***Development of an MLOps solution for ZOO Model***

I developed an enterprise-level machine learning system for object detection. To achieve this, I utilize Flask for the frontend application and TensorFlow serving hosted in Docker for inferencing on the GPU. The system will provide users with the ability to upload images for classification and view the results on a dedicated results page. This page will display the classified image with bounding boxes, a table of objects detected with their associated probability, and the original non-classified image for comparison.

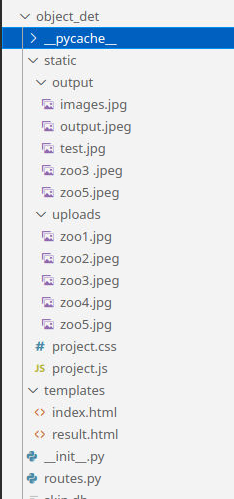
To enable image processing using TensorFlow serving, the backend python code is utilized REST protocols. This will allow us to render the required elements on each page, providing users with a seamless experience.

Our goal is to create a professional and user-friendly system that will meet the needs of our clients.

The following directories are included in this project:



* Utils: This directory contains object detection utilities that are used throughout the project.
* Pycache: This directory contains compiled Python bytecode files that are generated when the Python interpreter runs the code.
* Saved\_model: This directory contains the saved TensorFlow model for object detection. This model is used to detect objects in images.
* Protos: This directory contains protocol buffer files that are used by TensorFlow serving. These files define the structure of the data that is passed between different components of the TensorFlow system.
* Object\_detection: This directory contains scripts that are used for object detection. These scripts use the TensorFlow model to detect objects in images.
* Object\_detect: This files related to the web application, such as HTML templates, CSS stylesheets, and JavaScript code for displaying the user interface and handling user input.
* Data: This directory contains the label\_map.pbtxt file, which maps object labels to integer IDs. This file is used by the object detection scripts to identify the objects that are present in an image or video.
* Instance: This directory contains instance-specific data for the Flask application. This data includes configuration files, log files, and other data that is specific to a particular instance of the application.
* Docker\_commands: This directory contains scripts that are used for building and running Docker containers. Docker is a containerization technology that is used to package applications and their dependencies into a single container.
* Dockerfile: This file specifies the instructions for building a Docker image. The Docker image is used to create a container that can run the Flask application.
* Run.py: This file contains the Flask application code. The Flask application is a web application that provides a user interface for the object detection scripts.
* Requirements.txt: This file lists the Python packages that are required for the Flask application. These packages are installed automatically when the application is deployed.

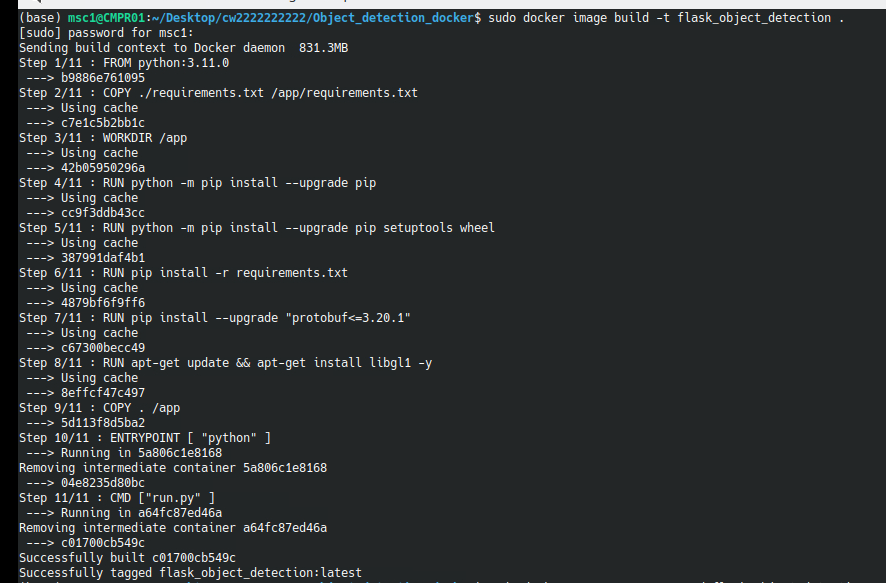
The web application files are kept in object\_det folder which includes:

* Static :
* Uploads : This includes the image files that I used to upload for detect the objects.
* Output : The result of object detect images are saved here.
* Project.css : The file contains CSS rules that define the look and layout of elements on the web pages, such as fonts, colors, margins, padding, and so on.
* project.js : The file contain sJavaScript code that adds interactivity and functionality to the web pages, such as handling user input.
* Templates:
* Index.html: This file serves as the main page of the Flask web application, where users can conveniently upload images for classification
* Output.html: This page displays the results of the classification process. This page showcases the classified image, complete with bounding boxes, as well as a table that contains the image name, detected objects, and their corresponding probabilities. Additionally, the page also presents the original non-classified image, allowing users to compare the results and gain a better understanding of the classification process. Overall, these pages work together seamlessly to provide users with a comprehensive and user-friendly experience.
* Routes.py : routes.py is a module in a Flask web application that defines the different routes or endpoints that the application can handle. It typically contains functions or classes that are decorated with route decorators such as @app.route('/endpoint'). The functions or classes define the behaviour of the application when a user accesses a particular URL endpoint, and can return HTML templates, JSON data, or other types of responses.
* \_\_init.py\_\_ : The code initializes a Flask app, sets configuration variables for image uploads and a secret key for security, specifies the database URI, imports routes from a Python script, and pushes the app context.

OUTPTS

Step 1: Run the below commands

* sudo docker image build -t flask\_object\_detection .



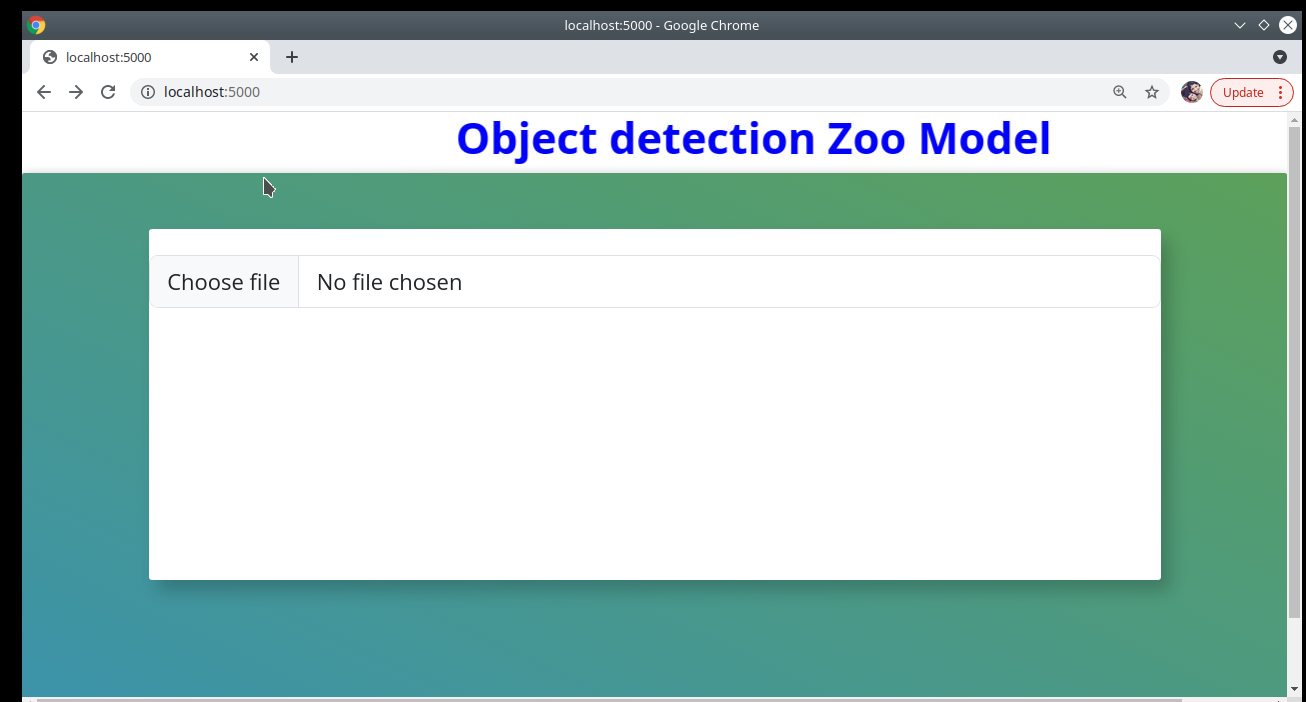
* sudo docker run -p 5000:5000 -d flask\_object\_detection



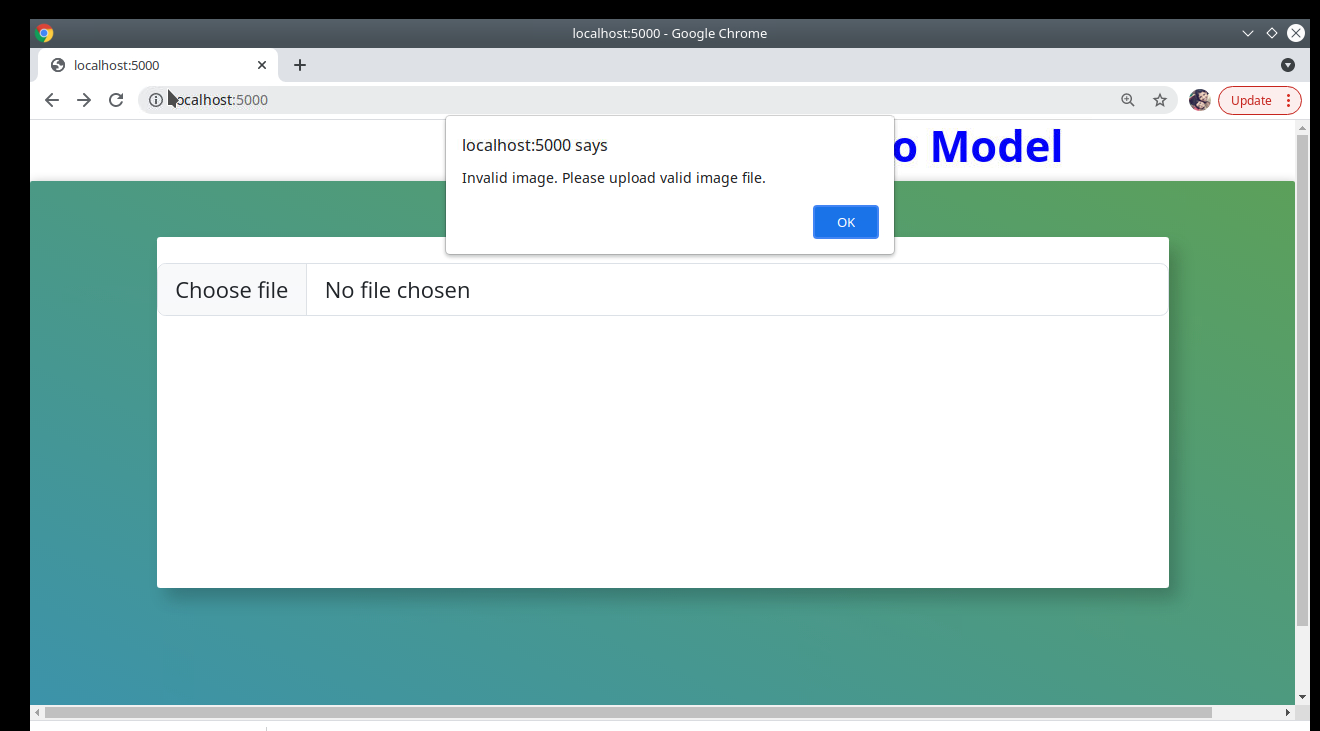
* sudo docker ps



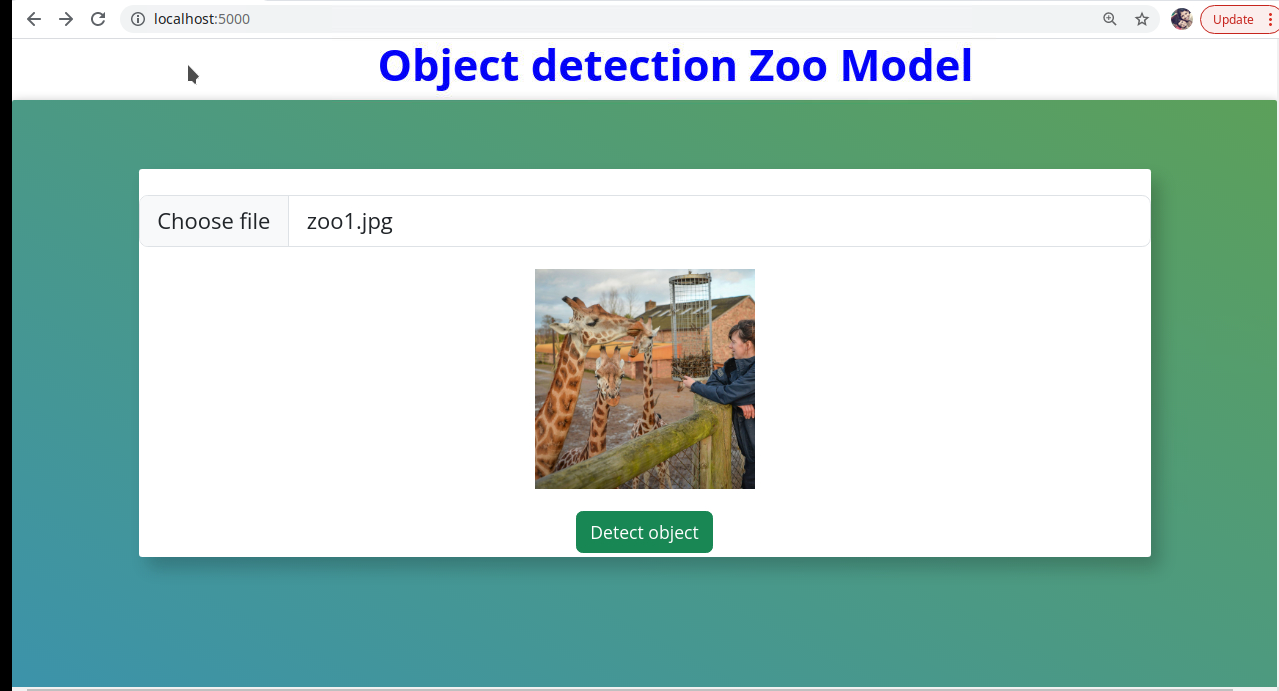
Step 2: Browse <http://localhost:5000/> in chrome. Then our web application will load.



If we click the choose file button, we can upload images for object detection. Here we can only upload jpg,jpeg and png images. If we try to upload any other format file, the site will shows an error like:

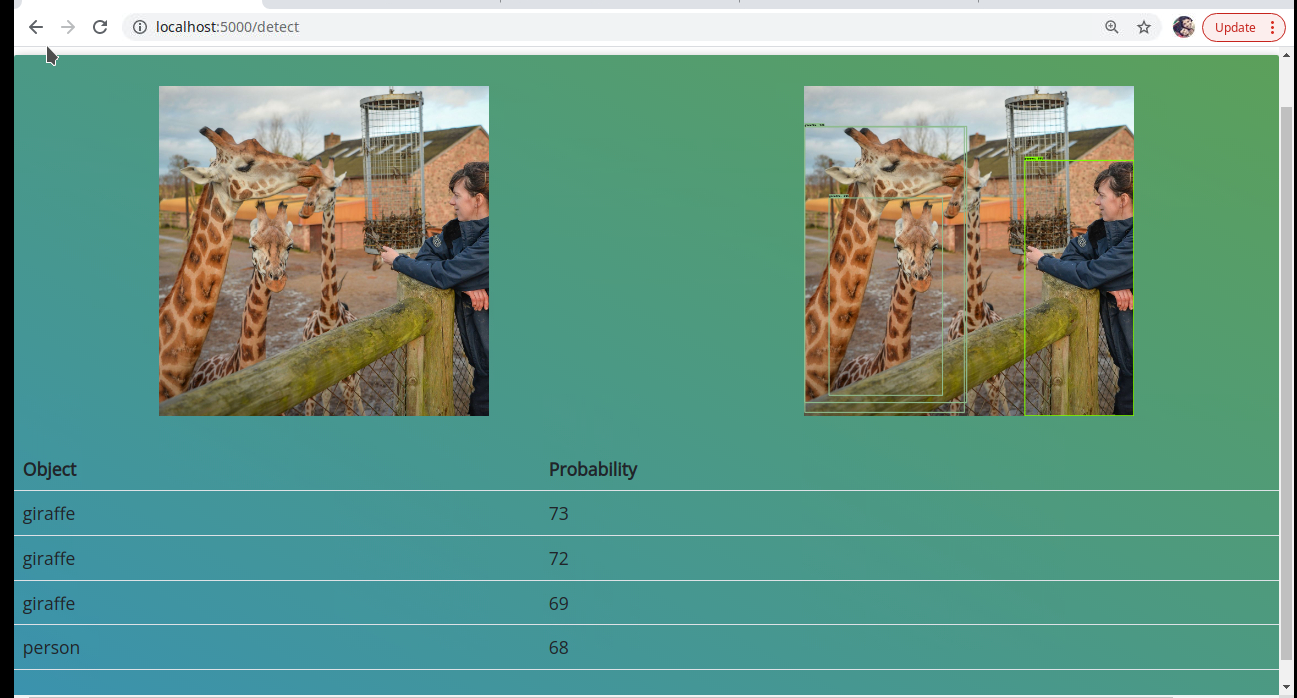


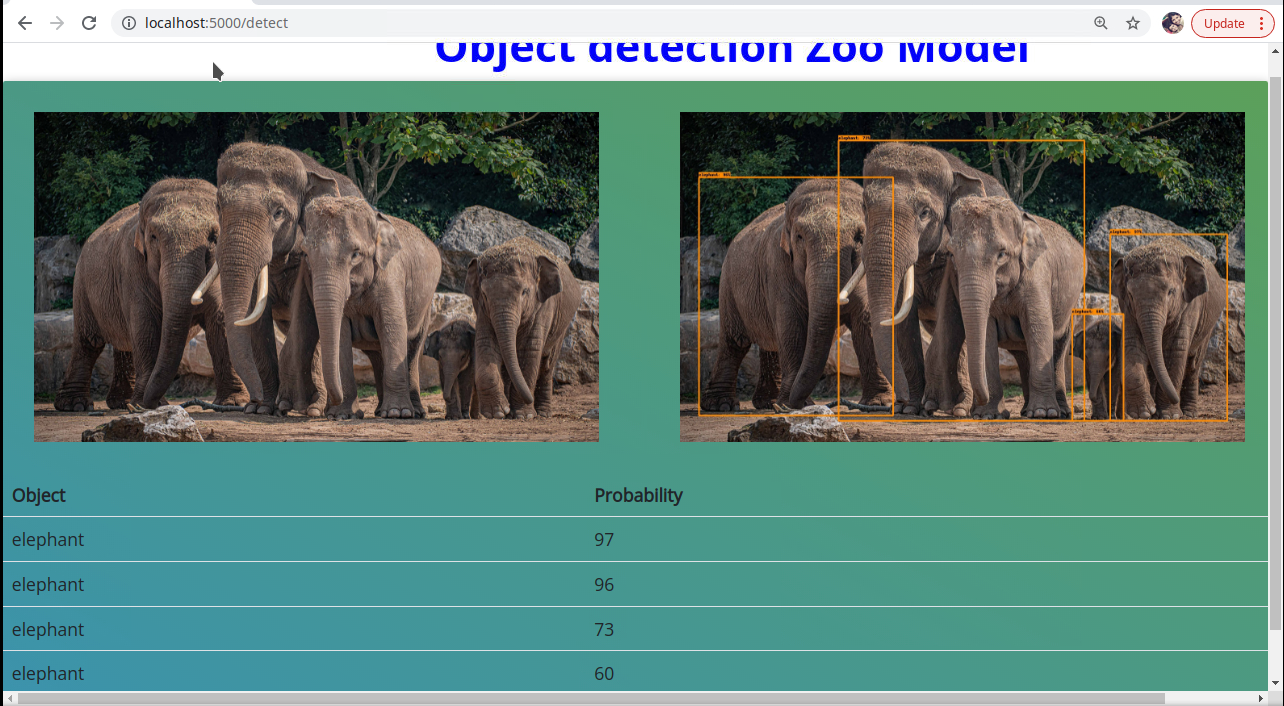


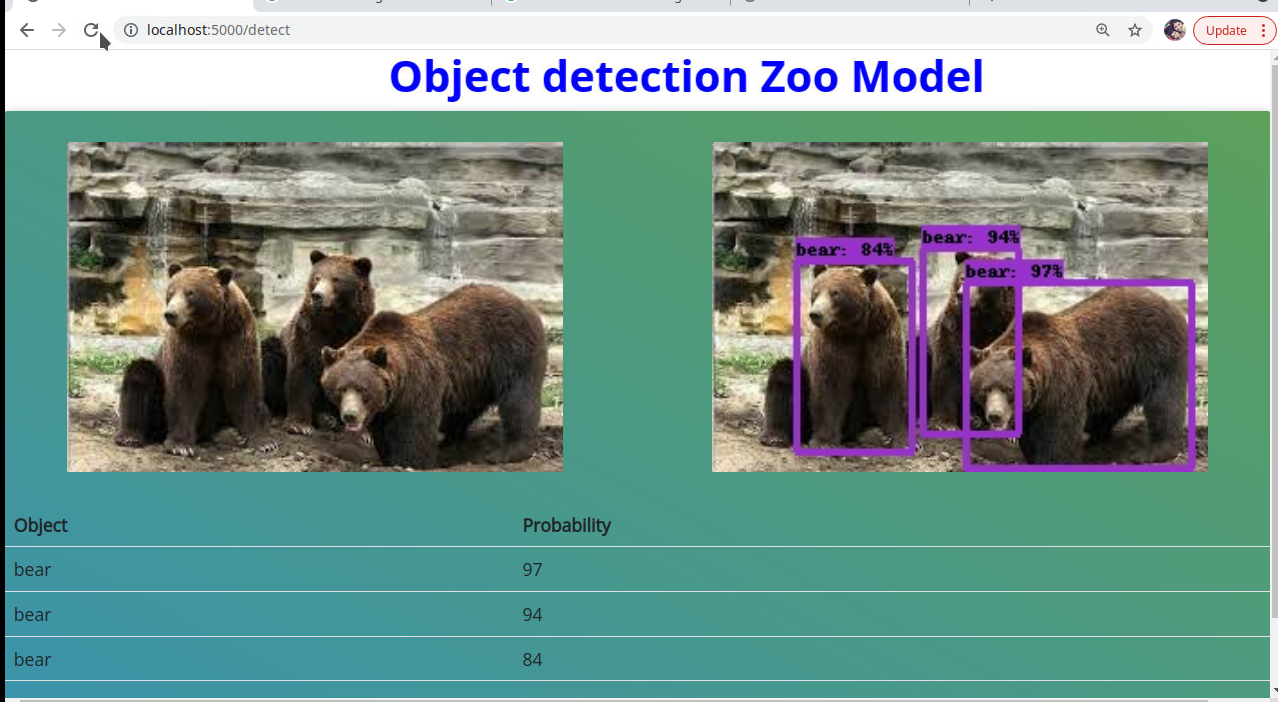


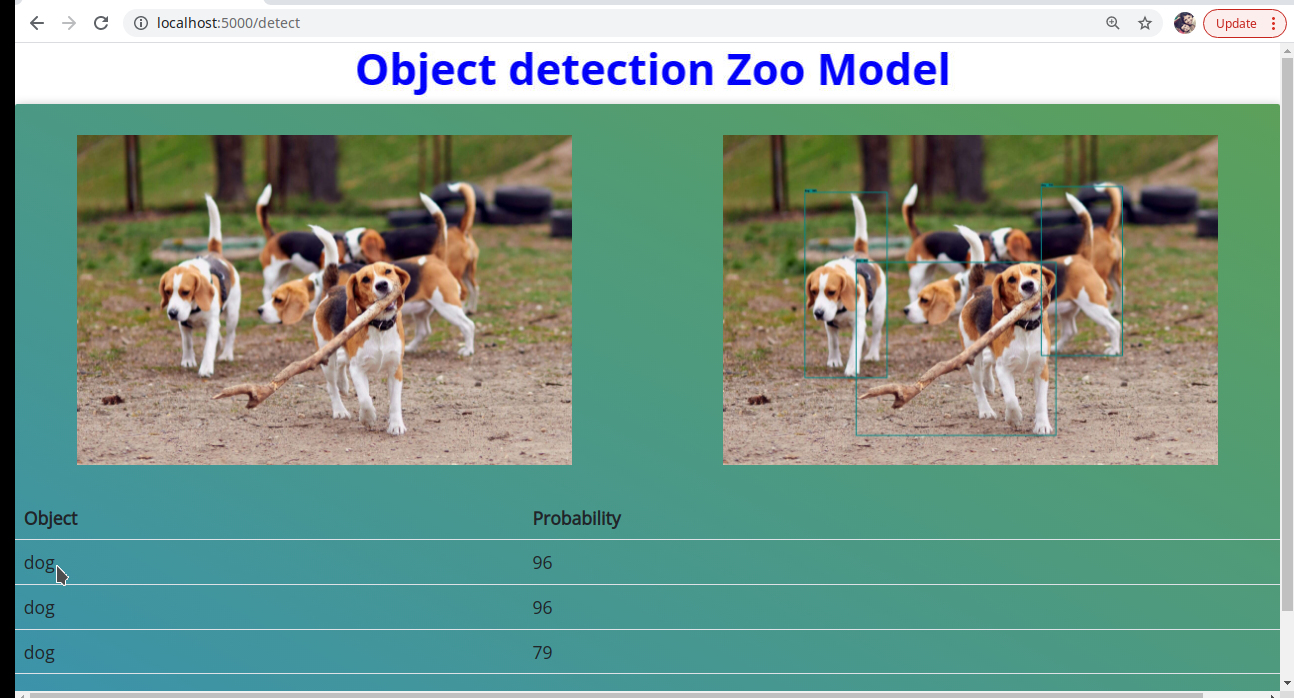


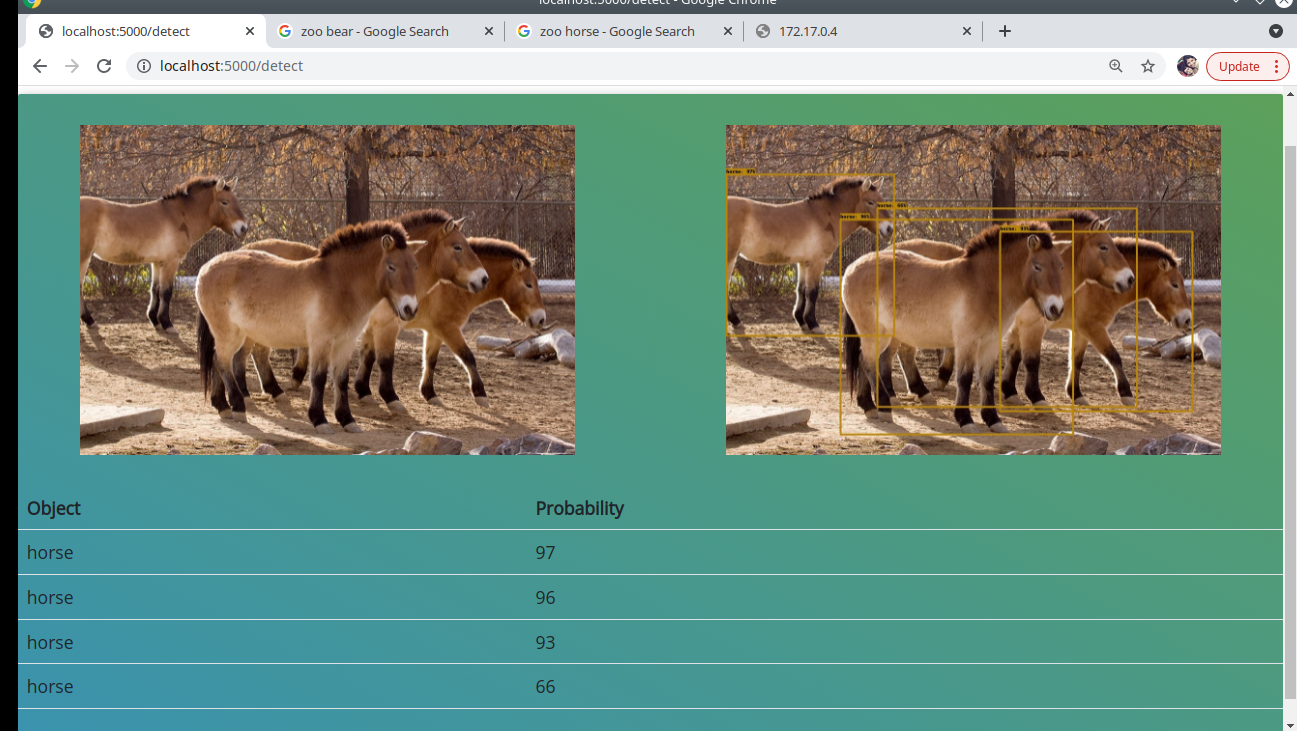
After successful uploading of the image, we can click on the Detect object button. Then we will get the below output which displays the classified image (with bounding boxes) along with a table containing the image name, objects detected and associated probability. Here I am adding the images of sample outputs and which is also saved in static/output folder.



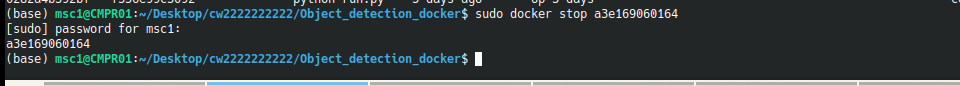








Step 3: Stop the container



Overall, this project is designed to provide a robust and scalable solution for object detection.