Pricing System Design Document

1. Introduction

1.1 Purpose

To build a system that hosts Netflix pricing which will enable Netflix to systematically change prices across all its global customers.

1.2 Scope

This document describes the implementation details of the Pricing System Application. The system primarily supports two mainly functions: 1) price changes per service plan by country and 2) access to a given customer that includes the current service plan charges applied for that customer.

2. Design Overview

2.1 Problem Description

As a global service, Netflix has prices defined for each service plan for country it supports. That means, price changes are rolled out for service plans by country. The system also needs to consider the pricing roll-outs for a future date.

2.2 Technologies Used

The system will be a Spring Boot Java application and Cassandra 3.11.3 will be used as the backend data store.

2.3 System Architecture

Figure 1. Depicts a high level system architecture. The system will be constructed from multiple distinct components:

- Load Balancer - The component responsible for distributing load across the cluster
- Authentication / Authorization - The security layer to ensure only valid and authorized users have access to the system and the data
- Cache - A fast access store to help improve request read performance and reduce load on the application and data layer for repeated requests
- Request Processor - The main application that controls and processes the requests and is responsible for talking with the database layer for storage
- Cassandra - The interface for storing, importing, and exporting the data model and raw collected data. It can scale to a very large number of customers and data

Note: As part of this exercise, only the components drawn in blue are implemented.

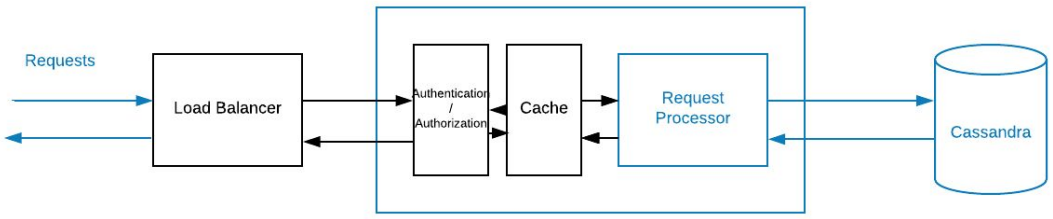

Figure 1: High level Design of the Pricing Application

3. Data Model and Storage

The ‘Customer’ table stores all the global customer information such as country, selected service plan, and the current price for the chosen plan. The ‘CustomerByServicePlan’ table stores all (including historical and future) price updates to the service plan for each of the customers. The ‘effectivedate’ indicates the date when the price for the corresponding service plan would go into effect. The table is partitioned by the ‘service plan’ and ‘country’ and clustered by ‘customer id’ and ‘effective date’. This partitioning structure helps limit the requests to a single partition in most cases thereby improve overall query performance.

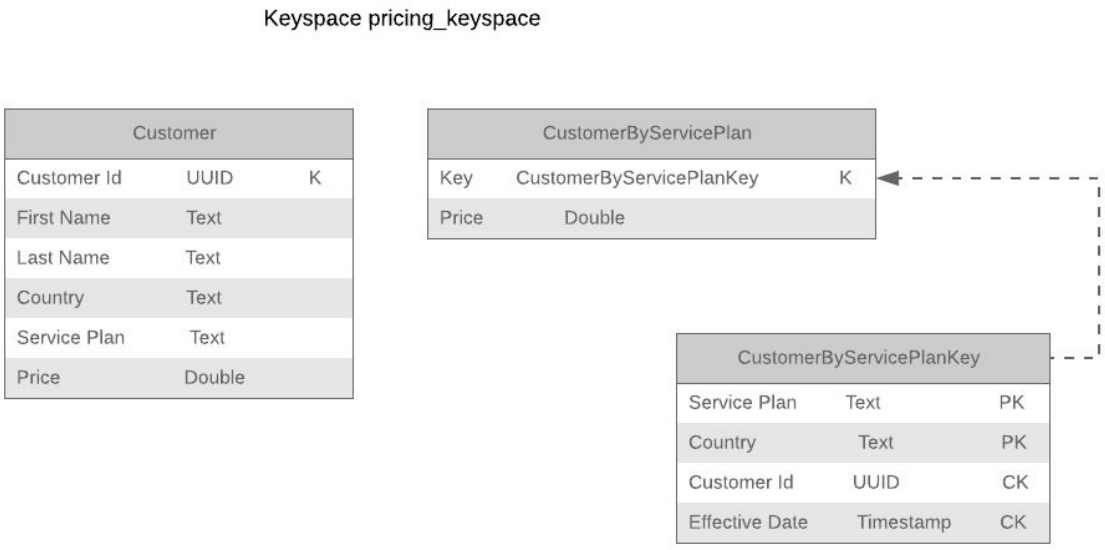

Figure 2: Pricing System Data Model

4. APIs

The following are few of the main APIs supported by this system apart from the basic CRUD apis for the customer model.

1. To update prices -
   Request type: PATCH
   URL: /customers/byserviceplan/{serviceplan}/bycountry/{country}/updateprice
   Request Body: {“price” : xxx.xx, “effectivedate”: “YYYY-mm-DD”}. If “effectivedate” is not provided, current date is picked by default.
2. To get all service plans by country -
   Request type: GET
   URL: /customers/byserviceplan/{serviceplan}/bycountry/{country}
3. To get a specific customer and check the updated price -
   Request type: GET
   URL: /customers/{id}
4. To get all customers
   Request type: GET
   URL: /customers

Sample queries:

1. If the price pushed is effective current date, then all the customers registered for the service plan and country will have their pricing updated with the new price
   Eg:
   Curl -H “Content-type: application/json” --request PATCH -d ‘{“price”: 57.0}’ http://localhost:8080/customers/byserviceplan/3S/bycountry/CAN/updateprice

   Curl --request GET http://localhost:8080/customers/**c37d661d-7e61-49ea-96a5-68c34e83db35**

   Issuing the GET request on the user with ID “c37d661d-7e61-49ea-96a5-68c34e83db35”, in country CAN registered for 3S service plan, should change the price to 57.0

2. If the price pushed is effective on a future date, then all the customers registered for the service plan and country continue to be charged the current price until the new price comes into effect
   Eg:
   Curl -H “Content-type: application/json” --request PATCH -d ‘{“price”: 58.0, “effectivedate”: “2018-12-12”}’ http://localhost:8080/customers/byserviceplan/3S/bycountry/CAN/updateprice

Curl --request GET
http://localhost:8080/customers/**c37d661d-7e61-49ea-96a5-68c34e83db35**

Issuing the GET request on the user with ID “c37d661d-7e61-49ea-96a5-68c34e83db35”, in country CAN registered for 3S service plan, shouldn’t change the price to 58.0. The customer should continue to have current price of 57.0.

5. Future

The components drawn in ‘black’ in Figure 1 need to be built in order to address different aspects of the system such as request load and help scale by introducing load balancers, cache to improve read performance, and ensure security of the system by incorporating authentication and authorization techniques such as SSO.