A marketing and communication engine

1. Should we create this as a new application or make it a part of the existing application? Please indicate your thought process?

It will be better to create this as a new application, as the new engine is a separate module and the existing application is independent of the new engine.

- It better be a black box to the existing application.
- Developer should be open to use new language and frameworks.
- The existing application does not need to know the technology behind the engine.
- It can be access through APIs easily.
- It will be easily modifiable, as the underlying rules can constantly change.
- Codes of both the application will be cleaner.
- The new application APIs will be accessed internally.

It will be even better to create two new applications one for marketing engine and one for communication engine, as communication engine itself is a separate module and will required lot of additional features in the future. So the **responsibility** of the applications should be divided.

- The existing application can use an communication application directly.
- The communication application will involves queues for processing a communication.
- The actual communication logic will be handled by a consumer listening to the queues.
- The communication application will have templates to be designed.
- The marketing application will have rules to be designed.
- The communication application should be flexible to use different technologies too depending upon the developer's expertise.
- The communication application will have to be auto scalable, increasing the number of queuing server or increasing the number of consumers.
- The marketing application will need to have a caching layer on the top.
- The retry logic or reminder logic of a communication may be required in the future, so they should be easily integrable.

2. If this, is a separate application, illustrate the communication between the 2 applications.

Let's have 3 applications. Existing, marketing and communication. It would have been easier if virtual credits or passes were handled by a different application too, but that would be out of scope of this design.

- The existing application will directly call the communication application API for real time communication like signup or when virtual credits or passes are assigned to a customer manually.
- The existing application will call the marketing application API for applicable options as per the rule both claimed, unclaimed and not needed to be claimed.
- The existing application needs to share a new read-only slave database with the marketing application. Rest of the logic will be handled by marketing application workers.
- The existing application needs to expose an HTTP API for claiming virtual credit and pass sent in any communication medium.

Design Document

1. Purpose of the SDD

The SDD documents and tracks the necessary information required to effectively define architecture and system design in order to give the development team guidance on the architecture of the system to be developed.

The document will contain the basic flow between components.

2. General Overview

We are following Service-oriented architecture.

2.1. Assumptions

The old application will provide the necessary endpoints to the end user, and it can use the marketing application and the communication application directly.

The new applications APIs will be accessed internally and need not be hosted on public IPs.

Marketing

2.1.1. Storage requirement

- Assuming max limit 100m of total users. Each user will have average 2 passes at a time, 2 expired passes and 1 virtual credit rows.
- We can clean the expired ones after a month.
- 500m rows in relational (will not be used as we are always querying using userId) or column family based nosql or in cache.
- 100m documents with list of 5 entries each in document based nosql.
- But the total size of the databases will remain same.
- Row size = per row requires 2B (userId) + 2B (value) + 1B (offerId) + 1B (templateId) + 2B * 3 (for allotted date, claimed date, expiry date) + 140 * 2B (140 characters description) + 8B (buffer) = 300B
- Total size of the database = 100m * 5 * 300B = 150000 mB = 150GB
- Size of sets of rules that can be applied will be negligible

2.1.2. Cache requirement

- 0

2.1.3. Bandwidth requirement

- 10k concurrent requests * 300B = 3MB

Communication

2.1.4. Storage requirement

- We have 1000 templates of each types of communication: email, sms and notification.
- 1000 * (1MB including attachments + 140B * 3 parts + 1kB) = 1GB
- Size of one communication record = 2B templateId, 1KB parameters = 12B
- We store logs of all records of communications sent for an year.
- Per day = 10m users per day = 12B * 10m = 120MB
- Per year = 12B * 10m * 365 = 44GB
- Queue size = 120MB / 24 = 5MB
- Total = 44GB + 1GB = 45GB

2.1.5. Cache requirement

- 0

2.1.6. Bandwidth requirement

- 10k max concurrent communication requests from main application or marketing application * 12B = 120KB

2.2. Constraints

The load on both the applications can be immense depending on the number of users and their activities on the main application.

- Fail-safe mechanism
- Scaling mechanism
- Handling real time requirement

3. Design Consideration

Both the applications, both the databases, the background workers, the queue servers, the queue consumers will all have minimum two instances. Both the applications will have a Load Balancer in front of them.

Marketing

- We are querying on the basis of userId. So a simple persistent cache will be sufficient.
- But in future, we may need to query on the basis of fields. For this, using a document based NoSQL like **Couchbase** will work wonders. We can pass NQL or MapReduce queries if required.
- Scaling will be easier.
- Background processes will handle the virtual credit and passes allotment part.

Communication

- The requests we receive can directly put into a queue and queue can get heavy as consumers will take sufficient time in sending.
- Consumers taking care of sending part should be autoscable.
- Separate realtime Queue as, realtime communications size will not be huge.
- Consumers will first check realtime queues then other queues.
- Communication logs requires write only and the writes should be fast.
- Write fast database like **Cassandra** will best.
- We need clearing older than a year old records. A simple CQL query will do the job.
- Logs may be needed for analytics but that is a background query. Hadoop Mapreduce can be used.

4. Database Design

Marketing

Pass	**schema**	**description**
id	int	Pass id with which this pass user has received
amount	int	Value the user can use
…..	**…..**	Other detail

VirtualCredit	**schema**	**description**
id	int	Rule id with which this pass user has received
amount	int	Value the user can use
oneTime	boolean	Whether a user can use this offer more than one time
…..	…..	Other detail

Rule	**schema**	**description**
id	int	Rule id with which this pass user has received
type	Enum	Pass or VirtualCredit
offerId	int	Id of the Pass or the VirtualCredit
templateId	int	Id of the template of communication
…...	…...	More detail required for rule engine

UserPass	**Intermediate schema**	**description**
expiresAt	date	Expiry date
createdAt	date	Alloted date for unclaimed and claimed date for claimed
amount	int	Value the the pass. Not using pass id as amount can be modified, so can be offerId.
ruleId	int	Rule id with which this pass user has received

UserVirtualCredit	**Intermediate schema**	**description**
expiresAt	date	Expiry date
createdAt	date	Alloted date for unclaimed and claimed date for claimed
amount	int	Value the the pass. Not using VC id as amount can be modified, so can be offerId.
ruleId	int	Rule id with which this amount user has received

UserOffers	**schema**	**description**
userId	long key	userId will be the key of the document
claimedPass	List of UserPass	List of passes claimed
unclaimedPass	List of UserPass	List of passes not claimed yet
claimedVirtualCredit	List of UserVirtualCredit	List of virtual credit claimed
unclaimedVirtualCredit	List of UserVirtualCredit	List of virtual credit not claimed yet

Communication

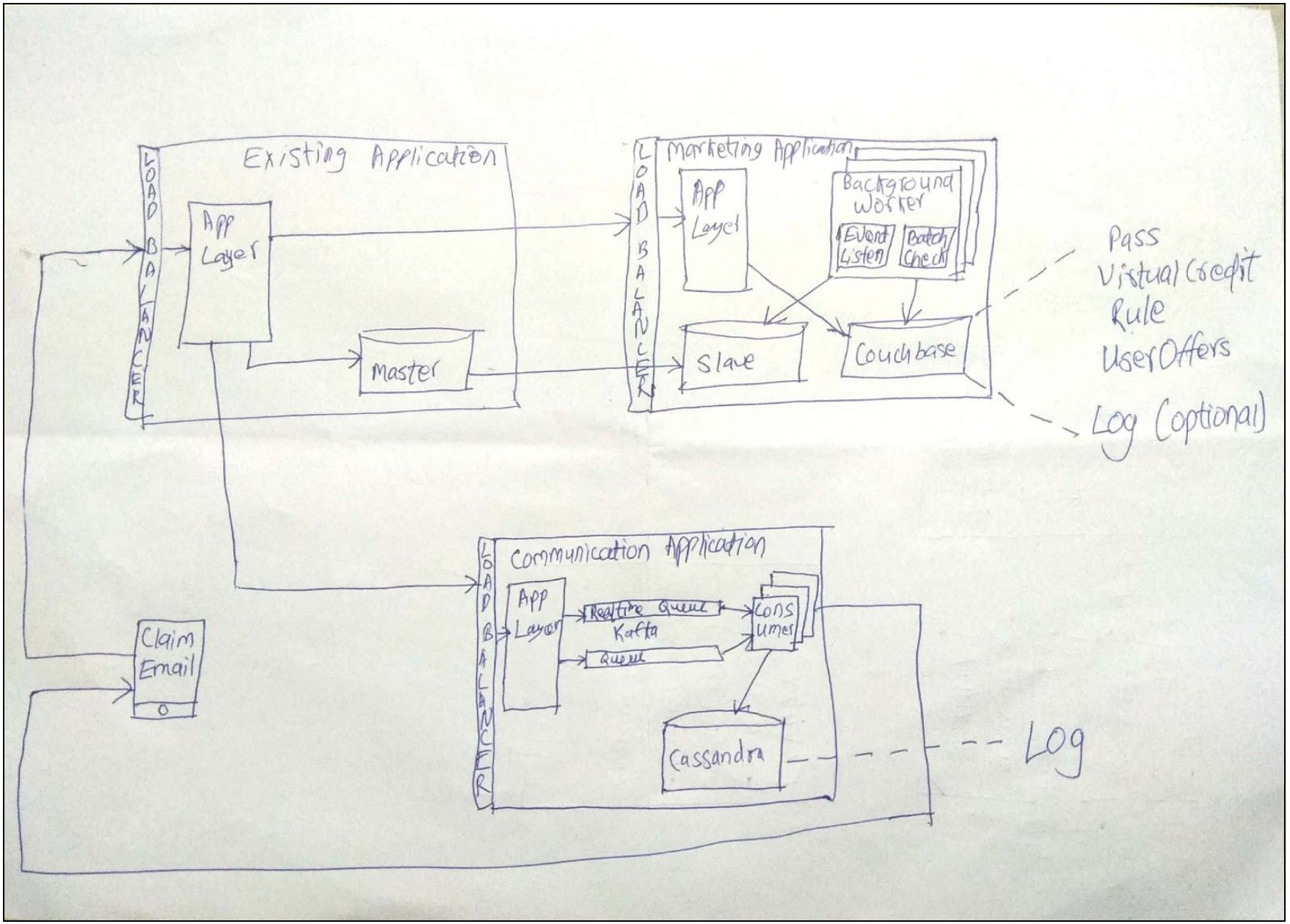
Template	**schema**	**description**
id	Int: key	Id of the template to be sent
mode	Enum	Email, SMS, Push Notification
content	text	content
description	text	Details for why this template will be used

MQCommunication	**Queue object**	**description**
templateId	int	Id of the template
parameters	List of strings	Parameters to be replaced in the template
receivers	text: optional	comma separated batch of emails or gcm ids

CommunicationLog	**schema**	**description**
userId	int	Id of the user
content	text	Content sent to the user
mode	Enum	Email, SMS, Push Notification
sentAt	date	Time the communication was sent
receivers	text: optional	comma separated batch of emails or gcm ids

5. System Architecture

Below is the system architecture and the interactions between components of the 3 applications.


Existing Application
LOAD BALANCER
App Layer
master
Marketing Application
LOAD BALANCER
App Layer
Background Worker
Event Listen
Batch Check
Slave
Couchbase
Pass
Virtual Credit
Rule
User Offers
Log (Optional)
Communication Application
LOAD BALANCER
App Layer
Realtime Queue
Kafka
Queue
Consumer
Cassandra
Log
Claim Email