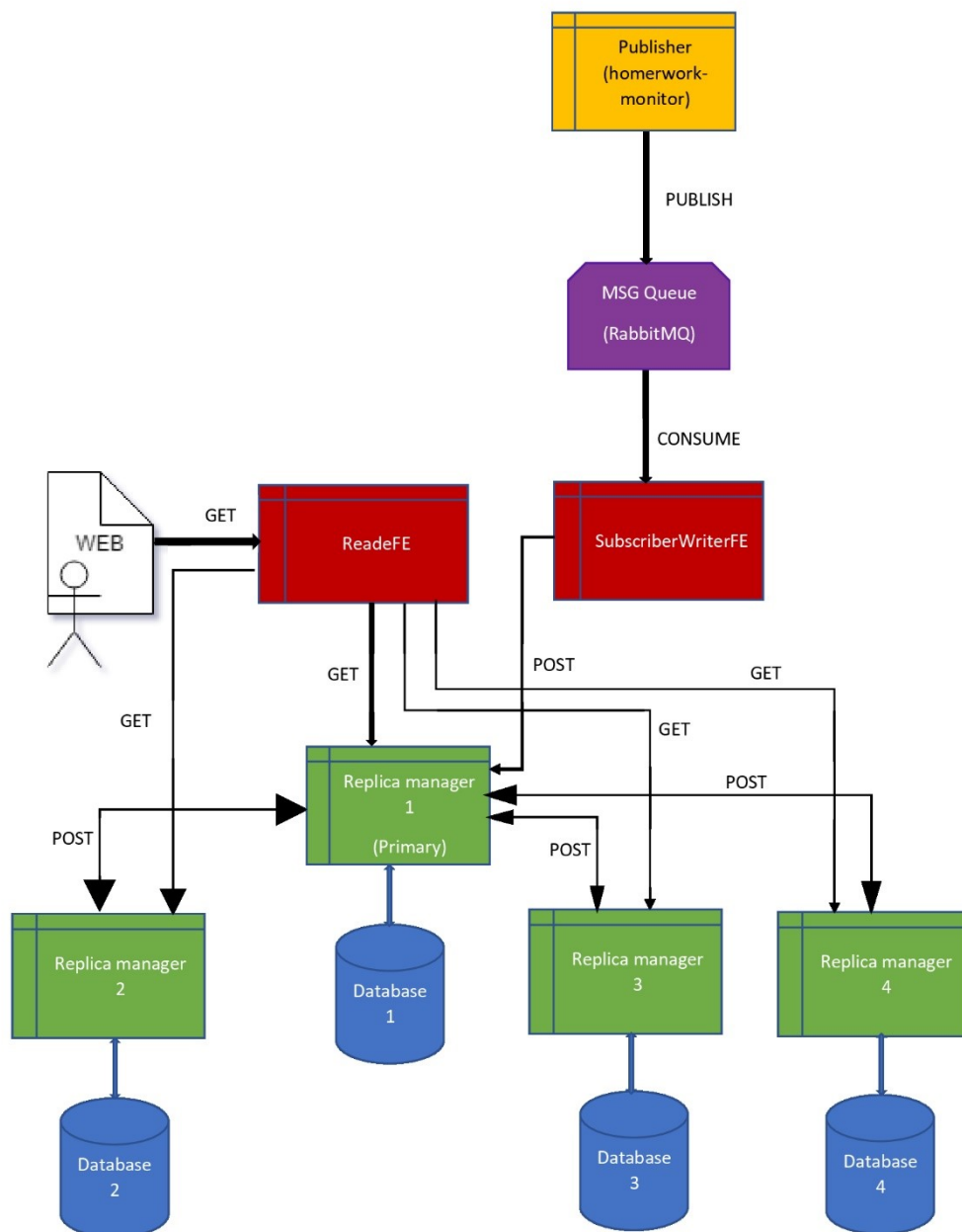


Progetto di sistemi distribuiti

Anno accademico: 2018/2019

Homework 2: 3 repliche primary-backup consistenti linearmente

Studenti: Biagio Fornitto e Matteo Bonaccorso



Il sistema è stato realizzato utilizzando il pattern Command Query Responsibility Segregation (CQRS).

Tra gli elementi dell'architettura troviamo un publisher (homerwork-monitor) che pubblica i dati su una coda rabbitMQ a cui è sottoscritto il SubscriberWriterFE. Quest'ultimo inoltra i dati al replica-manager di un database (primary) che si occupa di aggiornare e mantenere consistenti altre 3 repliche (backups). Sia il primary che le altre 3 repliche espongono i propri servizi di lettura-scrittura via interfaccia REST. Il ReaderFE è implementato attraverso una servlet ed espone 3 query sui dati.

La parte "Command" viene implementata attraverso il SubscriberWriterFE.

La parte "Query" è implementata attraverso il ReaderFE.

Il command model (Subscriber) invoca l'API di scrittura esposta dal replica-manager del primary che a sua volta invoca le API di scrittura dei backups. Il query model (Reader) può invocare le API di lettura esposte del primary e dai backups.

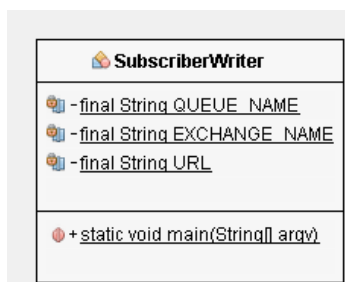
Di seguito vengono analizzati i componenti nel dettaglio.

- **Publisher**

Il publisher è una applicazione java EE che esegue il comando di sistema "netstat -i" e pubblica l'output su una coda RabbitMQ.

- **SubscriberWriterFE**

Il subscriberWriterFE è sottoscritto alla coda in cui pubblica il publisher e ricevuto il dato lo inoltra al primary attraverso delle API client Jersey. Prima di inviare il dato aggiunge un ID univoco alla richiesta, rappresentato dal tempo corrente espresso in nanosecondi.



- **Primary**

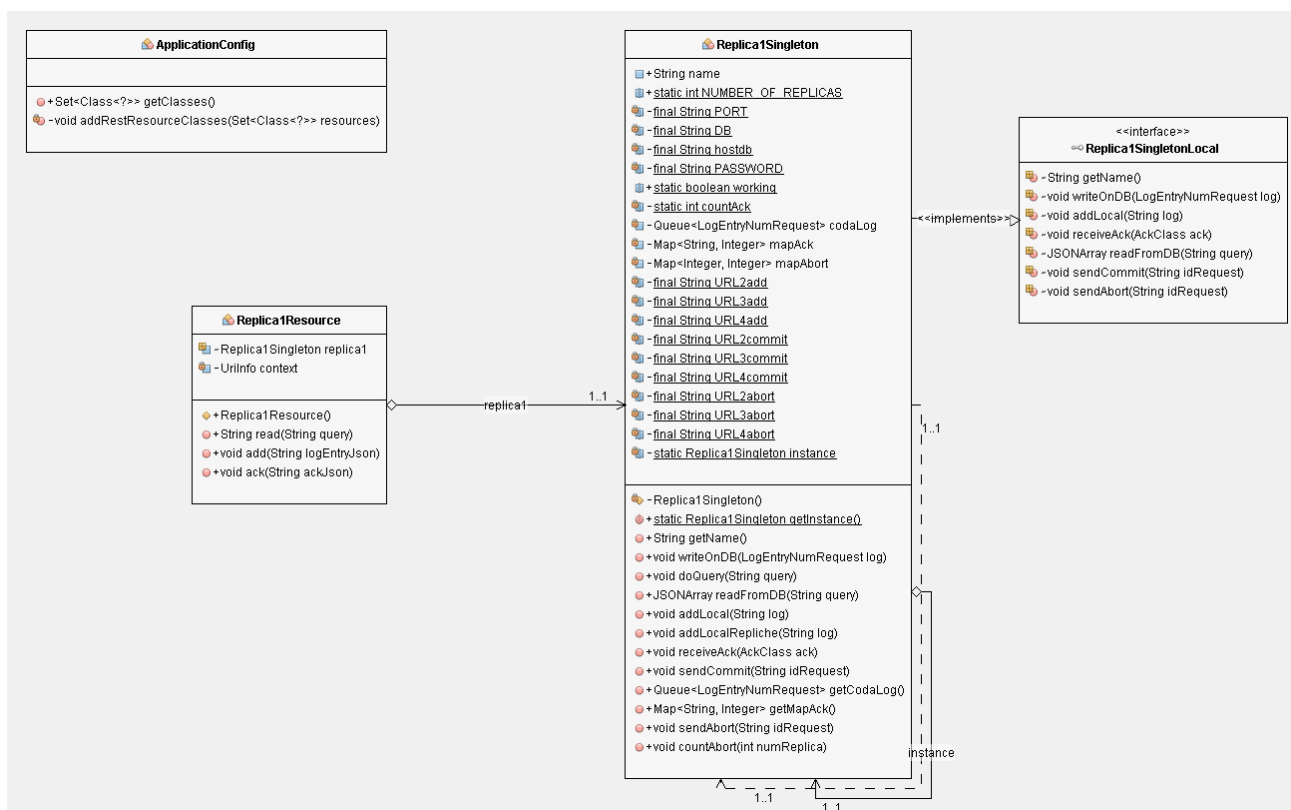
Il primary è implementato attraverso la classe "Replica1Resource" in cui è inserito il riferimento ad una classe singleton ("Replica1Singleton") ed espone i metodi di lettura e scrittura del database a cui fa riferimento. Ogni API della classe "Replica1Resource" richiama il relativo metodo della classe "Replica1Singleton". Le API esposte sono rispettivamente:

-API di lettura che viene richiamata dal ReaderFE;

-API di scrittura che viene richiamata dal SubscriberWriterFE;

-API di ricezione ack che viene richiamata dai backups per dare conferma dell'avvenuta ricezione di un dato.

Il replica-manager del primary riceve il dato dal SubscribeWriterFE, lo inserisce in una coda (log in locale) e lo inoltra alle altre repliche sfruttando delle API client Jersey e si mette in attesa per un dato periodo di tempo. Ricevuto l'ack da tutte le repliche fa il commit della richiesta nel suo database e invia il commit alle altre repliche, eliminando il dato dalla coda. Dopo di che si passa alla gestione della successiva richiesta. Se scade il timeout e tutte le repliche non hanno mandato l'ack, il primary inoltra un messaggio di abort con l'id della richiesta relativo e riprova ad inoltrare il dato. Alla terza volta che una replica non risponde con l'ack viene eliminata dal primary e non riceverà più nessun dato. Il primary continuerà a mantenere la consistenza con le repliche rimanenti. Se riceve una query dal ReaderFE la sottopone al database e ritorna il risultato richiesto.

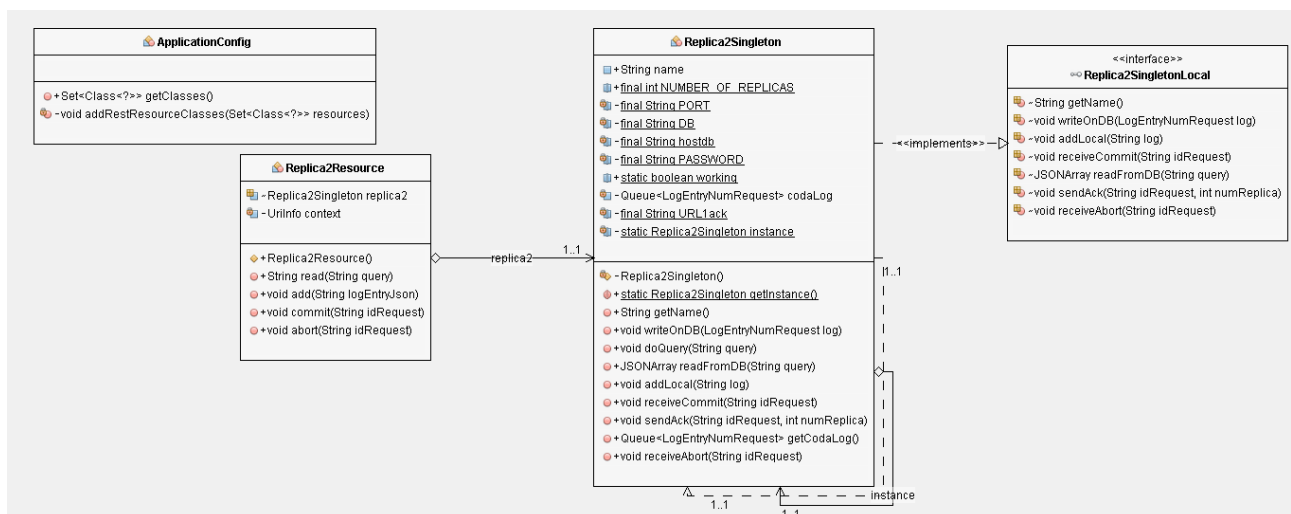


• Backup

I backup sono implementati allo stesso modo del primary, soltanto che le loro API di scrittura vengono invocate solo da quest'ultimo. Ogni API della classe "ReplicaNResource" richiama il relativo metodo della classe "ReplicaNSingleton". Le API esposte sono rispettivamente:

- API di lettura che viene richiamata dal ReaderFE;
- API di scrittura che viene richiamata dal primary (Replica1);
- API di abort che viene richiamata dal primary;
- API di ricezione commit che viene richiamata dal primary;

Il replica-manager di ogni backup riceve il dato dal primary, lo inserisce in una coda (log locale) e inoltra al primary l'ack della richiesta relativo sfruttando delle API client Jersey e si mette in attesa. Ricevuto il commit dal primary fa il commit nel proprio database ed elimina il dato dalla coda. Dopo di che si passa alla gestione della successiva richiesta. Se riceve un messaggio di abort da parte del primary con l'id della richiesta relativo, verifica se il dato è presente in coda e lo elimina. Se riceve una query dal ReaderFE la sottopone al database e ritorna il risultato richiesto.












• ReaderFE

Il readerFE è implementato attraverso una servlet che espone un'interfaccia web (pagina html).

La servlet in base alla query e alla replica selezionata (primary o backup) dall'utente invoca l'API di lettura corrispondente e restituisce il risultato formattato in JSON. Le query esposte richiedono:

- lista delle interfacce di sistema
- interfaccia con uso maggiore

-profilo di utilizzo di un'interfaccia

 Reader
 -final String URL1read  -final String URL2read  -final String URL3read  -final String URL4read
 #void processRequest(HttpServletRequest request, HttpServletResponse response)  #void doGet(HttpServletRequest request, HttpServletResponse response)  #void doPost(HttpServletRequest request, HttpServletResponse response)  + String getServletInfo()