

International Institute of Information Technology, Bangalore

Project Report

Spoken-English Evaluator using DevOps pipeline

**Under the Guidance of
Prof. B. Thangaraju**



Akshay Mahesh Gudiyawar
MT2020137

Vikrant Punia
MT2020049

Source code link: https://github.com/akshaymg99/SPE_Speech_Evaluator.git

Docker-hub link: <https://hub.docker.com/repository/docker/akshaymg99/speech-spe>

Table of Contents (Index):

Abstract -----→	3
Introduction -----→	3
System Configuration -----→	5
Design and Development -----→	6
Web application design -----→	7
Front end design -----→	7
Back end design -----→	10
Speech Analyzer Program Design -----→	12
Speech Record and text conversion -----→	12
Vocabulary strength calculation -----→	12
Syntax Analysis -----→	14
DevOps CI/ CD Pipeline -----→	15
Source code Management -----→	15
Build Stage -----→	18
Testing Stage -----→	18
Continuous Integration -----→	20
Continuous Delivery -----→	26
Continuous Deployment -----→	28
Continuous Monitoring -----→	30
Results and Discussion -----→	37
Scope for future work -----→	38
Conclusion -----→	38

Abstract

We have developed a web application which analyzes spoken English for basic interview questions and provides feedback based on syntax analysis (grammatical correctness), vocabulary-strength of the user, which will also indicate the relevance of the answer relative to question asked. In effect it acts like a AI based Mock Interview performance analyzer. Here, the web application displays the question on webpage, and the user has to speak the answer, so that the application records the audio in real-time. Details on mechanism and user interaction is explained in further sections ahead. The web-application takes in the audio input and converts it to text, then executes analyzer functions on the text-corpus. The analyzer functions include vocabulary strength calculation which is done by considering word frequencies in global English dictionary. Syntax analysis is done by using pre-trained BERT model, which is a machine learning model used for NLP (Natural Language Processing) tasks. The application also calculates unique word spoken which can be beneficial in assessing interview performance. So, in effect this web application helps users to check performance of their spoken English for basic Interview questions. This entire development process is done using DevOps CI/CD Pipeline approach. We carry-out version control of source-code, testing, build/dependency installations/environment setup, containerization, deployment, logging and monitoring using various DevOps tools. The tools which we used in this project are Git, Django-test-framework, python-logger, Jenkins, Docker, Rundeck and ELK Stack. The whole process from git-push to deployment is fully automated and hence performs well in industry environment. DevOps helps in collaboration for developers and operations team and to deploy quickly to the customers.

Introduction:

In this section, we provide insight into the problem statement and what effect does the solution have on respective community / industry.

Importance of problem statement: Here the problem statement attempts to solve the problem of providing performance feedback for students who are preparing for interviews related to their spoken English. It works on the audio input of the user's answer to a specific question. It is important for students to prepare for proper communicational skills, spoken English for their interviews. Therefore, this web application will prove to be useful in assessing their spoken English skills, their grammatical correctness and vocabulary strength, meaning how powerful words they use and how unique their words are. To avoid counting common stopwords such as 'the', 'and', we give them low scores. Apart from technical skills, since candidates are checked for proper communication skills, the need to improve their language skills is of vital importance.

This entire development should be carried out using DevOps CI/CD Pipeline. Thus making the process entirely automatic with proper production environment setting.

Solution Approach: We are building the web application in Django platform. The backend processing is done in Django (python framework), where it runs Machine learning programs required for analysis of the speech. The frontend is developed with HTML/CSS and bootstrap templates. JavaScript is used for functionalities such as keeping a timer for answering and changing the display texts of recording button with events. For recording the audio input of the user, we are using “speech_recognizer” module of python, which takes in audio signals and converts them into text corpus. Upon this text, we are performing vocabulary strength analysis, which considers word frequencies from global English dictionary <http://norvig.com/mayzner.html>. This score is calculated by taking inverse of their frequency, thereby giving reasonably less score for most commonly used words. We have also compiled some words which are relevant to the question asked and gave scores based on how relevant are those to the question asked. The python code calculated vocabulary scores by considering these two aspects. Next for the syntax analysis, the program first breaks the text corpus into individual sentences, and then processes them accordingly. We are using BERT Natural Language Processing Machine learning model which we can fine-tune to perform syntax analysis task. More explanation on this is given in further section. By this way, the application provides feedback report on whether the sentence is grammatically correct or not. Lastly, the application also provides the unique word count, which is done by using a python tokenizer and set-data-structure. This will complete the overall feedback on spoken English of the user’s answer. We have used DevOps tools to automate the entire process. It will help in quickly deploying the features through test, build checks and monitoring, it keeps the production cycle in healthy state.

The CI/CD pipeline for this project is built using the following tools:

1. Development: PyCharm, Django web framework(python), ML tools: pytorch, tensorflow, speech_recogniser, BERT NLP model.
2. Version control: Git, Git LFS, Github
3. Testing: Django-test framework (similar to Pytest)
4. Integration: Jenkins
5. Delivery: Docker, Docker hub, Jenkins
6. Deployment: Rundeck
7. Monitoring: Elastic Stack with filebeat

Source code link: https://github.com/akshaymg99/SPE_Speech_Evaluator.git

Docker-hub link: <https://hub.docker.com/repository/docker/akshaymg99/speech-spe>

System Configuration:

For this project the system configuration used for development/deployment are as follows:

System Details:

Model: HP-Omen

CPU: i7 9th (9750) generation CPU @ 2.6Ghz, Turbo 4.5Ghz, 6 Physical, 12 Logical cores

GPU: NVidia GTX 1650 (4GB Dedicated Memory)

RAM: 16GB DDR4, 2667 Mhz

Disk: SSD (30 GB Allocated to Ubuntu on dual boot partition)

OS Details:

Ubuntu 18.04 LTS

Kernel version: 5.4.0.72

Root Disk space: 30GB

Development tools details:

PyCharm IDE for Django-Python web development, version: 2020.3.3

Python version: 3.6.9

Django version: 3.2

ML Python library dependencies:

Pandas =0.24.2

Pydub =0.25.1

Numpy =1.16.2

Keras =2.3.1

SpeechRecognition =3.7.1

Torch =1.1.0

pytorch_pretrained_bert =0.6.2

tensorflow =1.14.0

DevOps tools:

Git version: 2.17.1 , git LFS version: 2.13.3

Jenkins version: 2.277.4

Rundeck version: 3.3.10

Docker version: 20.10.6

Elastic search version: 7.11.1

Logstash version: 7.11.1

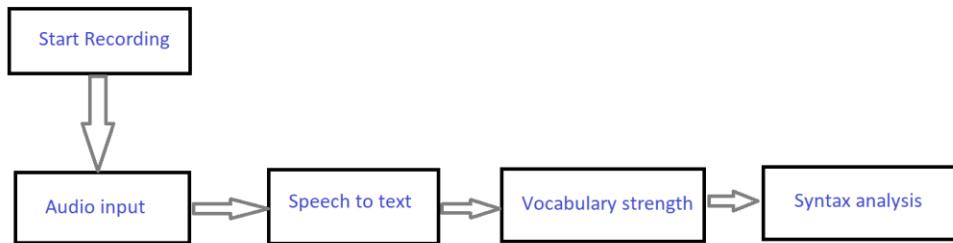
Kibana version: 7.11.1

Design and Development

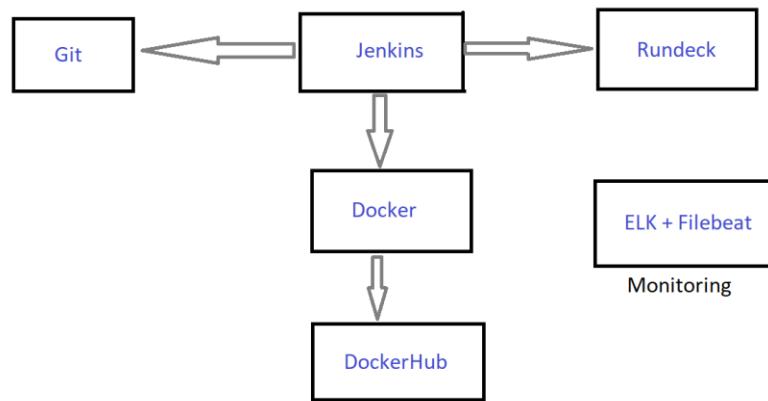
Design Architecture / Diagram

The following Figure shows the architectural diagram of our project, sub-divided into DevOps CI/CD Pipeline and Speech Analysis program.

Design Diagram of Speech Analysis Program:



Design Diagram of Development through DevOps CI/CD Pipeline:



Web application Design

Front-end Design

For designing the front-end webpages for the application, we had used HTML, CSS and JavaScript for setting events such as timer and button text changes on clicking it. Django gives the advantage of setting the base template and extending it to other pages.

First, we had designed a welcome page, where it shows the information on the features it provides (speech to text, vocab strength calculation and syntax checking). It provides a button which on clicking takes the user to Recording page, where the user can speak his answer to the question displayed. The welcome page is as shown below in the Figure [1].

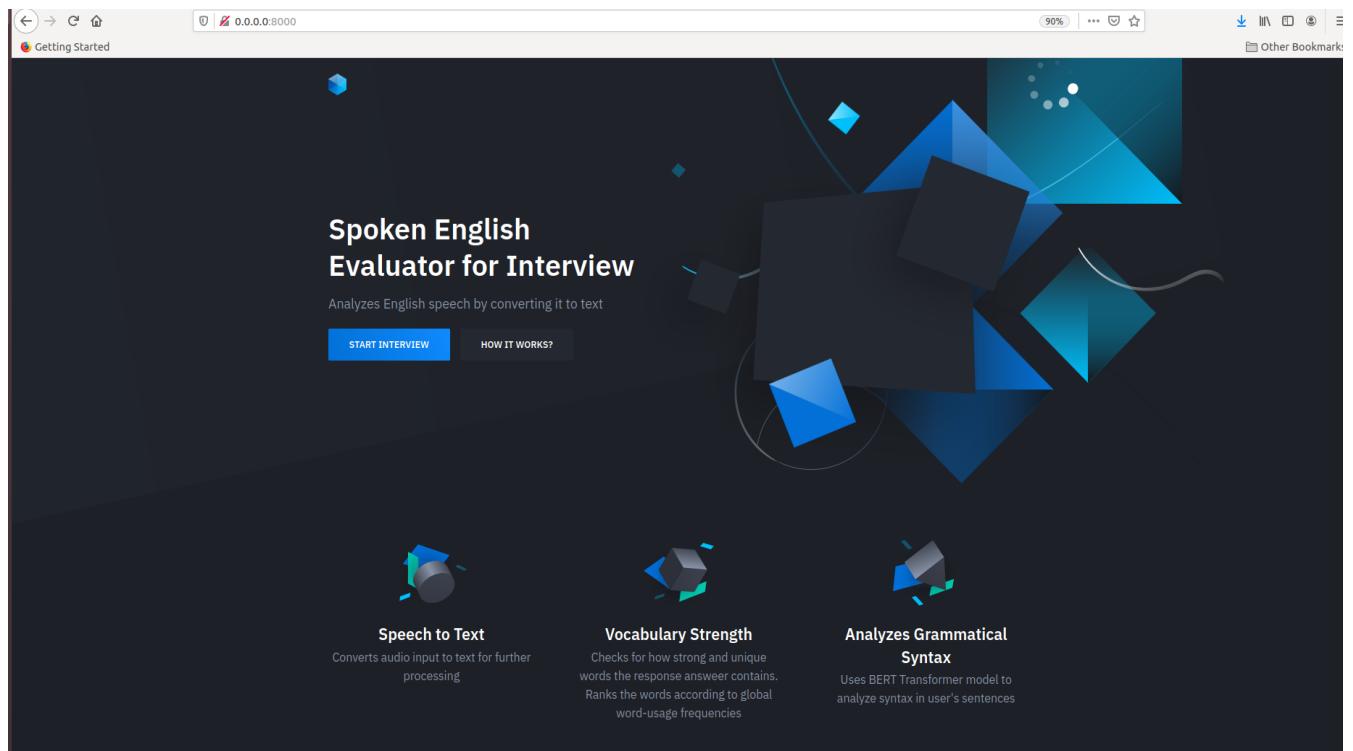


Figure [1] Home page of the web application

After clicking on “Start Interview” button, it takes the user to Record page where to question is displayed, and user has to click on “start recording” button to start the voice recording function. Figure [2] shows the Record web page before clicking the record button.

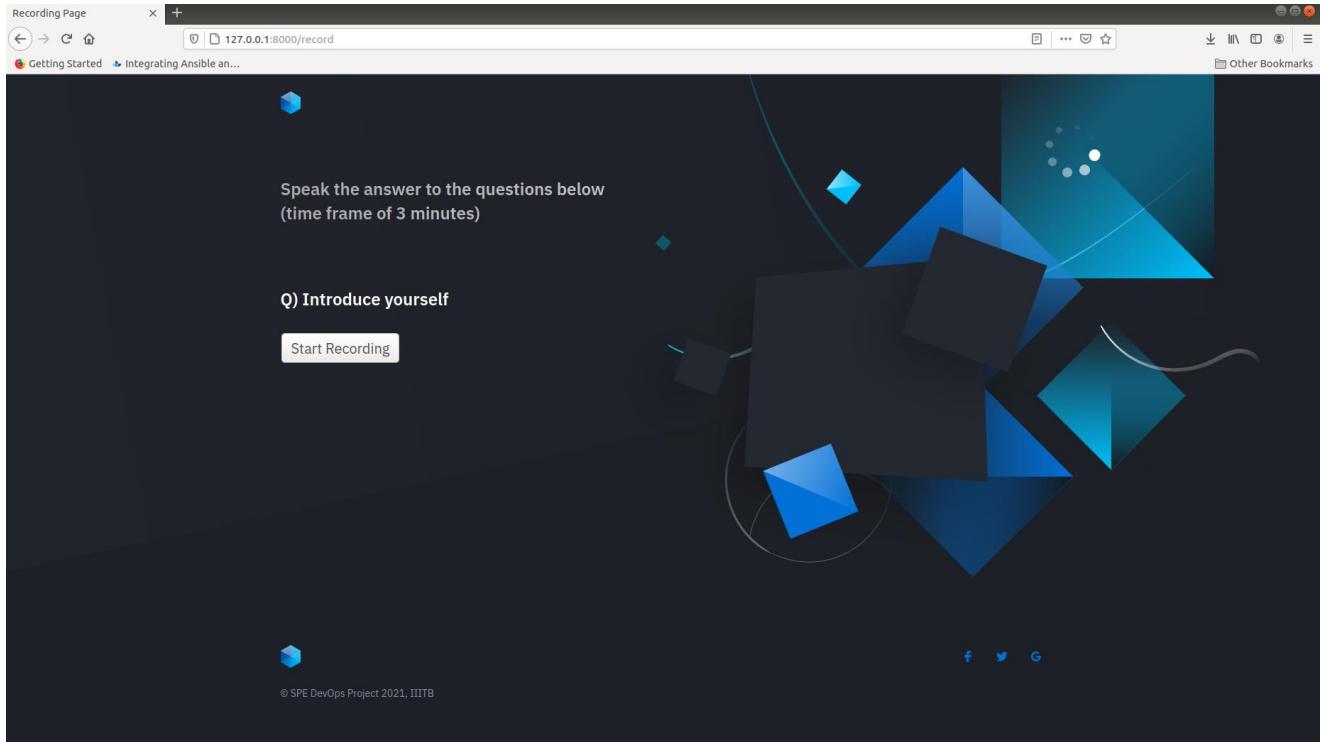


Figure [2] Record page before clicking the “start record” button (timer not started)

Next, when the user clicks the “Start record” button, the application calls the Speech Analyzer function using Django Backend. It also starts a timer which is displayed below the recording button. We have set the timer for question answering as 1 minute. So the application records the audio for the period of a minute. After the timer runs out, it displays “TIME UP” message. This functionality is programmed by using JavaScript. The code for it is shown below in Figure [3]. The webpage after recording is started is shown in Figure [4].

```

<script>
    function startTimer() {
        document.getElementById("mybtn").value="Recording...";
        var timeleft = 60000;
        var setTimer = setInterval(function() {

            var minutes = Math.floor((timeleft % (1000 * 60 * 60)) / (1000 * 60));
            var seconds = Math.floor((timeleft % (1000 * 60)) / 1000);

            document.getElementById("mins").innerHTML = minutes + "m "
            document.getElementById("secs").innerHTML = seconds + "s "

            if (timeleft < 0) {
                clearInterval(setTimer);
                document.getElementById("mins").innerHTML = ""
                document.getElementById("secs").innerHTML = ""
                document.getElementById("end").innerHTML = "TIME UP!!";
            }
            timeleft = timeleft - 1000;
        }, 1000);
    }

</script>

```

Figure [3] Timer function code in JavaScript

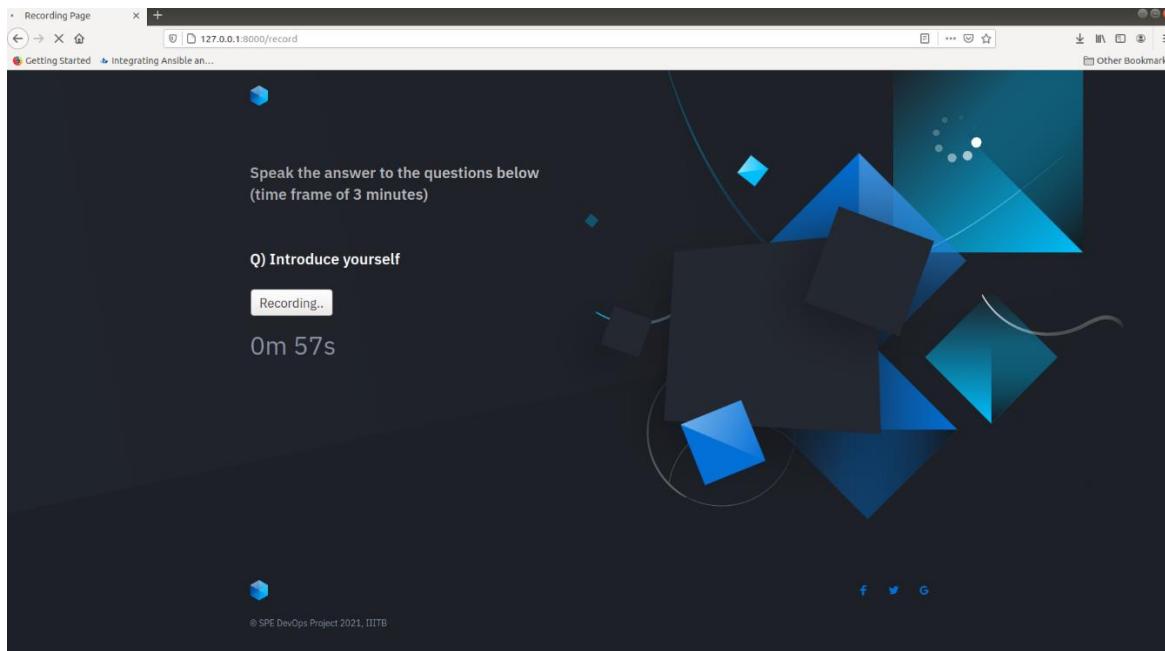


Figure [4] Recording webpage after the recording is started (Timer started)

We have also used JavaScript to change the text on recording button from “Start Recording” to “Recording..” to indicate to the user that recording is being done.

After the Analysis is complete by speech Analyzer function, it returns the output in the form of a list [sentence analysis, vocab-strength, unique words] to the Django view function in backend. Django takes this output and renders it in Result webpage in the form of html tables, which is dynamically filled and size adjusted as per the size of the output. Figure [5] shows an example output displayed after the analysis is carried out.

Report on Interview performance (Speech)	
Sentences	Analysis
[CLS] i did my undergraduate degree in electronics at hubli [SEP]	Correct
[CLS] what masters admission in bangalore [SEP]	Wrong
[CLS] and about time management [SEP]	Correct
[CLS] in the field of computers [SEP]	Correct
[CLS] and meet the requirements of the job [SEP]	Correct
[CLS] and have any certification and accomplishments in my profile [SEP]	Wrong
[CLS] pencil project and thought [SEP]	Wrong
[CLS] production and technical knowledge to face any difficulties and counters [SEP]	Wrong
[CLS] machine learning model [SEP]	Correct
[CLS] expectations of the company [SEP]	Correct

Vocabulary strength is based on how powerful/impactful words you use

Vocabulary strength	310.14	Above average. You have good understanding of english words. Try fine-tuning the words and aim at highest
---------------------	--------	---

Unique words used: 27

Figure [5] Result webpage after analysis is complete

Back-end Design

Django Framework renders webpages via view functions, and using “render” function which takes input parameters as request and the resultant webpage. In this web-application, the welcome page is rendered first whenever the application starts. Then using “URL” functionality, we are directing it to record webpage as shown in below Figure [6].

```
<h1 class="hero-title mt-0">Spoken English Evaluator for Interview</h1>
<p class="hero-paragraph">Analyzes English speech by converting it to text</p>
<div class="hero-cta">
  <a class="button button-primary" onclick="{% url 'record' %}">Start Interview</a>
  <a class="button" href="#">How it Works?</a>
</div>
```

Figure [6] Using URL method to trigger page rendering with onclick JS function

Next, to trigger execution of “Analyzer.py” which does all the speech analysis logic, we are using Django’s view functions. When the “Start Recording” button is clicked, we are using POST method of form to call the Analyzer.py program as shown below in Figure [6] and [7].

```
<h5 class="hero-cta">Q) Introduce yourself</h5><br>

<form method="post">
  {% csrf_token %}
  <input onclick="startTimer()" type="submit" value="Start Recording" name="run_analyzer" id="mybtn" />
  <!--
  <button id="mybtn" value="Start Recording" type="submit" name="run_analyzer" onclick="startTimer()"/>
  -->
</form>
```

Figure [6] Form POST method to trigger Analyzer program

```
17  def record(request):
18      if request.method == 'POST' and 'run_analyzer' in request.POST:
19          res = analyze()
20
21          if res is None:
22              logger.error("Didnt receive result analysis")
23          else:
24              logger.error("Analysis successful")
25
26          table_1_answers = []
27          table_1_sentences = []
28          for each in res[0]:
29              table_1_answers.append(res[0][each])
30              table_1_sentences.append(each)
31
32          table_1 = []
33          for i in range(len(table_1_sentences)):
34              temp = {}
35              temp['Sentence'] = table_1_sentences[i]
36              temp['Analysis'] = table_1_answers[i]
37              table_1.append(temp)
38
39          if res[1] >= 500:
40              vocab_analysis = "Excellent! Your knowledge of english vocabulary is impressive"
41          elif res[1] < 500 and res[1] >= 300:
42              vocab_analysis = "Above average. You have good understanding of english words. Try fine-tuning the words and aim at highest"
43          elif res[1] < 300 and res[1] >= 150:
44              vocab_analysis = "Average. Your knowledge of english vocabulary words is average. There is room for improvement"
45          elif res[1] < 150 and res[1] >= 75:
46              vocab_analysis = "Below average. Your vocabulary is sufficient for casual conversation, but not good for professional use"
47          elif res[1] < 75 and res[1] >= 25:
48              vocab_analysis = "Poor. You have rudimentary proficiency in english, and need to work hard"
49          elif res[1] < 25:
50              vocab_analysis = "You hardly know english vocabulary. Learn it before proceeding further"
51          else:
52              vocab_analysis = "Data not sufficient"
53
54
55          data = { 'table_1':table_1, 'vocab_strength':res[1], 'vocab_analysis':vocab_analysis, 'unique_words':res[2] }
56
57      return render(request, 'result.html', data )
```

Figure [7] View function to execute “Analyzer.py” program when POST method is received

The Django framework structure has files such as settings.py, urls.py, views.py which are vital on constructing the backend logic of a web-application. We shall explain them in brief. Settings.py contains important configuration instructions for the application such as directory path information, allowed hosts info, installed apps, template and static files paths, database information (if any), language settings, logger setups etc. “urls.py” contain webpage urls to which the Django can redirect when called. Its shown in Figure [8] below.

```

1  from django.urls import path
2  from . import views
3
4
5  urlpatterns = [
6      path('', views.index, name='index'),
7      path('record', views.record, name='record'),
8      path('result', views.result, name='result')
9  ]

```

Figure [8] urls.py file which contains webpage url info

“Views.py” contains the backend logic which we can implement on each web-page elements. Here we have used this to execute our Speech Analyzer function when recording button is clicked as shown in Figure [7]. In Result page, we are constructing a table dynamically according to the size of output sentences. This is done with for loop in Django as shown in Figure [9].

```

<div class="container">
    <h4> Report on Interview performance (Speech) </h4>
    <table border="2" cellpadding="5" cellspacing="5">
        <thead>
            <tr>
                <th>Sentences</th>
                <th>Analysis</th>
            </tr>
        </thead>
        <tbody>
            {% for i in table_1 %}
                <tr>
                    <td>{{i.Sentence}}</td>
                    <td>{{i.Analysis}}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>

    <h5>Vocabulary strength is based on how powerful/impactful words you use</h5>
    <table border="2" cellpadding="5" cellspacing="5">
        <tbody>
            <tr>
                <td>Vocabulary strength</td>
                <td>{{ vocab_strength }}</td>
                <td>{{ vocab_analysis }}</td>
            </tr>
        </tbody>
    </table>
    <h5>Unique words used: {{ unique_words }}</h5>
</div>
</section>
</main>

```

Figure [9] Populating Dynamic table with output

Speech Analyzer Program Design

In this section, we discuss about how speech analyzer works with screenshots of relevant code. We made use of ML models to perform these analysis task. This concept comes under the field of Natural Language Processing (NLP) of ML. Here we make use of a NLP model called ‘BERT’ which is a transformer model developed on enormous corpus of text. We take this model and fine-tune it our syntax analysis task.

Speech Record and text Conversion

We are doing the audio recording using “speech_recognition” module, which will use the system’s microphone to listen to audio signals, then for converting to that speech to text we are using “recognize_google” function provided by speech_recognition module. We have set the timer to one minute for it to record. The code for this is shown in Figure [10]

```

21     r = sr.Recognizer()
22     sentences = []
23     inp_corpus = ""
24     print("Starting voice recording")
25
26     while time.time() < t_end:
27         try:
28             with sr.Microphone() as source2:
29                 r.adjust_for_ambient_noise(source2, duration=1)
30                 audio2 = r.listen(source2)
31                 MyText = r.recognize_google(audio2)
32                 MyText = MyText.lower()
33                 inp_corpus += MyText
34                 sentences.append(MyText)
35
36         except sr.RequestError as e:
37             print("Could not request results; {0}".format(e))
38
39         except sr.UnknownValueError:
40             print("unknown error occurred")

```

Figure [10] Audio recording and text conversion code

Vocabulary Strength Calculation

Upon the text which speech-to-text converter generated, and after performing some filtering operations such as bad characters removal, we are performing vocabulary strength analysis, which considers word frequencies from global English dictionary <http://norvig.com/mayzner.html>. This score is calculated by taking inverse of their frequency, thereby giving reasonably less score for most commonly used words. We have also compiled some words which are relevant to the question asked and gave scores based on how relevant are

those to the question asked. The python code which makes use of dict datastructure, calculates vocabulary scores by considering these two aspects is shown in Figure [11].

```

43     ## Pre-processing
44     bad_chars = [';', ':', '!', "*", ",", "."]
45
46     inp_corpus = ''.join(i for i in inp_corpus if not i in bad_chars)
47     inp_corpus = inp_corpus.replace('&', 'and')
48     inp_corpus = inp_corpus.replace('@', 'at')
49     inp_corpus = inp_corpus.lower()
50     corpus_list = list(inp_corpus.split(' '))
51
52     ## setting weightage parameters for scores
53     alpha = 0.4
54     beta = 0.6
55
56     ## textfile containing words and their usage frequency
57     # script_dir = Path(__file__).parent
58     f = open(DEPENDENCIES_DICT_DIR, "r")
59     dictionary = dict()
60     list_lines = f.readlines()
61
62     ## creating word-frequency dictionary out of words-text file
63     rank = 1
64     for line in list_lines:
65         li = list(line.split(' '))
66         word, frequency = li[0], li[2]
67         dictionary[word] = rank
68         rank += 1
69
70     for each in dictionary.keys():
71         temp = (dictionary[each])
72         scaled = (temp - 1) * 99 / (len(dictionary) - 1)
73         scaled += 1
74         dictionary[each] = scaled
75
76     visited = []
77     score_1 = 0
78     unique = 0
79     for word in corpus_list:
80         if word in dictionary.keys():
81             if word not in visited:
82                 score_1 = score_1 + dictionary[word]
83                 visited.append(word)
84                 unique += 1
85
86     ## calculating score_2 with tag ranks
87     score_2 = 0
88     db = pd.read_csv(Q1_CSV)
89     db_dict = dict(db.values)
90     for word in corpus_list:
91         if word in db_dict.keys():
92             if word not in visited:
93                 score_2 = score_2 + db_dict[word]
94                 visited.append(word)
95                 unique += 1
96
97     strength = (alpha * score_1) + (beta * score_2)
98     strength = float("{:.2f}".format(strength))
99     f.close()

```

Figure [11] Vocabulary strength analysis

Here in this code, “Dependencies_dict_dir” is the path for global word count file. And “Q1.csv” is an excel file in which we had compiled words which are relevant to the question asked. We have set alpha and beta as 0.4 and 0.6, which are weightages given to the score calculated from above two files. The final score is the weighted mean of both the scores, which are calculated from above said files.

Syntax Analysis of Sentences

The text corpus output from speech recorder is broken into individual sentences for syntax analysis on them. To perform syntax analysis, we are using NLP BERT model, which is trained on enormous amount of text data beforehand. We fine-tune it to analyze syntax of each sentence and provide output on whether they are grammatically correct or not. Figure [12] shows the code

```

101     print("Starting syntax analysis")
102
103     device = "cpu"
104     if torch.cuda.is_available():
105         torch.cuda.empty_cache()
106     gc.collect()
107     model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
108     model.load_state_dict(torch.load(BERT_MODEL, map_location='cpu'))
109     tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
110
111     model.eval()
112     sentences = "[CLS] " + sentence + " [SEP]" for sentence in sentences]
113     labels = [0]
114     tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]
115     MAX_LEN = 128
116     predictions = []
117     true_labels = []
118     input_ids = pad_sequences([tokenizer.convert_tokens_to_ids(txt) for txt in tokenized_texts], maxlen=MAX_LEN,
119                               dtype="long", truncating="post", padding="post")
120     input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
121     input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")
122
123     attention_masks = []
124     for seq in input_ids:
125         seq_mask = [float(i > 0) for i in seq]
126         attention_masks.append(seq_mask)
127
128     prediction_inputs = torch.tensor(input_ids)
129     prediction_masks = torch.tensor(attention_masks)
130     prediction_labels = torch.tensor(labels)
131     model.eval()
132
133     with torch.no_grad():
134         logits = model(prediction_inputs.to(device).long(), token_type_ids=None,
135                         attention_mask=prediction_masks.to(device).long())
136         logits = logits.detach().cpu().numpy()
137         predictions.append(logits)
138
139     flat_predictions = [item for sublist in predictions for item in sublist]
140     flat_predictions = np.argmax(flat_predictions, axis=1).flatten()
141     flat_true_labels = [item for sublist in true_labels for item in sublist]
142
143     result = {}
144     for i in range(len(flat_predictions)):
145         if flat_predictions[i] == 1:
146             result[sentences[i]] = 'correct'
147         elif flat_predictions[i] == 0:
148             result[sentences[i]] = 'Wrong'
149

```

Figure [12] Syntax analysis code used for sentences

Here, we will explain the code in brief as follows: For training, sentences are taken from cola public dataset and are inserted with tokens CLS and SEP at beginning and end to make it work for BERT. These will function as sentence separators. We tokenize our sentences by making use of bert tokenizer using a base pre-trained model. To make each sentence length equal, we have set the length of sentence as 128 and padded remaining places with null strings. The main

concept of BERT is attention mask, which will tell on which values you have to focus and which not. The data is split into train and test datasets, and then tensors are constructed from it, which can be used for training with pytorch. We also define our data-loader which helps in loading data in batches, so that computational resource requirement is less. We get weights for various layers when we train this model. Once that is done, we separate weight parameters from bias, gamma, and beta parameters. We filter one group without these values and another with them. We had trained the above model with 4 epochs and used gradient forward pass to calculate the loss function. After this, we save the trained model and use it for syntax prediction on sentence inputs. Here in this project, we had trained the model separately and saved it in a directory to use it when necessary. The model size after training was about 420 MB, so we had to use github LFS storage to put it in our Github repo. Training the model while deploying it is impractical, so we had to train it beforehand, and then only use the saved model in the web-application.

DevOps CI/CD Pipeline

This section explains about the Software development method we had followed while developing this web-application.

Source Code Management

In CI/CD pipeline it is important to maintain versions of our source code, it will help us to keep track and revert back to any previous version when required. We are using Git /GitHub version control system for this project. Since one of the file (BERT model) was > 400MB size, we had to use GitHub LFS (Large File storage) which can track large files and store them in server.

The development of this application has been done incrementally. The commits done can be seen at https://github.com/akshaymg99/SPE_Speech_Evaluator/commits/master .

After developing code in an IDE, to post it to SCM (Git), we have to execute the following commands in our project working directory:

```
git remote add origin <remote repo url> // create connection to remote repo
git init // for initializing local system directory as git directory
git add . or git add /files // for adding files to staging area
git commit -m "message" // to take a snapshot of the current state of directory
git pull <remote repo url> // to synchronize remote repository with local repo
git push u origin <branch> // to push the local repo files to remote repository
```

```

akshay@akshay-OMEN: ~/Desktop/IIITB/SPE_Speech_Evaluator
File Edit View Search Terminal Help
Counting objects: 3, done.
Delta compression using up to 12 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 376 bytes | 376.00 KiB/s, done.
Total 3 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/akshaymg99/SPE_Speech_Evaluator.git
  6c67127..8f14f75 master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   Jenkinsfile

no changes added to commit (use "git add" and/or "git commit -a")
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git Jenkinsfile
git: 'Jenkinsfile' is not a git command. See 'git --help'.
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git add Jenkinsfile
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git commit -m "Added Docker push to pipeline"

```

Other git commands:

```

git revert <commit>      // undoes all changes made in <commit>
git status            // list which files are staged, unstaged & untracked
git log              // display commit history
git checkout b <branch>    // create & checkout a new branch
git merge <branch>        // merge <branch> into current branch

```

We are also using Git LFS for this project to store large model in repository. We have to install it separately in our system as shown below and also have to enable Git LFS pipeline in Jenkins.

```

akshay@akshay-OMEN: ~/Desktop/IIITB/SPE_Speech_Evaluator
File Edit View Search Terminal Help
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git lfs install
Updated git hooks.
Git LFS initialized.
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ 

```

Git LFS pushing the object to its repo:

```

akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git remote -v
origin  https://github.com/akshaymg99/SPE_Speech_Evaluator.git (fetch)
origin  https://github.com/akshaymg99/SPE_Speech_Evaluator.git (push)
akshay@akshay-OMEN:~/Desktop/IIITB/SPE_Speech_Evaluator$ git push origin master

A new release of gh is available: 1.4.0 → v1.9.2
https://github.com/cli/cli/releases/tag/v1.9.2

Uploading LFS objects:  0% (0/1), 87 MB | 416 KB/s

```

Configuring Git LFS in Jenkins Job:

The screenshot shows the Jenkins Pipeline configuration page. The 'Pipeline' tab is selected. In the 'Definition' section, 'Pipeline script from SCM' is chosen. Under 'SCM', 'Git' is selected. The 'Repositories' section shows a repository URL of https://github.com/akshaymg99/SPE_Speech_Evaluator.git and credentials set to 'akshaymg99/***** (github)'. The 'Branches to build' section has a branch specifier of */master. The 'Additional Behaviours' section contains a 'Git LFS pull after checkout' checkbox, which is checked. The 'Script Path' is set to 'Jenkinsfile' and 'Lightweight checkout' is checked. At the bottom, there are 'Save' and 'Apply' buttons.

Jenkins pulling the git LFS object

The screenshot shows the Jenkins Console Output for a pipeline job. The log output is as follows:

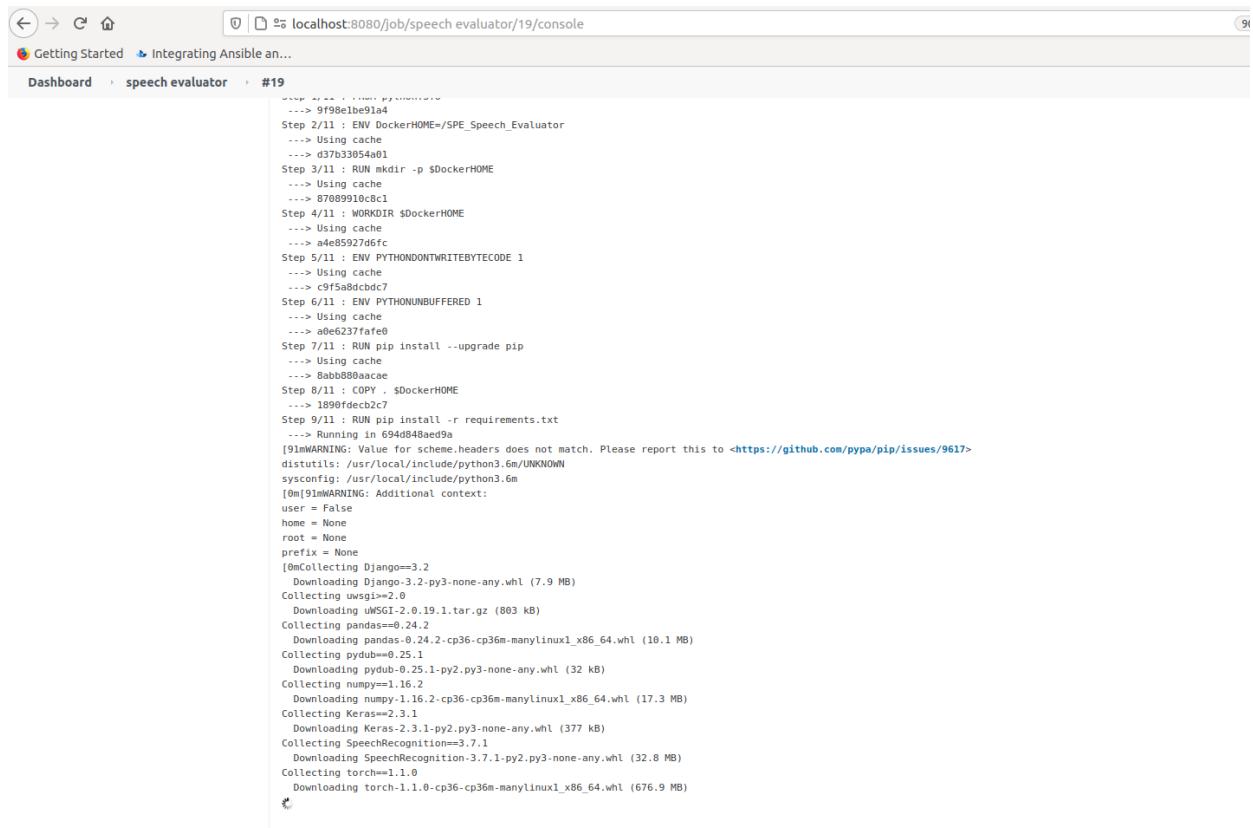
```

Started by user unknown or anonymous
Obtained Jenkinsfile from git https://github.com/akshaymg99/SPE\_Speech\_Evaluator.git
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/speech evaluator
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
using credential github
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/akshaymg99/SPE\_Speech\_Evaluator.git # timeout=10
Fetching upstream changes from https://github.com/akshaymg99/SPE\_Speech\_Evaluator.git
> git --version # timeout=10
> git --version # 'git version 2.17.1'
using GIT_ASKPASS to set credentials github
> git fetch --tags --progress -- https://github.com/akshaymg99/SPE\_Speech\_Evaluator.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision a00c338a6c9c8131090d932507033fab40854e64 (refs/remotes/origin/master)
Enabling Git LFS pull
> git config core.sparsecheckout # timeout=10
> git checkout -f a00c338a6c9c8131090d932507033fab40854e64 # timeout=10
> git config --get remote.origin.url # timeout=10
using GIT_ASKPASS to set credentials github
> git lfs pull origin # timeout=10

```

Build Stage

Build stage consists of installing dependencies and making the application ready to be complied/ installed. We used `pip3 freeze > requirements.txt` command to export the dependency requirements to a text file, which we are using it to install in Docker container later in this project. We are using `pip3 install -r requirements.txt` command in Dockerfile to install all the requirements in docker build stage. Below Figure [13] shows the dependencies getting installed while running Jenkins job.



```

[...]
Step 2/11 : ENV DockerHOME=$PSE_Speech_Evaluator
--> Using cache
--> d37b3d3054a01
Step 3/11 : RUN mkdir -p $DockerHOME
--> Using cache
--> 87089910c8c1
Step 4/11 : WORKDIR $DockerHOME
--> Using cache
--> a4e85927d6fc
Step 5/11 : ENV PYTHONUNWRITEBYTECODE 1
--> Using cache
--> c9f5a8dcdbd7
Step 6/11 : ENV PYTHONUNBUFFERED 1
--> Using cache
--> a0e6237fafe0
Step 7/11 : RUN pip install --upgrade pip
--> Using cache
--> 8abb080aace8
Step 8/11 : COPY . $DockerHOME
--> 1890fddec2c7
Step 9/11 : RUN pip install -r requirements.txt
--> Running in 694dd94baed9
[91mWARNING: Value for scheme.headers does not match. Please report this to <https://github.com/pypa/pip/issues/9617>
distutils: /usr/local/include/python3.6m/UNKNOWN
sysconfig: /usr/local/include/python3.6m
[0m[91mWARNING: Additional context:
user = False
home = None
root = None
prefix = None
[0m[91mCollecting Django==3.2
    Downloading Django-3.2-py3-none-any.whl (7.9 MB)
Collecting uwsgi==2.0
    Downloading uWSGI-2.0.19.1.tar.gz (803 kB)
Collecting pandas==0.24.2
    Downloading pandas-0.24.2-cp36-cp36m-manylinux1_x86_64.whl (10.1 MB)
Collecting pydub==0.25.1
    Downloading pydub-0.25.1-py3-none-any.whl (32 kB)
Collecting numpy==1.16.2
    Downloading numpy-1.16.2-cp36-cp36m-manylinux1_x86_64.whl (17.3 MB)
Collecting Keras==2.3.1
    Downloading Keras-2.3.1-py2.py3-none-any.whl (377 kB)
Collecting SpeechRecognition==3.7.1
    Downloading SpeechRecognition-3.7.1-py2.py3-none-any.whl (32.8 MB)
Collecting torch==1.1.0
    Downloading torch-1.1.0-cp36-cp36m-manylinux1_x86_64.whl (676.9 MB)
[0m

```

Figure [13] Dependency installation through requirements file

Testing stage

Testing is required to find any bugs or errors during application run, so developers write unit test cases to check the operations of the program. We are using Django's inbuilt test framework for testing the web-application. We are performing 3 types of tests for our application – webpage retrieval tests, csrf token check which helps in blocking unauthorized access, and lastly a logic test in which we are providing text-corpus as input and checking how will it output the result webpage. Figure [14] shows the test code which we consist of the test types we talked earlier.

```

1  from django.test import Client
2  from django.test import TestCase
3  import unittest
4
5  class TestApp(TestCase):
6      ## Checking if all webpages are retrieved successfully (status code-> 200 should specify the success)
7      def test_home_page(self):
8          self.c = Client()
9          csrf_client = Client(enforce_csrf_checks=True) ## enforcing CSRF Checks
10         self.response = self.c.get('') ## home page retrieval
11         print(self.response.status_code) ## output = 200 specifies the page was retrieved successfully
12         self.assertEqual(self.response.status_code, 200, msg='Home page retrieved successfully')
13
14     def test_record_page(self):
15         self.c = Client()
16         csrf_client = Client(enforce_csrf_checks=True) ## enforcing CSRF Checks
17         self.response = self.c.get('/record') ## record.html page retrieval
18         print(self.response.status_code) ## output = 200 specifies the page was retrieved successfully
19         self.assertEqual(self.response.status_code, 200, msg='Recording page retrieved successfully')
20
21     def test_result_page(self):
22         self.c = Client()
23         csrf_client = Client(enforce_csrf_checks=True) ## enforcing CSRF Checks
24         self.response = self.c.get('/result') ## result.html page retrieval
25         print(self.response.status_code) ## output = 200 specifies the page was retrieved successfully
26         self.assertEqual(self.response.status_code, 200, msg='Result page retrieved successfully')
27         print(self.response.content) ## checking content of the webpage
28
29     def test_Analysis_logic(self):
30         self.c = Client()
31         self.table_1 = []
32         self.temp = {}
33         self.temp['Sentence'] = "Hi, My name is Akshay, I am pursing Masters from IIITB"
34         self.temp['Analysis'] = "Correct"
35         self.table_1.append(self.temp)
36         self.res = []
37         self.res.append(8)
38         self.res.append(340)
39         self.res.append(50)
40         self.vocab_analysis = "Average. Your knowledge of english vocabulary words is average. There is room for improvement"
41         self.data = {'table_1': self.table_1, 'vocab_strength': self.res[1], 'vocab_analysis': self.vocab_analysis, 'unique_words': self.res[2]}
42         self.response = self.c.post('/record', self.data)
43         print(self.response.status_code)

```

Figure [14] Testing code for the web-application

Here, we are performing CSRF check in each webpage and later a logic check on result page output. Figure [15] shows the output of test case running. Status code 200 means successful page retrieval and the html content is the output of result page on providing text corpus input to it.

```

Creating test database for alias 'default'...
System check identified no issues (0 silenced).
200
200
200
200
b"\n!DOCTYPE html\n<html lang="en" class="no-js">\n<head>\n    <meta charset="utf-8">\n    <meta http-equiv="X-UA-Compatible" content="IE=edge">\n    <meta name="viewport" content="width=device-width, initial-scale=1">\n    <title>Recording Page</title>\n    <link href="https://fonts.googleapis.com/css?family=IBM+Plex+Sans:400,600" rel="stylesheet">\n    <link rel="stylesheet" href="static/dist/css/style.css" >\n    <script src="https://unpkg.com/scrollreveal@4.0.0/dist/scrollreveal.min.js"></script>\n    <style>\n        table{\n            border: 2px solid wheat;\n            padding: 15px;\n        }\n        td {\n            text-align: center;\n        }\n        th {\n            color: white;\n            text-align: center;\n        }\n        background-color: #00bab6;\n    </style>\n</head>\n<body class="is-boxed has-animations">\n    <div class="body-wrap">\n        <header class="site-header">\n            <div class="container">\n                <div class="site-header-inner">\n                    <div class="brand header-brand">\n                        -----\nRan 4 tests in 0.013s\nOK\nDestroying test database for alias 'default'..."

```

Figure [15] Testing output

Here, it shows the summary as Ran 4 test cases and there were no failures. To run the tests, we are using “python manage.py test” command in Django application’s root directory.

Continuous Integration

Continuous Integration (CI) refers to the process of integrating code changes with the existing code as and when it is written. We are using Jenkins tool for Integration all the stages of DevOps. It automates from git pull whenever there is new code push to deployment of application. After installing Jenkins, we also need to add credentials of git and Dockerhub which we need to access later. Figure [16] shows adding of git credentials in Jenkins.

Domain
Global credentials (unrestricted)

Kind
Username with password

Scope
Global (Jenkins, nodes, items, all child items, etc)

Username
akshaymg99

Password

ID
github

Description
github

Add Cancel

Figure [16] Adding Git credentials into Jenkins

Jenkins pipeline

A Jenkins pipeline gives us a graphical view of the various steps of a CI/CD pipeline. It allows us to link different Jenkins jobs and allows us to check their progress during execution. Here in this project, the first job consists of 4 stages: SCM checkout, SCM git pull, Docker build and Dockerhub push. After successfully completing these stages, Jenkins triggers another job, which is a Rundeck job, it pulls the latest docker image from hub, and deploys it in system, here the deployment system is our local system itself. This way Jenkins handles all the tasks automatically from git pull to triggering deployment task. Below Figure [17] shows the Jenkinsfile for our project. We can see different stages named for their operations. Figure [18] shows the Visualization of pipeline stages, which helps in developers understanding how each stage is being executed and what amount of time does each stage take.

```

1  pipeline{
2      agent any
3
4      stages{
5
6          stage('SCM git pull'){
7              steps{
8                  git credentialsId: 'github',
9                  url: 'https://github.com/akshaymg99/SPE_Speech_Evaluator.git'
10             }
11         }
12
13         stage('Docker Build'){
14             steps{
15                 sh "docker build . -t akshaymg99/speech-spe:latest "
16             }
17         }
18
19         stage('DockerHub Push'){
20             steps{
21                 withCredentials([string(credentialsId: 'docker-hub', variable: 'dockerHubPwd')]) {
22                     sh "docker login -u akshaymg99 -p ${dockerHubPwd}"
23                 }
24                 sh "docker push akshaymg99/speech-spe:latest "
25             }
26
27         }
28
29     }
30 }
31
32 }
```

Figure [17] Jenkins pipeline linking various stages of development

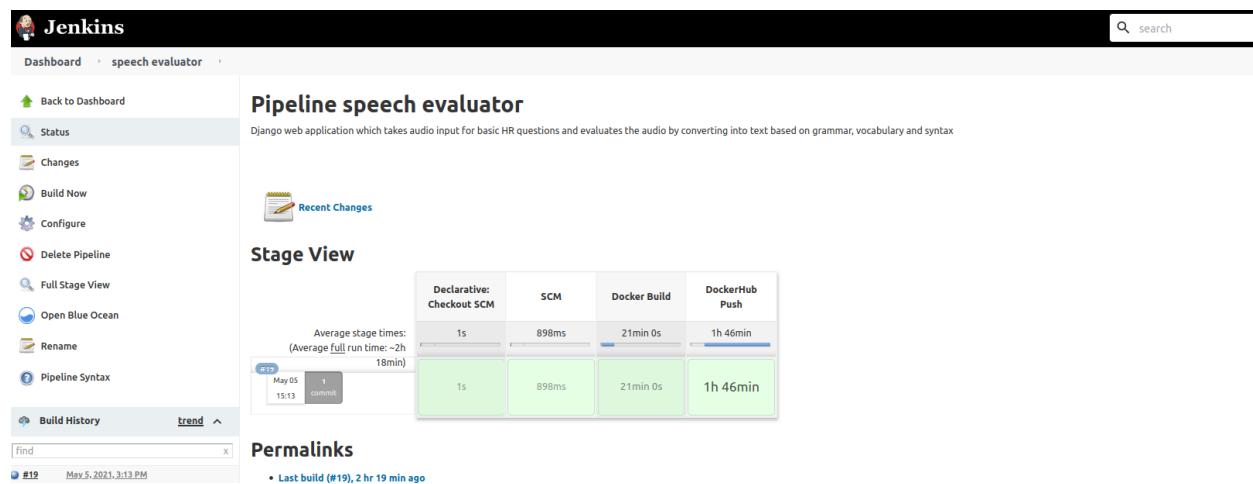


Figure [18] Jenkins pipeline visualization

We are also using Jenkins to trigger the deployment task which is a Rundeck Job. We will explain this Rundeck job later in detail in Deployment part. After configuring a Rundeck Job, it gives a unique UUID key, which we should use in Jenkins to link that task with Jenkins. After linking, we need to setup a build trigger which after successful completion of previous job (in which we have pushed the docker image to hub) it starts the Rundeck job. Figure [19] shows configuring Jenkins with Rundeck job with UUID key.

The screenshot shows the Jenkins configuration interface for a job. The top navigation bar includes tabs for General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions. The General tab is selected.

General Tab:

- Description: "This runs after the build of speech-evaluator is successful
It runs the rundeck job which deploys the pulled container (in localhost machine)"
- [Plain text] [Preview]
- Checkboxes for various Jenkins features like Commit agent's Docker container, Define a Docker template, Discard old builds, GitHub project, etc.
- An "Advanced..." button.

Source Code Management Tab:

- None (radio button selected)
- Git (radio button)

Build Triggers Tab:

- Build after other projects are built (checkbox selected)
- Save and Apply buttons.

Post-build Actions Tab:

Rundeck Action Configuration:

- Rundeck Instance: RundeckProd
- Job user (optional): (empty field)
- User password (optional): Concealed (password field)
- Token (optional): Concealed (password field)
- Job Identifier: 67992a9e-dd08-405f-beaa-88e66b4547e9
- Job options (optional): (large text area)
- Node filters (optional): (large text area)
- SCM Tag (optional): (empty field)
- Wait for Rundeck job to finish? (checkbox): (unchecked)
- Include Rundeck job output? (NOTE: requires Wait for Rundeck job to finish) (checkbox): (unchecked)
- Tail Logging? (NOTE: requires Wait for Rundeck job to finish & Include Rundeck job output) (checkbox): (unchecked)
- Should fail the build? (checkbox): (unchecked)

Save and Apply buttons are located at the bottom of the Rundeck configuration section.

Figure [19] Configuring Jenkins with Rundeck UUID key to trigger it

We can also see that the successful completion of “speech_evaluator” job in Jenkins which consists of stages git pull to docker hub push triggers “deploy speech” job of Rundeck in Jenkins. Figure [20] shows successful trigger of Rundeck deploy job in console.

```
[Pipeline] // stage
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Triggering a new build of deploy spech-spe #1
Finished: SUCCESS
```

Figure [20] Jenkins triggering Rundeck deploy job

Figure [21] shows “deploy speech” running successfully in which it had executed the Rundeck job. More on Rundeck deployment will be explained in Deployment part of this report.

```
Started by upstream project "speech evaluator" build number 19
originally caused by:
Started by user unknown or anonymous
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/deploy spech-spe
Instance 'RundeckProd' with rundeck user 'admin': Notifying Rundeck...
Looking for jobID : 67992a9e-d008-405f-beaa-88e66b4547e9
Notification succeeded ! Execution #32, at http://localhost:4440/project/speech-spe-deploy/execution/show/32 (status : running)
Finished: SUCCESS
```

Figure [21] Jenkins executing Rundeck job

Following Figures [22], [23], [24], [25], [26], [27] shows Jenkins console output for running “speech evaluator” task (which has stages from git pull to docker hub push).

```
[Pipeline] checkout
The recommended git tool is: git
using credential github
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/speech evaluator/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/akshayng99/SPE_Speech_Evaluator.git # timeout=10
Commit message: "Removed git lfs script in Jenkinsfile"
> git rev-parse --verify HEAD # timeout=10
Resetting working tree
> git reset --hard # timeout=10
> git clean -fd # timeout=10
Fetching upstream changes from https://github.com/akshayng99/SPE_Speech_Evaluator.git
> git --version # timeout=10
> git --version # git version 2.17.1'
using GIT_ASKPASS to set credentials github
> git fetch --tags --progress .. https://github.com/akshayng99/SPE_Speech_Evaluator.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 0826493aaef2a8040ce86ae8e30199812ce8e0f (refs/remotes/origin/master)
Enabling Git LFS pull
> git config core.sparsecheckout # timeout=10
> git config sparse.checkouts=@{0}ce86ae8e30199812ce8e0f # timeout=60
using GIT_ASKPASS to set credentials github
> git lfs pull origin # timeout=60
Commit message: "Removed git lfs script in Jenkinsfile"
[Pipeline]
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] {
[Pipeline] git
The recommended git tool is: git
using credential github
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/speech evaluator/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/akshayng99/SPE_Speech_Evaluator.git # timeout=10
Fetching upstream changes from https://github.com/akshayng99/SPE_Speech_Evaluator.git
> git --version # timeout=10
> git --version # git version 2.17.1'
using GIT_ASKPASS to set credentials github
> git fetch --tags --progress .. https://github.com/akshayng99/SPE_Speech_Evaluator.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git config core.sparsecheckout # timeout=10
> git checkout -f 0826493aaef2a8040ce86ae8e30199812ce8e0f (refs/remotes/origin/master)
> git branch -a -v --no-abbrev # timeout=10
> git branch -D master # timeout=10
> git checkout -b master 0826493aaef2a8040ce86ae8e30199812ce8e0f # timeout=10
Commit message: "Removed git lfs script in Jenkinsfile"
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
```

Figure [22] git checkout stage in Jenkins, note that it also uses git LFS

```
Dashboard > speech evaluator > #19
Collecting numpy<1.21,>=1.19.2
  Downloading numpy-1.19.2-cp36-cp36m-manylinux1_x86_64.whl (7.0 MB)
Collecting Keras==2.3.1
  Downloading Keras-2.3.1-py3-none-any.whl (377 kB)
Collecting SpeechRecognition==3.7.1
  Downloading SpeechRecognition-3.7.1-py2.py3-none-any.whl (32.8 MB)
Collecting torch==1.1.0
  Downloading torch-1.1.0-cp36-cp36m-manylinux1_x86_64.whl (676.9 MB)
Collecting pytorch_pretrained_bert==0.6.2
  Downloading pytorch_pretrained_bert-0.6.2-py3-none-any.whl (123 kB)
Collecting sqlparse==0.2.2
  Downloading sqlparse-0.4.1-py3-none-any.whl (42 kB)
Collecting pytz
  Downloading pytz-2021.1-py2.py3-none-any.whl (510 kB)
Collecting asgiref==4.x>=3.2
  Downloading asgiref-3.3.4-py3-none-any.whl (22 kB)
Collecting python-dateutil>2.5.0
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
Collecting six==1.9.0
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting keras_applications>=1.0.6
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
Collecting pyaml
  Downloading PyYAML-5.4.1-cp36-cp36m-manylinux1_x86_64.whl (640 kB)
Collecting h5py
  Downloading h5py-3.1.0-cp36-cp36m-manylinux1_x86_64.whl (4.0 MB)
Collecting keras_preprocessing==1.0.5
  Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
Collecting scipy==0.14
  Downloading scipy-1.5.4-cp36-cp36m-manylinux1_x86_64.whl (25.9 MB)
Collecting tqdm
  Downloading tqdm-4.60.0-py2.py3-none-any.whl (75 kB)
Collecting requests
  Downloading requests-2.25.1-py2.py3-none-any.whl (61 kB)
Collecting boto3
  Downloading boto3-1.17.66.tar.gz (98 kB)
Collecting regex
  Downloading regex-2021.4.4-cp36-cp36m-manylinux2014_x86_64.whl (722 kB)
Collecting typing_extensions
  Downloading typing_extensions-3.10.0.0-py3-none-any.whl (26 kB)
Collecting botocore<1.21.0,>=1.20.66
  Downloading botocore-1.20.66-py2.py3-none-any.whl (7.5 MB)
Collecting jmespath<1.0.0,>=0.7.1
  Downloading jmespath-0.10.0-py2.py3-none-any.whl (24 kB)
Collecting s3transfer<0.5.0,>=0.4.0
  Downloading s3transfer-0.4.2-py2.py3-none-any.whl (79 kB)
Collecting urllib3<1.27,>=1.25.4
  Downloading urllib3-1.26.4-py2.py3-none-any.whl (153 kB)
Collecting cached_property
  Downloading cached_property-1.5.2-py2.py3-none-any.whl (7.6 kB)
Collecting charset<5,>=3.0.2
  Downloading charset-4.0.0-py2.py3-none-any.whl (178 kB)
Collecting certifi=>2017.4.17
  Downloading certifi-2020.12.5-py2.py3-none-any.whl (147 kB)
Collecting idna<3,>=2.5
```

Figure [23] Installing requirements / dependencies from requirements.txt file

```
Dashboard > speech evaluator > #25
Get: 0 http://security.ubuntu.com/ubuntu bionic-security InRelease [100 kB]
Get: 4 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get: 5 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [2183 kB]
Get: 6 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [2150 kB]
Get: 7 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [2582 kB]
Get: 8 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [452 kB]
Get: 9 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [423 kB]
Get: 10 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [1411 kB]
Fetched 9454 kB in 1min 21s (117 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
dconf-gsettings-backend dconf-service dmetup fontconfig fontconfig-config
fonts-dejavu-core glib-networking glib-networking-common
glib-networking-services gsettings-desktop-schemas i965-v4-driver kmod
krb5-locales libaac3-0 libasound2 libasound2-data
libasound2-plugins libas3-0 libasncn3-0 libavc1394-0 libavcodec57
libavdevice57 libavfilter6 libavformat57 libavresample3 libavutil55
libddpplus0 libbluray2 libbs2b0 libbsd0 libcaca0 libcairo2 libcap2
libcdio-cdda2 libcdio-paranoia2 libcdio17 libchromaprint1 libcroco3
libcryptsetup12 libcryptstalid3 libdatriel libdc1394-22 libdconf1
libdevmapper1.02.1 libdrm-amdgpu1 libdrm-common libdrm-intel1
libdrm-nouveau2 libdrm-radeon1 libdrm2 libedit2 libegl-mesa0 libegl1 libelf1
libfftw3-double3 libfftw3-single3 libflac8 libflitel libfontconfig1
libfreetype2 libfrbirid0 libgbm1 libgd-pixbuf2.0-0 libgd-pixbuf2.0-bin
libgd-pixbuf2.0-common libgll1-mesa-dri libglapi-mesa libgvnd0
libglx-mesa0 libglx0 libgme0 libgraphite2-3 libgsml libgsnap1-krb5-2
libharfbuzz2b libice6 libidn1 libicu61883-0 libipq4c0 libjack-jackd2-0
libjbig2 libjpeg-turbo8 libjpeg8 libjson-c-3 libjson-glib-1.0-0
libjs-on-glib-1.0-common libjs-crypt01 libkeyutil1 libkmmod2 libkrb5-3
libkrb5support0 liblwm10 libltdl17 libmp3lame libmpg123-0 libmysofa0
libnewt0.52 libnl0 libnss-systemd libnumul libogg0 libopenal-data
libopenal libopenjp2-7 libopenmpmp0 libopus0 liborc-0.4-0 libpam-systemd
libpango-1.0-0 libpangocairo-1.0-0 libpangoft2-1.0-0 libpcaccess0
libpango-5.2-0 libpixman-1.0 libpng16-16 libpolkit-agent-1-0
libpolkit-backend-1.0 libpolkit-gobject-1-0 libpop0 libpostproc54
libproxylv libpulse0 libpulsesp libraw1394-11 librsvg2-2 librsvg2-common
librubberband libsampleter0 libsd12-2.0-0 libsensors0 libshine3 libstlsg2
libsm6 libsnapd-glib1 libsnappy1v1 libsndfile1 libsndio16.1 libsoodium23
libsoup2.4-1 libsoxr0 libspeex1 libspeexdsp1 libssh2-openssl-4 libvresample2
libswscale4 libsystemd libtbl1 libthai-data libthai0 libtheora0 libtiff5
libtwolame0 libudev1 libusb-1.0-0 libva-drm2 libva-x11-2 libva2 libvdpaul
libvorbis0a libvorbisenc2 libvorbisfile3 libvpx5 libwvpack1
libwayland-client0 libwayland-cursor0 libwayland-egl libwayland-egl-mesa
libwayland-client0 libwayland-server0 libwebp6 libwebpmux3 libwebrtc-audio-processing1 libwrap0
libx11-6 libx11-data libx11-xcb1 libx264-152 libx265-146 libxa6
libxcb-driv2-0 libxcb-driv3-0 libxcb-glx1 libxcb-present libxcb-render0
libxcb-shaped libxcb-shm0 libxcb-sync1 libxcb-xfixes0 libxcb1 libcursor1
libxdamage1 libxdp0 libxext6 libxfixes3 libxi6 libxinerama1 libxkbcommon0
libxrandr2 libxrender1 libxshmfence1 libxs1 libxtst6 libxv1 libxvidcore4
libxxf86vm1 libzmq5 libzvbi-common libzvbi0 linux-sound-base mesa-va-drivers
mesa-vdpau-drivers multiarch-support networkd-dispatcher policykit-1
```

Figure [24] Installing sound drivers required for microphone usage for recording audio

```

Dashboard > speech evaluator > #19
> git branch -a -v --no-abbrev # timeout=10
> git branch -D master # timeout=10
> git checkout -b master 0826493aa6f2a8040ce86ae8ea30199812ce8e0f # timeout=10
Commit message: "Removed git lfs script in Jenkinsfile"
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Docker Build)
[Pipeline] sh
+ docker build . -t akshaymg99/speech-spe:latest
Sending build context to Docker daemon 1.271GB
Step 1/11 : FROM python:3.6
--> 9f98e1be91a4
Step 2/11 : ENV DockerHOME=/SPE_Speech_Evaluator
--> Using cache
--> d37b3054a01
Step 3/11 : RUN mkdir -p $DockerHOME
--> Using cache
--> 87089910c61
Step 4/11 : WORKDIR $DockerHOME
--> Using cache
--> a4e85927d6fc
Step 5/11 : ENV PYTHONDONTWRITEBYTECODE 1
--> Using cache
--> c9f5a8d0bcd7
Step 6/11 : ENV PYTHONUNBUFFERED 1
--> Using cache
--> a0e6237fafe0
Step 7/11 : RUN pip install --upgrade pip
--> Using cache
--> 8abb88aaeae
Step 8/11 : COPY . $DockerHOME
--> 1890fdec2b7
Step 9/11 : RUN pip install -r requirements.txt
--> Running in 694d848ae9
[91mWARNING: Value for scheme.headers does not match. Please report this to distutils: /usr/local/include/python3.6m/UNKNOWN
sysconfig: /usr/local/include/python3.6m
[0m[91mWARNING: Additional context:
user = False
home = None
root = None
prefix = None
[0mCollecting Django==3.2
  Downloading Django-3.2-py3-none-any.whl (7.9 MB)
Collecting uwsgi>=2.0
  Downloading uwsgi-2.0.19.1.tar.gz (803 kB)
Collecting pandas==0.24.2
  Downloading pandas-0.24.2-cp36-cp36-manylinux1_x86_64.whl (10.1 MB)
Collecting pydub==0.25.1
  Downloading pydub-0.25.1-py3-none-any.whl (32 kB)
Collecting numpy==1.16.2
  Downloaded numpy-1.16.2-cp36-cp36-manylinux1_x86_64.whl (7.9 MB)

Fetched 34.3 MB in 3min 40s (156 kB/s)
(Reading database ... 246226 files and directories currently installed.)
Preparing to unpack .../jenkins_2.277.3_all.deb ...
Unpacking jenkins (2.277.3) over (2.263.3) ...
Setting up jenkins (2.277.3) ...
Processing triggers for systemd (237-3ubuntu10.43) ...
Processing triggers for ureadahead (0.100.0-21) ...
ureadahead will be reprofiled on next reboot
akshay@akshay-OMEN:~$ systemctl start jenkins
akshay@akshay-OMEN:~$ systemctl restart jenkins
akshay@akshay-OMEN:~$ systemctl start rundeckd
akshay@akshay-OMEN:~$ ps
  PID TTY      TIME CMD
11659 pts/1    00:00:00 bash
20930 pts/1    00:00:00 ps
akshay@akshay-OMEN:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
akshaymg99/speech-spe   latest   e0988908dabc  5 days ago   946MB
<none>              <none>   1dcfcf310991  5 days ago   946MB
akshaymg99/speech-spe   <none>   77fbfd2f253d7 8 days ago   946MB
python               3.6      9f98e1be91a4  3 weeks ago  875MB
akshay@akshay-OMEN:~$ docker ps
CONTAINER ID        IMAGE       COMMAND    CREATED     STATUS      PORTS     NAMES
REPOSITORY          TAG      IMAGE ID      CREATED     SIZE
<none>              <none>   1890fdec2b7  6 minutes ago  2.16GB
akshaymg99/speech-spe   latest   e0988908dabc  5 days ago   946MB
<none>              <none>   1dcfcf310991  5 days ago   946MB
akshaymg99/speech-spe   <none>   77fbfd2f253d7 9 days ago   946MB
python               3.6      9f98e1be91a4  3 weeks ago  875MB
akshay@akshay-OMEN:~$ 
```

Figure [25] Docker image is built

```

Dashboard > speech evaluator > #19
[Pipeline] sh
Warning: A secret was passed to "sh" using Groovy String interpolation, which is insecure.
Affected argument(s) used the following variable(s): [dockerHubPwd]
See https://jenkins.io/redirect/groovy-string-interpolation for details.
+ docker login -u akshaymg99 -p ****
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /var/lib/jenkins/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[Pipeline] // withCredentials
[Pipeline] sh
+ docker push akshaymg99/speech-spe:latest
The push refers to repository [docker.io/akshaymg99/speech-spe]
74758d7b4986: Preparing
d6268cae9245: Preparing
24ef41ac6a: Preparing
404fec62c27: Preparing
18e1f8943d42: Preparing
62f3105aaaf52: Preparing
709d45915a7: Preparing
a47f1585472b: Preparing
651326e9f1ca: Preparing
5d5962699bd5: Preparing
a42439ce9650: Preparing
26270c5e25fa: Preparing
e2c6ff4462357: Preparing
62f3105aaaf52: Waiting
709d45915a7: Waiting
a47f1585472b: Waiting
651326e9f1ca: Waiting
a42439ce9650: Waiting
26270c5e25fa: Waiting
e2c6ff4462357: Waiting
18e1f8943d42: Layer already exists
404fec62c27: Layer already exists
62f3105aaaf52: Layer already exists
709d45915a7: Layer already exists
a47f1585472b: Layer already exists
651326e9f1ca: Layer already exists
5d5962699bd5: Layer already exists
a42439ce9650: Layer already exists
26270c5e25fa: Layer already exists
e2c6ff4462357: Layer already exists
24ef41ac6a: Pushed

```

Figure [26] Docker image pushed to hub

```

Dashboard > speech evaluator > #19
74758d7b4986: Preparing
d6268cae9245: Preparing
24ef141ac6a1: Preparing
404cfec62c27: Preparing
18e1f8943d42: Preparing
62f3105aaaf52: Preparing
709d4591f5a7: Preparing
a47f1585472b: Preparing
6513266e9f1ca: Preparing
5d5962699bd5: Preparing
a42439ce9650: Preparing
26270c5e25fa: Preparing
e2c6ff462357: Preparing
62f3105aaaf52: Waiting
709d4591f5a7: Waiting
a47f1585472b: Waiting
6513266e9f1ca: Waiting
a42439ce9650: Waiting
26270c5e25fa: Waiting
e2c6ff462357: Waiting
18e1f8943d42: Layer already exists
404cfec62c27: Layer already exists
62f3105aaaf52: Layer already exists
709d4591f5a7: Layer already exists
a47f1585472b: Layer already exists
6513266e9f1ca: Layer already exists
5d5962699bd5: Layer already exists
a42439ce9650: Layer already exists
26270c5e25fa: Layer already exists
e2c6ff462357: Layer already exists
24ef141ac6a1: Pushed
74758d7b4986: Retrying in 5 seconds
74758d7b4986: Retrying in 4 seconds
74758d7b4986: Retrying in 3 seconds
74758d7b4986: Retrying in 2 seconds
74758d7b4986: Retrying in 1 second
6268cae9245: Pushed
74758d7b4986: Pushed
latest: digest: sha256:75577ceb829c34af976dbb8be9f59b4a158b7c87611a8ef0160196f648dbece size: 3064
[Pipeline] 
[Pipeline] // stage
[Pipeline] 
[Pipeline] // withEnv
[Pipeline] 
[Pipeline] // node
[Pipeline] End of Pipeline
Triggering a new build of deploy speech-spe #9
Finished: SUCCESS

```

Figure [27] Build success of “speech_evaluator” job

Continuous Delivery

A deliverable in Software Engineering is an artifact that is ready to be delivered to the client. It thus is the end product of a SDLC. Continuous Delivery (CD) is the practice of generating deliverable as soon as code changes happen. CD requires that CI pipeline be in place first. Hence, CI is a prerequisite of CD. Here, the deliverable is in the form of a docker image. The image consists of everything that our project requires to run including OS, dependencies, a web server (here Django’s uwsgi), sound drivers for audio input etc. The tools used here for creating a CD pipeline are Docker and Jenkins.

Building and Publishing Images

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. So, first we create a Dockerfile. The Dockerfile is placed in the root directory of the project. It is shown below in Figure [28].

```

1  FROM ubuntu:18.04
2
3  RUN apt-get update && apt-get install -y \
4      python3.6 \
5      python3-pip
6
7  RUN mkdir /SPE_Speech_Evaluator
8
9  WORKDIR /SPE_Speech_Evaluator
10
11 COPY . /SPE_Speech_Evaluator
12
13 RUN apt-get -y install locales
14 RUN touch /usr/share/locale/locale.alias
15 ENV LANG=en_US.UTF-8 \ LANGUAGE=en_US \ LC_ALL=en_US.UTF-8
16
17 RUN echo $LANG
18 RUN python3 -c 'import locale; print(locale.getpreferredencoding())'
19
20 RUN apt-get update \
21     && apt-get install -y pulseaudio alsa-utils alsa-base ffmpeg
22
23 RUN apt-get update \
24     && apt-get install libportaudio2 libportaudiocpp0 portaudio19-dev libsndfile1-dev -y \
25     && pip3 install pyaudio
26
27 RUN apt-get install -y pulseaudio
28
29 CMD python3 -c "import pyaudio"
30
31 COPY requirements.txt /SPE_Speech_Evaluator
32
33 RUN pip3 install -r requirements.txt
34
35 COPY . /SPE_Speech_Evaluator
36
37 EXPOSE 8000
38
39 CMD python3 manage.py runserver 0.0.0.0:8000

```

Figure [28] Dockerfile placed in root of project

In Jenkins pipeline, we configure a stage to build docker image of our web-application and publish it to DockerHub. In the above Dockerfile, we use base OS as Ubuntu 18.04, and install python3 and pip first. Then we set the working directory and install and set language as utf-8 in environment. Then we install necessary sound drivers, alsa drivers and codecs which are required to record sound through microphone device. Pulse audio and pyaudio is required to record audio. Then we install all the required dependencies through requirements.txt file. We expose port 8000 of built docker container through which we can access our web-application. The last command is Django's server run command.

Jenkins builds and pushes the image to Hub through commands “`docker build . -t akshaymg99/speech-spe:latest`” and “`docker push akshaymg99/speech-spe:latest`” as shown in Jenkinsfile Figure [17].

We can also see the pushed docker image in our Docker hub account as shown in Figure [29].

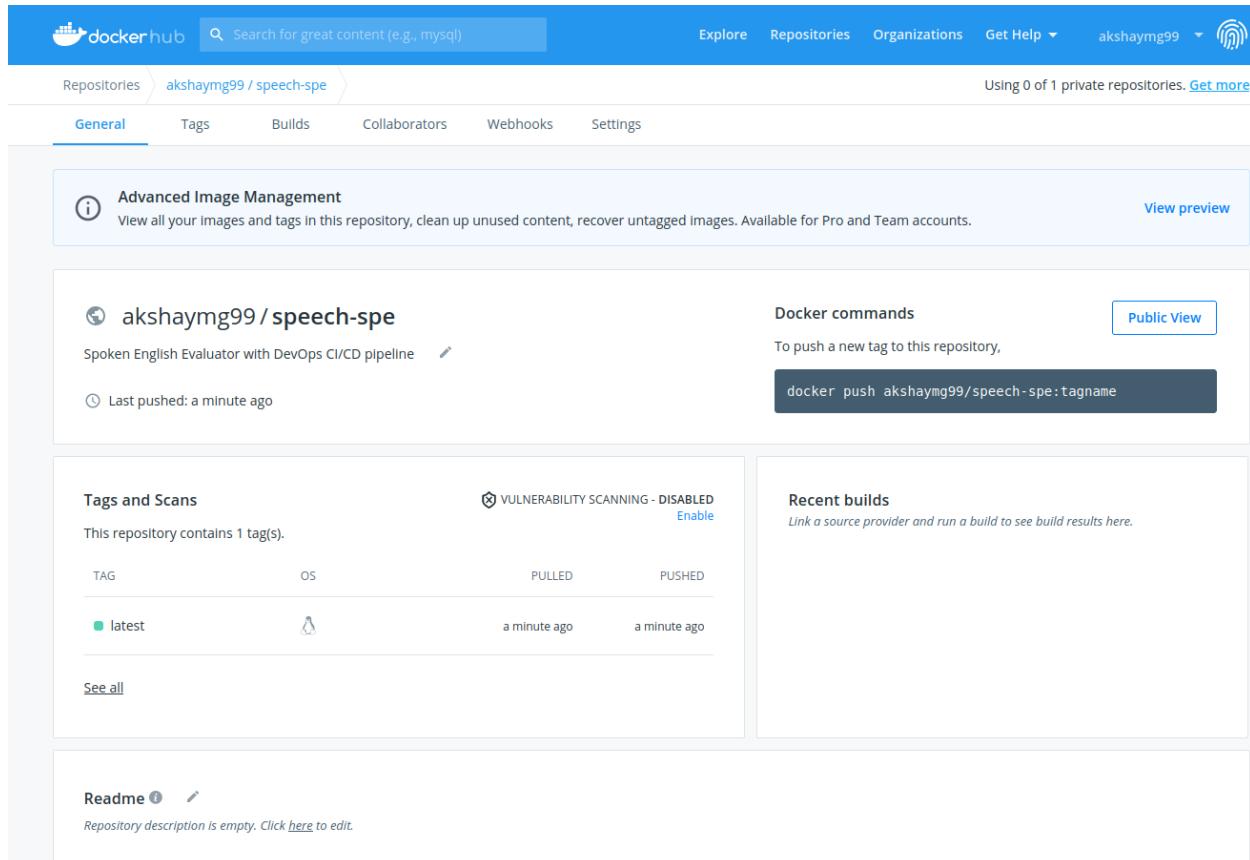


Figure [29] Docker Hub account with pushed images

Continuous Deployment

Continuous deployment is a strategy for software releases wherein any code commit that passes the automated testing phase is automatically released into the production environment, making changes that are visible to the software's users.

After the deliverable (image in our case) is created and published to Dockerhub, we use Rundeck to fetch the image and deploy it in a container. The Rundeck job is invoked by Jenkins.

Creating Rundeck Project and Job

We create a Rundeck project and Rundeck job that deploys the generated Docker image locally. To do so, we follow these steps:

1. Create a new Rundeck job

Create New Job: Deploy Docker Image

Job Name: Deploy Docker Image

Description:

```
1 - This rundeck job deploys docker image which contains Django web application of Speech Evaluator for SPE Project
```

The first line of the description will be shown in plain text, the rest will be rendered with Markdown. [More...](#)

Cancel Create

2. Workflow description

Workflow must have at least one step

Workflow

If a step fails:

- Stop at the failed step.
- Run remaining steps before failing.

Strategy: Node First

Execute all steps on a node before proceeding to the next node.

Global Log Filters [+ add](#)

Workflow steps:

1. docker container prune
2. docker rmi -f akshaymg99/speech-spe:latest
3. docker pull akshaymg99/speech-spe:latest
4. docker run -d -p 8000:8000 akshaymg99/speech-spe:latest

+ Add a step

Cancel Create

3. Configure Node to execute it locally

4. Note down the UUID of the job to put it in Jenkins

Workflow must have at least one step

Log level: Normal Debug

Multiple Executions: Yes

Allow this job to be executed more than one time simultaneously?

Limit Multiple Executions? Max number of multiple executions. Use blank or 0 to indicate no limit.

Timeout The maximum time for an execution to run. Time in seconds, or specify time units: "120m", "2h", "3d". Use blank or 0 to indicate no timeout. Can include option value references like "\${option.timeout}".

Retry Maximum number of times to retry execution when this job is directly invoked. Retries will occur if the job fails n times out, but not if it is manually killed. Can use an option value reference like "\${option.retry}".

Retry Delay The time between the failed execution and the retry. Time in seconds, or specify time units: "120m", "2h", "3d". Use blank or 0 to indicate no delay. Can include option value references like "\${option.retry.delay}".

Log Output Limit Enter either maximum total line-count (e.g. "100"), maximum per-node line-count ("100/node"), or maximum log file size ("100MB", "100KB", etc.), using "1GB", "1MB", "1B", "1" as Giga-, Mega-, Kilo- and bytes.

Log Limit Action Hat with status: [Failed/Retried, or very slow](#)

Truncate and continue

Action to perform if the output limit is reached.

Default Tab: Nodes Log Output HTML

Default tab to display when you follow an execution.

UUID: 67992a9e-dc08-405f-beaa-0be60b4547e9

Cancel Create

We have shown how to use Jenkins to invoke Rundeck deploy job using UUID in Continuous Integration Part. We have also explained on how to use trigger to start the execution of this job. Figure [30] shows docker deployed locally running in Rundeck console.

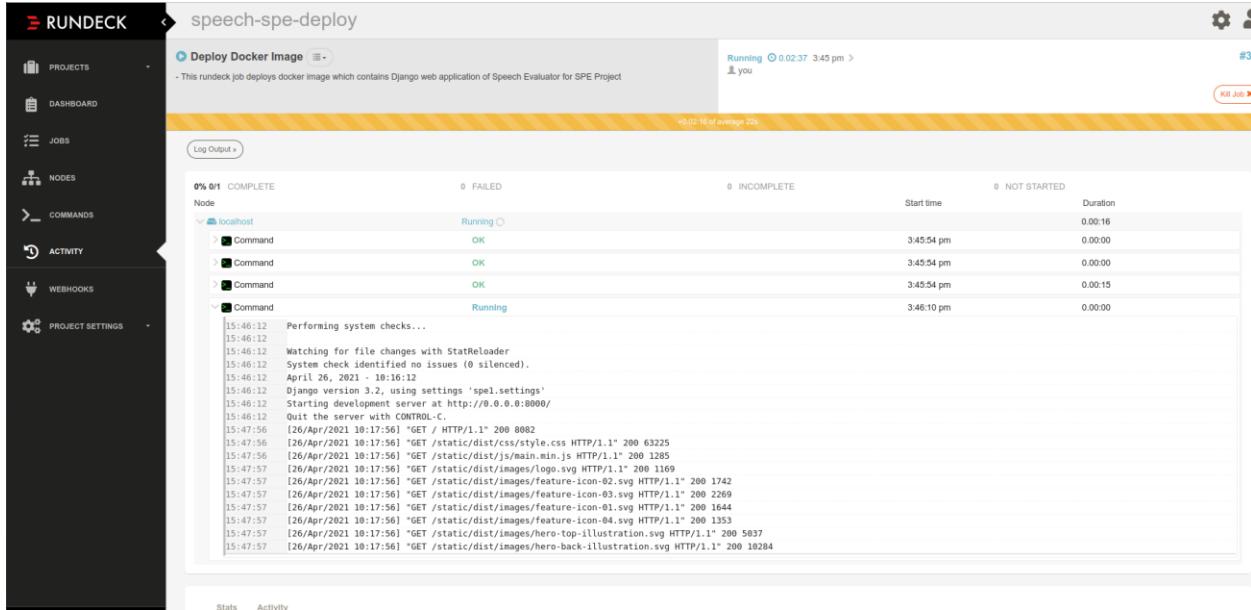


Figure [30] Rundeck deploying docker container locally

Continuous Monitoring

Monitoring refers to monitoring the deployed artifacts in real-time to keep a check on faults and measure performance. This is done by analyzing logs generated by the system. The Elastic Stack, also known as ELK Stack, comprises of Elasticsearch, Logstash, Kibana and Beats.

Elasticsearch is a modern search and analytics engine which is based on Apache Lucene. It is used as to store and index logs and can be then queried to extract meaningful insights. It can be used for numerous types of data including textual, numerical, geospatial, structured, and unstructured.

Logstash is a tool that is used for parsing logs. It is very useful in parsing unstructured logs and giving them structure so that logs can be efficiently searched and analyzed. Log aggregated and processed by Logstash go through 3 stages collection, processing and dispatching.

Kibana adds a visualization layer to the Elastic Stack. It is a browser-based user interface that can be used to search, analyze and visualize the data stored in Elasticsearch indices. Beats are a collection of open source log shippers that act as agents installed on the

different servers in your environment for collecting logs or metrics. These shippers are designed to be lightweight in nature they leave a small installation footprint, are resource efficient, and function with no dependencies. Common Beats that are used today included Filebeat, Metricbeat, Winlogbeat, Packetbeat, etc.

Here we use the Elastic Stack to perform log analysis in this project. We are use Filebeat to collect logs generated by Django's web application, which are then given to Elasticsearch for analysis. Kibana is used to charts to visualize the data diagrammatically.

Configuring Logging in Django

Django provides loggers to generate logs and push them to either files or Logstash as required. Below Figure [31] shows Loggers configured in Django framework. We have configured it to generate logs and store them in a file. These settings are present in setting.py file

```

24     LOGGING = {
25         "version": 1,
26         "disable_existing_loggers": False,
27         "root": {"level": "INFO", "handlers": ["file"]},
28         "handlers": {
29             "file": {
30                 "level": "INFO",
31                 "class": "logging.FileHandler",
32                 "filename": os.path.join(BASE_DIR, 'debug.log'),
33                 "formatter": "app",
34             },
35         },
36         "loggers": {
37             "django": {
38                 "handlers": ["file"],
39                 "level": "INFO",
40                 "propagate": True
41             },
42             "django.request": {
43                 "handlers": ['file'],
44                 "level": 'INFO',
45             },
46            "": {
47                 "level": 'INFO',
48                 "handlers": ['file'],
49             },
50         },
51     },
52     "formatters": {
53         "app": {
54             "format": (
55                 u"%(asctime)s [%(levelname)-8s] "
56                 "(%(module)s.%(funcName)s) %(message)s"
57             ),
58             "datefmt": "%Y-%m-%d %H:%M:%S",
59         },
60     },
61 },
62 }
63

```

Figure [31] Logging configuration in Django

Then we use Filebeat to fetch the logs present in the root directory of project (/debug.log file). Configuring of Filebeat is shown in below figures. It is present at /etc/filebeat/filebeat.yml

```

# For more available modules and options, please see the filebeat.reference.yml sample
# configuration file.

# ===== Filebeat inputs =====

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

- type: log

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /var/log/*.log
    - /home/akshay/Desktop/IIITB/SPE_Speech_Evaluator/*.log
    #- c:\programdata\elasticsearch\logs\*

  # Exclude lines. A list of regular expressions to match. It drops the lines that are
  # matching any regular expression from the list.
  #exclude_lines: ['^DBG']

  # Include lines. A list of regular expressions to match. It exports the lines that are
  # matching any regular expression from the list.
  #include_lines: ['^ERR', '^WARN']

  # Exclude files. A list of regular expressions to match. Filebeat drops the files that
  # are matching any regular expression from the list. By default, no files are dropped.
  #exclude_files: ['.gz$']

  # Optional additional fields. These fields can be freely picked
  # to add additional information to the crawled log files for filtering
  #fields:

  # Optional additional fields. These fields can be freely picked
  # to add additional information to the crawled log files for filtering
  #fields:
  #  level: debug
  #  review: 1

# ===== Filebeat modules =====

filebeat.config.modules:
  # Glob pattern for configuration loading
  path: ${path.config}/modules.d/*.yml

  # Set to true to enable config reloading
  reload.enabled: true

  # Period on which files under path should be checked for changes
  #reload.period: 10s

# ===== Elasticsearch template setting =====

setup.template.settings:
  index.number_of_shards: 1
  #index.codec: best_compression
  #_source.enabled: false

# ===== General =====

# The name of the shipper that publishes the network data. It can be used to group
# all the transactions sent by a single shipper in the web interface.
#name:

# The tags of the shipper are included in their own field with each
# transaction published.
#tags: ["service-X", "web-tier"]

# Optional fields that you can specify to add additional information to the

```

```

#fields:
# env: staging

# ===== Dashboards =====
# These settings control loading the sample dashboards to the Kibana index. Loading
# the dashboards is disabled by default and can be enabled either by setting the
# options here or by using the `setup` command.
setup.dashboards.enabled: true

# The URL from where to download the dashboards archive. By default this URL
# has a value which is computed based on the Beat name and version. For released
# versions, this URL points to the dashboard archive on the artifacts.elastic.co
# website.
#setup.dashboards.url:

# ===== Kibana =====

# Starting with Beats version 6.0.0, the dashboards are loaded via the Kibana API.
# This requires a Kibana endpoint configuration.
setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the default (http and 5601)
  # In case you specify and additional path, the scheme is required: http://localhost:5601/path
  # IPv6 addresses should always be defined as: https://[2001:db8::1]:5601
  host: "localhost:5601"

  # Kibana Space ID
  # ID of the Kibana Space into which the dashboards should be loaded. By default,
  # the Default Space will be used.
  #space.id:

# ===== Elastic Cloud =====

# These settings simplify using Filebeat with the Elastic Cloud (https://cloud.elastic.co).

# The cloud.id setting overwrites the `output.elasticsearch.hosts` and
# `setup.kibana.host` options.
# Configure what output to use when sending the data collected by the beat.

# ----- Elasticsearch Output -----
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]
  index: "logstash-%{[yyyy.MM.dd]}"
setup.template:
  name: 'logstash'
  pattern: 'logstash-*'
  enabled: false
  # Protocol - either `http` (default) or `https`.
  #protocol: "https"

  # Authentication credentials - either API key or username/password.
  #api_key: "id:api_key"
  #username: "elastic"
  #password: "changeme"

# ----- Logstash Output -----
#output.logstash:
  # The Logstash hosts
  #hosts: ["localhost:5044"]

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
  #ssl.certificateAuthorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client Certificate Key
  #ssl.key: "/etc/pki/client/cert.key"

# ===== Processors =====
processors:
  - add_host_metadata:
      when.not.contains.tags: forwarded

```

YAML ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

Filebeat fetches the logs and sends it to Elasticsearch. There, we can make an index pattern to filter the log data. Figure [32] shows creating index pattern in Elasticsearch.

Figure [32] Creating index pattern

Next we select that saved index pattern for visualization as shown in Figure [33]

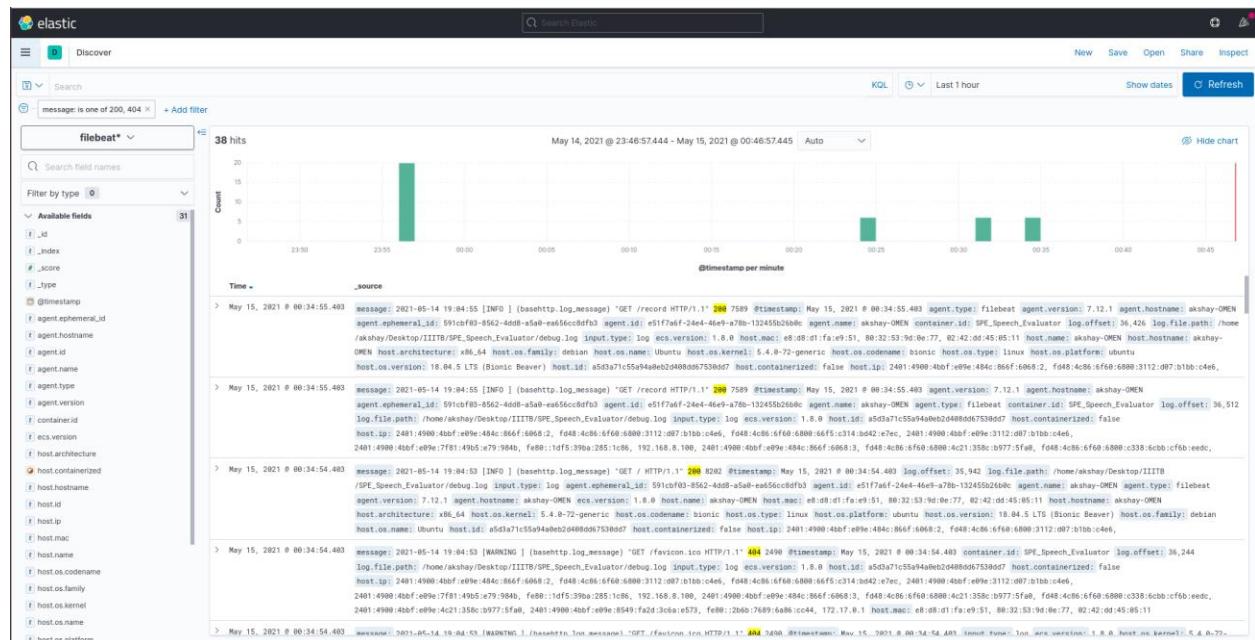
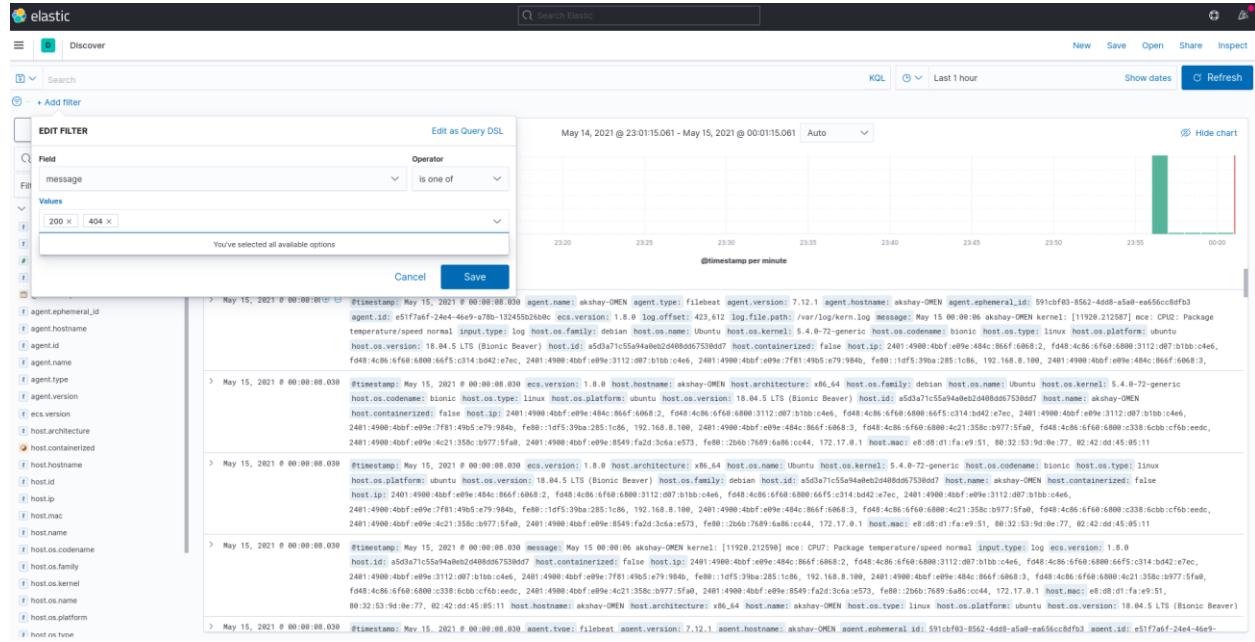


Figure [33] Selecting index pattern

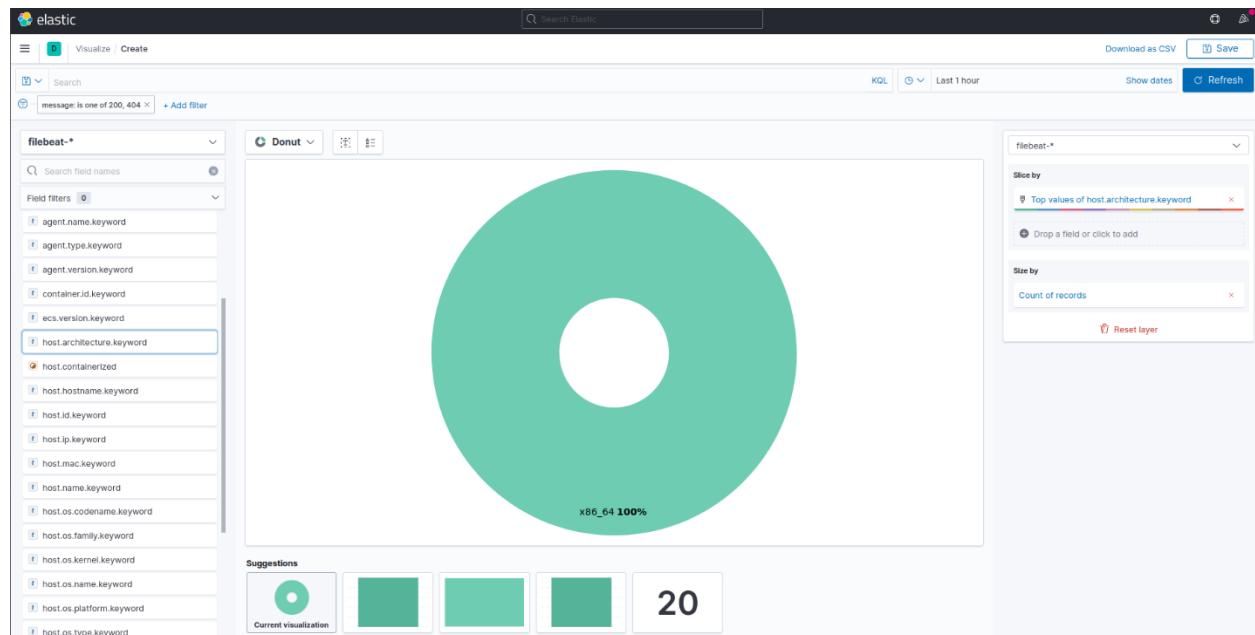
Then we add filters to select data which contains status code 200 and 404 which will help us identify when the webpages where successfully retrieved or not. We save this query here.



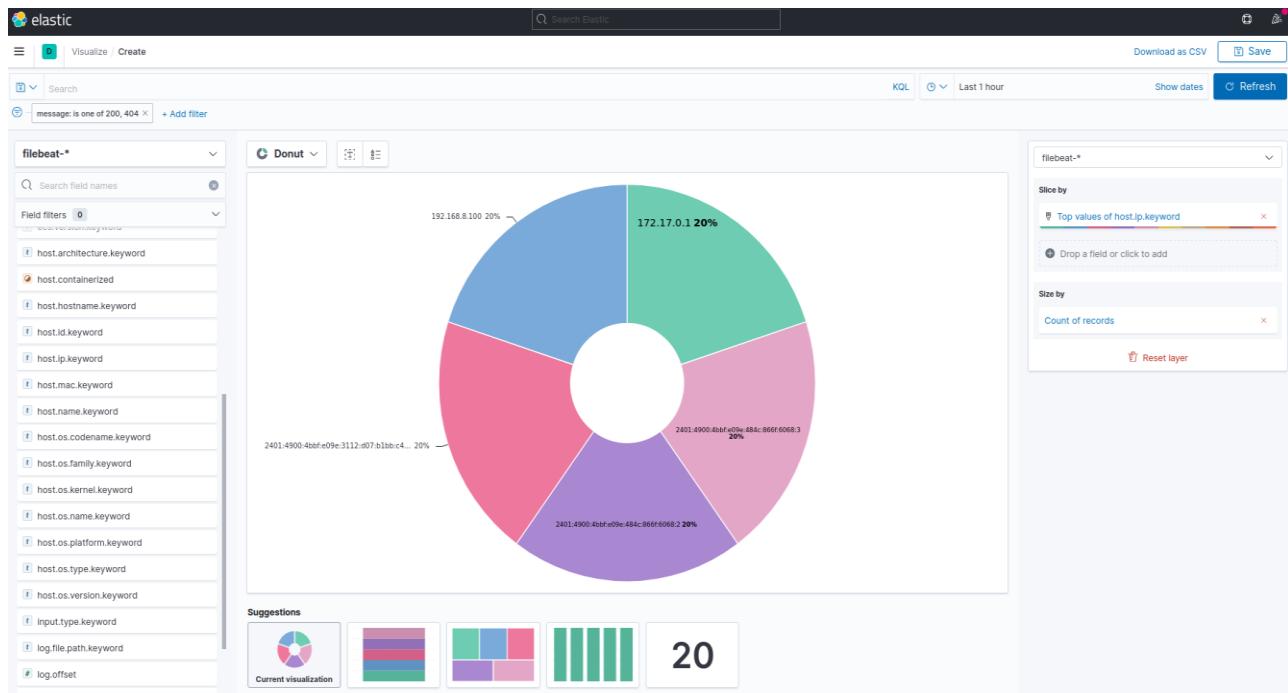
After that, we open Kibana visualization, and select the saved query to display visualizations.

Below are some interesting analytics obtained visually from the logs.

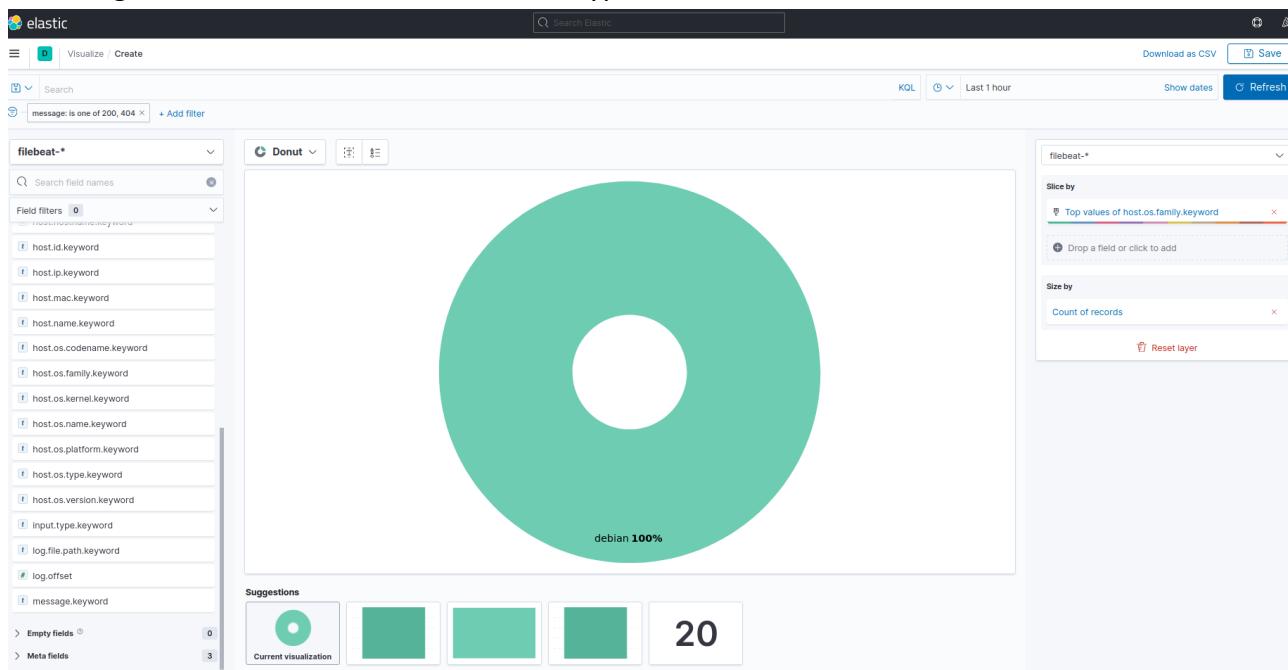
This shows information about the host architecture in the form of a pie chart.



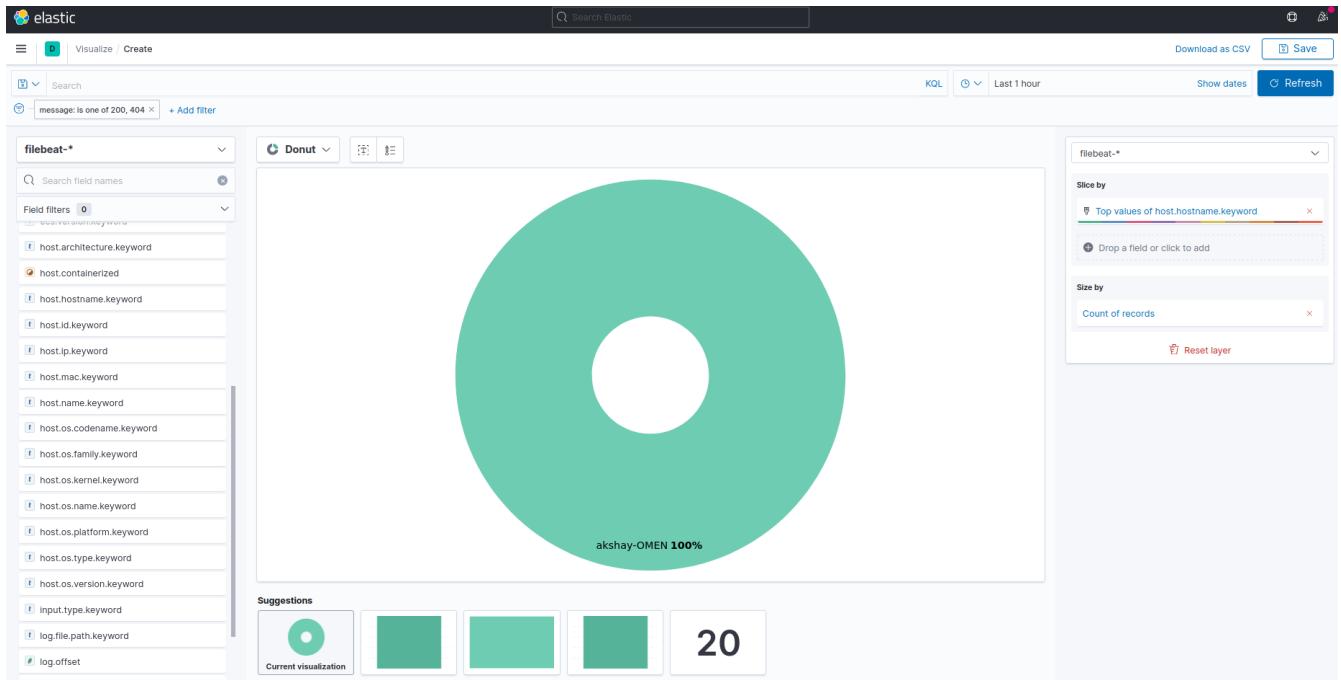
Next figure shows information about host IP address.



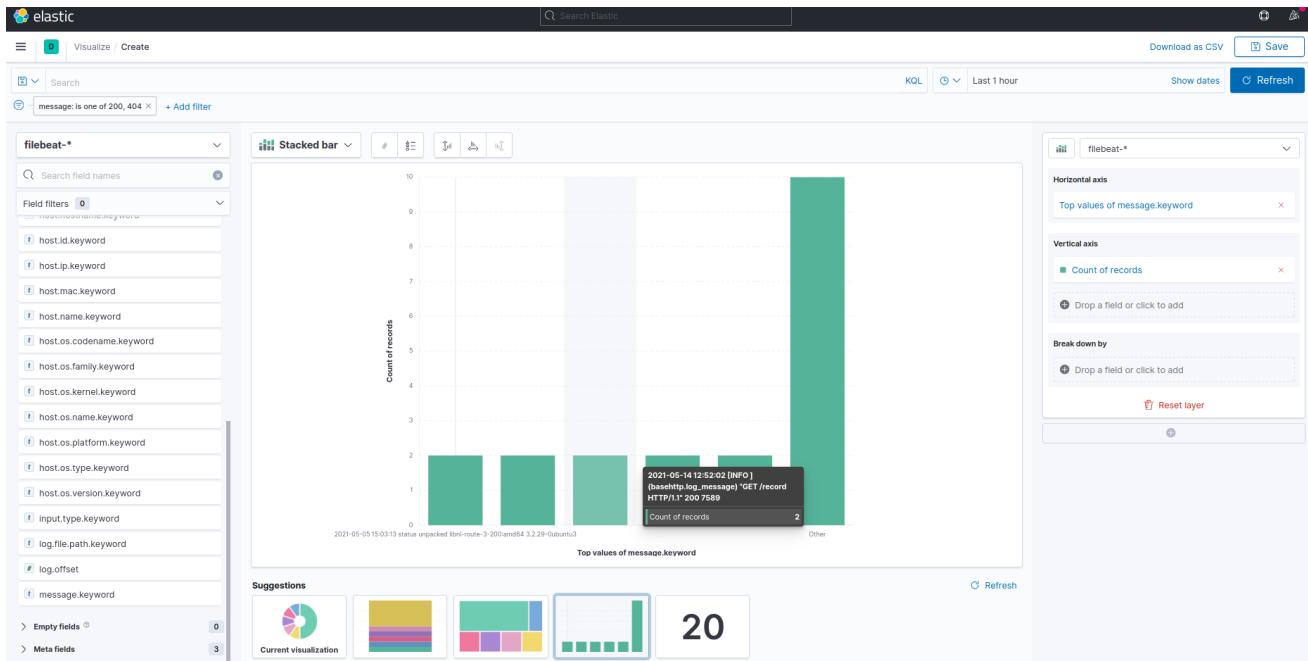
Below figure shows information on host OS type:



Next Figures shows information about hostname:



The following Figure shows information on message requests in the form of bar chart:



Results and Discussions:

The web-application calculates result in the form of sentence analysis, vocabulary strength and unique word count. Figure [33] shows these result displayed in result webpage.

The screenshot shows a web browser window with the URL `0.0.0.0:8000/record`. The page title is "Report on Interview performance (Speech)". Below the title is a table with two columns: "Sentences" and "Analysis". The table contains ten rows of data. A summary section below the table states "Vocabulary strength is based on how powerful/impactful words you use" and provides a breakdown of vocabulary strength. At the bottom, it notes "Unique words used: 27".

Sentences	Analysis
[CLS] i did my undergraduate degree in electronics at hubli [SEP]	Correct
[CLS] what masters admission in bangalore [SEP]	Wrong
[CLS] and about time management [SEP]	Correct
[CLS] in the field of computers [SEP]	Correct
[CLS] and meet the requirements of the job [SEP]	Correct
[CLS] and have any certification and accomplishments in my profile [SEP]	Wrong
[CLS] pencil project and thought [SEP]	Wrong
[CLS] production and technical knowledge to face any difficulties and counters [SEP]	Wrong
[CLS] machine learning model [SEP]	Correct
[CLS] expectations of the company [SEP]	Correct

Vocabulary strength is based on how powerful/impactful words you use

Vocabulary strength	310.14	Above average. You have good understanding of english words. Try fine-tuning the words and aim at highest
---------------------	--------	---

Unique words used: 27

Figure [33] Result displayed on webpage

We find that speech to text conversion is not perfect and it also sometimes fails to properly separate the sentences as we speak. We used available open source google speech recognition library, so we couldn't make this feature perfect as more optimal libraries which were not free to use. This is the only drawback of this application.

Scope for Future Work

We built an audio based system to evaluate speech for interviews. It could be taken forward and additional video analysis could also be added in future. Features such as body language analysis, emotion / confidence level analysis, eye contact all could be checked in a person's response video. This would make interview performance analysis much better.

Conclusion

In this project, we built a responsive web-application and automated the entire SDLC using DevOps toolchain. This makes the development team and operations team work productively as the DevOps pipeline gives the comfort of making code changes easily and also reduces the chances of encountering errors in production. The toolchain allows companies to quickly build, test and deploy new versions of their products.