

# System Spec Document for Clubshop System

31, August 2017

---

**Masaya Arai**

IT Innovation India Pvt. Ltd.

Office No. 302, Pentagon 3, Magarpatta City, Hadapsar,  
Pune, Maharashtra 411028

<b>INTRODUCTION</b>	<b>3</b>
PURPOSE	3
DOCUMENT RELATIONSHIP	3
SYSTEM SCOPE AND SYSTEM FEATURES	4
<b>DEVELOPMENT POLICY</b>	<b>4</b>
BASIC POLICY	4
SYSTEM ASSUMPTION	4
<b>SYSTEM REQUIREMENT</b>	<b>5</b>
PERFORMANCE REQUIREMENT	5
SCALABILITY REQUIREMENT	5
DURABILITY REQUIREMENT	5
DISASTER RECOVERY REQUIREMENT	5
SECURITY REQUIREMENT	5
OPERATION REQUIREMENT	6
OTHER REQUIREMENT	6
<b>OVERVIEW</b>	<b>7</b>
SYSTEM OVERVIEW DIAGRAM	7
System Environment	8
Region	8
Availability Zone	8
Environmental Separation	8
Network Security	8
PLATFORM CONSTRUCTION	9
SOFTWARE CONSTRUCTION	10
NETWORK CONSTRUCTION	10
<b>TRANSACTION PROCESSING METHOD</b>	<b>11</b>
STRATEGY AND POLICY	11
DESIGN	11
OL-A1	12
OL-B1	13
BT-A1	14
<b>PERFORMANCE IMPROVEMENT METHOD</b>	<b>15</b>
STRATEGY AND POLICY	15
LOAD BALANCING DESIGN	15
Making a Plan for Load Balancing	15
FLOW CONTROL DESIGN	16
CACHING DESIGN	17

Static Contents	17
Business Data	18
<b>SCALABILITY METHOD</b>	<b>19</b>
STRATEGY AND POLICY	19
SERVER SCALABILITY DESIGN	19
STORAGE SCALABILITY DESIGN	19
NETWORK SCALABILITY DESIGN	19
<b>DURABILITY METHOD</b>	<b>20</b>
STRATEGY AND POLICY	20
SERVER DURABILITY DESIGN	20
NETWORK DURABILITY DESIGN	20
<b>DISASTER RECOVERY METHOD</b>	<b>21</b>
STRATEGY AND POLICY	21
DISASTER RECOVERY DESIGN	21
<b>SECURITY METHOD</b>	<b>22</b>
STRATEGY AND POLICY	22
AUTHENTICATION DESIGN	22
DATA ENCRYPTION DESIGN	22
NETWORK ENCRYPTION DESIGN	22
AUDIT LOG DESIGN	22
<b>OPERATION METHOD</b>	<b>23</b>
STRATEGY AND POLICY	23
BATCH	23
TIME SYNCHRONIZATION DESIGN	23
SERVER REBOOTING SCHEDULE DESIGN	23
LOG MAINTENANCE DESIGN	23
DATA BACKUP DESIGN	23
SYSTEM BACKUP DESIGN	23
MONITORING	23
SERVER MONITORING DESIGN	23
PROCESS MONITORING DESIGN	23
LOG MESSAGE MONITORING DESIGN	23
DATA THRESHOLD MONITORING DESIGN	23
<b>DEVELOPMENT &amp; TEST METHOD</b>	<b>24</b>
STRATEGY AND POLICY	24
DEVELOPMENT ENVIRONMENT DESIGN	24

LIBRARY MANAGEMENT DESIGN	24
<b>BILLING METHOD</b>	<b>25</b>

## INTRODUCTION

### PURPOSE

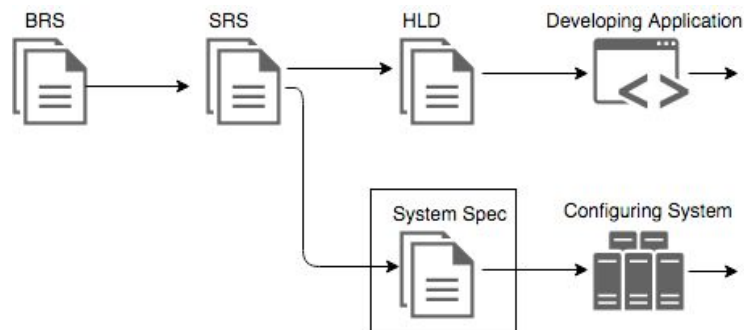
The “Clubshop system” is aggregation service of many useful coupons for all end-users in India by gathering a lot of online shops and clubs information.

This document provides core strategies and methodologies using infrastructure techniques on AWS for optimally designing “Clubshop system” in order to post coupon data quickly, securely and certainly.

### DOCUMENT RELATIONSHIP

This document is written by basis on SRS and input to configuring system.

We show the role of this document below diagram.

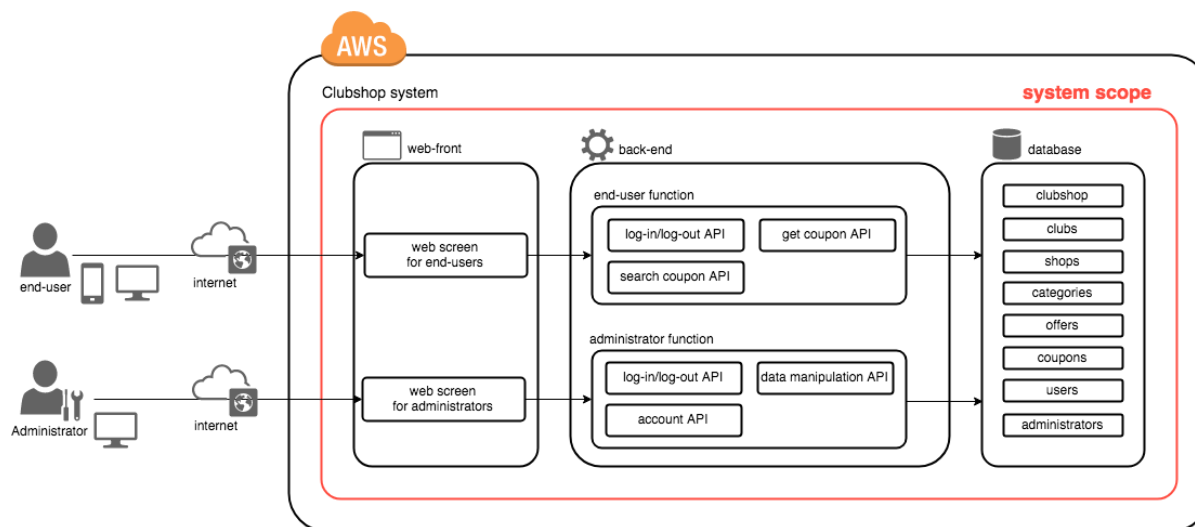


## SYSTEM SCOPE AND SYSTEM FEATURES

Following diagram shows scope of system.

This document DOES NOT provide implementation details or infrastructure configurations, only including the way of designing for future strategies.

We assume that this system is mainly available in India.



## DESIGN POLICY

### BASIC POLICY

This document describe about ideal design of infrastructure for clubshop system. (When this system is implemented, you should have consideration for budget.)

### SYSTEM ASSUMPTION

Each service should be used in same AWS region from the viewpoint of low latency.

We can select different AWS regions only If the region which we currently use are not provide some service which we want to use.

## SYSTEM REQUIREMENT

### PERFORMANCE REQUIREMENT

- The system must return response to end-users within at least **1 second, 10,000 concurrent users** about online access.
- The system **does not define data capacity requirement** because of unlimited resource on AWS.

### SCALABILITY REQUIREMENT

- All function and server should be **scalable** when demand changes in the future.

### DURABILITY REQUIREMENT

- Each function and server should be developed with **redundancy** as possible.
- All of incident which happens in this system should be restored to usual status within **3 hours**.

### DISASTER RECOVERY REQUIREMENT

- **Backup environment** should be developed preventing from disaster recovery such as humongous earthquake, large tidal wave or pandemic power failure.
- The system should be recover from disaster recovery within **72 hours**.
- This system should synchronize business data to backup environment, aim for within **24 hours as RPO (Recovery Point Objective)**.
- We should **manually** switch from production environment to backup environment in according with some procedure when disaster occurred.

### SECURITY REQUIREMENT

- The system must **encrypt all of data on the network** between end point and the system via internet.
- The system must **encrypt confidential data** in database. (like end-user's password)
- The system shall allocate each end-users and administrators with **appropriate role** to call API properly.
- The system shall get **audit log** so that can trace.
- All password including special character (like '@', '\$', '#' or '\_') should be **changed on a regular basis**.

## OPERATION REQUIREMENT

- The system must be available between **24 hours, 365 day** as long as possible.
- This system should acquire **system backup** when changing system configurations.
- Business data (stored in database) should be acquired as backup at least current **2 generations**.
- Operational logs should be automatically acquired and backed up using storage in a **daily basis**.

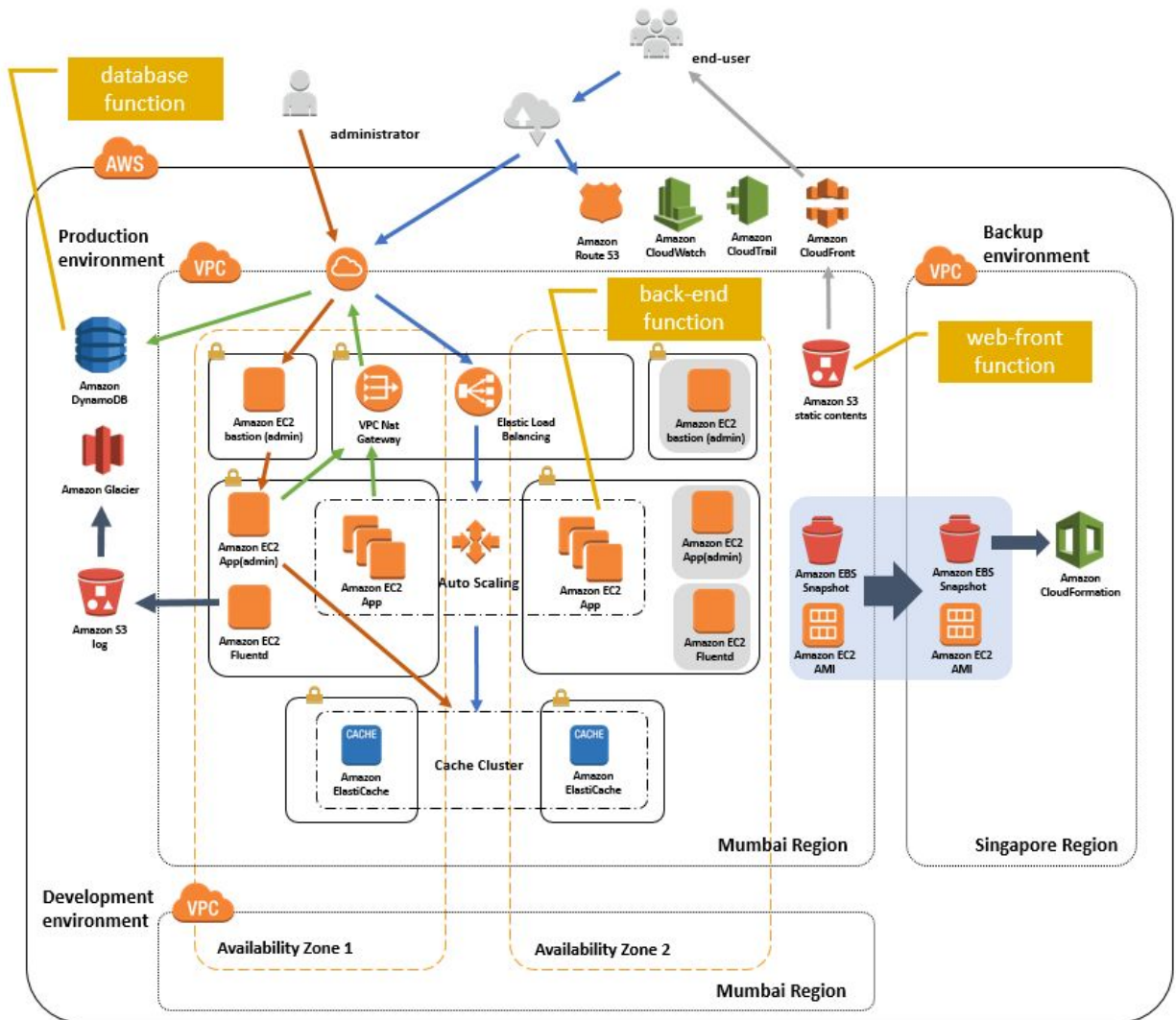
## OTHER REQUIREMENT

- The system should prepare **environment for application development** on AWS.

## OVERVIEW

### SYSTEM OVERVIEW DIAGRAM

System overview is illustrated following the figure.







To design architecture, this document indicate some specific prerequisites.

## System Environment

We decide to make full use of AWS as system platform. AWS has many service to launch some service easily, and we can reduce a lot of cost and labor for the management.

## Region

Production environment of this system should located in Mumbai region following reason.

- Ensuring data ownership & compliance in India
- Minimizing data latency
- Using necessary AWS service and function for business goals

## Availability Zone

This system should be generally utilize two availability zone in Mumbai region to achieve high availability in terms of business data and infrastructure components .

## Environmental Separation

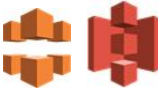




In this system, VPC (Virtual Private Cloud) should be divided into 3 parts for production, disaster recovery and development because of manageability and enhanced security.





## Networking

We split network into three segment in order to ensure the security threat. (DMZ segment, Trusted Segment and Database Segment)

Private IP address should be assigned to each segments as many as possible for scalability.

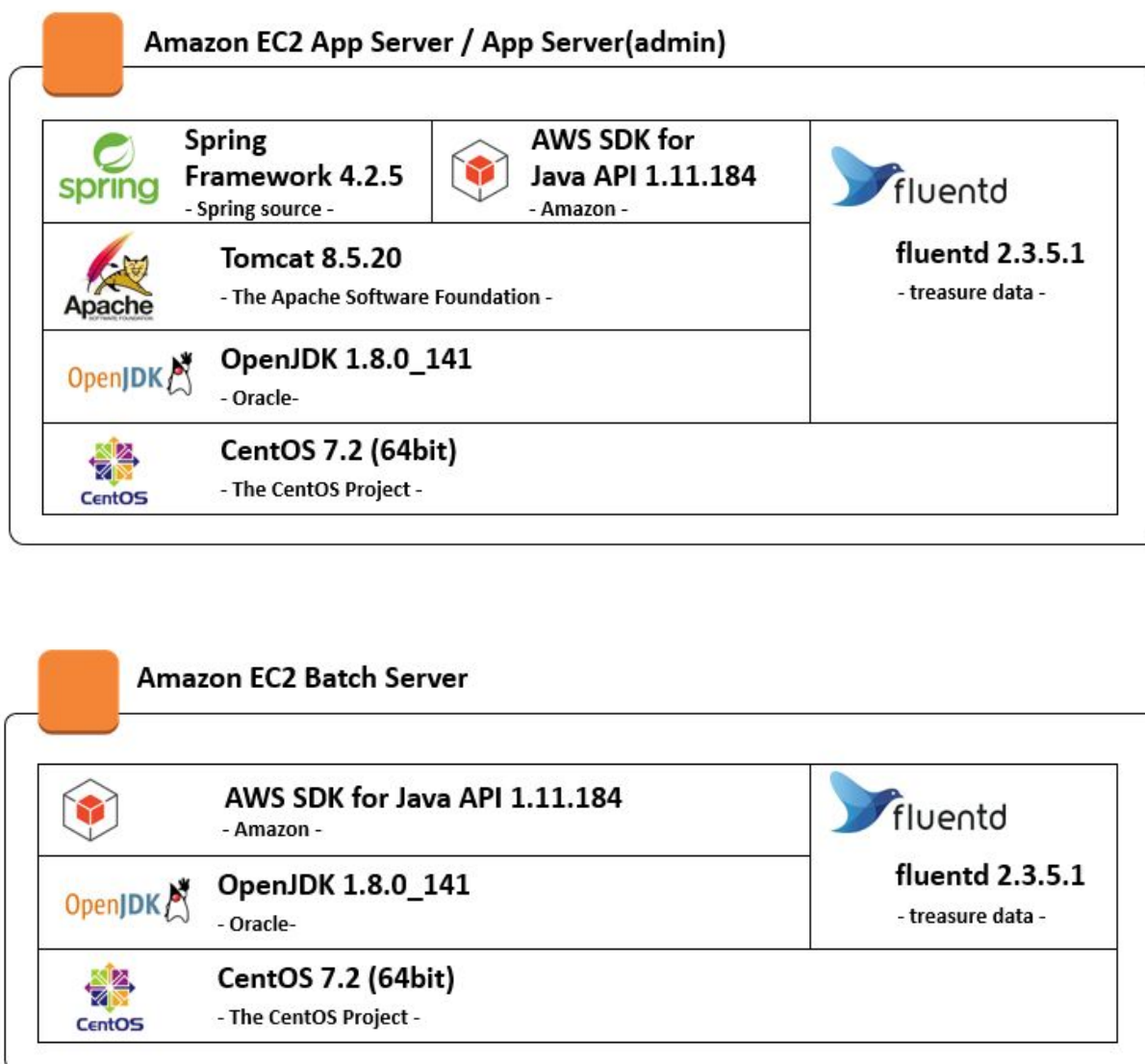
## PLATFORM CONSTRUCTION

Function	Description	Services	How to use
web-front	1. Offering user interface screens for end-users. 2. Offering user interface screens for administrators.  # including html files, some images and javascript codes.		The system provides web-front function from Amazon S3 by uploading components. This design is one of best practice which is recommended by Amazon. To optically convey contents for end-users, we'll use CloudFront.
back-end	1. Providing end-user's APIs like a "getCoupons". 2. Providing administrator's APIs like a "createClub".		API's are core function for the system and there are called rather frequently, so the system should be scalable using Elastic Load Balancer, many EC2 instance with Auto Scaling technology.
database	1. The database function stores all business data like clubs, shops user registration or coupon information.		We use DynamoDB as NoSQL service with consideration for ease to migration from current database (MongoDB) and fast response. Especially, in regard to like session data, we use ElastiCache as responses are returned more quickly.
data backup	1. Backup business data for recovery in the event of loss. 2. Requirement for auditing.		Data back design should be attained using S3 and Glacier in terms of cost merit.
log collecting	1. For Investigation for when some problem occurs.		There are some type of logs in this system, and these status change drastically during auto scaling. To simplify about collecting log, this system should be constructed with automatic log aggregator using fluentd.(open source data collector)

Function	Description	Services	How to use
monitoring	1. Checking service status. 2. Managing auto scaling. 3. Collecting log for AWS service.		This system should use CloudWatch in order to monitoring AWS services uniformly. This system should apply for CloudWatch to collect transaction log such as S3 transaction.
disaster recovery	1. Preventing calamities from halting system.	  	Disaster recovery strategy is option for this system, but if implements, we need to transfer some components(AMI, storage snapshot, business data)

## SOFTWARE CONSTRUCTION

Following Diagram shows lists of third party products which are needed for Clubshop services.



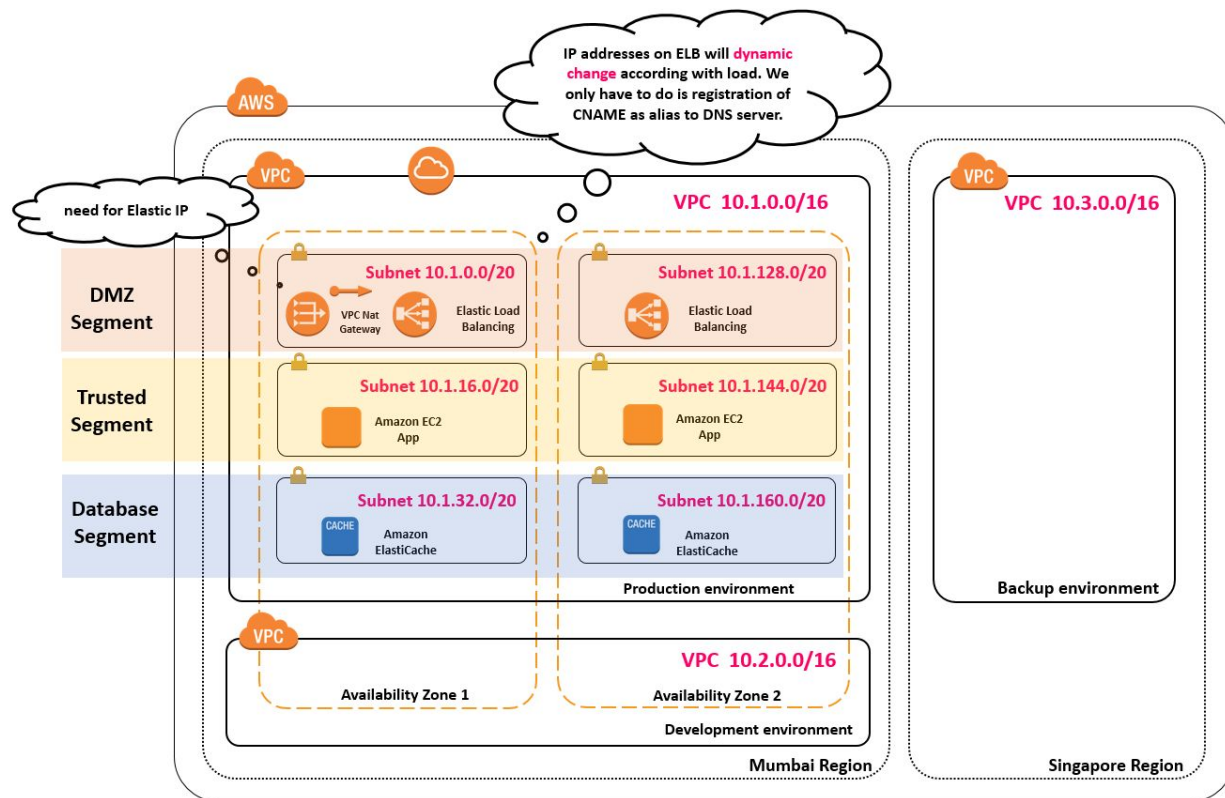
## NETWORK CONSTRUCTION

This section shows description for network design including allocation of IP address, list of network function.

- Private IP address should be assigned to each segments as many as possible for scalability.
- Elastic IP address should be allocated to NAT Gateway from EC2 instance to Internet or region services(like a DynamoDB).
- Customer Gateway service is not used in this system because of no requirement for connecting on-premises system or other bases.
- Allocation policy of IP address for production environment should be applied to Development environment.

The below table and figure shows network design about allocating network address on each subnet.

VPC	AZ	Segment	Allocation IP Address	memo
Production	-	-	10.1.0.0/16	
	AZ1	DMZ	10.1.0.0/20	
		Trusted	10.1.16.0/20	
		Database	10.1.32.0/20	
		(Reserve)	10.1.48.0 - 10.1.127.255	
	AZ2	DMZ	10.1.128.0/20	
		Trusted	10.1.144.0/20	
		Database	10.1.160.0/20	
		(Reserve)	10.1.176.0 - 10.1.255.255	
Development	-	-	10.2.0.0/16	
Backup	-	-	10.3.0.0/16	



## TRANSACTION PROCESSING METHOD

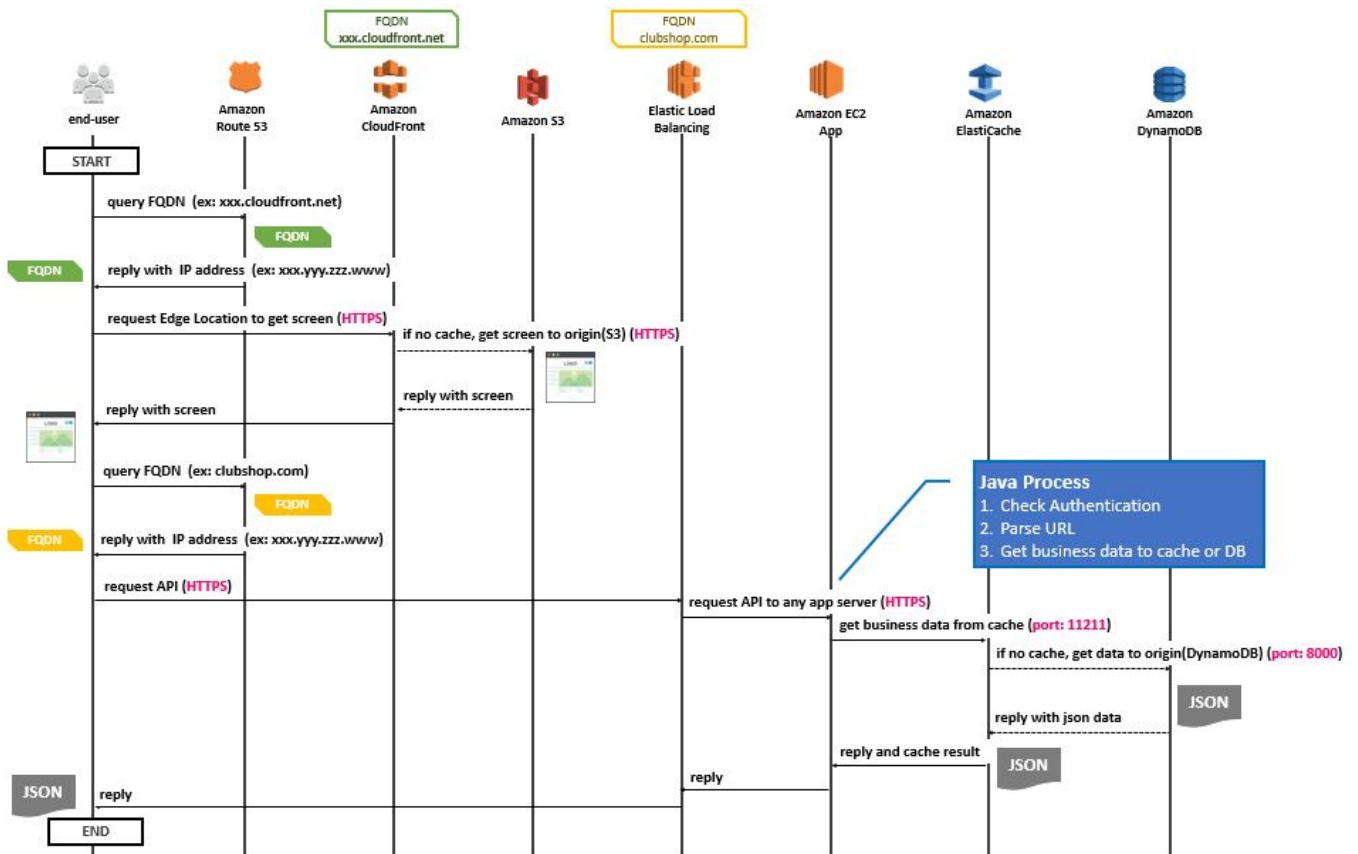
### STRATEGY AND POLICY

### DESIGN

The below tables and flowcharts detail how to flow transaction.

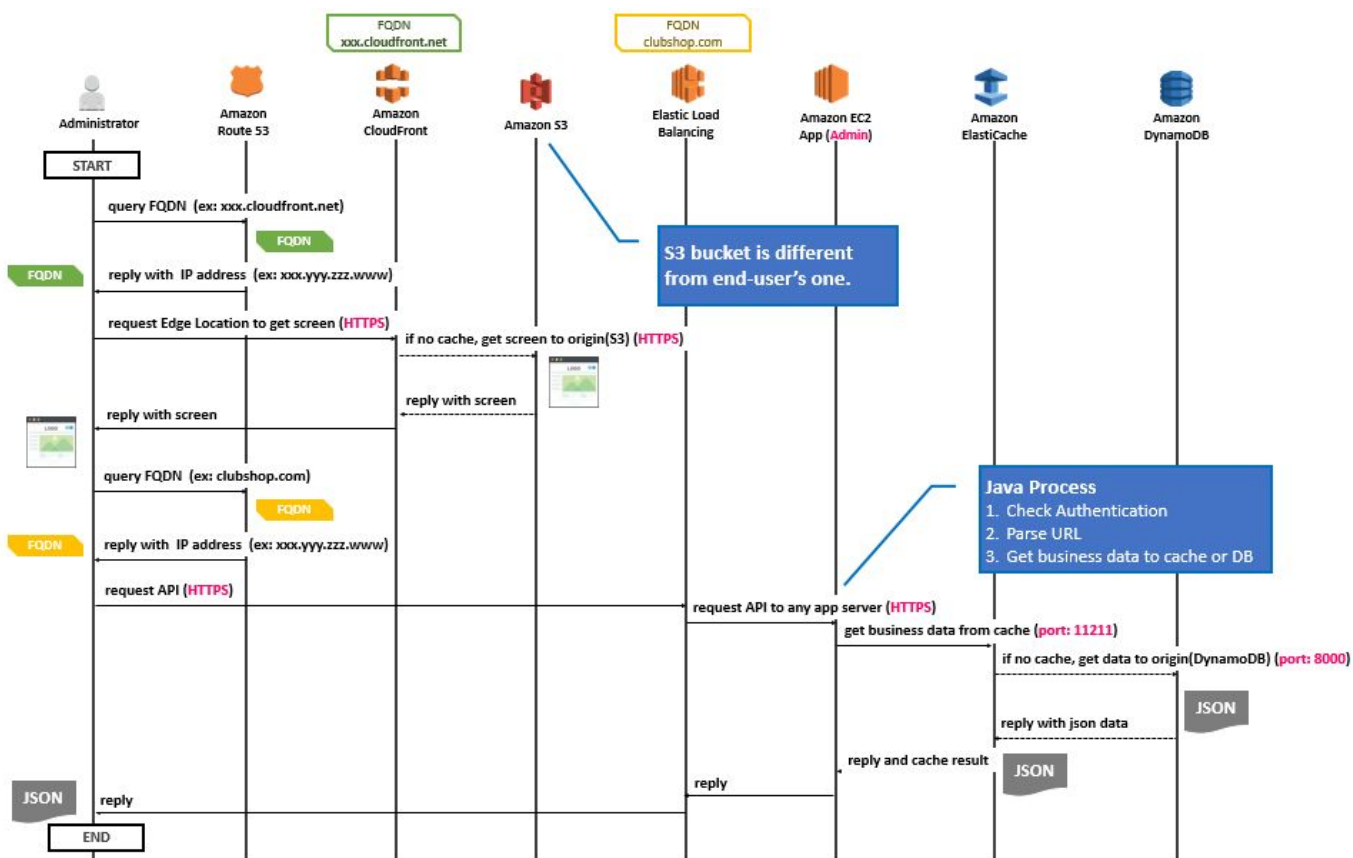
ID	Class	User	pattern	flows	Example of business
OL-A1	Online	end-users	Online Access Clubshop system	(Internet) -> [Route53] -> [CloudFront] -> (Amazon S3) -> [Elastic Load Balancer] -> [EC2 app server] -> [ElastiCache] -> (DynamoDB)	getOffer API, getCoupon API
OL-B1		Administrators	Online Access Clubshop system	(Internet) -> [Route53] -> [CloudFront] -> (Amazon S3) -> [Elastic Load Balancer] -> [EC2 app server(admin)] -> [ElastiCache] -> (DynamoDB)	createClub API, createOffer API
BT-A1	Batch	system	Log collecting	EC2 app server -> EC2 fluentd server -> S3 -> Glacier	

## OL-A1

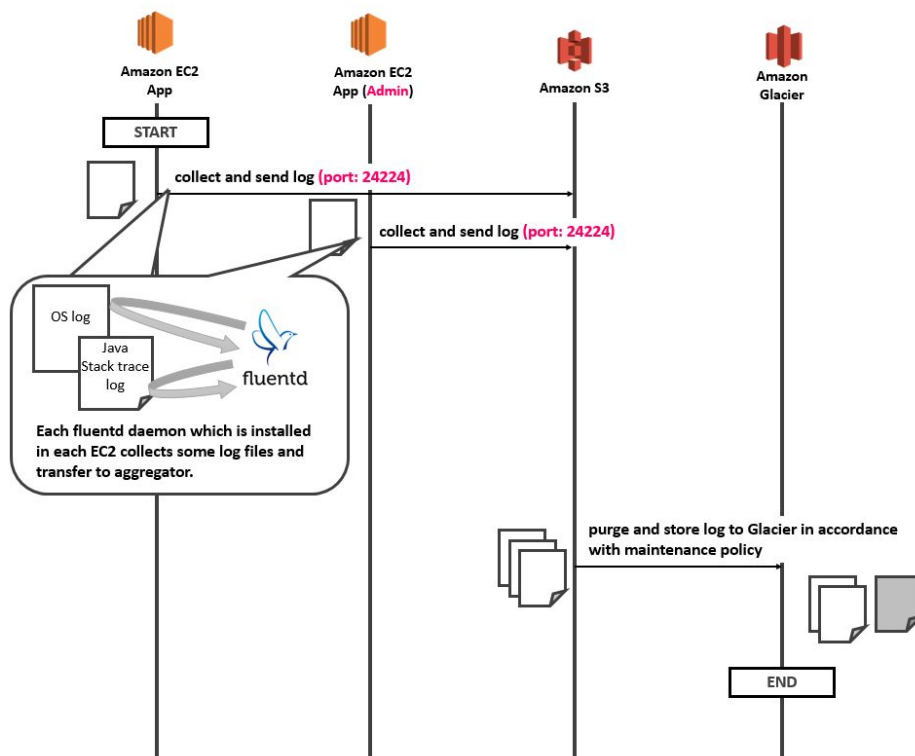




## OL-B1



BT-A1



## PERFORMANCE IMPROVEMENT METHOD

### STRATEGY AND POLICY

- This system should implement high performance architecture to meet demand changes of end-users .
- We should design performance improvement using load balancing techniques, flow control structure and caching function on AWS.

### LOAD BALANCING DESIGN

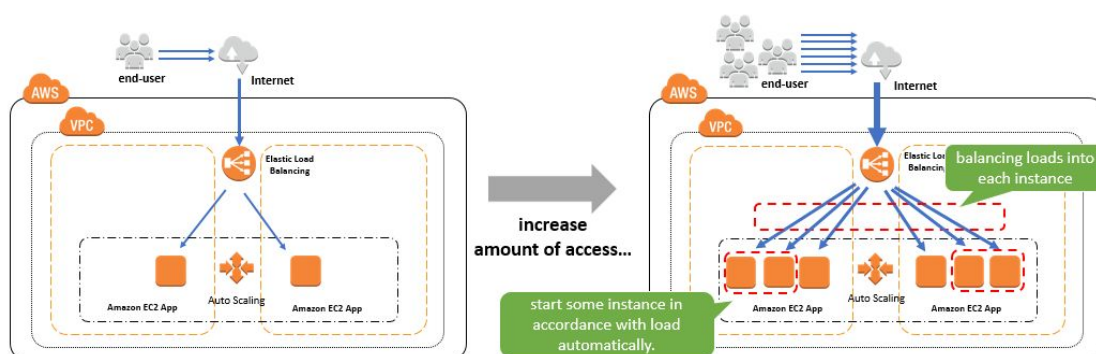
Some load balancer shall appropriately set up in this system to handle many transaction for online process. AWS prepares some type of load balancing method. These services are managed by AWS and have stretchability, we don't have to control these appliances ourselves. This means that it is pretty straightforward to scale these system anytime according to increasing traffic.

We think application server should be scalable because high load consists of end-user's transaction and these concentrate on application server. On the other hand, we'll have to take a rain check about using Load balancing for Administrator's server because of not many access amount and cost merit.

So, we decide to use load balancer with below policies.

#### Making a Plan for Load Balancing

- This system should use "**Standard Load Balancer**" because of no requirements for application layer control.
- Load Balancer should allows end-user to access with only **HTTPS** protocol.
- Load Balancer should locates in **DMZ Segments** (can be access from Internet).
- Load Balancer should use **health check request** to check whether each servers is active or not.



## FLOW CONTROL DESIGN

In this system, we don't have to design flow control because transaction flow is controlled by Amazon ELB automatically. Each request without session data like cookies is routed to the EC2 instance with the smallest load.

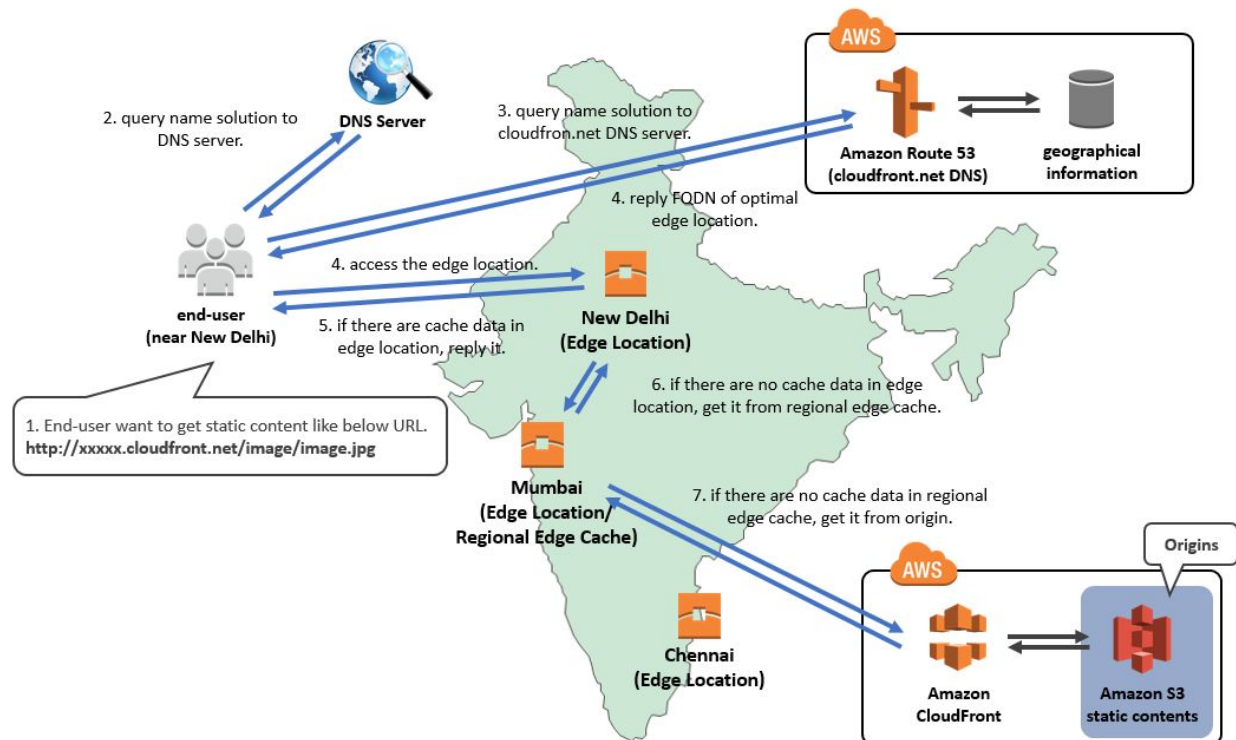
Reference:


<http://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-sticky-sessions.html>

## CACHING DESIGN

### Static Contents

To reply static contents like html file or some images swiftly, we take a strategy using CloudFront service on AWS. It is expected that this system deal with a lot of images like clubshop image or shop picture, so this design is desirable in terms of good UX(User Experiences) for enormous number of end-users.



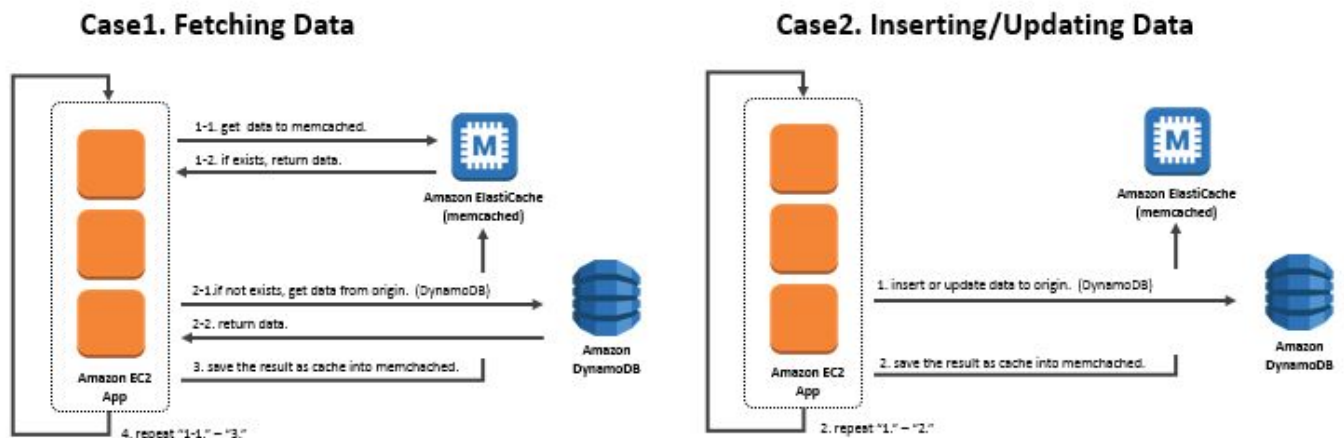


End-user can receive all static data from the nearest edge location containing cache data and we can reduce too many requests to origin.

## Business Data

Accepting caching technology offers the clubshop system great benefit with regard to returning all transaction response quickly and efficiently.

Above this reason, we decide to implement this technique in accordance with the following procedure.



We adopt memcached as Amazon ElastiCache for the following reasons.

- Simple mechanism
- Availability for flexibly Scaling out with demand changes
- Caching object

Using memcached, It is desirable to store volatile information to memcache. Here, we arrange main judge list whether each data should be stored in memcached or not in terms of consistency.

Data type	Data name	availability	reason
master	Club data	Yes	Update with no frequency.
	Store data	Yes	Update with no frequency.
	Category data	Yes	Update with no frequency.
	Offer data	Yes	Update with no frequency.
Transaction	Coupon data	No	Update frequently.
	Session data	Yes	Volatile data & reference only.

## SCALABILITY METHOD

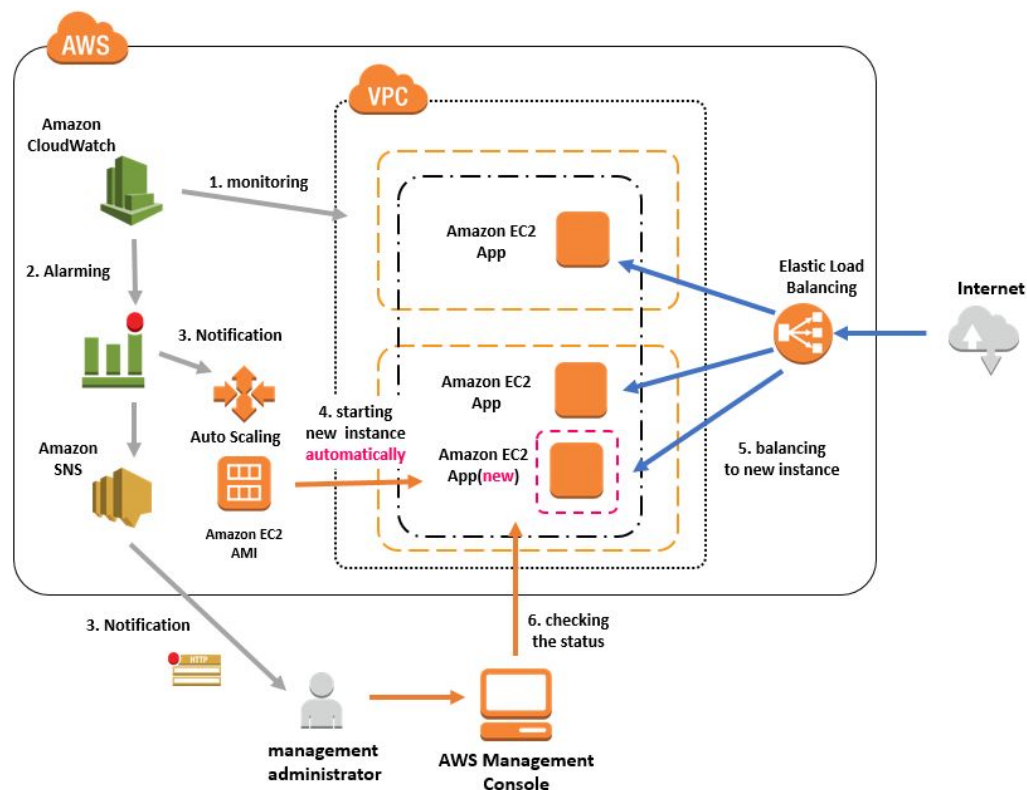
### STRATEGY AND POLICY

- Eliminating infrastructure capacity needs in advance, the system should implement scale up and scale down architecture for cost optimization with AWS service.
- We should utilize Elastic Load Balancing Service to distribute incoming traffic to application servers.
- To achieve a bigger scale, Loosely-coupled design should be taken in this system.

### SERVER SCALABILITY DESIGN

In this section, we design server scalability. The targets of servers are below.

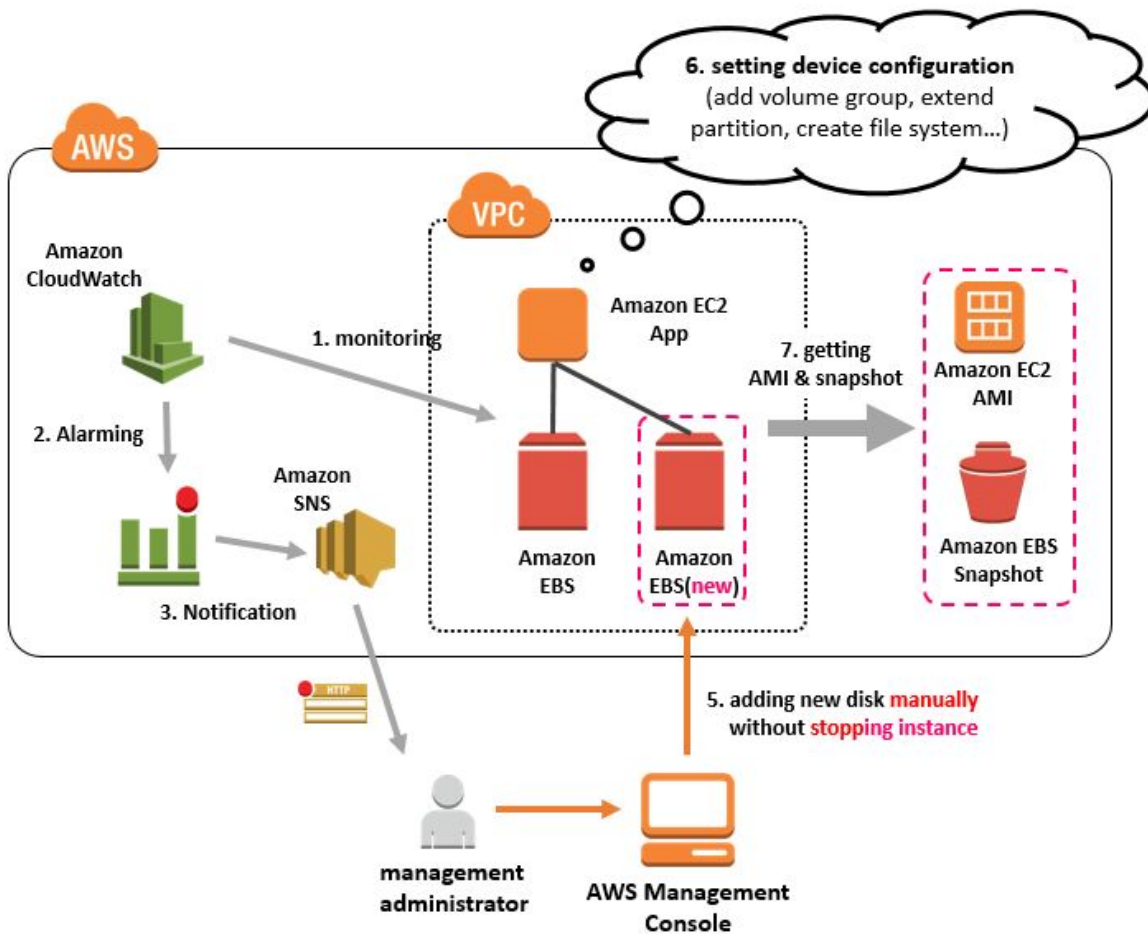
- EC2 instance App
- EC2 instance App(admin)
- EC2 instance fluentd
- ElastiCache (memcached)



## STORAGE SCALABILITY DESIGN

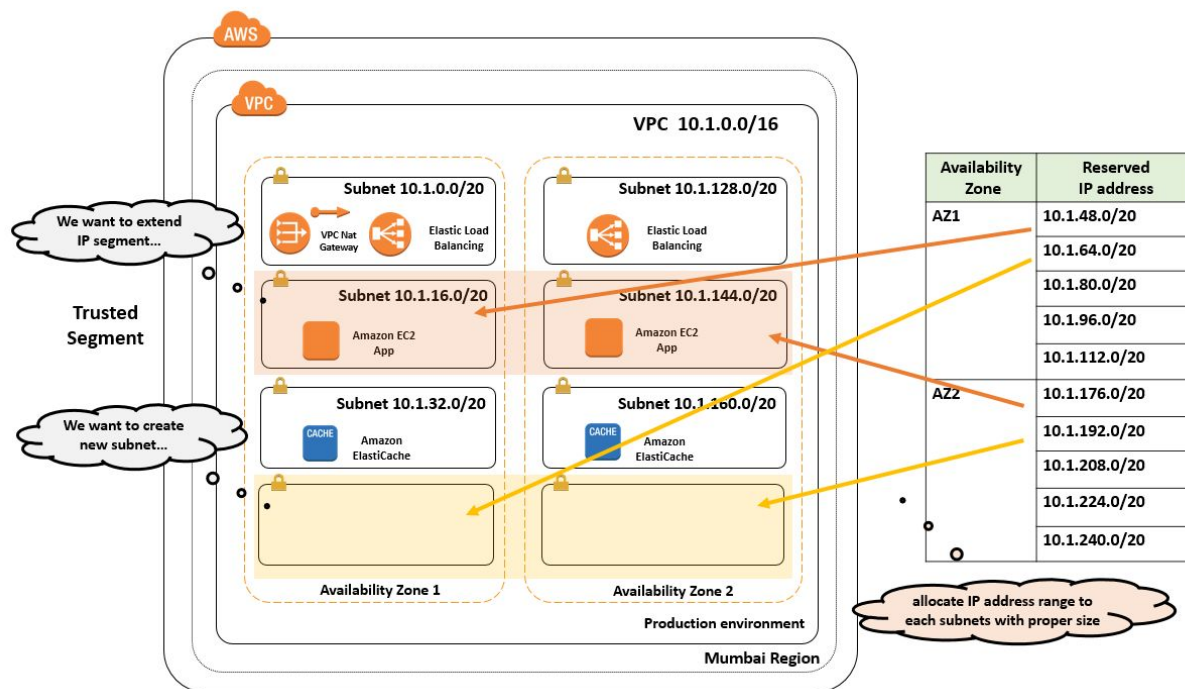
In this section, we design storage scalability. The targets of servers attached storage are below.

- EC2 instance App
- EC2 instance App(admin)
- EC2 instance fluentd





## NETWORK SCALABILITY DESIGN





## DURABILITY METHOD

### STRATEGY AND POLICY

Eliminate SPOF

### SERVER DURABILITY DESIGN

### NETWORK DURABILITY DESIGN



## DISASTER RECOVERY METHOD

### STRATEGY AND POLICY

Automate to make architectural experimentation easier

### DISASTER RECOVERY DESIGN

## SECURITY METHOD

### STRATEGY AND POLICY




- Security should be implemented at all layers, applications data, servers and network in consideration of time-consuming for management.
- Authentication framework for application user (End-user and Administrator) and management administrator (those who can use AWS console) should be designed appropriately for protecting any threat. (This document only refer to AWS account)
- Each layer security should be set with a least privilege.
- The system should acquired audit log to achieve traceability.
- In consideration of cost, we pass on implementation for WAF (Web Application Firewall) and antivirus software. In the future, we are going to design as necessary.

### AUTHENTICATION DESIGN

#### Application User

This section provides policy for controlling access to some AWS services in infrastructure layer.

- Following best practice on AWS, This system prohibit using “root user” for maintenance as a general rule.
- Each EC2 instances needs Access Key and Secret Access to access AWS service like ElastiCache, DynamoDB, S3 and so on. These authentication information are assigned by each user account, and they must be managed in responsibility of us. To secure firmly, we should allocate each account with each appropriate servers or user groups.
- We apply for policy of the least privilege.
- The users with the higher user authority level should be authenticated using MFA (Multi Factor Authentication) like “Google Authenticator”.
- like below table.

Group	Username (naming rule)	Resource	Description
root	(email address)	All service	Should be freezed(Not allowed to user)
administrator	admin1, admin2,...	All service	For infrastructure management.
developer	dev1, dev2,...		For developers who check some status of data or server. This account should be only allocated with query privilege of DynamoDB.
ec2app	app		For Application server to access DB and send some logs. We don't need to set configuration for ElastiSearch about IAM.
ec2batch	batch		For Batch server to perform backup and transferring data(DR).

Additionally, the system complies with following policies.

- Each accounts should be distinguished by difference of environment(production, development and backup).
- Each Password should be configured in compliance with below items. (Password policy can't be set to each account, only common settings.)
  - More than 7 length
  - Require at least one uppercase and one lowercase letter
  - Require at least one number and one non-alphanumeric character
  - Disable password expiration
  - Prevent password reuse

#### Reference:

Creating Your First IAM Admin User and Group

[http://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started\\_create-admin-group.html](http://docs.aws.amazon.com/IAM/latest/UserGuide/getting-started_create-admin-group.html)

## S3 API Permissions

<http://docs.aws.amazon.com/AmazonS3/latest/dev/using-with-s3-actions.html>

## DynamoDB API Permissions: Actions, Resources, and Conditions Reference

<http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/api-permissions-reference.html>

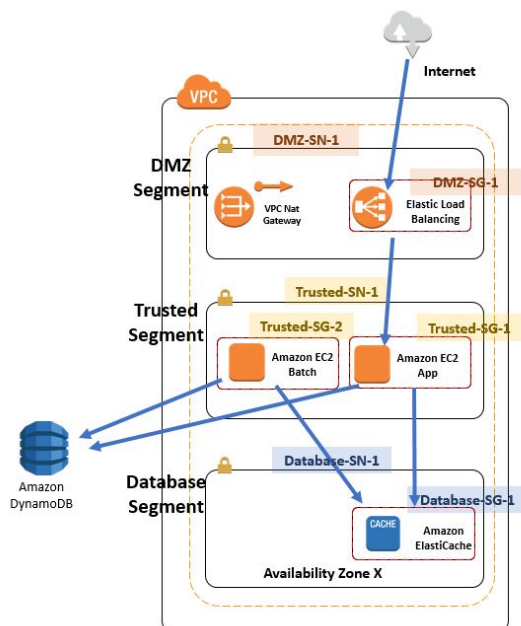
## DATA ENCRYPTION DESIGN

- Some business data should be encrypted to prevent from information leakage. This system should encrypts administrator's and user's password on Database.
- Inbound and outbound Transaction data on the internet should be encrypted as HTTPS protocols.

## NETWORK SECURITY DESIGN

Each segment should be accessed and restricted for only required transactions.

Especially, SSH protocol is permitted only for administrators network address.



Point	Point	Direction	Protocol	Permission
Security Group	DNZ-SG-1	Inbound	HTTP/HTTPS	Permit
		Outbound	all	Permit
	Trusted-SG-1	Inbound	SSH/HTTP/HTTPS	Permit
		Outbound	all	Permit
	Trusted-SG-2	Inbound	SSH	Permit
		Outbound	all	Permit
Subnet	Database-SG-1	Inbound	memcached	Permit
		Outbound	all	Permit
	DMZ-SN-1	Inbound	all	Permit
		Outbound	all	Permit
	Trusted-SN-1	Inbound	all	Permit
		Outbound	all	Permit
	Database-SN-1	Inbound	all	Permit
		Outbound	all	Permit



## ANTI-VIRUS DESIGN

All server

## AUDIT LOG DESIGN

## OPERATION METHOD

## STRATEGY AND POLICY

## BATCH

TIME SYNCHRONIZATION DESIGN

SERVER REBOOTING SCHEDULE DESIGN

LOG MAINTENANCE DESIGN

DATA BACKUP DESIGN

Data lifecycle

-> keeping 30days delete after 3years

SYSTEM BACKUP DESIGN

## MONITORING

SERVER MONITORING DESIGN

PROCESS MONITORING DESIGN

LOG MESSAGE MONITORING DESIGN

DATA THRESHOLD MONITORING DESIGN





## DEVELOPMENT & TEST METHOD

### STRATEGY AND POLICY

Test systems at production scale

We can create a production-scale test environment on demand

Apply AWS when security test

Split production and dev env using tag

### DEVELOPMENT ENVIRONMENT DESIGN

### LIBRARY MANAGEMENT DESIGN

## BILLING METHOD

Tips

SLBのウォームスタンバイ

セキュリティテストの事前深刻

Chaos monkey

制限増加申請が必要