PRODUCT INFORMATION MANAGEMENT

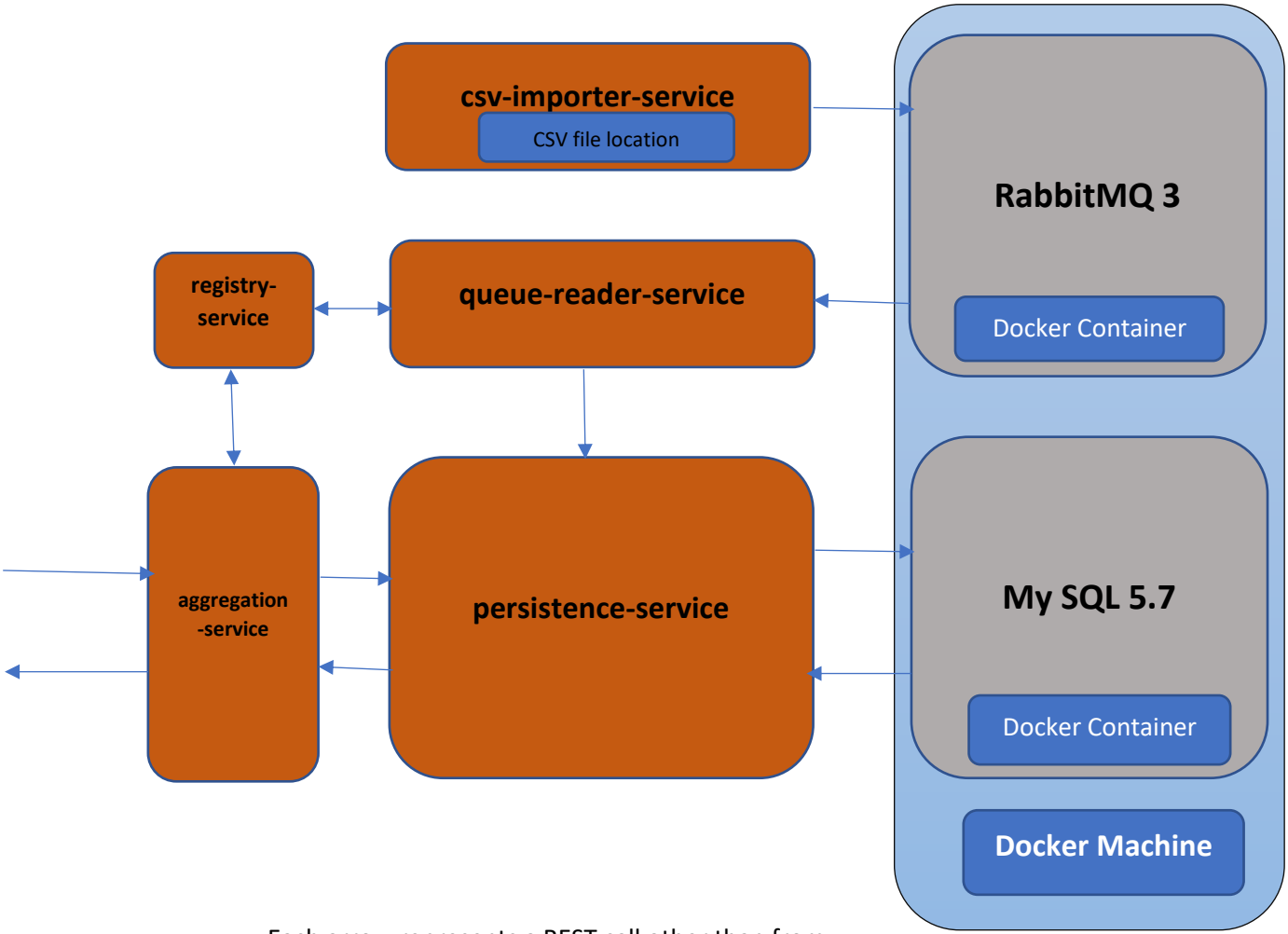
Table of Contents:

1) Purpose of the document

The purpose of this document is to describe the 'Product Information Management' system. The purpose of this system is to save huge volumes of data to the database from CSV files, and to do CRUD operation and to get daily create and update statistics from it.

2) Overall Architecture

Following is the overall architecture of this system.



Each arrow represents a REST call other than from and to RabbitMQ

2.1) registry-service

This is simple Eureka service discovery server.

2.2) csv-importer-service

This is non-web-based spring boot multi-threaded application. One thread keeps checking input directory for csv files. All the csv files that are found, are each assigned to a separate worker thread to parse and post to MQ.

Input directory, number of worker threads, polling interval of csv file searching thread are configurable among other things.

2.3) queue-reader-service

This service reads data from MQ, it is configurable how many listener threads are active. Objects retriever from MQ are sent to persistence service either for create or update. **This service uses EhCache3.4 (JSR 107 complaint) as a request cache. If same date comes from the queue more than once in a defined timeperiod, then it doesn't send that data to persistence service** to be stored in data base, thus effectively reducing the load on persistence service. Object lifespan in cache is configurable.

2.4) persistence-service

This service uses hibernate to communicate with database. It exposes CRUD operation over Product table. It exposes endpoints for other services to interact with it. This service uses RestRepository and it is transactional. This service never deletes any data, it moves data from main table to delete_product table on delete request.

2.5) aggregator-service

This service is exposed to outside world to provide CRUD operation and statistics over product data. **Internally it asynchronously calls persistence service** for its operations. **This service uses a JSR 107 compliant persistent EhCache3.4 to store the data.** Each time any create, update or read request hits this service, it checks cache first, thus substantially reducing the response time.

Read all, daily statistics requests are directly fetched from persistence service to provide caller the fresh snapshot of data. In case of read all, it saves all the data in cache for further use.

All cache related time, asynchronous timeouts and thread count are configurable among other things.

3) Endpoints

URL	Input	HTTP method	Output	Output Type
/product/read/{id}	Uuid string of product	GET	Product details for given id (HATEOAS enabled)	application/hal+json, application/json
/product/read/all	NA	GET	All product details from database (HATEOAS enabled)	application/hal+json, application/json
/product/create	Product data in JSON format	POST	Product details after persistence (HATEOAS enabled)	application/hal+json, application/json
/product/update/	Product data to be updated in JSON format	PUT	Product details after persistence (HATEOAS enabled)	application/hal+json, application/json
/product/delete	Uuid of product	DELETE	NA – resp code 200	NA
/product/current/stat	NA	GET	Number of products created, and number of products updated on current	application/json

4) How to Setup

Following are steps to setup the environment:

4.1) unpack the shared Git bundle using `git clone repo.bundle <repo-name>`

4.2) Import and build all five services available inside folder extracted from step one using gradle

4.3) Open docker tool box or docker for windows terminal and go to folder "docker" inside extracted package.

Run "docker-compose up -d" – this will do the following:

a. Start the RabbitMQ on port 5672
b. Start the RabbitMQ management on port 15672
c. MQ Username – guest
d. MQ Password – guest
e. Start MySQL database on port 3306
f. Crete new instance "db"
g. Create two tables named "product", "delete_product"
h. DB username – user, root
i. DB password - password

4.4) Start "**registry-service" first.** It will start on port 5001.

4.5) Get the docker machine IP. Mention that IP inside application.properties file inside persistence-service. (since DB is running in docker container). Also get the IP address where eureka server is hosted and mention it in application.properties file. Start **"persistence-service"** second. It will start on port 5002.

Application.properties is properly documented to know where to put the IP addresses.

4.6) Open application.properties of **csv-importer-service** and change the following things:

a. `spring.rabbitmq.host=192.168.99.100 <IP address of docker machine from step 4.3>`

open "app.config" and mention following details:

a. **file.input.path=./input <location where CSV files are placed, by default it will be inside input folder in project>**
b. **file.output.path=./output <location where CSV files should go after processing, by default it will be inside output folder in project>**

Start csv-importer-service third

4.7) Open application.properties of **queue-reader-service** and change the following things

a. `spring.rabbitmq.host=192.168.99.100 <IP address of docker machine from step 4.3>`
b. eureka.client.serviceUrl.defaultZone= http://localhost:5001/eureka
<address where eureka server is hosted>

open "app.config" and mention following details:

a. consumer.thread.min=1 <minimum number of consumer threads>
b. consumer.thread.max=10 <maximum number of consumer threads>
c. cache.capacity=5000000 <object capacity of request cache>
d. cache.idle.time=30 <idle time in seconds after which object will be evicted from cache>
e. cache.live.time=60 <alive time in seconds after which object will be evicted from request cache>

Start queue-reader-service as fourth.

4.8) open application.properties file inside aggregation-service and do the following:

a. eureka.client.serviceUrl.defaultZone= http://localhost:5001/eureka
<address where eureka server is hosted>

Open app.config and do the following:

a. async.thread.max=10 <number of threads for async processing>
b. async.call.timeout=1000 <async call timeout for all requests other then read all>
c. async.read.all.timeout=60000 <async call timeout for read all request>
d. data.cache.capacity=500000000 <persistence cache capacity>
e. data.cache.idle.time=120 <idle time in seconds after which object will be evicted from cache>
f. data.cache.live.time=300 <alive time in seconds after which object will be evicted from cache>
g. data.cache.persistence.size=200 <size in MB for disk storage for cache>

Start aggregator-service fifth.