Progetto Distributed Systems & Big Data

Fernando Riccioli, Daniele Lucifora

January 4, 2025

Abstract

Il progetto riguarda lo sviluppo di un'applicazione di gestione delle informazioni finanziarie, basata su un'architettura a microservizi e comunicazione tramite gRPC. L'applicazione consente la registrazione e gestione degli utenti, l'aggiornamento dei ticker finanziari e la raccolta di dati relativi ai valori azionari. Utilizzando un database MySQL, i microservizi sono separati in due categorie principali: uno per la gestione degli utenti e uno per la gestione delle informazioni sui ticker azionari. La comunicazione tra client e server avviene tramite chiamate gRPC, sfruttando i protocolli di serializzazione Protobuf. Inoltre, il progetto implementa un sistema di cache per garantire l'unicità delle operazioni (at-most-once), evitando chiamate duplicate per le stesse operazioni. L'applicazione supporta operazioni di registrazione utente, aggiornamento dei ticker, eliminazione degli utenti e recupero dei valori storici e media degli ultimi valori selezionati.

Tecnologie utilizzate:

- gRPC: Protocollo di comunicazione tra client e server.
- MySQL: Database relazionale per memorizzare le informazioni.
- TTLCache (from Cachetools): Cache con un time-to-live (tempo di vita) per gli elementi memorizzati. Usata per implementare la politica at-most-once.

Homework 1

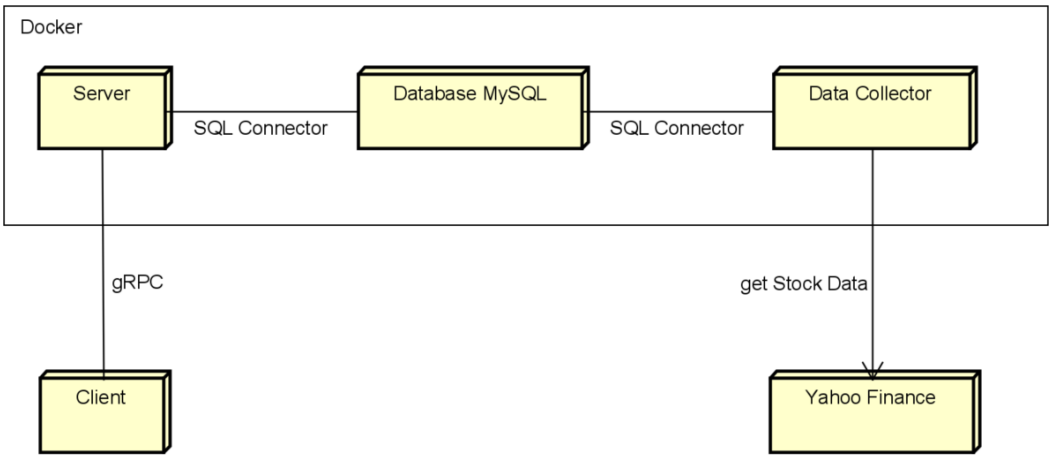

Figure 1: Diagramma Architetturale di Client-Server e Data Collector.

Homework 2

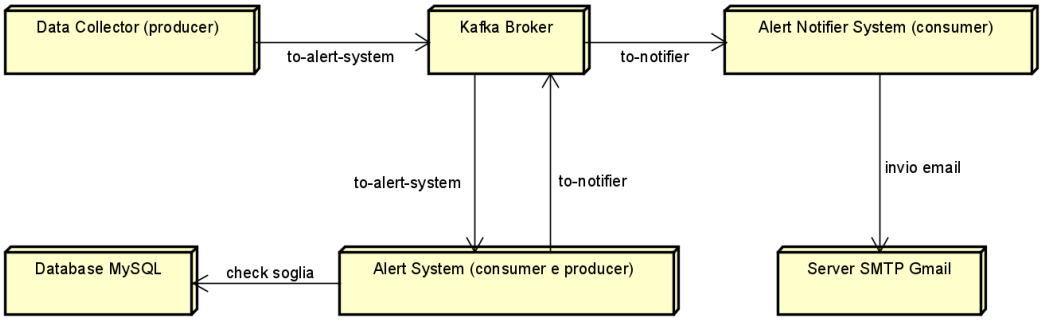

Figure 2: Comunicazione dei microservizi tramite Kafka.

Sistema di notifiche tramite Kafka

È stata inserita la possibilità di definire due valori di soglia, un minimo e un massimo, associati al ticker di ciascun utente, in maniera da ricevere delle notifiche al raggiungimento di uno dei due valori. Il sistema di notifica di posta elettronica è implementato tramite l'aggiunta di un singolo broker kafka che permette la comunicazione indiretta tra Data Collector, Alert System e Alert Notifier System. L'invio delle email avviene attraverso l'indirizzo `alert.notifier.system@gmail.com` creato appositamente per l'invio di email di notifica. Attraverso la libreria `smtplib` l'Alert Notifier System invia al server SMTP Gmail messaggi di notifica con il ticker dell'azione come oggetto e la soglia raggiunta come testo del messaggio.

Pattern CQRS

È stato implementato il pattern Command Query Responsibility Segregation per gestire in maniera indipendente i servizi di scrittura e di lettura del Server gRPC. Al server sono stati aggiunti `ComandoAggiornaUtente`, `ComandoRegistraUtente` e `ComandoCancellaUtente` che vengono forniti come parametri di ingresso per i metodi di handling del Command Service. Per l'implementazione della Query è stata utilizzata la classe Query Service che comprende i metodi per il recupero dell'ultimo valore e della media degli ultimi valori per un determinato ticker.

Homework 3

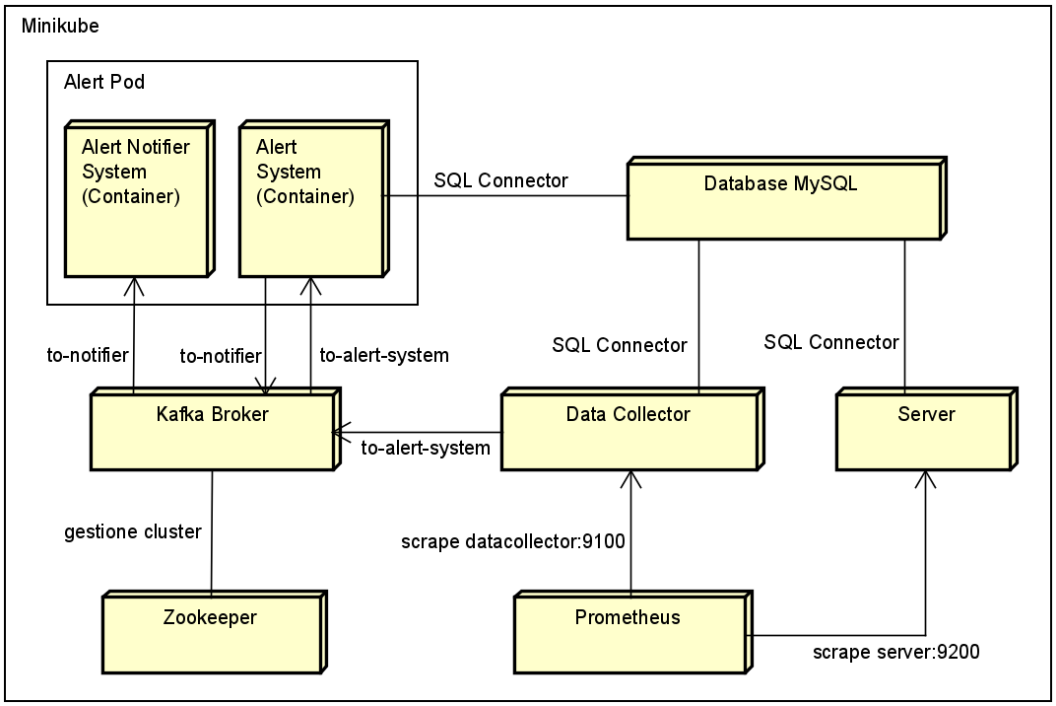

Figure 3: Diagramma Architetturale dei pod all'interno di Minikube.

Prometheus

È stato sviluppato un sistema di monitoraggio basato sull'exporter Prometheus. Prometheus stesso è stato inserito tra i microservizi gestiti. I microservizi soggetti al monitoraggio sono il Server gRPC e il Data Collector. Le metriche raccolte sono:

- **Server gRPC**
  - Numero Registrazioni: Metrica di tipo Counter che indica il numero di volte che la funzione `RegistraUtente` viene chiamata. Non corrisponde necessariamente al numero effettivo delle registrazioni dato che l'utente può essere duplicato o i dati inseriti possono essere non corretti.
  - Tempo Recupero Valore: Metrica di tipo Gauge che indica il tempo di esecuzione della funzione `Recupera Valore` chiamata dal server. La funzione comprende la connessione al database, l'esecuzione della query e la restituzione dell'oggetto Valore della comunicazione gRPC.
- **Data Collector**
  - Iterazioni DC: Metrica di tipo Counter che indica il numero di iterazioni del Data Collector tramite il ciclo `while`. Il tempo di attesa tra un ciclo e il successivo è di 60 secondi.
  - Tempo Risposta YF: Metrica di tipo Gauge che indica il tempo di risposta di Yahoo Finance nel ritorno dell'ultimo valore azionario. Il calcolo della metrica è inserito all'interno della funzione `recupera_ultimo_valore`.

Attraverso il file `prometheus.yaml` vengono impostate le porte 9100 e 9200 del Data Collector e del Server gRPC rispettivamente per la raccolta dei dati. È possibile accedere

alla Prometheus Web UI da locale all'indirizzo `localhost:9090` dopo la creazione dei cluster, seguendo la procedura descritta nel documento di Build & Deploy.

Kubernetes

I microservizi presenti sono stati deployati ed eseguiti su piattaforma Kubernetes. Il cluster Kubernetes è stato creato ed eseguito localmente, su un singolo nodo, attraverso l'uso di Minikube. Sono state utilizzate sia immagini Docker da repository remote che immagini Docker locali.

Remote Docker Images:

- MySQL: version 8.0
- Prometheus: latest Prom Image
- Kafka broker: latest Confluentinc Image
- Zookeeper: latest Confluentinc Image

Local Docker Images:

- Alert Notifier
- Alert System
- Data Collector
- Server

Le immagini Docker locali vengono dapprima create localmente utilizzando i Dockerfile presenti, poi viene assegnato un tag ad ogni immagine ed infine viene effettuato il push delle immagini alla repository Docker remota. Le immagini presenti nella repository Docker remota vengono in seguito caricate nel cluster Minikube. La configurazione Kubernetes è stata fatta attraverso la generazione dei manifest. Per ogni microservizio è stato configurato un Pod, tranne per i microservizi Alert Notifier e Alert System che sono stati configurati all'interno di un unico Pod. Grazie a ciò viene garantita una maggiore efficienza, dovuta al fatto che Kubernetes gestisce un Pod in meno e i due container hanno lo stesso ciclo di vita. L'esecuzione del Client in `localhost` ci obbliga a creare un tunnel dalla porta locale verso la porta dei Pod Server e Prometheus all'interno del cluster Kubernetes. Le porte utilizzate sono rispettivamente, 50051 per la comunicazione con il Server e 9090 per la comunicazione con il sistema di monitoraggio Prometheus.[1]

[1] Questo documento è stato formattato e impaginato utilizzando LaTeX, il linguaggio di Markup inventato da Leslie Lamport, pioniere dei sistemi distribuiti.