International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 08 | Aug 2019 www.irjet.net p-ISSN: 2395-0072

Hosting NLP based Chatbot on AWS Cloud using Docker

Deeba Unnisa[1], Sesha Bhargavi Velagaleti[2]

[1]*Mtech Student, Dept. of Information Technology, GNITS, Hyderabad*
[2]*Assistant Professor, Dept. of Information Technology, GNITS, Hyderabad*

Abstract - *Cloud computing is growing every day in the technology world and almost all the companies are trying to migrate their existing applications to cloud and focusing on developing cloud native applications. As cloud provides scalability, availability, improved performance and it mainly reduces the manageability from the companies end, so they do not have to manage and maintain the resources to support their applications as it is taken care by the cloud vendors. One such famous cloud vendor is Amazon that provides amazon web services (AWS) which include multiple services for compute, storage, networks, database, machine learning, artificial intelligence, content delivery, management tools, analytics, internet of things, security and identity, etc.,*

This paper focuses on developing an NLP based chatbot that queries from users and provide them relevant answers and or perform the required actions, later hosting the chatbot on AWS cloud using Elastic container service by dockerizing the application.

Key Words: AWS ECS, cloud computing, chatbot, docker, natural language processing.

1. INTRODUCTION

Any company would like to delight their customers to maintain a better relationship with them as customers play a major role in keeping the business running. So, organizations try to incorporate better technologies which can enhance the user experience and adds value to their product. In this paper we will be considering a scenario of legacy application which stores files in multiple binders and folders, so if a user has to search for a file they have to either remember where each file is present, or they have to navigate through multiple folders to get that file, which is a time consuming and tedious task.

1.1 Proposed system

The proposed solution for this problem is to use an NLP based chatbot that takes queries from users and gives them the relevant answer and or perform actions. The entire data from different files will be indexed in Elasticsearch as it is suitable to perform fast and efficient full text search on large volumes of data.

Then we create a web API that connects with the chatbot UI designed using Angular version 6, this API acts as a search API and mediator between the data layer that is Elasticsearch and the chatbot user interface. The API is responsible for taking natural language query from the chatbot and processing it then searching for data in Elasticsearch index. Once data is found the API returns data to the UI which then displays the data to user in a specific format to enhance user experience. Once the chatbot is working fine and is giving us the desired results, we will be containerizing it using Docker to provide cross platform compatibility. After the containerization process is completed the entire Application will be hosted on an AWS ECS cluster to enhance the availability and decrease response time to end users.

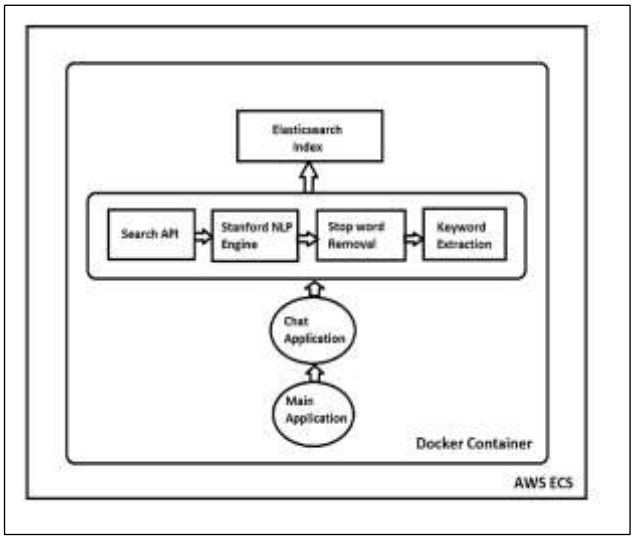

Fig -1: Proposed system

2. METHODOLOGY

There are mainly two stages that this paper covers, the first one is designing an NLP based chatbot and the second is to containerize it and host it on AWS cloud. For designing the chatbot we use techniques like Natural Language Processing (NLP), Application Programmable Interface (API), User Interface (UI) and for deploying the chatbot we use Docker and Amazon Web Services (AWS) which are discussed in detail in the following session.

2.1 Natural Language Processing (NLP)

NLP is a branch of artificial intelligence which studies how humans can interact with the computer in their natural language and how the computer understands and interprets human language. For this scenario we will be

using Stanford core NLP, it provides multiple modules to perform NLP related tasks such as lexical analysis, syntactic analysis, semantic analysis, disclosure integration and pragmatic analysis which also includes multiple sub tasks. For the chatbot we will be focusing mainly on parts of speech tagging and stop word removal as the users will be passing natural language queries and we need to process the queries based on each word and its parts of speech like noun, verb, preposition, etc.,

2.2 Elastic search

Elastic search is famous for full text search, it can be used to index an entire document and then traverse through the document to look for particular words. It stores data in the form of documents and each document looks like a JSON (JavaScript Object Notation) we can say that it is a document-oriented database which is designed to store, retrieve and manage semi structured data. Elasticsearch behind the scene uses Lucene's Standard Analyzer to index documents with precision. Elasticsearch will act as a database and data access layer between the Chatbot and the search API, all the application data will be present on Elasticsearch in Metadata index, where every word in each document will be indexed and stored in an inverted matrix. Here we will be using Bulk API to index the documents in Elasticsearch and Search API to search for the documents, for viewing the documents in Kibana we will be using Get API. We have around 4 lakhs of JSON files which contain application related data. Kibana is an analytics and visualization platform which allows to view data stored on the Elasticsearch clusters. We can search, view and interact with the data using the features provided by Kibana. It also provides different ways to view data in the form of graphs, pie diagram, charts, bar diagram, etc., which helps in data analysis.

2.3 Search API

To make our chatbot work we need to have a mechanism which acts as a bridge between the user interface and the data, for this we use ASP .NET to build a web API to perform search operation on the data stored in Elasticsearch. Here our aim is to take the query from user through UI, send it to the search API where we can utilize the tools provided by Stanford Core NLP to perform stop word removal, once the stop words are eliminated and we have the required keywords, the keywords will be searched against the Elasticsearch data based on the priority values set for each field in JSON in the config file. Once the keywords match Elasticsearch sends the documents back to search API which formats the documents for simplicity and sends them to the UI where they are displayed to the user.

2.4 User Interface

The aim is to design a simple chatbot which takes input from user and gives them the desired results, to achieve this we use Angular 6 and Bootstrap to develop the UI. The users can enter their queries either in text format or use the text to speech feature provided by mic, once the query is entered the chatbot sends this query to the API where it is processed, and the results are fetched, these results are displayed to the user in text format where the exact value of result is highlighted using an underline, the user can just click on the underlined text which is a hyperlink which takes users to the exact location inside the binder from where this value was fetched, this mechanism is called as "Deep Linking". Deep linking is performed by utilizing the fields received from the search API in the form of result and forming a hyperlink by constructing a meaningful URL from the fields. Chatbot also takes feedback from the users to improve its performance in the future and provide more interactive experience to the users.

2.5 Docker Containerization

Containerization allows us to create containers for applications, where each container acts as an individual computer system with all the dependencies for application like java, python, etc., pre-installed in it. In this case the user/developer does not have to worry about cross platform compatibility. Docker and Kubernetes are one of the tools that facilitates containerization of various applications. Containerized applications can also be hosted on different cloud environments making it ten times more convenient and cost effective.

Once we are satisfied with the functionality of our chatbot it is time to dockerize it which will allow us to test the application on a Linux container and host the application on a registry. We have four different services which helps us to run the chatbot successfully Elasticsearch, Kibana, Search API, User Interface. Therefore, we require 4 docker images one for each service. A docker file is script that gives out steps that a Docker should take to build a new image. For the index data in Elasticsearch we use docker volumes, Volumes are independent of the containers they are created separately and are utilized by the container while it is running, when the container is destroyed the volume and data inside volume is intact hence, they provide persistent data storage.

To run all the images in one go we use Docker compose, it is a tool which is used to run multi-container applications. To configure the services Docker compose uses a YAML file, using which with one command we can spin up all the containers at the same time. This is suitable for all types of environments such as developing, testing, staging and production. Steps to use Docker compose are as follows:

1. Create Docker file for all the services that your application requires, or utilize the images provided by Docker hub for licensed software.

2. Define all the services in "docker-compose" file, so that they can be run together and also specify any volumes or networks if required within this file.

3. To run the entire application at a time run "docker-compose-up" which will get all your services up and running in Docker containers.

2.6 Hosting the chatbot on AWS

It is easy to host containerized applications on AWS cloud rather than directly hosting the application, this can be done using AWS ECS (Elastic Container Service) and AWS ECR (Elastic Container Registry). Prerequisites for doing this is to set up a VPN, security groups, routing etc., on AWS.

Elastic Container Service (ECS) - It allows us to host Docker containers using orchestration service and it is highly scalable. ECS takes care of managing and scaling the cluster of virtual machines so we do not have to do it manually. Using API calls we can query the applications running on Docker inside ECS i.e. launch or stop the containers. This is how an AWS ECS works, AWS provides ECR (Elastic Container Registry) service which will be responsible for building and storing images, it is similar to Docker hub. These images are then utilized by ECS while defining the application requirements. Then it is our choice to either use EC2 containers to launch our application on Windows or Linux, or to go for AWS Fargate which is more of a server less option. Once the containers are running on ECS it takes care of managing the services and scaling resources when needed.

3. RESULTS AND DISCUSSION

This paper provides a way to remove latency and give better user experience by minimizing the efforts required by a user to find result, the chatbot will give user answers and perform required actions on the fly, so that users do not have to remember the location where each file is placed, and they do not have to manually navigate to that location to access the file. User can simply ask a question to the chatbot and it will find the file and give it to user.

After creating the chatbot it is deployed on the AWS cloud using docker, docker allows us to run applications inside a container, which means developers do not have to worry about the application not working on a system with different configuration. As docker will make sure that all the dependencies of our application are packaged in the form of a docker image which runs inside a Windows or Linux container.

The advantage of hosting docker image on AWS is it is cost effective, it allows us to scale up and scale down the resources based on the work load, it is suitable for easy access, moreover we do not have to manage the containers and images once they are deployed, AWS will take care of this automatically.

Fig -2: NLP based Chatbot

4. CONCLUSION

To make the chatbot more intelligent and efficient we can have block chain mechanism where any changes to the index will be approved by block chaining and also the feedback provided by user will help us to build more intelligent chatbot which exactly depicts the purpose of "Deep Learning". Block Chaining can also be used to feed this user specified feedback based on accuracy of answers provided by chatbot, in such case for a feedback to be valid it needs to be approved by at least 3 blocks in our case users. Which means if only one user is unsatisfied by a certain response then it won't hold much value to the chatbot when compared to the case were ten users specify the same feedback.

Another enhancement that can be done as a part of future scope for is to use Machine Learning services provided by AWS like "Amazon Lex" which is used to build chatbot, Amazon's "Alexa" also uses Lex service, in a way we can say that Lex in the brain of Alexa.

REFERENCES

[1] A Method to Extract Essential Keywords from a Tweet using NLP tools (by Tharindu Weerasooriy, Nandula Perera and S.R. Liyanage,2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (IC Ter))

[2] An overview of Artificial Intelligence based chatbots and an example chatbot application (by Naz Albayrak, Aydeniz Ozdemir and Engin Zeydan, 2018 26th Signal Processing and Communications Applications Conference (SIU))

[3] https://nlp.stanford.edu/

[4] https://d1.awsstatic.com/whitepapers/Deep_Learning_on_AWS.pdf?did=wp_card&trk=wp_card

[5] https://d1.awsstatic.com/whitepapers/docker-on-aws.pdf?trk=wp_card