Bonotel XML Revamping

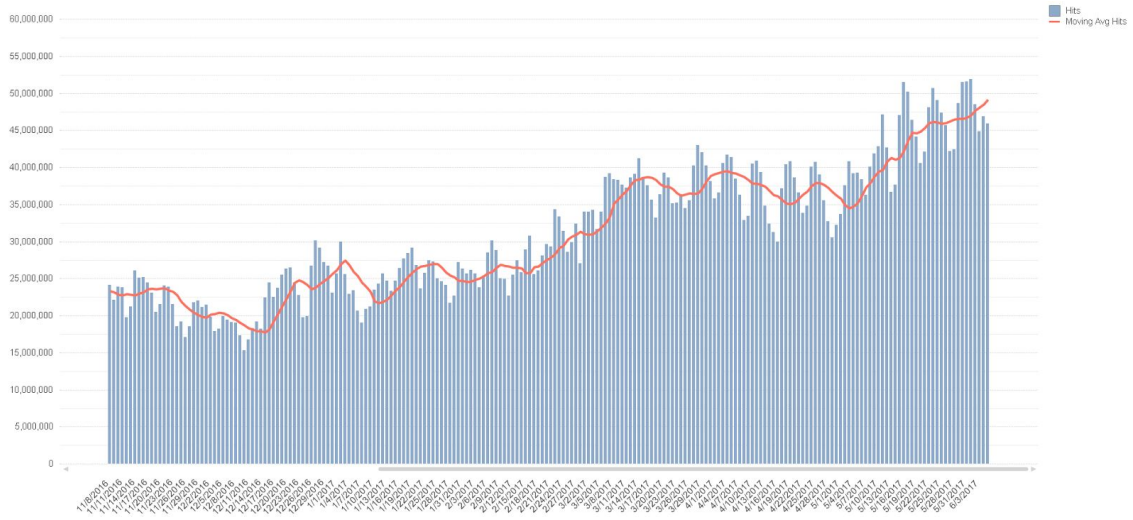
Version 1.0.0

Revisions

Version	Author	Modified On	Description
1.0.0	Varatharaja Kajamugan	26/06/2017	Initial Draft

1. Introduction

The objective of this document is to highlight the limitation in the current application and propose a reengineered solution that will cater for future growth with greater scalability and improved performance. The proposed architectural revamp will cater for increased traffic while maintaining persistent stability throughout the system. The architectural revamp primarily focuses on the development of micro architectural design principles thus allowing for modularized refactoring of the existing monolith Web App.



Traffic Pattern

1.1. Related Issues

BOCLNT-6002 : Revisit table structure & index
BOCLNT-8405 : Traffic Increase and System Capacity - Time to scale
BOCLNT-6291 : Re design static data refresh module
BOCLNT-6294 : Removing struts & using simple servlet model

1.2. Existing Limitations

The following limitations were identified in the corresponding layers of the current application :

Infrastructure

- **Application server resource utilization**

Current application is not fully utilising the J2EE stack, which is provided by JBoss Application server. Due to this, JBoss is consuming more resources, which is really not required by the application.

Data Layer

- **Process intensive DB business logic (PLSQL Logic)**

The XML application implements process intensive (time consuming) business logics within the Postgres DB functions and stored procedures; thus limiting the application's scalability. It also requires DB expertism when refactoring and reengineering PLSQL logic.

- **Limitations in single data structure**

In the current application, all the services are using a single data structure (table structure). Due to this, changes in one service impacts other services.

- **Limitations in existing static data structure**

Current static data structure has limitations such as, inconsistency in content refreshing process as well as high JVM memory utilization.

Business Layer

- **Feature specific code structure**

In current code most of the business objects are feature specific with less abstraction. This limits the reusability of features and components.

- **Increased response time**

It has been noticed that increasing number of hotels cause increased response time.

Testing

- **Insufficient usage of unit/integration testing frameworks**

Current application code has insufficient unit/integration tests, which impacts efficiency and quality.

Deployment

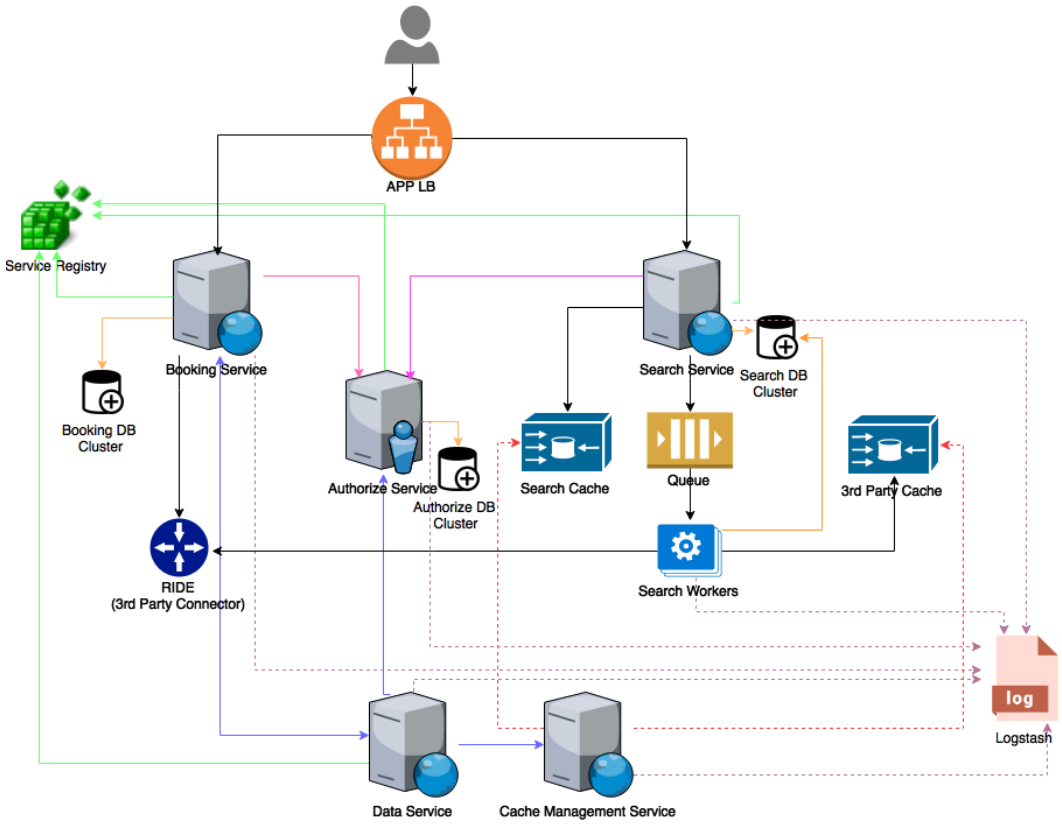
- **Deployment complexity**

Interdependent modules makes deployment complexed. Since entire system is encapsulated as one, even a small customization requires a full deployment across all servers.

2. Solution

This section will describe the solution to overcome above limitations. Target architecture is designed based on service oriented. Based on operations & domains, system is divided into number of macro services.

2.1. Target Architecture



Search Service

Search Service does Search XML parsing & maps expensive Searches (eg : multi hotels search or city search) into number of lightweight requests (hotel searches). Search Service connects with Authorize Service for user authentication/authorisation. Broken lightweight requests will be sent to Queue to process by Search workers. Once they are processed, the responses will be consumed back by Search Service & aggregated response will be sent back to User.

Search Workers

Search Workers consumes the lightweight searches from the queue, process them & put back the response of each searches into Queue.

Booking Service

Booking Service provides Reservation, Modification, Cancellation & Lookup functionalities. Booking Service connects with Authorize Service for user authentication/authorisation

Authorize Service

Authorize Service provides user authentication & token validation features. Search Service or Booking Service connect to this service for user authentication & other services connect to this for validate the tokens.

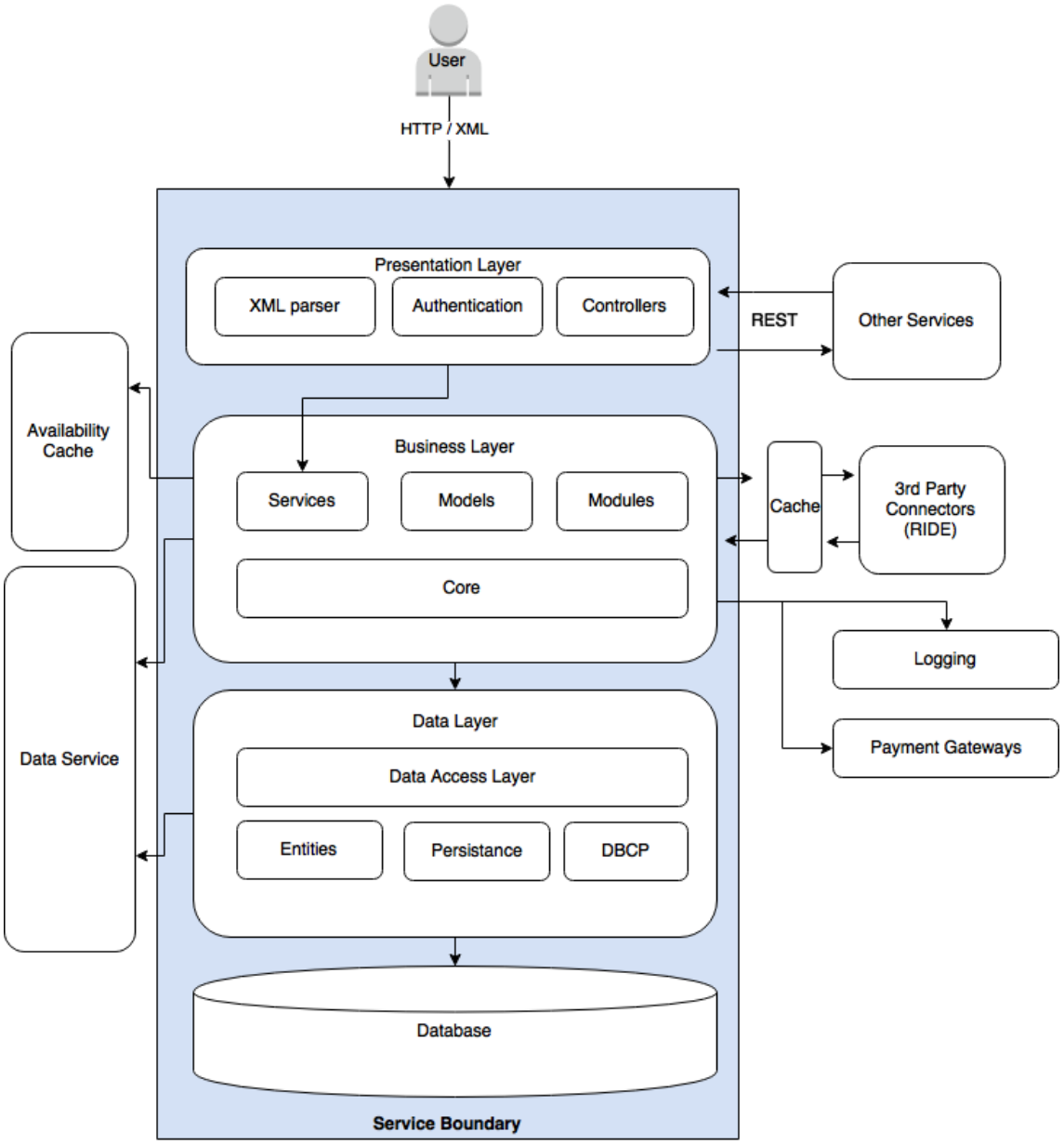
Data Service

When we break the monolithic architecture into number of services, each services will have their own databases. Data Service is to provide data consistencies among the services. Data Service Component describes the design for this service.

Cache Management Service

Cache Management Service is used to manage the availability cache with purging the caches when data changes & dynamically change the caching algorithm. Cache Service Component describes the design for this service.

2.2. Application Architecture



Availability Caching

This will cache the availability processed responses based on the key value (key will have request parameters) & serve them when the similar requests come. Cache Management Service provide data consistency on the cached results, where this will purge the cache, if corresponding data got changed.

Service Boundary

Service Boundary defines the layers & components each services mandatorily have. Components resides outside of the boundary, can be optional for services based on their requirement.

Security Component

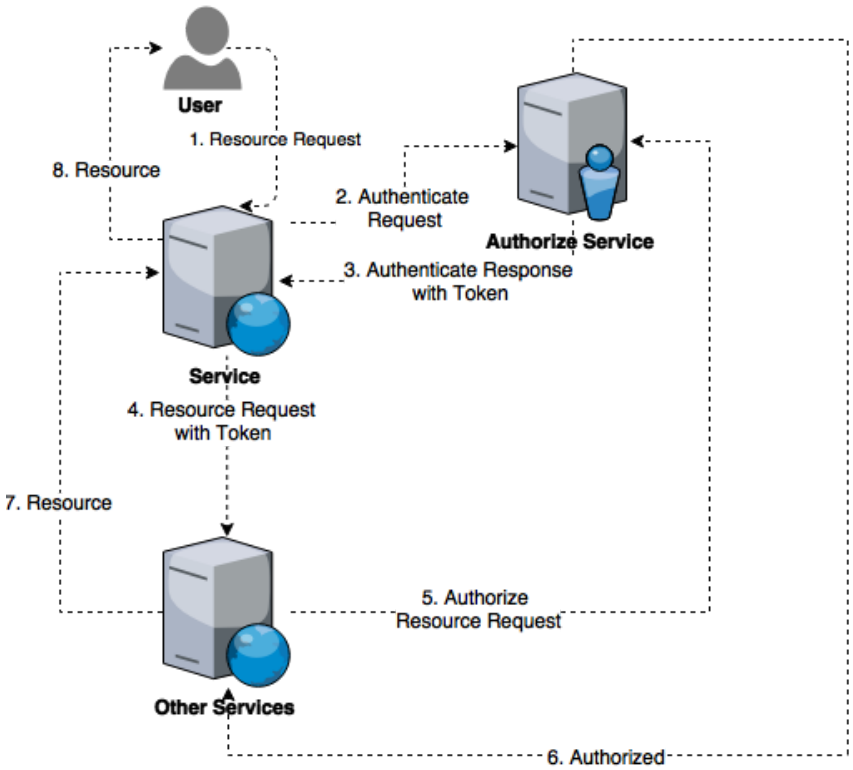
Each services will have Security Component, which communicate with Authorize Service to validate access token passed by other services.

Core

Core component will have set of Services (Java interfaces), Modules, Models & Utility libraries for particular services. This component is reusable and extendable component for any custom features of the service. Each service will extend this component by developing customized Services (Java interfaces), Modules & Models.

2.3. Components

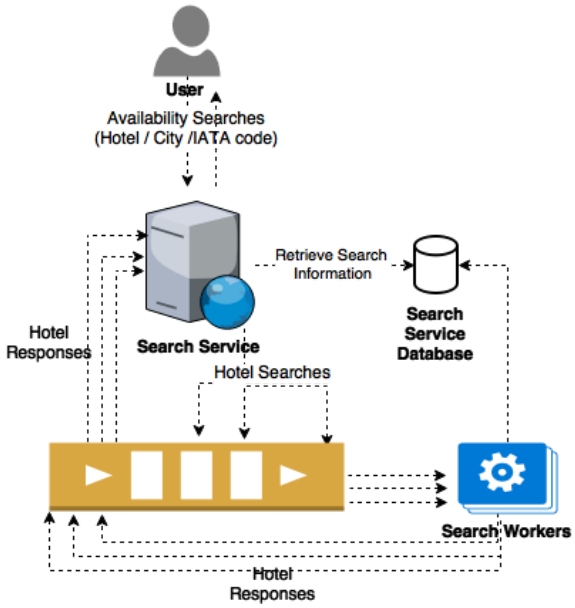
Authentication and Authorization



User will request for a resource using standard authentication credentials (username with password or token) . The Service (Search or Booking) will authenticate the request using Authorize Service. Once the request is authenticated Authorize Service will provide a local token to the Service in the response. Then

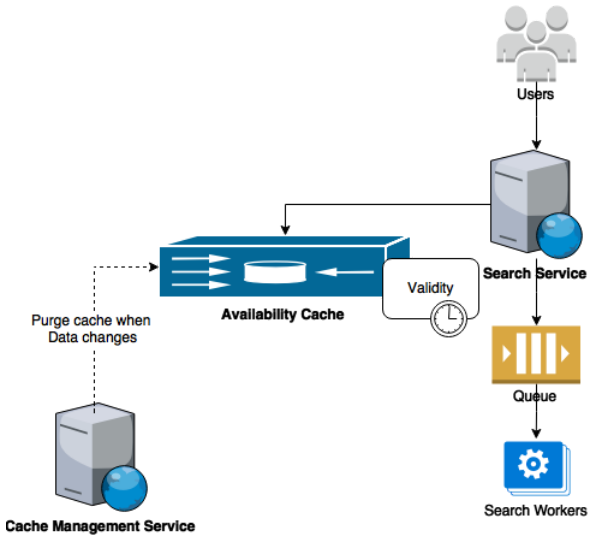
Service will use the local token to communicate with other services. Local token will be validated by the correspondent services using Authorize Service.

Search Service Map-Reduce



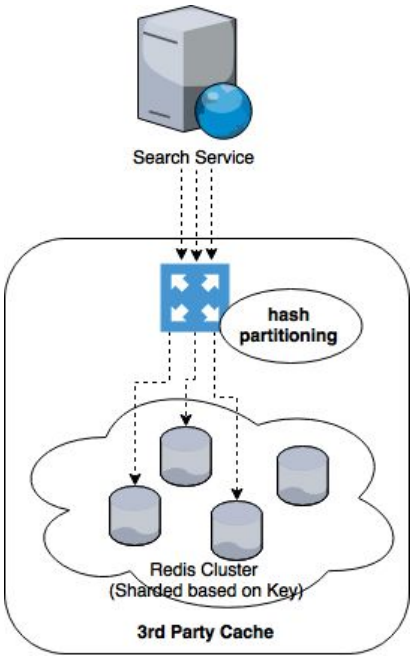
Currently multi hotel searches & city searches are taking long response time. To resolve this issue, Search Service will be break the requests into number Hotel Searches & put them into the queue. Search Workers will process the queue & put back responses to the queue. Search Service consume the responses & send back the aggregated response to the User. We expect this will improve XML availability response time.

Availability Cache



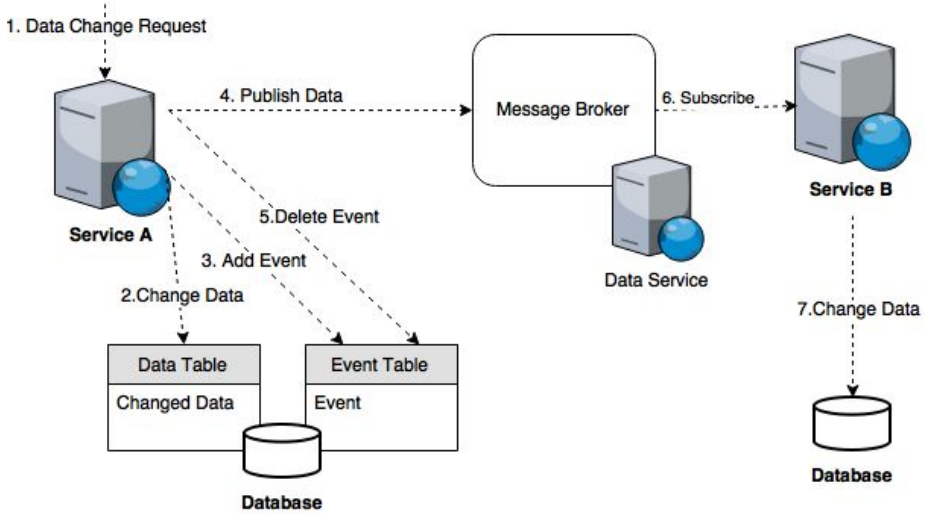
This component is a new layer to Search Service. Availability Cache will store processed availability responses based on request keys. Each cache will have the validity period to expire. Also Cache Management Service will purge related cache when corresponding data got changed. We expect this will improve XML availability response time as well as reduce number of hits to backend services.

3rd Party Cache



Current 3rd party caching solution is having scalability & high availability limitations. This component introduce redis cluster to cache the 3rd party results based on hash partitioning using the key (key is composed by request parameters). This will provide the flexibility of scaling up the 3rd party caching solution for increasing traffic.

Data Service



Data Service provide the data consistencies among the services, where each services use their own databases & structures. Each services will maintain a event table to store event triggers when any data changed by the service. Once the event is successfully published to the Data Service, the corresponding event event will be removed from the Event table. In the other side, All the services, which are subscribed for particular event, consume the event & update as their own data.

Data Trigger Service (Temporary)

This service will be used in migration phases, since this document is focussing on XML revamping, this migration needs a temporary service publish the data change triggers from existing databases to new till whole Bonotel system (XML, BO & WEB). This service is used for that purpose.

2.4. Technology Selection

Application Server

Options : Wildfly 10.1 or Apache Tomcat 8.
We recommend to go with Apache Tomcat, since XML application is not fully utilizing JBoss/Wildfly enterprise stack (eg : EJB), Application server itself giving a overhead in resource utilization. Please find the resource utilization comparison between WildFly 10.1 & Apache 8 in Annexure 1.

Java

Java 8

Web/XML/REST Layer

Spring boot 1.5
Currently we use struts, we decided to go with Spring as it is more lightweight. Since our target architecture is based on micro service architecture, we recommend to go with spring boot.

Caching Layer

Technology is not decided yet. We will decide the technology based on performance, resource utilization & ability to have dynamic caching algorithm based on the traffic pattern.

Data Layer

Hibernate 5
JTA

Unit/Integration Test

Junit 4
Sprint boot test 1.5 (integration test)

Logging

Logstash 5.4.2

Authentication

OAuth 2

3. Implementation & Migration Approach

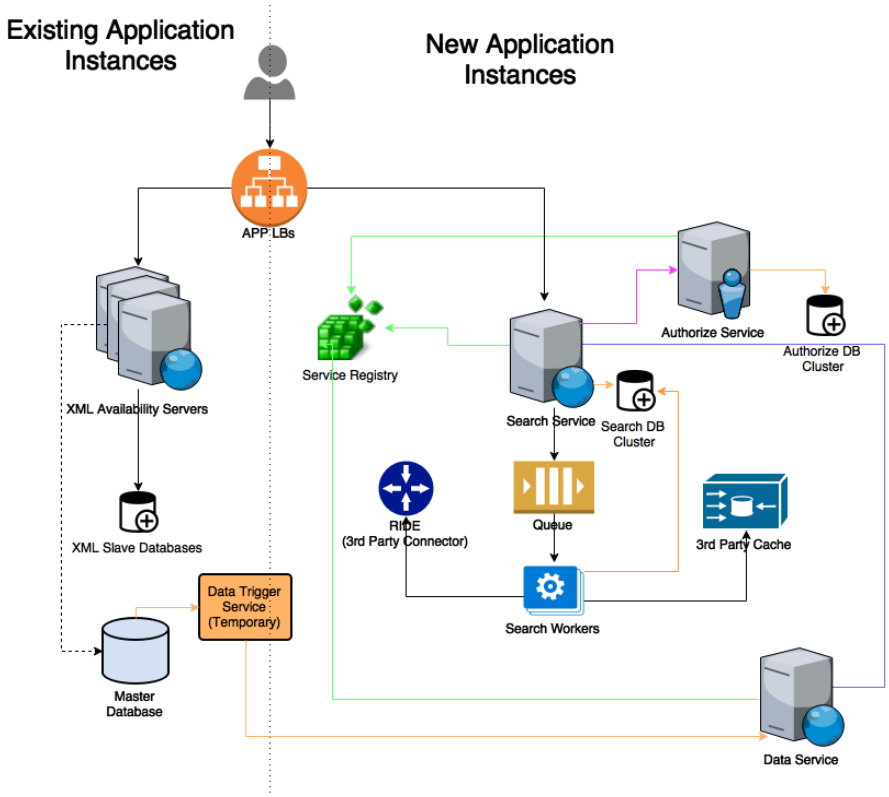
Implementation & Migration can be done in few phases.

3.1. Phase 1 (Search)

Implementation

- Implement Service Registry
- Implement Data Service
- Implement Data Trigger Service
- Implement Authorize Service
- Implement Search Service

Migration



- Integrate Data Service between existing Data Structure and Search Service Data structure using Data Trigger Service & check ACID features with the Data
- Integrate Data Service between existing Data Structure and Authorize Service Data structure using Data Trigger Service & check ACID features with the Data
- Test the Search Service features with the new data structure
- Performance test with the new data structure
- Enable live traffic gradually by adding new instances into LB.

- Monitor
- Disable Traffic from Old instances gradually

3.2. Phase 2 (Cache)

Implementation

- Integrate Cache into Search Service
- Implement Cache Management Service

Migration

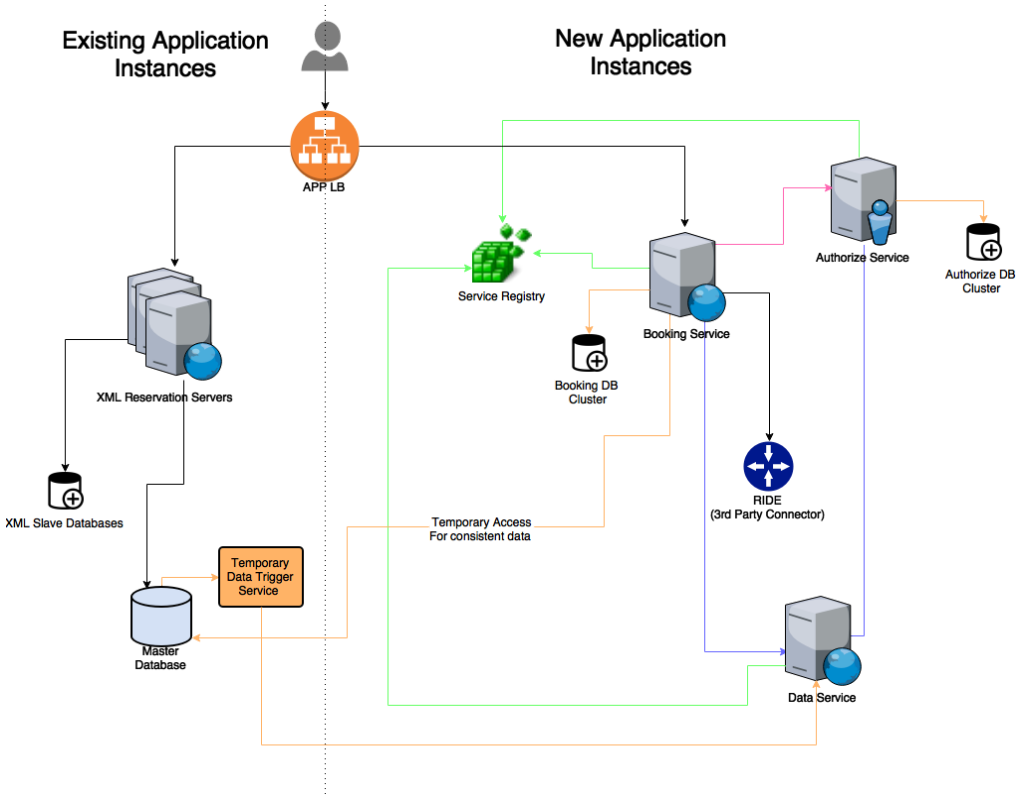
- Integrate Data service with Cache Service Data structure using Data Trigger Service
- Test the Purging feature with the data change
- Tests the Search features with Cache enabled
- Performance test with Cache enabled
- Enable live traffic gradually by adding new instances into LB
- Monitor
- Disable Traffic from Old instances gradually

3.3. Phase 3 (Booking)

Implementation

- Implement Payment Service
- Implement Booking Service

Migration



- Integrate Data Service between existing Data Structure and Payment Service Data structure using Data Trigger Service & check ACID features with the Data
- Test Booking features with the new data structure
- Performance test with the new data structure
- Enable live traffic gradually by adding new instances into LB.
- Monitor
- Disable Traffic from Old instances gradually

4. Challenges/Risks

Risks in Business

- Peak season starts in November, by that time system should be ready state to handle target hits. Application performance optimization and system scale up projects should be completed.
- System modernization/revamping requires considerable effort and human resource bandwidth. This will impact to capacity(Dev,QA) for target feature deliveries.
- Parallel work streams of features and non features may increases the complexity in both integration and deployment phases. This can effect delivery time-lines, system stability.
- Infrastructure(H/W) required for the target architecture may increases cost. (Required proper capacity planning)

Drawbacks

Increase Overhead

- Service interactions are required validation of every input parameters takes place. This can increase the response time and cpu thereby reduce the overall performance.
- Things can get more complicated when remote calls experience latency.
- Size of data massages transferring via each service interactions can increase overall network bandwidth depend on the number of hits/callings requires.

Complex service management

- May requires more infrastructure(H/W) and services fit into new architecture to respond timely manner and support HA with expected traffic level. This may poses a big challenge to manage lot of services.

Investment Cost

- Implementation of new architecture may requires a considerable upfront investment by means of technology, development, and human resource.

Dependencies (Technical)

Java 8

- Dependency library versions (struts, hibernate, jdbc-drivers ..etc)
- Supplier connectivities (TLS/SSL)
- Payment processes connectivities (TLS/SSL)

Tomcat

- EJB utilities

- Transaction handling of the conformation flow(hibernate) and Tomcat deployment

Dependencies (Non Technical)

- Revamping requires code changes of Bonotel application and RIDE application. Code integration will be dependable.
- Requires multiple divisions involvement of human resources (RIDE, Bonotel and Network Engineers)

5. Annexures

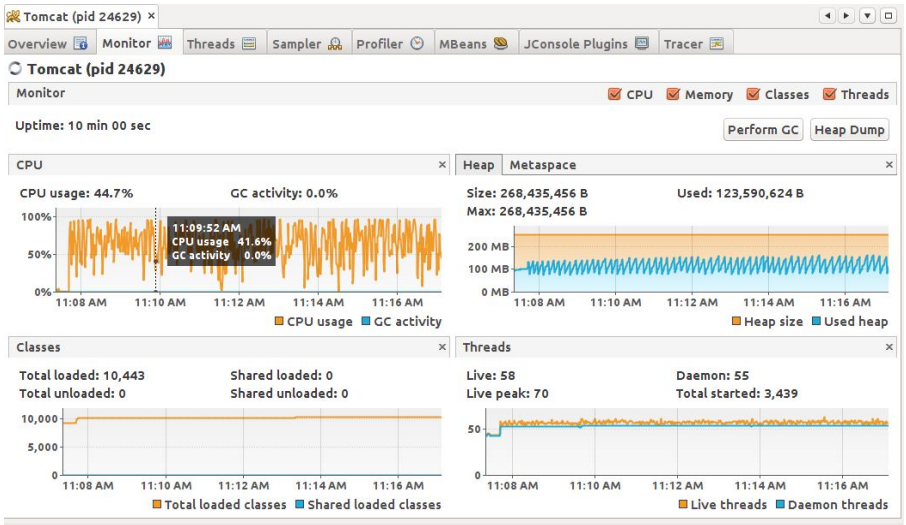
5.1. Annexure 1

We have done performance test using a sample program on both Wildfly 10 & Tomcat 8, which is prototyping following :

- REST interface
- Postgres connectivity using Hibernate
- Postgres connectivity using DBCP
- Multi Threads (Internal - DB query & External - HTTP/REST)

Wildfly 10

Tomcat 8



Summary

Recommend to go with Apache Tomcat for XML redesign based on following reasons:

- Wildfly is loading the enterprise stack, which is almost not used by the XML, which causes more resource utilization.
- Tomcat utilizes the memory & threads in well manner.