Design Document

For

PRODUCT MANAGEMENT SERVICE

Khandekar Mohammad Saleh

Table of Contents

1. **Introduction**
2. **Service Overview**
   1. **High Level Description**
   2. **Technology Stack**
3. **Service Architecture**
   1. **High Level Architecture**
   2. **Deployment**
4. **API Details**
   1. **Add Product API**
   2. **Retrieve Product API**
   3. **Delete Product API**
5. **Non Functional Requirements**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Changes | Version |
| Initial | 27/02/2018 | First Version | v1.0 |

1. Introduction

This document is created to outline the design for the “Product Management Service”. This document includes a high level design of the system with technical details of each sequence. This document also describes the deployment architecture of the service briefly.

2. Service Overview

2.1 High Level Description

Products are one of the most important building blocks in a big retail application. So, managing product details (add, update and delete) is one of the most important modules of the whole applications. Since, there can be multiple clients of the product management module, it’s better to have it as a separate micro-service which can be managed independently. That’s why in this document we’ll see how the new architecture visualizes the product management system as a simple yet very efficient micro-service.

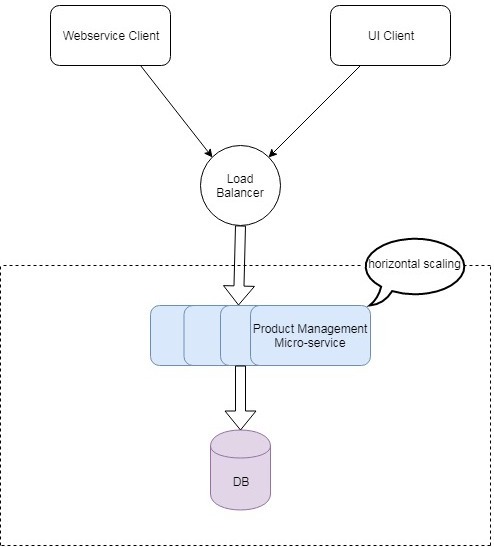
This service provides 3 simple RESTful APIs:

* Add a new Product.
* Retrieve all products depending one simple predicate.
* Remove a product from a catalogue.

2.2 Technology Stack

3. Service Architecture

3.1 High Level Architecture



3.2 Deployment [TBD]

The deployment architecture is out-of-scope for v1.0. But below is the proposed architecture of the same:

4. API Details

4.1 Add Product API

URL: http://<host>:<port>/products

HTTP Method: POST

Request Body:   
{    
   **"createdOrUpdatedBy"**:"string",  
   **"productDescription"**:"string",  
   **"productName"**:"string",  
   **"productType"**:"string"  
}

Response Body:

{    
   **"productDescription"**:"string",  
   **"productId"**:"string",  
   **"productName"**:"string",  
   **"productType"**:"string"  
}

4.2 Retrieve Product API

URL: http://<host>:<port>/products/{product\_type}

HTTP Method: GET

Response Body:

[{    
   **"productDescription"**:"string",  
   **"productId"**:"string",  
   **"productName"**:"string",  
   **"productType"**:"string"  
}]

4.3 Delete Product API

URL: http://<host>:<port>/products/{product\_id}

HTTP Method: DELETE

Response Body:

{}

5. Non Functional Requirements

Monitoring: For single node mentoring we can use default actuator support in Spring Boot. But for cluster monitoring, we have to go for a more sophisticated solution like Grafana.

Service Discovery and Load Balancing: If we use AWS EC2 cluster, we can use ELB for load balancing.