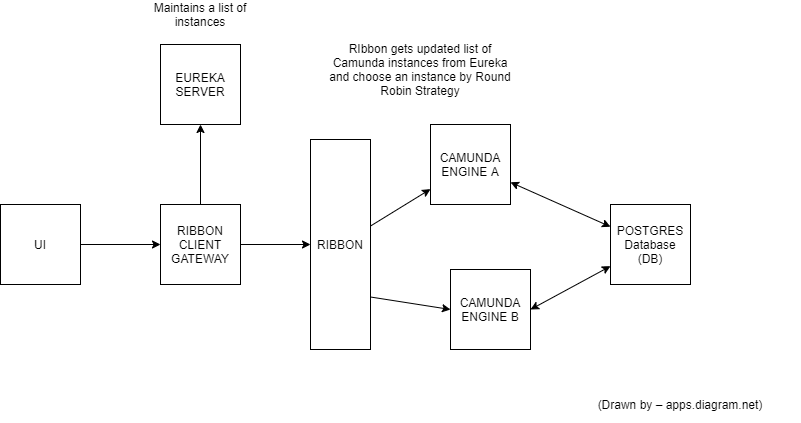
CLIENT SIDE LOAD BALANCING WITH CAMUNDA

This document aims to provide an overview of Client Side Load Balancing with 2 camunda engines having a shared database.

# Architecture Diagram –



# Project details –

* **Microservices** – 4 Microservices are created as a part of this implementation. These are -
  + Eureka Server : Used for Service Registry
  + Camunda Engine A: Eureka registered microservice which provides Spring boot embedded Camunda Engine with Pizza delivery BPM (A mock BPM which will be triggered by Ribbon client gateway)
  + Camunda Engine B: Eureka registered microservice which provides Spring boot embedded Camunda Engine with Pizza delivery BPM (A mock BPM which will be triggered by Ribbon client gateway)
  + Ribbon Client Gateway: Ribbon Client Service which will consume endpoints exposed from Camunda’s REST APIs. Also, contains the default configuration of Ribbon
* **Front End –** An angular application where once user clicks the “order” button, the Ribbon Client gateway is called, which in turn, triggers a BPM process shown in the later parts of the document.

# Working of the load balancer –

* Eureka Server is the Service Registry which maintains the list of all the microservices as KEY-VALUE pair. Key is the serviceId and value is the instance information(host, port)
* The Ribbon-Gateway-Client microservice uses Ribbon as Load balancer and consumes Camunda’s REST Endpoints.
* Ribbon microservice talks to Eureka Server and gets information about instances of Camunda Engines to choose from.
* Then, Using Round Robin strategy (default), Ribbon-client-gateway microservice consumes the Rest Endpoints from the Camunda Engine such that load is balanced
* The same can also be achieved without using Eureka server also but with that we need to manually specify the list of instances of Camunda Engine in Ribbon-client-gateway microservice. But If Eureka is used (preferred approach and the one used in above implementation), there’s no need to manually specify the list of instances of Camunda Engine because it takes the instance information from Eureka using which it can scale up also without manual intervention, if a new instance is registered in the Eureka Service Registry.

# To Test the above implementation-

* Using Front End, Select any Pizza Type and Quantity, and click “Order” button which will trigger the Ribbon-client-gateway’s *“/start”* endpoint, which in turn, triggers Camunda’s Rest Endpoint to start a workflow instance
* If you want to directly trigger the API, trigger the below endpoint –

Method type: GET

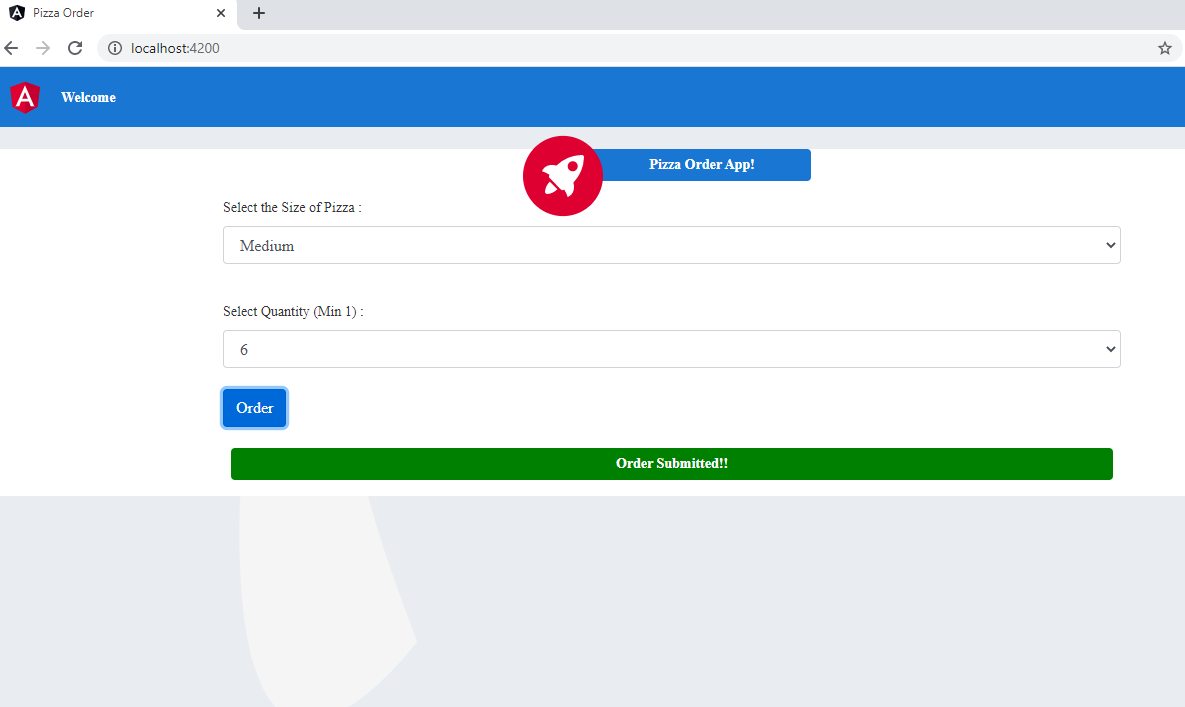
Method Endpoint: [http://localhost:8888/start?key=OrderProcessing&pizzaType={pizzaType}&quantity={quantity}](http://localhost:8888/start?key=OrderProcessing&pizzaType=%7bpizzaType%7d&quantity=%7bquantity%7d)

Response – True, if process Instance started successfully, otherwise false.

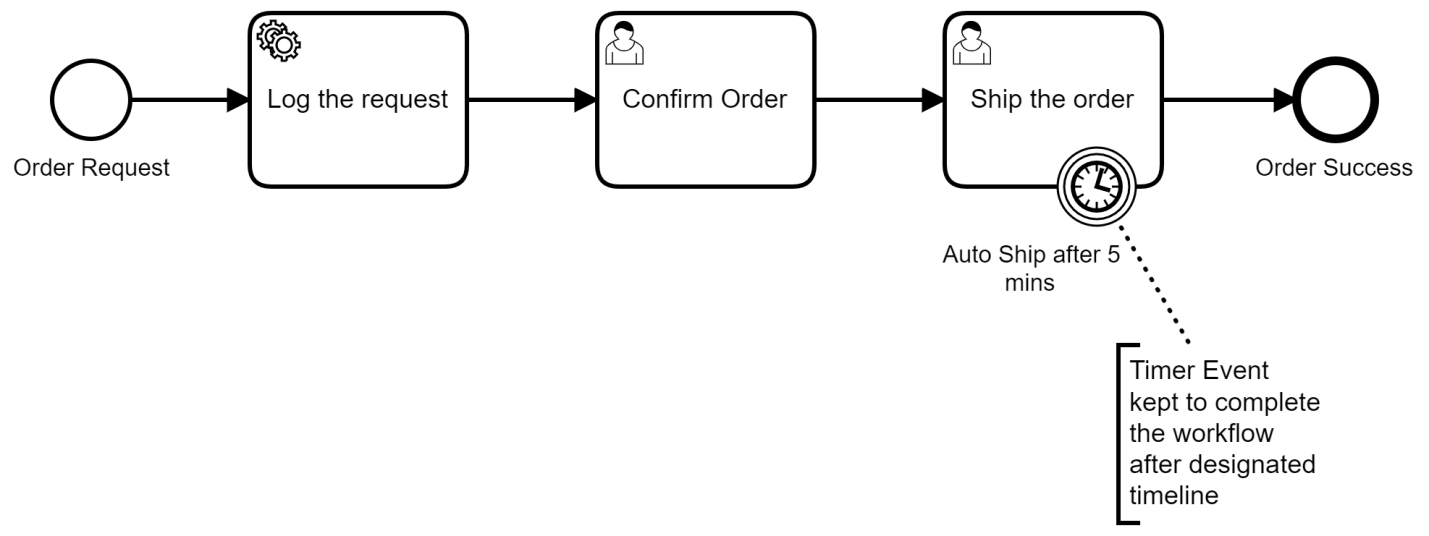
* Now to check if the instance is getting started via engine a or engine b, A special variable has been added to Camunda’s process variables which shows which Engine was used to trigger the instance. (Please refer to the screenshot section). Also, the same is logged at the console level.

# Screenshots –

## Front-End App



## BPMN Diagram



## Screenshot to show that the request is going to both engines

