Report sviluppo progetto

Daniele Caldarini, Michele Tedesco, Danilo Ponti, Luca Lomasto, Lorenzo Vaccaro

Compito:
Creare un ambiente virtuale basato su Apache Kafka.
Effettuare il porting di applicazioni e script di esempio per JMS.

Descrizione del repository:
Sono state realizzate due applicazioni basate su comunicazione asincrona e scambio di messaggi su Apache Kafka (https://github.com/Meyke/simpleKafkaProject.git).
La prima applicazione è basata sullo scambio di messaggi in modo asincrono tra un singolo produttore e un singolo consumatore tramite Kafka.
La seconda,più complessa, prevede lo scambio di messaggi tra un produttore e più consumatori con un filtro per i messaggi.
Sono stati creati due ambienti virtuali basati su Docker Swarm (installato all'interno di macchine virtuali Vagrant).

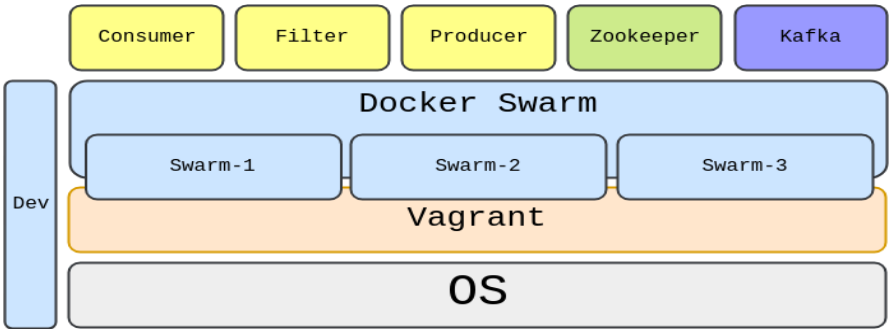
In particolare nel seguito viene descritta l'applicazione multiple-producer-consumer.

Tecnologie d'interesse:
Nel corso del progetto è stato sperimentato l'utilizzo di kafka attraverso le seguenti tecnologie:

- **Apache Kafka**: tecnologia d'interesse.
- **Zookeper**: come server di acquisizione del flusso per kafka
- **Java**: per lo sviluppo della logica applicativa.
- **Gradle**: per la build del software.
- **Vagrant**: come strumento per la gestione e la creazione di ambienti virtuali su cui eseguire Docker e Docker-Swarm
- **Docker**: per la creazione dei contenitori.
- **Docker-Swarm**: per l'orchestrazione dei servizi nell'applicazione multiple-producer-consumer.

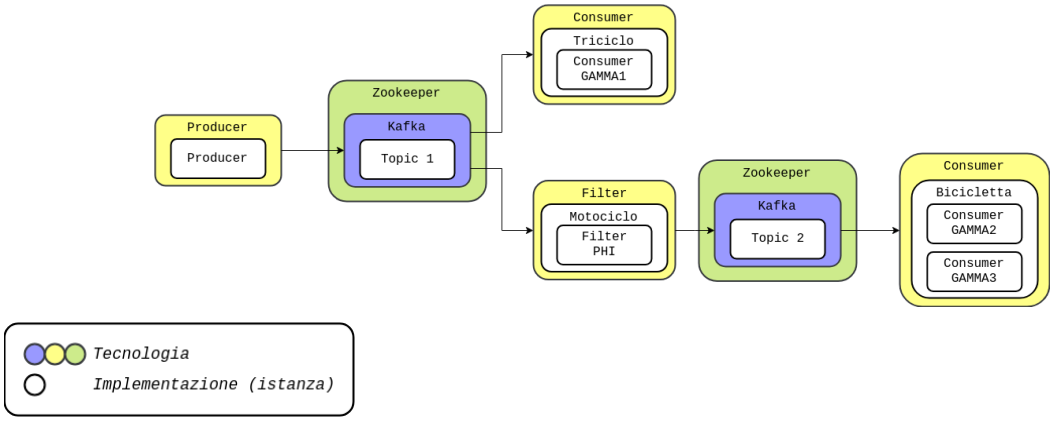
.

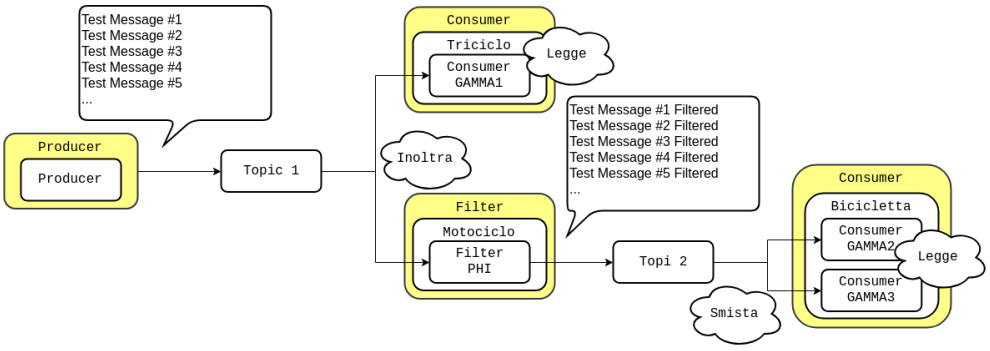
Ambiente di sviluppo:



Il seguente schema (da leggere dal basso verso l'alto) descrive in maniera abbastanza generale l'ambiente utilizzato per lo sviluppo dell'applicazione.
Viene utilizzato Vagrant per la gestione e la creazione di un ambiente virtuale basato su Docker Swarm. In particolare l'ambiente è costituito da tre macchine virtuali per la creazione dello Swarm e una macchina "dev" più che altro utilizzata nella fase di sviluppo del software.

Logica Applicativa:
Nel seguito vengono mostrati due diagrammi che descrivono la logica dell'applicazione *multiple-producer-consume*r; il primo con riferimento anche alle tecnologie che implementano i vari oggetti, il secondo con una vista di esempio su come i messaggi vengono scambiati e filtrati.
In particolare l'applicazione è costituita da un produttore che invia messaggi a un canale chiamato "topic1". Questi messaggi vengono a loro volta consumati da un consumatore "ConsumerGamma1" e da un filtro "Filter PHI". In particolare il filtro attua un processo di trasformazione dei messaggi (viene semplicemente concatenato al messaggio la stringa "Filtered") e li invia al canale "topic2", quest'ultimo costituito da due partizioni. A questo punto i messaggi sono consumati da un gruppo formato da due consumatori che, secondo la politica di smistamento dei messaggi adottata per default da kafka, si divideranno il flusso di messaggi (in maniera round robin).





Build, deployment e esecuzione:

Per quanto riguarda la build, è stato usato Gradle.

Per il deployment dell'applicazione, ogni servizio è costruito per girare su un proprio contenitore Docker sul quale è stato installato il software necessario per l'esecuzione dell'applicazione e il servizio stesso. Come già descritto nella sezione relativa all'ambiente di sviluppo, l'orchestrazione di contenitori è gestita da Docker Swarm.

Per l'esecuzione si rimanda al README del sottoprogetto nel repository).

Problemi e soluzioni:

- Un problema (più che altro un dettaglio) emerso durante lo sviluppo e il test dell'applicazione è stato quello per cui il servizio che crea i topic non riusciva a generarli poichè questi risultavano già creati. Infatti esplorando i log del servizio veniva segnalato come questi già esistessero. Il problema era dovuto al fatto che Kafka di default crea i topic che vengono utilizzati come sorgente/destinazione dai consumatori/produttori.
  Per evitare la creazione automatica dei topic durante l'esecuzione del servizio produttore abbiamo quindi aggiunto una linea al file di configurazione di Kafka: **"echo 'auto.create.topics.enable=false' >> $KAFKA_HOME/ config/ server. properties".** Questa riga di codice shell viene eseguita all'avvio del servizio Kafka e modifica la configurazione di Kafka. Garantisce che i topics vengano creati manualmente dal servizio che crea i topic.
- Parecchi problemi sono stati riscontrati anche per far si che i contenitori con i relativi servizi fossero avviati nell'ordine giusto (Kafka dopo Zookeeper, servizio per la creazione dei topic dopo Kafka, …). Infatti, come è scritto sulla documentazione Docker, **l'attributo depends_on del file di configurazione di docker swarm (docker-stack.yml) dalla versione 3 (appunto quella di docker swarm) non funziona più.** Come rimedio si è optato nell'utilizzo di script che gestiscono l'ordine di esecuzione (effettuano un monitoraggio dei server e avvertiscono se quel servizio , es. zookeeper, kafka, si è avviato). Tale soluzione è stata suggerita dalla documentazione di Docker. Lo script che gestisce l'attesa è chiamato "wait-for-zookeeper.sh" per attendere l'avvio di Zookeeper e "wait-for-kafka.sh" per

attendere l’avvio di Kafka. Per la rivelazione dei servizi attivi abbiamo scelto di usare telnet, interrogando il contenitore di kafka e zookeeper.

- Molti problemi sono sorti anche in merito alla replicazione del server Kafka; problemi che ci hanno impedito di trovare una soluzione corretta ed efficiente al problema e che ci hanno quindi portato ad abbandonare la questione. In particolare ciò che segue è quello che abbiamo provato a fare per replicare il server di Kafka e i problemi riscontrati nella ricerca di una soluzione.
    - Il primo tentativo è stato quello di replicare Kafka specificando nello stack di Swarm di creare due repliche dello stesso servizio Kafka con le informazioni riguardanti topic e flusso di messaggi condivise tra i due servizi. In questa soluzione sono sorti problemi dovuti al fatto che non era possibile creare due broker con lo stesso id sullo stesso server Zookeeper.
    - Allora si è cercato di replicare anche Zookeeper in modo da far girare Kafka su due diversi server Zookeeper, ma questo ha portato a vari problemi nella gestione delle diverse repliche e nella comunicazione e gestione di queste.
    - Infine si è cercato di abbandonare una replicazione ad opera di Swarm a favore di una replicazione “manuale”, specificando nello stack due diversi servizi Kafka che con broker con id diversi e informazioni sui messaggi scambiati condivise. Anche questa soluzione però ha portato a svariati problemi nella gestione della replicazione(che poi vera e propria replicazione non è).

  In conclusione non siamo, con il tempo a nostra disposizione e con la documentazione a nostra disposizione, riusciti a trovare una soluzione a questo problema che fosse funzionante ed efficiente. I problemi maggiori sono nati dalla scarso supporto che Swarm, ad oggi, da nella risoluzione di alcuni problemi di orchestrazione(vedi problema con depens_on in precedenza). In effetti molte funzioni presenti nella documentazione ancora non sono supportate e sono dunque ignorate in Docker versione 3 in modalità Swarm.