



INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY, BANGALORE

SOCIAL DISTANCING MONITOR

(UNDER THE GUIDANCE OF PROFESSOR B THANGARAJU AND NEHA KOTHARI (TA))

GROUP MEMBERS:

ANKITA DUTTA (MT2021017)

RICHA VARMA (MT2021107)

TABLE OF CONTENTS

1. ABSTRACT	
1.1 FEATURES	
2. WHAT IS DEVOPS?	
3. INTRODUCTION	
A. APPLICATION OVERVIEW	
B. OBJECT DETECTION MODEL - YOLOv3	
4. SYSTEM CONFIGURATION	
5. TECHNOLOGY STACK AND DEVOPS TOOLS	
6. SOFTWARE DEVELOPMENT LIFECYCLE	
6.1. INSTALLATIONS	
6.2. SOURCE CONTROL MANAGEMENT	
6.3. CONTINUOUS INTEGRATION WITH JENKINS	
6.3.1 INTEGRATING JENKINS WITH GITHUB	
6.3.2. BUILDING AND TESTING WITH UNITTEST	
6.3.3 DOCKER MANAGEMENT	
6.3.4. CONTINUOUS DEPLOYMENT WITH ANSIBLE	
6.3.5 JENKINS PIPELINE	
6.4. MONITORING	
7. CODE WALKTHROUGH	
8. RESULT AND DISCUSSION	
8.1. USER INTERFACE	
8.1.1. USER LOGIN PAGE	
8.1.2. DASHBOARD	
8.1.3. UPLOAD PAGE	
8.1.4. CUSTOMIZE DASHBOARD	
9. CHALLENGES	
10. SCOPE FOR FUTURE WORK	
10.1. INTEGRATION WITH CCTV FEED	
10.2. MASK DETECTION	
11. CONCLUSIONS	
12. REFERENCES	

1. Abstract



The COVID-19 pandemic has led to a dramatic change in people's lives across the globe. It has caused a plethora of socio-economic concerns and a massive amount of loss of human life. Amidst this chaos, we have learned that social distancing is an effective method to restrict the spread of the coronavirus. However, it can often be a challenge in public places to practice social distancing due to reasons like large crowds, work requirements, or simple ignorance of people. In these scenarios, the authorities (such as government, college administrations, and workplaces) can have a hard time enforcing and monitoring if the people on the premise abide by the physical distancing rules.

Our application, the Social Distancing Monitor (*hereinafter referred to as SDM*), proposes tackling this issue by developing an application that aids authorities in enforcing social distancing of their premise and performing crowd control as and when necessary. We have used YOLOv3, a state-of-the-art object detection algorithm combined with functionalities that cater to the physical distancing issue.

1.1 Features

User can:

- Upload the footage of the premise
- Display the uploaded video footage
- Download the processed video showcasing the social distancing violations and their count
- Plot the graph for the number of violations for the current day
- Set a threshold for the number of violations
- Display the time of the day and number of violations when the threshold was crossed
- Toggle between light and dark themes for the application

The application is to be used by the authoritative bodies of organizations, and hence the administrative staff is the primary user here.

2. What is DevOps?

DevOps can be best explained as people working together to conceive, build and deliver secure software at top speed. DevOps practices enable software developers (Devs) and operations (Ops) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement.

Stemming from an Agile approach to software development, a DevOps delivery process expands on the cross-functional approach of building and shipping applications faster and more iteratively. In adopting a DevOps development process, you decide to improve the flow and value delivery of your application by encouraging a more collaborative environment at all stages of the development cycle.

DevOps represents a change in mindset for IT culture. In building on top of Agile, lean practices, and systems theory, DevOps focuses on incremental development and rapid software delivery. Success relies on creating a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes.

2.1 Reasons to choose DevOps Methodology

1. Quicker mitigation of software defects

With better communication and collaboration between operations and software development, you can identify and mitigate defects at any stage of the development cycle. The same culture can be applied to Application development, where flaws prove costlier.

2. Better resource management

During the application and software development stage, developers and testers constantly wait for resources to arrive, causing delivery delays. Agile with DevOps ensures that the app development comes into the testing phase much quicker than existing operations.

3. Reduced human errors

DevOps reduces the chances of human errors during the development and operations process by deploying frequent iterations. Lower the application failure rate with multiple deployments in the process in a defined timeline.

4. Enhanced version control

Emphasizing the individuals and interactions, DevOps allows the developers to leverage programmable dynamic infrastructure at all stages of the software application development cycle. It enables version control and automated coding options.

5. Stable operating environment

Stability is the key to any business platform, and DevOps is established to bring stability with reliability. Organizations with DevOps get their deployment 30 times faster than their rivals, with 50% lesser chances of failure.

2.2 DevOps Pipeline

A DevOps pipeline consists of different stages to move the code to production in the software development process. The various stages are as follows:

Stage-1: Source Code Control

In this stage, developers get to work writing applications or updates within an editor. This can be as simple as a text editor or as detailed as an integrated development environment. First, all code is checked into a centralized source code repository. The pipeline is triggered once the developer commits the code to the Source Code Control system, then the next stage is executed.

Stage-2: Build and Test

This stage involves compiling the code to create a deployable build. After a successful build, automated testing is performed to ascertain that the build meets the required functional, performance, design, and implementation requirements.

Stage-3: Release Automation

The runtime artifact (i.e. the compiled code) is saved in a repository in this stage. The repository may also contain an older version of a runtime artifact that can be redeployed during rollback.

Stage-4: Deploy

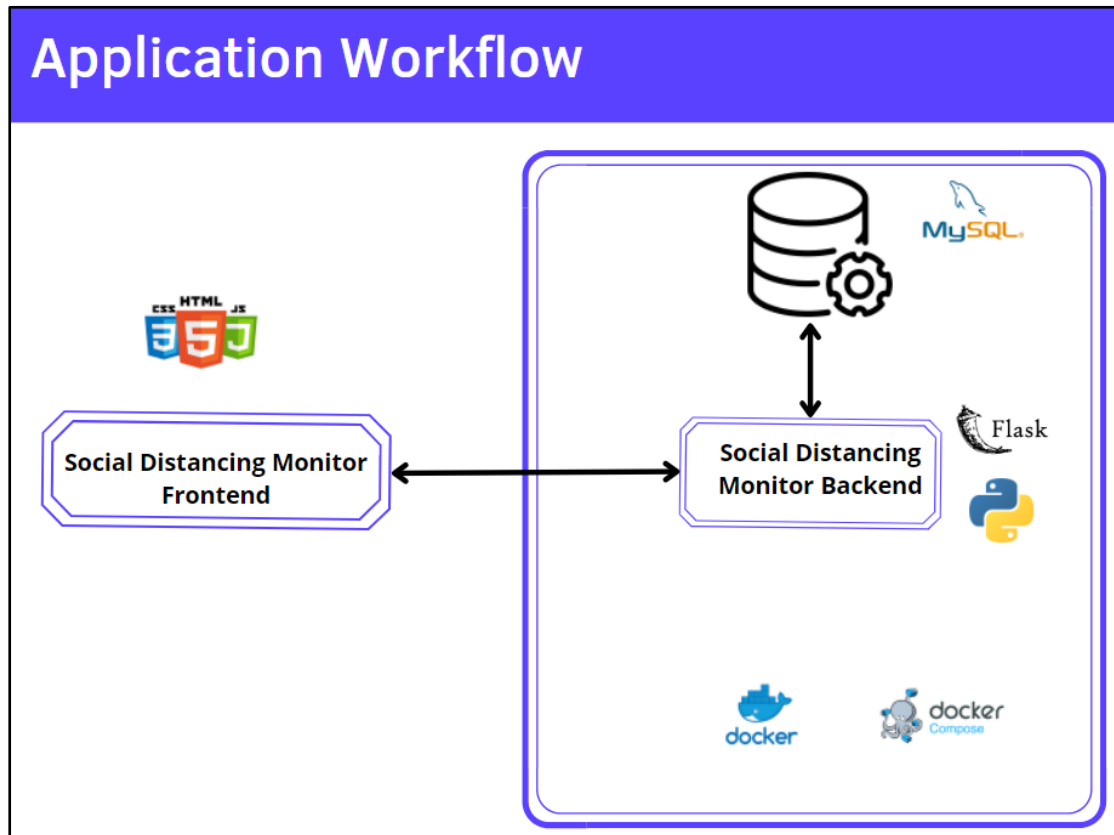
The last stage is deploying the runtime artifact to the production environment. Once the build has successfully passed through all the required test scenarios, it is ready to deploy to the live server.

A DevOps pipeline is focused on automating and connecting these stages of different tasks performed by several teams, such as continuous integration for developers, test automation for testers, and code deployment by release managers.

Different Organizations use different tools to build this pipeline. However, the main challenge is connecting all these tools as part of a DevOps pipeline.

3. Introduction

3.1 Application Overview:



SDM frontend is built using HTML, CSS, and JavaScript, and the backend is developed using Flask API and Python. MySQL is used to store all the data obtained from the application. Finally, we use DockerHub to fetch the images of our application and database and run them in multiple containers by using Docker-Compose. In addition to this, we also have four more containers running for Filebeat and ELK.

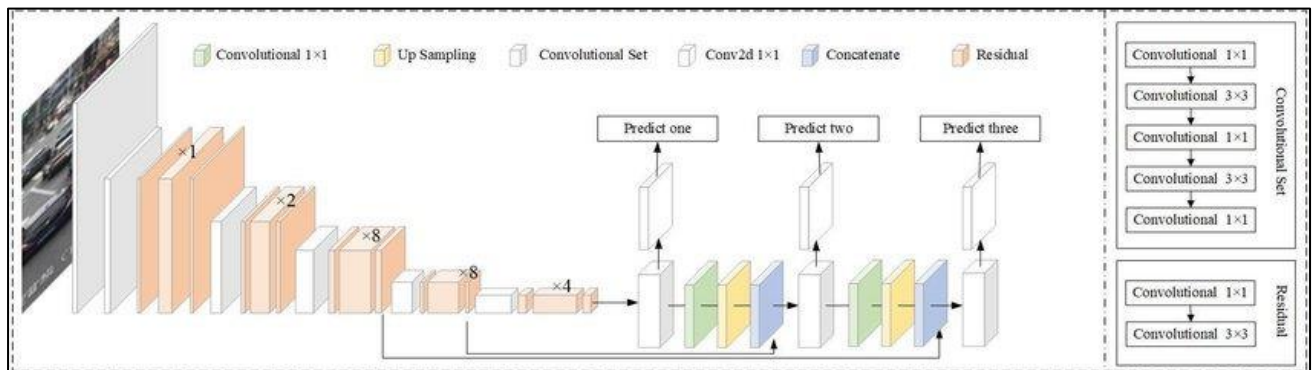
3.2 Object Detection Model - YOLOv3 (You Only Look Once, Version 3)

Object detection is one of the most challenging tasks in computer vision. While there are numerous object detection algorithms, we have used the state-of-the-art YOLOv3 model for our project.

YOLOv3 is a Convolutional Neural Network (CNN) that performs object detection in real-time. CNNs are classifier-based models that process input images/videos as structured data arrays and identify patterns between them. YOLOv3 has the advantage of being much faster than other networks and still maintains accuracy.

We have leveraged the object detection capabilities of the YOLOv3 model and developed the logic to cater to the goals of SDM.

The YOLOv3 Architecture:



The YOLOv3 model is trained on the COCO dataset. The COCO dataset consists of the following 80 classes of objects.

'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck',
'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench',
'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra',
'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove',
'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli',
'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant',
'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard',
'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book',
'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'

3.3 Social Distance Monitoring Algorithm

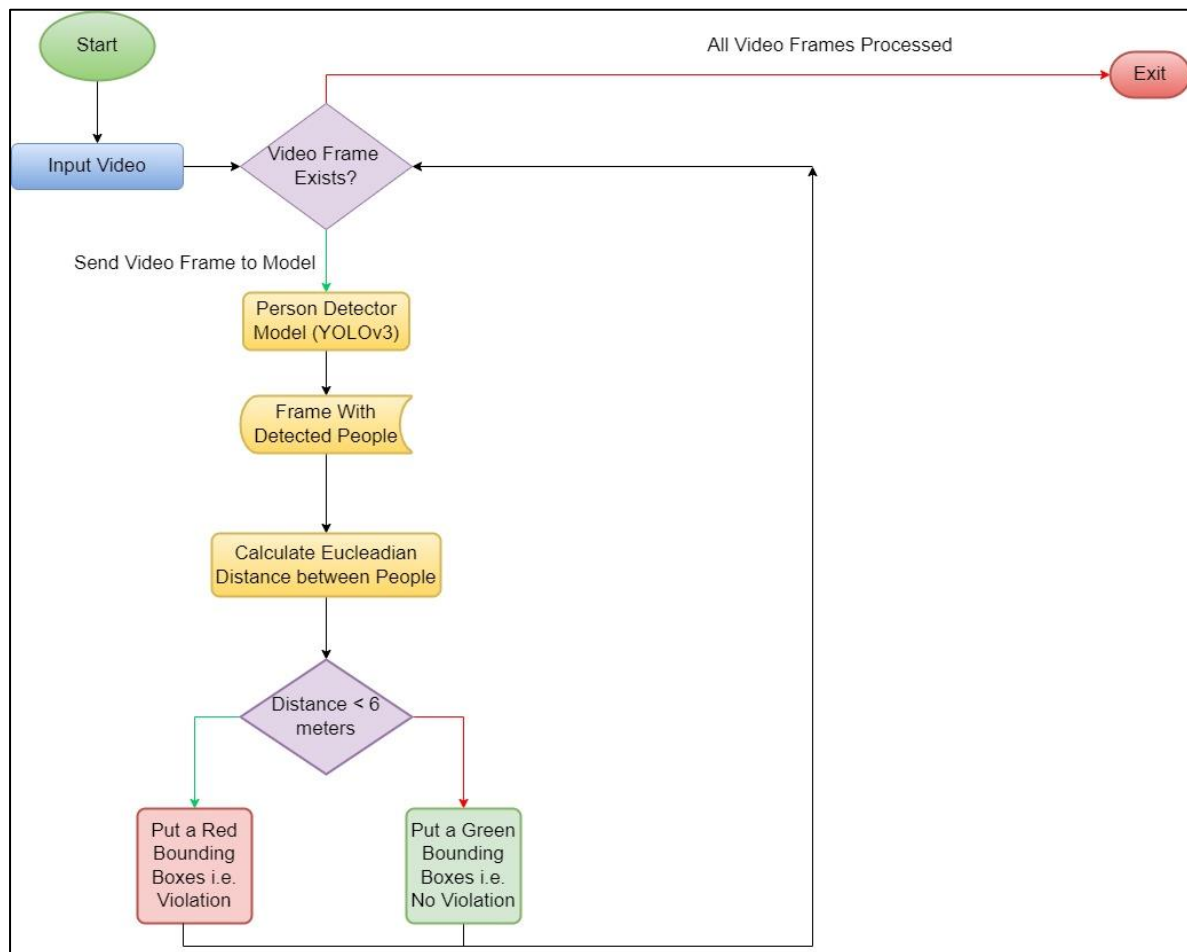
Among the classes in which the YOLOv3 model is trained, we are concerned about the person class. Our goal is to identify people in the input video footage and verify whether they maintain social distancing. Hence after detecting people in the video frames, we find the centroid of each person and measure the distance between all possible pairs of people. If the distance between them is lesser than the standard norms, then it is marked as a violation, i.e., red boxes around the people who violate the social distancing norm.

Windowing Technique: As the video is getting processed frame by frame, it is a possibility that the number of violations stays constant throughout multiple consecutive frames.

Challenge: Logging the violations for every frame can be redundant and lead to a massive expansion in the number of entries in the database without adding much value.

Hence we process the frames with a window size of 10, i.e., we take the maximum violation observed in 10 consecutive frames and store the maximum of it. In this way, we can control the number of entries logged into the database without losing peak information, i.e., peak violations.

Below is a high-level overview of the social distancing algorithm in place. This is the critical part of the processing taking place in the backend.



4. System Configuration

We have used the below configurations for frontend, backend, and monitoring purposes.

Platform: Ubuntu 20.04 LTS

Kernel Details: Linux/UNIX

RAM: 6 GiB

No of vCPU: 1

Memory volume: 100 GiB

5. Technology Stack and DevOps Tools

Frontend: HTML, CSS, and JavaScript

Backend: Flask API, Python

Database: MySQL

Source Control Management: GitHub

Containerization: Docker

Continuous Integration (CI): Jenkins

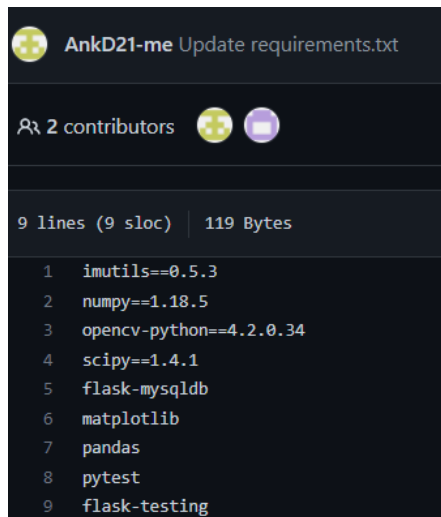
Continuous Deployment (CD): Ansible

Monitoring: Filebeat, ElasticSearch, Logstash, Kibana (ELK Stack)

6. Software Development Lifecycle

6.1. Installations

To run the application along with the backend model, we require multiple installations such as:



```
AnkD21-me Update requirements.txt
2 contributors
9 lines (9 sloc) | 119 Bytes
1  imutils==0.5.3
2  numpy==1.18.5
3  opencv-python==4.2.0.34
4  scipy==1.4.1
5  flask-mysqldb
6  matplotlib
7  pandas
8  pytest
9  flask-testing
```

We have put all the libraries to be installed into a requirements.txt file. So when we build our Docker image, we can add run command to execute this file and have all the necessary installations in place.

For the object detection task, libraries such as numpy, opencv-python, imutils, and scipy are required. At the same time, pandas and matplotlib help us display the gathered statistics from the processed input video as output. Finally, flask-db helps integrate our backend with the MySQL database, and PyTest and Flask-testing provide the testing support.

To verify that our required libraries do them correctly, we can try to run the application locally using:

```
python3 app.py
```

```
python3 main.py
```

6.2 Source Control Management

A Source Code Management (SCM) is a software tool used by programmers to manage the source codes. Source code management systems allow us to track our code change, see a revision history for our code, and revert to previous project versions when needed. Source code management systems help streamline the development process and provide a centralized source for our code.

We required a source code management system so that we could collaborate on code as a team and work independently to develop different parts of the project. We used Git as the SCM tool for our project.

To start tracking our project, we created a repository on GitHub and added the other as a collaborator. On accepting the request to collaborate, both of us had access to the repository. Using the repository link, we cloned the repositories in a directory on our local system. We can check the presence of the .git folder in our directory by viewing the hidden files.

We used branches to develop parts of the project in an isolated space and then merged our elements. We had to pull the latest code from git, resolve conflicts, and then push the changes to GitHub for merging.

We used the following git commands during the development of our project-

`git clone <repository url>` - This command copies the entire data on the git URL

`git checkout -b <branch_name>` - This command creates a new branch with the name as in 'branch_name'

`git add <changed files>` - This command adds changes in the working directory to the staging area

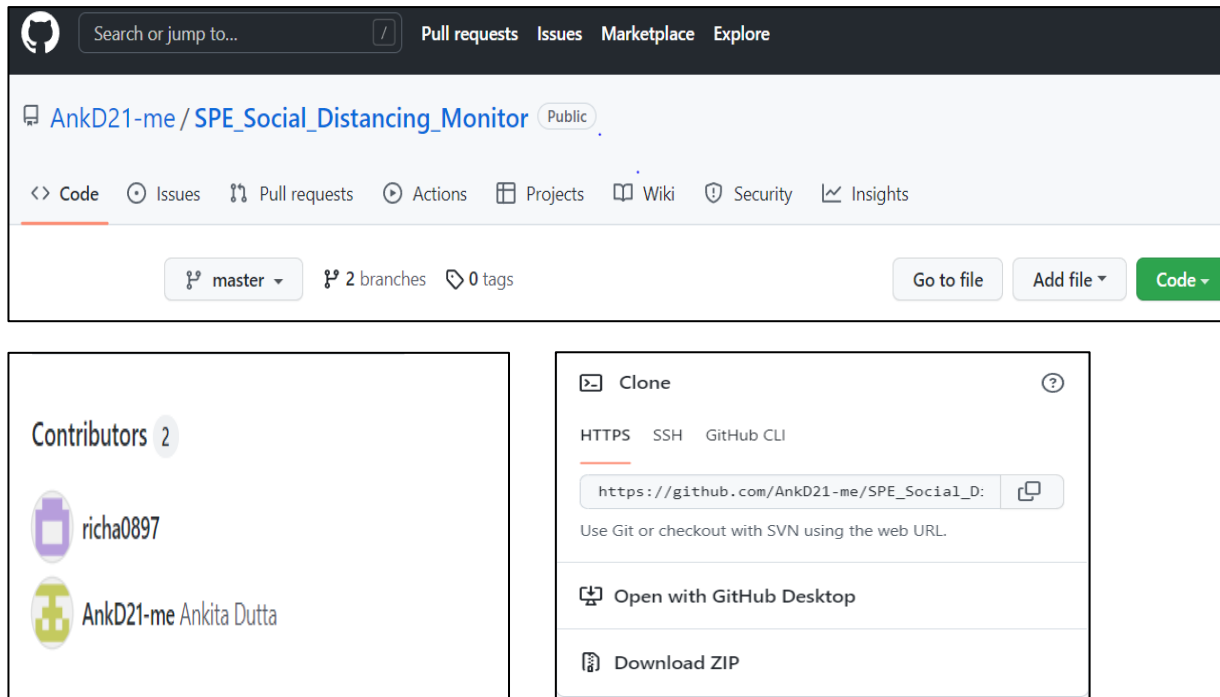
`git commit -m "message while committing"` - This command is used to save your changes to the local repository, with -m used to provide a concise description that helps your teammates (and yourself) understand what happened.

`git checkout master` - This command switches to the master branch

`git pull` - This command is used to update the local version of a repository from a remote.

`git merge <branch_name>` - This command is used to integrate changes from another branch.

`git push` - This command will push all the latest code to the repository.



Our Repository Link: https://github.com/AnkD21-me/SPE_Social_Distancing_Monitor

6.3 Continuous Integration with Jenkins

Continuous Integration (CI) is a development practice where developers integrate their code into a shared repository frequently, preferably several times a day. In this development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Jenkins is used as an orchestrator to integrate different stages of the DevOps Pipeline. For example, we integrate SCM, Build, Test, Release and Deploy stage using a Jenkins pipeline project. We need to install a set of plugins and make specific configurations to integrate these stages.

We need to perform the following steps to use Jenkins for our project-

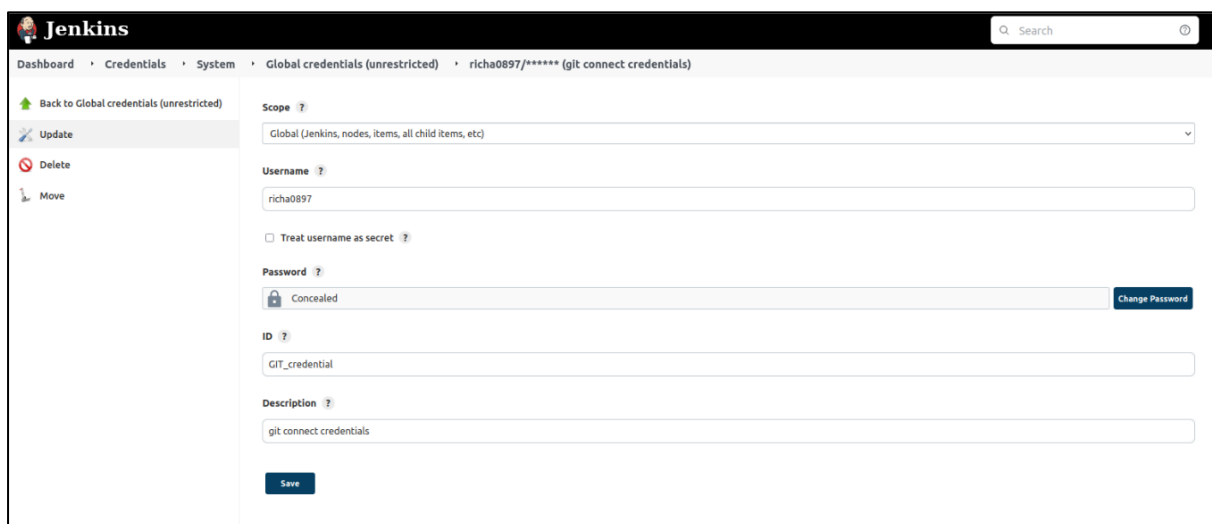
1. Start the Jenkins server
2. Integrate Jenkins with Git and GitHub. Use webhooks to trigger build on every push to the remote repository automatically
3. Integrate with Docker to push the image to Docker Hub
4. Integrate with Ansible to pull the image from Docker Hub and deploy it to the production environment.
5. Create a pipeline project and connect all the stages

6.3.1. Integrating Jenkins with GitHub:

We would want a build to be triggered automatically on every push of a new feature or any modifications to our source code to our GitHub repository. To facilitate this, the following steps need to be performed-

We need to install the Git and the GitHub plugins. To do so, we go to Manage Jenkins on the Jenkins Dashboard → Manage plugins, → Install Git Plugin, and Install GitHub Plugin

We also need to set credentials for our GitHub Repository on Jenkins; otherwise, we won't be able to connect to it. For that, we need to go to Manage Jenkins → Manage Credentials → Global Credentials → Add Credentials → Under Kind, we select Username and Password, and under Scope, we choose Global option → Enter your Github username and password → Save



The screenshot shows the Jenkins 'Add Credentials' form. The breadcrumb trail is: Dashboard > Credentials > System > Global credentials (unrestricted) > richa0897/***** (git connect credentials). The left sidebar has links: Back to Global credentials (unrestricted), Update, Delete, and Move. The main form fields are: Scope (Global (Jenkins, nodes, items, all child items, etc)), Username (richa0897), Treat username as secret (unchecked), Password (Concealed), ID (GIT_credential), and Description (git connect credentials). A 'Save' button is at the bottom.

We also need to add a path to the git executable file. We can use the command 'which git' to find out this path. We go to Manage Jenkins → Global Tool Configuration → Under Git; we add the path.



The screenshot shows the Jenkins 'Git' configuration page. The breadcrumb trail is: Dashboard > Manage Jenkins > Global Tool Configuration > Git. The left sidebar has links: Git Installations, Git, and Name. The main form fields are: Name (Default), Path to Git executable (git), and Install automatically (unchecked). A 'Delete Git' button is at the bottom right.

To trigger build on every push to the repository, we need to use the GitHub hook trigger for GITScm Polling. A webhook is an outbound HTTP request that helps you create a relationship between your GitHub repository and a particular URL, in this case, a

pipeline. After you configure the webhook, changes you make in your GitHub repository will trigger the pipeline and its associated tasks.

It's common to use GitHub webhook when Jenkins is hosted on any cloud. But if Jenkins is running on localhost, the webhook doesn't work. So, for such cases, ngrok comes into play. In addition, ngrok is an excellent tool for creating a tunnel to a person's IP. We use this if we want to securely expose our local webserver to the internet and capture all traffic for detailed inspection and replay.

To start using ngrok, we perform the following steps. Here we use port 8080 instead of port 80.

1 Download ngrok
ngrok is easy to install. Download a single binary with zero run-time dependencies.
[Mac OS X](#) [Windows](#)
Download for Linux
[Mac \(32-Bit\)](#) [Windows \(32-Bit\)](#) [Linux \(ARM\)](#)
[Linux \(32-Bit\)](#) [FreeBSD \(64-Bit\)](#)
[FreeBSD \(32-Bit\)](#)

2 Unzip to install
On Linux or OSX you can unzip ngrok from a terminal with the following command. On Windows, just double click ngrok.zip.

```
$ unzip /path/to/ngrok.zip
```


Most people keep ngrok in their user folder or set an alias for easy access.

3 Connect your account
Running this command will add your account's authtoken to your ngrok.yml file. This will give you more features and all open tunnels will be listed here in the dashboard.

```
$ ./ngrok authtoken 2uTLB8h4Uae19yuHCK1A
```

4 Fire it up
Read [the documentation](#) on how to use ngrok. Try it out by running it from the command line:

```
$ ./ngrok help
```


To start a HTTP tunnel on port 80, run this next:

```
$ ./ngrok http 80
```

```
ngrok by @inconshreveable

Session Status      online
Account             varma.richavarma@gmail.com (Plan: Free)
Version             2.3.40
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://0e45-103-156-19-229.ngrok.io -> http://localhost:8080
                    https://0e45-103-156-19-229.ngrok.io -> http://localhost:8080

Connections         ttl    opn    rt1    rt5    p50    p90
                    4      0      0.00   0.01   6.31   9.66

HTTP Requests
-----
GET /favicon.ico           200 OK
GET /static/de61580b/css/simple-page-variables.css 200 OK
GET /static/de61580b/images/svg/logo.svg         200 OK
GET /static/de61580b/css/simple-page.theme.css     200 OK
GET /static/de61580b/css/simple-page-forms.css     200 OK
```

In the unpaid version of ngrok, the generated IP is temporary, so one will need to change the IP details at the required locations from time to time.

To enable webhook, we perform the following steps-

1. Go to the GitHub repository page in the web browser.
2. Click the Settings tab.
3. In the navigation pane, click WebHooks.

4. Click Add Webhook.
5. In the Payload URL field, paste the URL generated by ngrok followed by /github-webhook/
6. In the Content-type field, select application/json.
7. Below Which events would you like to trigger this webhook select Push to trigger build on push to the repository.
8. Ensure that the Active check box is selected. This option keeps the webhook enabled and sends notifications whenever an event is triggered.
9. Click Add webhook to complete the webhook configuration in GitHub Enterprise.

The screenshot shows the 'Manage webhook' page in GitHub. The left sidebar contains a navigation menu with options like General, Access, Moderation options, Code and automation, Webhooks (selected), Environments, Pages, Security, and Integrations. The main content area is titled 'Webhooks / Manage webhook' and has two tabs: 'Settings' and 'Recent Deliveries'. The 'Settings' tab is active, showing fields for 'Payload URL' (https://0e45-103-156-19-229.ngrok.io/github-webhook/), 'Content type' (application/json), and 'Secret'. Below these are 'SSL verification' options (Enable SSL verification is selected) and 'Which events would you like to trigger this webhook?' (Just the push event is selected). At the bottom, the 'Active' checkbox is checked, with a note: 'We will deliver event details when this hook is triggered.'

The screenshot shows the 'Webhooks' list page in GitHub. The left sidebar is the same as the previous screenshot. The main content area is titled 'Webhooks' and has an 'Add webhook' button in the top right. Below the title, there is a description: 'Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.' A table lists the configured webhooks, showing a single entry with a green checkmark, the URL 'https://0e45-103-156-19-229.ngrok.io/github-webhook/', and the event 'push'. The entry has 'Edit' and 'Delete' buttons next to it.

We need to make changes to the Jenkins URL to receive the payload. We go to Manage Jenkins→Configure Systems→Under Jenkins URL; we specify the IP generated using ngrok. Under Github Server→, we select manage hooks, and under the advanced option, we override the hook trigger with the payload URL, which is the same as what was specified on GitHub.

Dashboard • configuration

☐ Restrict project naming

Jenkins Location

Jenkins URL [?](#)

System Admin e-mail address [?](#)

Serve resource files from another domain

Resource Root URL [?](#)

URLs of resource files (CSS, JavaScript, etc.) will be served from the Jenkins URL. For best results, please use the same domain as the Jenkins URL.

Override Hook URL

☒ Specify another hook URL for GitHub configuration

Shared secrets

Additional actions [?](#)

6.3.2 Building Flask Application and Testing it with unittest:

For our flask application to work, we need to run three python scripts, app.py, social_distance_det.py, and main.py, and in that order.

For testing our application, we have used the unittest framework of python. We can have a single function, a class, or an entire module as the unit for testing. Unit testing aims to validate the smallest unit first and then gradually move to other units. The unittest in python is inspired by Java's JUnit.

Testing frameworks typically hook into your test's assertions so that they can provide information when an assertion fails. For example, the unittest module provides several helpful assertion utilities for testing.

We had to perform the following steps for testing-

1. Import the TestCase class from unittest
2. Create TryTesting, a subclass of TestCase
3. Write a method in TryTesting for each test
4. Use one of the self.assert* methods from unittest.TestCase to make assertions

Using the unittest framework, there are three possible outputs – OK, FAIL, and ERROR.

1. OK: This is the output when all the tests are passed.
2. FAILED: This is the output shown when all the tests did not pass successfully. For example, the output shows FAIL when an AssertionError exception is raised.
3. ERROR: This was the output when all the test cases did not pass, and it raised an exception other than the AssertionError.

```
(venv) (base) ank@ankd:~/Desktop/spe_major_sdm$ python3 test.py
...rvarma rv123
...
-----
Ran 6 tests in 0.077s

OK
```

```
tests passed (100%)
✓ spe_major_sdm
✓ test.py
✓ FlaskTest
  ✓ test_index
  ✓ test_login
  ✓ test_logout
  ✓ test_dashboard
  ✓ test_upload
  ✓ test_connection
```

```
def test_logout(self):
    tester= app.test_client(self)
    response = tester.get("/logout",follow_redirects=True)
    status_code=response.status_code
    self.assertEqual(status_code,200)
    assert response.request.path=='/login'
```

```
def test_dashboard(self):
    tester= app.test_client(self)
    response = tester.get("/dashboard")
    status_code=response.status_code
    self.assertEqual(status_code,200)
```

6.3.3. Docker Management

Docker is an open-source centralized platform designed to make, deploy, and run applications. Docker uses containers on the host's operating system to run applications. Instead of creating an entire virtual OS, it allows applications to use an equivalent Linux kernel as a system on the host computer. Containers ensure that our application works in any environment like development, test, or production.

We create the docker image of our application and push it to DockerHub using Jenkins. For this, we need to integrate Docker with Jenkins.

We need to containerize our application. So, we build an image of our application containing information about all the necessary packages, files, folders, and commands required to run the application on deployment. We release this image on the docker hub. To integrate Docker with Jenkins, we need to add docker hub credentials to

The screenshot shows the Jenkins 'Global credentials (unrestricted)' configuration page for a user named 'richavarma'. The page is titled 'richavarma/***** (docker hub account creds)'. On the left, there are navigation links: 'Back to Global credentials (unrestricted)', 'Update', 'Delete', and 'Move'. The main form contains the following fields:

- Scope:** A dropdown menu set to 'Global (Jenkins, nodes, items, all child items, etc)'.
- Username:** A text input field containing 'richavarma'.
- Treat username as secret:** An unchecked checkbox.
- Password:** A text input field with a lock icon and the word 'Concealed'. A 'Change Password' button is to the right.
- ID:** A text input field containing 'credentials-id'.
- Description:** A text input field containing 'docker hub account creds'.
- Save:** A blue button at the bottom of the form.

Jenkins. For this, we go to Manage Jenkins→Manage Credentials→Global→Add Credentials→Fill in the required details as given in the image below-

To build the image, we need one Dockerfile, which will be at the same level as our source folder.

Dockerfile

```
# Use an official Python runtime as an image
FROM python:3.6

# The EXPOSE instruction indicates the ports on which a container
# will listen for connections
# Since Flask apps listen to port 5000 by default, we expose it
EXPOSE 5000

# Sets the working directory for following COPY and CMD instructions
# Notice we haven't created a directory by this name - this instruction
# creates a directory with this name if it doesn't exist
WORKDIR /app

#Installing yolov3 wts
RUN wget https://pjreddie.com/media/files/yolov3.weights

# Install any needed packages specified in requirements.txt
COPY requirements.txt /app
COPY app.py /app
COPY main.py /app
COPY social_distance_det.py /app
COPY test.py /app
COPY templates /app/templates
COPY static /app/static
COPY yolo-coco /app/yolo-coco
COPY Args_Folder /app/Args_Folder
RUN pip install -r requirements.txt
RUN cp yolov3.weights /app/yolo-coco/
```

1. We have used the official python runtime python 3.6 as our base image
2. We expose port 5000 for running our flask application
3. We set the default working directory of our container to /app
4. We need to install the yolov3 weights here. This is because the yolov3.weights file is too large to be pushed to the GitHub repository. So to get the weights file for our application, we need to download it from the internet while building the application image. We can find the weights file inside the container created from this image.
5. We copy the requirements.txt in our remote server to the /app directory in the container. Other files and associated directories are also copied
6. We install all the required imports for our application from the requirements.txt file using pip install. This sets up the environment to run our application inside the container created from this image
7. We copy the yolov3.weights to the /app/yolo-coco folder path in the last line.

Docker-Compose

```
version: '3.3'
services:
  app:
    image: richavarma/mlops_repo:latest
    command: >
      sh -c "python app.py wait_for_mysqldb &&
        python social_distance_det.py &&
        python main.py"
    environment:
      - DISPLAY=:0
    depends_on:
      - mysqldb
    ports:
      - "5000:5000"
    volumes:
      - /tmp/.X11-unix:/tmp/.X11-unix
      - /home/richa/Desktop/final_proj/SPE_Social_Distancing_Monitor/sdm_logfile.log:/app/sdm_logfile.log

  mysqldb:
    image: mysql:5.7
    command: --default-authentication-plugin=mysql_native_password
    ports:
      - "5423:5423"
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_ROOT_HOST=%
    volumes:
      - ./db:/docker-entrypoint-initdb.d:/ro
    tty: true

  elasticsearch:
    container_name: elasticsearch
    image: elasticsearch:7.7.1
    environment:
      - bootstrap.memory_lock=true
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
```

1. We have used Docker compose version 3.3
2. We have used six services in our docker-compose, namely app, mysqldb, Elasticsearch, Logstash, Kibana and filebeat
3. We have pulled the image of our app, MySQL, Elasticsearch, and Kibana from the DockerHub
4. Our app depends on the mysqldb container, and so that is spun up first
5. This is followed by running the python files in the app container to get our flask application started
6. The port 5000 is exposed to talk to the container from outside the container
7. For the mysqldb we expose port 5423. We put the init.sql script to docker-entrypoint-initdb.d so that the database and tables get created with some initial data.
8. We also use volumes to persist the data stored in different containers and pass the environment variables to the container as required.
9. The descriptions of other containers required for monitoring are mentioned under the monitoring tag in this report

DockerHub Link: https://hub.docker.com/repository/docker/richavarma/mlops_repo

6.3.4. Continuous Deployment with Ansible

Ansible is an open-source automation platform. It is an automation engine that runs ansible playbooks. Playbooks are defined tasks where we define environments and workflows. There are two types of machines in the Ansible architecture: control nodes and managed hosts. Ansible is installed and run from a control node, and this machine also has copies of your Ansible project files.

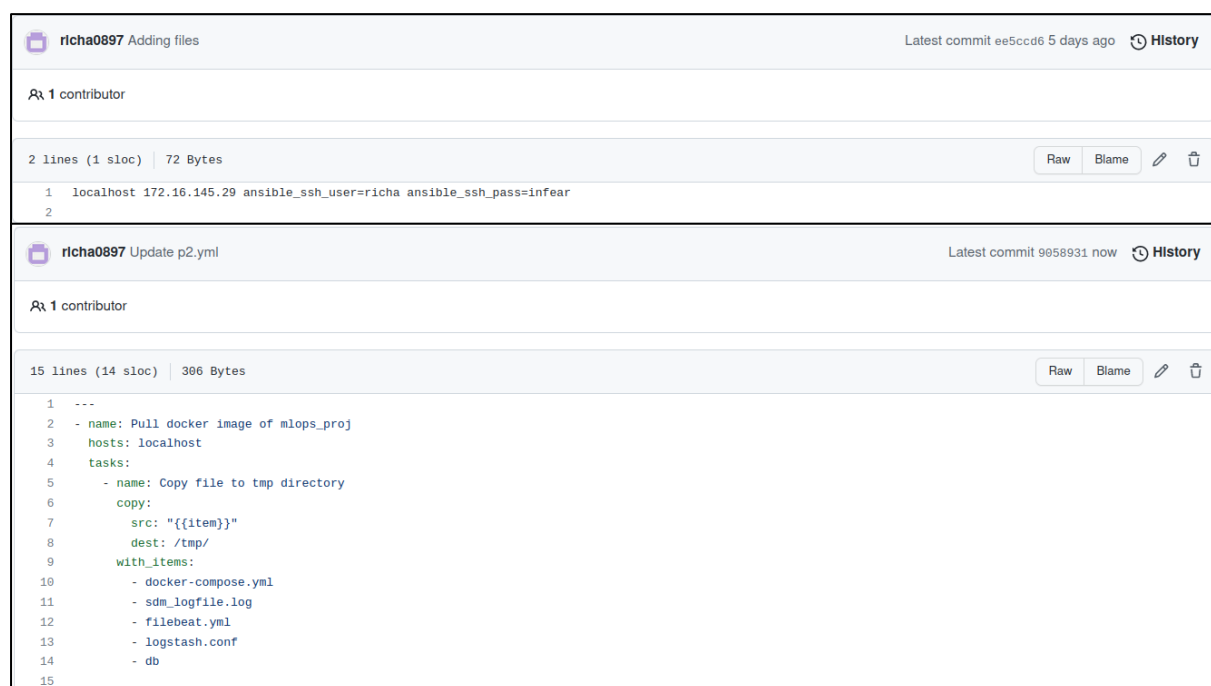
Managed hosts are listed in an inventory, which organizes those systems into groups for easier collective management. The inventory can be defined in a static text file or dynamically determined by scripts that get information from external sources.

Playbooks are the files where Ansible code is written. Playbooks are written in YAML format. YAML stands for Yet Another Markup Language. Playbooks are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks.

Working with Ansible:

For Ansible to work, make sure you have python and ssh server installed on your controller and managed nodes.

The contents of our playbook and inventory files can be seen below-



```
richa0897 Adding files
Latest commit ee5ccd6 5 days ago History
1 contributor
2 lines (1 sloc) | 72 Bytes
1 localhost 172.16.145.29 ansible_ssh_user=richa ansible_ssh_pass=infeare
2

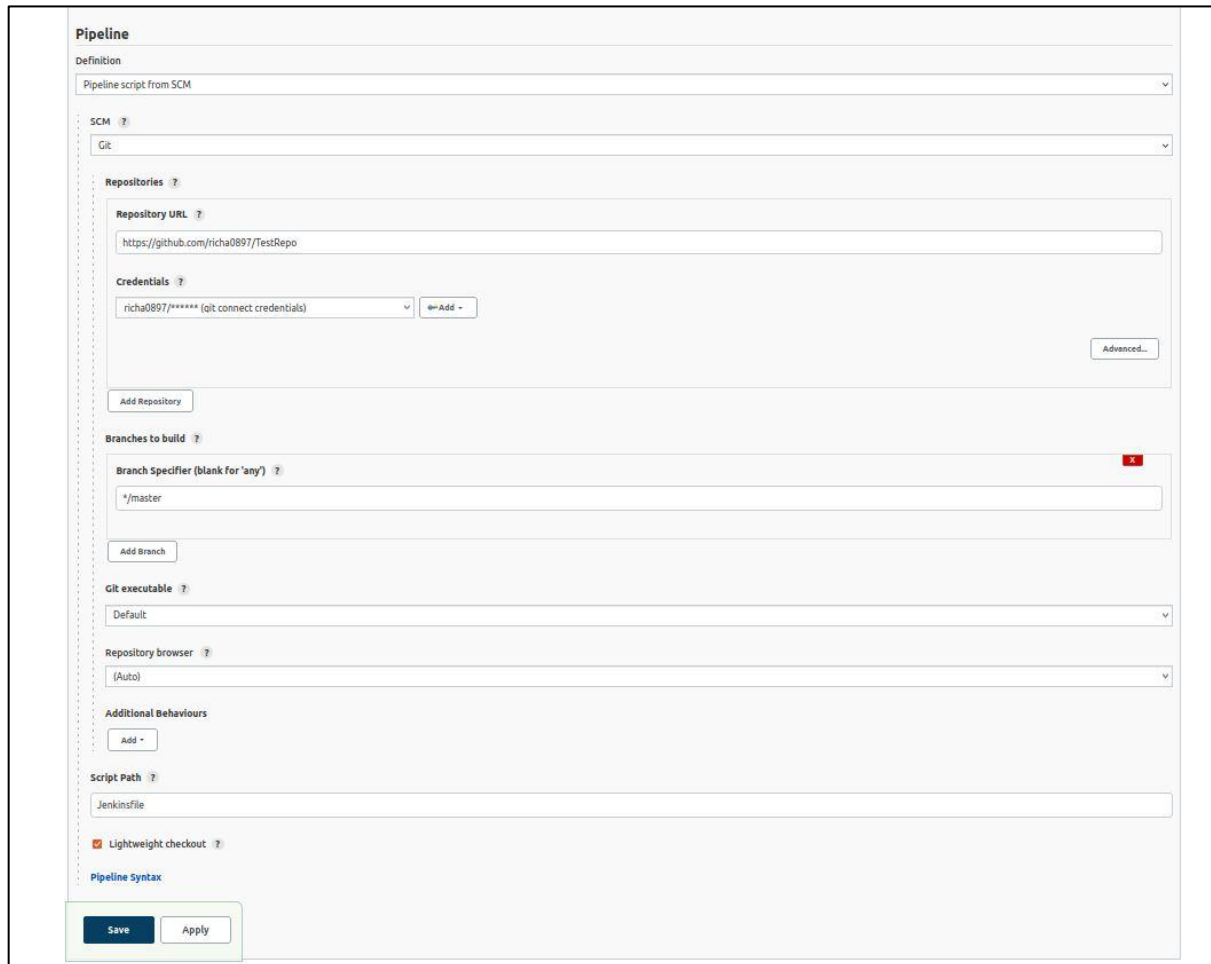
richa0897 Update p2.yml
Latest commit 9058931 now History
1 contributor
15 lines (14 sloc) | 306 Bytes
1 ---
2 - name: Pull docker image of mlops_proj
3   hosts: localhost
4   tasks:
5     - name: Copy file to tmp directory
6       copy:
7         src: "{{item}}"
8         dest: /tmp/
9       with_items:
10        - docker-compose.yml
11        - sdm_logfile.log
12        - filebeat.yml
13        - logstash.conf
14        - db
15
```

Inventory file – we have deployed the files on localhost using ssh

Playbook file- We deploy only the files required to run our application on the client-side. These files include docker-compose.yml, sdm_logfile.log, filebeat.yml, logstash.conf, and the directory db containing the init.sql script for setting up our database with the required tables and some initial values. All these files are deployed under the /tmp directory.

6.3.5. Jenkins Pipeline

We need to create a pipeline project and connect the different stages of the DevOps pipeline. We go to the Jenkins Dashboard→New Item→, give a name to the item, and select pipeline project→Add item.



The screenshot shows the Jenkins Pipeline configuration page. The 'Definition' dropdown is set to 'Pipeline script from SCM'. The 'SCM' dropdown is set to 'Git'. The 'Repositories' section shows a 'Repository URL' of 'https://github.com/richa0897/TestRepo' and 'Credentials' set to 'richa0897/***** (git connect credentials)'. There is an 'Add Repository' button. The 'Branches to build' section shows a 'Branch Specifier (blank for \'any\')' of '*/master' and an 'Add Branch' button. The 'Git executable' is set to 'Default' and the 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section has an 'Add +' button. The 'Script Path' is set to 'Jenkinsfile'. The 'Lightweight checkout' checkbox is checked. At the bottom, there are 'Save' and 'Apply' buttons.

We select GitHub Hook to trigger for GitScm Polling under build triggers.

Under Pipeline Script, we choose Pipeline Script from SCM and do the following settings-

We are creating Jenkinsfile that has our pipeline syntax. The Jenkinsfile has to be pushed into our repository, and the path to the file needs to be specified in the script path. For our project, the Pipeline script can be seen below-

```
55 lines (48 sloc) | 1.14 KB
1 pipeline {
2   agent any
3   environment {
4     imageName = ""
5   }
6   stages {
7
8     stage('GIT CLONE') {
9       steps {
10        git url: 'https://github.com/AnkD21-me/SPE_Social_Distancing_Monitor.git', branch: 'master'
11      }
12    }
13
14    stage('Build') {
15      steps {
16        echo 'building'
17      }
18    }
19
20    stage('Test') {
21      steps {
22        script{
23          sh "pip install -r requirements.txt"
24          sh "python3 test.py"
25        }
26      }
27    }
28  }
```

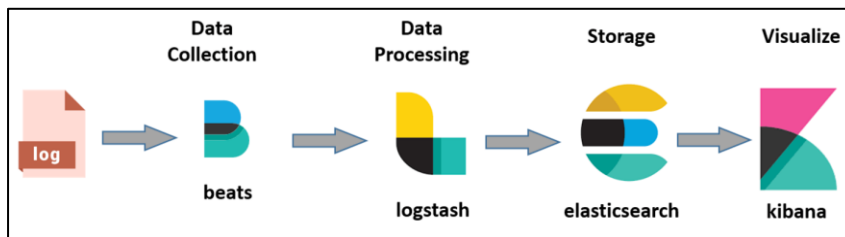
```
1 stage('Build Docker Image'){
2   steps {
3     script {
4       imageName = docker.build "richavarma/mlops_repo:latest"
5     }
6   }
7
8   stage('Push Docker Image')
9   {
10    steps {
11      script{
12        docker.withRegistry('', 'credentials-Id') {
13          imageName.push()
14        }
15      }
16    }
17  }
18
19  stage('Deploy using Ansible'){
20    steps{
21      ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'inventory', playbook: 'p2
22    }
23  }
24
25 }
```

Under pipeline syntax, we need to give the path to our inventory and playbook files in the repository for deployment using Ansible.

We apply and save all the changes. This will get our pipeline up and ready. The build gets triggered automatically on any feature push, and all the pipeline stages are executed one after the other. Finally, the image of our build is pushed to our DockerHub repository. Ansible pulls the latest image from the DockerHub and deploys it to the production environment.

6.4 Monitoring

We have used Filebeat, ElasticSearch, Logstash, and Kibana (ELK Stack) to continuously monitor the logs generated due to activities in the application.



Filebeat: This is a lightweight shipper for forwarding and centralizing log data. It can be installed as an agent on our servers. Filebeat monitors the log files, collect logs and forward them to Elasticsearch or Logstash for indexing.

Logstash: This is an open-source, lightweight, server-side data processing pipeline that allows us to collect data from different sources, transform it on the fly, and send it to your desired destination. It is often used as a data pipeline for Elasticsearch.

Elasticsearch: This is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

Kibana: This data visualization and exploration tool is used for application monitoring and log and time-series analytics cases. It offers powerful and easy-to-use features such as histograms, line graphs, pie charts, heat maps, etc.

To begin with, we have pulled the Filebeat, ElasticSearch, Logstash, and Kibana images from DockerHub.

We update the docker-compose file to include the ELK stack services and set up volumes. The first volume is used to fetch the log file generated in the application container and make it persist in the volume.

```
volumes:
  - /tmp/.X11-unix:/tmp/.X11-unix
  - ./mylog:/sdm_logfile.log
```

Once the log file is available in the volume, Filebeat can read from the volume to begin the monitoring pipeline.

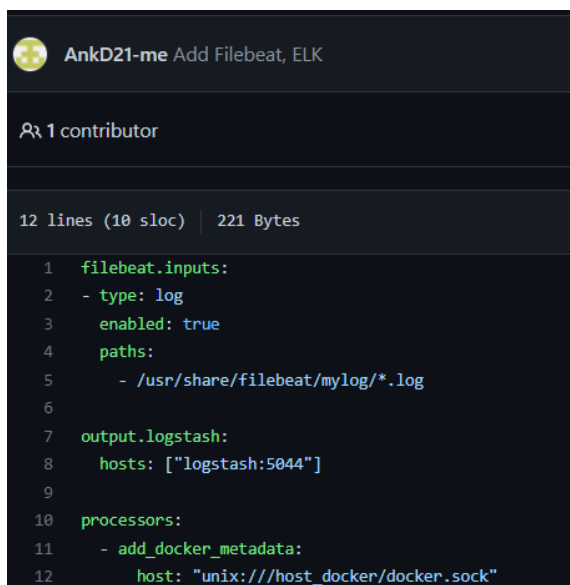
First, we have to set up the Filebeat service in docker-compose.yml and mount the same volume location where the application container writes the logs.

```

filebeat:
  user: root
  container_name: filebeat
  image: docker.elastic.co/beats/filebeat:7.7.1
  links:
    - logstash:logstash
  depends_on:
    - logstash
  volumes:
    - /var/run/docker.sock:/host_docker/docker.sock
    - /var/lib/docker:/host_docker/var/lib/docker
    - ./mylog:/usr/share/filebeat/mylog
    - ./filebeat.yml:/usr/share/filebeat/filebeat.yml

```

We have created the filebeat.yml file to specify the input and output path. The Filebeat container reads the log file from the volume and stores it in the below-mentioned path under filebeat.inputs. The logs are passed onto logstash, which listens at port 5044. Additionally, we have set up the processor to pass on docker metadata.



```

1  filebeat.inputs:
2    - type: log
3      enabled: true
4      paths:
5        - /usr/share/filebeat/mylog/*.log
6
7  output.logstash:
8    hosts: ["logstash:5044"]
9
10 processors:
11   - add_docker_metadata:
12     host: "unix:///host_docker/docker.sock"

```

The next step is to configure the logstash.conf file. This includes the input path, filters (optional), and the output path. Here the output path is elasticsearch which listens at port 9200. Additionally, we also redirect the logstash output to the console using stdout to verify if the logstash is working correctly.

 AnkD21-me Add Filebeat, ELK

1 contributor

18 lines (15 sloc) | 162 Bytes

```
1  input {
2    beats {
3      port => 5044
4    }
5  }
6
7  filter {
8
9  }
10
11 output {
12   elasticsearch {
13     hosts => ["elasticsearch:9200"]
14   }
15   stdout {
16     codec => rubydebug
17   }
18 }
```

The logstash service in docker-compose is set up as follows:

```
76  logstash:
77    container_name: logstash
78    image: logstash:7.7.1
79    ulimits:
80      memlock:
81        soft: -1
82        hard: -1
83    volumes:
84      - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
85    ports:
86      - 5044:5044
87    links:
88      - elasticsearch:elasticsearch
89    depends_on:
90      - elasticsearch
91    stdin_open: true
92    tty: true
93    network_mode: bridge
94    logging:
95      driver: "json-file"
96      options:
97        max-size: "10m"
98        max-file: "50"
```


The critical aspect here is that the volume is set up to send the logstash.conf file to the logstash container. Also, the service is linked to the elasticsearch service, which is next in our monitoring pipeline.

Now we add the elasticsearch service to docker-compose.yml.

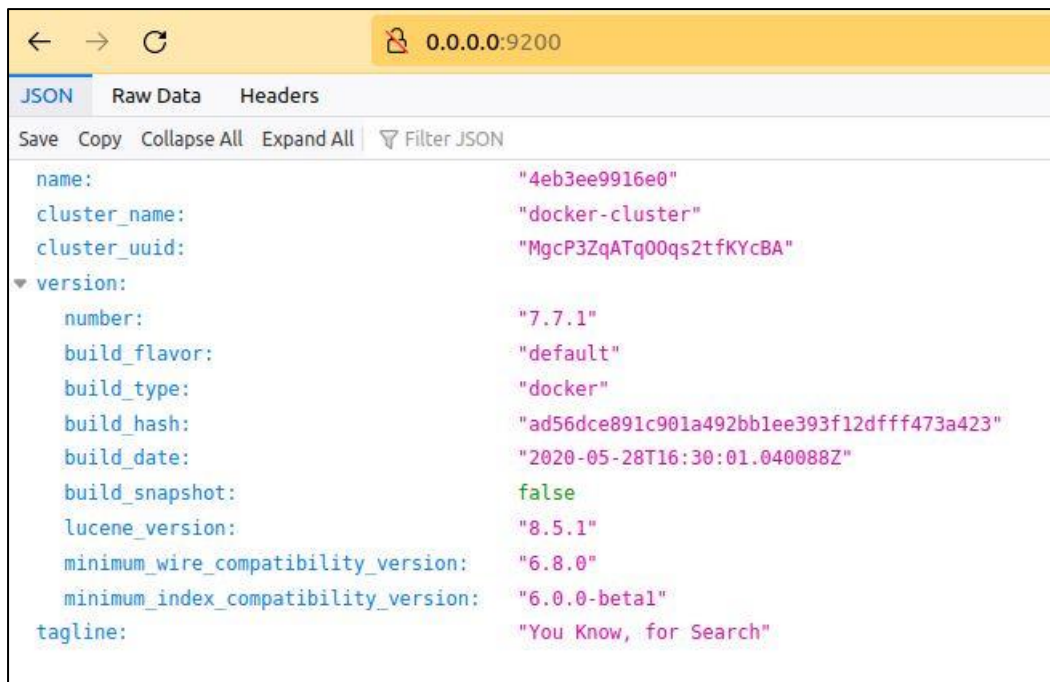
```
elasticsearch:
  container_name: elasticsearch
  image: elasticsearch:7.7.1
  environment:
    - bootstrap.memory_lock=true
    - discovery.type=single-node
    - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
  ulimits:
    memlock:
      soft: -1
      hard: -1
  ports:
    - 9200:9200
    - 9300:9300
  stdin_open: true
  tty: true
  network_mode: bridge
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "50"
```

Finally, we add the Kibana service to docker-compose.yml. This will be responsible for the visualization of our log. We are setting the dependency as elasticsearch to get the visualization.

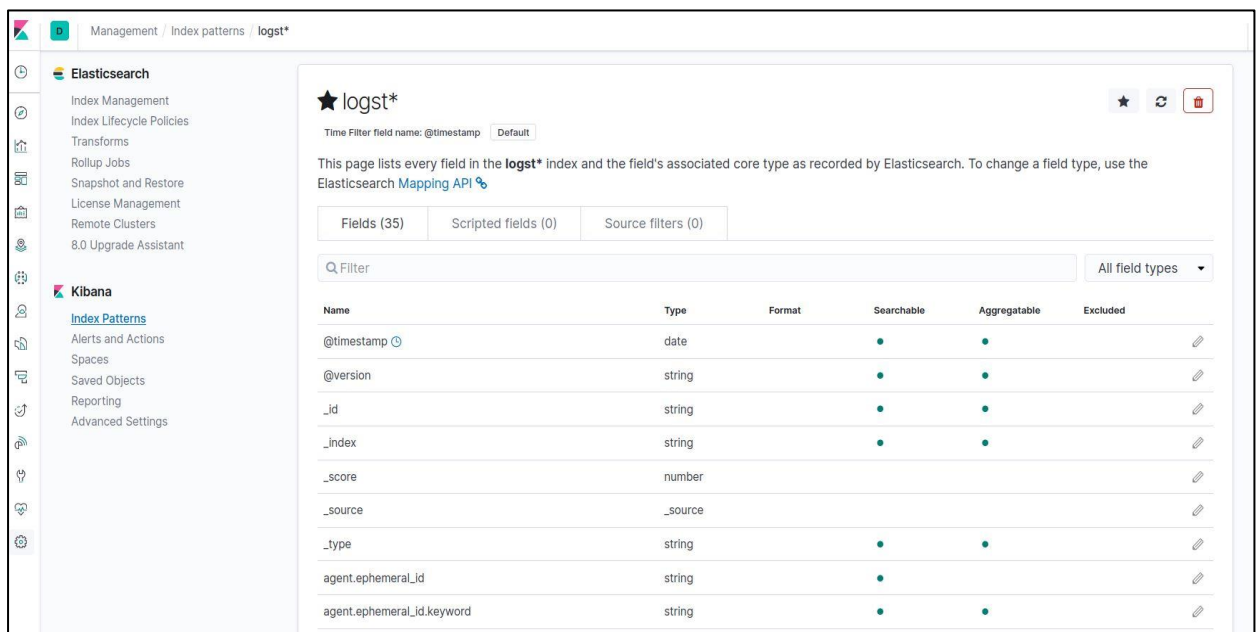
As the setup is complete now, we can do: `sudo docker-compose up` to check if the services are started correctly.

```
kibana:
  container_name: kibana
  image: kibana:7.7.1
  ulimits:
    memlock:
      soft: -1
      hard: -1
  ports:
    - 5601:5601
  environment:
    - ELASTICSEARCH_URL=http://elasticsearch:9200
  links:
    - elasticsearch:elasticsearch
  depends_on:
    - elasticsearch
  stdin_open: true
  tty: true
  network_mode: bridge
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "50"
```

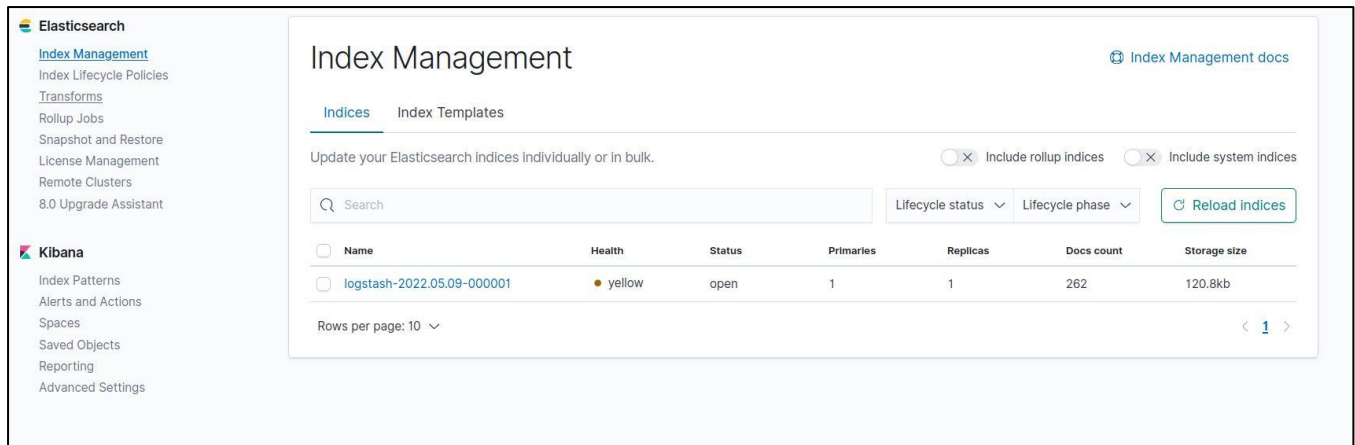
Elasticsearch is running at 9200 port.



Now we can go to the Management tab in Kibana and look for the index pattern obtained from elasticsearch. It will be available under the create index pattern option. After setup, we will get the below screen with all the field types available in our logs.

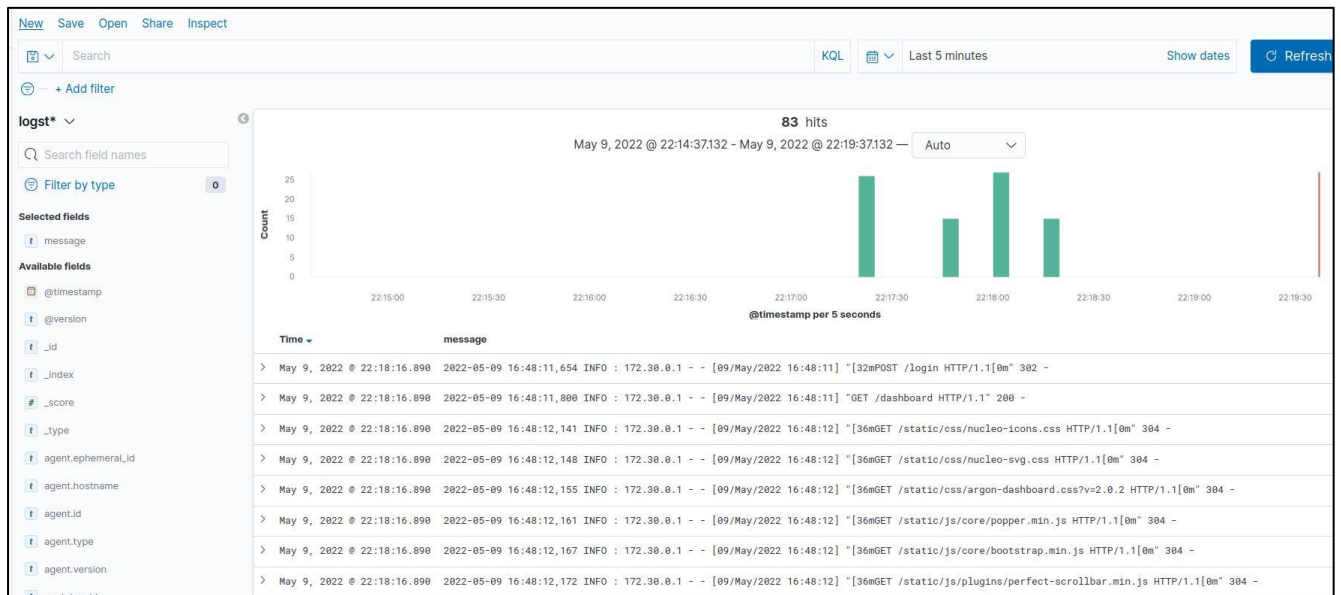


We can verify the status of the index under elasticsearch's index management option.



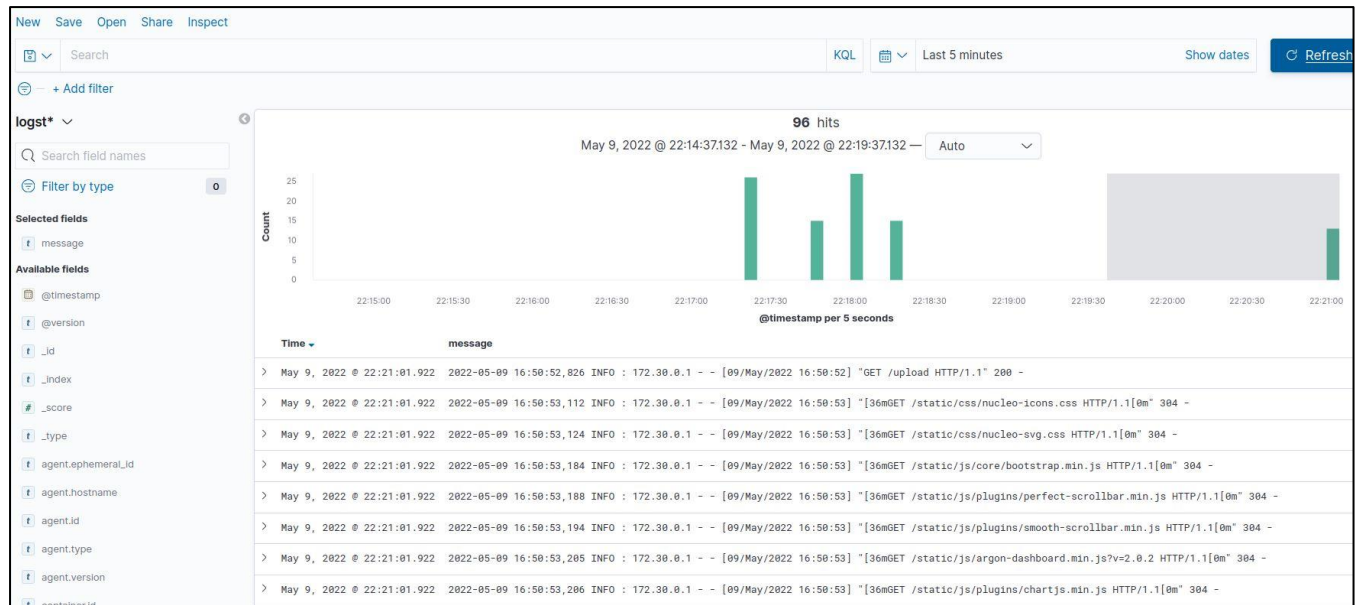
Now we are ready to visualize our logs.

To demonstrate continuous monitoring, we first visualize the current logs in the `sdm_logfile.log` generated in the application container.

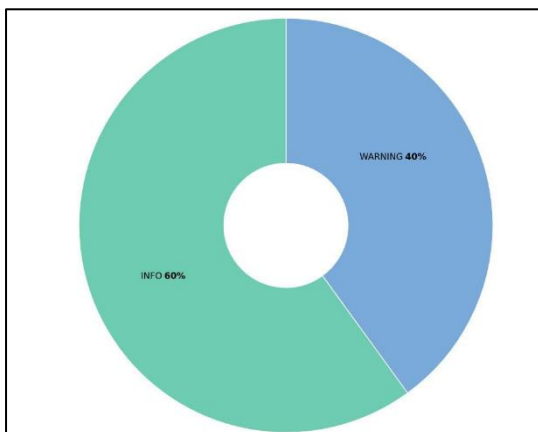


Currently, we have 83 hits, and we've filtered the "message field"; hence the messages are displayed in the dashboard.

Now we go to the upload page of our application, which will generate new logs. To see that the log stream automatically gets fetched by Filebeat and is being received at Kibana for visualization, we click 'Refresh'. We can see in the Kibana dashboard that the number of hits has been updated to 96 and that the latest message is GET /upload, which is in line with the action performed at the application frontend.



Pie chart: Log Level Distribution



This visualization was done immediately after launching the application. Hence the distribution is as seen above. With navigating through the application, the distribution gets updated.

7. API Documentation And Code Walkthrough

END POINT	HTTP METHOD	DESCRIPTION
0.0.0.0/5000/	POST	Renders the login page
0.0.0.0/5000/login	POST	Checks login data with database. On successful login redirect to dashboard else render the login form again
0.0.0.0/5000/dashboard	GET	Renders dashboard page
0.0.0.0/5000/upload	GET	For uploading the video footage
0.0.0.0/5000/logout	GET	Logs us out of the application back to the login page
0.0.0.0/5000/show_violations	GET	Fetches violation data from the database
0.0.0.0/5000/graph	GET/POST	Creates the visualisation for the violations on a day and displays the violations vs time graph

app.py: This file contains all the configuration details needed to run the program. The input folder point to the folders where the uploaded video by the admin user will be stored. The output folder holds the processed video consisting of the detected violations in the input video. In addition, the database credentials are also mentioned in this file.

```

app.py > ...
4  UPLOAD_FOLDER = 'static/uploads/'
5  OUTPUT_FOLDER = 'static/output/'
6
7  app = Flask(__name__)
8
9  app.secret_key = "secret key"
10 app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
11 app.config['OUTPUT_FOLDER'] = OUTPUT_FOLDER
12 app.config['MAX_CONTENT_LENGTH'] = 16 * 2048 * 2048
13
14 # Enter your database connection details below
15 app.config['MYSQL_HOST'] = 'mysqladb'
16 app.config['MYSQL_USER'] = 'root'
17 app.config['MYSQL_PASSWORD'] = 'root'
18 app.config['MYSQL_DB'] = 'spe_proj'
19

```

main.py: This file consists of all the API code that can be accessed by the front end, starting from login to display of violations graph. App routes included here are:

- /: This is the home route, i.e., the page we land upon launching the application.
- /login: The username and password are verified here and set the session as true.

```

@app.route('/')
def home():
    return render_template('login.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    msg = ''
    if request.method == 'POST':
        db=MySQLdb.connect(app.config['MYSQL_HOST'],app.config['MYSQL_USER'],app.config['MYSQL_PASSWORD'],app.config['MYSQL_DB'])
        cursor=db.cursor()
        username = request.form['username']
        password = request.form['password']
        cursor.execute('SELECT username,password FROM admins WHERE username = %s AND password = %s', (username, password))
        admin = cursor.fetchone()
        print(username,password)
        if admin:
            session['loggedin'] = True
            session['username'] = admin[0]
            return redirect('/dashboard')
        else:
            msg = 'Incorrect username/password!'
    return render_template('login.html', error=msg)

```

- /logout: Log out of the application and pop out the session.
- /dashboard: Render the dashboard page.
- /upload: Render the upload page. When the post method is set, the input video is sent to the backend for violation detection, and the social detector program is called to begin video processing. The processing frame is displayed in a pop-up frame.


```

@app.route('/logout')
def logout():
    session.pop('loggedin', None)
    session.pop('username', None)
    if os.path.isfile(os.path.join(app.config['OUTPUT_FOLDER'], 'outputgraph.jpg')):
        os.remove(os.path.join(app.config['OUTPUT_FOLDER'], 'outputgraph.jpg'))
    # Redirect to login page
    return redirect(url_for('login'))

@app.route('/dashboard')
def dashboard():
    return render_template('dashboard.html')

@app.route('/upload')
def upload_form():
    return render_template('upload.html')

```

```

@app.route('/upload', methods=['POST'])
def upload_video():
    if 'file' not in request.files:
        flash('No file part')
        return redirect(request.url)
    file = request.files['file']
    if file.filename == '':
        flash('No video selected for uploading')
        return redirect(request.url)
    else:
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        print(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        print('upload_video filename: ' + filename)
        print(os.path.join(app.config['OUTPUT_FOLDER'], filename))
        flash('Video successfully uploaded and displayed below')
        net,ln,LABEL=instantiate_model()
        model_run(os.path.join(app.config['UPLOAD_FOLDER'],filename),os.path.join(app.config['OUTPUT_FOLDER'], filename))
        print("Model Finished Processing")
        return render_template('upload.html', filename=filename)

```

- /show_violations: Fetches the threshold set by the admin user and fetches the database entries with the number of violations greater than the threshold.

```

@app.route('/show_violations', methods=['GET', 'POST'])
def show_violations():
    violations = request.form['violations']
    db=MySQLdb.connect(app.config['MYSQL_HOST'],app.config['MYSQL_USER'],app.config['MYSQL_PASSWORD'],app.config['MYSQL_DB'])
    cursor=db.cursor()
    cursor.execute('SELECT * FROM violation_db where no_of_violations >= %s and DATE(created_at) = CURDATE()',(violations,))
    result = cursor.fetchall
    timing = []
    no_vio = []
    for i in cursor:
        timing.append(i[0])
        no_vio.append(i[1])
    print("Time = ", timing)
    print("No. of Violations = ", no_vio)
    vio_df = pd.DataFrame(list(zip(timing, no_vio)), columns = ['Time', 'No. of Violations'])
    table = vio_df.to_html(classes='table table-striped')
    return render_template("dashboard.html",table=table)

```

- /graph: Visualize the violations of the current day.

```
@app.route('/graph', methods=['GET', 'POST'])
def graph():
    db=MySQLdb.connect(app.config['MYSQL_HOST'],app.config['MYSQL_USER'],app.config['MYSQL_PASSWORD'],app.config['MYSQL_DB'])
    cursor=db.cursor()
    cursor.execute('SELECT TIME(created_at), no_of_violations from violation_db where DATE(created_at) = CURDATE();')
    result = cursor.fetchall
    timing = []
    no_vio = []
    for i in cursor:
        timing.append(str(i[0]))
        no_vio.append(int(i[1]))
    print("Time = ", type(timing[0]))
    print("Time = ", timing)
    print("No. of Violations = ", no_vio)
    plt.figure(figsize=(10,6), tight_layout=True)
    ax = plt.subplot(111)
    #plotting
    ax.plot(timing,no_vio, 'o-', linewidth=2)
    #customization
    plt.xticks(range(len(timing)), timing, size='small')
    plt.xlabel('Time of Violation')
    plt.ylabel('No. of Violations')
    plt.title('All Violations Today')
    l = plt.fill_between(timing, no_vio)
```

The logging filename and format are defined here. Then, the logfile is created automatically and logs are generated on running the application.

```
import logging

logging.basicConfig(filename='sdm_logfile.log', level=logging.INFO, format=f'%(asctime)s %(levelname)s : %(message)s')
```

We are finally setting the port number as 5000 to run the application.

```
if __name__ == "__main__":
    app.run(port=5000, debug=True)
```

detection.py: This file consists of the logic to detect person class in the input video frames. Below is the snippet of the code from the file.

```
def detect_people(frame, net, ln, personIdx=0):
    # grab the dimensions of the frame and initialize the list of
    # results
    (H, W) = frame.shape[:2]
    results = []

    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
    net.setInput(blob)
    layerOutputs = net.forward(ln)
```

social_distance_det.py: Once we have detected the people in detection.py, we use multiple functions to check the social distancing violations.

check_violations() takes the centroid of people pairwise, and if the distance is lesser than the threshold, we add them to our violate set.

```
def check_violations(results):
    violate=set()
    if len(results)>=2:
        centroids=[r[2] for r in results]
        D = dist.cdist(centroids, centroids, metric="euclidean") #distance matrix D
        for i in range(0, D.shape[0]):
            for j in range(i+1,D.shape[1]):
                if D[i, j] < config.MIN_DISTANCE:
                    violate.add(i)
                    violate.add(j)
    return violate
```

add_to_db() adds the violations to the database after applying the windowing technique as mentioned in section 3.3 of this report.

```
def add_to_db(viol_frame):
    final_data={}
    max_key=max(viol_frame.items(),key=lambda k:k[1])
    print(max_key)
    print(type(max_key))
    final_data[max_key[0]]=max_key[1]
    db =MySQLdb.connect(app.config['MYSQL_HOST'],app.config['MYSQL_USER'],app.config['MYSQL_PASSWORD'],a
    cursor=db.cursor()
    for key,value in final_data.items():
        cursor.execute('Insert into violation_db values (%s,%s)',(key,value))
        db.commit()
    db.close()
```

instantiate_model() sets up the model, weights, configuration file, and other necessary details.

```
def instantiate_model():
    labelsPath = os.path.join(config.MODEL_PATH, "coco.names")
    print(labelsPath)
    LABELS = open(labelsPath).read().strip().split("\n")
    print(LABELS)
    #loading yolov3 model and its weights
    weightsPath = os.path.join(config.MODEL_PATH, "yolov3.weights")
    configPath = os.path.join(config.MODEL_PATH, "yolov3.cfg")
    print("[INFO] loading YOLO from disk...")
    net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
    # determine only the *output* layer names that we need from YOLO
    ln = net.getLayerNames()
    ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
    #these output layers find bounding boxes at different scales, at eac
    #non maximum suppression is used to find the best bounding box
    #these layers are passed to detect_people function for making the pr
    return net,ln, LABELS
```

model_run() performs the task of processing the video and stores the detections frame by frame.

```

def model_run(input_path,output_path,display,net,ln,LABELS):
    print("[INFO] accessing video stream...")
    print(input_path)
    vs = cv2.VideoCapture(input_path)
    writer = None
    count=0
    list_viols={}

    while True:
        grabbed,frame=vs.read() #reading video frame by frame
        if not grabbed: #if frame not found i.e we have reached the end of video
            break
        count+=1
        frame=imutils.resize(frame,width=700) #resizing the frame
        results = detect_people(frame, net, ln, personIdx=LABELS.index("person")) #c
        #results contains coordinates of bounding boxes with their confidence, cent

        violations=check_violations(results)
        dateB = datetime.now()
        timeNow = dateB.strftime('%Y-%m-%d %H:%M:%S')
        list_viols[timeNow]=len(violations)
        if(count%10==0):
            add_to_db(list_viols)
            list_viols={}
            print("[INFO] One window processed...")

    if output_path!= "" and writer is None:
        # initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*'MP4V') ## fourcc = cv2.cv.CV_FOURCC('m', 'p', '4', 'v')
        writer = cv2.VideoWriter(output_path, fourcc, 25,(frame.shape[1], frame.shape[0]), True)

    if writer is not None:
        writer.write(frame)

```

Below is the snippet of the backend output when the video is being processed. The timestamp and number of violations are stored in the database.

```

<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:04', 18)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:09', 12)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:18', 14)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:20', 12)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:29', 15)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:40', 12)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:43', 10)
<class 'tuple'>
[INFO] One window processed...
('2022-05-11 01:51:50', 11)

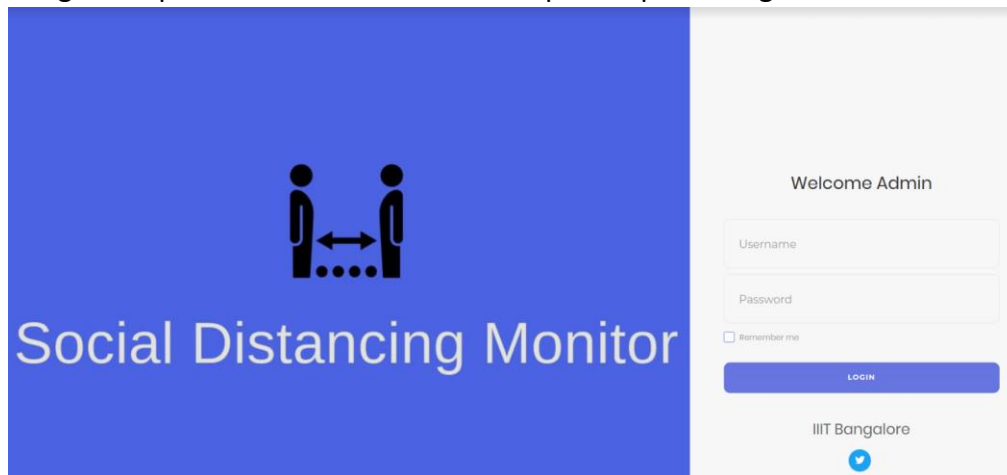
```

8. Result and Discussion

8.1 Application Interface

8.1.1. User Login Page

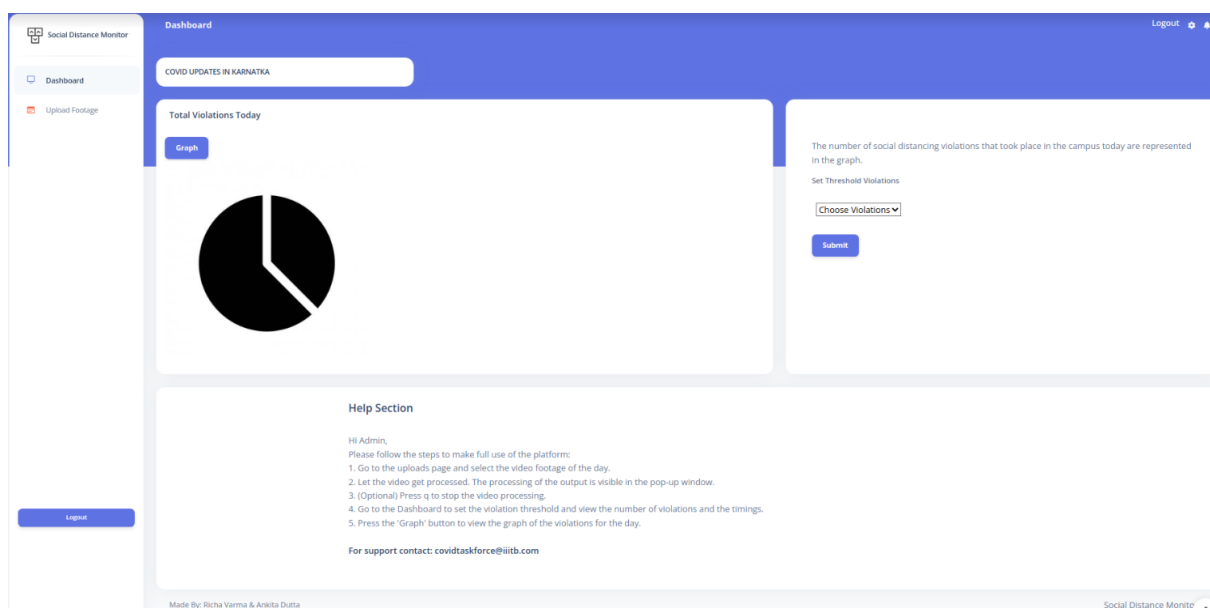
On launching the application, we land on the login page. Here the admin user can provide their credentials to log into the application. In addition, the official Twitter handle of IIIT Bangalore is provided to check the latest updates pertaining to COVID 19 and more.



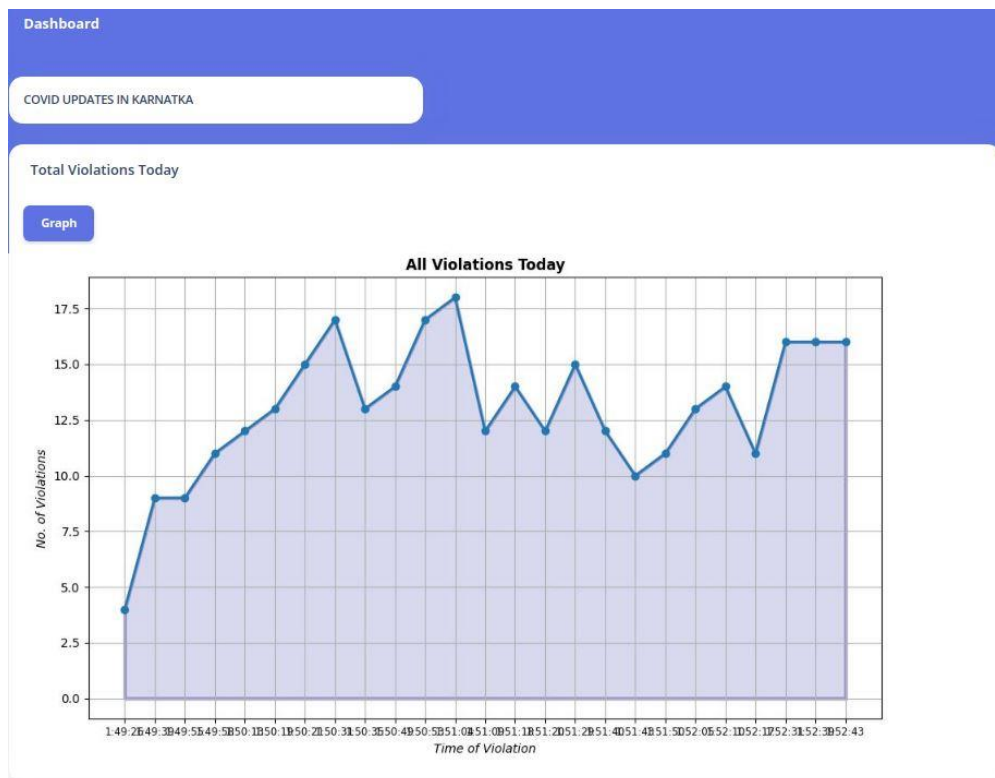
8.1.2. Dashboard

This page displays all the statistics obtained by processing the input video. It essentially offers insight to the admin user about the rate of the social distancing violations to take necessary actions.

The COVID updates section offers a link to the official page of the government of Karnataka dedicated to curbing the virus.



The graph section plots the number of violations with respect to the time stamp.



The set threshold option allows the admin user to select a threshold that they consider to be a dangerous level of social distancing violation. Then, the violation data is fetched from the database and displayed in a tabular format. The timestamp helps in understanding the time of the day witnessing high violations.

The number of social distancing violations that took place in the campus today are represented in the graph.

Set Threshold Violations

Choose Violations ▼

Choose Violations

0 Violations

10 Violations

15 Violations

20 Violations

25 Violations

30 Violations

35 Violations

40 Violations

45 Violations

50 Violations

The number of social distancing violations that took place in the campus today are represented in the graph.

Set Threshold Violations

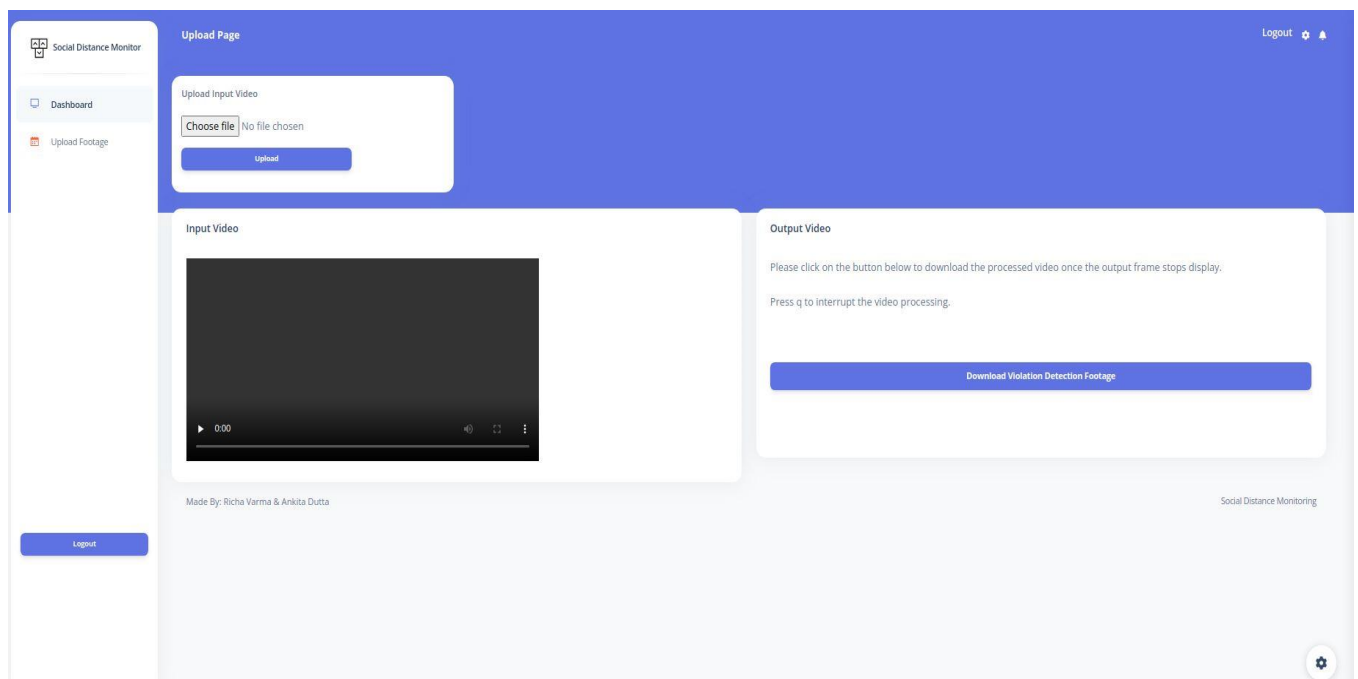
15 Violations

Submit

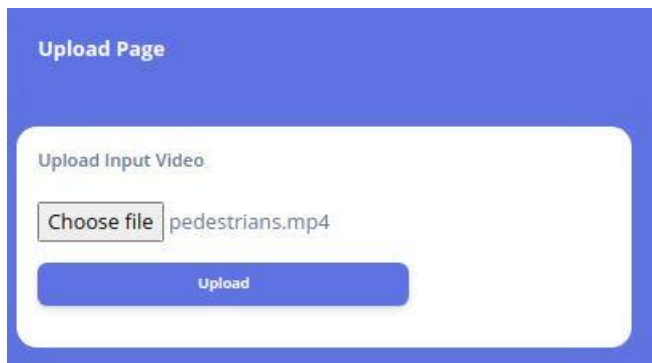
	Time	No. Of Violations
0	2022-05-11 01:50:21	15
1	2022-05-11 01:50:31	17
2	2022-05-11 01:50:53	17
3	2022-05-11 01:51:04	18
4	2022-05-11 01:51:29	15
5	2022-05-11 01:52:31	16
6	2022-05-11 01:52:39	16
7	2022-05-11 01:52:43	16

8.1.3. Upload Page

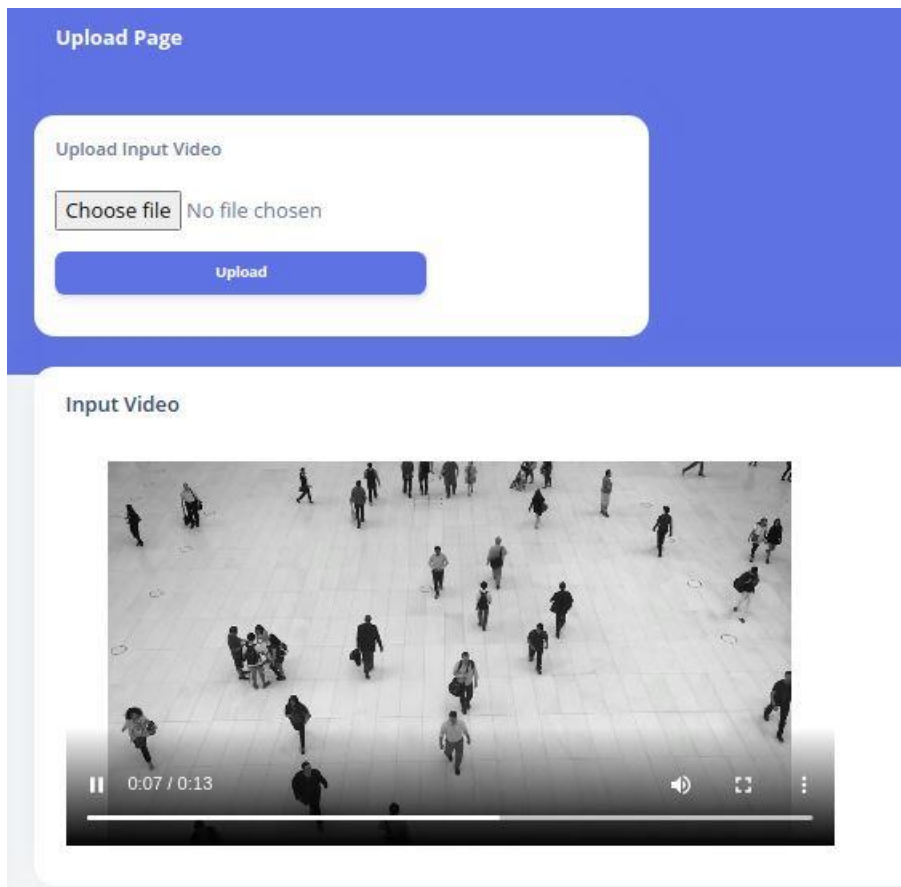
On this page, the admin user can upload the day's video footage for processing.



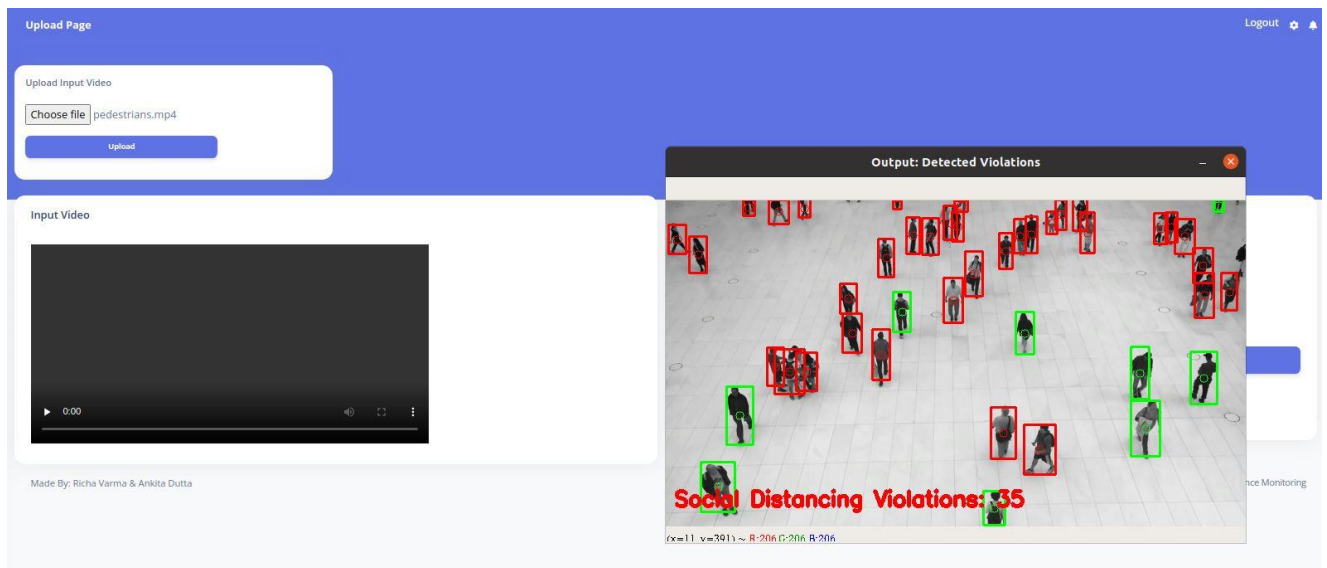
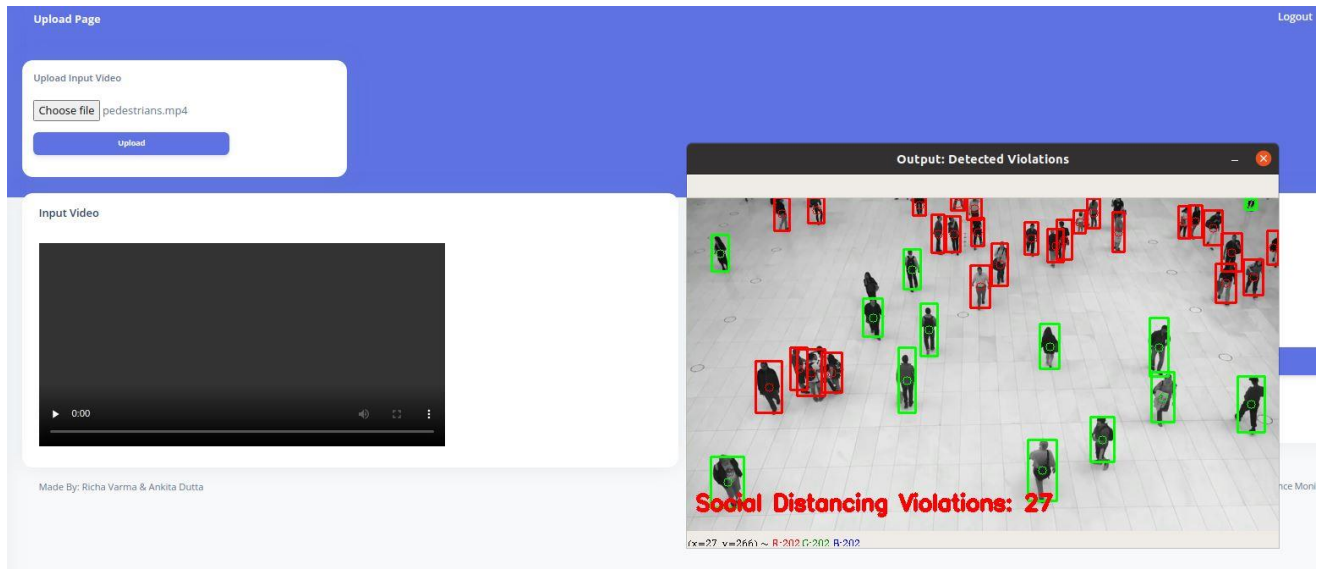
The first feature is upload video, where the admin user can select the video footage to check the distancing violations. The processing starts as soon as the upload button is clicked.



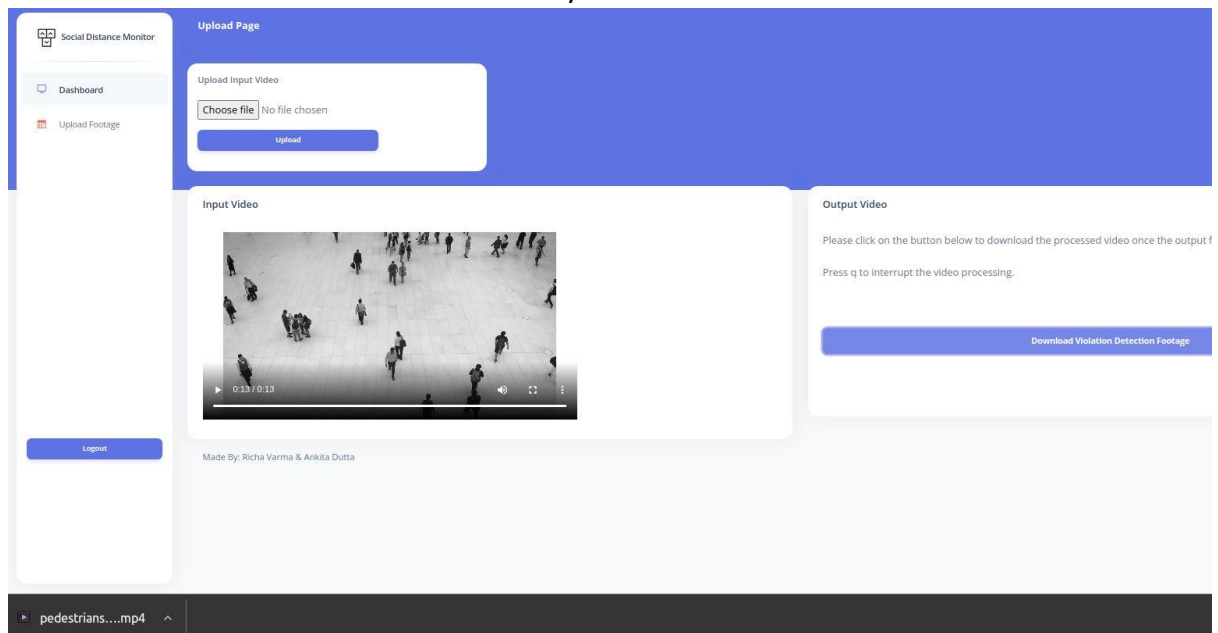
In the input video section, the uploaded video can be displayed.



As the upload button is clicked, the output processing has started, as seen in the frames below. We can see the changing number of violations as the video progresses.



Once the video is processed, we can click on the download button, and the processed video will be downloaded into the admin user's system.



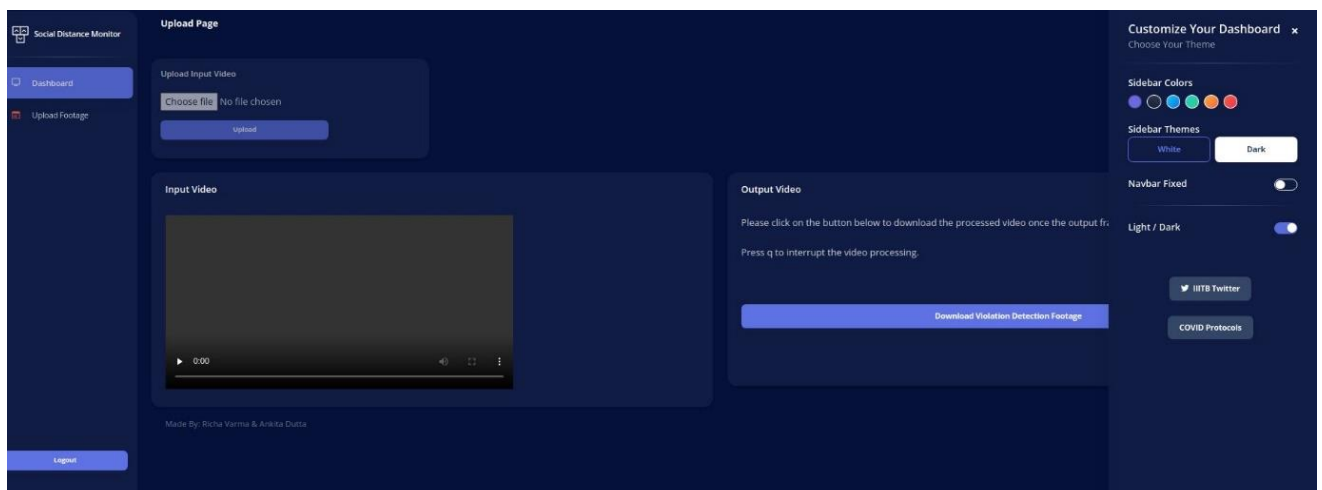
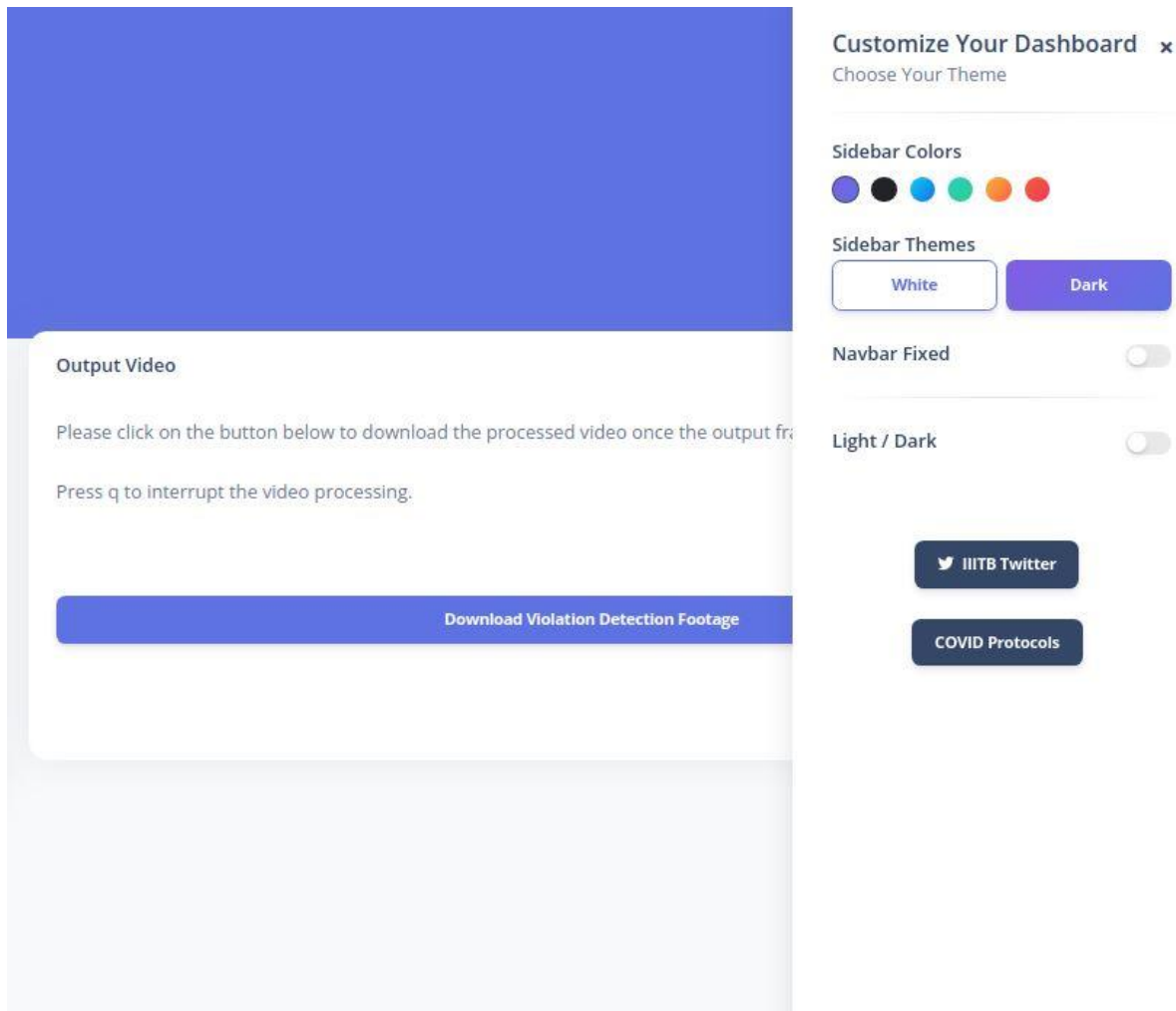
Below is the played video after download. The people in red bounding boxes are the ones violating the distancing rules.



8.1.4. Customize Dashboard

This is a feature that allows the admin user to customize the application's appearance by switching between light and dark themes.

Also, there are links available to IIIT Bangalore's official Twitter handle and Covid protocols by World Health Organisation (WHO).



9. Challenges

i) Docker Container Interaction with localhost hardware

The crucial part of our project is processing a video frame by frame and getting the violations for every frame. However, we know that the container is isolated from the local system. As a result, it cannot leverage the hardware capabilities of the localhost to access the GUI and process the video frames.

Hence we need to modify the client system so that the video can be processed when the application is launched from inside the container.

Set the environment variable DISPLAY as :0. This variable tells our application how to communicate with localhost GUI (X Server).

X Server is used for bitmap displays on Unix systems.

export DISPLAY=:0

Give the docker permission to access the hardware of the localhost using the command:

xhost +local:docker

ii) Kibana port already in use

Find the process ID that is occupying the port 5601, i.e., Kibana's port, by running the commands on the terminal:

```
sudo lsof -i :5601
```

```
sudo kill <pid>
```

```
(base) ankd@ankd:/tmp$ echo $DISPLAY
:0
(base) ankd@ankd:/tmp$ xhost +local:docker
non-network local connections being added to access control list
```

iii) Permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock

Run within project directory: `sudo chmod 666 /var/run/docker.sock`

```
(base) ankd@ankd:/tmp$ sudo chmod 666 /var/run/docker.sock
```

iv) Loading yolov3.weights into GitHub is not allowed due to the large size

Essentially we need to store the yolov3.weights file in the Yolo-coco folder in GitHub. However, the size of this file is about 250 MB and hence cannot be pushed to GitHub.

Hence, we instead put a run command in the Dockerfile to directly download the weight into the application Docker image.

RUN wget <https://pjreddie.com/media/files/yolov3.weights>

10. Scope And Future Work

10.1 Integration with cctv footage-

We can extend our project capabilities to accommodate cctv footage so that an organisation/institute can keep track of violations on a day-to-day basis and take necessary steps to enforce social distancing on their premises.

10.2 Mask Detection-

We can customise the yolo model for detecting masks as well. With addition of this feature we can further enhance the project's abilities to ensure that people are abiding by the covid protocols.

11. Conclusion

We managed to build a web application based on computer vision to check violations of social distancing norms. This entire application was automated using DevOps tools like Jenkins for continuous integration, Ansible for configuration management and deployment, Docker for containerization, Unittest for testing and ELK for monitoring.

12. References

https://codethechange.stanford.edu/guides/guide_flask_unit_testing.html

<https://www.elastic.co/blog/how-to-display-data-as-a-percentage-in-kibana-visualizations>

<https://phoenixnap.com/kb/kibana-tutorial>

<https://gursimar27.medium.com/run-gui-applications-in-a-docker-container-ca625bad4638>

<https://stackoverflow.com/questions/60139952/how-to-connect-2-running-containers-to-each-other-in-docker-compose>

<https://onexlab-io.medium.com/docker-compose-mysql-initdb-4c3388047dea>

<https://stackoverflow.com/questions/61364113/dockerfile-how-to-download-a-file-using-curl-and-copy-into-the-container>

<https://cimentadaj.github.io/blog/2020-11-25-deploying-mysql-database-using-docker/deploying-mysql-database-using-docker/>