1 . Identify measurement

In order to test the functionality of pushing notification, we need to define ‘Good notification Performance’.

1). **Performance**: More than 99% notifications can be pushed to users' device successfully if device is online.(*Assume 99% means good performance)

2). **Low Latency**: If a user's device is online, messages should be bush within given time. This can be checked with timestamp, for example less than 30 seconds for online devices.

3). **Status Exchange**: Messages that should be pushed are properly detected, i.e. messages have status: pre-push/pushed/pending/failed.

Pushed messages are switched from the "pre-push" state to the "pushed" state in the DataBase once confirmed message was received on device.

Further checking may also change it into failed or pending by users information from database, for example: *last_online_time* from user’s device table query by user_id*,* if a device is more than 15 minutes since *last_online_time* for all devices, this message will also labeled as ‘Pending’.

4) **Correctness**: Messages pushed are correct content and are properly formatted. In order to achieve this target, each Endpoint need to open message ***Debug- mode***.

A ***Debug-Mode*** means for each message received, Debug-mode also check the content and format, finally returns message to server with *message_id*, *received*, send_time, receive*_time, content, format_correct, user_id, device_id, etc.* When Testing Server received this message, it can label the messages status and generate test reports.

For example, returned object form device:

```
{
    "message_id":FJK274969,
    "content": {...},
    "send_time":20160619123455.23,
    "Receive_time":20160619123456.04,
    "format": true, //means format is correct, otherwise false
    "User_id": HJ09876234,
    "device_id":UUID-c8fff94ba-9964110-698b53-f7b037,
```

```
    ...
}
```

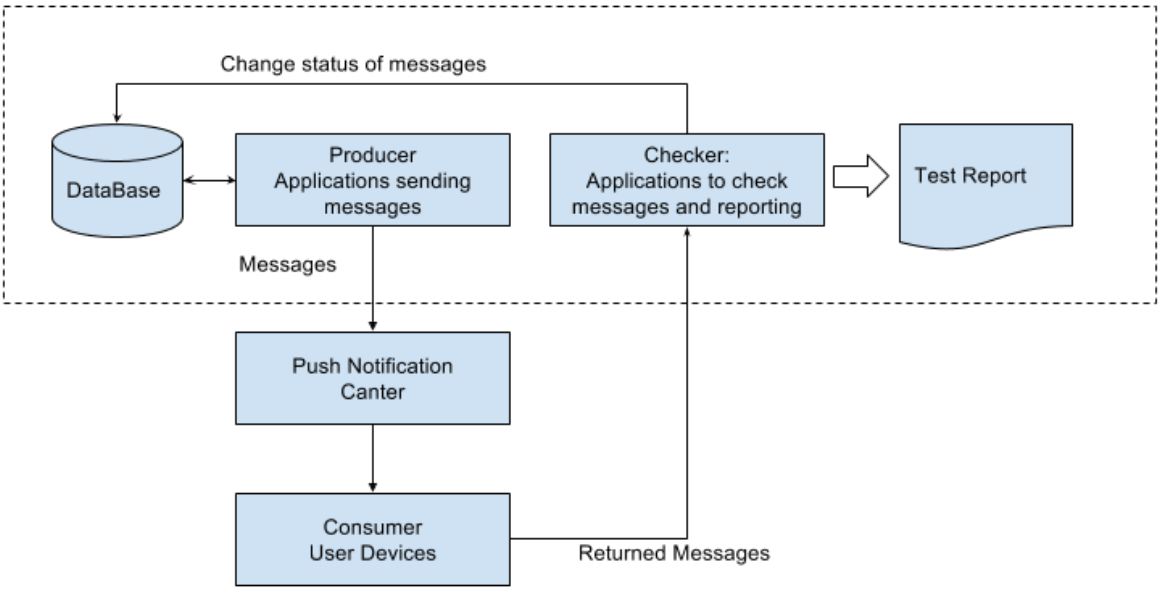
2. System Component

In order to deploy these tests, here are the components for this system.

Producer: Applications that generate and send messages out, continuously read database,get message information, user and device information, then send messages to user's devices.

Checker: Applications are running on test servers to deal with returned messages, and generate reports.

Consumer: Endpoints running slack app, can be IOS device/Android etc. In order to check the status and content of messages, Debug_mode is turned on, to check messages and generate returned object.



Workflow for Testing Messages

Database: Storage layer for saving users' information and message information. Since this tests deal with user table, message table, this DataBase can be PostgreSQL, for it is rational database with stable and good performance.

Here are the attribute of all tables, the types of each attribute can be decided as the value.

User Table: *user_id(Primary key), user_name, time_zone* …(other information), ...

Device Table: *device_id(Primary key, or UUID), device_name, user_id, last_online_time, ...*

Message Table: *messgae_id(Primary key), message_name, messge_content, target_user, send_time, sent_time, send_status, retry_times, ...*

3. Test Automation

This test can be automated with Jenkins, Ansible, Artifactory, with Github/Bitbucket and JUnit reader.

Jenkins schedule all tasks to run. Before run each task, checkout code from github/bitbucket, build binary package(application) in Artifactory, and install into servers, if application need to be update.

Once the testing environment is installed, all tests are good to go, including performance-test, api-test, correctness-test and load-test. When devices returned messages to Checker, Checker checks all messages, and changes status in Database, then generate JUnit format report.