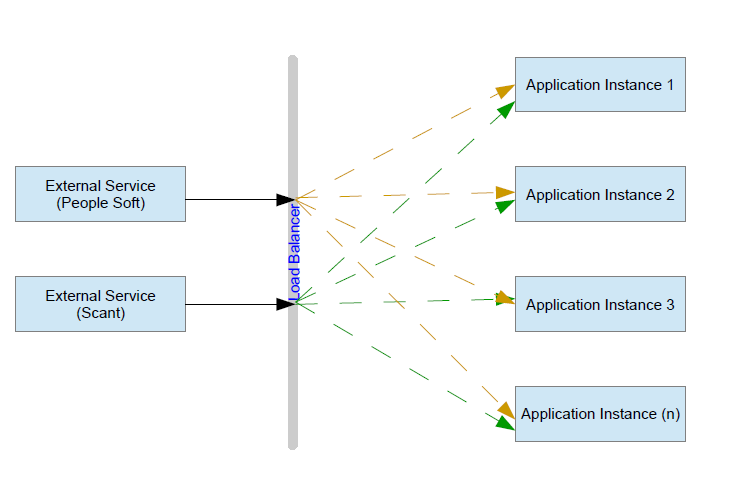
# Instruction

One major issue in the current SMART Operations Board design is that events may not be executed in the order they were originated.  This occurs at all integration layers:

* PS/Scan <--> App Server
* App Server --> App Server
* App Server<--> Browser



For example, when SCAN sends 3 ordered events in a row, APP Server, b/c of threading could process the events out of order, processing 2 before 1 or 3 before 2.  This also happens b/w App Servers with Commands.  At the Browser level, while less likely, it is also possible that the Browser receives Commands out of order...causing the application within the browser to no longer be in sync with that in the App Server (refresh fixes).

## Project Overview

The sequential events processing solution will be built on the following two notions

* Every event CREATOR will assign unique SEQUENCE-NUMBER to every event it creates
* Every event RECEIVER will be processing events only after all smaller SEQUENCE-NUMBERs were fully processed.

## Objective

The objectives of this project are to:

* Come up with the high level design for all sequential events processing issues in the clustered OB environments
* Low level design for PS/Scan calling OpsBoard’s web services.

## Business Cases

Here are a couple of business cases:

* When Scan recalls vehicle, it calls three OB WebServices – setPerson, upVehicle, setSnowDetails. If these calls are processed by OB out of order, the equipment data gets corrupted, board’s recent activities become inconsistent, and manual reconciliation is required.
* When two users are working with the same board and making personnel or equipment assignments to the same task, the assignments data gets corrupted, and board’s recent activities become inconsistent.

## In Scope

* High level technical design
* Low level technical design for Scan calling OB setEquipment web service
* Skeleton implementation for Scan calling OB setEquipment web service

## Assumptions

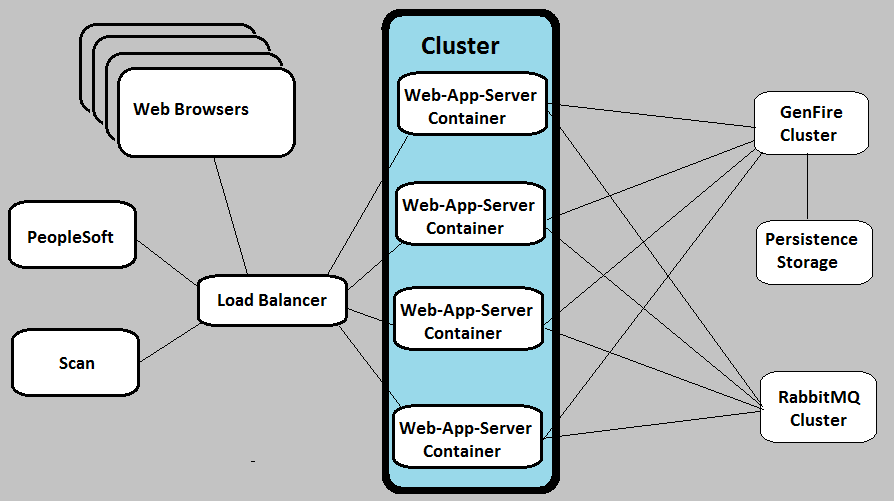
Scan & PeopleSoft will generate and pass along their own sequence numbers for every OpsBoard web service call

# 

# High Level Technical Design Diagram

## High Level Diagram

The following diagram illustrates all new and existing technical resources required for this project



## Event Flows

Here the list of all flows along with event (command) CREATORS and RECRIVERS:

|  |  |  |
| --- | --- | --- |
| Process Description | CREATOR | RECEIVER |
| PeopleSoft calls OB web services | PeropleSoft | OpsBoard |
| Scan calls OB web services | Scan | OpsBoard |
| OpsBoard is calling Scan’s web services | OpsBoard | Scan |
| OpsBoard is calling PeopleSoft’s web services | OpsBoard | PeopleSoft |
| OpsBoard server is sending messages to another OpsBoard server via Rabbit MQ | OpsBoard | OpsBoard |
| OpsBoard is sending command to browser via WebSocket | OpsBoard | Browser |
| Browser sends commands to OpsBoard server via WebSocket | Browser | OpsBoard |

## Technical Specifications

Application Servers

Several Tomcat servers will be clustered and accessed by external systems and web browsers via load balancer. The fundamental flow changes will be related to:

1. All commands coming from browser will be assigned unique sequence number, saved in GemFire, and sent to RabbitMQ.
2. All events coming from Scan and PeopleSoft will be coming with their own sequence numbers, saved in GemFire, and sent to RabbitMQ.
3. RabbitMQ will be publishing these messages to queues for all application servers, and whichever picks up each message first will process it. The rest of the servers which pick up the same message later, will simply ignore it.
4. When UI gets commands back from the server, it’ll sort them in the recent activities panel using sequence numbers.



1. Commands and events with unrelated Business IDs (different equipment and different personnel) will be processed without considering sequence numbers. Only commands and events related to the same Business ID will be processed strictly sequentially.

GemFire Servers

At least two servers will be clustered, and, in addition to their normal OB utilization, they’ll be responsible for:

* Generating sequence numbers for all OpsBoard commands
* Keeping events processing status data
* Providing optimistic locking support for sequential event processing

RabitMQ servers

At least two servers will be clustered, and responsible for:

* Listening to its server-to-server exchanges
* Creating dynamic queues for every clustered application server
* Sending event-related messages to these queue

Web Servers

Will be functioning as reverse proxies to application servers, as well as static content providers

Database Servers

Relational database servers will be providing the main persistence data storage for all OpsBoard persistence needs

# Low Level Technical Design

## SCAN is calling OpsBoardWeb Service

**Class Diagram**



**Receiving Scan Request - Sequence Diagram**

****

**Processing Scan Request - Sequence Diagram**

****

# Test Plan

## Prerequisites – codding for testability

Since it’s a one-time testing effort, we’ll heavily utilize log4j debugging messages in our code, so we can use the Log Stash to verify sequential processing

## Commands Testing

We’ll use DEVX clustered environment, and create Selenium suite which will emulate 10 users performing the following operations on the same task:

* Assign Equipment for ABC-xyz
* Assign Person John Doe-xyz

**Assertions**

* After each step Selenium will assert the expected assignment on the screen
* Additionally, we’ll also verify LogStash records and make sure that all commands were processed sequentially

## Scan & PeopeSoft Testing

We’ll use DEVX clustered environment, and create JUnit which will emulate the following sequence of calls for 20 vehicles:

* Call downEquipment web service for ABC-xyz with sequence number 1
* Call setEquipment **condemned** web service for ABC-xyz with sequence number 2
* Call setEquipment **recalled** web service for ABC-xyz with sequence number 3
* Call upEquipment web service for ABC-xyz with sequence number 4
* Call setSnowDetails web service for ABC-xyz with sequence number 5

**Assertions**

* JUnit will assert equipment and recent activities data in the database.
* Additionally, we’ll also verify LogStash records and make sure that all commands were processed sequentially