Community based Social Network: Neighborhood

CMPE 281-01

BY

TEAM GROUP-7

POOJA PATEL **SWARNLATA KUMARI**

SIDDHARTH GORE **SURAJ BHUKEBAG**

Under the guidance of

Dr. Jerry Gao

May/2017

ABSTRACT

Building a safer and happier community requires its members to collaborate and help each other to analyze and solve community issues. A community social network provides a common platform for its members to share their views, request services and provide feedback on the community services. However, one of the challenge in existing social network platform is its people to people mapping which creates a dependency on the member and not on the role by which the member is associated with the community.

In this project, we propose a cluster based social community network as a solution for neighborhood community services. Each cluster requires its members to be associated to their role in the community which is user node. Services of the community are aggregated to the service nodes of the cluster and a messaging link is created between user nodes through service nodes to communicate with each other for a community service.

Keywords: Community social network, Cluster based social network, Neighborhood community

Table of Contents

Chapter 1. Project Overview

1.1 Introduction

The social network system supports three types of users: moderator, admin and users of the system. Moderator is responsible for the creation of cluster, whereas admin is responsible for the cluster administration activities. Admin should be able to view, edit, create and delete the clusters, services, users etc.

The cluster system supports three type of nodes: smart node, service node and user node. Service node represents the services for which user node interact with each other via message links. Example of service node in a neighborhood community is ‘leasing services’ having tenant, and leasing office user nodes associated with this service node. Users assigned to tenant and leasing office user node interact with each other for lease related services. Each service node combines a set of services which the node supports.

Smart Node - A smart node in the community system is a workflow which facilitates a set of decision rules to automate few services of the system and to provide configurations for inter cluster communications.

Service Node - A service node type is stored in the MYSQL database. For e.g. SN_LEASING_SERVICE (neighborhood cluster leasing services).

User Node - A user node type is stored in the relational MYSQL database. For e.g. UN_TENANT (neighborhood cluster tenant) is a type of user node which has all the tenants associated as users. Users linked to node UN_TENANT are tenants. Similarly leasing office users are linked to user node leasing office which can be stored as UN_LEASING_OFC. One user node can have multiple users assigned, and one user can be mapped to multiple user nodes.

Linking service node and user node, below diagram present service node and user node linkage in a neighborhood cluster:

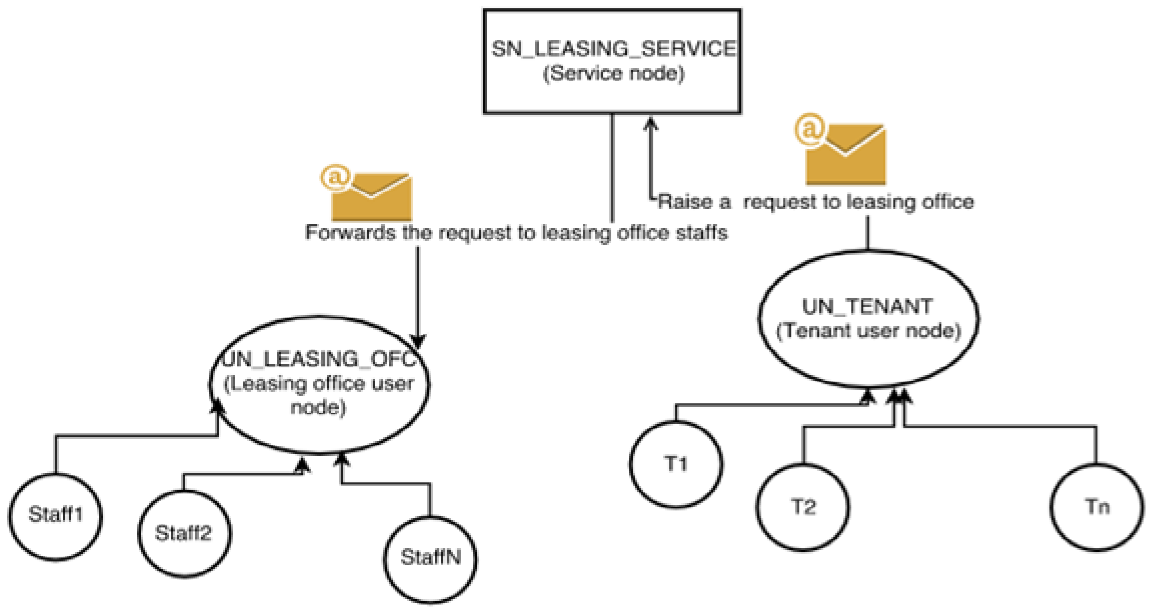

SN_LEASING_SERVICE
(Service node)
Raise a request to leasing office
Forwards the request to leasing office staffs
UN_TENANT
(Tenant user node)
UN_LEASING_OFC
(Leasing office user node)
Staff1
Staff2
StaffN
T1
T2
Tn

Figure 1.1

Similarly, moderator and admin are the two type of user nodes who associated to each other though cluster management service node. Moderator are the users mapped to user node UN_MODERATOR and admin are users mapped to user node UN_ADMIN. The association can be represented as below:

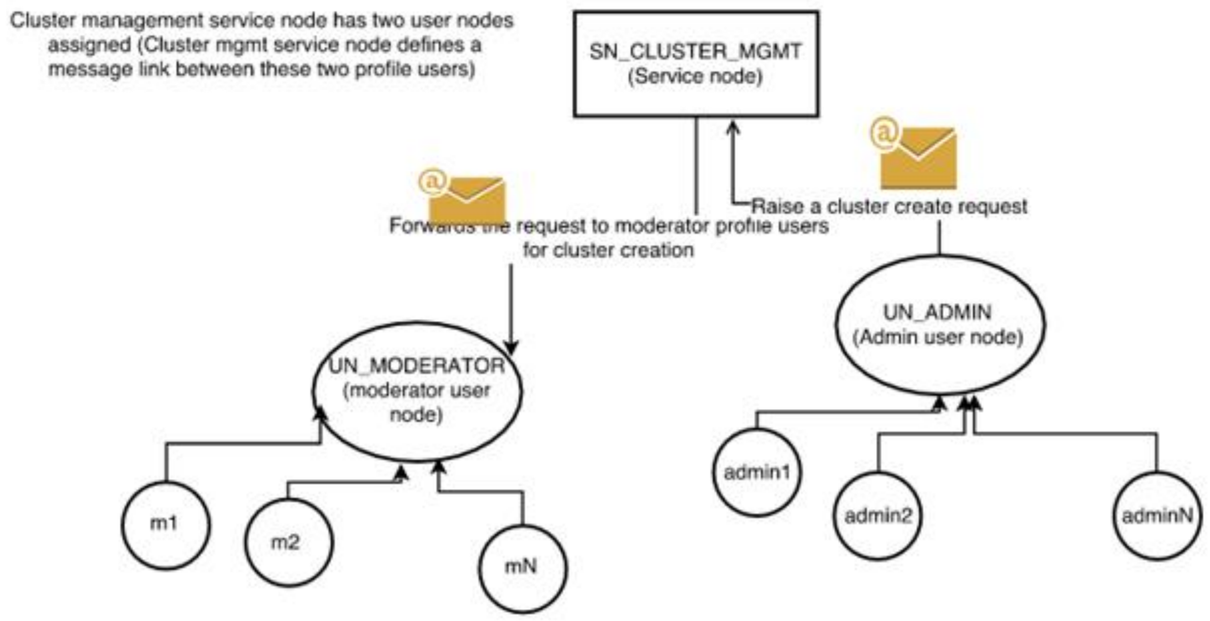

Cluster management service node has two user nodes assigned (Cluster mgmt service node defines a message link between these two profile users)
SN_CLUSTER_MGMT
(Service node)
Raise a cluster create request
Forwards the request to moderator profile users for cluster creation
UN_ADMIN
(Admin user node)
UN_MODERATOR
(moderator user node)
m1
m2
mN
admin1
admin2
adminN

Figure 1.2

The user linked to admin node (admin1, admin2…adminN) can login to the admin dashboard of neighborhood cluster and is able to manage all the clusters associated with him.

1.2. Social Network Functional Components

The functionalities provided by the social network community network can be broadly classified into three groups:

- **Social Network Modeling Component:**

  Graphical User Interface: Three types of users i.e. administrator, moderator and the users should be able to login to a graphical interface to use cluster services and configurations. The users should be able to login by providing valid credentials.

  Node Creation and Maintenance - The admin should be able to add, update, delete and configure the user, service and smart nodes.

  Message Links - The admin should be able to configure the message link among nodes.

- **Messaging Component**:

  The system should implement a messaging component to provide message linkage between nodes. A mail server can be configured to provide features which avoids people to people messaging.

- **A social Network Cluster:**

  Cluster Creation and Maintenance - The system should support basic cluster services to create, update, configure, view and delete a cluster.

Chapter 2. Project Architecture and Infrastructure

2.1. System Architecture

2.1.1. High Level System Architecture

Each neighborhood cluster is deployed in AWS elastic beanstalk configured to auto scaling. Diagram shows multiple cluster user accessing WEB UI client pages on EC2 instance which in turn process the request to server on beanstalk. Several clusters could share same resources at schema level. An email notification is sent from spring boot API to enable messaging notifications. Relational database MySQL is used for maintaining master data and NoSQL data store MongoDB is used for maintaining user data, services, decision data for services. AWS cloudwatch is used for monitoring system resources on AWS.

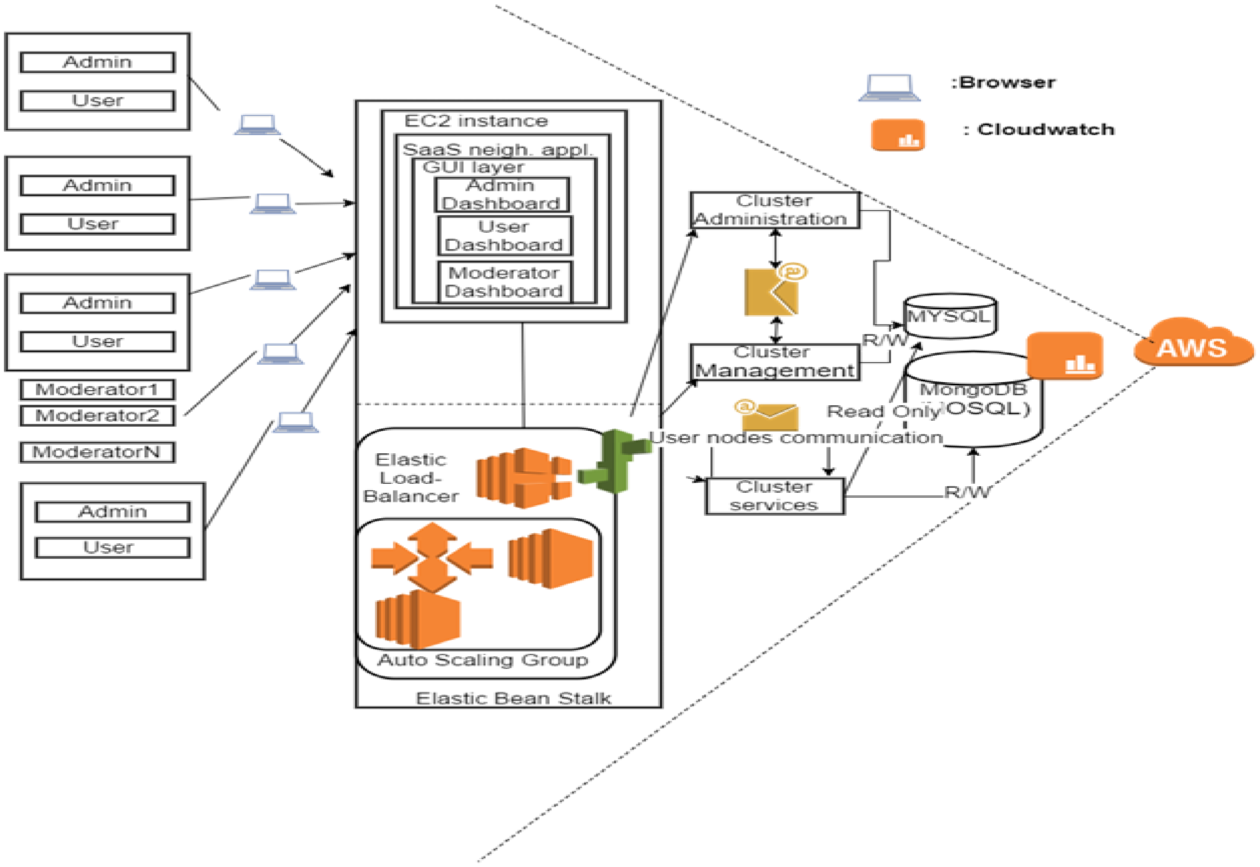


Figure 2.1 High Level System Architecture

2.1.2. Layered System Architecture

Figure 2 provides the architecture of neighborhood social network cluster community. The various layers of the architecture are User layer, Client layer, Network layer, Security layer, Message layer, Application layer, Compute layer, Data layer and management layer. Users in user layer are the community users (admin, and user) and system moderators. The request generated from the user UI is routed to the security layer in amazon cloud infrastructure through user browser and network. The messaging layer triggers automated notification and action based on the smart node decision rule configuration of the cluster. Spring Boot REST API application image is deployed to amazon elastic beanstalk. Two types of database are used in this application: MYSQL relational database for storage and retrieval of structured information such as cluster detail, service node detail and smart node decision rule configuration etc. MongoDB NoSQL database is used to store unstructured user data, user community forum activity data and the details of service incidents raised by the users. The management layer of the cluster makes use of cloud watch to monitor the clusters and to manage clusters billing at more granular level. The system also features analytics reports on cluster profiles and usage etc.

- **Users Layer:** This layer consists of Users from community who will interact with the neighborhood community social network.

- **Client Layer:** This layer is basically the application front end system that is used by moderator, admin and users. It has different dashboards for every type of users. It provides authentication and session management services. Clients makes HTTP requests for accessing servicing.

- **Network Layer:** Both wireless and wired network processes the requests using various communication protocols such as TCP (Transport control protocol), SMTP (Simple mail transfer protocol) etc.

- **Security layer:** Authentication, session, database access, data read write access etc. are handled by the security layer of the community network system.

- **Messaging Layer:** Java libraries are used to trigger email notification to users and to process workflow requests of the communication system.

- **Application Layer:** Java Spring boot framework is used to create RESTful APIs and the WAR file of the application is deployed to auto scaling configured amazon beanstalk.

- **Compute Layer:** An amazon EC2 instance is used to provide client tier (WEB UI) to the users of the community software. The instance communicates with amazon beanstalk to access RESTful APIs which access the MYSQL and NOSQL databases configured in the beanstalk environment.

- **Management Layer:** Amazon AWS cloudwatch alarms are set to trigger email notifications and to take automated action on network traffic threshold, CPU threshold, Storage threshold etc.

- **Database Layer:** This layer encapsulates all the database related transaction services and exposes them in the form of REST APIs. This keeps database related code independent of business logic.

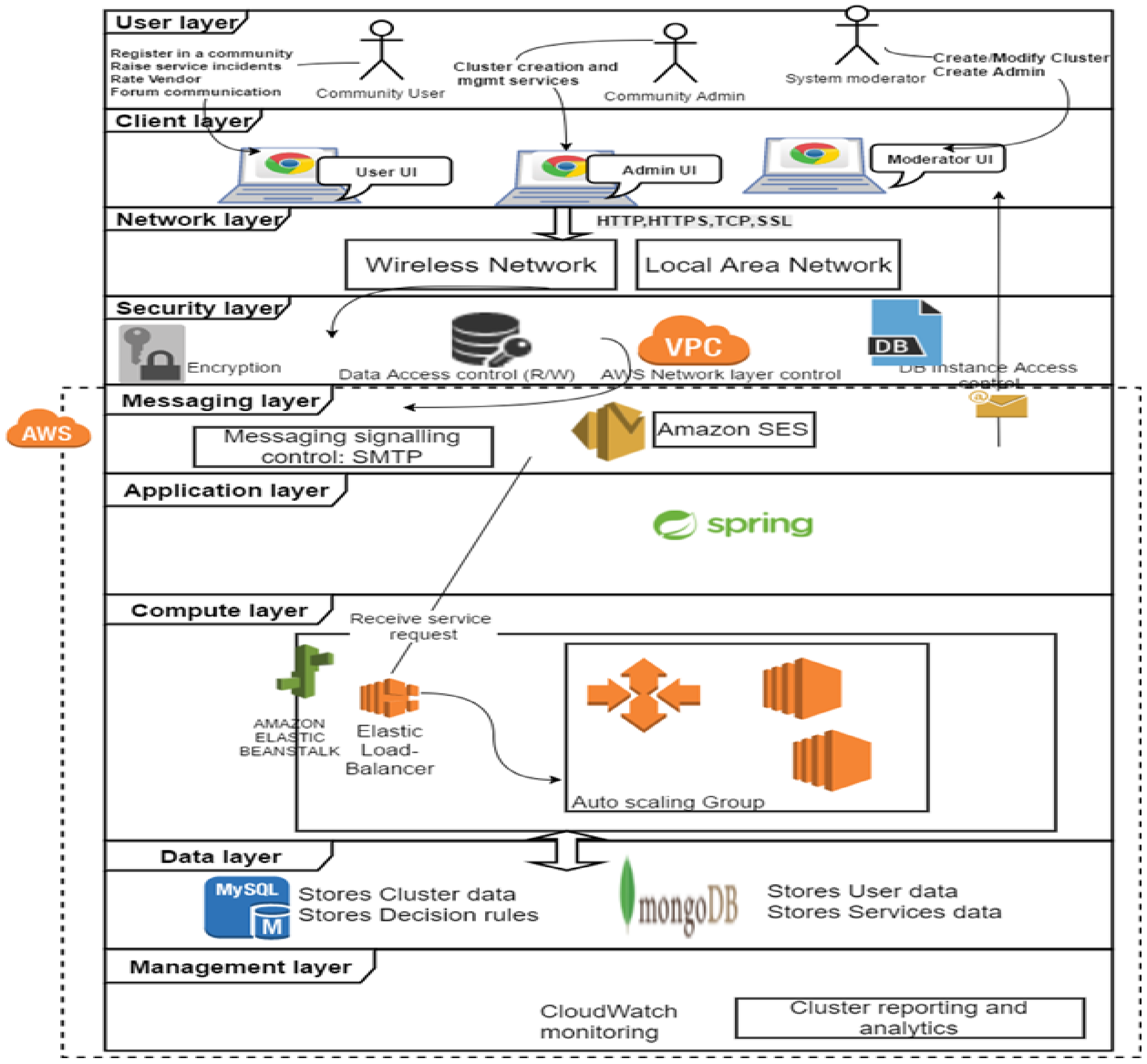


Figure 2.2 Layered System Architecture

2.1.3. Cloud Layer Architecture

- Neighborhood community application will interact with Spring Boot Framework components in the cloud.

- Spring data JPA is used for relational DB interaction.
- Spring data MongoDB is used for NoSQL DB interaction.
- Spring Cloud AWS context provides access to storage and email services

Spring Cloud AWS core provides security and configuration setup services.

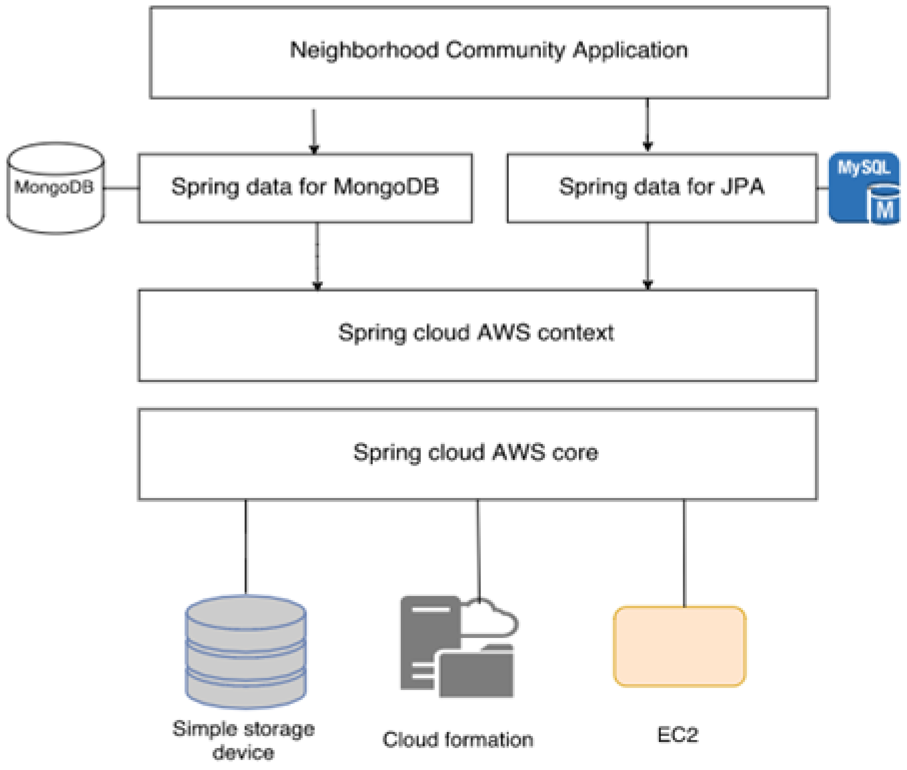


Figure 2.3 Cloud Layer Architecture

2.2. System Infrastructure

Neighborhood community SaaS application nodes: Smart node, service node and user node are the software components implemented using spring boot REST API framework. A community is comprised of community admin and community users. Moderator, the third type of user are the system administrators of the community platform responsible for creation and management of the clusters. The community admin is responsible for the cluster services and user administration and community users are the clients requesting services in the community system. The SaaS application is deployed on Amazon cloud infrastructure, a EC2 instance hosts the web pages and is configured to handle user traffic. The server tier is deployed to elastic beanstalk with auto scaling configured to handle RESTful API requests. The basic infrastructure of the community is as displayed in the image below:

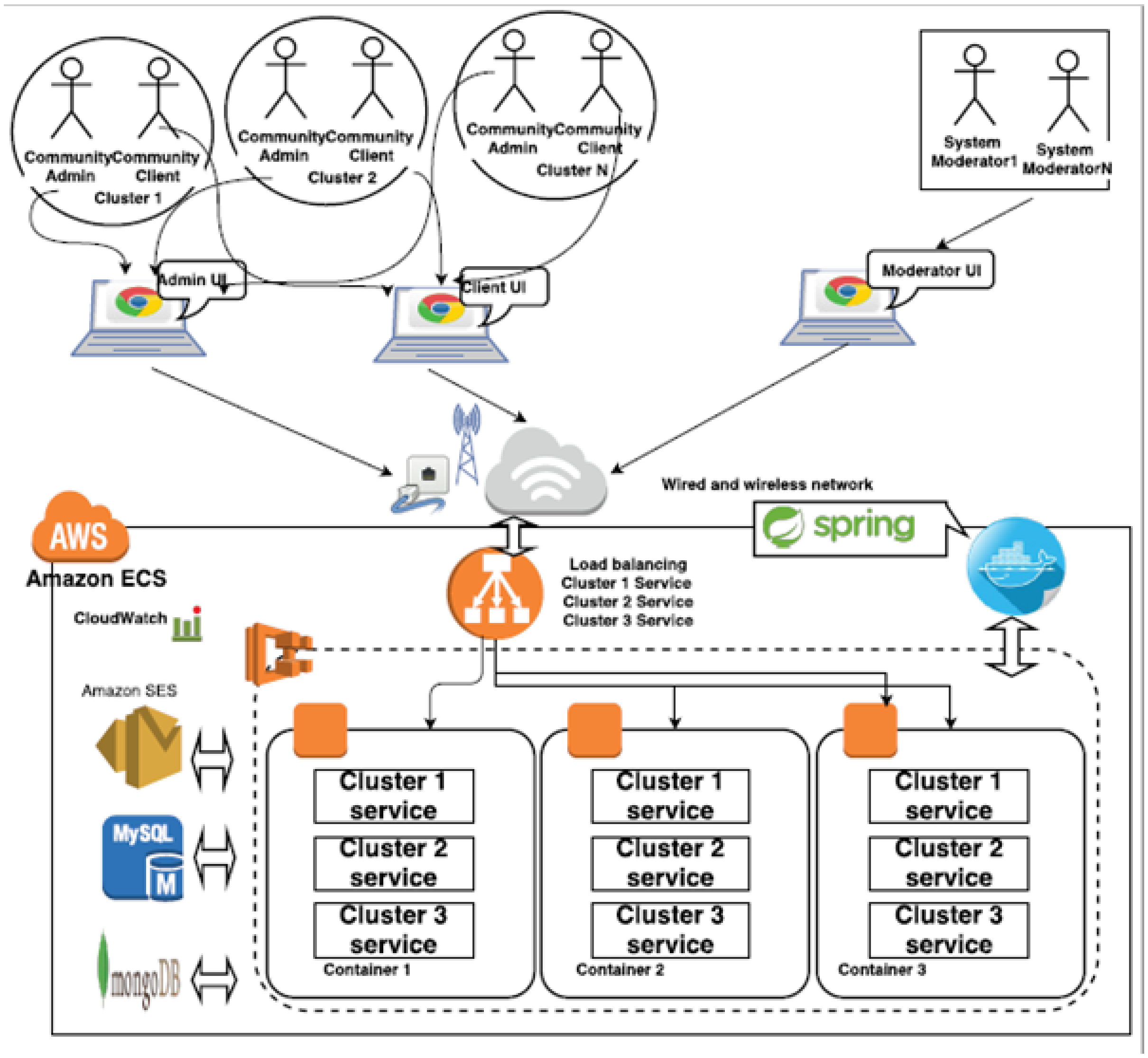


Figure 2.4 System Infrastructure

Chapter 3. System Component Design and Analysis

3.1. User Layer Design

3.1.1. Admin Dashboard

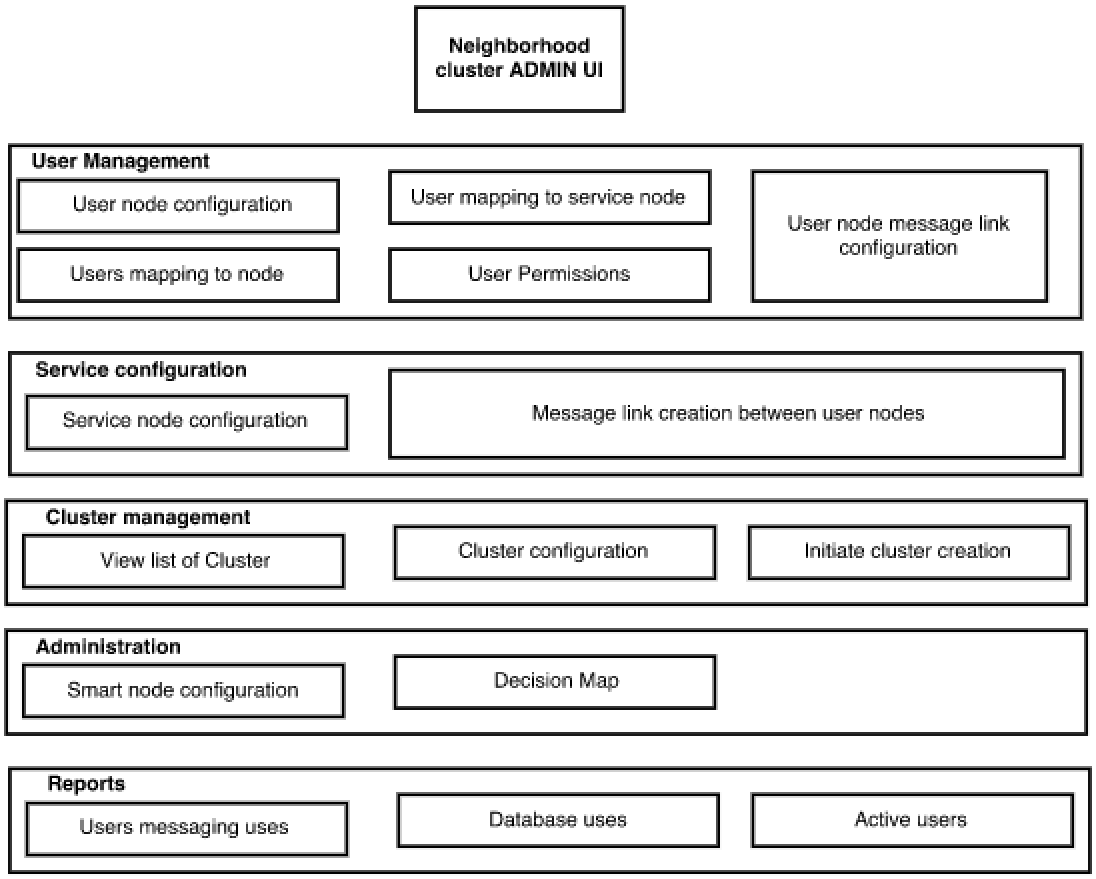


Figure 3.1 Admin Dashboard

● **User Management:** This component provides functionalities for admin such as User node configuration, mapping user to service node, providing messaging link, setting permissions for user.

● **Service Configuration:** Admin can configure services for a cluster through this component and can define communication link between the users.

● **Cluster Management:** Cluster initiation request, deletion requests, viewing all the clusters and configuring clusters is provided in this component.

● **Administration:** Admin configures smart node through this component and defines the decision map for serving user requests.

● **Reports:** This component provides options to admin such as checking all active users in the cluster, all users and inactive users that are part of clusters.

3.1.2. User Dashboard

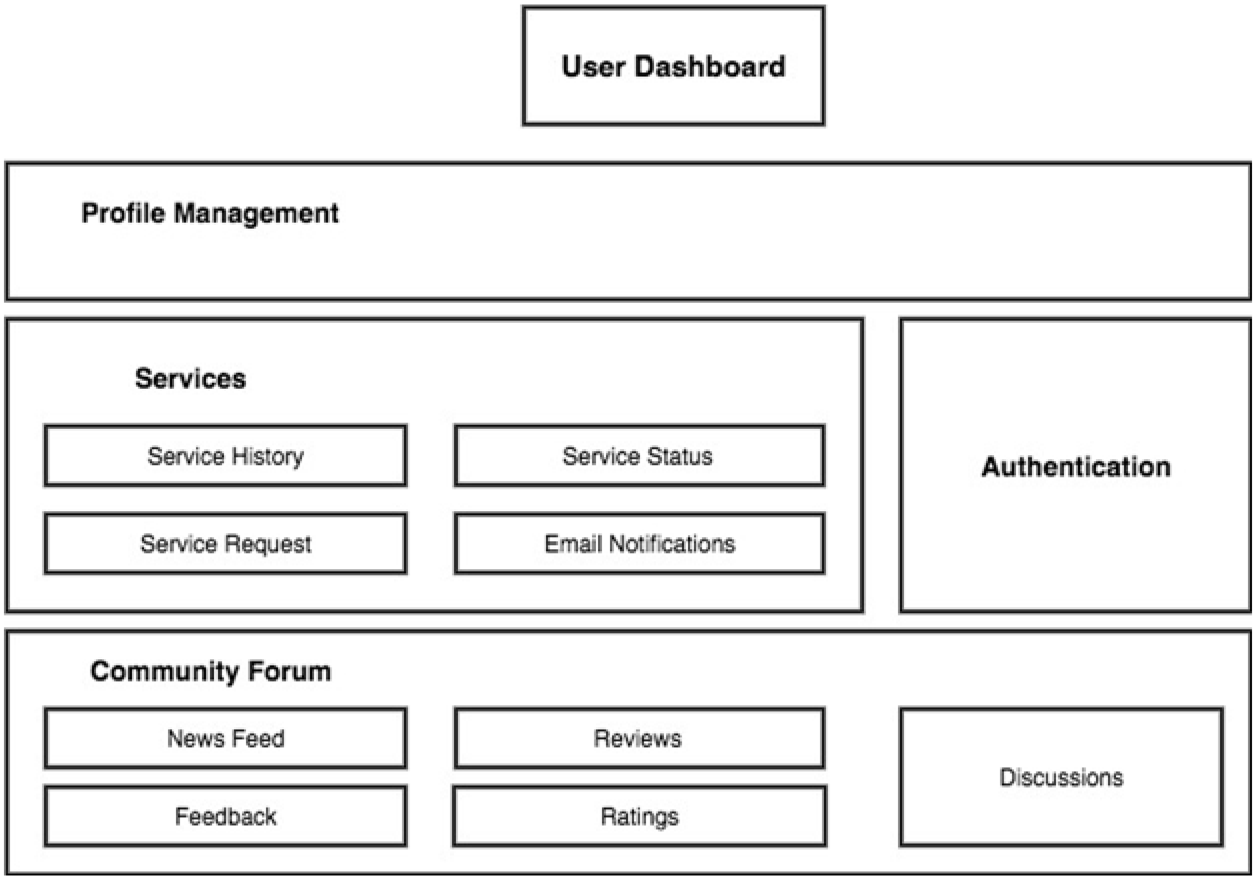


Figure 3.2 User Dashboard

● **Profile Management:** This section contains functionalities with respect to management of user profile such as change First name, Last name, Email Address, Telephone Number and Password. After Changes are done user will click on the Submit Button and the user details will get modified in the database.

● **Services:** This section deals with the services offered by the community. The user can raise a request for a new service, can view the status of the raised requests, can see the list of service requests raised in the past and for each service, email notifications would be sent to the

Community Admin.

● **Authentication:** This is the first page that appears when a user clicks on the User section. In this section, the user credentials will be verified. If the username and Password are present in the database, then only the access would be granted to the user. As soon as the user is logged in the session would be maintained for that user throughout.

● **Community Forum:** In this section, the user will have the facility to rate, review and provide feedback for the services offered by that community. He can also view all the discussions of other users as well and can see their feedback and ratings too.

3.1.3. Moderator Dashboard

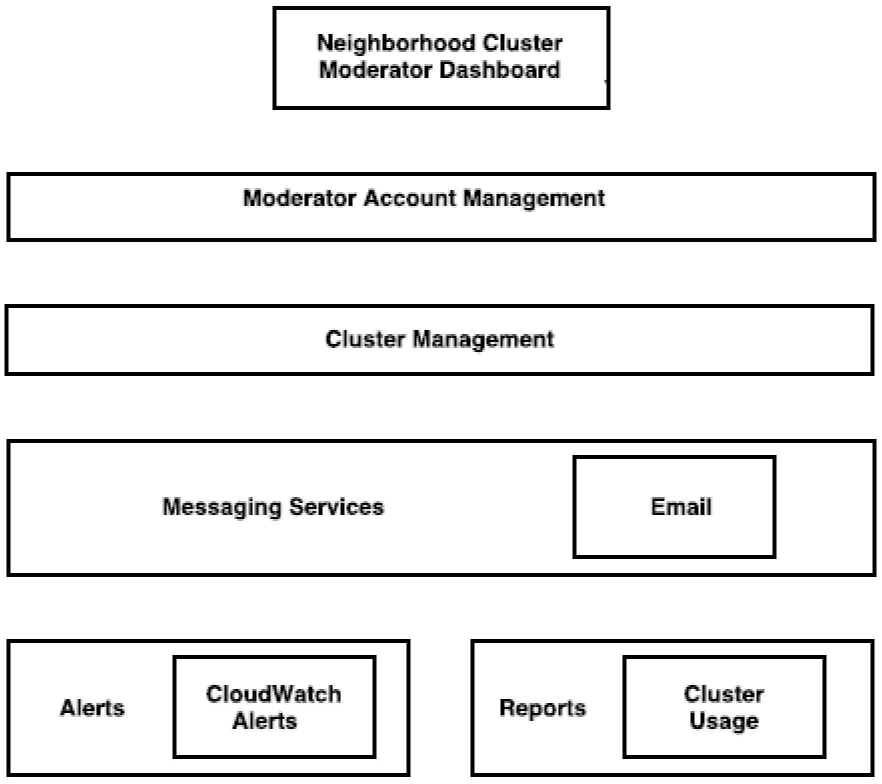


Figure 3.3 Moderator Dashboard

● **Moderator Account Management:** This component provides functionalities for a moderator to update personal details such as name, contact information and email information.

● **Cluster Management:** Moderator is provided with options to create a cluster based on admin's requests, delete a cluster on admin's request and view all the clusters in the system.

● **Messaging service:** Email is used to provide messaging service for cluster creation requests, cluster deletion requests.

● **AWS Cloudwatch Alerts:** This component is used for monitoring cloud resources for the application and alerts are being sent to moderator based on the configuration.

3.2. Service Layer Design

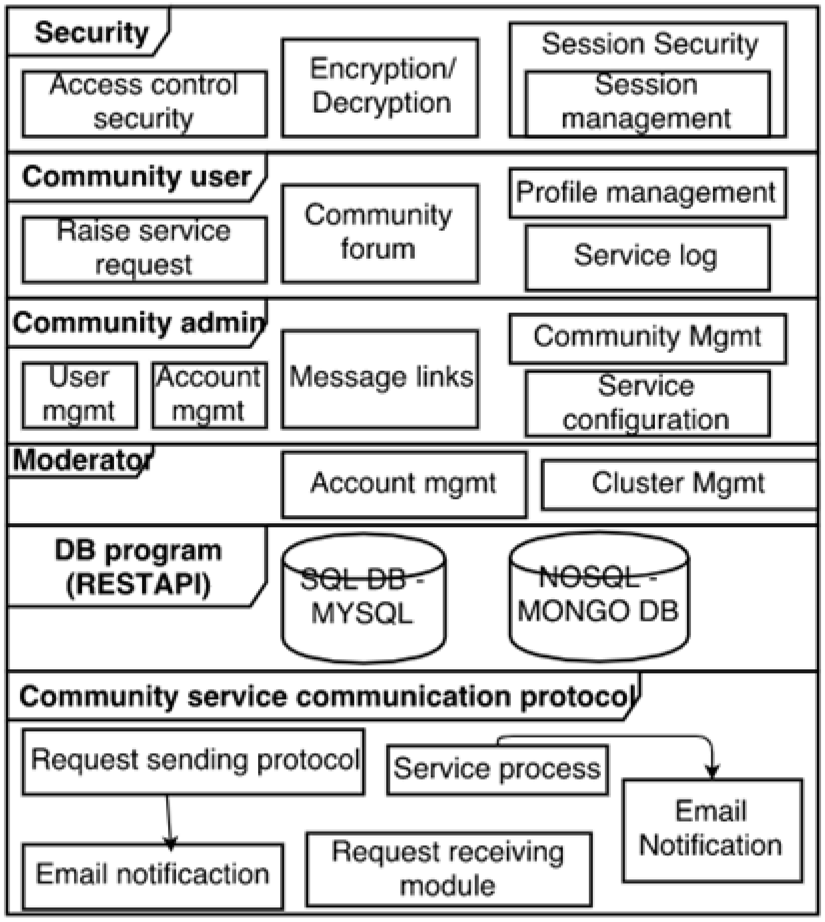


Figure 3.4 Service Layer Design

The service layer design describes Neighborhood Community Network's services at every level. At the top we have access control, encryption and decryption and session management for providing secure access to the application.

Next, we have different types of users accessing application. Depending on the type of user, application provides different functionalities. Moderator has options for cluster management and account management. Community admin has options for community management, configuring services for the community, defining messaging links among clients and account management.

User has options to request for a service, profile management, service logs and access to forum of the community.
Application makes use of both relational database and unstructured database for storage. Email notifications are sent to respective entity whenever

3.2.1. Database Layer Design

The master data of the system such as cluster, user node type, service node etc. are stored in relational database system MYSQL, whereas the user data such as service list, users, messaging links, auto email templates etc. are stored in MongoDB. Relational database is used for storing structured data such as cluster, service node, user node etc. Whereas, MongoDB is used to store unstructured data such as user data, service data, message links and raised service requests details. Service request and user data may contain the images, audio or video files, storing these in MongoDB makes it easier to process and retrieve the information in the form of key value pairs.

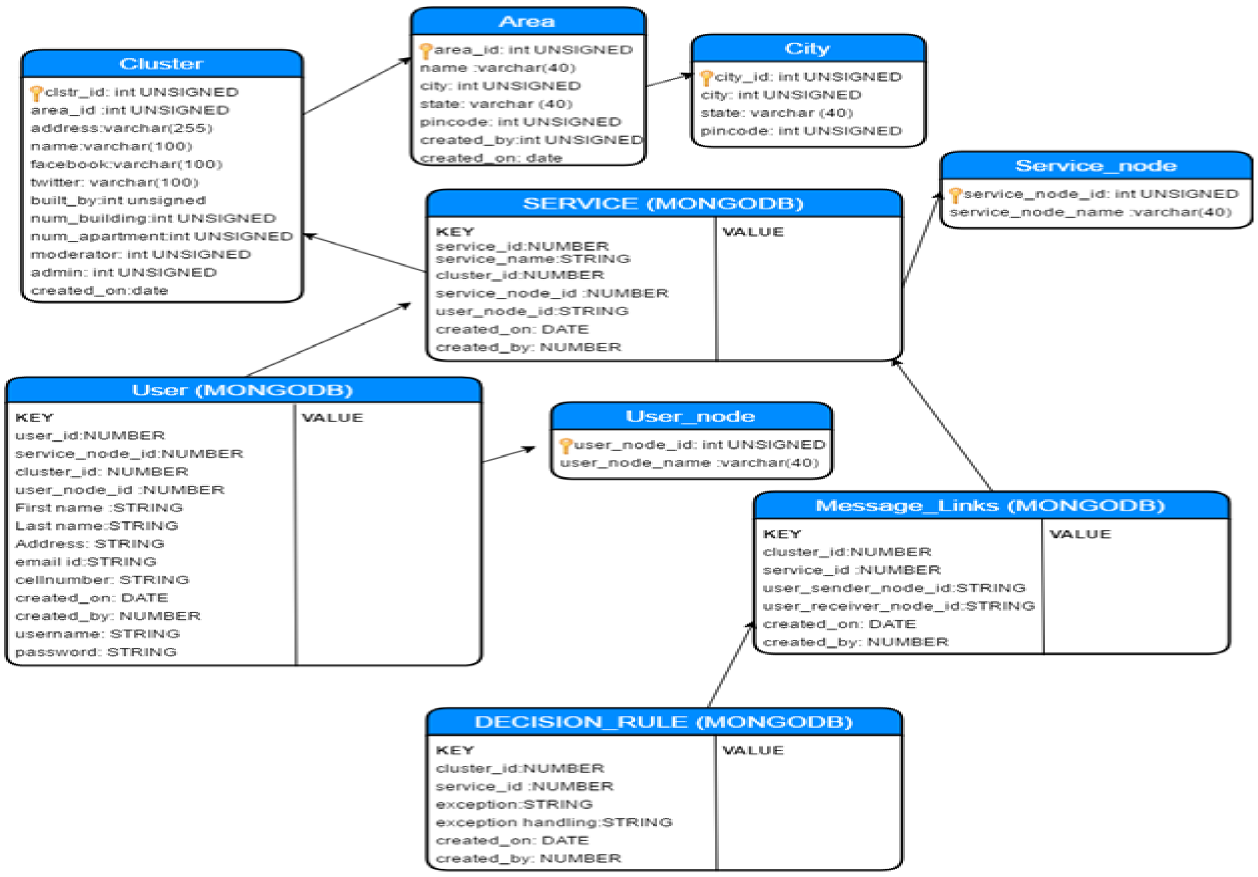


Figure 3.5 User Database Layer Design

3.2.2. MySQL Database Design

Cluster Data:

Three tables cluster, area and city are created, the cluster table stores the details such as cluster name, area id, number of buildings, number of apartments, admin requesting the cluster, and moderator who creates the cluster. The city table is used as a value table and is prepopulated with the list of cities and state information. The area table is created by the administrator which is mapped as a drop down in cluster creation form.

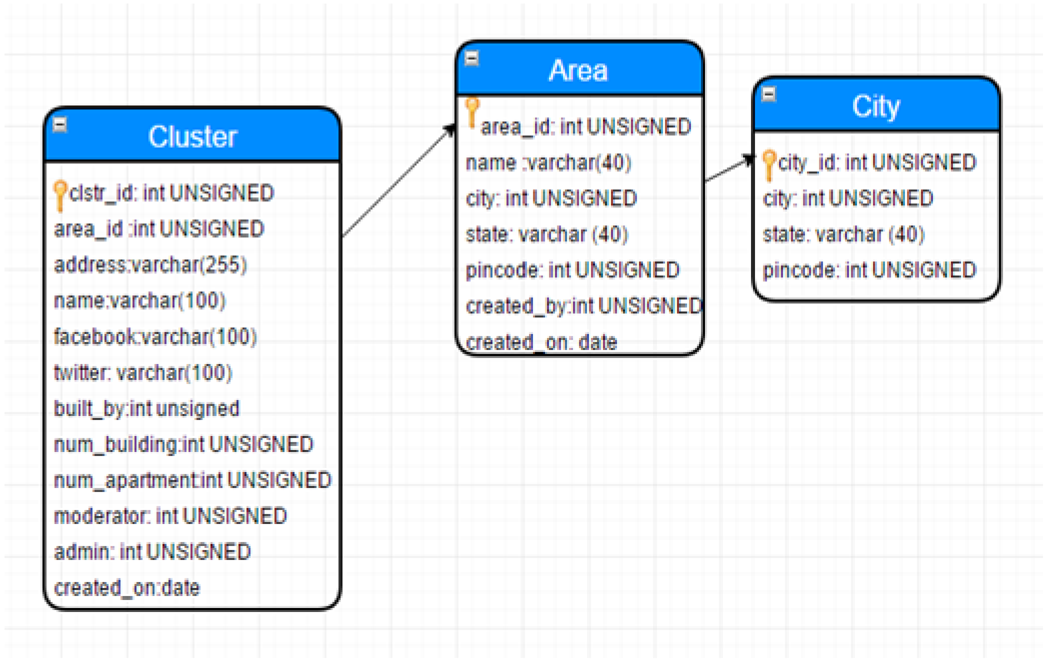


Figure 3.6 Cluster Tables

Service node data:

This relational table stores the service node name and ID, the example of service node can be maintenance, rent management etc. ID is an auto increment integer assigned to each service node. A service node can be associated with multiple clusters.

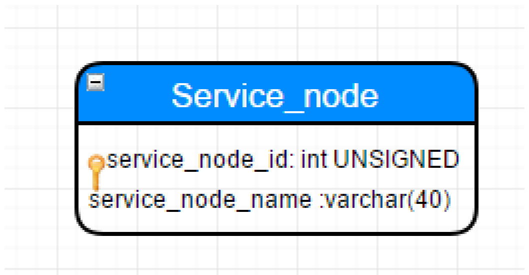


Figure 3.7 Service Node Table

User Node Data:

This relational table stores the user node name and ID, the example of user node can be tenant, leasing office, third party vendor etc. ID is an auto increment integer assigned to each user node. A user node can be associated with multiple clusters.

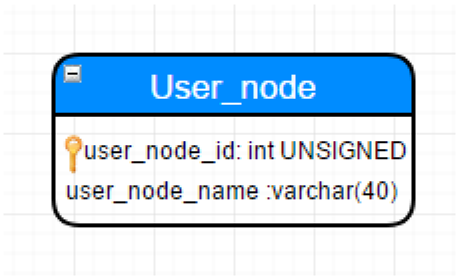


Figure 3.8 User Node Table

3.2.3. MongoDB database design

Users data:

User data can be unstructured as it may contain photos, links etc., a MongoDB database is used for user data storage. Each user can have multiple user nodes assigned. In our neighborhood community, a user is a person leasing an apartment. For example, apartment 100 is one user of the neighborhood community. Let us consider this user as u1. There can be two user nodes in community tenant and tenant representative. User u1 can be linked to both tenant user node as well as tenant representative node.

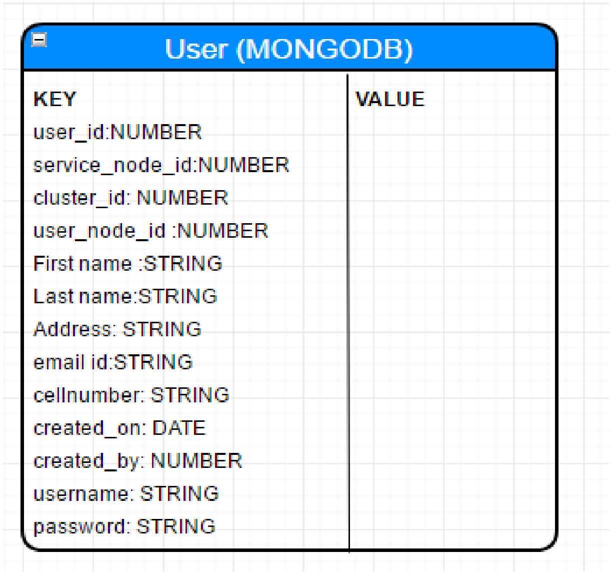
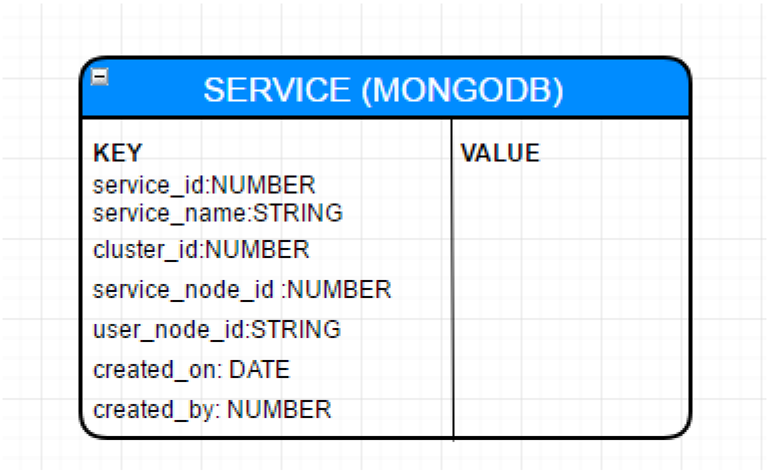


Figure 3.9 User table

Services:

The service node is assigned multiple services in a neighborhood community, for e.g. Maintenance node can have plumbing, painting etc. services assigned to it. The services of a node can be added/removed based on the community changing requirements. Also, there will be various service requests raised for each service by community users. Hence, services of the community are stored in MongoDB service database.



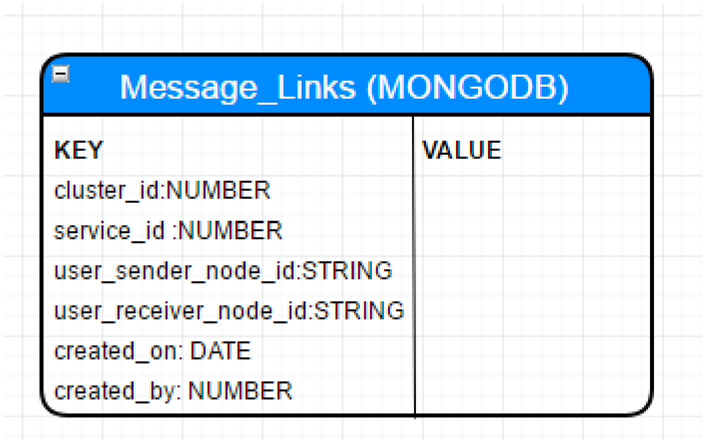
SERVICE (MONGODB)

KEY	VALUE
service_id:NUMBER service_name:STRING cluster_id:NUMBER service_node_id :NUMBER user_node_id:STRING created_on: DATE created_by: NUMBER	

Figure 3.10 Service Table

Message Links:

Each service request requires notifications to be sent and received between stakeholders of the community. A message link defines a communication configuration between user nodes. Hence a message link is configured for each service of the neighborhood community.



Message_Links (MONGODB)

KEY	VALUE
cluster_id:NUMBER service_id :NUMBER user_sender_node_id:STRING user_receiver_node_id:STRING created_on: DATE created_by: NUMBER	

Figure 3.11 Message Link Table

Chapter 4. Project Design

Requirements are categorized based on different users in the system. The system supports Moderator, Admin and User actors. The roles are defined based on responsibilities and those are defined as below.

Moderator: Acts like super user for whole system. Responsible for maintaining clusters.
Admin: Every cluster is managed by an Admin, who is responsible for Managing Cluster, Managing Services and Managing Users in cluster.
User: A user will be able to avail the services and see the service history for a cluster.

Below is snapshot of use cases for each role in the system.

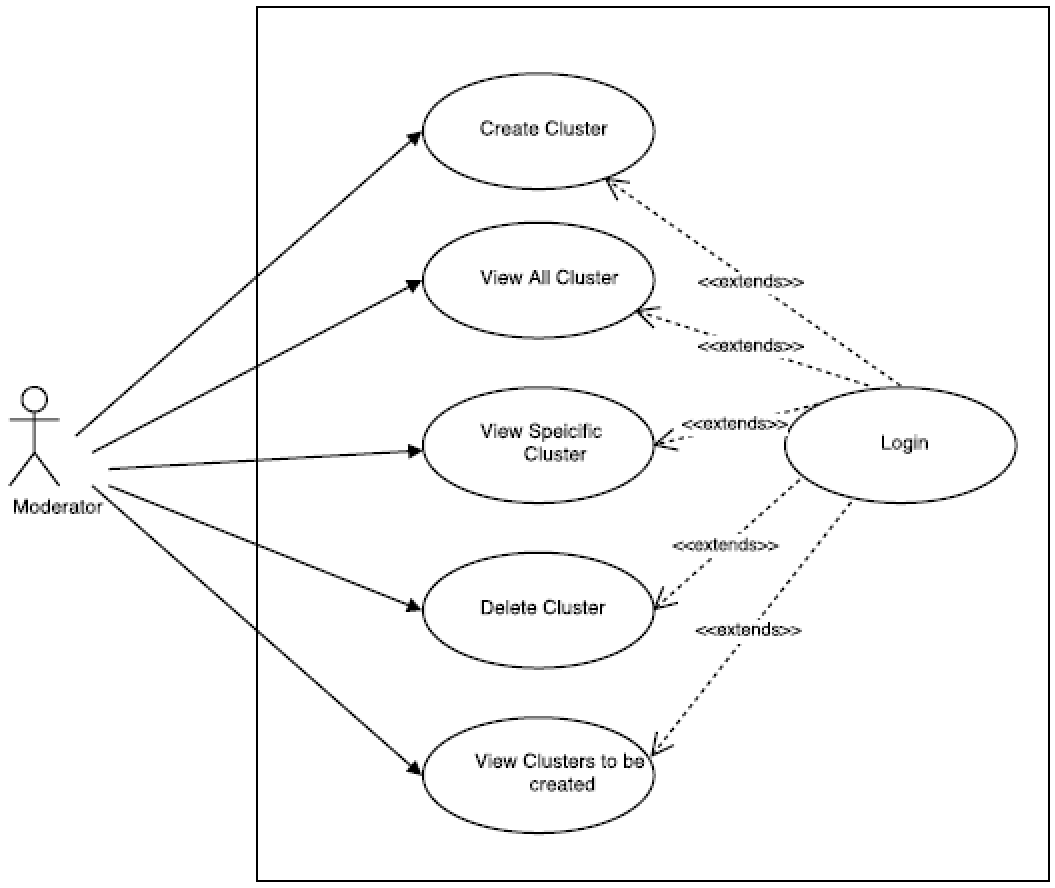


Figure 4.1 Moderator Use Case

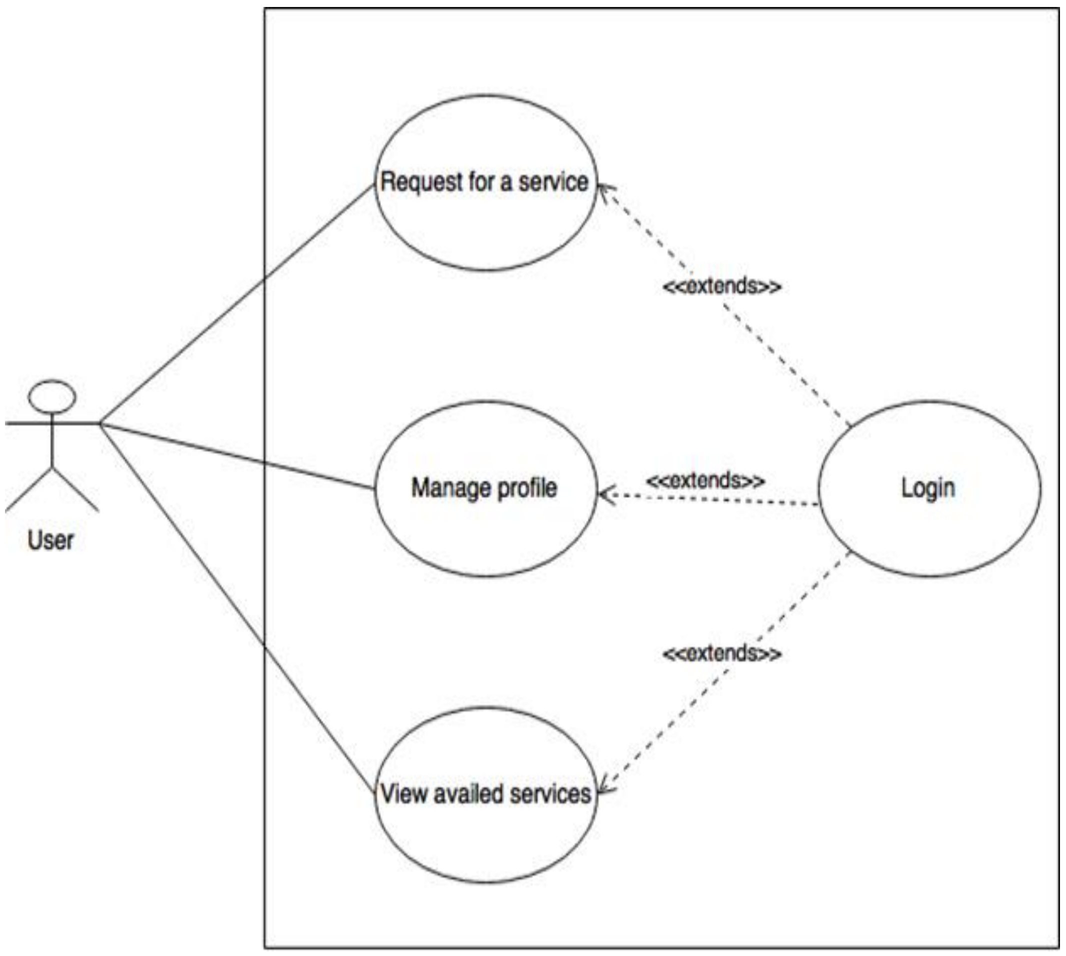
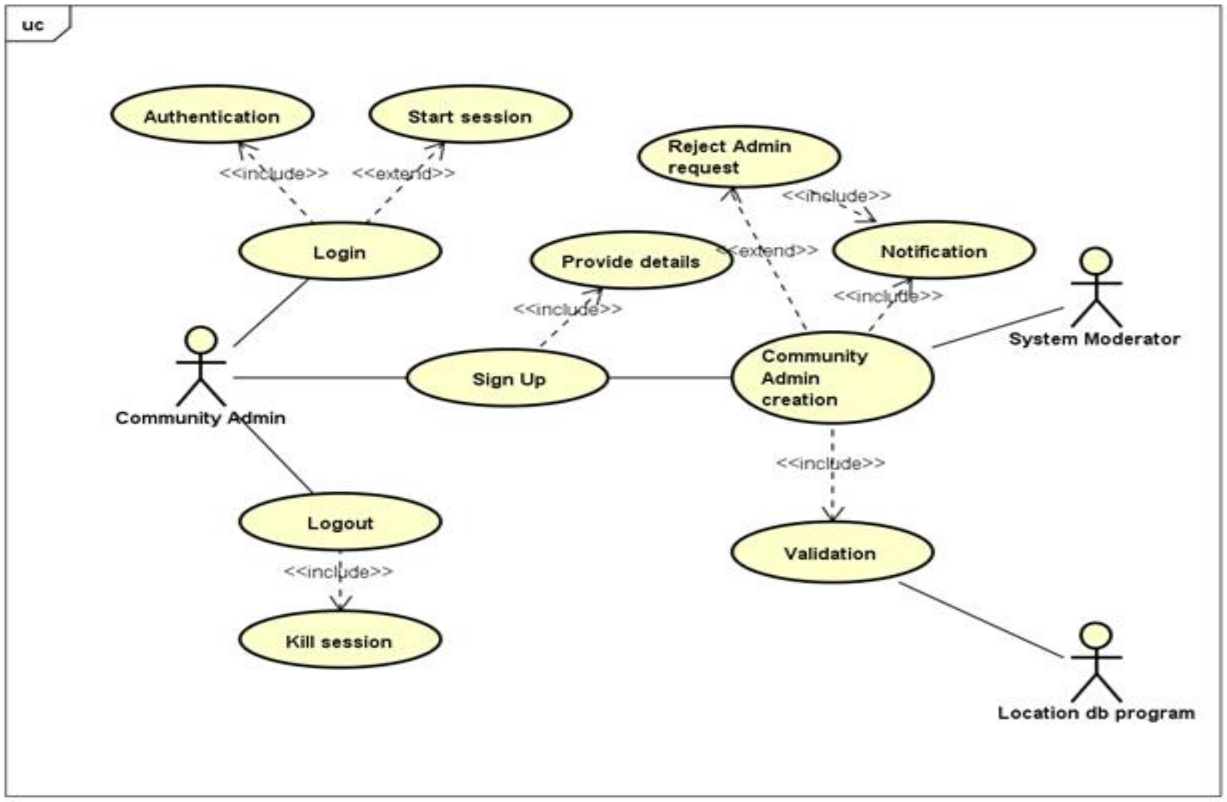


Figure 4.2 User Use Case



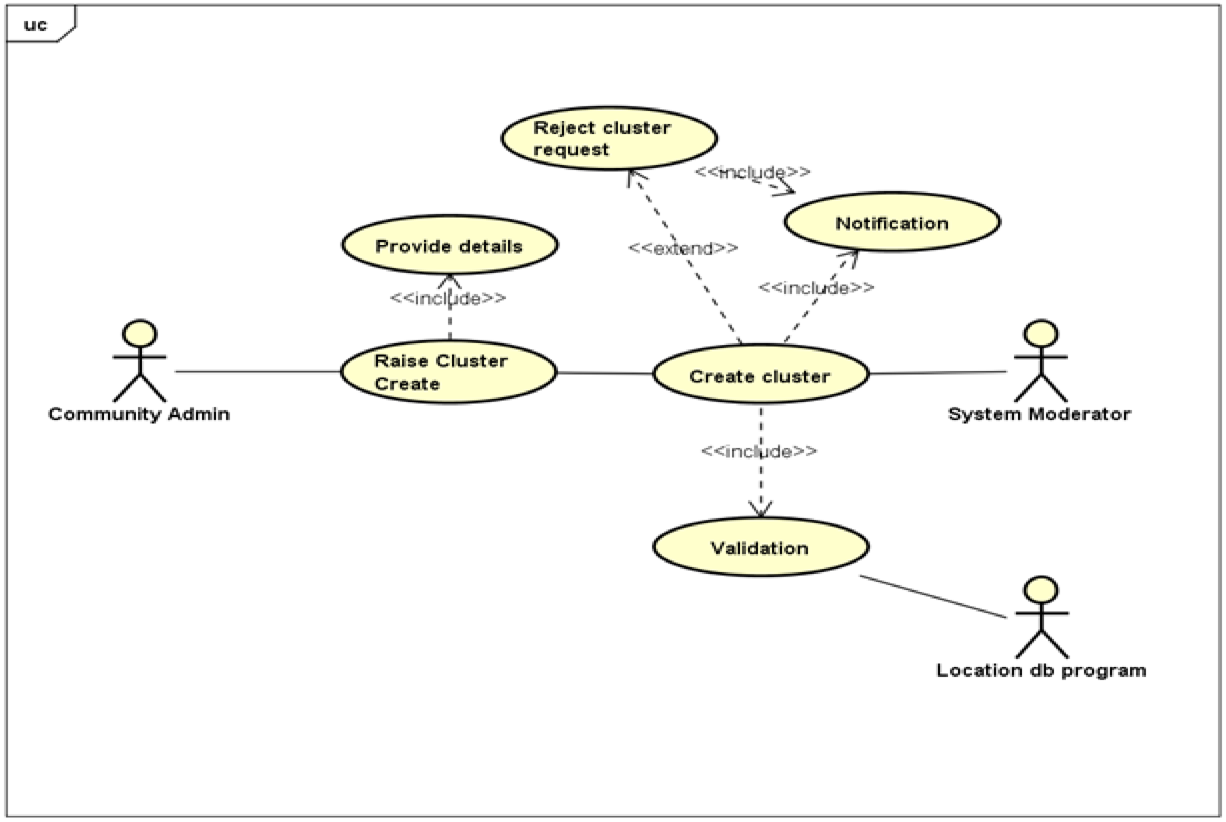


Figure 4.3(a)&(b) Admin Use Case

Sequence Diagram of Specific Functions

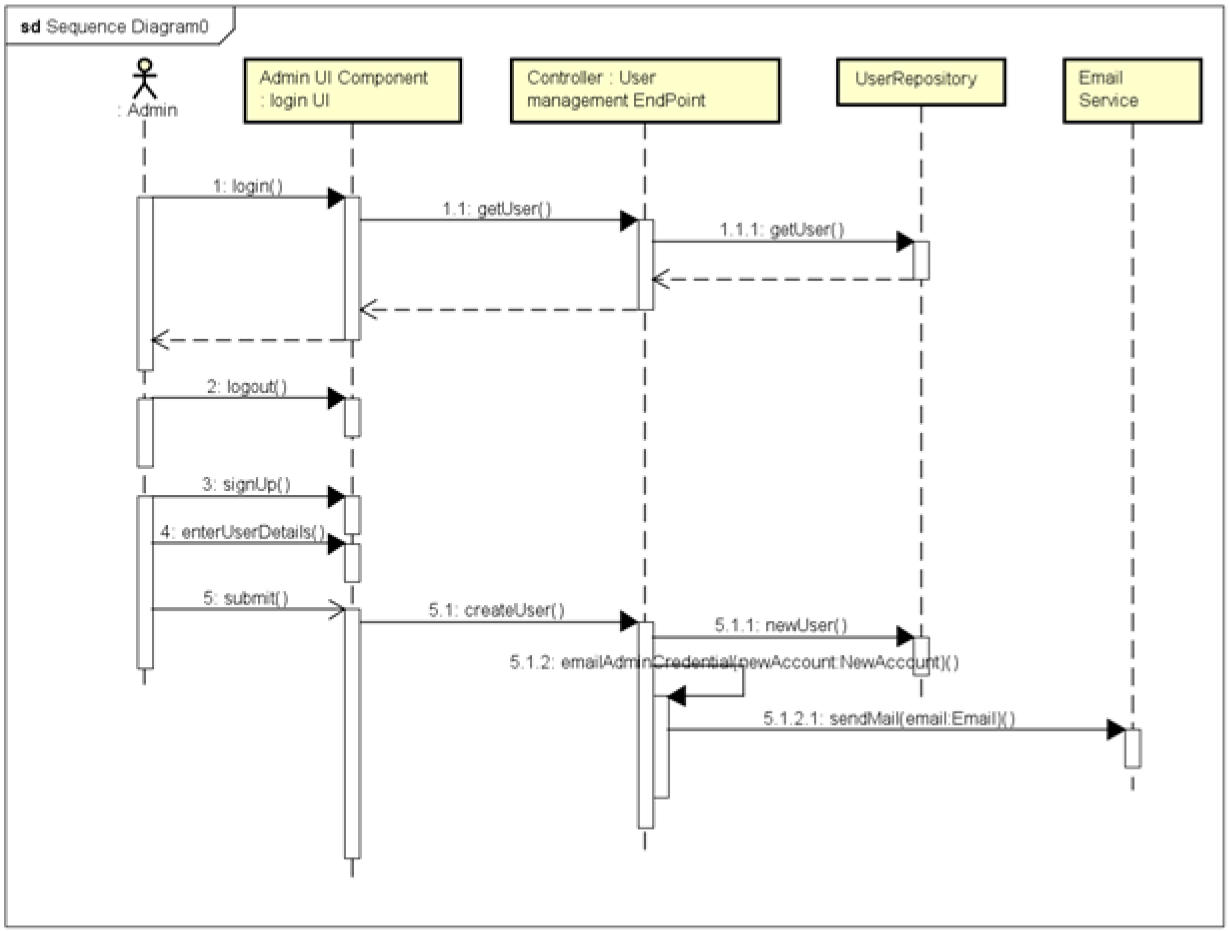


Figure 4.4

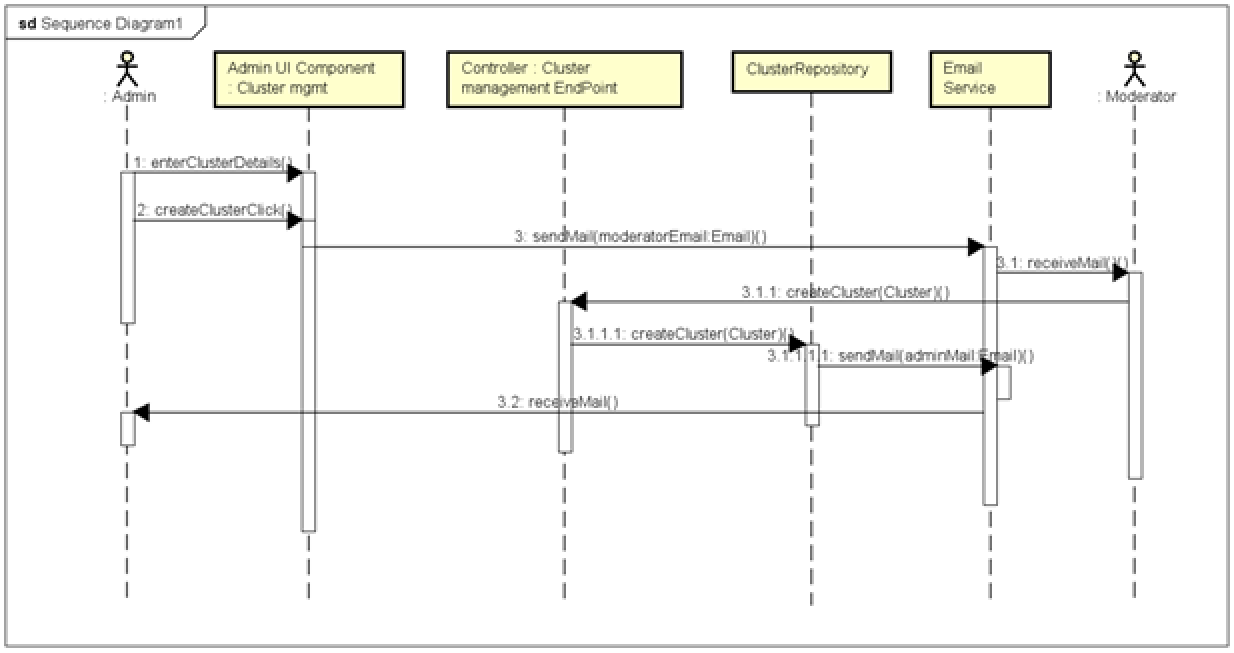

sd Sequence Diagram1
: Admin
Admin UI Component : Cluster mgmt
Controller : Cluster management EndPoint
ClusterRepository
Email Service
: Moderator
1: enterClusterDetails()
2: createClusterClick()
3: sendMail(moderatorEmail:Email)()
3.1: receiveMail()()
3.1.1: createCluster(Cluster)()
3.1.1.1: createCluster(Cluster)()
3.1.1.1.1: sendMail(adminMail:Email)()
3.2: receiveMail()

Figure 4.5

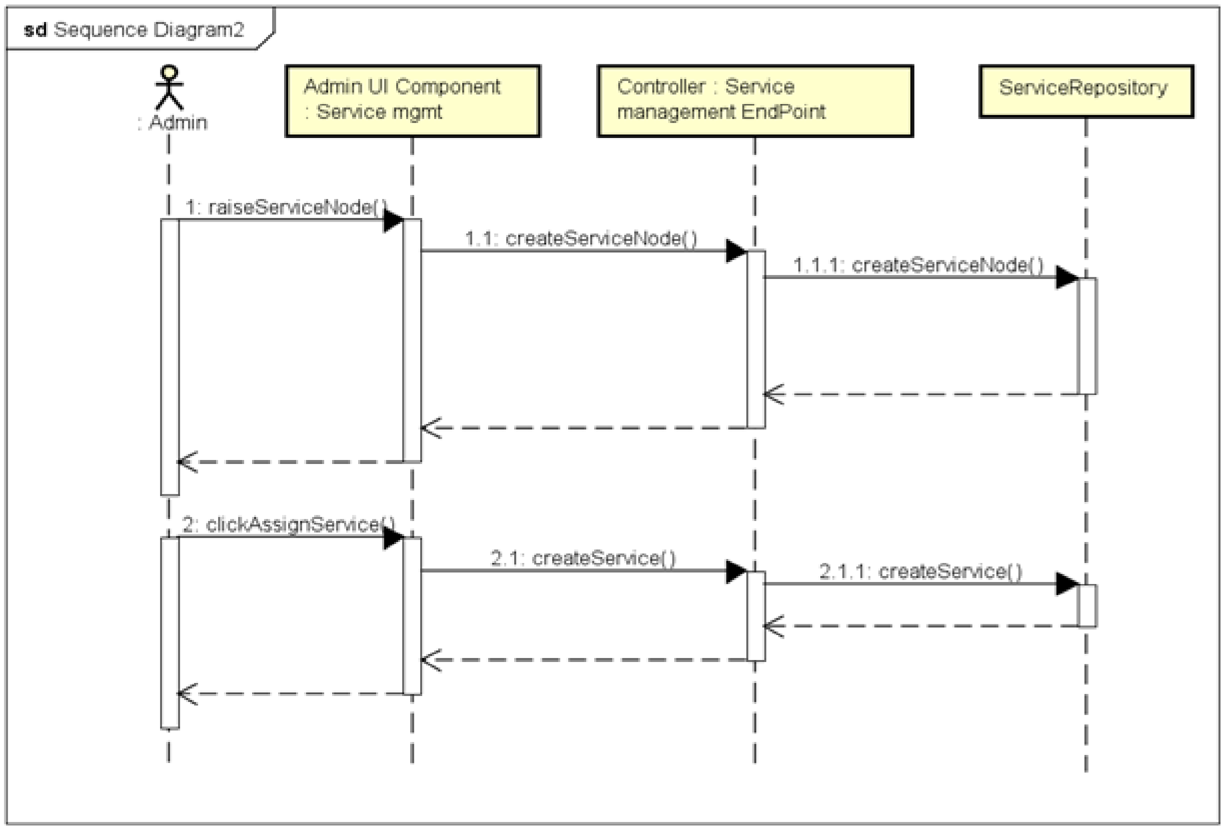


Figure 4.6

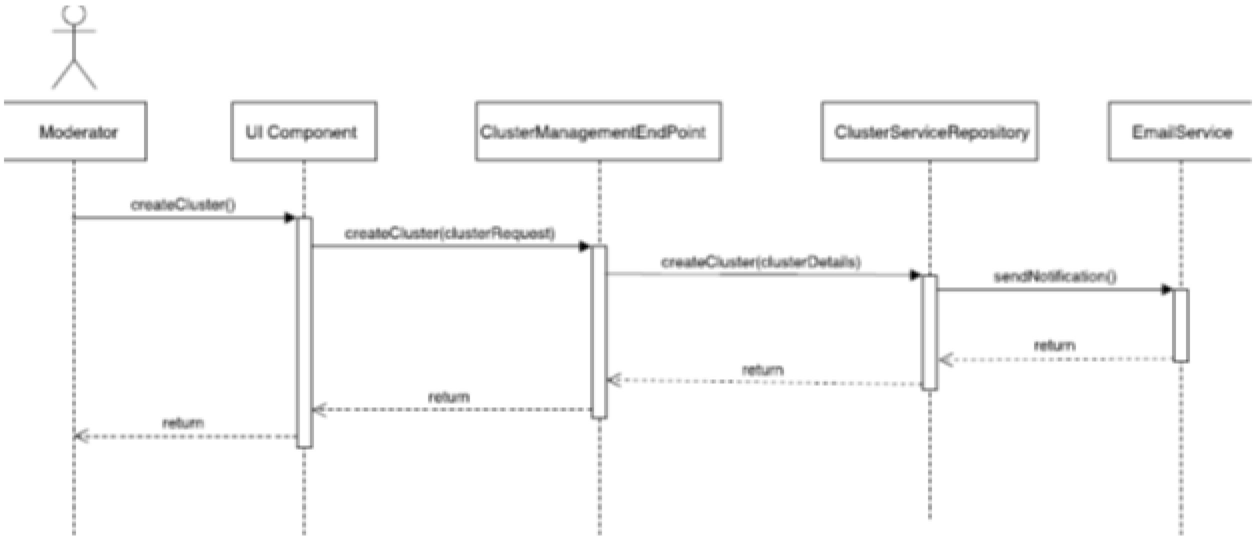


Figure 4.7

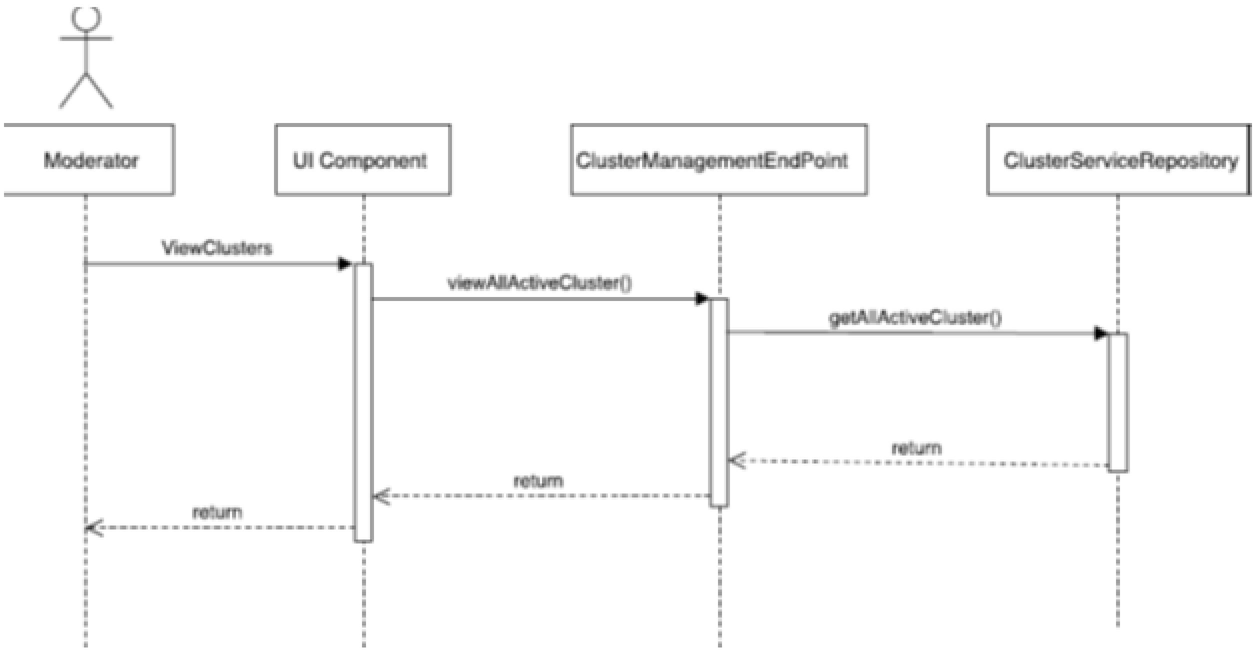


Figure 4.8

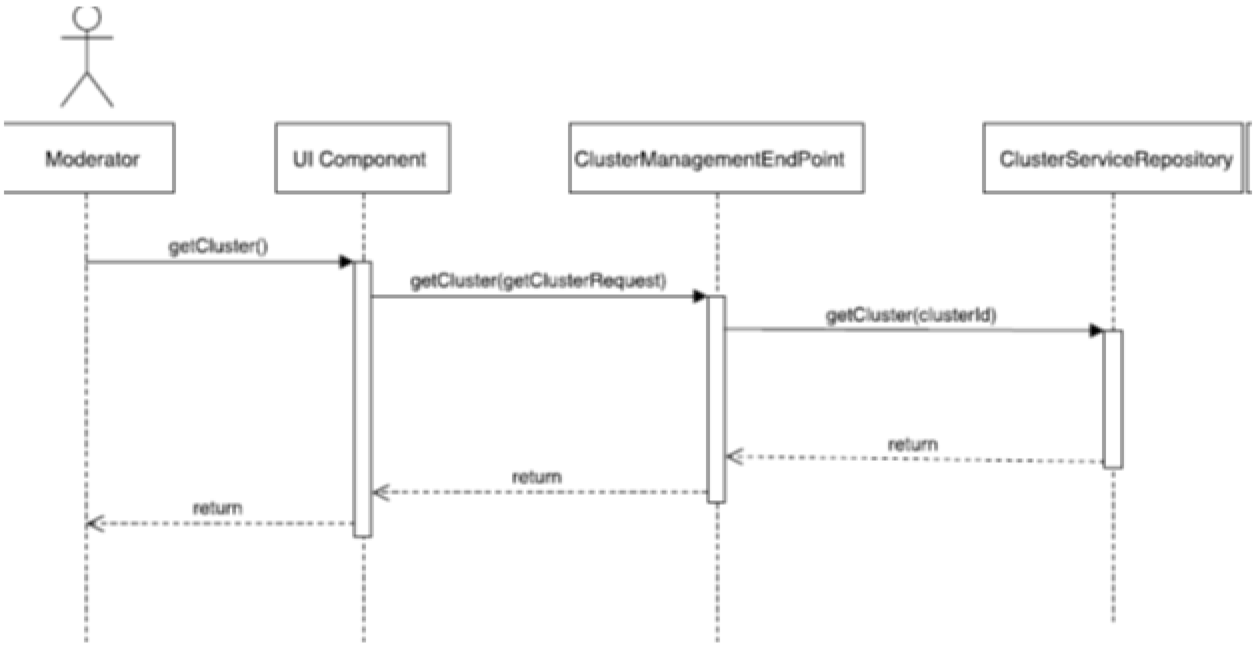


Figure 4.9

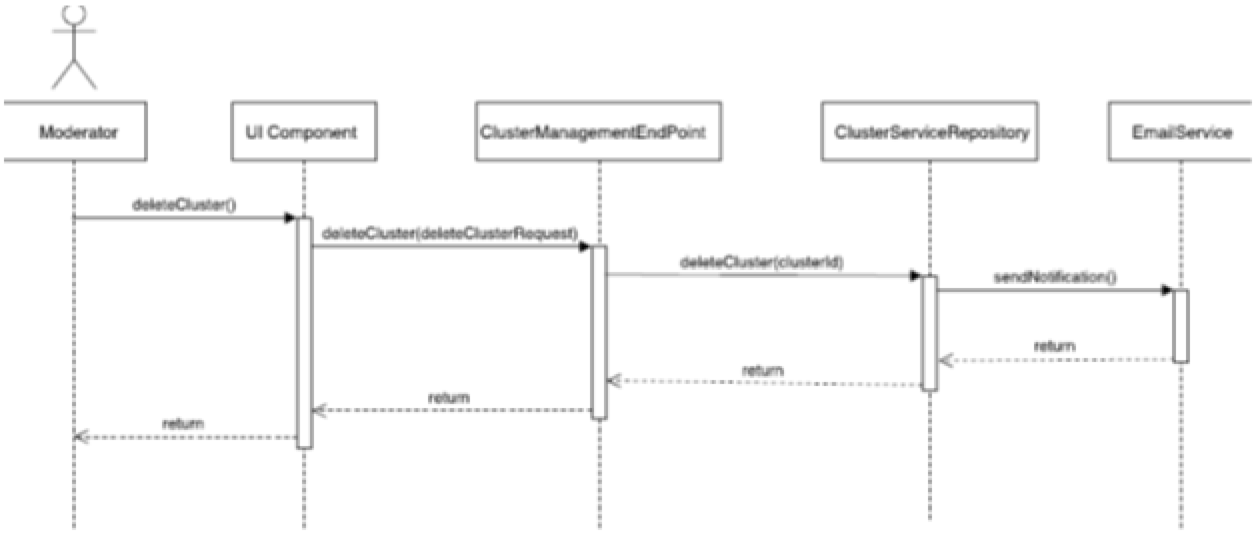


Figure 4.10

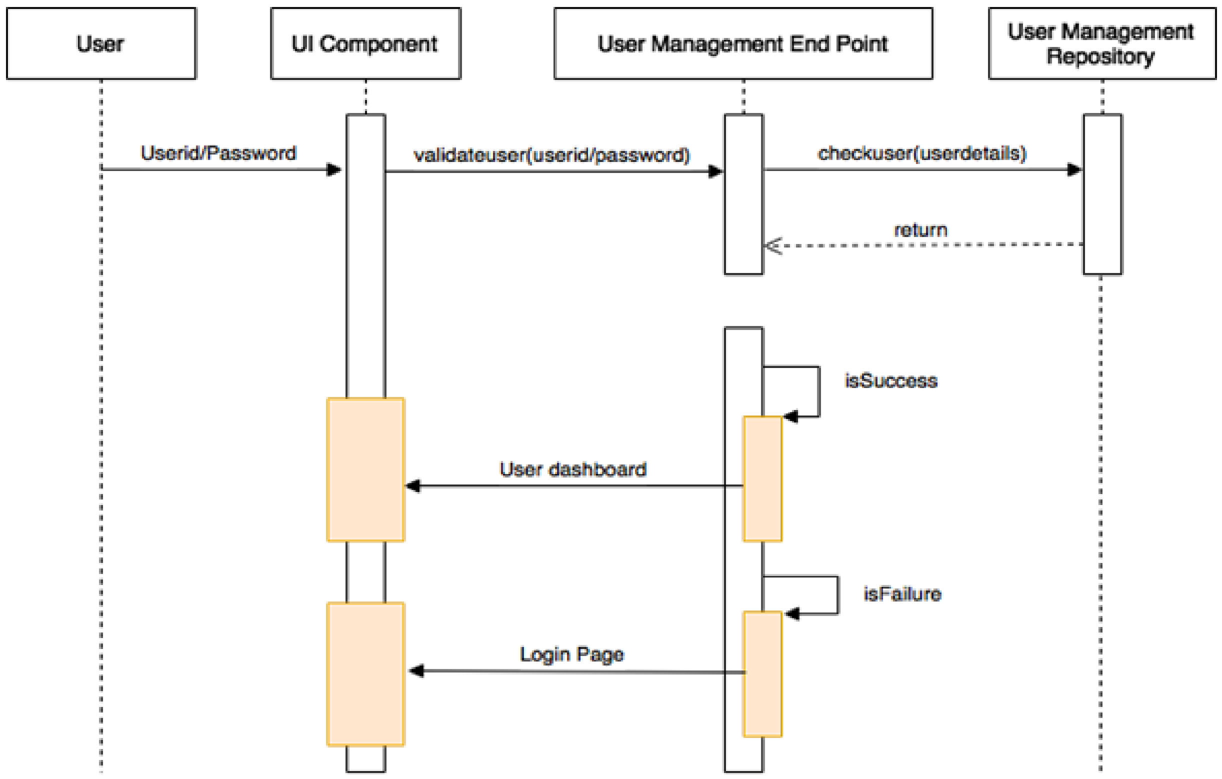


Figure 4.11

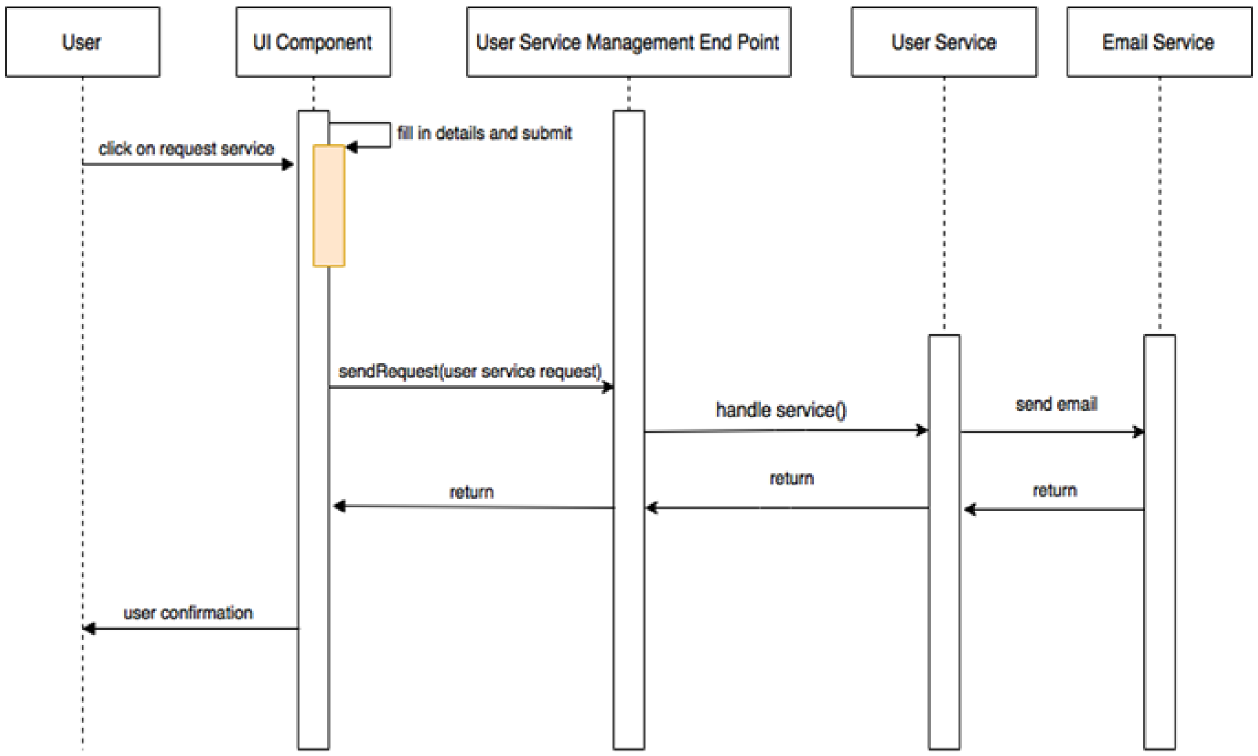


Figure 4.12

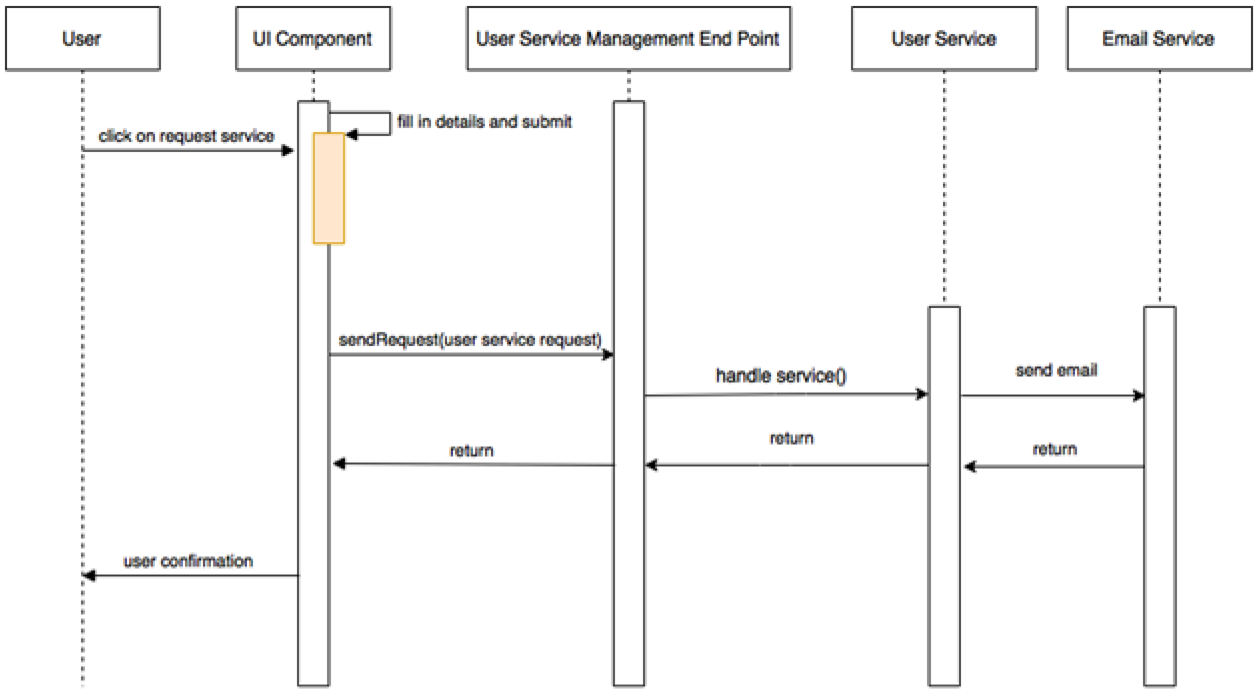

User
UI Component
User Service Management End Point
User Service
Email Service
fill in details and submit
click on request service
sendRequest(user service request)
handle service()
send email
return
return
return
user confirmation

Figure 4.13

Chapter 5. Technologies Used

5.1. Client Layer

Figure 5.1

Here is the snapshot of technologies that have been used in developing the client later for the Neighborhood application. HTML5 and CSS3 along with Bootstrap is used for developing web pages. JQuery is used for making REST API calls from client application and parsing the request and response of APIs.

5.2. Server Layer

Figure 5.2

Server layer includes RESTful APIs which are consumed by Neighborhood client layer. The services are implemented using Spring Boot framework. Relational database MySQL is used for maintaining Cluster, Services and Users relate data. A NoSQL dB store MongoDB is proposed for maintaining user interactions in the community network. Eclipse is used as IDE for development and MySQL Workbench as console for managing MySQL database. Finally, Postman is used for testing RESTful APIs.

5.3. Infrastructure Layer

Figure 5.3

Infrastructure layer includes deployment environment for both client application and server application. Elastic beanstalk is used for deploying backend system as it provides provision for load balancing and auto scaling based on the traffic on the system. Client application is deployed on an apache server on EC2.

Chapter 6. Project Development

6.1. Scope of The Implementation

Below table describes the list of use cases that are implemented as of now and future scope.

	Use Case	Status
1.	As a Moderator, I should be able to login into system and should be able to approve clusters to be created.	Completed
2.	As a moderator, I should be able to edit my details and update them.	Completed
3.	As a admin, I should be able to login into the system and view dashboard, different options such as Cluster management, User management and Service management.	Completed
4.	As a admin, I should be able to create a new cluster, edit existing cluster, delete already created clusters and view all the clusters. (Configuring Clusters)	Completed
5.	As a admin, I should be able to create service nodes, update service nodes, delete service nodes, create services for cluster, edit them and delete them. (Configuring services for Cluster)	Completed
6.	As a admin, I should be able to create, edit, update User nodes. Also, create, delete, update Users for a cluster. (Configuring users for Cluster)	Completed

7.	As a user, I should be able to login into system and view available services, avail a service, view discussions that are happening in the neighborhood.	Completed
8.	As a User, once a new service is availed, respective department’s person should be notified by email regarding the requested/availed service.	Completed
8.	As a registered user for Neighborhood, I should be able to add comments on existing/reported discussions.	Future Scope (API is ready)
9.	As a registered user, I should be able to send personal messages to other users in the cluster.	Future Scope (API ready)
10.	As a user in one neighborhood, I should be able to send messages to users in different Cluster.	Future Work

Table 6.1

6.2. Client Tier UI Development and Integration

Community social network users web UI is developed using Bootstrap, CSS, HTML5 and client side scripting is done using jQuery. AJAX call requests the RESTful APIs. The social network login page provides an option for each type of user login as in the figure below:

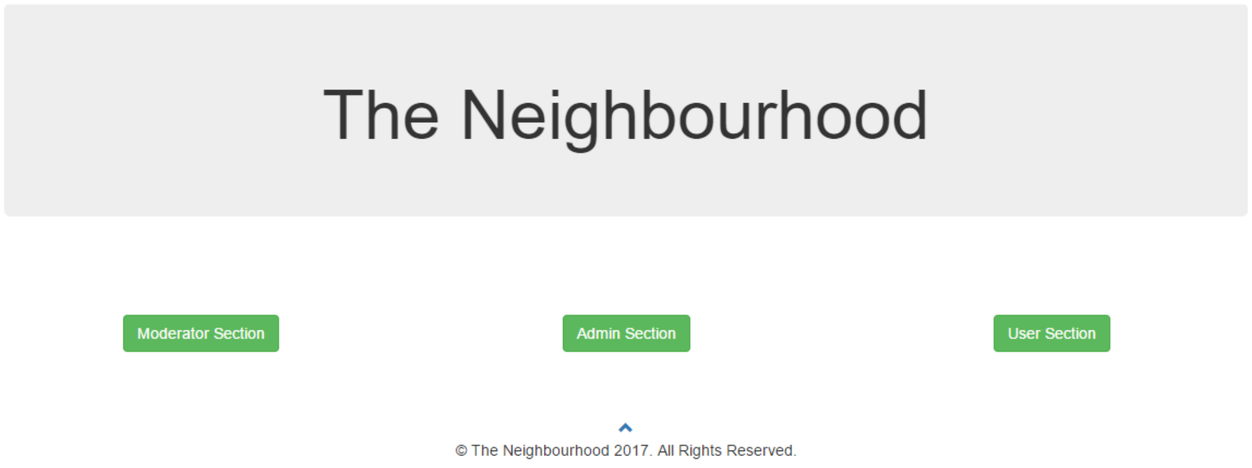


Figure 6.1

Admin UI:

Admin user registers to the SaaS application by providing the required details. Admin account is created upon verification and approval from moderator of the system. Once approved, the admin receives an email notification. The admin can then login to his neighborhood community dashboard.

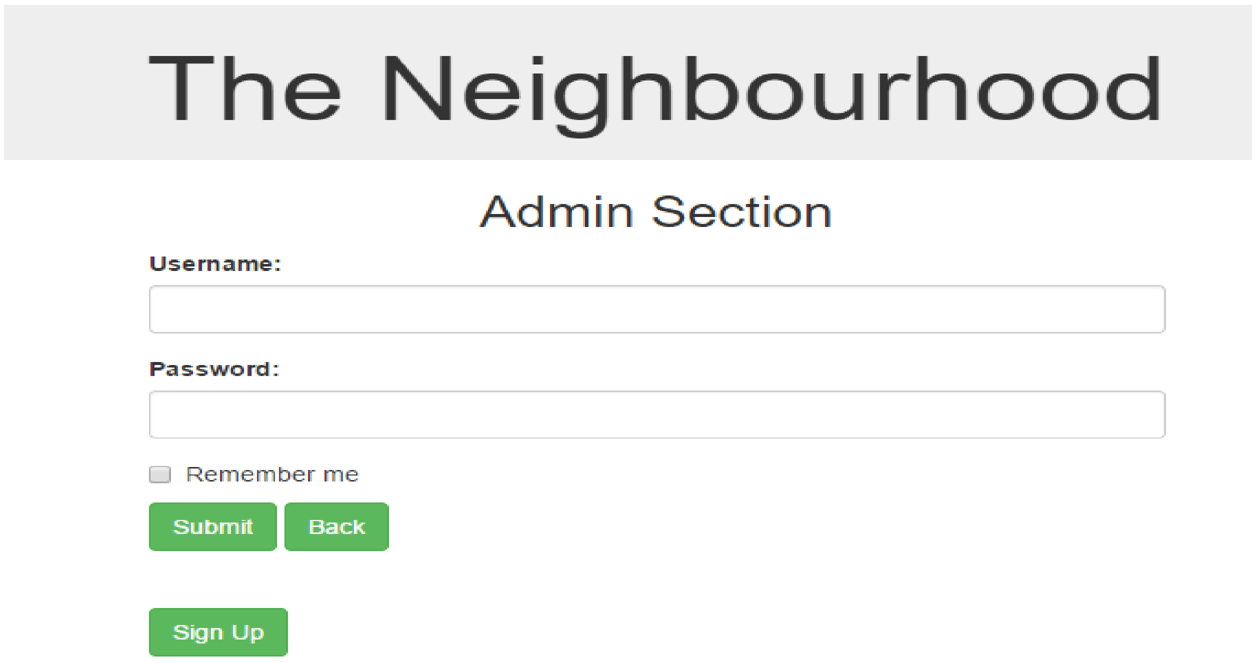
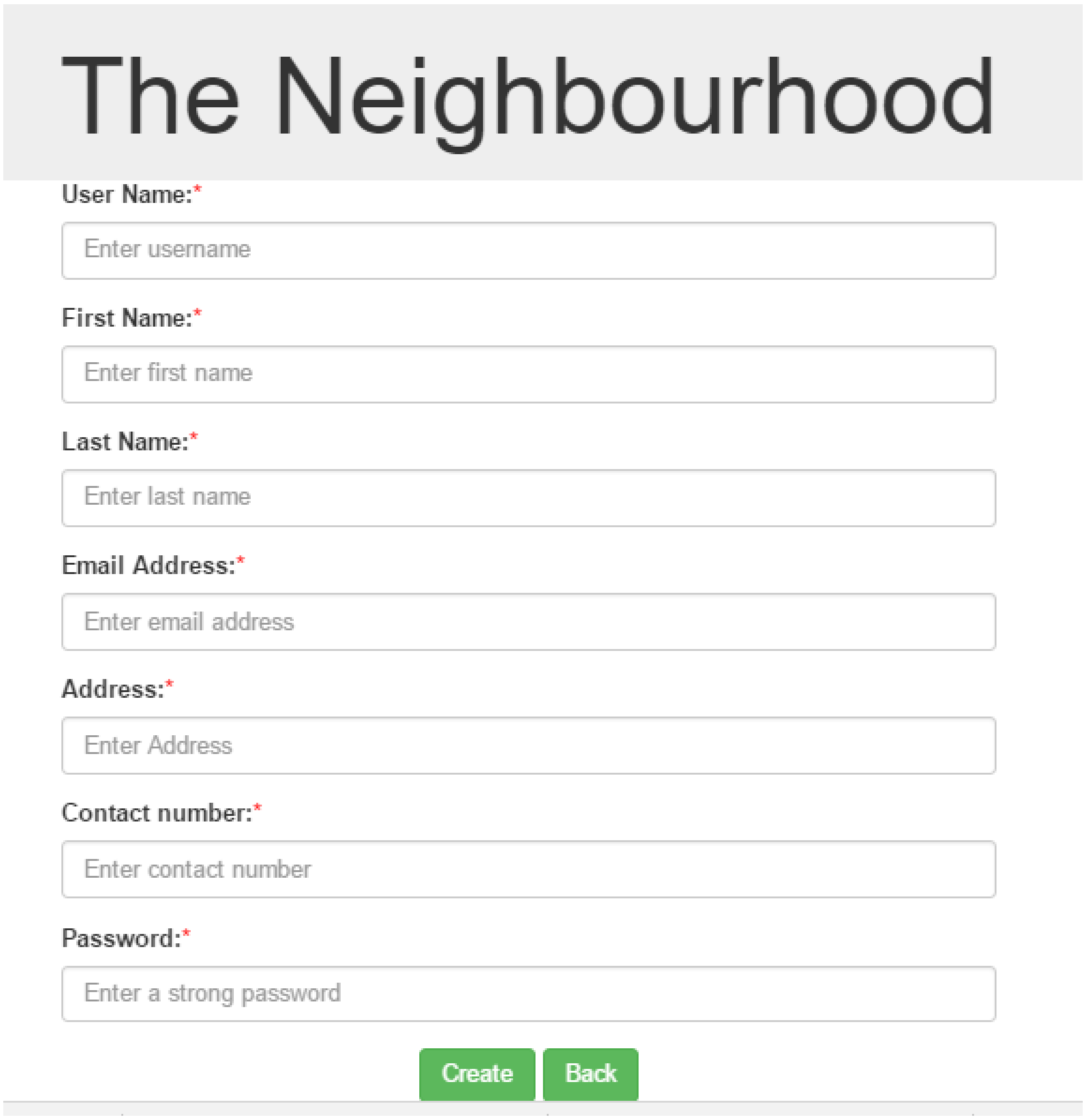


Figure 6.2 Admin Login Page



The Neighbourhood

User Name:*

Enter username

First Name:*

Enter first name

Last Name:*

Enter last name

Email Address:*

Enter email address

Address:*

Enter Address

Contact number:*

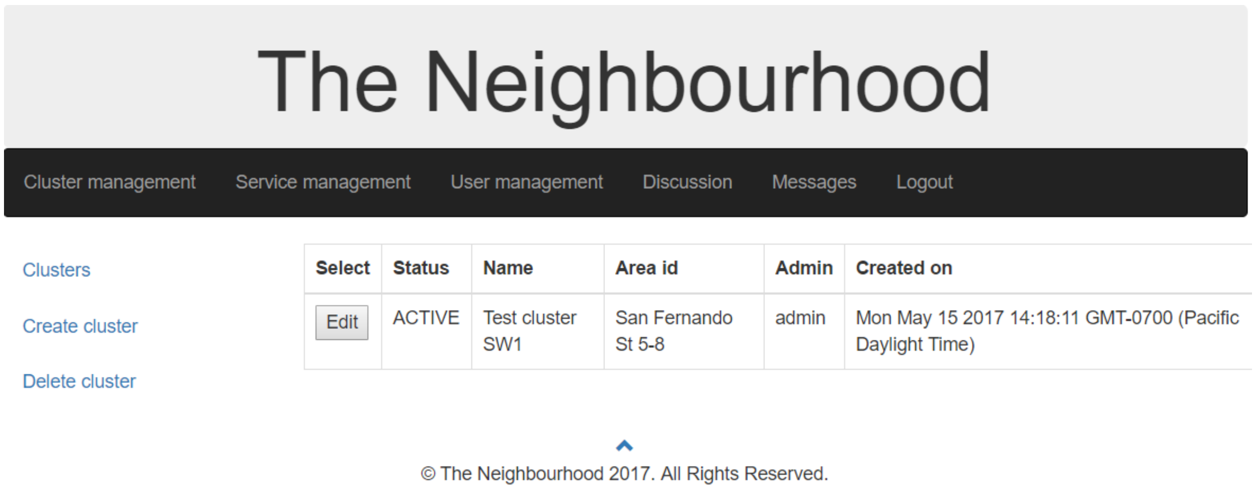
Enter contact number

Password:*

Enter a strong password

Figure 6.3 Create User

Admin dashboard is provided with three sections cluster management, service management and user management respectively. In addition, the admin also has a section to monitor the community discussions.

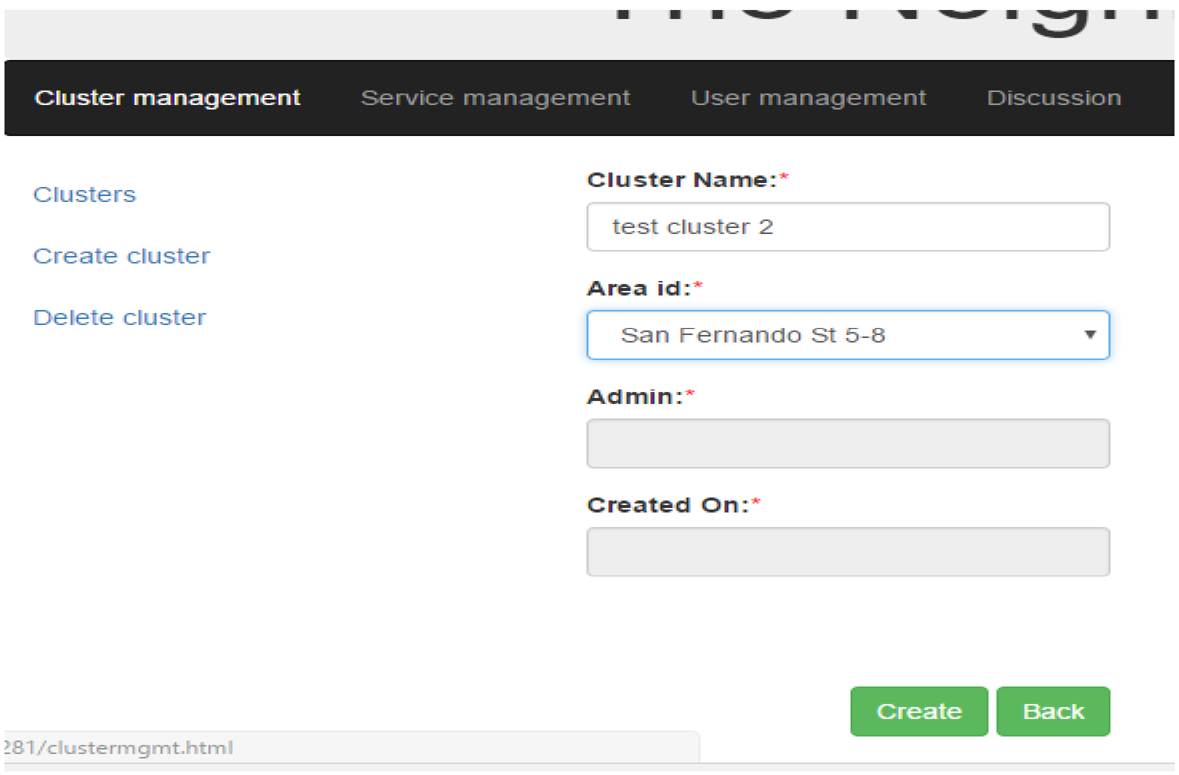


Select	Status	Name	Area id	Admin	Created on
Edit	ACTIVE	Test cluster SW1	San Fernando St 5-8	admin	Mon May 15 2017 14:18:11 GMT-0700 (Pacific Daylight Time)

© The Neighbourhood 2017. All Rights Reserved.

Figure 6.4 List of clusters

The admin should be able to raise cluster creation request to moderator by providing required neighborhood details, a cluster created will be in NEW status. The cluster becomes active once the moderator creates the requested cluster. An email notification is sent to the admin of the cluster on cluster creation.



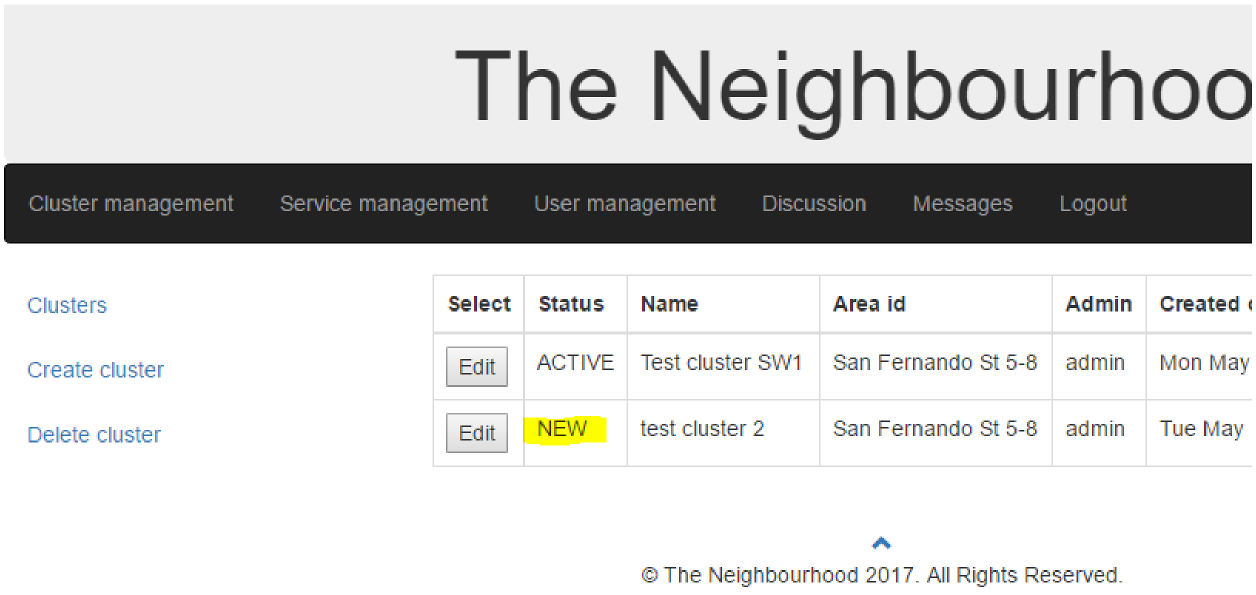
Cluster Name:* test cluster 2

Area id:* San Fernando St 5-8

Admin:*

Created On:*

Figure 6.5 Create Cluster

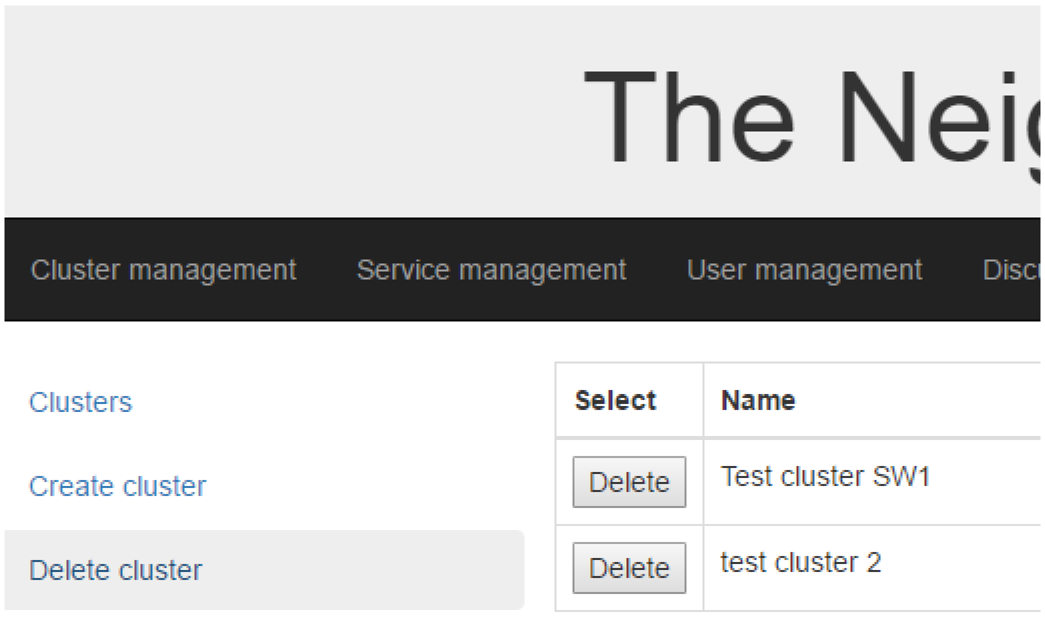

The Neighbourhoo

Select	Status	Name	Area id	Admin	Created
Edit	ACTIVE	Test cluster SW1	San Fernando St 5-8	admin	Mon May
Edit	NEW	test cluster 2	San Fernando St 5-8	admin	Tue May

© The Neighbourhood 2017. All Rights Reserved.

Figure 6.6 List of Clusters

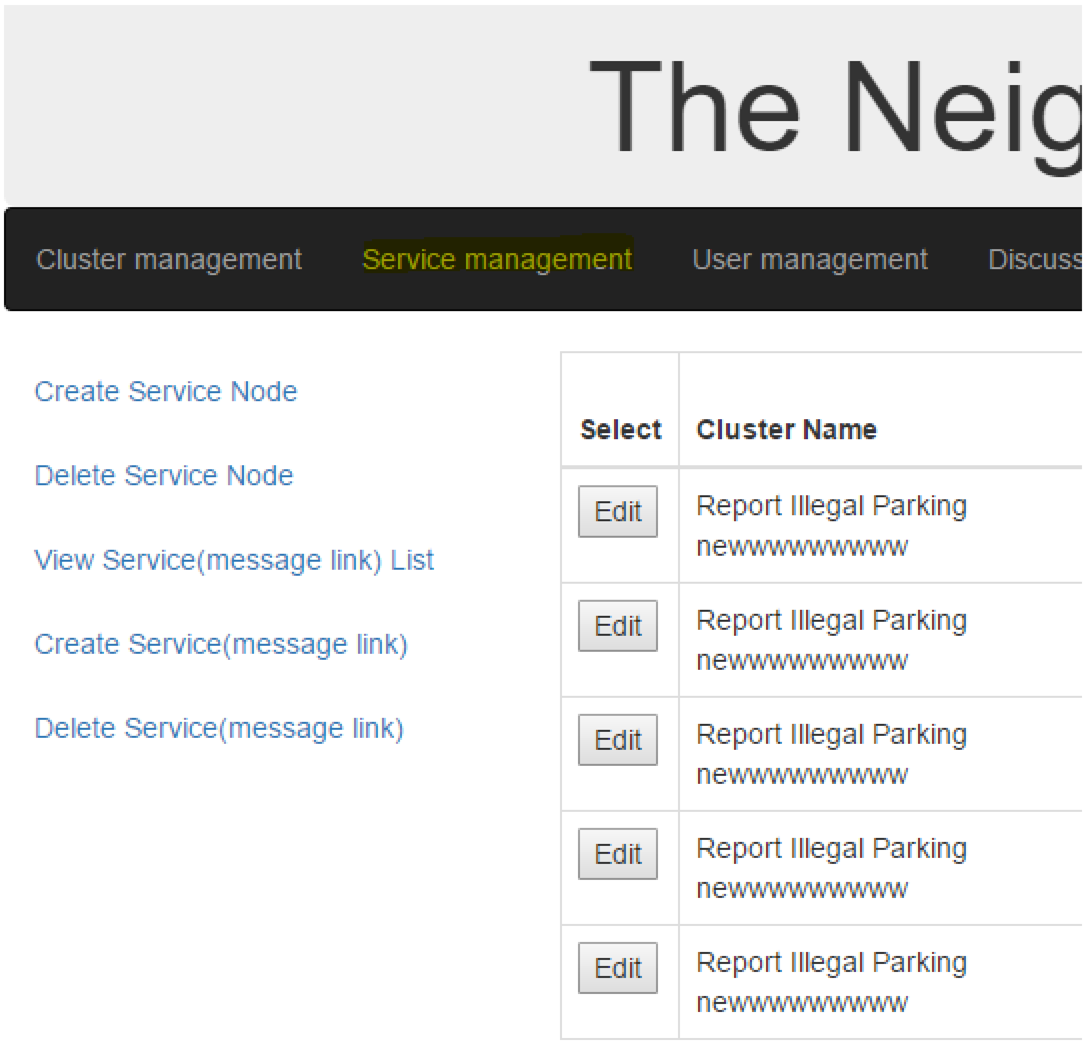
The admin can edit the cluster detail by clicking on the edit button against cluster. Also, the admin has an option to delete the cluster. All the details configured with the cluster are deleted with the cluster deletion.


The Nei

Select	Name
Delete	Test cluster SW1
Delete	test cluster 2

Figure 6.7 Delete a cluster

The Service management section of the admin dashboard provides admin with service node and service configuration options.



Select	Cluster Name
Edit	Report Illegal Parking newwwwwwwwww
Edit	Report Illegal Parking newwwwwwwwww
Edit	Report Illegal Parking newwwwwwwwww
Edit	Report Illegal Parking newwwwwwwwww
Edit	Report Illegal Parking newwwwwwwwww

Figure 6.8 Service List

The services created can be edited and deleted by the cluster admin.

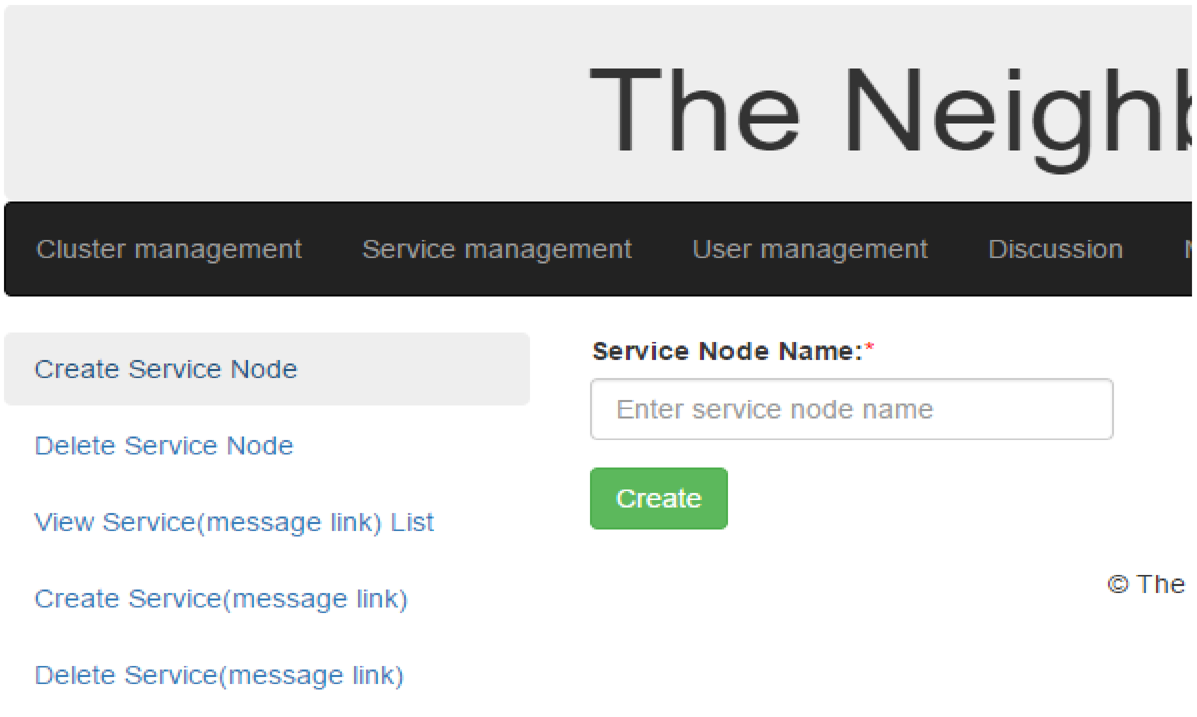
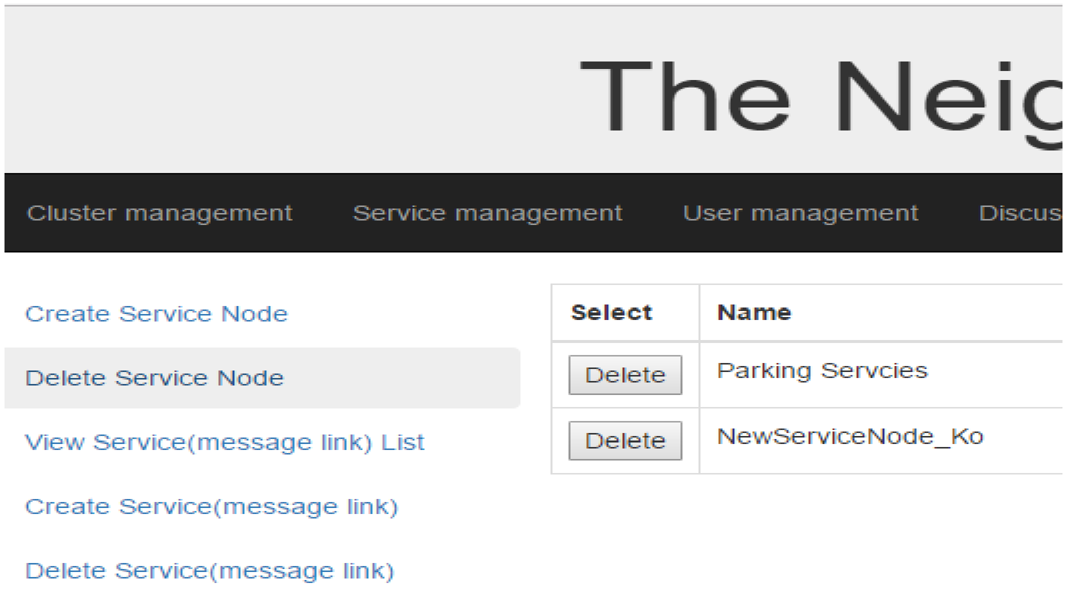
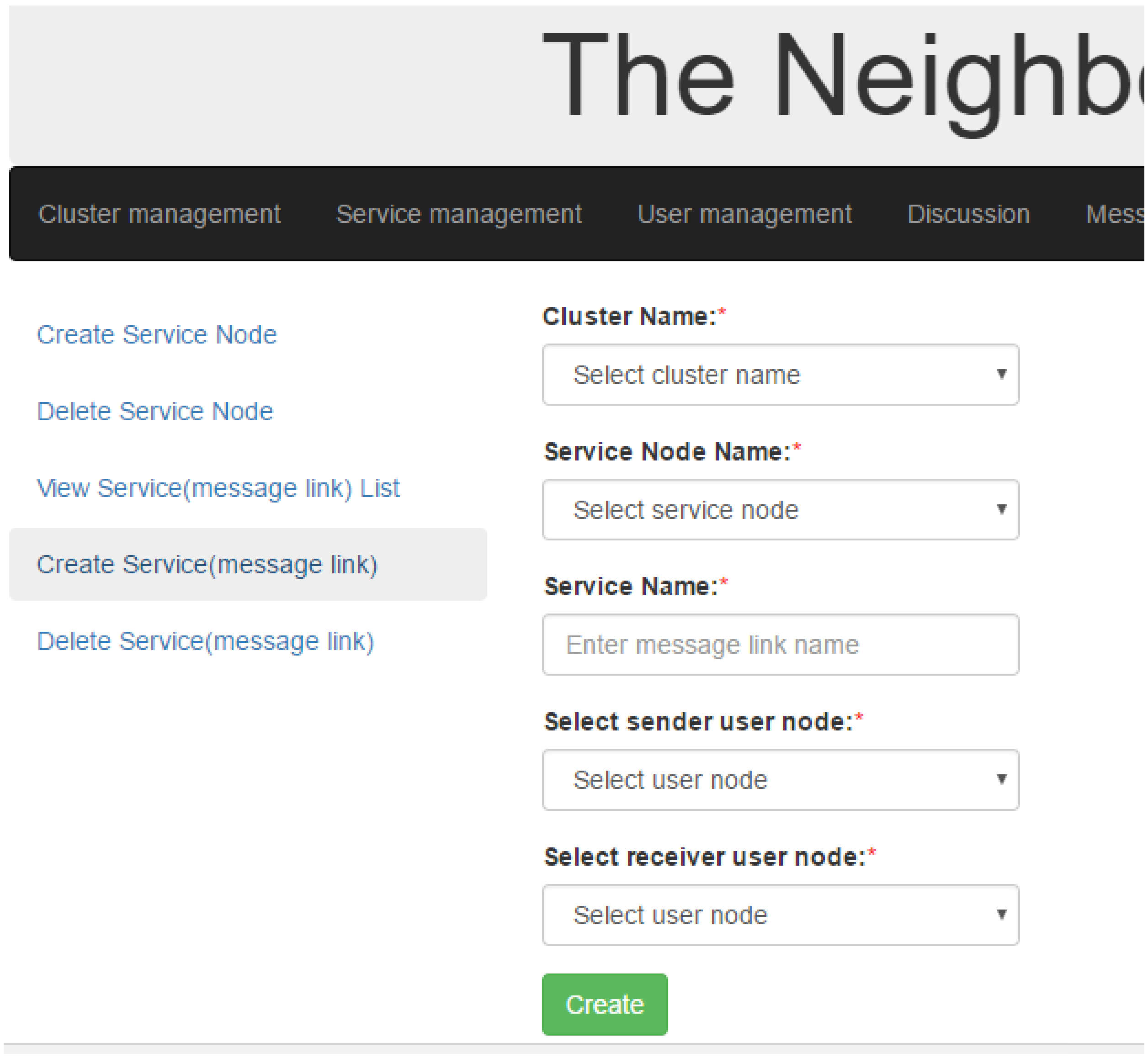


Figure 6.9 Create Service Node



Select	Name
Delete	Parking Servcies
Delete	NewServiceNode_Ko

Figure 6.10 Delete Service Node



Cluster Name:*
Select cluster name

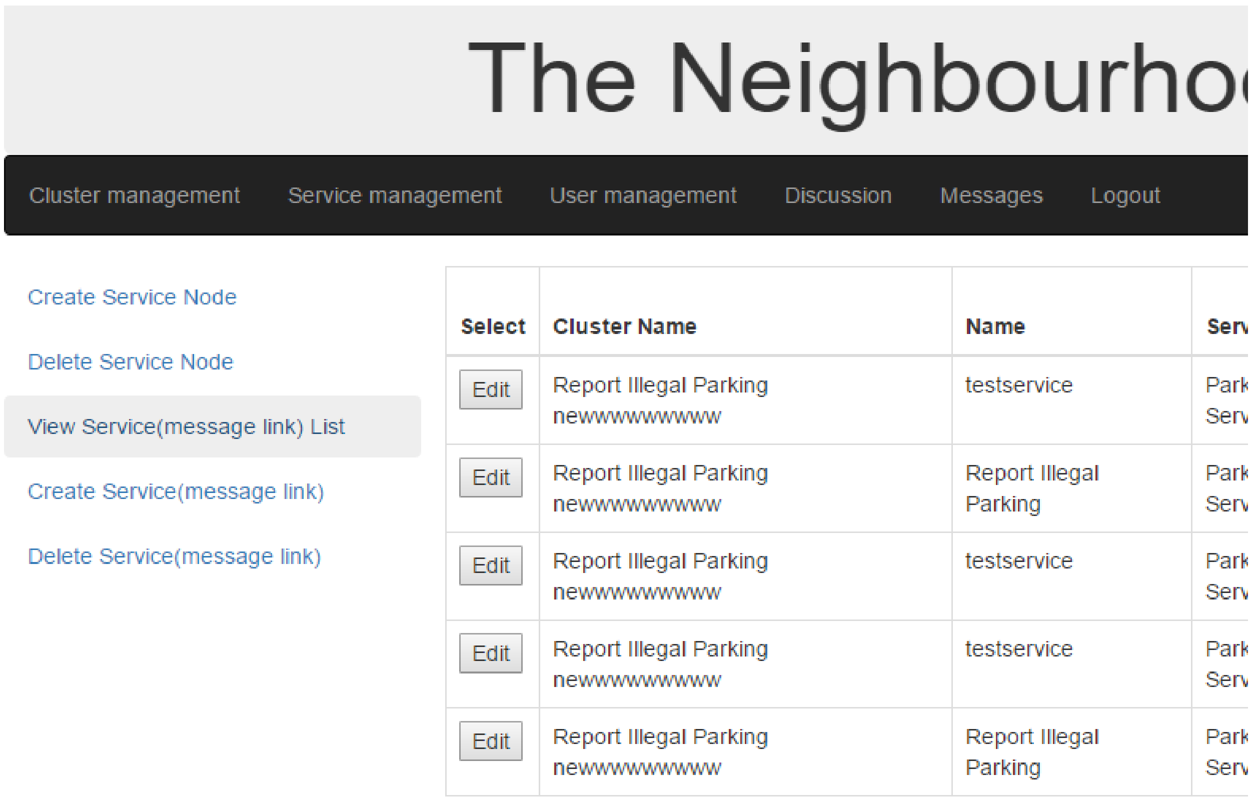
Service Node Name:*
Select service node

Service Name:*
Enter message link name

Select sender user node:*
Select user node

Select receiver user node:*
Select user node

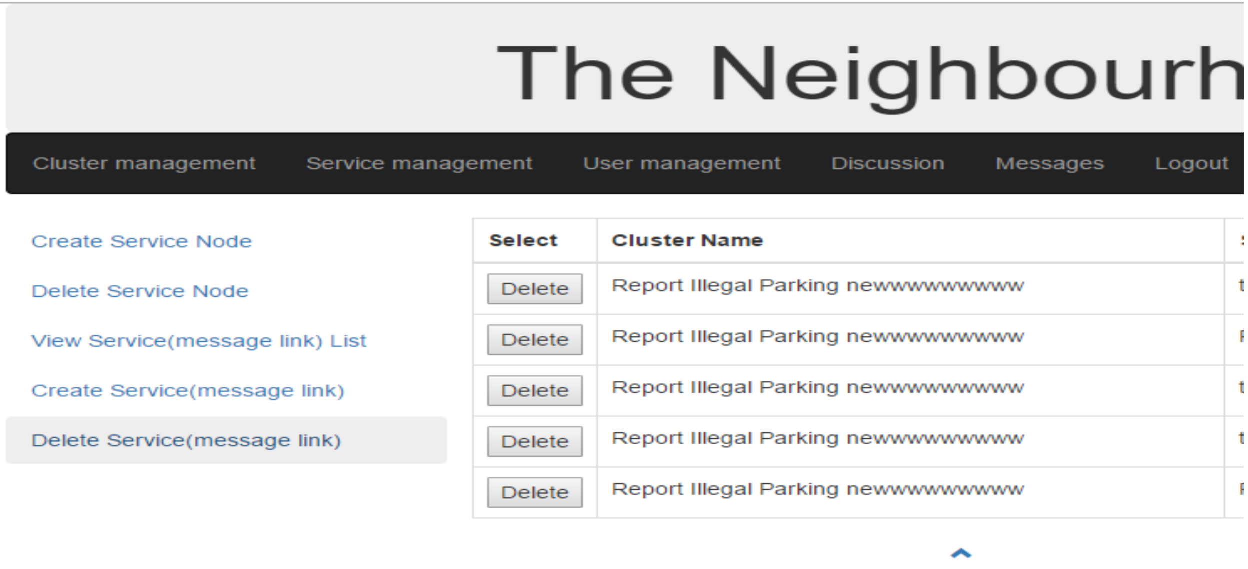
Figure 6.11 Create Service Link



Select	Cluster Name	Name	Serv
Edit	Report Illegal Parking newwwwwwwwww	testservice	Park Serv
Edit	Report Illegal Parking newwwwwwwwww	Report Illegal Parking	Park Serv
Edit	Report Illegal Parking newwwwwwwwww	testservice	Park Serv
Edit	Report Illegal Parking newwwwwwwwww	testservice	Park Serv
Edit	Report Illegal Parking newwwwwwwwww	Report Illegal Parking	Park Serv

Figure 6.12 View Service List

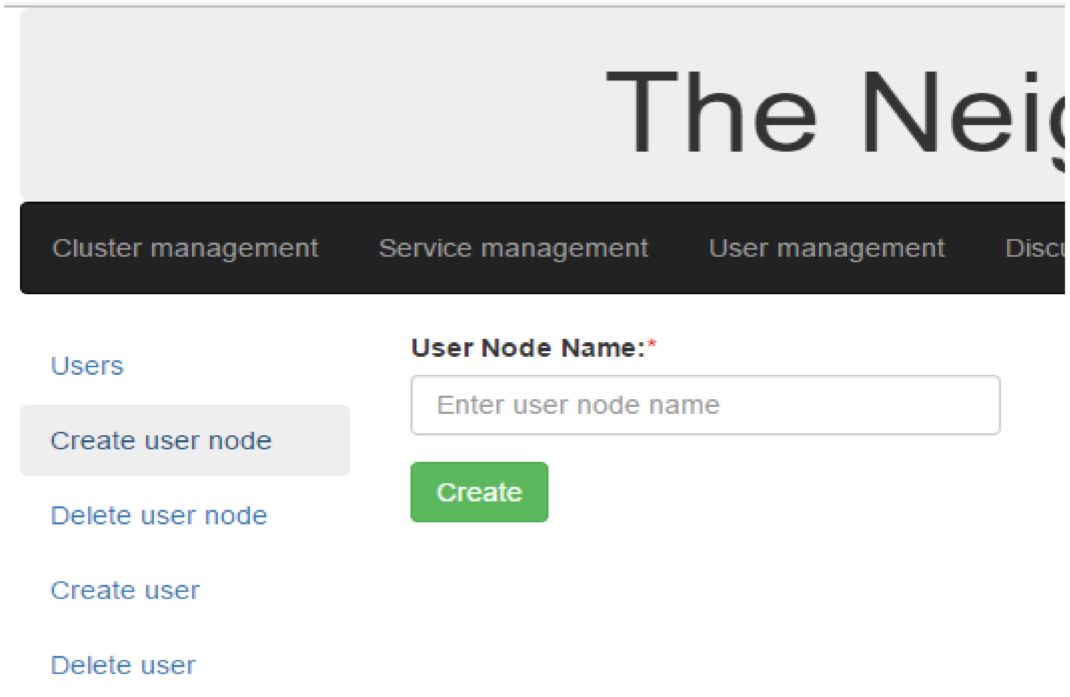
The message links can be edited by the cluster admin. The cluster admin can also delete the message links created between user nodes:



Select	Cluster Name	
Delete	Report Illegal Parking newwwwwwwwww	t
Delete	Report Illegal Parking newwwwwwwwww	F
Delete	Report Illegal Parking newwwwwwwwww	t
Delete	Report Illegal Parking newwwwwwwwww	t
Delete	Report Illegal Parking newwwwwwwwww	F

Figure 6.13 Delete Service

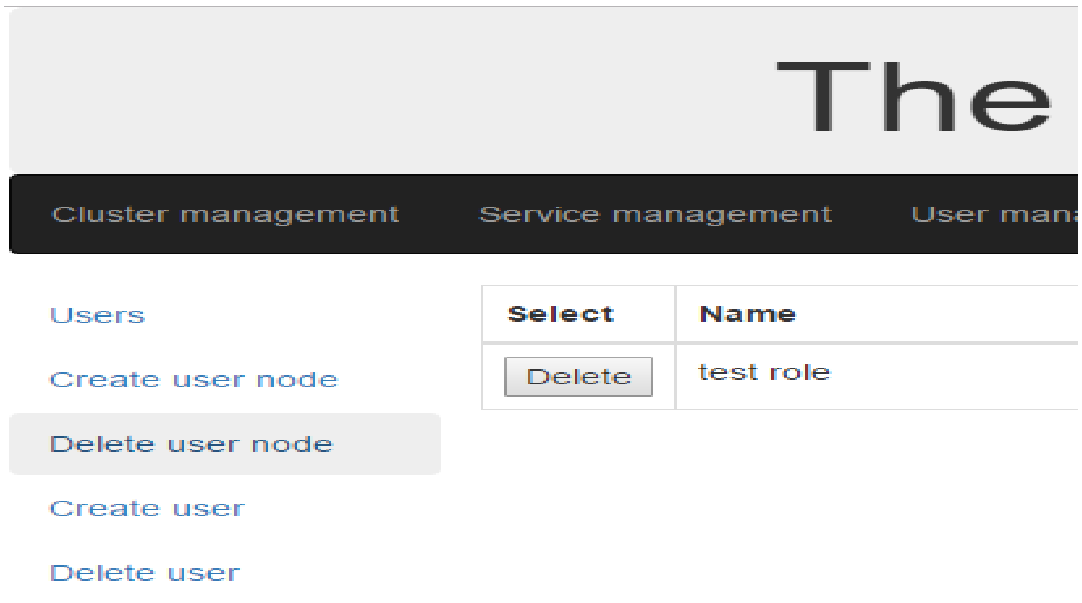
The user management section of the admin dashboard provide admin with options to create user nodes for a cluster, configure user nodes, add users to the user nodes, configure and delete users of a cluster as in the screen shots below:



User Node Name:*

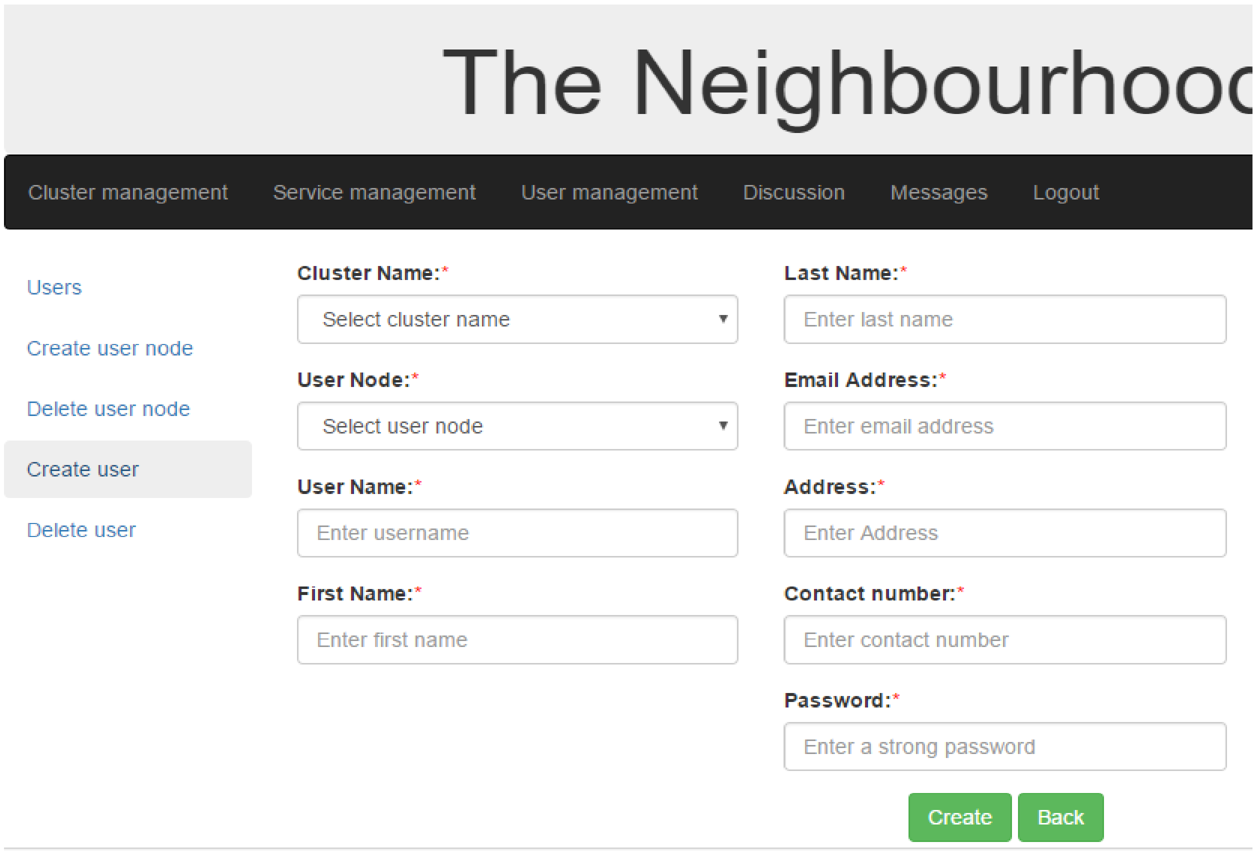
Enter user node name

Figure 6.14 Create User Node



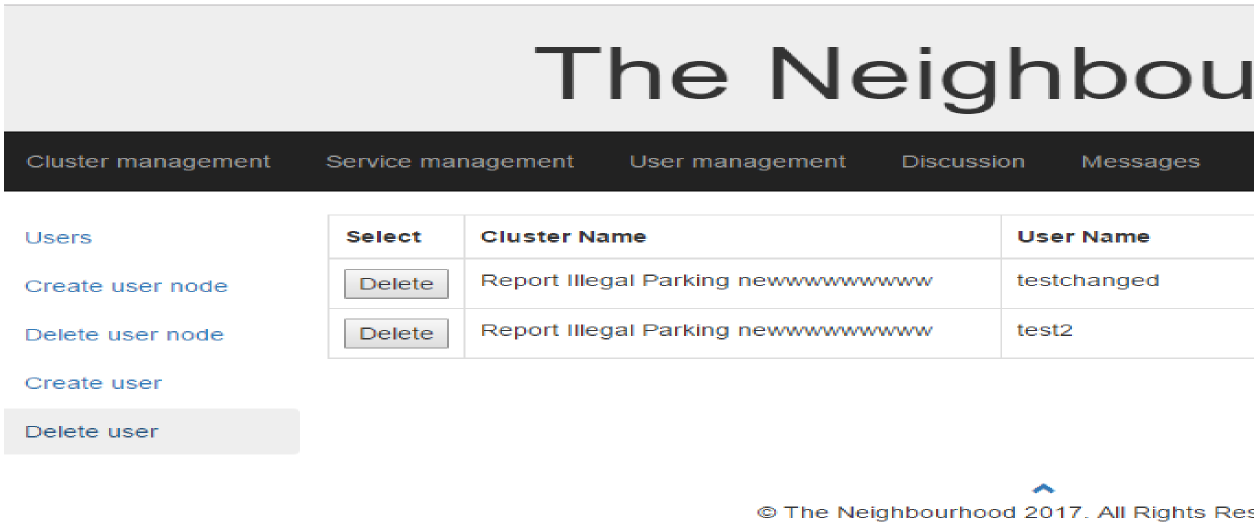
Select	Name
Delete	test role

Figure 6.15 Delete User Node



Cluster Name:*	Select cluster name
User Node:*	Select user node
User Name:*	Enter username
First Name:*	Enter first name
Last Name:*	Enter last name
Email Address:*	Enter email address
Address:*	Enter Address
Contact number:*	Enter contact number
Password:*	Enter a strong password

Figure 6.16 Create User



Select	Cluster Name	User Name
Delete	Report Illegal Parking newwwwwwwwww	testchanged
Delete	Report Illegal Parking newwwwwwwwww	test2

© The Neighbourhood 2017. All Rights Res

Figure 6.17 Delete User

Moderator UI:

The moderator of community social network software platform is responsible for creating the clusters and approving admin users. The moderator dashboard consists of cluster management and user management section. Users in moderator dashboard are the administrators of community cluster.

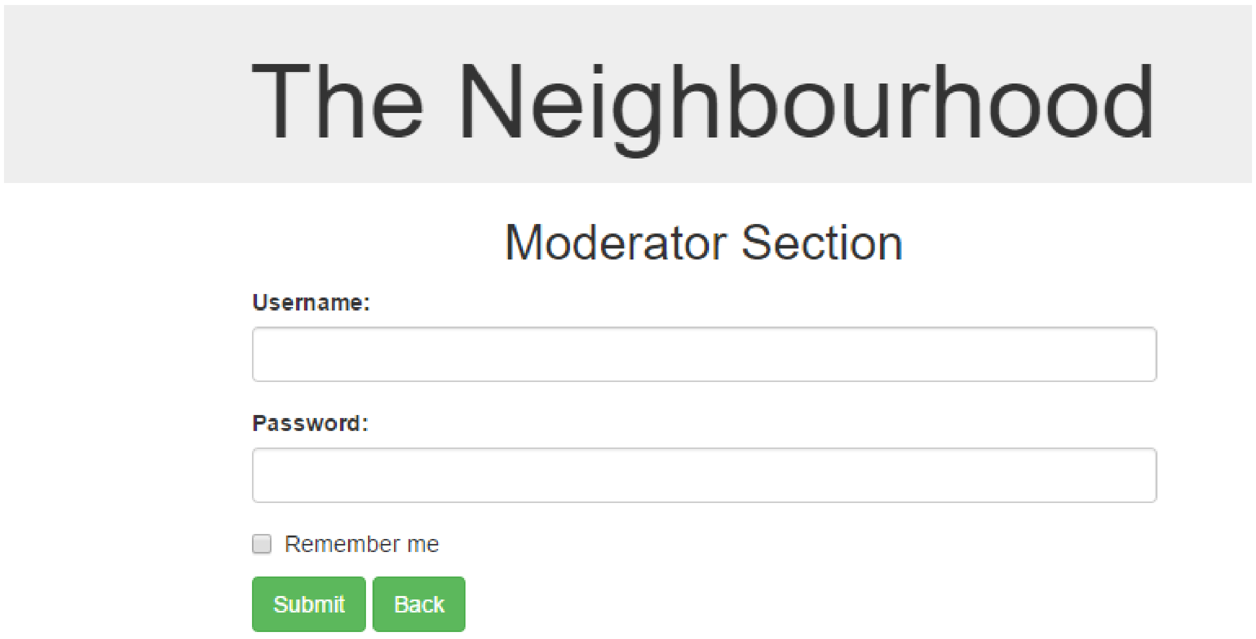
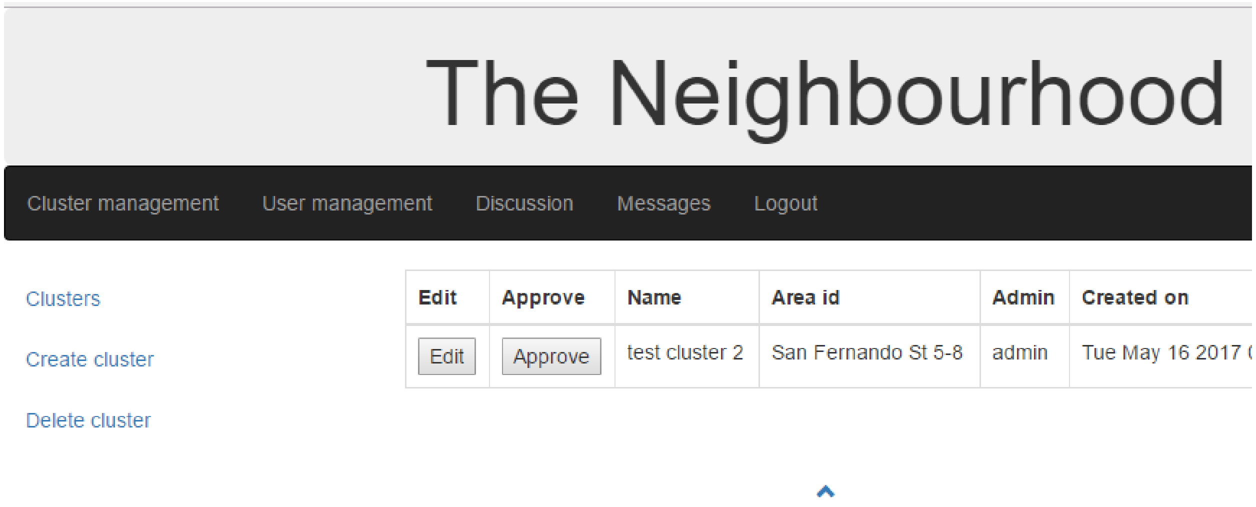


Figure 6.18 Moderator Login

The moderator can either directly approve the cluster to be created or can also edit the cluster details before creating the cluster.

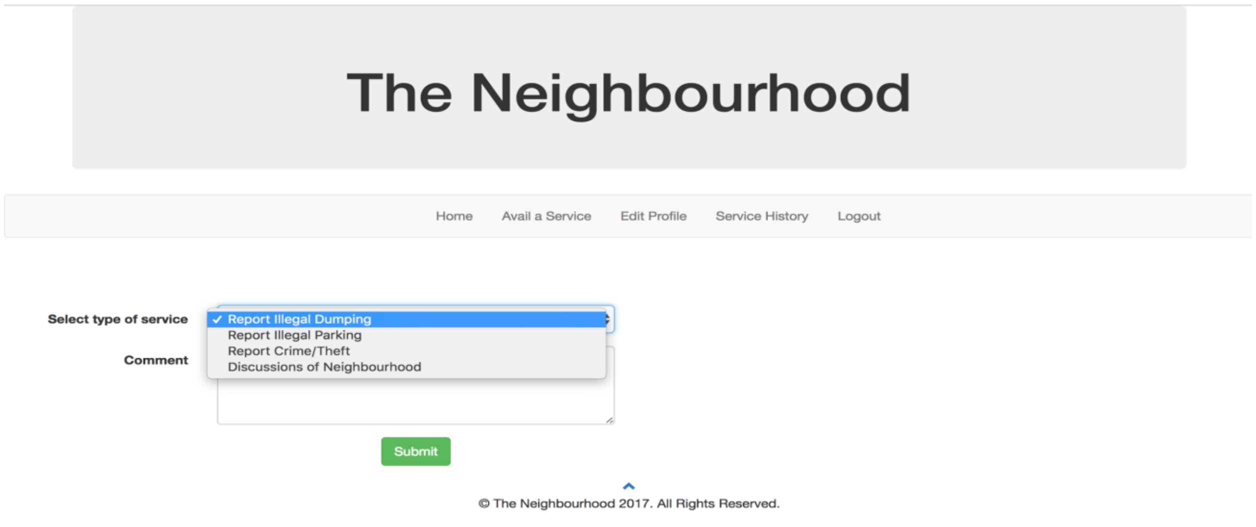


Edit	Approve	Name	Area id	Admin	Created on
Edit	Approve	test cluster 2	San Fernando St 5-8	admin	Tue May 16 2017

Figure 6.19 List of Clusters

User UI:

The user of community social network software platform has the functionality of availing various types of services provided in the community. Also, the user can edit his profile and view already availed services. Users of the community share common discussions where they talk about various services and issues.

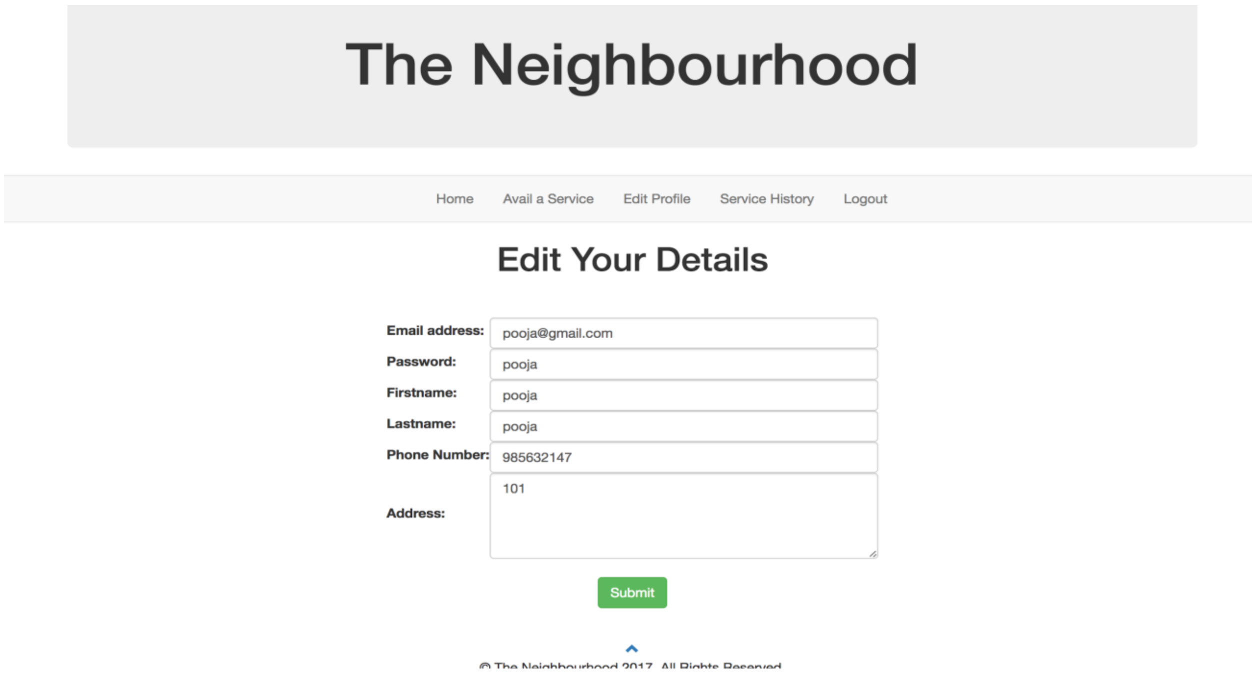

The Neighbourhood

Select type of service: ✓ Report Illegal Dumping
Report Illegal Parking
Report Crime/Theft
Discussions of Neighbourhood

Comment

© The Neighbourhood 2017. All Rights Reserved.

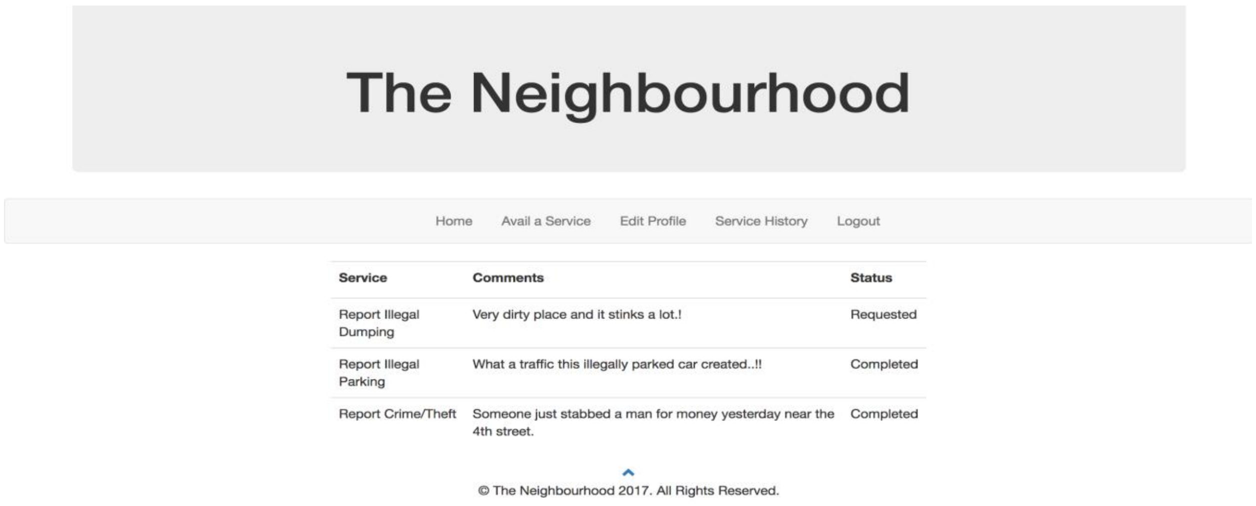
Figure 6.20 Avail a service


The Neighbourhood

Edit Your Details

Email address:	pooja@gmail.com
Password:	pooja
Firstname:	pooja
Lastname:	pooja
Phone Number:	985632147
Address:	101

Figure 6.21 Edit user details



Service	Comments	Status
Report Illegal Dumping	Very dirty place and it stinks a lot.!	Requested
Report Illegal Parking	What a traffic this illegally parked car created..!!	Completed
Report Crime/Theft	Someone just stabbed a man for money yesterday near the 4th street.	Completed

© The Neighbourhood 2017. All Rights Reserved.

Figure 6.22 View availed services

6.3. Server Tier API Development

Spring Boot application is developed to handle the community requests. Tables in this section provide the detail of end points created to access API from web UI.

Action	Sign up
HTTP Method	POST
API	{base URL}/user

Request	{
	“name”:”Alex”,
	“username”:”alex”,
	“password”:”pwd”,
	“address”:”101 san Fernando”,
	“contactNumber”:”10232455”
	}

Response success	{
	"code": "202",
	"msg": "User requested for signup",
	“user”: {
	“name”:”Alex”,
	“username”:”alex”,
	“password”:”pwd”,
	“address”:”101 san Fernando”,
	“contactNumber”:”10232455”
	}
	}

Response failure	{"code": "403", "msg": "User already exists"}

Table 6.2: Spring Boot application API call for signup

Action	Request for Cluster Creation
HTTP Method	POST
API	{base URL}/cluster

Request	{"name": "101-apt",
	    "address": "101 San Fernando",
	    "facebook": "sbhu",
	    "twitter": "sjs",
	    "buildingNo": "1245",
	    "aptNumber": "444",
	    "admin":{"id":1},
	    "area":{"id":1}}

Response success	{
	 "code": "201",
	 "msg": "Successfully created Cluster with id : 4",
	 "cluster": {
	   "id": 4,
	   "name": "101-apt",

```
   "address": "101 San Fernando",
   "facebook": "sbhu",
   "twitter": "sjs",
   "buildingNo": "1245",
   "aptNumber": "444",
   "dateCreated": 1494023031921,
   "admin": {
    "id": 1,
    "firstName": "suraj",
    "lastName": null,
    "username": null,
    "password": null,
    "address": null,
    "email": null,
    "contactNumber": null,
    "dateCreated": 0,
    "role": {
     "id": 1,
     "name": "admin"
    }
   },
   "area": {
    "id": 1,
    "name": "San Jose",
    "city": {
     "id": 1,
     "name": "San Jose",
     "state": null,
     "pincode": null
    },
    "state": null,
    "pincode": null
   },
   "statusValue": "NEW"
  }
 }
```

Response failure	{"code": "403", "msg": "Cluster location error"}

Table 6.3: Spring Boot application API call for cluster creation

Action	Add service to cluster
HTTP Method	POST
API	{base URL}/clusterservice

Request	{
	"cluster": {"id":1},
	"service": {"id":2},
	"contactPerson": {"id":3}
	}

Response success	{
	"code": "201",
	"msg": "Successfully added service to Cluster with id : 4"
	}

Response failure	{"code": "403", "msg": "Service node already exists"}

Table 6.4: Spring Boot application API call for creating service in a cluster

Action	Get service list for a Cluster
HTTP Method	GET
API	{base URL}/clusterservice/{cluster_id}

Request	{
	 "cluster": {"id":1}}

Response success

```
{
 "code": "202",
 "msg": "Fetched services for a cluster",
 "clusterServicesList": [
   {
     "id": 1,
     "cluster": {
       "id": 1,
       "name": "101-apt",
       "address": null,
       "facebook": null,
       "twitter": null,
       "buildingNo": null,
       "aptNumber": null,
       "dateCreated": 1493795712270,
       "admin": {
         "id": 1,
         "firstName": "suraj",
         "lastName": null,
         "username": null,
         "password": null,
         "address": null,
         "email": null,
         "contactNumber": null,
         "dateCreated": 0,
         "role": {
           "id": 1,
           "name": "admin"
         }
       },
       "area": {
         "id": 1,
         "name": "San Jose",
         "city": {
           "id": 1,
           "name": "San Jose",
           "state": null,
           "pincode": null
         },
         "state": null,
      "pincode": null
    },
    "statusValue": "ACTIVE"
  },
  "service": {
    "id": 3,
    "serviceName": "AC problem",
    "serviceNode": {
      "id": 1,
      "name": "Maintainance"
    },
    "createdBy": {
      "id": 1,
      "firstName": "suraj",
      "lastName": null,
      "username": null,
      "password": null,
      "address": null,
      "email": null,
      "contactNumber": null,
      "dateCreated": 0,
      "role": {
        "id": 1,
        "name": "admin"
      }
    },
    "dateCreated": 1493790784799
  },
  "responsibleUser": {
    "id": 1,
    "firstName": "suraj",
    "lastName": null,
    "username": null,
    "password": null,
    "address": null,
    "email": null,
    "contactNumber": null,
    "dateCreated": 0,
    "role": {
      "id": 1,
          "name": "admin"
        }
      }
    }
  ]
}
```

Response failure	{"code": "403", "msg": "No service found"}

Table 6.5: Spring Boot application API call to get service list for a cluster

Action	User authentication
HTTP Method	POST
API	{base URL}/login

Request	{
	"username”:”user”,
	“password”:”password”
	}

Response success	{
	"code": "201",
	"msg": "User authenticated",
	“status”:true,
	“user”:{
	}
	}

Response failure	{"code": "403", "msg": " User Not authenticated ", “status”:false}

Table 6.6: Spring Boot application API call to authenticate users

Action	User creation
HTTP Method	POST
API	{base URL}/user

Request	{
	"firstName":"park",
	"lastName":"park",
	"username":"park",
	"password":"park",
	"role":{"id":5},
	"createdBy":{"id":5},
	"address":"101 San",
	"email":"surajbhukebag@gmail.com",
	"contactNumber":"123654889"
	"cluster":{"id":12}
	}

Response success	{
	"code": "202",
	"msg": "Successfully created User with id : 9",
	"user": { user object }
	}

Response failure	{"code": "403", "msg": " User creation failed" }

Table 6.7: Spring Boot application API call to create users

Action	User creation
HTTP Method	GET
API	{base URL}/adminusers/admin_id
Request	NA

Response success	{
	"code": "202",
	"msg": "Totally there are 3 users",
	"userList": [
	{ user object }, { user object }]
	}

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.8: Spring Boot application API call to get all users created by Admin

Action	User deletion
HTTP Method	DELETE
API	{base URL}/user/user_id
Request	NA

Response success	{
	"code": "202",
	"msg": "Successfully deleted User with id : 3"
	}

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.9: Spring Boot application API call to delete specific user

Action	User Updation
HTTP Method	PUT
API	{base URL}/user/user_id

Request	{
	“id”:1
	"firstName":"park",
	"lastName":"park",
	"username":"park",
	"password":"park",
	"role":{"id":5},
	"createdBy":{"id":5},
	"address":"101 San",
	"email":"surajbhukebag@gmail.com",
	"contactNumber":"123654889"
	"cluster":{"id":12}
	}

Response success	{
	"code": "202",
	"msg": "Successfully Updated User with id : 1",
	“user” : { user object }
	}

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.10: Spring Boot application API call to update specific user

Action	Create Service Node
HTTP Method	POST
API	{base URL}/servicenode

Request	{
	 "name": "Parking Servcies",
	 "createdBy": {"id":6}
	 }

Response success	{
	 "code": "202",
	 "msg": "Successfully created Service node with id : 9",
	 "serviceNode": {
	 "id": 9,
	 "name": "Parking Servcies",
	 ………..
	 }
	 }

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.11: Spring Boot application API call to create Service Node

Action	Update Service Node
HTTP Method	PUT
API	{base URL}/servicenode

Request	{
	 “id” : 1
	 "name": "Parking Servcies",
	        "createdBy": {"id":6}
	}

Response success	{
	  "code": "202",
	  "msg": "Successfully updated Service node with id : 1",
	  "serviceNode": {
	 "id": 9,
	 "name": "Parking Servcies",
	 ………..
	}
	}

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.12: Spring Boot application API call to update specific service node

Action	Delete Service Node
HTTP Method	DELETE
API	{base URL}/servicenode/id
Request	NA
Response success	{ "code": "202", "msg": "Successfully deleted Service node with id : 1", }
Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.13: Spring Boot application API call to delete specific service node

Action	Create service for Cluster
HTTP Method	POST
API	{base URL}/clusterservice

Request	{"name":"Report Illegal Parking", "cluster":{"id":3}, "sender":{"id":1}, "receipient":{"id":4}, "serviceNode":{"id":1}, "createdBy":{"id":2} }

Response success	{
	  "code": "202",
	  "msg": "Successfully created Service for cluster",
	  "clusterServices": {
	  ……
	  }
	}

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.14: Spring Boot application API call to create service for a cluster

Action	Create service for Cluster
HTTP Method	GET
API	{base URL}/clusterservicebyadmin/id

Request	

Response success	{
	"code": "202",
	"msg": "Fetched services for a cluster",
	"clusterServicesList": [
	{
	"id": 5,
	"cluster": {
	"id": 4,…..
	}] }

Response failure	{"code": "403", "msg": " Something went wrong" }

Table 6.15: Spring Boot application API call to get services created by specific admin

Action	Create service request in Cluster
HTTP Method	POST
API	{base URL}/userdiscussion

Request	{ "clusterServices":{"id":2}, "comments":"There is a car parked infront of Subway on 3rd san fernando street. Please take action", "createdBy":{"id":2} }
Response success	{ "code": "202", "msg": "Successfully created User Discussion with id : 10", "userDiscussions": { "id": 10, "clusterServices": { "id": 2, ……… }
Response Failure	{"code": "403", "msg": " Something went wrong" }

Table 6.16: Spring Boot application API call to create service request

Action	Create service request in Cluster
HTTP Method	POST

API	{base URL}/commentuserdiscussion

Request	{
	"userDiscussion":{"id":1},
	"createdBy":{"id":7},
	"comments":"thankmyou"
	}

Response success	{
	 "code": "202",
	 "msg": "Successfully added Comment for Discussion",
	 "userDiscussionComments": {
	 "id": 2,
	 "userDiscussion": {
	 "id": 1,
	….
	}

Response Failure	{"code": "403", "msg": " Something went wrong" }

Table 6.17: Spring Boot application API call to add comment for a service request in cluster

Action	Get service requests of a specific cluster
HTTP Method	GET

API	{base URL}/userdiscussion/id
Request	NA

Response success	{
	  "code": "202",
	  "msg": "Successfully fetched Discussions for cluster",
	  "userDiscussions": [
	  {
	            "id": 1,
	}
	}

Response Failure	{"code": "403", "msg": " Something went wrong" }

Table 6.18: Spring Boot application API call to get all service requests for a Cluster

Chapter 7. Testing

Test Cases

Test Case #: 01 System: User System Role: System User Description: Avail a service	Test Case Name: Avail a service
Pre-conditions: Login as a system user and gain the authorization.	

Step	Action	Expected Result	Pass/Fail	Comment
1	Go to “user-availservice” page	Enter the page	Pass	
2	Choose type of service	Enter	Pass	
3	Fill the comments	Enter	Pass	
4	Click “Submit”	Submitted successfully	Pass	

Table 7.1 Test Case 1

Test Case #: 02 System: User System Role: System User Description: Edit profile	Test Case Name: Edit Profile
Pre-conditions: Login as a system user and gain the authorization.	

Step	Action	Expected Result	Pass/Fail	Comment
1	Go to “user-editprofile” page	Enter the page	Pass	
2	Edit user details	Enter	Pass	
3	Click “Submit”	Submitted successfully	Pass	

Table 7.2 Test Case 2

Test Case #: 03 System: User System Role: System User Description: View service history	Test Case Name: Service History
Pre-conditions: Login as a system user and gain the authorization.	

Step	Action	Expected Result	Pass/Fail	Comment
1	Go to "user-servicehistory" page	Enter the page	Pass	
2	View availed services	Enter	Pass	

Table 7.3 Test Case 3

Test Case #: 04 System: Admin System Role: Community Admin Description: Sign up to social network platform	Test Case Name: Sign Up
Pre-conditions: N/A.	

Step	Action	Expected Result	Pass/Fail	Comment
1	Click on sign up button on admin section	Enter the page	Pass	

2	Provide details and click on create	Admin create request sent to moderator	Pass	

Table 7.4 Test Case 4

Test Case #: 05 System: Admin System Role: Community Admin Description: Raise a cluster create request to moderator	Test Case Name: Raise a cluster create request
Pre-conditions: Admin has successfully logged in	

Step	Action	Expected Result	Pass/Fail	Comment
1	Click on create cluster button and provide the necessary details	A cluster row is created at cluster manageme nt section with status ‘NEW’	Pass	

Table 7.5 Test Case 5

Test Case #: 06 System: Admin System Role: Community Admin Description: Edit cluster details	Test Case Name: Edit cluster details

Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on edit cluster button against cluster row and provide the necessary details	A cluster details are updated	Pass	

Table 7.6 Test Case 6

Test Case #: 07 System: Admin System Role: Community Admin Description: Delete cluster		Test Case Name: Delete cluster		
Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on delete cluster button against the cluster row	Selected cluster is deleted	Pass	

Table 7.7 Test Case 7

Test Case #: 08 System: Admin System Role: Community Admin	Test Case Name: Create service node	

Description: create service node				
Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on create service node option in service management section and enter the service node name to be created	Service node is created	Pass	

Table 7.8 Test Case 8

Test Case #: 09 System: Admin System Role: Community Admin Description: Delete service node		Test Case Name: Delete service node		
Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on delete service node button against service node row	Service node is deleted	Pass	

Table 7.9 Test Case 9

Test Case #: 10 System: Admin System Role: Community Admin Description: create user node		Test Case Name: Create user node		
Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on create user node option in user management section and enter the user node name to be created	User node is created	Pass	

Table 7.10 Test Case 10

Test Case #: 11 System: Admin System Role: Community Admin Description: create, edit and delete messaging link		Test Case Name: Messaging link		
Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on create messaging link in service management section of the admin and provide service	Message link is created between user nodes	Pass	

	node name, sender and recipient user node name along with cluster detail			

Table 7.11 Test Case 11

Test Case #: 12 System: Admin System Role: Community Admin Description: delete user node	Test Case Name: Delete user node			
Pre-conditions: Admin has successfully logged in				
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on delete user node option	User node is deleted	Pass	

Table 7.12 Test Case 12

Test Case #: 13 System: Admin System Role: Community Admin Description: Create, edit and delete users of a cluster	Test Case Name: Create and delete user			
Pre-conditions: Admin has logged into his dashboard successfully				
Step	Action	Expected Result	Pass/Fail	Comment

1	Click on Create user, provide cluster, user node and user details to create a user of cluster	User is created and an email notification is sent to the user	Pass	
2	Click on delete user against the user row	User is removed from the cluster	Pass	

Table 7.14 Test Case 14

Test Case #: 14 System: Moderator system Role: System Admin Description: Approve admin cluster create request	Test Case Name: Create cluster

Pre-conditions:
Moderator has successfully logged in to his account

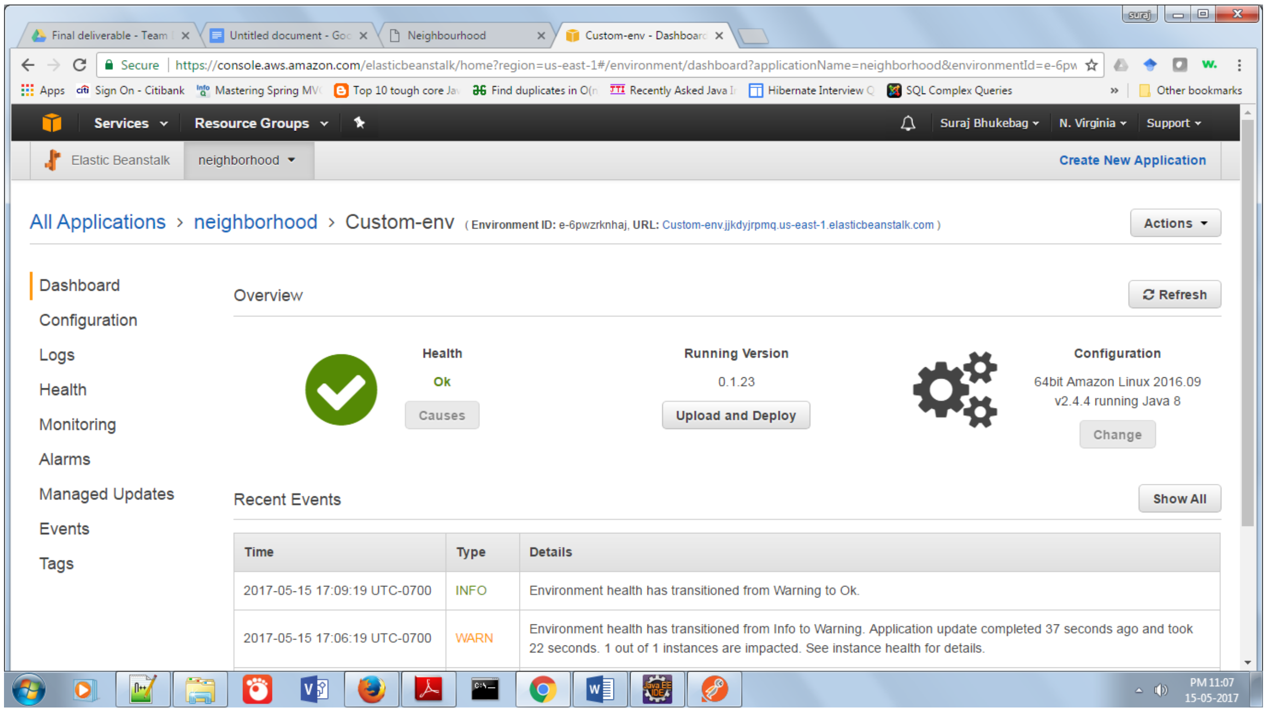
Step	Action	Expected Result	Pass/Fail	Comment
1	Click on approve button against a cluster row to approve the cluster creation request.	The cluster status changes to ACTIVE from new.	Pass	

Table 7.15 Test Case 15

Chapter 8. Deployment

Neighborhood web application and backend system both are deployed on Amazon Web Services(AWS). The Elastic beanstalk service offered by AWS provides configuration management where in auto scaling and load balancing can be configured for applications. We are using elastic beanstalk for backend system which requires load balancing and auto scaling.

Here are the snapshots of deployed application and some configuration parameters.



(Environment ID: e-6pwzrknhaj, URL: Custom-env.jkdyjrpmq.us-east-1.elasticbeanstalk.com)

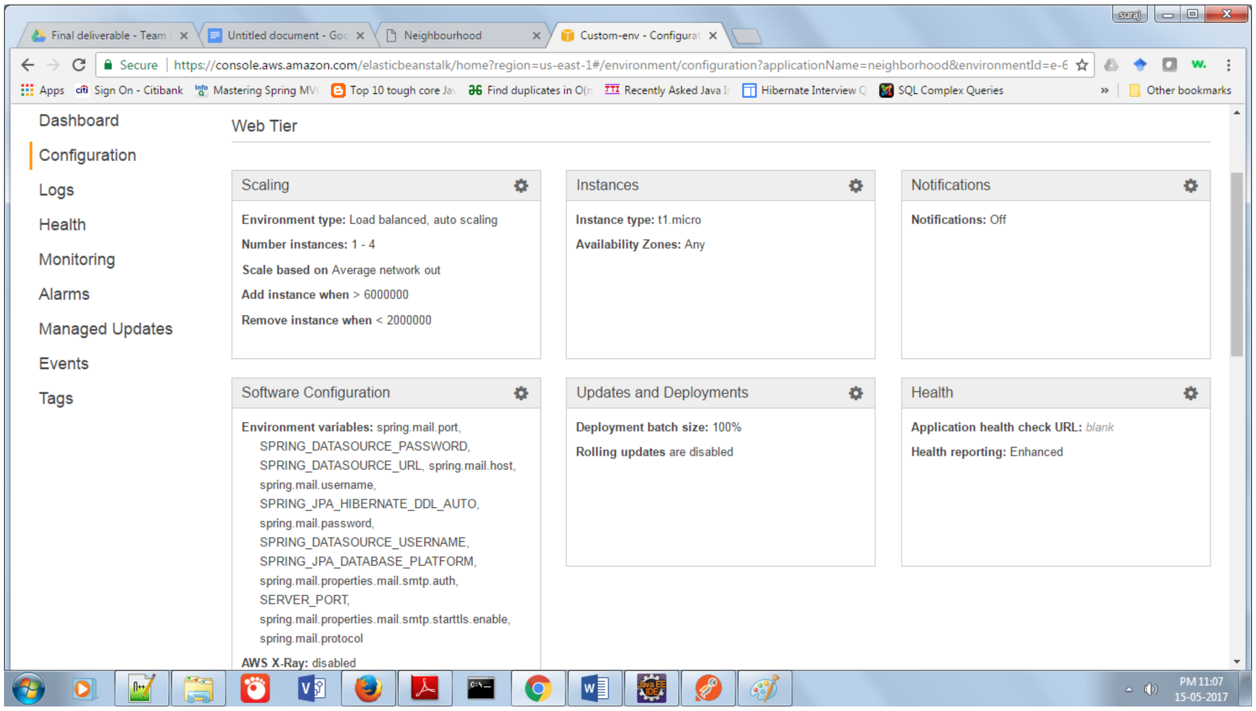
Health: Ok

Running Version: 0.1.23

Configuration: 64bit Amazon Linux 2016.09 v2.4.4 running Java 8

Time	Type	Details
2017-05-15 17:09:19 UTC-0700	INFO	Environment health has transitioned from Warning to Ok.
2017-05-15 17:06:19 UTC-0700	WARN	Environment health has transitioned from Info to Warning. Application update completed 37 seconds ago and took 22 seconds. 1 out of 1 instances are impacted. See instance health for details.

Figure 8.1



Web Tier

Scaling

Environment type: Load balanced, auto scaling
Number instances: 1 - 4
Scale based on Average network out
Add instance when > 6000000
Remove instance when < 2000000

Instances

Instance type: t1.micro
Availability Zones: Any

Notifications

Notifications: Off

Software Configuration

Environment variables: spring.mail.port, SPRING_DATASOURCE_PASSWORD, SPRING_DATASOURCE_URL, spring.mail.host, spring.mail.username, SPRING_JPA_HIBERNATE_DDL_AUTO, spring.mail.password, SPRING_DATASOURCE_USERNAME, SPRING_JPA_DATABASE_PLATFORM, spring.mail.properties.mail.smtp.auth, SERVER_PORT, spring.mail.properties.mail.smtp.starttls.enable, spring.mail.protocol
AWS X-Ray: disabled

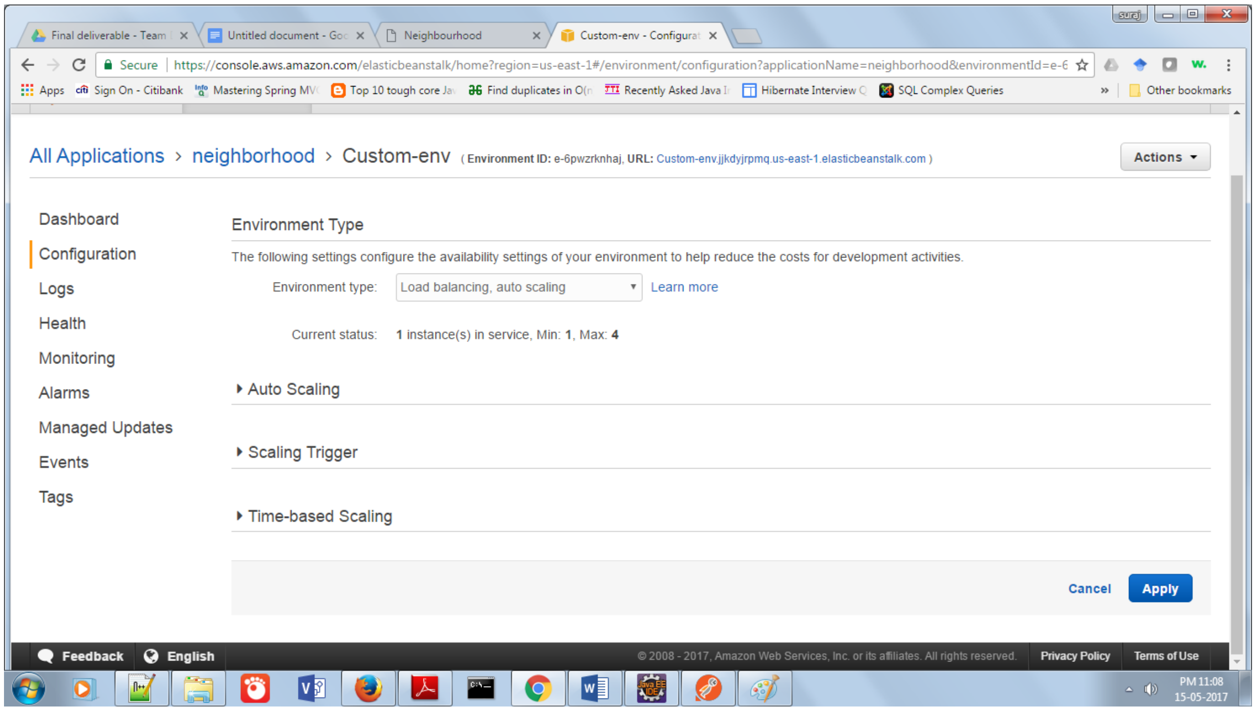
Updates and Deployments

Deployment batch size: 100%
Rolling updates are disabled

Health

Application health check URL: blank
Health reporting: Enhanced

Figure 8.2



All Applications › neighborhood › Custom-env (Environment ID: e-6pwzrknhaj, URL: Custom-env.jjkdyjrpmq.us-east-1.elasticbeanstalk.com)

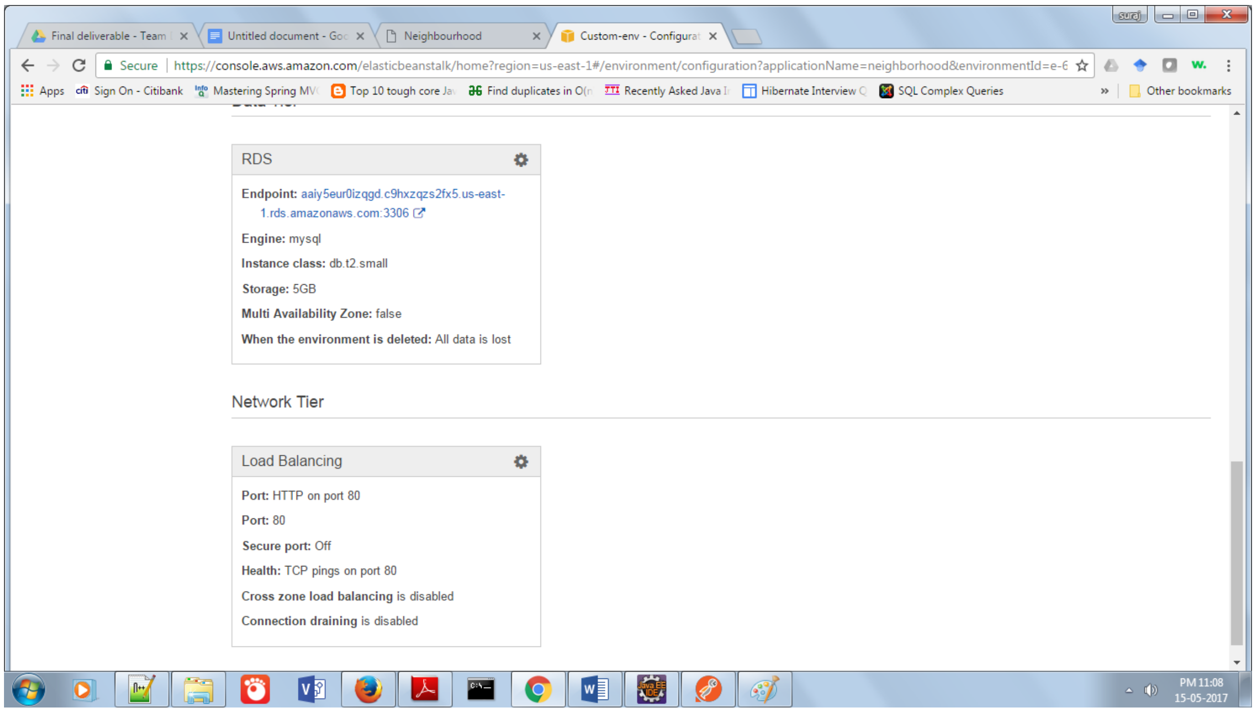
Environment Type

The following settings configure the availability settings of your environment to help reduce the costs for development activities.

Environment type: Load balancing, auto scaling

Current status: **1** instance(s) in service, Min: **1**, Max: **4**

Figure 8.3



RDS

Endpoint: aaiy5eur0izqgd.c9hxzqzs2fx5.us-east-1.rds.amazonaws.com:3306

Engine: mysql

Instance class: db.t2.small

Storage: 5GB

Multi Availability Zone: false

When the environment is deleted: All data is lost

Network Tier

Load Balancing

Port: HTTP on port 80

Port: 80

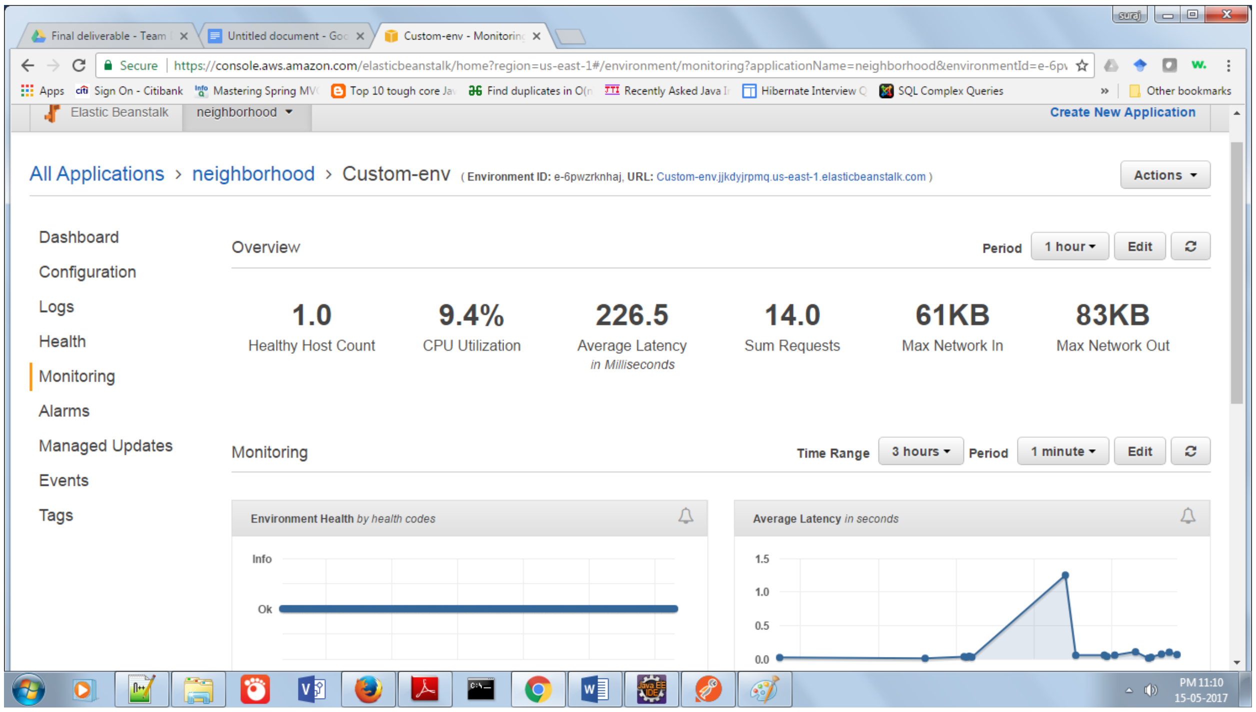
Secure port: Off

Health: TCP pings on port 80

Cross zone load balancing is disabled

Connection draining is disabled

Figure 8.4



All Applications › neighborhood › Custom-env (Environment ID: e-6pwzrknhaj, URL: Custom-env.jjkdyjrpmq.us-east-1.elasticbeanstalk.com)

Overview — Period: 1 hour

1.0	9.4%	226.5	14.0	61KB	83KB
Healthy Host Count	CPU Utilization	Average Latency *in Milliseconds*	Sum Requests	Max Network In	Max Network Out

Monitoring — Time Range: 3 hours, Period: 1 minute

Figure 8.5

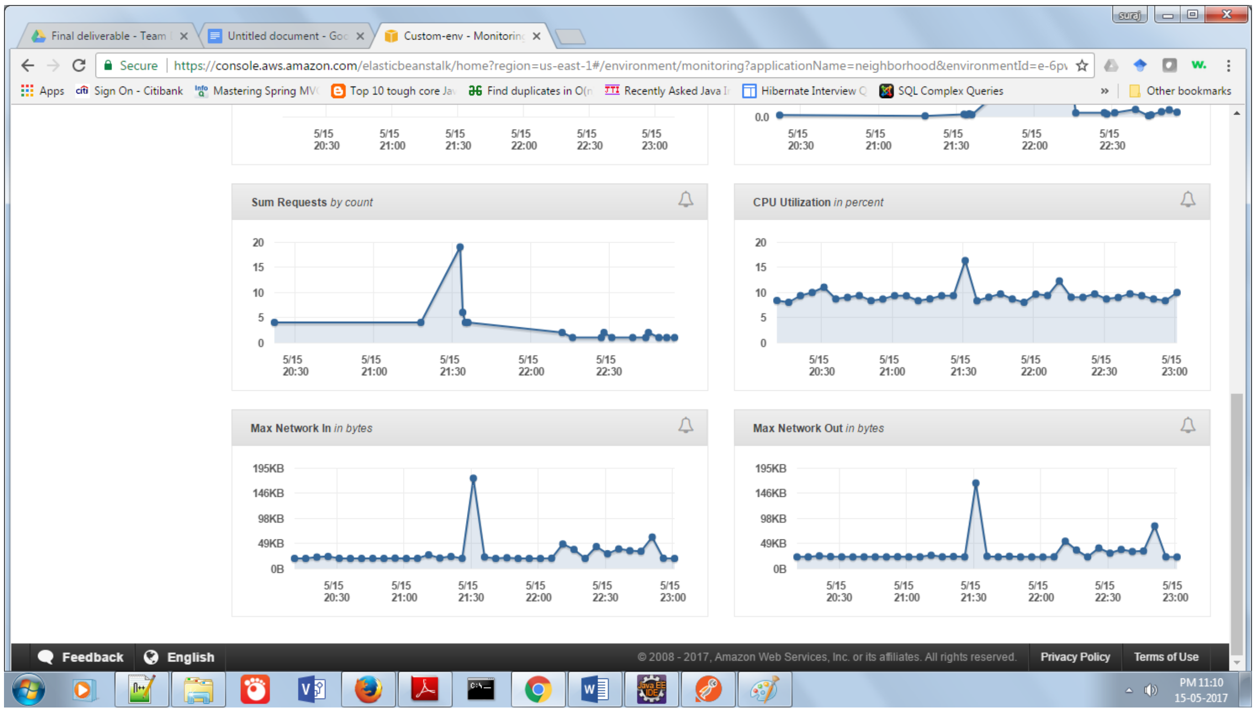
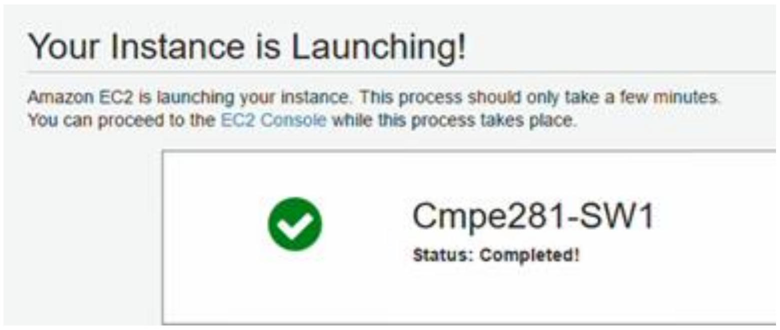


Figure 8.6

EC2 instance WEB UI deployment:



Your Instance is Launching!

Amazon EC2 is launching your instance. This process should only take a few minutes. You can proceed to the EC2 Console while this process takes place.

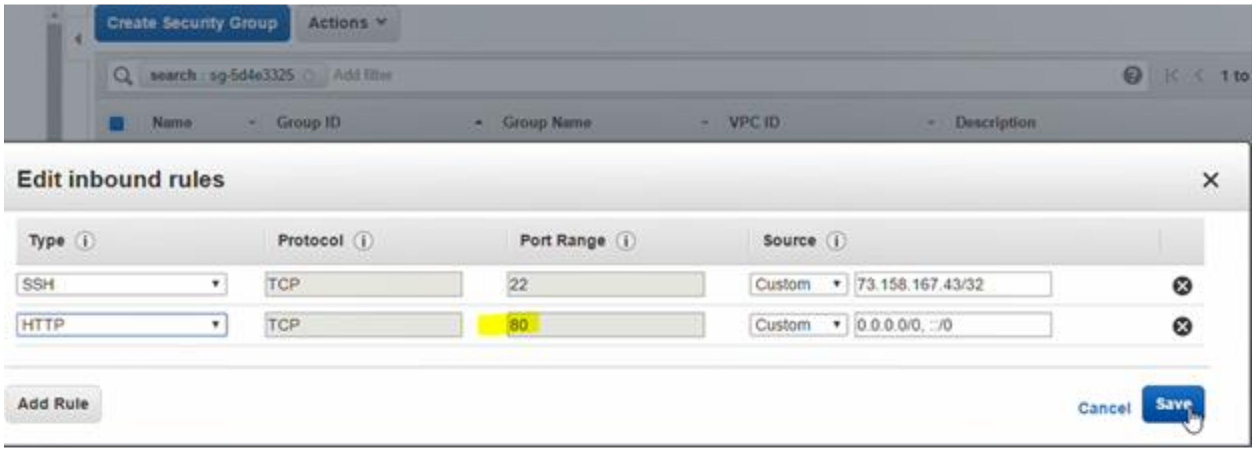
Cmpe281-SW1

Status: Completed!

Figure 8.7

Security group configuration:

Configure the running EC2 instance to access the deployed application over internet.

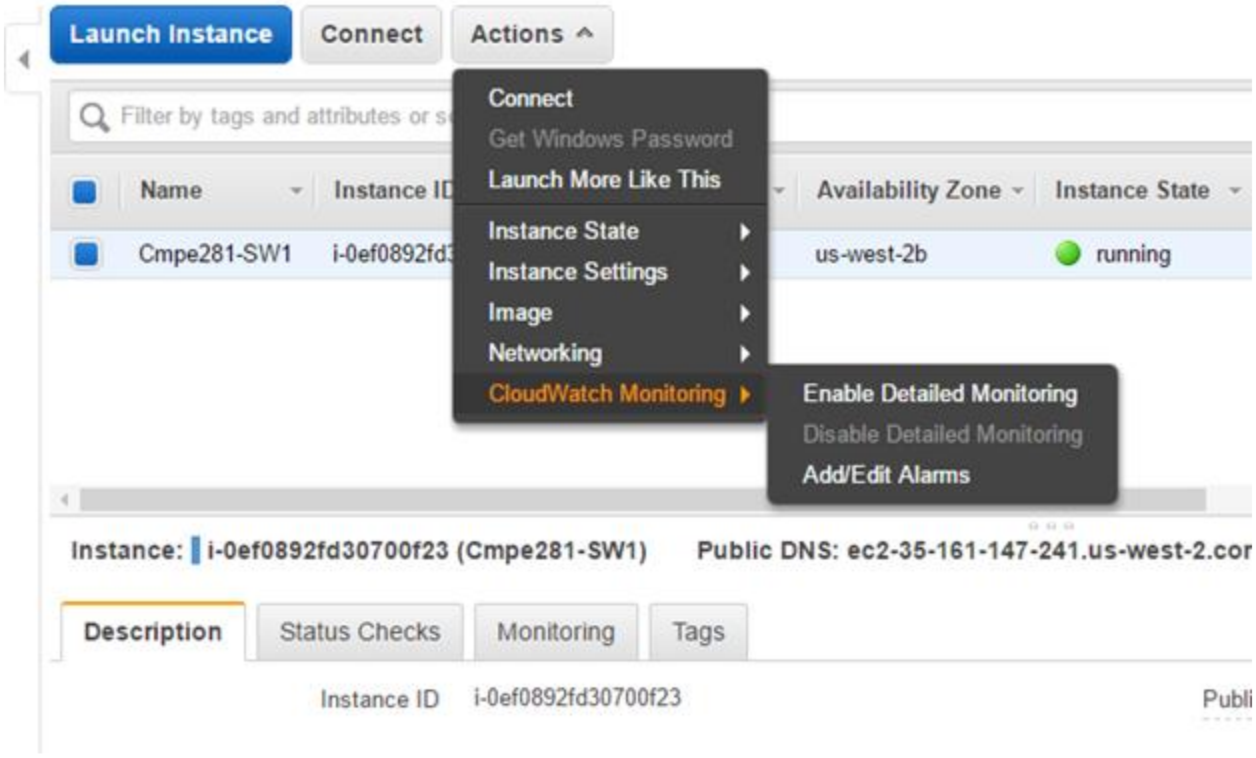


Edit inbound rules

Type	Protocol	Port Range	Source	
SSH	TCP	22	Custom	73.158.167.43/32
HTTP	TCP	80	Custom	0.0.0.0/0, ::/0

Figure 8.8

Detailed CloudWatch monitoring is enabled from the Action menu bar drop down option 'Enable Detailed Monitoring'.

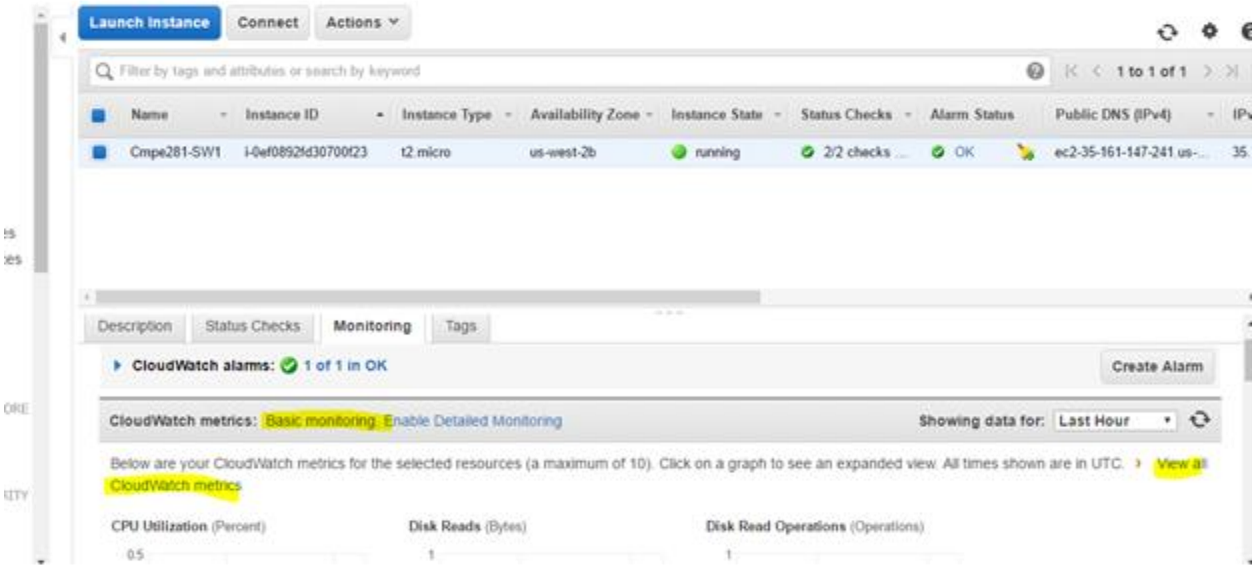


Name	Instance ID	Availability Zone	Instance State
Cmpe281-SW1	i-0ef0892fd	us-west-2b	running

Instance: i-0ef0892fd30700f23 (Cmpe281-SW1)   Public DNS: ec2-35-161-147-241.us-west-2.cor

Instance ID   i-0ef0892fd30700f23

Figure 8.9



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv
Cmpe281-SW1	i-0ef0892fd30700f23	t2.micro	us-west-2b	running	2/2 checks ...	OK	ec2-35-161-147-241.us-...	35.

CloudWatch alarms: 1 of 1 in OK

CloudWatch metrics: Basic monitoring. Enable Detailed Monitoring

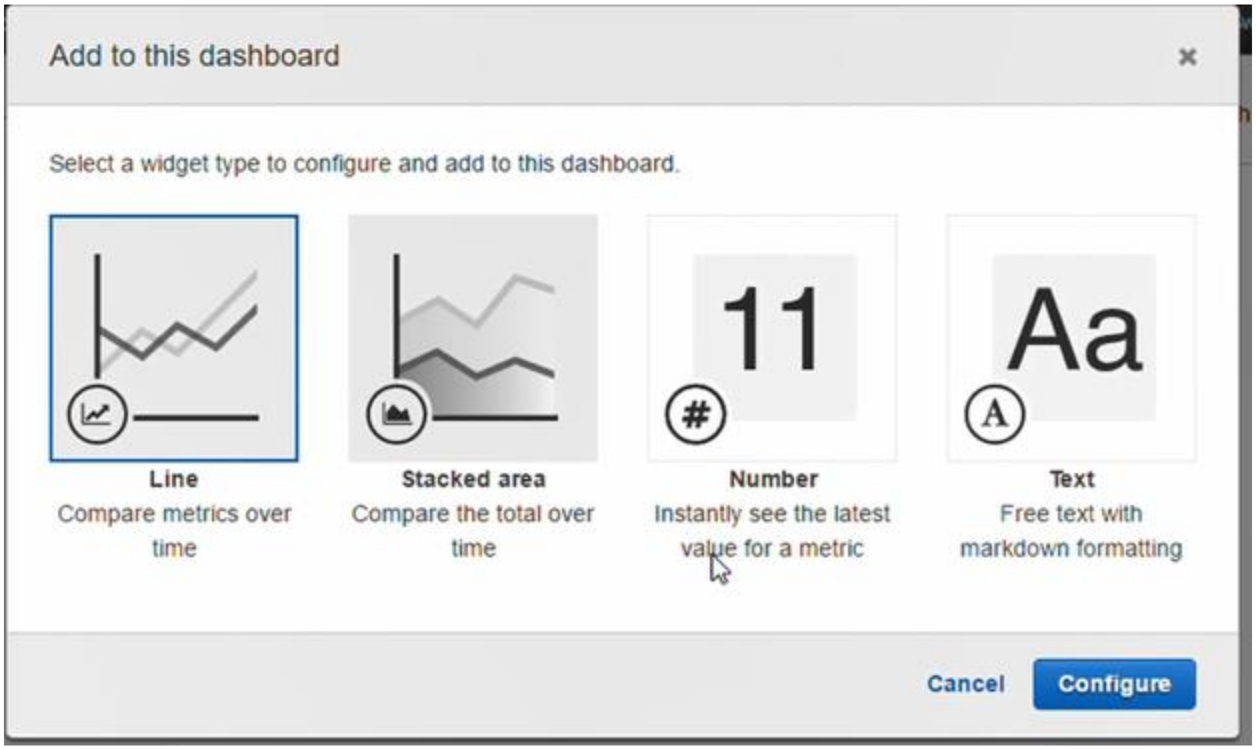
Showing data for: Last Hour

Below are your CloudWatch metrics for the selected resources (a maximum of 10). Click on a graph to see an expanded view. All times shown are in UTC. › View all CloudWatch metrics

CPU Utilization (Percent) Disk Reads (Bytes) Disk Read Operations (Operations)

Figure 8.10

Create and Configure the Cloudwatch Dashboard:



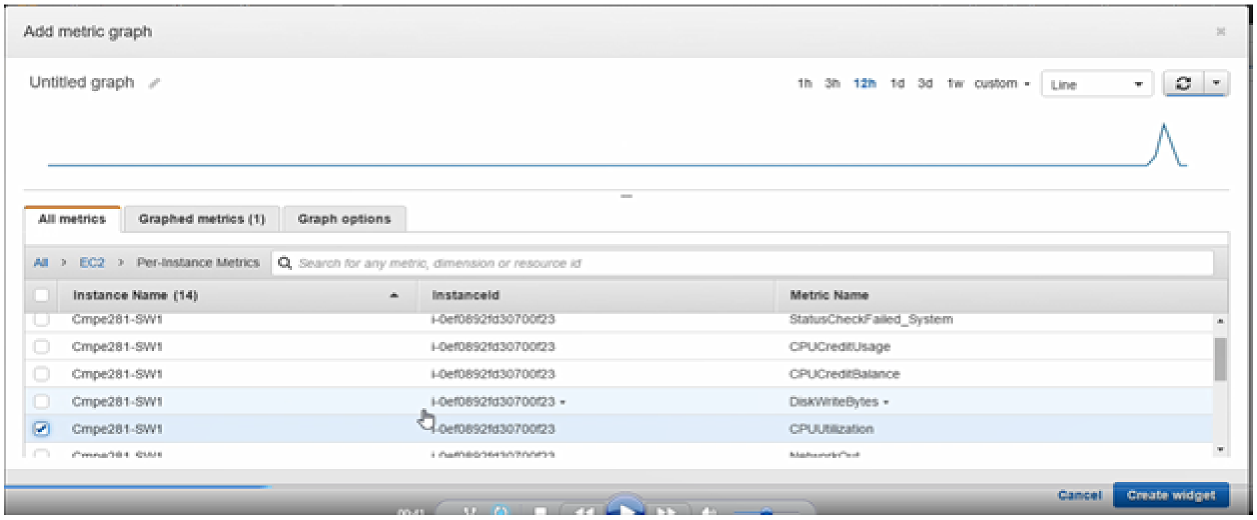
Add to this dashboard

Select a widget type to configure and add to this dashboard.

Line
Compare metrics over time
Stacked area
Compare the total over time
Number
Instantly see the latest value for a metric
Text
Free text with markdown formatting
Cancel
Configure

Figure 8.11

CPU utilization:



Add metric graph

Untitled graph

1h 3h 12h 1d 3d 1w custom Line

All metrics | Graphed metrics (1) | Graph options

All > EC2 > Per-Instance Metrics

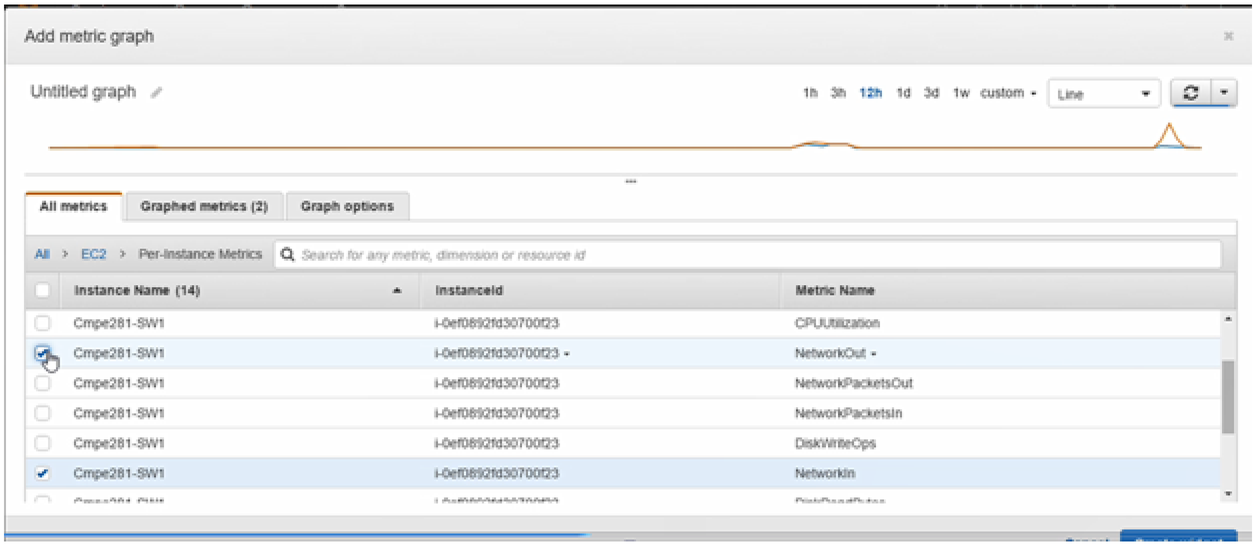
Search for any metric, dimension or resource id

	Instance Name (14)	InstanceId	Metric Name
	Cmpe281-SW1	i-0ef0892fd30700f23	StatusCheckFailed_System
	Cmpe281-SW1	i-0ef0892fd30700f23	CPUCreditUsage
	Cmpe281-SW1	i-0ef0892fd30700f23	CPUCreditBalance
	Cmpe281-SW1	i-0ef0892fd30700f23	DiskWriteBytes
✓	Cmpe281-SW1	i-0ef0892fd30700f23	CPUUtilization

Cancel Create widget

Figure 8.12

Network In/Out:



Add metric graph

Untitled graph

1h 3h 12h 1d 3d 1w custom Line

All metrics | Graphed metrics (2) | Graph options

All > EC2 > Per-Instance Metrics

Search for any metric, dimension or resource id

	Instance Name (14)	InstanceId	Metric Name
	Cmpe281-SW1	i-0ef0892fd30700f23	CPUUtilization
✓	Cmpe281-SW1	i-0ef0892fd30700f23	NetworkOut
	Cmpe281-SW1	i-0ef0892fd30700f23	NetworkPacketsOut
	Cmpe281-SW1	i-0ef0892fd30700f23	NetworkPacketsIn
	Cmpe281-SW1	i-0ef0892fd30700f23	DiskWriteOps
✓	Cmpe281-SW1	i-0ef0892fd30700f23	NetworkIn

Figure 8.13

CloudWatch alarm is created to monitor the resource utilization:

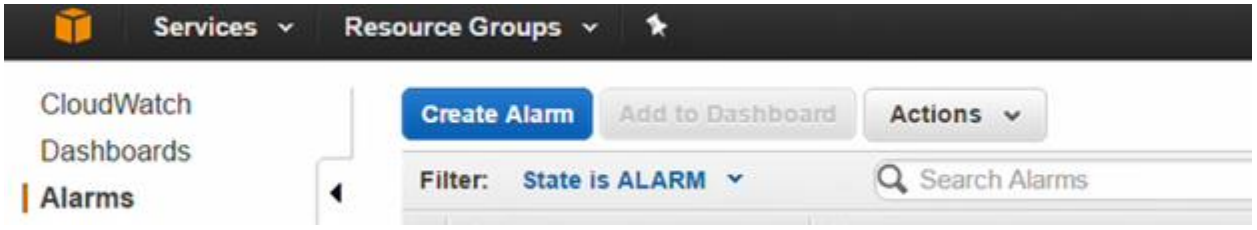
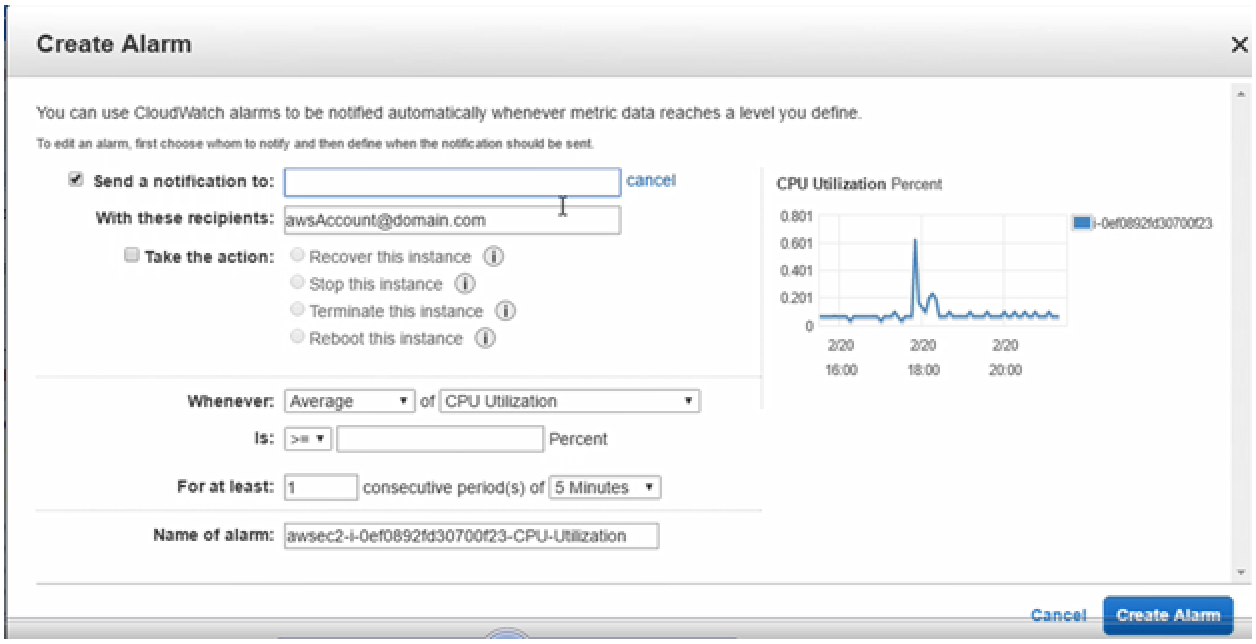


Figure 8.14



You can use CloudWatch alarms to be notified automatically whenever metric data reaches a level you define.

To edit an alarm, first choose whom to notify and then define when the notification should be sent.

[✓] Send a notification to: cancel

With these recipients: awsAccount@domain.com

[ ] Take the action:
- Recover this instance
- Stop this instance
- Terminate this instance
- Reboot this instance

Whenever: Average of CPU Utilization

Is: >= Percent

For at least: 1 consecutive period(s) of 5 Minutes

Name of alarm: awsec2-i-0ef0892fd30700f23-CPU-Utilization

Figure 8.15

Chapter 9. Conclusion

A neighborhood community is taken as an example for community based social network. Functionalities for each type of user are defined and based on those requirements the system is designed. We have considered three different types of users who would be using the system. A moderator, who is super user of the system can manage different clusters of the communities. An admin, responsible for requesting a cluster, managing it, configuring services and users for the cluster. Finally, end users who would be added to a cluster by admin can make use of platform to access different available services. Currently we have services like, Report Illegal Dumping, Report Illegal Parking, Report Crime/Theft in the Neighborhood. Also, there is an option to post what's new in the neighborhood. We have tried to create a collaborative platform for community to make it more social, safe and secure. Future scope of the platform includes, communication between clusters, providing cross community services to users, users to users communication within cluster and across clusters.

Chapter 10. References

[1] Linked.Gov - Government Social Network and Workflow Software Defined, Fuyao Zhang, Khanh Dao, Yu Cheng, Yuexing Yin, *May 2017*.

[2] Edge Based Mobile Service System for Illegal Dumping, Dr. Jerry Gao.