# Pricing Service - Design Document

[Pricing Service - Design Document 1](#_Toc519364622)

[Assumptions 2](#_Toc519364623)

[Functional Requirements 2](#_Toc519364624)

[Non- Functional Requirements 2](#_Toc519364625)

[High Level Architecture 2](#_Toc519364626)

[Pricing service 3](#_Toc519364627)

[Price-Updater 3](#_Toc519364628)

[CustomerPriceUpdater 3](#_Toc519364629)

[Rules 3](#_Toc519364630)

[Workflow/ sequence diagram 3](#_Toc519364631)

[SCALABILTY 4](#_Toc519364632)

[API Details 4](#_Toc519364633)

[Countries 4](#_Toc519364634)

[Customers 5](#_Toc519364635)

[Service plan info 8](#_Toc519364636)

[Database Schema 9](#_Toc519364637)

[Technologies/Frameworks/Libraries used 10](#_Toc519364638)

[Test cases tested 11](#_Toc519364639)

## Assumptions

1. Netflix services are available in multiple countries
2. Netflix has 3 service plans - Basic(1s), Normal(2S) and Premium(4S) based on number of concurrent streams
3. Each service plan has a different price for different countries.
4. Each Customer belongs to one country and uses one service plan.

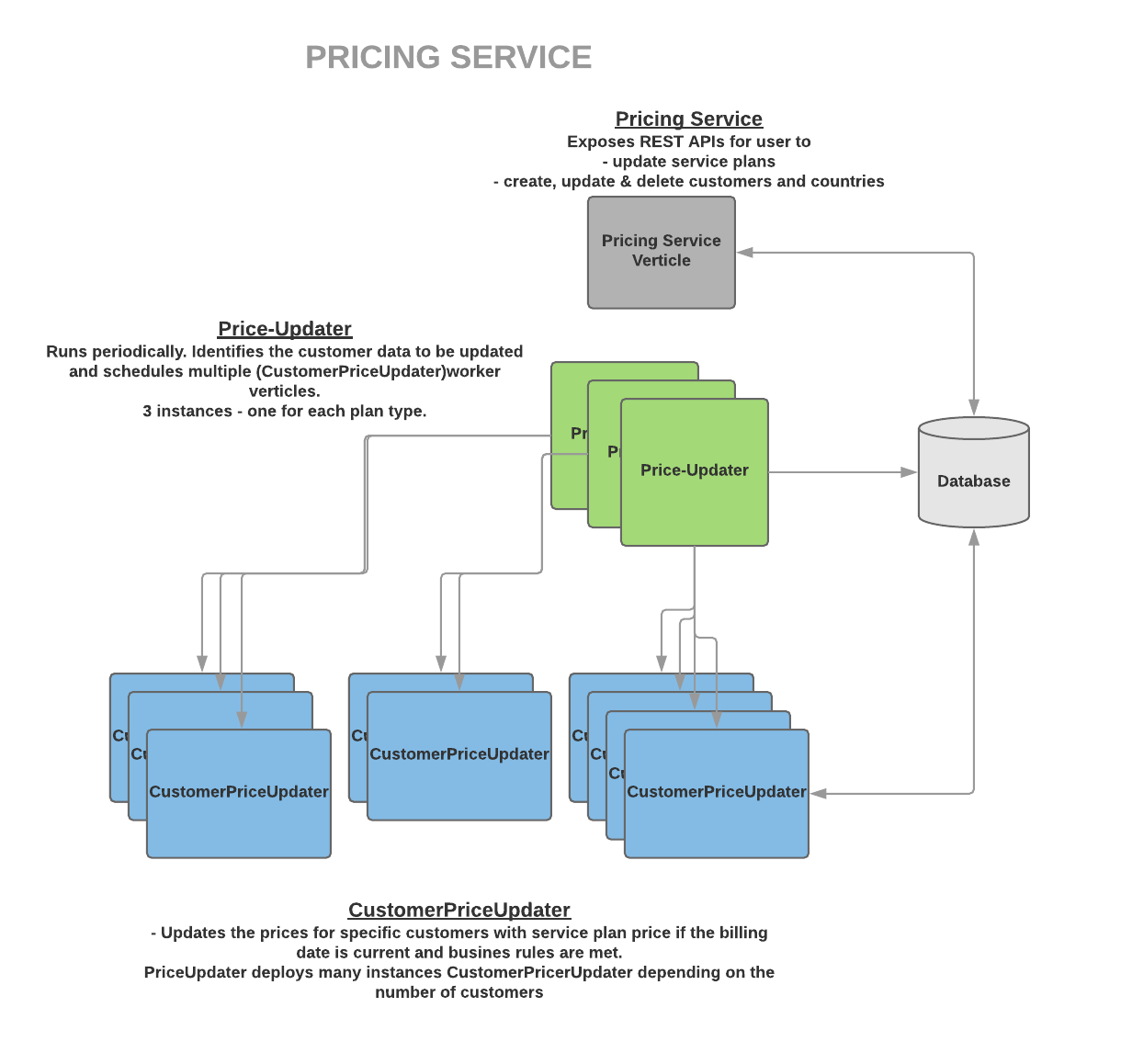
## Functional Requirements

1. User(Netflix Admin) is able to update the price of service plan for a country
2. New Price change will be effective for the customer on the next billing date. This might not be the case for all customers. There may be exceptions based on some business rules

## Non- Functional Requirements

* Service has to be scalable and efficient enough to support 100M+ customers

## High Level Architecture

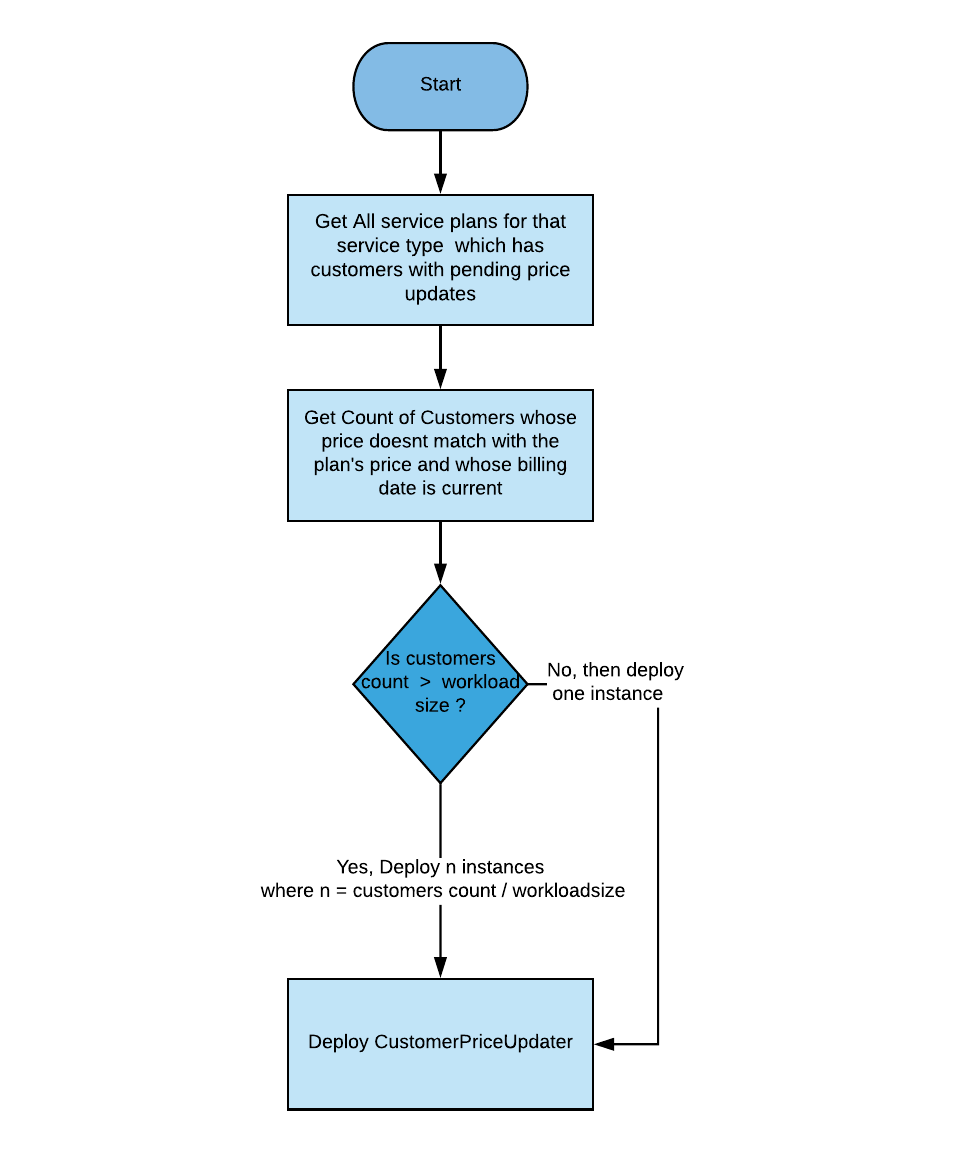


### Pricing service

This service is deployed as vertx-web verticle which exposes REST APIs

* To create, update and delete countries.
* Provides an update service plan API which can be used to update the price of specific plan for a country or update the business rules for the plan.
* To create, update and delete customers. When customers are created the price is automatically retrieved from the service plan’s price for that country.

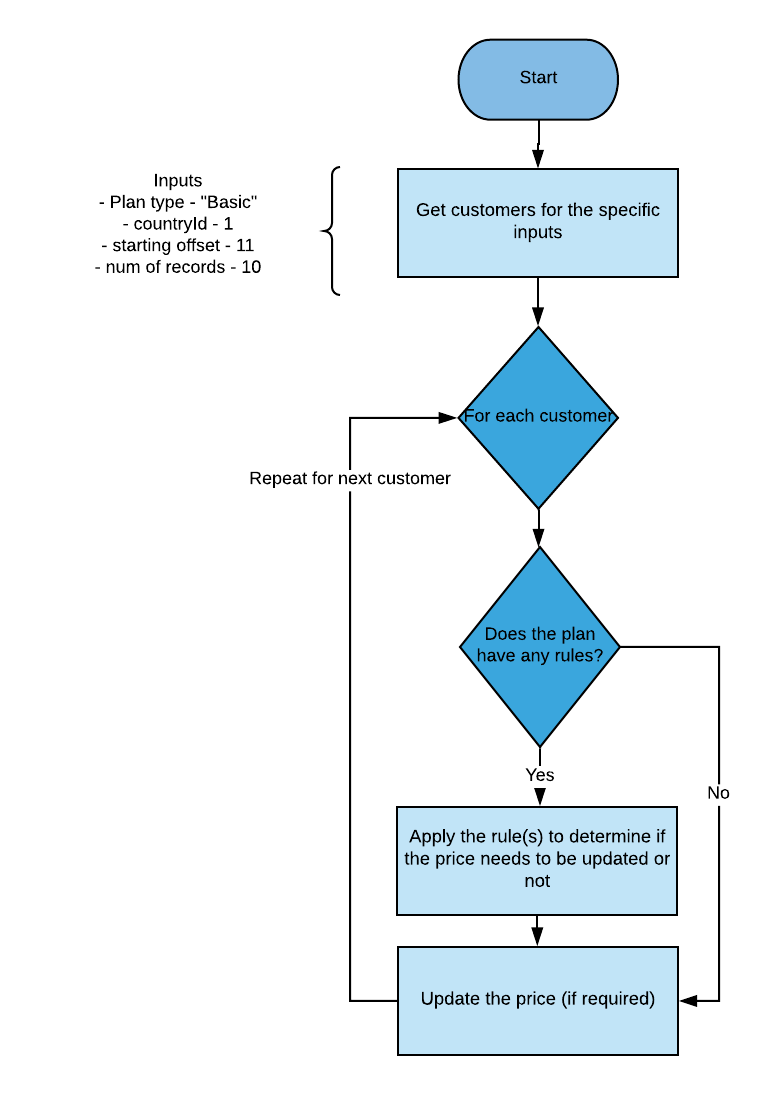
### Price-Updater



This service is also deployed as verticle which runs periodically. There are 3 instances of this service deployed one for each service plan (Basic(1s), Normal(2S) and Premium(4S)). Identifies all the service plans for which there are some customers whose price needs to be updated. For each service plan, identifies the number of customers whose billing date is current. If the number of customers are greater than the workload size, splits the count and deploys as many CustomerPriceUpdater verticles as required. For example: number of customers is 30 and workload size is 10 then 3 instances of CustomerPriceUpdater verticle will be deployed with the following config

* country id,
* service plan type
* starting offset of customer &
* number of customers to process

### CustomerPriceUpdater



This service is triggered by PriceUpdater and it receives the above mentioned inputs. Queries the exact set of customers and for each customer checks if the price needs to be updated or not based on the business rules mentioned in service plan. If the rules match then the price is updated, otherwise it is not. If all the customers of the service plan has been processed, then the priceUpdateCompleted flag in Service\_plan\_info table is updated to true.

This verticle is marked completed once it completes its task. This is Worker thread verticle and it uses once of the thread from existing worker thread pool.

### Rules

Each Service plan can have many rule names. Rules are used to provide exceptions for price updates for certain customers. There are 2 rules available :

* new\_customer\_rule – If the customer’s signup date is less than 6 months, then customer’s price will not be changed until he reaches 6 months.
* loyal\_US\_customer\_rule – If the customer has been with Netflix for more than 5 years, then price will be 2 dollars less than new price or current price whichever is greater.

**Note:** - As of now rules, can only be programmatically created. But, this can be changed in the future in way that user can provide the rules using APIs and that can be applied. Depending on the complexity of the rules, we can decide to use existing rule engines or develop our own rule language and engine by using DSL/ ANTLR etc.,

## Scalability

This design is flexible to be scaled both vertically and horizontally.

Based on the infrastructure available and using the appropriate workload size and worker thread pool size. We will be able to increase/decrease the number of verticle instances on the same machine.

With minor changes, we will be able to deploy these services in multiple machines thus enabling us to horizontally as well.

## API Details

### Countries

#### POST /countries – Create Country

##### Sample Input:

{

"name": "Russia",

"code": "RUS",

"currency": "Ruble"

}

##### Sample response:

{

"id": 6,

"name": "Russia",

"code": "RUS",

"currency": "Ruble"

}

#### PUT /countries/{id} – Update Country by ID

##### Sample Input:

{

"name": "Russia",

"code": "RUS",

"currency": "Ruble"

}

##### Sample response:

{

"id": 6,

"name": "Russia",

"code": "RUS",

"currency": "Ruble"

}

#### GET /countries – Get ALL Countries

##### Sample response:

[

{

"id": 1,

"name": "United Kingdom",

"code": "UK",

"currency": "Euro"

},

{

"id": 2,

"name": "India",

"code": "IN",

"currency": "INR"

},

{

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

}

]

#### GET /countries/{id} – Get Country by ID

##### Sample response:

{

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

}

#### DELETE /countries/{id} – Delete Country by id

##### Sample response:

{

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

}

### Customers

#### POST /customers – Create customer

##### Sample input:

{

"name":"John Doe",

"email":"johndoe@gmail.com",

"planType":"premium",

"country\_id":1,

"signUpTimestamp": 1507953846

}

##### Sample response:

{

"id": 1,

"name": " John Doe",

"email": " johndoe@gmail.com ",

"planType": " premium ",

"price": 5.99,

"country": {

"id": 1,

"name": "United Kingdom",

"code": "UK",

"currency": "Euro"

},

"signUpTimestamp": 1507953846,

"nextBillingTimestamp": 1533063000

}

#### PUT /customers/{id} – Update Customer by ID

##### Sample input:

{

"name":"John Doe",

"email":"johndoe@gmail.com",

"planType":"premium",

"country\_id":1,

"signUpTimestamp": 1507953846

}

##### Sample response:

{

"id": 1,

"name": " John Doe",

"email": " johndoe@gmail.com ",

"planType": " premium ",

"price": 5.99,

"country": {

"id": 1,

"name": "United Kingdom",

"code": "UK",

"currency": "Euro"

},

"signUpTimestamp": 1507953846,

"nextBillingTimestamp": 1533063000

}

#### GET /customers – Get ALL Customers

##### Sample response:

[

{

"id": 1,

"name": "morethan6mnths",

"email": "morethan6mnths@gmail.com",

"planType": "premium",

"price": 3.99,

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

"signUpTimestamp": 1507057200,

"nextBillingTimestamp": 1509649200

},

{

"id": 2,

"name": "morethan5yrs",

"email": "morethan5yrs@gmail.com",

"planType": "premium",

"price": 6.99,

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

"signUpTimestamp": 1310616246,

"nextBillingTimestamp": 1313208246

}

]

#### GET /customers/{id} – Get Customer by ID

##### Sample response:

{

"id": 1,

"name": "morethan6mnths",

"email": "morethan6mnths@gmail.com",

"planType": "premium",

"price": 3.99,

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

"signUpTimestamp": 1507057200,

"nextBillingTimestamp": 1509649200

}

#### DELETE / customers/{id} – Delete Customer by ID

##### Sample response:

{

"id": 1,

"name": "morethan6mnths",

"email": "morethan6mnths@gmail.com",

"planType": "premium",

"price": 3.99,

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

"signUpTimestamp": 1507057200,

"nextBillingTimestamp": 1509649200

}

### Service plan info

#### PUT /servicePlans – Create or Update a service plan

##### Sample input:

{

"country\_id" : 3,

"planType": "premium",

"price": 8.99,

"rules":"loyal\_US\_customer"

}

Sample response:

{

"id": 3,

"planType": "premium",

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

"price": 8.99,

"rules": "loyal\_US\_customer"

}

#### GET / servicePlans – Get ALL Service Plans

##### Sample response:

[{

"id": 3,

"planType": "premium",

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

"price": 8.99,

"rules": "loyal\_US\_customer"

}]

#### GET / servicePlans /{planType}/country/{id} – Get A Service plan for a Plan Type and Country

##### Sample response:

{

"id": 3,

"planType": "premium",

"country": {

"id": 3,

"name": "United States",

"code": "US",

"currency": "USD"

},

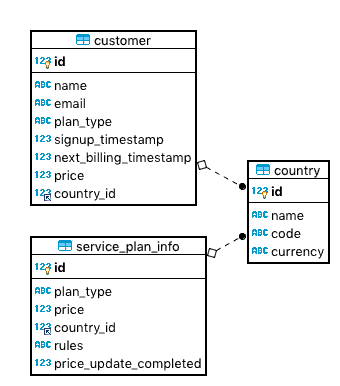
"price": 8.99,

"rules": "loyal\_US\_customer"

}

## 

## Database Schema

Pricing service consists of three tables – Country, customer and Service\_plan\_info. 

#### Country

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Type** | **System\_populated** | **Description** |
| id | integer | Y | Unique identifier |
| name | string | N | Country name (ex – United States) |
| Code | String | N | Country code (ex – US) |
| Currency | String | N | Currency (ex – USD) |

#### Customer

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Type** | **System\_populated** | **Description** |
| id | integer | Y | Unique identifier |
| name | string | N | Customer name |
| email | String | N | Customer email |
| plan\_type | String | N | Service plan type can have the following values - basic(1S), normal (2S) & premium (4S), where S denotes the number of concurrent streams |
| Signup\_timestamp | long | N | Date at which the user signed up for a subscription. Unit in seconds(epoch timestamp) |
| Next\_billing\_date\_timestamp | long | Y | Populated by the system on creation. Signup\_timestamp + 30 days. Unit in seconds(epoch timestamp) |
| Price | float | Y | Price is populated by the system on creation. Value is retrieved from the service\_plan\_info table for the same plan\_type and country\_id. This value is further updated when the service plan price changes. |
| Country\_id | integer | N | Each customer belongs to one country. Foreign key points to the ID field in country table |

#### Service\_plan\_info

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Type** | **System\_populated** | **Description** |
| id | integer | Y | Unique identifier |
| plan\_type | string | N | Service plan type can have the following values - basic(1S), normal (2S) & premium (4S), where S denotes the number of concurrent streams |
| price | float | N | Price of the subscription |
| Country\_id | integer | N | Each service plan belongs to one country. One country has 3 service plans. Foreign key points to the ID field in country table |
| rules | string | N | Business Rule names that has to be applied for the service plan. Can have more than one rule name in comma separated format  Ex - loyal\_US\_customer, new\_customer\_rule |
| Price\_update\_completed | boolean | Y | If the service plan price has been updated to all the customers than this fields is set to true(1) otherwise false(0) |

## Configuration

|  |  |
| --- | --- |
| Property name | Description |
| db.host | Database host name |
| db.port | Database port number |
| db.name | Database name |
| db.user | Database username |
| db.password | Database password |
| app.port | Port number at which the pricing-service which exposes REST APIs is exposed |
| priceupdater.job.time | Periodic time interval for the price updater job to get executed. Units in seconds |
| updater.workload.size | Number of records to be processed by the CustomerPriceUpdater verticle |

## Technologies/Frameworks/Libraries used

1. vertx-web
2. vertx-core
3. vertx clustering using apache ignite
4. mysql database
5. jeasy-rules

## Test cases tested

1. Create/Update/Get/Delete Countries
2. Create/Update/Get/Delete Customers
3. Create a customer with signup date as current date. Verify if the next billing date is 30 days from today
4. Update service plan with new price – verified that the REST API works
5. Create service plan for country US and for plan basic with price 5.99. Create a customer for plan\_type basic. – Verify the customer price is 5.99
6. Create a customer whose next billing date is today for country US with basic service plan. Update the price to 6.99 for basic service plan for country US – verify if CustomerPriceUpdater has updated the price to 6.99 and check if the priceUpdateCompleted flag is set to true in the serviceplan table.
7. Create a customer whose
   * Sign up date is 1 month from now
   * next billing date is today
   * for country US
   * with basic service plan

Update the basic service plan price for country US to 7.99 with new customer rule. – verify that the customer price is NOT updated

1. Create a customer whose
   * Sign up date is 7 month from now
   * next billing date is today
   * for country US
   * with basic service plan

Update the basic service plan price for country US to 7.99 and with new customer rule. – verify that the customer price is updated

1. Create a customer whose
   * Sign up date is 6 years from now
   * next billing date is today
   * for country - US
   * with basic service plan
   * current price – 7.99

Update the basic service plan price for country US to 10.99 with loyal\_US\_customer\_rule. – verify that the customer price is updated to 8.99

1. Create a customer whose
   * Sign up date is 7 month from now
   * next billing date is today
   * for country US
   * with basic service plan
   * current price is 7.99

Update the basic service plan price for country US to 10.99 with loyal\_US\_customer\_rule. – verify that the customer price is updated to 10.99

1. Create a customer whose
   * Sign up date is 6 yrs from now
   * next billing date is today
   * for country US
   * with basic service plan
   * current price is 8.99

Update the basic service plan price for country US to 9.99 with loyal\_US\_customer\_rule. – verify that the customer price is 8.99

1. verify if there are 3 instances of priceUpdater running at the periodic interval specified in the config.properties file. One for each plan type.
2. Create 30 customers
   * For country US
   * With basic service plan
   * Next billing date is today

Update the basic service plan price for country US to 11.99. workload size is 10– verify if there are 3 instances of customerPriceUpdater running and 1st instance processes first 10 records(0-10), 2nd instance process next 10(11-20) and then 3rd instance processes next 10(21-30). Make sure that same customer record is not processed by 2 instances.

1. Create a customer for country US, with service plan basic and billing date is 2 days from now. Update the price for basic service plan in country US. – verify that the customer price is NOT updated