UsingEureka

In un'architettura a *Microservizi*, i servizi spesso devono individuarsi a vicenda. Invece di codificare la posizione dei servizi, è possibile utilizzare un meccanismo di individuazione dei servizi, per trovarli dinamicamente in base al loro nome.

Eureka funge da registro in cui i servizi possono registrarsi e individuare altri servizi registrati.

I punti salienti del servizio Eureka per il discovery dei microservizi sono:

- *Integrazione con Spring Cloud*: Eureka è un componente chiave della suite *Spring Cloud Netflix* e si integra facilmente con le applicazioni Spring Boot, fornendo astrazioni e configurazioni semplificate.
- *Service Registry Centralizzato*: Eureka Server funge da registro centrale in cui le istanze dei microservizi si registrano automaticamente, fornendo informazioni sulla loro posizione (indirizzo IP e porta).
- *Registrazione Automatica*: I microservizi si registrano automaticamente con Eureka Server all'avvio e annullano la registrazione all'arresto, riducendo la necessità di configurazione manuale.
- *Service Discovery Dinamico*: I client Eureka (altri microservizi) possono interrogare il server per scoprire le posizioni delle istanze di servizio di cui hanno bisogno, consentendo una comunicazione dinamica senza codificare indirizzi IP e porte.
- *Heartbeat e Monitoraggio dello Stato di Salute*: I client Eureka inviano regolarmente heartbeat al server per indicare che sono ancora attivi e funzionanti. Il server utilizza questi heartbeat per monitorare lo stato di salute dei servizi registrati e rimuovere dal registro le istanze non funzionanti.
- *Load Balancing*: Eureka può aiutare a implementare il bilanciamento del carico tra le diverse istanze di un servizio. I client possono recuperare un elenco di tutte le istanze disponibili di un servizio e quindi utilizzare algoritmi di bilanciamento del carico per distribuire le richieste.
- *Tolleranza agli Errori*: Eureka Server è progettato per essere altamente disponibile e può essere distribuito in cluster. Include anche una modalità di "auto-preservazione" che impedisce l'eliminazione massiccia di istanze integre in caso di problemi di rete temporanei tra i client e il server.
- *Dashboard UI*: Eureka Server fornisce un'interfaccia utente web che consente di visualizzare lo stato dei servizi registrati e le informazioni sulle istanze.
- *Metadati*: Le istanze di servizio possono fornire metadati aggiuntivi al momento della registrazione, che possono essere utilizzati dai client per decisioni di routing o di bilanciamento del carico più intelligenti.

Atiivare Eureka

Si può attivare con *eurekaOnly.yml*:

```
docker-compose -f eurekaOnly.yml -p eureka up
```

EurekaServiceConfig

La libreria com.netflix.discovery fornisce diverse classi di configurazione per le istanze Eureka.

- La interfaccia *EurekaInstanceConfig* definisce le proprietà che un'istanza Eureka deve fornire per registrarsi e comunicare con l'Eureka Server
- La classe *com.netflix.appinfo.MyDataCenterInstanceConfig* fornisce implementazioni predefinite per alcune di queste proprietà, come il nome host e l'indirizzo IP dell'istanza.

MyDataCenterInstanceConfig

- getHostName(): Il nome host che il servizio dovrebbe registrare con Eureka.
- getIpAddress(): L'indirizzo IP che il servizio dovrebbe registrare con Eureka.
- getLeaseExpirationDurationInSeconds: Indica al server Eureka per quanto tempo un'istanza deve essere considerata "UP" e disponibile, dall'ultima volta che l'istanza ha inviato un heartbeat. Il DEFAULT è 90sec
- getLeaseRenewalIntervalInSeconds: intervallo, in secondi, con cui l'istanza invia il suo heartbeat (rinnova il lease) all'Eureka Server. Il DEFAULT è 30sec

Il meccanismo di heartbeat e lease è fondamentale per la resilienza e l'accuratezza del Service Discovery:

- *Rilevamento Rapido dei Failures*: Se un'istanza crasha o diventa irraggiungibile, smette di inviare heartbeat. Eureka la rimuoverà dal registro relativamente in fretta (dopo la durata del lease), impedendo ai client di tentare connessioni a un'istanza "morta".
- *Evitare Istanza Fantasma*: Previene che istanze non più attive rimangano indefinitamente nel registro, causando errori per i client.
- *Scalabilità*: Permette a Eureka di gestire un gran numero di istanze in modo efficiente, senza la necessità di un monitoraggio attivo costante da parte del server su ogni singola istanza.

Visualizzare le istanze registrate

```
localhost:8761
http://localhost:8761/eureka/apps
```

Eliminare una istanza registrata

```
curl -X DELETE http://localhost:8761/eureka/apps/CTXBASICROBOT/192.168.1.132
```

eurekaDemo2025

Project eurekaDemo2025

Introduciamo un primo esempio di uso di Eureka in un progetto Java che non usa Spring Boot. Il progetto defnisce un servizio Java che si registra su un server Eureka e un altro servizio che scopre il primo e lo usa.

- Conviene creare una sottoclasse di MyDataCenterInstanceConfig e sovrascrivere metodi quali getHostName() o getIPAddress() per implementare la logica di risoluzione appropriata per nostra infrastruttura.

  In questo esempio, questa sottoclasse è: *EurekaServiceConfig.java*

ServiceRegistred.java	Un servizio Java (server TCP) che si registra sul server Eureka con il nome **ctxeureka** rimanendo attivo per 10 minuti. Da attivare con: gradlew runRegister

ServiceUsageSimple.java	Un programma che scopre il servizio registrato col nome **ctxeureka** e lo usa. Da attivare con: gradlew runDiscover

Eureka con CommUtils

```
public static String getMyPublicip()
public static String getServerLocalIp()
public static boolean ckeckEureka()
public static DiscoveryClient createEurekaClient( )
public static DiscoveryClient registerService(  EurekaInstanceConfig config )
public static String[] discoverService(EurekaClient eurekaClient, String serviceName)
public static String[] discoverService( String serviceName )
public static String getEnvvarValue(String envvarName)
```

Un actorqak che si registra su Eureka

- *eurekademo2025.qak*
- *CallerServiceqakTcp.java*
- *eurekademo2025Usage.qak*

CommUtils per l'uso di Eureka

La classe *CommUtils* fonrice i seguenti metodi **static** usando le seguenti classi di libreria:

```
import com.netflix.appinfo.EurekaInstanceConfig;
import com.netflix.discovery.DefaultEurekaClientConfig;
import com.netflix.discovery.DiscoveryClient;
```

Dipendenze per Eureka

```
// Dipendenza per Eureka Client
implementation 'com.netflix.eureka:eureka-client:1.10.18'
//1.10.18 compatibile con immagine Spring Cloud Netflix 3.x

// Dipendenza per Jersey Client (per fare richieste HTTP)
implementation 'com.sun.jersey:jersey-client:1.19.1'

// https://mvnrepository.com/artifact/com.netflix.servo/servo-core
implementation 'com.netflix.servo:servo-core:0.13.2'
```

eureka-client.properties

Il file deve trovarsi bel claspath del progetto. Di solito lo si inserisce nella directory main.resources speciifcando, come informazione più importante, gli indirizzi IP su cui cercare il servizio Eureka.

```
eureka.serviceUrl.defaultZone=http://localhost:8761/eureka/, http://192.168.1.132:8761/eureka/, ht

# Intervallo di rinnovo del lease (cuore pulsante)
eureka.leaseRenewalIntervalInSeconds=60

# Intervallo di espirazione del lease
eureka.leaseExpirationDurationInSeconds=60
```

Metodi per l'uso di Eureka

boolean checkEureka()	restituisce true se Eureka è stato trovato
DiscoveryClient createEurekaClient( )	crea un Eureka client con riferimento al file *eureka-client.properties*

DiscoveryClient createEurekaClient(instanceConfig)	crea un client usando le informazioni date dall'oggetto passato come parametro
registerService(EurekaInstanceConfig instanceConfig)	registra un servizio descritto dall'oggetto passato come parametro
String[] discoverService(String serviceName)	restistuisce la coppia **host:port** del servizio in un array di due componenti facendo il discovery con riferimento al file *eureka-client.properties*
String[] discoverService(EurekaClient eurekaClient, String serviceName)	restistuisce la coppia **host:port** del servizio in un array di due componenti facendo il discovery con riferimento al file *eureka-client.properties*