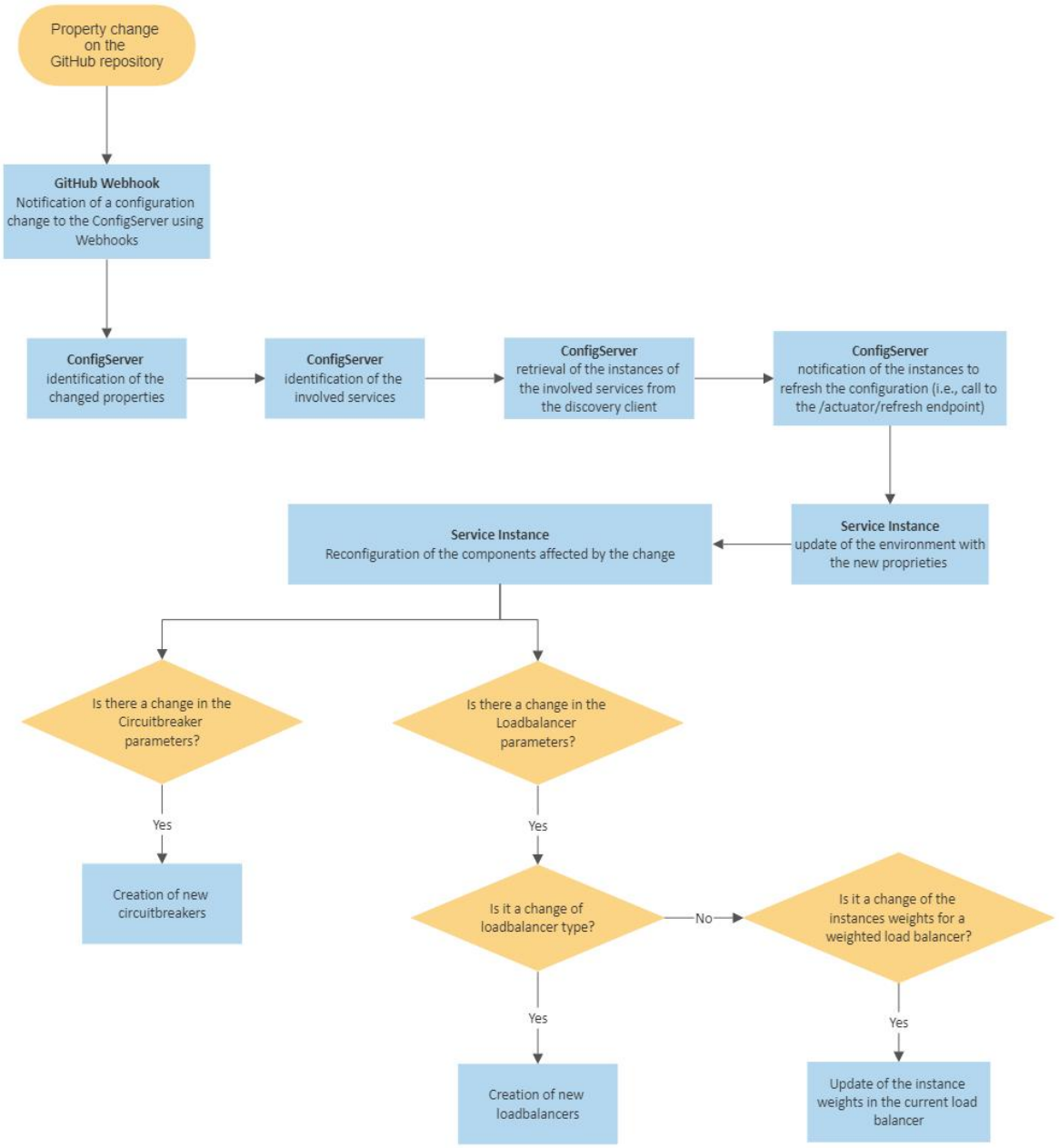
Implementation-specific adaptation scenarios

Scenario	**Observable properties**	**Adaptation example**
Service unavailable	Success or failure of each service invocation or pinging the status. ***Probes***: - `/actuator/prometheus` endpoint - `/actuator/health` endpoint	- Select equivalent service (i.e., use a service offering the same interfaces and providing the same functionalities) - Invoke (load balanced) instances in parallel ***Actuators***: - *The instances manager (to be defined - kubernetes, docker, shell scripts, …)*
QoS not satisfied	QoS indicators such as: - Response time - Uptime - Recovery time - Resource usage - *Any other meaningful QoS indicator* ***Probes***: - `/actuator/prometheus` endpoint	- Select equivalent service (i.e., use a service offering the same interfaces and providing the same functionalities) - Invoke (load balanced) instances in parallel - Shutdown of an “abnormal” instance - Change configuration to improve the QoS indicators (i.e., change of properties on the configuration repository) ***Actuators***: - *The instances manager (to be defined - kubernetes, docker, shell scripts, …)* - `/actuator/shutdown` endpoint - Config Server
Change in QoS requirements (e.g., new non-functional requirement or change of an existing requirement)	QoS requirements specification ***Probes***: - requirements specification source (i.e., where QoS requirements are specified)	- Update of the managing system internal configuration (e.g., which QoS indicators should be observed, cost function, external services score, …)
Cost constraints not satisfied	Variables of the cost function ***Probes***: - `/actuator/prometheus` endpoint - configuration properties	- Select equivalent service (i.e., use a service offering the same interfaces and providing the same functionalities) ***Actuators***: - *The instances manager (to be defined - kubernetes, docker, shell scripts, …)* - Config Server

Change in external services “score” (i.e., change in the rank of equivalent external services)	Properties of the services (e.g., average response time, cost, …) ***Probes***: - `/actuator/prometheus` endpoint of the proxies	- Select equivalent service (i.e., use a service offering the same interfaces and providing the same functionalities) ***Actuators***: - *The instances manager (to be defined - kubernetes, docker, shell scripts, …), to manage the proxies to use* - Config Server (i.e., change of the property indicating the proxy – and of the external service – to use)

Event-Driven Process Diagram for the configuration change



Pattern per migliorare QoS

Client-side load balancing

A client-side load balancing allows an application to distribute the load among the available instances of the service target of the request.
The load balancer obtains a list of available instances from a `ServiceInstanceListSupplier`, which in our case uses a client-side discovery service (i.e., Netflix Eureka). Then, the instance is selected according to a specific load balancing algorithm.
For the purpose of this project, we implemented two algorithms:

- Round-Robin: a client request is forwarded to each server in turn, cyclically.
- Weighted Round-Robin: a variant of the Round-Robin algorithm, where each instance has a specific weight, indicating the number of requests that the instance should accept before the other requests are routed to another instance. This weight measures the degree of robustness and responsiveness of the specific service instance.

A load balancer is used whenever a microservice contacts another one, both in the API Gateway and in the “standard” microservices, where Feign clients are used. OpenFeign is a framework used to build declarative REST clients for Spring Boot apps.
It makes the REST interface of the client service appear as a Java interface through the code.

Circuit Breaker pattern

Due diversi circuit breaker, uno per ciascun proxy, sono presenti all’interno dell’ordering service. Il Circuit Breaker pattern viene utilizzato per sopperire a potenziali guasti dei servizi di terze parti utilizzati, e al tempo stesso evitare ulteriore carico ai servizi di terze parti non funzionanti ed evitare tempi di risposta eccessivi per l’utente finale. Per reagire a tali guasti o errori vengono invocati dei metodi di fallback. In particolare:

- Per il servizio di pagamento, il metodo di fallback consente all’utente di completare l’ordine pagando in contanti alla consegna/ritiro.
- Per il servizio di consegna, il metodo di fallback fornisce all’utente la possibilità di ritirare l’ordine al ristorante.

Lista attuatori

- **GET *INSTANCE_URL*/actuator/shutdown**
  Per chiedere la terminazione di una determinata istanza
- **POST *INSTANCE_URL*/actuator/refresh**
  Per chiedere a una determinata istanza di riconfigurarsi. Lancia un evento del tipo `RefreshScopeRefreshedEvent`
- Modifica dei file di configurazione e **push su**lla **repository GitHub di configurazione**, con conseguente notifica al `Config Server`, che richiede ai microservizi interessati di aggiornare i propri parametri dalla repository.
  I parametri su cui si interviene sono:
  - proprietà dei **Circuit Breaker**:
    - *Sliding Window Size*: dimensione della sliding window in cui conservare l'esito (success/fail) delle ultime n richieste.
    - *Failure Rate Threshold*: soglia che se superata dal failure rate (tasso di chiamate fallite nella window corrente) causa l'apertura del circuito.
    - *Slow Call Duration Threshold:* soglia temporale che definisce la durata massima entro la quale una chiamata non è considerata "slow".
    - *Slow Call Rate Threshold:* soglia che se superata dallo slow call rate (tasso di chiamate considerate lente nella window corrente) causa l'apertura del circuito.
    - *Wait Duration In Open State:* tempo di permanenza nello stato Open prima di transitare allo stato Half-Open.
    - *Permitted Number Of Calls In Half-Open State:* la dimensione della window nello stato Half-Open.

  - proprietà dei **Load Balancer.** Attualmente:
    - tipo di LB da usare per ogni servizio
    - peso di default delle istanze di un servizio (nel caso di un weighted load balancer)
    - peso di una singola istanza di un servizio (nel caso di un weighted load balancer).
  - **indirizzi** dei microservizi "accessori" utilizzati (Eureka Registry Server, MySQL Server e API Gateway), utili nel caso di configurazione distribuita
  - **indirizzi** dei servizi di terze parti per i rispettivi proxy
  - se necessario, una qualsiasi altra proprietà del managed system

Lista probe

- **`GET` *`INSTANCE_URL`*`/actuator/prometheus`**
  Da qui è possibile ottenere, per ciascun microservizio:
  1. stato, parametri e statistiche dei **circuitbreaker** (CB), come:
     Una slow call è una chiamata che supera il tempo di esecuzione specificato nelle proprietà del CB
     a. numero e tempo medio di elaborazione di ogni chiamata passante per il CB, suddivisi in base all'esito {failed, ignored, successful}
     b. numero di chiamate nel buffer quando lo stato è CLOSED e loro tempo medio e massimo di esecuzione, suddivisi per esito {failed, successful} + failure rate
     c. numero di chiamate ricevute quando il CB è APERTO (esito chiamata: "not permitted")
     d. slow call rate
     e. numero di slow call, suddivise per esito {failed, successful}
  2. total, max e average **response time** per ciascuna **chiamata** a ogni endpoint **REST**
  3. (**utilizzo** memoria, CPU e spazio su disco)

- **`GET` *`INSTANCE_URL`*`/actuator/health`**
  - Stato del microservizio (e.g., UP)
  - Stato della connessione al database (dove utilizzato)
  - Stato del discovery client (Eureka Client)