Real World Smart Chatbot for Customer Care using a Software as a Service (SaaS) Architecture

Godson Michael D'silva[1, *], Sanket Thakare[2], Sharddha More[1], and Jeril Kuriakose[1]

[1]Information Technology Department, St. John College of Engineering and Technology, Palghar, India, dsilvagodson@gmail.com
[2]Computer Department, Thakur College of Engineering & Technology, Mumbai, India

Abstract— It's being very important to listen to social media streams whether it's twitter, facebook, messenger, LinkedIn, email or even company own application. As many customers may be using this streams to reach out to company because they need help. The company have setup social marketing team to monitor this stream. But due to huge volumes of users it's very difficult to analyses each and every social message and take a relevant action to solve users grievances, which lead to many unsatisfied customers or may even lose a customer. This papers proposes a system architecture which will try to overcome the above shortcoming by analyzing messages of each ejabberd users to check whether it's actionable or not. If it's actionable then an automated Chatbot will initiates conversation with that user and help the user to resolve the issue by providing a human way interactions using LUIS and cognitive services. To provide a highly robust, scalable and extensible architecture, this system is implemented on AWS public cloud.

Keywords— Ejabberd; AWS Lambda; Machine Learning; LUIS; Chatbot; API Gateway; Cognitive Services.

I. INTRODUCTION

Nowadays it's a vital for any company to have the infrastructure to listen to social media streams whether it's twitter, facebook, messenger, LinkedIn, email or even company own application. Because most of the customers are using this streams to reach out to company as they need help regarding a particular product or service. The company have to build a platform by which the customer care representatives reach out to this people to ensure that they are happy. Till now the companies have set up huge customer care centers, where the users have to call, go through the IVR process which will redirect them to the customer care representatives. This is usually very time consuming and frustrating for the customer who has to wait until their chances come to talk to the customer representatives. In today's world, no has that much time to talk with the customer care representatives. This have resulted in many unsatisfied customer over the years. Even though the company spending so much on customer care service, there are not able to give an optimal outcome.

The emerging trend of online chatting is gaining popularity, where the customer can chat with the customer care representatives at point of time from anywhere. This give the freedom for the customers to express their issue in a more elaborated way. As it turns out, AI is starting to truly become mainstream and 2017 looks like it's going to be full of new technologies and platforms. The Chatbots is one of those technologies which is going to change the way we do conversations till now. A Chatbot is a service, powered by rules and sometimes artificial intelligence that you interact with via a chat interface. The service could be any number of things, ranging from functional to fun, and it could live in any major chat product.

To papers tries to build a system which can resolve the above issues. For such overpowering system, this paper proposes a Robust, Scalable and Extensible architecture with technologies stack consisting of ejabberd server, AWS web services and Chatbots. In this paper, the ejabberd server will handle the chatting part, the AWS web services will analyses the messages to check whether any actionable message has come and the Chatbot will act as a customer representative to resolve the issues the customer has been facing. Moreover this papers also illustrates addition of room functionality in ejabberd server to make participant in that room persistent over time.

Organization of paper: Section II and III studies the literature survey and related theory. Section IV proposes the conceptual framework used in our paper and section V describes the implementation part. Section VI mentions the results obtained. Discussion and future events are mentioned in section VII and section VIII concludes the paper.

II. LITERATURE SURVEY

In August 2011,Jaume Jordán et.al [1] proposed Multi-Agent Systems which are suitable to provide a framework that allows to perform collaborative processes in distributed environments. In a customer support system with operators attending incidences, the problem to solve is to find out the best solution for the problems reported to the system. An argumentation framework for a Multi-Agent System applied to customer support is proposed to help agents to reach an agreement and jointly solve incidences. In this paper [2], G. B. Satrya et al. discuss remnant data from private chat, secret chat, and hidden chat in social messenger applications for Android by explaining the artifacts which are produced by social messengers. Also provide interpretations of generated messages as well as how they relate to one another. Based on the investigation results of Android forensics and analysis in this paper, an analyst or investigator will be able to read, reconstruct, and present the chronology of the messages which have been created by the user. By using two smartphones with different brands and different Android OS versions as experimental objects, we conducted a digital investigation in a forensically sound manner.

In august 2015, Bogdan Ionescu et al. [3] Introduced new chat-driven architecture for collaborative rich-text and media editing and consuming. The chat has a central role in coordinating group activities. The goal of the new platform is the production of information or content by one or all of the members of the group synchronously. Central to this new concept are operational transformations and collaborative "Smart Objects". These objects are entirely web-based and contain information related to their embedding in the content, as well as interactive functions to assist the users in their collaborative work. Real-life experiments using this new platform are described at the end.

In this paper [4] Haoliang Wang et.al proposes an interaction design of a social TV real-time chatting application, called Touch Talk, which aiming at providing a better experience on program communication and comments sharing with each other in real-time while watching TV. For this purpose, we investigated some basic properties of social TV and analyzed viewer's behavior. Based on the results of these research, the formation of a platform and some design principles for social TV are proposed, making it reasonable and meaningful to design Touch Talk.

III. Related Theory

In this paper the public cloud based architecture is discussed in detail. In this proposed system architecture is implemented majorly over the technologies, such as Ejabberd server, AWS lambda, API Gateway and Chatbot. This technologies are elaborated in detail in this section below.

A. Ejabberd Server

The ejabberd is an open source instant messaging server written in erlang. It is cross-platform, distributed, fault-tolerant, and based on open standards to achieve real-time communication. It uses XMPP protocol on the backend. In ejabberd, the mobile reliability layer helps manage mobile network's disconnection, message deliveries, and a consistent conversation on any online device, push notification and mobile interfaces. Message is not loss in ejabberd because of bad network connectivity. Each time when message send, ejabberd check message delivery with an acknowledgement provide by device.

B. AWS Lambda

AWS Lambda is an event-driven, server less computing platform, provided that user can write their application code in languages supported by AWS Lambda (that is, Node.js, Java, and Python). Application code running using the AWS Lambda standard runtime environment and resources provided by Lambda without provisioning or managing servers. AWS From a few requests per day to thousands per second, lambda executes code only when needed and scales automatically. User can also run code for virtually using AWS lambda with any type of application or backend service along with zero administration.

C. Amazon API Gateway

The Amazon API Gateway is a service which is fully managed that makes it easy for developers to create, publish, maintain, monitor, and secure APIs. It is front door to data logic and services which has no minimum fees or start-up process. API Gateway helps developers deliver robust, secure, and scalable mobile and web application back ends. Developers securely connect mobile and web applications to business logic hosted on AWS Lambda, APIs hosted on Amazon EC2, or other publicly addressable web services hosted inside or outside of AWS using API Gateway.

D. Chatbot

The Chatbot stands for chatter robot. It is a computer program which has the ability to chat with human users. It is a conversational programs used for chatting application in such a way that the human user cannot realize that he is talking to a software program. Chat bots are used in applications such as e-commerce customer service, call centers and Internet gaming. Chat bots are applying in areas where purposes are typically limited to conversations regarding a specialized purpose and not for the entire range of human communication. A Chatbot is an artificial intelligence computer software program developed to simulate intelligent conversation through written or spoken text.

E. ML Lambda

The Machine learning lambda makes it better experience for developers of all skill levels to use machine learning technology. The Machine Learning lambda guides developers through the process of creating ML models and then generates predictions for the applications. The Machine Learning on lambda is based on the same proven, highly scalable, ML technology used for years by data scientist community. Powerful algorithms are used to create ML models by finding patterns in the existing data. Then, Machine Learning uses these models to process new data and generate predictions for the application.

IV. Conceptual Framework

The proposed system architecture is a highly scalable and robust as it's implemented over AWS public cloud. This architecture consist of five main components such as ejabberd server, API gateway, lambda, ML lambda and Chatbot as illustrated in Figure 1. The client can be a web application or a mobile client which communicates with the ejabberd server through an xmpp protocol. The ejabberd will allocated a unique jabber id for every registered xmpp client. The xmpp client can send one to one message to other users or they can create a room and invite other users to join the room and have a conversation. The ejabberd server in this architecture is capable of handling offline messages and persisting users in a room unless and until they are being removed or they themselves exit from that room.

The ejabberd server post the messages communicated over its channel to the API gateway whether it's a one to one message or group chat message. The API gateway is robust and scalable to handle enormous amount of post request at a same time. This gateway injects this message's payload to the lambda which is a computational function running to inspect whether the injected message is actionable or not. The way the lambda does this is by consulting the machine learning lambda which is trained with a huge number of datasets, so that it accurately predicts which messages are actionable and which are not. The machine learning lambda consist of a supervised learning algorithm called logistic regression which is widely used and more precise.

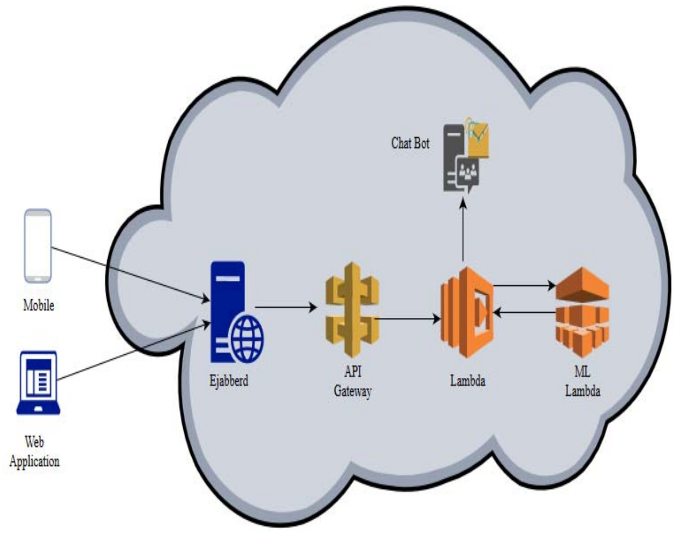


Figure 1: Generalized Architecture

The machine learning lambda return the response to the lambda whether a message is actionable or not. Based on this response the lambda takes the action. If the messages is actionable then it will trigger it to the Chatbot with the users jabber id or conferences id and If the is not actionable then it will take no action. The Chatbot is an automated bot which is powered with LUIS (Language Understanding Intelligent Service) and Cognitive service API. Due to which the Chatbot communicates with the user in a more human way. As this Chatbot's are implemented in a lambda function on cloud it's highly scalable and faster in providing responses. The Chatbot initiates conversation with the user who were having grievances about any services and try to get more details of the problem and provide a solution to the user.

V. IMPLEMENTATION

This proposed architecture is divided into three main section that is persisting the users in a room even when they go offline, analyzing the group and one to one messages in ejabberd and lastly the Chatbot conversion operation. All this operation are elaborated in detail below.

A. Persisting Users in Room Operation

When a room is created in ejabberd by a particular user then by default that user is added in that group. Other users can join that room based on its conferences id. The admin user can invite other users to join or even remove an existing user from the room. The ejabberd server, removes the users from the room when that user goes offline. So when the user come online, he has to again send a stanza to join the room. And moreover that user will not receive the messages which were send in the room when he was offline. In this proposed system, this drawback is overcome by making the user persist in that group even if that user goes offline. When the user has join the room its details are persisted in the MySQL database. So whenever the user goes offline his timestamp is recorded in the database table. Now when that user come online again his offline timestamp is checked and from that timestamp only all the message are send to that user. Thus he can view all the message which were send when he was offline.

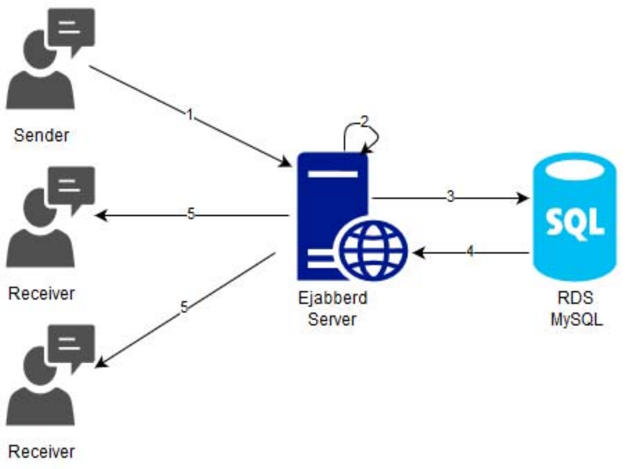


Figure 2: Room Persistent in Ejabberd Operation

This operation assumes that a room is being created and the future operation is illustrated in Figure 2 with steps as follows:

1. User sends a message stanza to the room consisting of the jabber id, conferences id of the room.
2. The Ejabberd server receives the stanza, validates that it a valid stanza or not and then store it.
3. The ejabberd server then fires a query to get jabber id of all the users who are a participants of that room to the muc_room table.
4. The SQL database send the list of jabber ids as a response of the query to ejabberd server.
5. The ejabberd server then forward this message to all the jabber id except the sender.

B. Process Regular conversion Operation

In this operation the client is communicating with the ejabberd server through a XMPP protocol. The ejabberd server is robust and fault tolerant server, which provides a facility for offline storage of message with the users are not online. The client has to register to this server, after which it will be provided with a unique jabber id. Now with the help of this jabber id, the user can have one to one chat with other users or else they can create a room and add users in that room for group chat. The detailed operation is formulated in below steps as illustrated in Figure 3.

1. The mobile or a web application send a chat message which can be a group or a one to one message to the ejabberd server through a XMPP protocol.

2. The ejabberd server check the jabber id of the sender whether it's valid or not and based on the chat type either group chat or one to one chat, post the message payload to the respective API Gateway.
3. The API Gateway validates and redirects this payload to respective lambda.
4. The lambda send each payload to machine learning models
5. The machine learning models will decides whether a payload is actionable or not.
6. The machine learning model returns the relevant result to the lambda.
7. Lambda check if the result is Actionable then forwards the payload to the Chatbot or if result is not actionable then does not take any action.

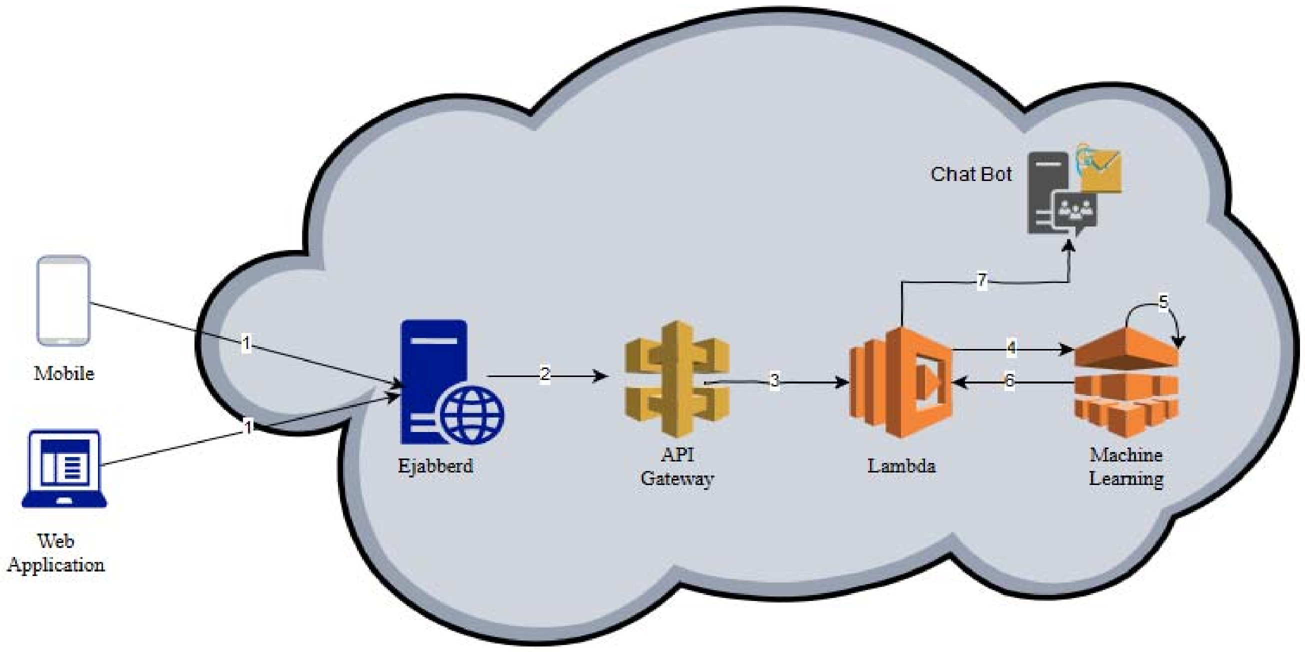


Figure 3: Analysing Group and One to One Messages from Ejabberd Operation

C. Chatbot Conversion Operation

In this operation the actionable payloads on which a necessary action needs to be taken are injected to this Chatbot through lambda as illustrated in the above operation. The Chatbot is a powered with intelligent services such as LUIS (Language Understanding Intelligent Service) and other cognitive services. So that it does the conversation with the user or group of users in a more human like way. The Chatbot is implemented on a bot framework which is running on an ec2 instances in AWS cloud.

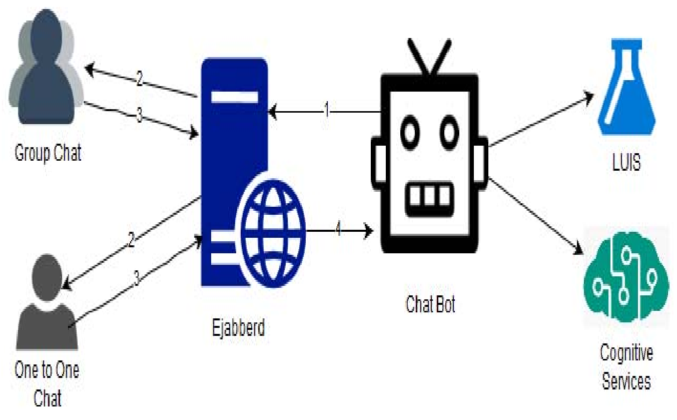


Figure 4: Representation of Nodes and Relationship

The detailed steps of the operation is formulated below as illustrated in Figure 4.

1. The Chatbot understand the issues and starts the conservation with the respective jabber id or room id by sending the message to ejabberd.

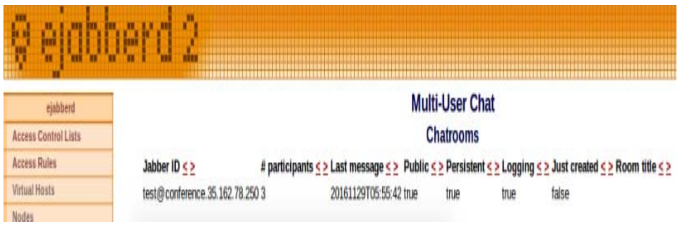
2. The ejabberd checks the payload and forwards the message to the respective jabber id or room id.
3. The user replies with the issues issue details which is send to ejabberd server.
4. The ejabberd server forwards the message to the Chatbot which the help of the cognitive services api and machine learning replies and try to solve the issue.

VI. RESULTS

In this paper a highly robust, scalable, pluggable and faster architecture is proposed which tries to simply the process of customer service using Chatbots. The results discussed in this papers follow a flow where first the users in a room created in ejabberd are made persistent. After that the chats either it be a group chat or one to one chat are posted to the API gateway which injects it's to the lambda. In the lambda the relevant processing of whether the message is an actionable or not is done by consulting the ML lambda. The actionable messages are send to the Chatbot which then initiates the conversion with the respective user and tries to resolve their issue by communicating with them in a more human way.

A. Making Users Persistent in Room

In the proposed system the admin user creates a room called 'test' as illustrated in Figure 5. The admin user adds peter and jack in the room. They starts their conversion in the room as shown in Figure 6.

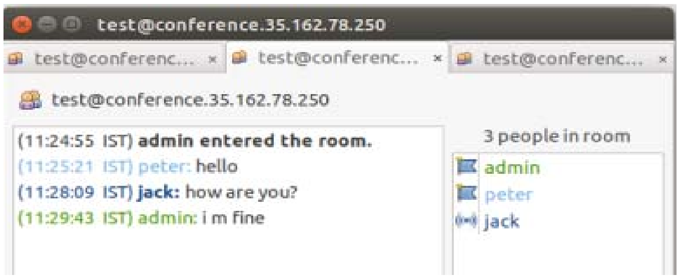


Multi-User Chat

Chatrooms

Jabber ID	# participants	Last message	Public	Persistent	Logging	Just created	Room title
test@conference.35.162.78.250	3	20161129T05:55:42	true	true	true	false	

Figure 5: Ejabberd Web Portal

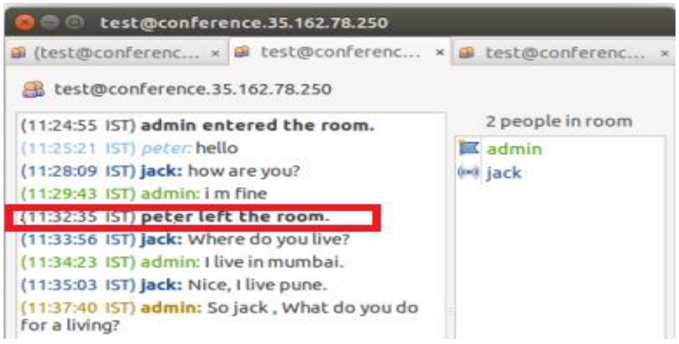


(11:24:55 IST) admin entered the room.
(11:25:21 IST) peter: hello
(11:28:09 IST) jack: how are you?
(11:29:43 IST) admin: i m fine

3 people in room
admin
peter
jack

Figure 6: Room Chat Conversation

After some time peter goes offline. The user jack and admin continues their discussion while peter being offline as illustrated in Figure 7. So when peter comes online, the ejabberd server check if that user is a participant of any room by communicating with the database.



(11:24:55 IST) admin entered the room.
(11:25:21 IST) peter: hello
(11:28:09 IST) jack: how are you?
(11:29:43 IST) admin: i m fine
(11:32:35 IST) peter left the room.
(11:33:56 IST) jack: Where do you live?
(11:34:23 IST) admin: I live in mumbai.
(11:35:03 IST) jack: Nice, I live pune.
(11:37:40 IST) admin: So jack , What do you do for a living?

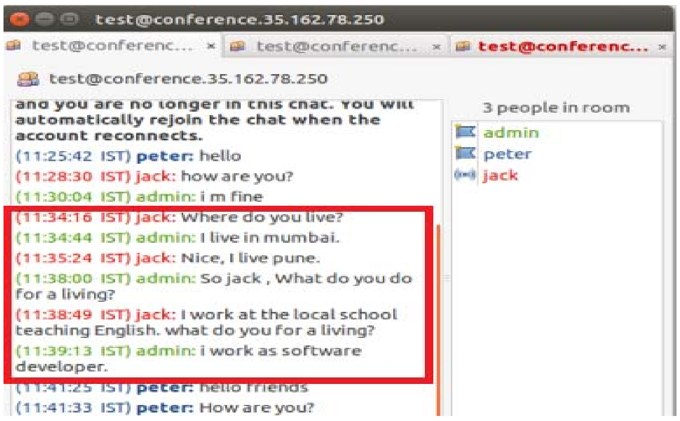
2 people in room
admin
jack

Figure 7: Room Chat Conversation with offline user

If the user is a participant of a room then the ejabberd server automatically joins him to that room and send that user all the messages from the time he had gone offline as illustrated in Figure 8.

B. Posting Messages to Lambda from Ejabberd

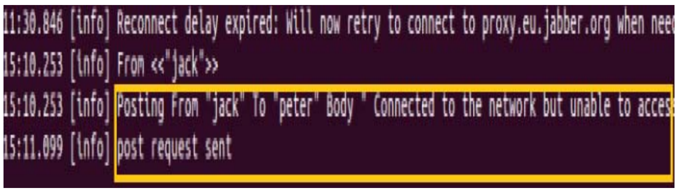
When ejabberd server receives a message stanza coming from client, it first checks whether the stanza is coming from an authorized source and based on it post this message stanza as a JSON to the Api Gateway as shown in Figure 9.



and you are no longer in this chat. You will automatically rejoin the chat when the account reconnects.
(11:25:42 IST) peter: hello
(11:28:30 IST) jack: how are you?
(11:30:04 IST) admin: i m fine
(11:34:16 IST) jack: Where do you live?
(11:34:44 IST) admin: I live in mumbai.
(11:35:24 IST) jack: Nice, I live pune.
(11:38:00 IST) admin: So jack , What do you do for a living?
(11:38:49 IST) jack: I work at the local school teaching English. what do you for a living?
(11:39:13 IST) admin: i work as software developer.
(11:41:25 IST) peter: hello friends
(11:41:33 IST) peter: How are you?

3 people in room
admin
peter
jack

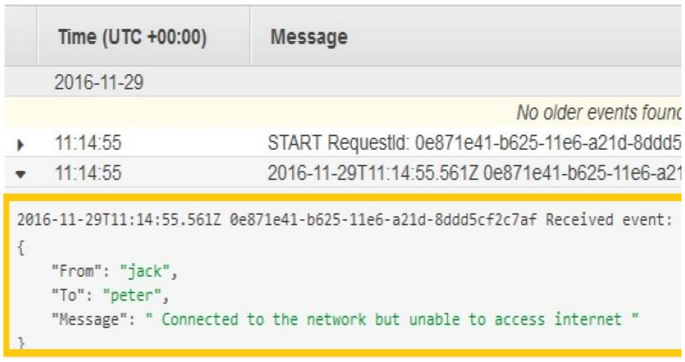
Figure 8: Room Chat Conversation with offline user joining



```
11:30.846 [info] Reconnect delay expired: Will now retry to connect to proxy.eu.jabber.org when nee
15:10.253 [info] From <<"jack">>
15:10.253 [info] Posting From "jack" To "peter" Body " Connected to the network but unable to acces
15:11.099 [info] post request sent
```

Figure 9: Ejabberd Console Terminal

Amazon API Gateway provides developers with a simple, flexible, fully managed, pay-as-you-go service that handles all aspects of creating and operating robust APIs for application back ends. The API Gateways forwards this message to the lambda which is manage compute service that allows developers to run code in response to events without setting up or managing any compute infrastructure. The AWS lambda logs the message received from the API Gateway as illustrated in Figure 10.



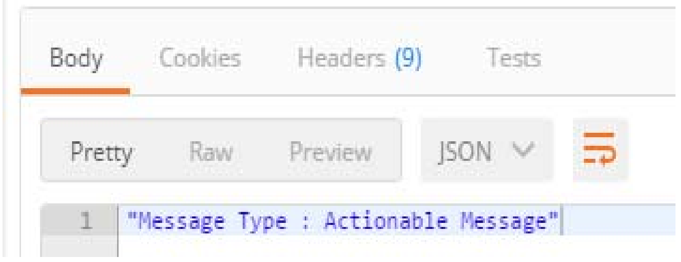
```
Time (UTC +00:00)    Message
2016-11-29
                                   No older events founc
▸ 11:14:55     START RequestId: 0e871e41-b625-11e6-a21d-8ddd5
▾ 11:14:55     2016-11-29T11:14:55.561Z 0e871e41-b625-11e6-a21

2016-11-29T11:14:55.561Z 0e871e41-b625-11e6-a21d-8ddd5cf2c7af Received event:
{
    "From": "jack",
    "To": "peter",
    "Message": " Connected to the network but unable to access internet "
}
```

Figure 10: AWS Cloud Watch Logs

C. *Lambda Consulting ML-Lambda for Actionable Messages*

The lambda is used to decide whether each message coming is actionable or not. The way the lambda decide is by consulting a machine learning lambda model. The machine learning lambda model is trained with a training set of message, so that it can predict whether a given message is actionable or not. In the proposed system logistic regression algorithm is used for training the machine learning lambda. The lambda response is recorded of the actionable messages in postman is illustrated in Figure 11.

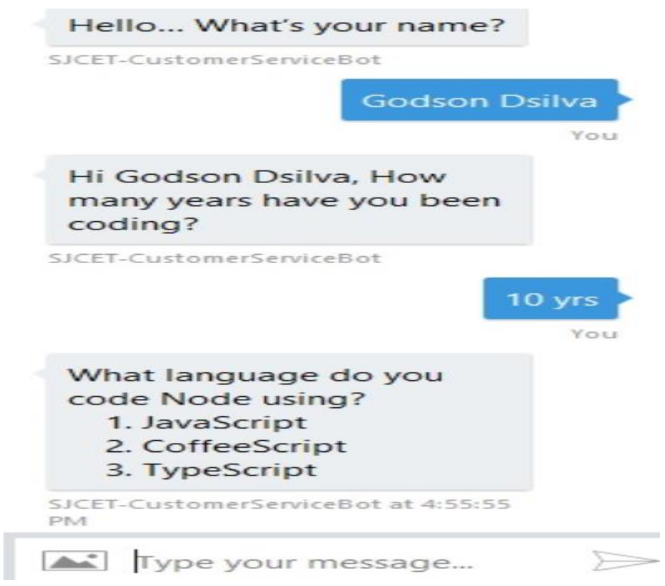


```
"Message Type : Actionable Message"
```

Figure 11: Postman ML Lambda response

D. *Lambda Chatbot Conversation with Ejabberd Users:*

The lambda triggers the Chatbot with jabber id of users having actionable messages. The Chatbot initiates chat with the respective jabber id user through ejabberd. The Chatbot is built on bot framework and is powered with Luis and cognitive services to talk with user in more human way. The Chatbot interact users with various options of card, list images and not just plain old text. The Chatbot automates the process of capturing the issues details from the user and lodged a complaint request of it or provide a feasible solution to the users. The user can interact with the Chatbot at his own feasible time and moreover never have to wait for any support. The Chatbot interaction is illustrated in Figure 12.



Hello... What's your name?
SJCET-CustomerServiceBot

Godson Dsilva
You

Hi Godson Dsilva, How many years have you been coding?
SJCET-CustomerServiceBot

10 yrs
You

What language do you code Node using?
1. JavaScript
2. CoffeeScript
3. TypeScript
SJCET-CustomerServiceBot at 4:55:55 PM

Figure 12: Chatbot Conversation

VII. Discussion and Future Events

In [5], [6] the authors mentioned the benefits of using cloud for online signature verification. The effectiveness of using cloud as a platform for wireless sensor networks is mentioned in [7], [8], [9], and [10]. The need for better conversational interface is discussed in [11], [12], and [13]. The efficacy of SaaS in malware detection is studied in [14]. Generally, direct connection is maintained in a one-to-one connection, if 10 one-to-one messages are sent, it creates 10 separate connections using up 10 times the bandwidth. By using ejabberd the above overhead can be reduced. While using ejabberd the subscribers are loosely coupled and it also provides high scalability. Inflexible semantic coupling and message delivery issues are some of the limitations of using ejabberd.

VIII. Conclusion

Walking in footsteps of the emerging trend of online chatting which is gaining popularity, where the customer can chat with the customer care representatives at point of time from anywhere. This give the freedom for the user to express their feeling, problems in a more elaborated way. The proposed system architecture in this paper focus on analyzing this social chats by identifying whether the messages from the customers are actionable or not. All the actionable messages are send to the Chatbot which tries to resolve the issues faced by the user by initiating the conversation with the customer in a more human way. This save lots of money and resources of the company used for customer service and even making the customer more and satisfied. As this proposed system is implemented on AWS public cloud, it make this system capable of handling enormous amount of user base.

IX. REFERENCE

[1] J. Jordán, S. Heras and V. Julian, “A customer support application using argumentation in multi-agent systems," in 14th International Conference on Information Fusion pp. In Press. (2011) Bibtex, Aug 2011.

[2] G. B. Satrya, P. T. Daely and S. Y. Shin, " Android forensics analysis: Private chat on social messenger,"in Eighth International Conference on Ubiquitous and Future Networks 2016 (ICUFN 2O16), TU Wien, Vienna, Austria, Aug 2016.

[3] Bogdan Ionescu, Cristian Gadea and Bogdan Solomon, "A chat-centric collaborative environment for web-based real-time collaboration," in IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics (SACI),Timsoara, Romania, Aug 2015.

[4] Haoliang Wang, Chunhong Zhang and Ming Li," Social TV real-time chatting application design," in International Symposium on Wireless Personal Multimedia Communications (WPMC), Jan 2015.

[5] V. A. Bharadi and G. M. DSilva, “Online Signature Recognition Using Software as a Service (SaaS) Model on Public Cloud,” in 2015 IEEE International Conference on Computing Communication Control and Automation, pp. 65–72, Feb. 2015.

[6] G. M. DSilva and V. A. Bharadi, “Modified Online Signature Recognition Using Software as a Service (SaaS) Model on Public Cloud,” IEEE International Conference on Information Processing, December 16-19, 2015.

[7] Kuriakose, Jeril, et al. "A review on mobile sensor localization." International Symposium on Security in Computing and Communication. Springer Berlin Heidelberg, 2014.

[8] Kuriakose, Jeril, and Sandeep Joshi. "A Comparative Review of Moveable Sensor Location Identification." International Journal of Robotics Applications and Technologies (IJRAT) 3.2 (2015): 20-37.

[9] Kuriakose, Jeril, et al. "Assessing the Severity of Attacks in Wireless Networks." Proceedings of the International Conference on Recent Cognizance in Wireless Communication & Image Processing. Springer India, 2016.

[10] Kuriakose, Jeril, et al. "A review on localization in wireless sensor networks." Advances in signal processing and intelligent recognition systems. Springer International Publishing, 2014. 599-610.

[11] McTear, Michael, Zoraida Callejas, and David Griol. "Creating a Conversational Interface Using Chatbot Technology." The Conversational Interface. Springer International Publishing, 2016. 125-159.

[12] Lin, Lue, Luis Fernando D'Haro, and Rafael Banchs. "A Web-based Platform for Collection of Human-Chatbot Interactions." Proceedings of the Fourth International Conference on Human Agent Interaction. ACM, 2016.

[13] AWS Documentation, https://aws.amazon.com/documentation , Accessed on 09.04.2016, 7:09 AM.

[14] More, Shraddha S., and Pranit P. Gaikwad. "Trust-based Voting Method for Efficient Malware Detection." Procedia Computer Science 79 (2016): 657-667.