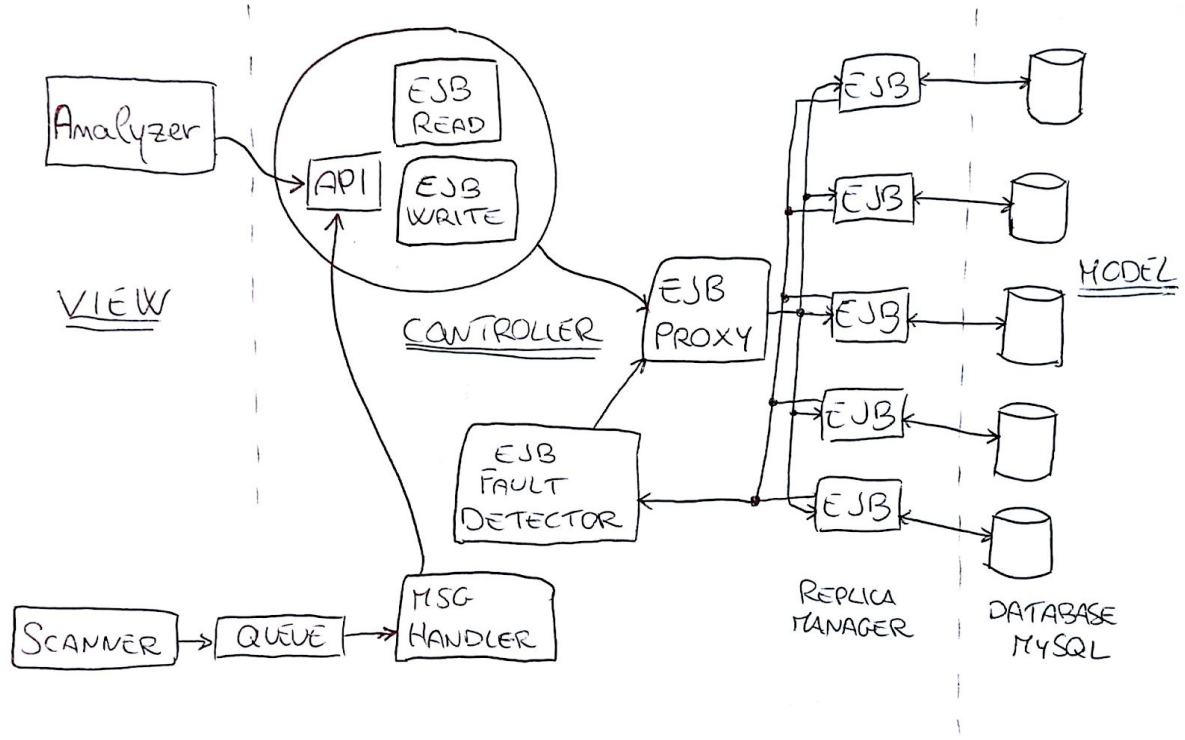
Progettazione di Sistemi Distribuiti - a.a. 2017/2018

Silvia Sottile - Luca Musumeci

Architettura



Il sistema è stato realizzato utilizzando come pattern di riferimento il Model View Controller. Qui di seguito si elencano i layer di cui è composto il sistema e la loro implementazione interna:

- **Presentation Layer**: La parte di Presentation viene gestita da un EJB che utilizza la REST API (con metodo GET) per poter eseguire la read. Per consumare tali API, viene utilizzato un Analyzer, un'applicazione sviluppata con Angular 2.
- **Business Layer**: La Business logic è implementata all'interno del data manager. Ogni replica possiede un EJB destinato alla sua gestione. Il protocollo utilizzato per la consistenza delle repliche è basato su quorum. Inoltre, è presente un EJB che si occupa della gestione dei fault basato su timeout.
- **Data Layer**: Questo layer contiene le diverse repliche. Le varie repliche saranno avviate con più container Docker.

Addizionali a questa struttura sono gli *scanner*, la *coda* e i *MSG Handler*.

Scanner

Lo *Scanner* è stato implementato come una classe Java che va a leggere, riga per riga, il file in cui sono contenuti i log. Ogni volta che c'è un entry contenente la parola chiave "WARN", l'intera riga di log viene pubblicata nella coda.
La coda è implementata con RabbitMQ, un message broker open-source, che implementa un sistema a code per l'invio e la ricezione dei messaggi. RabbitMQ mette a disposizione una ConnectionFactory per la creazione della connessione tra lo Scanner e RabbitMQ in esecuzione sulla macchina alla porta di default 5672. Viene quindi aperto un canale, impostato l'exchange type e inviate tutte le linee di log con la funzione basicPublish. Quando lo Scanner non ha più niente da leggere, vengono chiusi il canale e la connessione.

MSGHandler

I *MSG Handler* sono implementati come delle classi Java che consumano il contenuto della coda e, filtrando ulteriormente per contenuto, invieranno i dati al Data Manager. Per fare ciò, vengono utilizzate delle Regular expression per individuare i timestamp, gli ID macchina e i messaggi. Se la riga di log che viene prelevata dalla coda contiene i pattern specificati dalle regEx, viene salvata su DB effettuando la richiesta HTTP POST all'indirizzo specifico delle API.

Analyzer

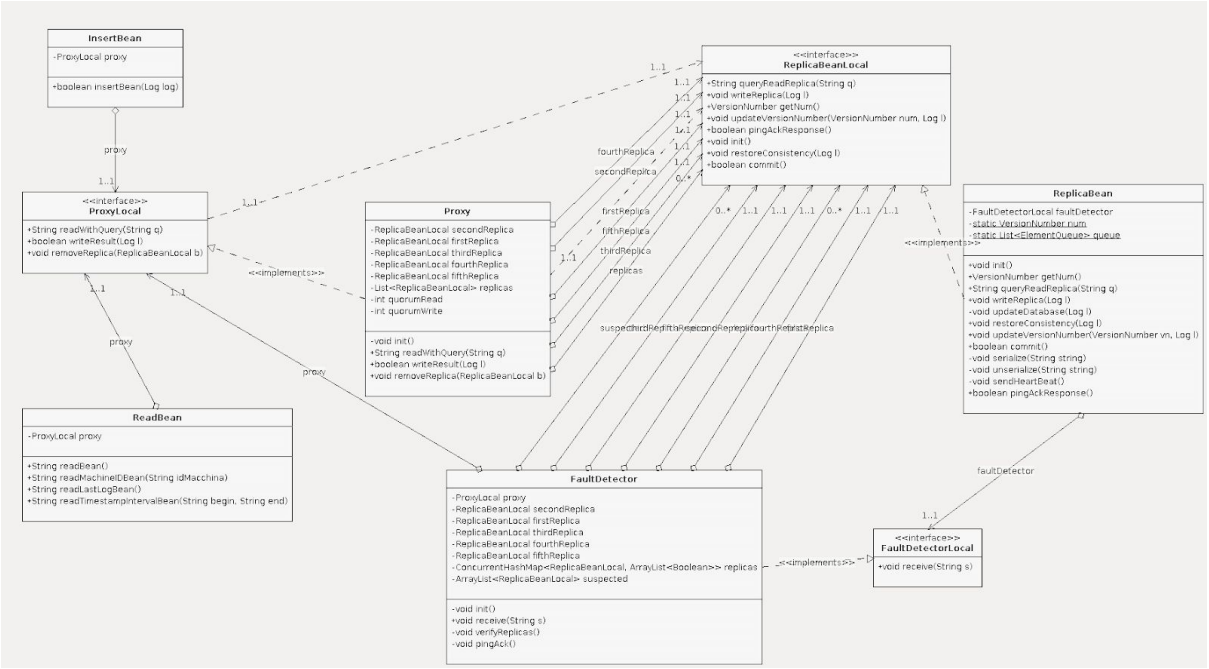
L'*Analyzer* si occupa di inviare particolari query al Data Manager. Per gestire la sua interazione con il DM vengono utilizzate le REST API. Le API disponibili sono:

- Eventi ordinati nel tempo per tutte le macchine:
  http://localhost:8080/QuorumProject-war/gestione/log/get
- Eventi ordinati nel tempo per una singola macchina:
  http://localhost:8080/QuorumProject-war/gestione/log/get/{idMacchina}
- Ultimo evento:
  http://localhost:8080/QuorumProject-war/gestione/log/getLast
- Eventi in una certa finestra temporale (timestamp nel formato "2018-02-18 16:43:03"):
  http://localhost:8080/QuorumProject-war/gestione/log/timestamp/{timestampBegin}/{timestampEnd}

E' stato implementato utilizzando la tecnologia JavaScript, nello specifico Angular 2 (angular-cli) e Node.js. Per semplificarne l'esecuzione, è disponibile una docker image che contiene le dependencies necessarie.

Data Manager

Il *Data Manager* è il fulcro della gestione della consistenza delle repliche. Qui di seguito è raffigurato il diagramma delle classi:



Gli EJB presenti nel sistema sono i seguenti:

- **InsertBean** e **Readbean**, che insieme costituiscono il punto di connessione tra front-end e back-end;
- **Proxy**, EJB Singleton che si occupa della consistenza delle repliche. Contiene l'implementazione dell'algoritmo quorum-based (si veda il paragrafo "Quorum protocol" per i dettagli);
- **FaultDetector**, EJB Singleton che si occupa di rilevare i fault nelle repliche (si veda il paragrafo "Fault Detector" per i dettagli);
- **ReplicaBean**, EJB Stateless che rappresentano i Replica Manager delle varie repliche, in particolare effettuano le varie query ai DB per scrivere/leggere i dati.

Quorum protocol

L'EJB che si occupa della consistenza è un EJB Singleton che funge da Proxy. Questo riceve le richieste dai MSG Handler e dall'Analyzer e si occupa di raccogliere il quorum:

- Read quorum: Settato a Qr = 2
- Write quorum: Settato a Qw = 4

Si è scelto di utilizzare un EJB Singleton poiché la concorrenza viene direttamente gestita dal container, rendendo più semplice la gestione al programmatore. Tale scelta è adeguata nel caso proposto, poiché la mole di dati analizzata e il numero di repliche presenti non è particolarmente elevata. In sistemi più complessi risulterebbe più opportuno utilizzare un EJB Stateless per permettere un maggior sfruttamento della pool di thread generata dai Replica Manager.
Le operazioni performate dal Proxy sono due:

Read

1. Il Proxy contatta *Qr* repliche. Per poter garantire l'accesso concorrente a diversi lettori, ogni lettore utilizza una ArrayList che contiene le repliche attualmente attive. La scelta delle *Qr* repliche viene seguita randomicamente, eseguendo una shuffle e contattando le prime *Qr* repliche.
2. Richiesto il Version Number di queste repliche, il Proxy confronta i due valori ottenuti.
3. Il Proxy legge dalla replica più aggiornata e fornisce all'Analyzer il dato.

Write

1. Il proxy, alla ricezione di una richiesta di write da parte dei MSG Handler, contatta i RM per raccogliere il quorum, richiedendo il loro Version Number ed inviando il dato da scrivere.
2. I RM conservano il dato nella loro log queue e lo marcano come non disponibile. In seguito, inviano il loro Version Number al proxy.
3. Il proxy riceve i diversi Version Number:
    a. Se rispondono almeno 4 repliche, la scrittura può avvenire. il Proxy confronta i Version Number e invia ai RM il Version Number più alto. N.B: se una delle repliche che rispondono al proxy non è allineata (cioè ha un VN minore delle altre) non è necessario che lo sia prima che avvenga la scrittura poiché stiamo supponendo uno scenario in cui non è necessaria una tale consistenza.
    b. Se rispondono meno di 4 repliche, il quorum di scrittura non è raggiunto e la scrittura non va a buon fine. In tal caso, il dato viene perso e il client proseguirà normalmente. N.B. Se una replica è in fault e il Fault Detector non ha ancora avvisato il proxy, è possibile che alcune scritture avvengano sulle repliche non in fault, anche se non si è raggiunto il quorum. Per questo motivo, è stato previsto un meccanismo di mantenimento della consistenza e della atomicità dell'operazione di commit.
4. I Replica Manager aggiornano il loro Version Number basandosi su quello inviato dal proxy e riordinano la loro coda in base a quest'ultimo, marcando la

scrittura come disponibile. Se l'elemento in testa alla coda è disponibile, il RM esegue la scrittura del dato nella replica. Questo procedimento continua fino a quando la testa della coda ha valori marcati come disponibili.

I Version Number utilizzati sono costituiti da una coppia <sq, ID>:

- sq è un valore intero utilizzato per l'ordinamento;
- ID è l'identificativo della replica.

Tale gestione permette scritture in Total Ordering. La connessione tra i Replica Manager e il Proxy è supposta immediata per semplicità. Ciò comporta che, alla richiesta di un Version Number da parte del Proxy, il Replica Manager risponda immediatamente alla richiesta.
E' stata scartata l'idea di un Sequencer per stabilire i VN perché risulterebbe un collo di bottiglia nel caso in cui il sistema sia soggetto ad uno Scale In o Scale Out.

Fault Detector

L'EJB che si occupa dei fault è un Singleton che implementa un fault detector basato su timeout. Esso riceve costantemente dai RM un messaggio di Heartbeat, che segnala l'attività della replica. Tale messaggio viene inviato dalla replica se un opportuno test di connessione dà esito positivo. Se il Fault Detector non riceve un messaggio di HeartBeat da una specifica replica per un periodo di tempo, esso passerà ad effettuare un controllo più specifico, eseguendo un ping-ack sulla specifica replica. Tale ping-ack impone un test di connessione con timeout più elevato. Se entro questo periodo la replica non risponde, essa è considerata in crash: il Fault Detector informerà il Proxy di non contattare più quella replica. Il sistema tollera il crash al più di una replica. A seguito del crash di più di una replica, la scrittura viene bloccata e si potrà eseguire solo la lettura. A seguito del crash di 4 o più repliche, il sistema non sarà in grado di effettuare le letture. Quando una replica è in Fault e torna operativa, non è previsto alcun meccanismo di recovery. Tuttavia, il Fault Detector continuerà a ricevere il segnale di HeartBeat di quella replica, ma lo ignorerà nella verifica delle repliche in Fault. Le repliche sono fisse e l'eventuale aggiunta di un nuovo elemento a caldo non è gestito.