Software Architecture and Platforms
Assignment 03

Marco Fontana

February 15, 2025

1 Introduzione

Obiettivo di questo assignment era quello di rendere l'applicazione event driven, utilizzare per il deploy la tecnologia kubernetes e aggiungere una 'abike' che raggiungesse in autonomia la stazione di ricarica più vicina una volta scarica e l'utente ogni volta che venisse richiesto.

2 Event driven architecture

Per rendere l'applicazione event driven si è utilizzata la tecnologia 'kafka' che permette, mediante event broker, di scambiare eventi tra i microservizi.

Il layer di infrastructure contiene un consumer kafka, che rileva eventi dalle sottoscrizioni effettuate e chiama il 'service' nel layer in cui è contenuto che effettua le operazioni sugli elementi raccolti, in modo da mantenere separata l'esecuzione della logica associata agli elementi del dominio e la rilevazione degli eventi dai quali sono raccolti. All'interno del service un producer di kafka genera ulteriori eventi che possono essere raccolti dai consumer degli altri micro servizi in modo da aggiornare i propri dati riguardo cambiamenti effettuati dal servizio del micro servizio considerato.

Nel dettaglio:

- 'user-service' genera un evento kafka riguardo l'aggiunta di un utente al micro servizio, questo è raccolto dai micro servizi 'ride-service' e le interfacce utente, in modo che possano aggiornare i propri database con il dato che verrà poi utilizzato da essi.
- 'bike-service' genera un evento kafka riguardo l'aggionra di nuove biciclette al micro servizio, questi sono raccolti dai micro servizi 'ride-service' e le interfacce utente, in modo che possano aggiornare i propri database con il dato che verrà poi utilizzato da essi.

- 'ride-service' genera diversi eventi riguardo gli aggiornamenti che vengono effettuati sulle biciclette e gli utenti ad ogni cambiamento che viene rilevato riguardo ogni dato che li compone (es: posizione, crediti, batteria, direzione, ...), in modo che 'user-service' e 'bike-service' possano aggiornare il proprio database in base ai cambiamenti che vengono effettuati in 'ride-service', mentre le interfacce 'admin' e 'user' mostrano a video tutti i cambiamenti effettuati dai micro servizi in tempo reale.

Mediante event sourcing è possibile applicare tutti gli eventi salvati dall'event broker in modo da ricreare lo stato attuale di un oggetto a partire dagli eventi raccolti.

3 Deployment

La tecnologia utilizzata è quella di kubernetes, essendo da questa tecnologia richiesto che le immagini docker fossero presenti su docker-hub, sono state pubblicate tutte le immagini necessarie ad eseguire l'applicazione al seguente link "https://hub.docker.com/repository/docker/marcofontana/sap-ass-03/general".

Sono stati usati diversi oggetti di kubernetes per effettuare il deploy usando questa tecnologia invece di docker:

- Pods: oggetti che possono contenere uno o più container generati a partire da una immagine docker. I Pods sono effimeri e possono essere interroti e ricreati autonomamente a seconda delle necessità.
- Deployment: oggetti utilizzati per maneggiare Pods in autonomia, i Pods non sono controllati dall'utente ma dai Deployment che aggiungono un livello di astrazione ulteriore. Un campo 'selector' gestisce la connessione tra Pods e Deployment, sono etichette uniche che sono utilizzate per identificare i Pods che il Deployment deve gestire.
- ConfigMap: configura i container del Pod specificati nel Deployment con i dati qui specificati.
- Service: effettuano 'service discovery', sono il punto di contatto dell'applicazione e reindirizzano le richieste ai Pods dei nodi del cluster. Questo è dovuto al fatto che i Pods sono effimeri e possono essere spostati tra i vari nodi, quindi il loro indirizzo IP può cambiare nel tempo. Esistono diversi tipi di Service, quelli utilizzati in quest applicazione sono:
  - ClusterIP: espone il Service ad un IP che è raggiungibile solamente dall'interno del cluster.
  - NodePort: espone il Service all'IP del nodo con una porta prefissata.

4 Agent Bike

Le ABike sono agenti in grado di leggere l'ambiente e prendere determinate azioni in base ad esso.
Per poterle implementare sono state aggiunte alcune alcune funzioni prima non presenti:

- Environment: una classe Environment contiene 4 stazioni posizionati in punti fissi sulla mappa che rappresentano stazioni di ricarica che le biciclette possono raggiungere in autonomia quando la loro batteria raggiunge lo '0' nel modo che è spiegato di seguito. Per rendere più evidente che le biciclette raggiungessero le stazioni non appena si scaricassero, la batteria di ciascuna di esse è stata impostata a 1, in modo che possano scaricarsi poco dopo l'avvio di una corsa e si possa vedere subito come esse si spostino autonomamente alla stazione di ricarica più vicina a dove si trovavano al momento in cui la batteria ha raggiunto lo 0.

- Posizione degli utenti: al momento della creazione dell'utente si può specificare una posizione (coordinate x e y) che indica la posizione di esso nella mappa.

Gli agenti creati sono di tipo 'Model Based Reflex Agents' in quanto incapsulano uno stato interno che dipende dalla percezione dell'ambiente avvenuta in passato, ed in base allo stato in cui si trovano effettuano decisioni differenti.

Gli stati in cui possono trovarsi dipende dal tipo di problema che deve essere risolto:

- Raggiunta di una stazione quando la batteria è scarica:

    1. L'agente si trova allo stato iniziale dove effettua una lettura dell'ambiente per rilevare la stazione a lui più vicina
    2. Comincia a muoversi verso la stazione mentre continua ad effettuare letture sull'ambiente per verificare se nel frattempo viene rilevata una stazione più vicina
    3. Quando raggiunge la stazione, ricarica la propria batteria e termina ogni altro tipo di azione.

- Raggiunta di un utente quando richiesto:

    1. L'agente si trova allo stato iniziale dove effettua una lettura per rilevare la posizione dell'utente
    2. Comincia a muoversi verso l'utente
    3. Una volta raggiunto l'utente cessa ogni altro tipo di attività