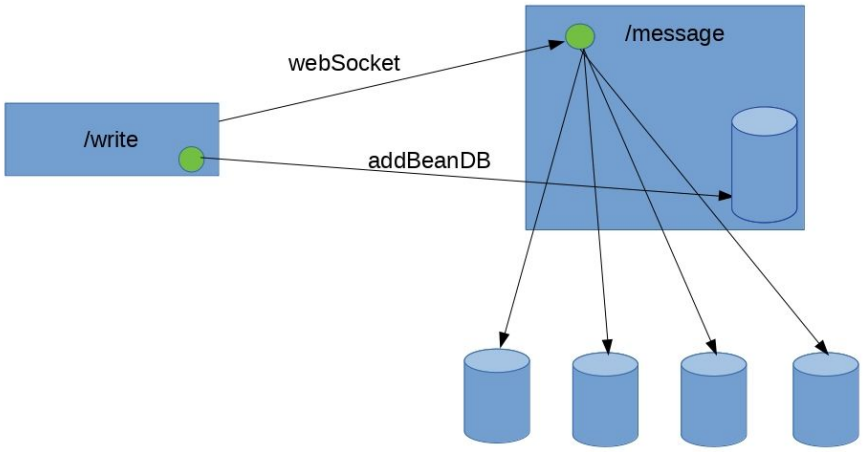
1) query1 = "SELECT * FROM log ORDER BY timestamp ASC;
2) query2 = "SELECT * FROM log WHERE timestamp BETWEEN value1 AND value2;

/write.

Per questo caso d'uso, ovvero inserimento dei dati letti dall'handler, al fine di rendere effettivo l'algoritmo di primary backup passivo si è pensato di utilizzare due canali di comunicazione:

- inserimento diretto nel db primary tramite jdbc;
- passaggio dei messaggi tramite websocket al nodo che ospita il primaryDB.

(L'utilizzo di una ws è stato fatto solamente per fini didattici, Mysql prevede features proprietarie per la gestione delle repliche).



L’invio del messaggio da parte del primary verso i backup, avviene in ordine sequenziale (tramite ciclo for).

/leader.

Alla caduta del leader (primary), viene applicato l’algoritmo paxos (leader election). A votazione avvenuta, risulta necessario comunicare al DM il nuovo leader, in quanto trattandosi di una replicazione passiva, il DM ha la necessità di sapere a chi dover comunicare la scrittura delle nuove entry.

In un file “leader” viene scritto l’indirizzo IP del nodo leader, per cui nella fase di start viene preso l’ultimo valore scritto. A run-time un’eventuale caduta del leader provoca l’avvio di una leader election scritto in bash composto da una fase di elezione e una fase di scrittura su file.Nella fase finale viene inoltre avviato uno script in python che andrà a leggere dal file (all’interno del server nodo) e aprire una ws verso il leaderEndPoint, il quale ricevuto il messaggio (OnMessage), si preoccuperà di sovrascrivere il proprio file leader.

Inoltre, vi è un controllo sulla caduta dei backup nel quale il primary e tutti i nodi che non riescono a contattare (ping-ack) un nodo backup, cancellano da un file l’ip del nodo down cosicchè il primary non proverà ad effettuare commit sul nodo down.

DATABASE

Per lo sviluppo di questo progetto si è deciso di utilizzare Mysql DB. Ogni container integra un db, contenente il db *sistemidistribuiti* ed una table *log,* e glassfish come application server.

La leader election viene effettuata attraverso uno script in bash (*leaderelection.sh*) che, ad avvenuta 'morte' del leader, avvia una fase di confronto degli id dei nodi 'vivi', il nodo ad essere eletto sarà colui che possiede id più alto. Ad elezione avvenuta, viene scritto il nuovo leader su file, e notificato al DM tramite ws (/leader).