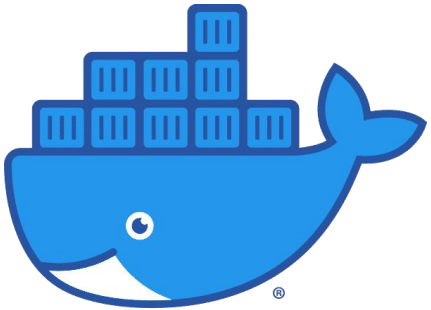
# SAEFA: a Self Adaptive eFood App

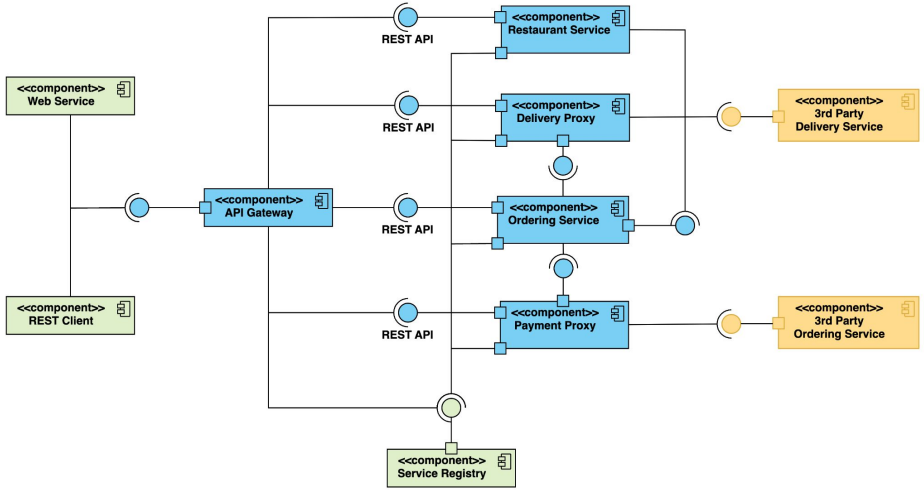
# Framework and Technologies





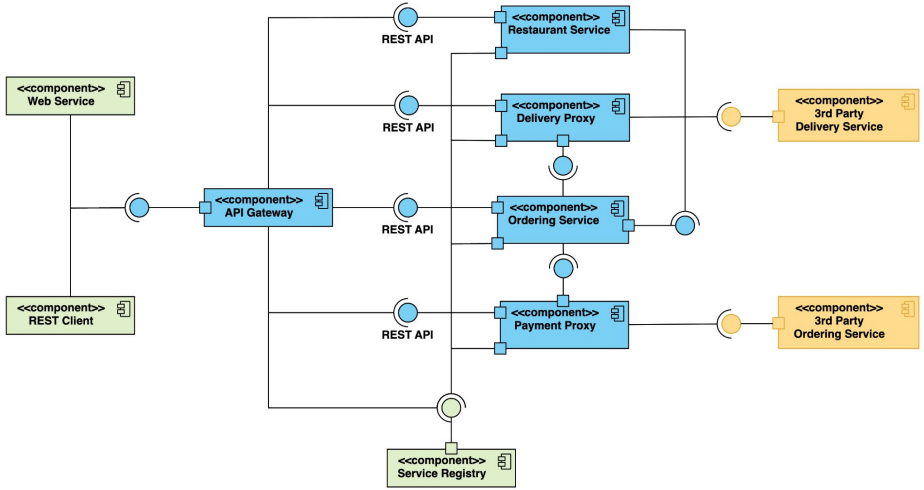
# Microservices architecture

- **Restaurant Service:** servizio utilizzato per la ricerca dei ristoranti e i relativi menù, e per la notifica degli ordini effettuati ai ristoranti stessi.
- **Ordering Service**: servizio utilizzato per la creazione e la processazione degli ordini.
- **Delivery Proxy**: servizio utilizzato per la comunicazione con il servizio di consegna di terze parti.
- **Payment Proxy**: servizio utilizzato per la comunicazione con il servizio di pagamento di terze parti.



# Microservices architecture

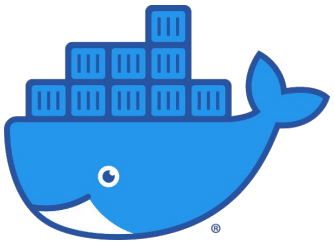
- **Service Registry**: servizio a cui tutti gli altri microservizi si registrano fornendo il loro nome e l'indirizzo a cui sono raggiungibili. Fornisce automaticamente all'API gateway i suddetti indirizzi e, in caso di molteplici istanze dello stesso servizio, effettua un load-balancing di tipo round robin.
- **API Gateway**: servizio di gateway per il filtraggio e il reindirizzamento delle richieste REST con cui si interfacciano i client esterni per interagire con il sistema. I servizi registrati nel Service Registry sono raggiungibili all'indirizzo `indirizzo_gateway/Nome_Servizio`



# Setup attuale

- Service Registry in esecuzione su un container **AWS**, raggiungibile all'indirizzo (statico, noto agli altri microservizi) : http://52.208.38.53:8761/

- I restanti microservizi vengono eseguiti individualmente in singoli container **Docker**. Un'apposita configurazione è necessaria per far sì che i container possano comunicare tra loro ed essere raggiunti dal Service Registry. Tale configurazione è eseguita automaticamente tramite uno script bash di setup.

# Potenziali Attuatori e Probe

Attuatori:

- **CircuitBreaker pattern**: implementato in Spring dal framework Resilience4j. Il Managing System modifica le soglie e i parametri per l'attivazione del meccanismo. Gli scenari individuati prevedono tale attivazione in caso di fault dei servizi di terze parti o dei relativi proxy.

- **External service selection**: la selezione dei servizi di terze parti viene effettuata dal Managing System e comunicata al Managed mediante la modifica di file di configurazione.

- **Load Balancing**: il Managing System effettua load balancing allocando dinamicamente nuove istanze dei microservizi, che iscrivendosi al Service Registry verranno contattati con schedulazione round robin.

# Potenziali Attuatori e Probe

Probe:

- **AspectJ**: framework per il supporto alla programmazione orientata agli aspetti in Java. Il vantaggio è quello di poter creare *wrapper* intorno a metodi selezionati di ciascun microservizio per eseguire codice prima e/o dopo la loro esecuzione. Attualmente usato in combinazione con **Slf4j+Lombok** per il logging dell'esecuzione dei metodi di ciascun microservizio.

- **Prometheus (forse)**: framework per il monitoraggio di parametri quali il numero di richieste elaborate dai servizi e i rispettivi tempi di esecuzione.

- **Actuator**: framework per il monitoraggio customizzabile di parametri del sistema quali utilizzo di ram e cpu, lista di richieste elaborate, tempo medio di elaborazione ecc.

# Installation Guide

Prerequisites: Docker, JDK 16.0+, Gradle, having a public ip address (it is necessary to enable port forwarding from the gateway to the hosting machine for the port interval 58080:58085. Forwarding on port 8761 is needed too if the Eureka service is hosted on the same machine).

**Step 1)** Clone the repo, navigate to the directories `/saefa/[*]-service/src/main/resources/` and open the `application.properties` file.
**Step 2)** Edit the `eureka.client.serviceUrl.defaultZone=` http://your_ip:8761/eureka/ parameter, changing the ip address to the public address of the machine that will run the Eureka Service Registry service.

# Installation Guide

**Step 3)** Navigate to the directory `/saefa/eureka-service-registry` .

- To run the service locally in a docker container, execute the commands `$ bash dockerBuild.sh` and `$ bash dockerRun.sh`. The former will create the Docker container and the latter will start it.
- Instead, to build the .jar file and run it on a different machine, run the command `$ gradle build` in the same directory. The `.jar` file (i.e., the deployable artifact) can be found under `/saefa/eureka-service-registry/build/libs/eureka-service-registry-latest.jar`.

# Installation Guide

**Step 4)** Navigate to `/saefa` and run the command `$ bash setup.sh`. This will automatically build all the services, create one docker container each in the same virtual network and start them. The services will automatically register themselves to the *eureka-service* once up and running. The API gateway can be reached on the port 58080 of the hosting machine.

To build and run the artifacts (i.e., the .jar files), navigate to each microservice main directory and run the command `$ gradle build` in the same directory. The `.jar` file can be found under `/build/libs/*-latest.jar`.

You can overwrite the configuration properties passing them as environmental variables.