inzva Applied AI Program
Week 5

Machine Learning Model Deployment

Duygu Ay

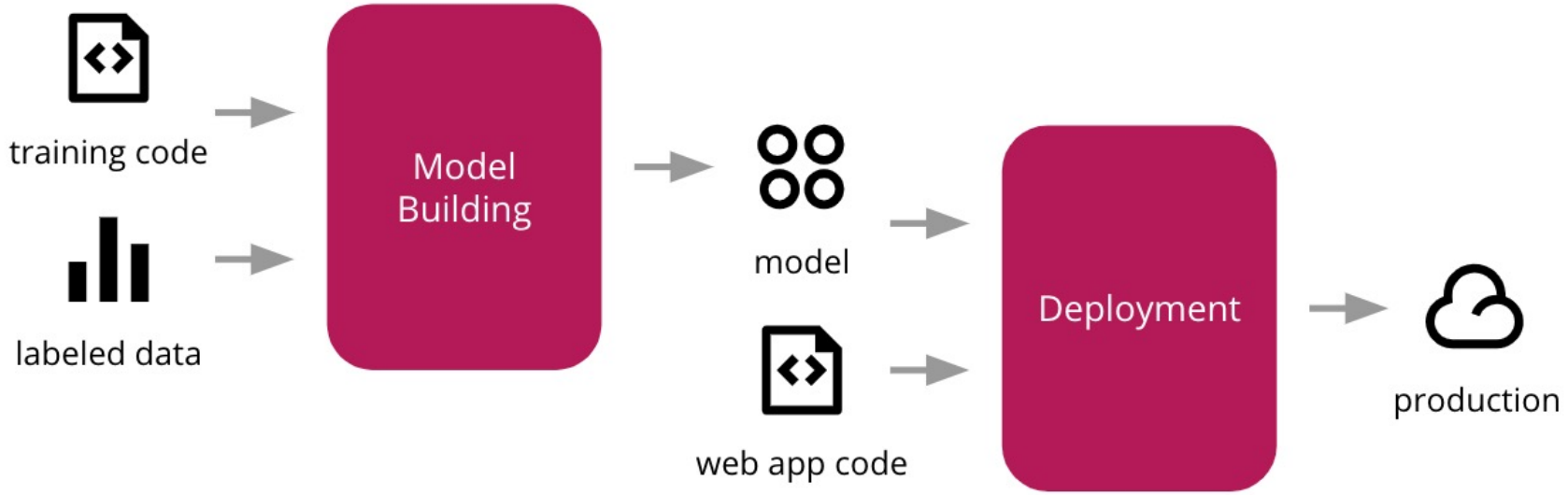
21.08.2021

Today’s Topics

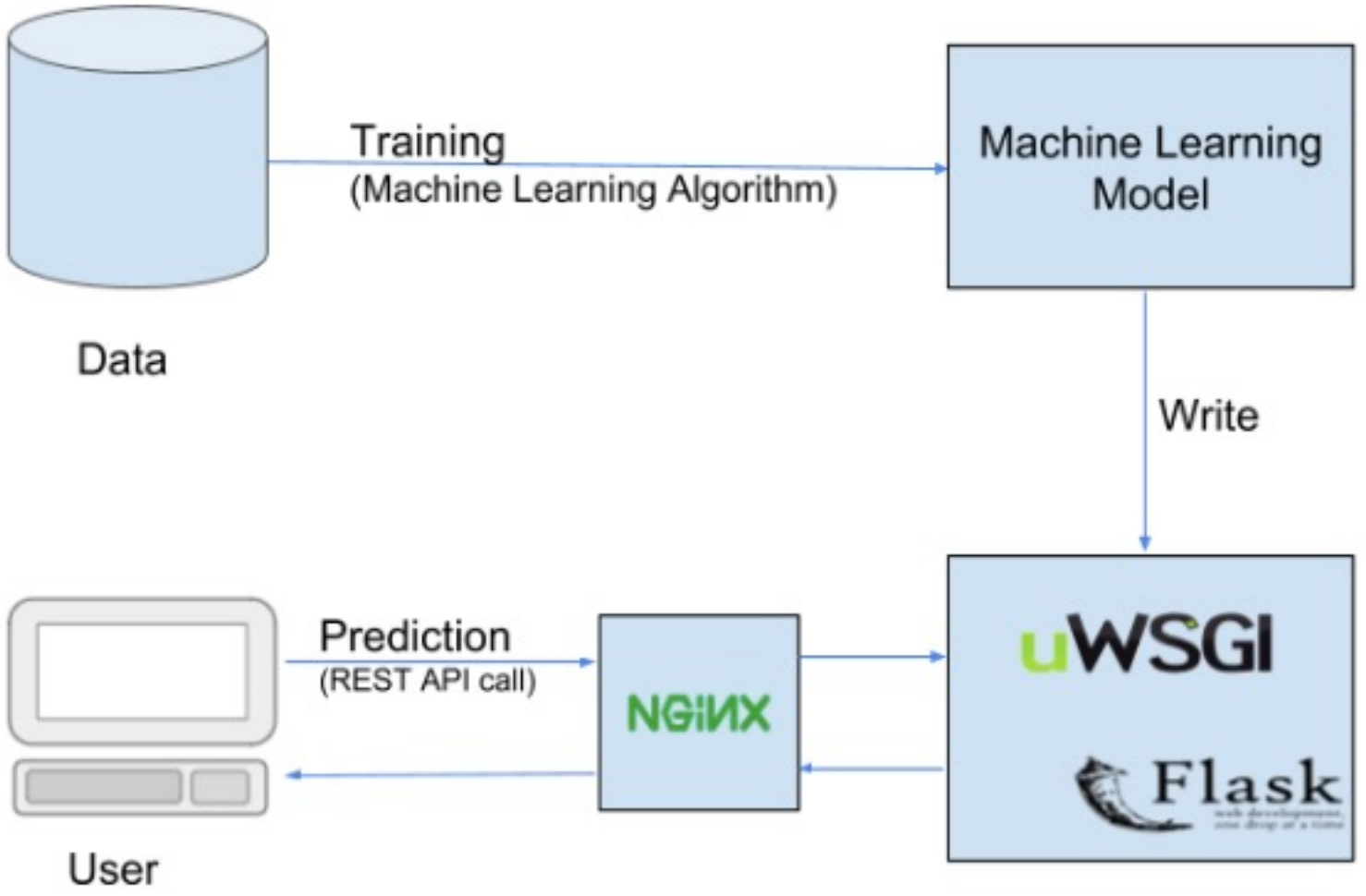
- Front end web development with HTML & CSS
- Deploying ML models with Flask (Back end)
- Google cloud server setup
- Serving Flask Applications with uWSGI and Nginx
- Git basics
- Model deployment with App Engine on GCP
- Model deployment with Streamlit

What is Deployment?

Deployment is **the method by which you integrate a machine learning model into an existing production environment to make practical business decisions based on data**.



Deploying with Flask and Serving Web App with uWSGI and Nginx



ML Model: Diabetes Prediction

- This program builds a classifier for Pima Indians Diabetes dataset - https://www.kaggle.com/uciml/pima-indians-diabetes-database.
- It is a binary (2-class) classification problem.
- There are 768 observations with 8 input variables and 1 output/target variable. The variable names are as follows:
  - Pregnancies: Number of times pregnant
  - Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
  - BloodPressure: Diastolic blood pressure (mm Hg)
  - SkinThickness: Triceps skin fold thickness (mm)
  - Insulin: 2-Hour serum insulin (mu U/ml)
  - BMI: Body mass index (weight in kg/(height in m)^2)
  - DiabetesPedigreeFunction: Diabetes pedigree function
  - Age: Age (years)
  - Outcome: Class variable (0 or 1) 268 of 768 are 1, the others are 0

Front end Web Development with HTML & CSS

What is HTML?

- Hyper Text Markup Language
- **NOT** a programming language
- Markup language for creating webpages and documents
- Building blocks of the web
- Does not need a web server
- Files must end with the **.html** extension
- Runs in a web browser
- **index.html** is the root/home page of a website
  - http://something.com : loads **index.html** file
  - http://something.com/about : loads **about.html** file

HTML Page Structure

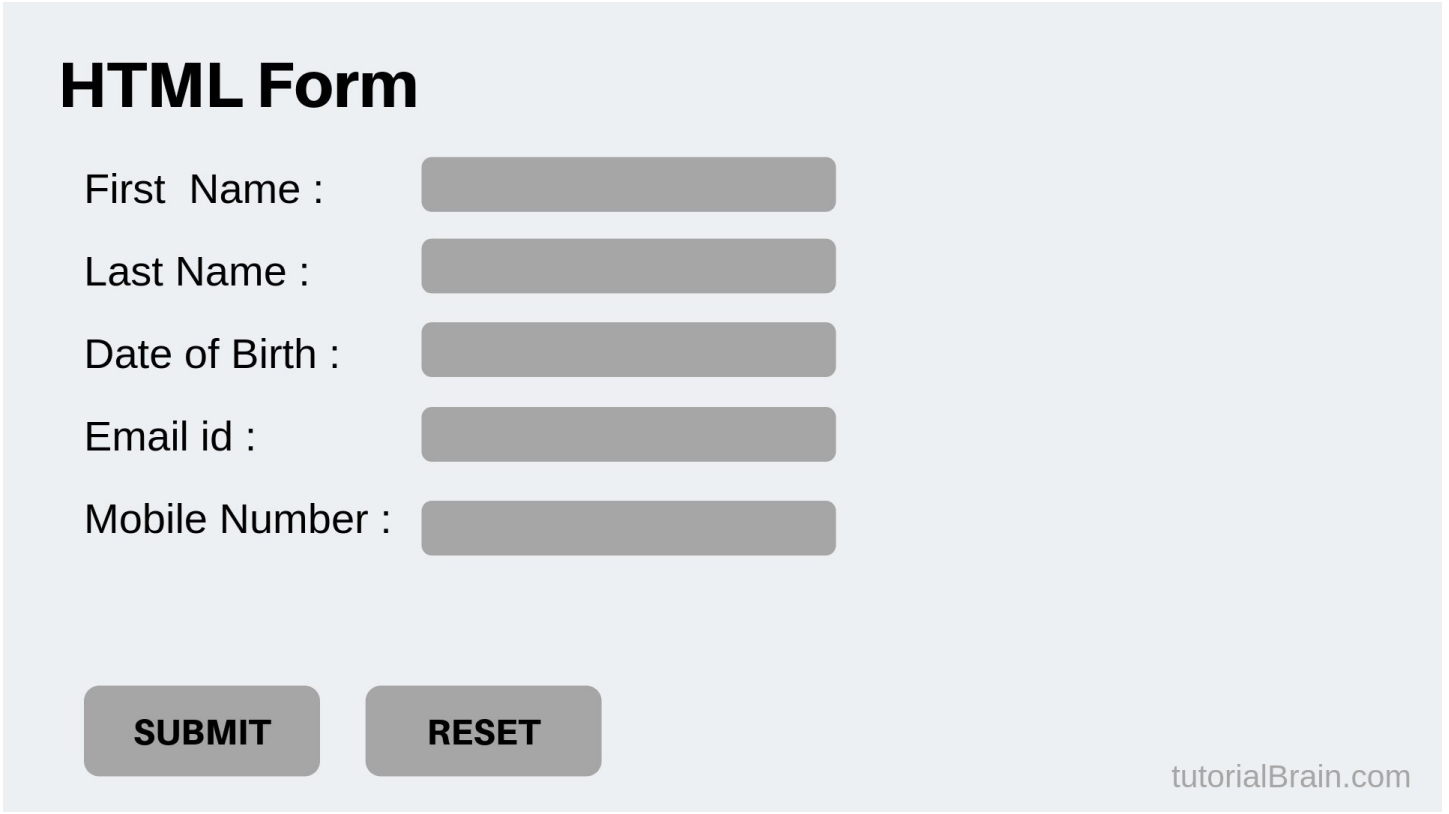
- Element names surrounded by angle brackets.
- Normally come in pairs.
- End tag is same but with a forward slash.

Tags in HTML

- The <nav> tag defines a set of navigation links.
- The tag defines a division or a section in an HTML document.
- The <a> tag defines a hyperlink, which is used to link from one page to another.
- The
 tag inserts a single line break.
- <meta> tags always go inside the <head> element, and are typically used to specify character set, page description, keywords, author of the document, and viewport settings.
- The <input> tag specifies an input field where the user can enter data.

For other tags visit https://www.w3schools.com/tags/.

Creating Forms



What is CSS?

- Cascading Stylesheets
- **NOT** a programming language
- Styling language
- Used for website layout and design

Methods for Adding CSS

Inline CSS: Directly in the html element. (not preferrable!)

Internal CSS: Using <style> tags within a single document.

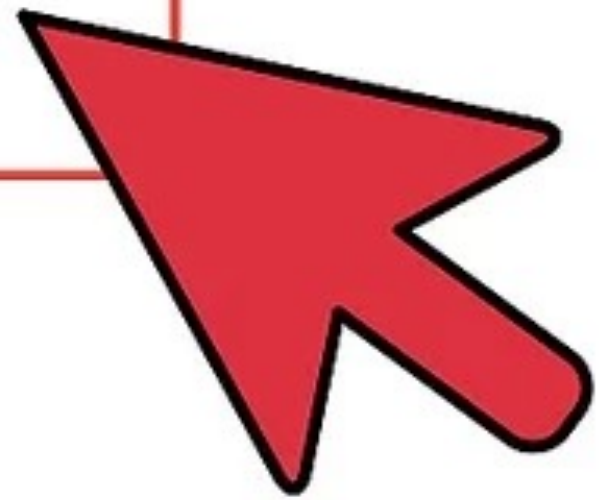
External CSS: Linking an external .css file.

CSS Selectors

Selector	Role
p{ }	Tag selector, **all p tags**
#para{ }	Id para (***unique***)
.para1{ }	Class para1 (***multiple***)
p.para{ }	P tag with class para
P .para{}	P with child having class para
div p{}	Div with child p tag
*{}	All tags{ **Universal Selector**}
h1, h3, h5{}	Only h1, h3 and h5 (***grouping***)
.para a{ }	A with parent para class
body{}	Parent of all tags

Creating Classes in CSS

```
<!DOCTYPE html>
<html>
<head>
<style>
.cities {
    background-color:black;
    color:white;
    margin:20px;
    padding:20px;
}
</style>
</head>
<body>

<div class="cities">
<h2>London</h2>
```



Deploying ML models with Flask (Back end)

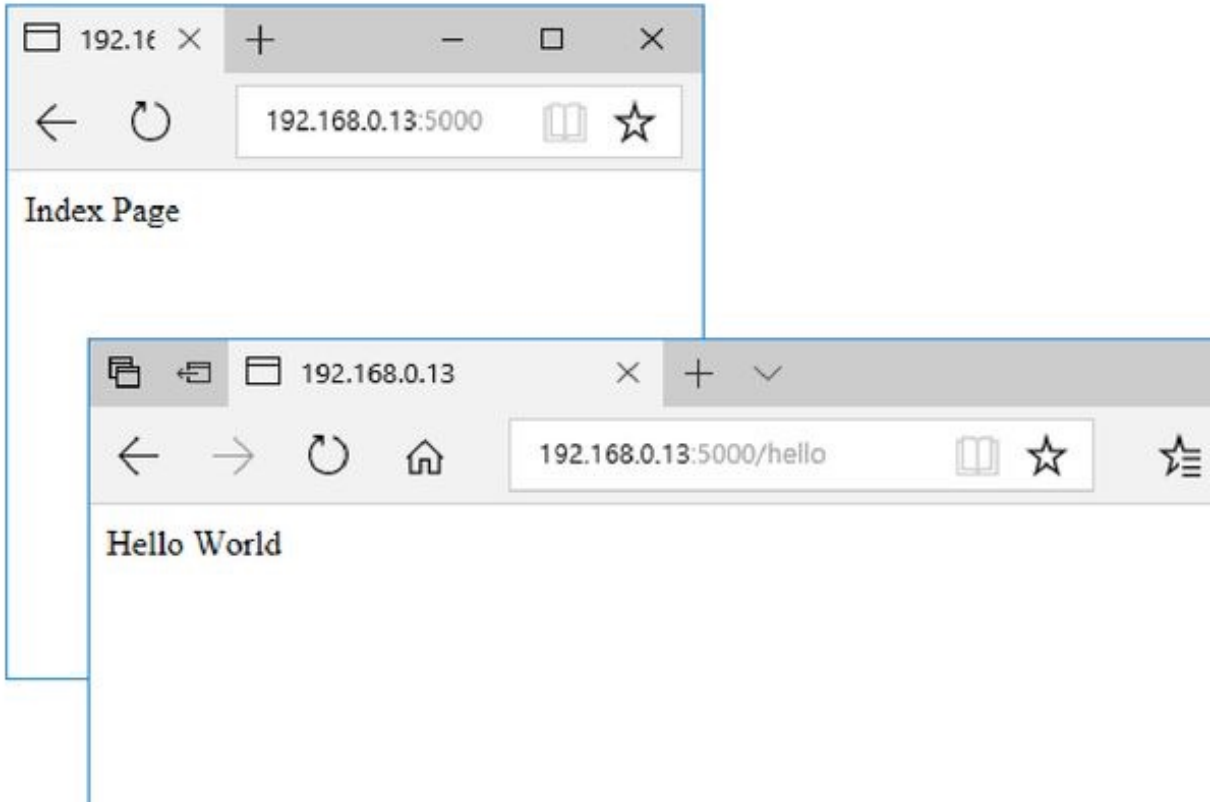
What is Flask?

- Flask is a web framework, it’s a Python module that lets you develop web applications easily.
- It is a lightweight micro framework.
- It enables web application developers to write applications without worrying about low-level details such as protocol, thread management, and so on.

Routing

- route() decorator is used to bind a function to a URL

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return 'Index Page'


@app.route('/hello')
def hello():
    return 'Hello World'


if __name__ == '__main__':
    app.run(host='0.0.0.0')
```



Debug Mode

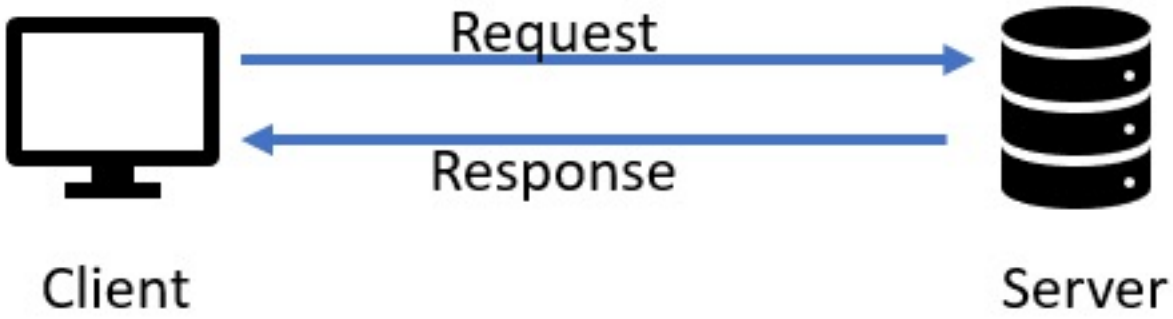
- The debugger **allows executing arbitrary Python code from the browser**.
- If we write app.run() and our application is under development, it should be restarted manually for each change in the code. So for every change in the program we have to restart the server and try to observe the changes.
- To overcome this problem, enable debug support. The server will then reload it self if the code changes.

Link to HTML File

- render_template() is used to generate output from a template file that is found in the application's templates folder.

Request

- To access the incoming data in Flask, you have to use the request object.
- The request object **holds all incoming data from the request**.



HTTP Methods

- The method is the type of action you want the request to perform and is sent from the client to the server on every request.
- GET - Used to fetch the specified resource. It is used to return data at a specified resource/location.
  - Returning text
  - Rendering templates

- POST - Used to create new data at the specified resource.
  - POST requests should be used to create NEW resources (New users, devices, posts, articles, datasets etc..)

Other files for app

- Create __init__.py file (for setting python package)
  - **from flask import Flask**
  - **myapp=Flask(__name__,template_folder='templates',static_folder='static:css')**
  - **from myapp import main**
- Create run.py file outsite of app folder
  - **from myapp import myapp**
  - **if __name__ == "__main__": myapp.run(debug=True)**
- Creating requirement.txt
  - **pip install pipreqs**
  - **pipreqs ./**
- Import myapp in flask framework file
  - **from myapp import myapp**
- Try
  - **python run.py**

Google Cloud (GCP) Server Setup

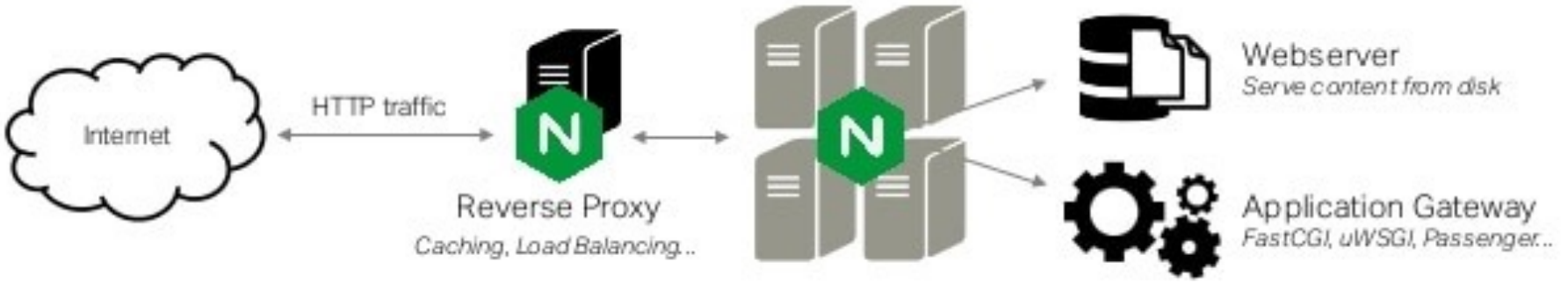
- Create project
- Create VM instance from Compute Engine
- Create SSH key on local computer
  - **ssh-keygen -t rsa -f ~/.ssh/”folderName” -C “username”**
  - **cd ~/.ssh and cat “foldername”.pub**
    - **Copy and past the key to VM instance by editing.**
- Connect server from your local computer
  - **ssh –i ~/.ssh/”folderName” “username”@[IP address of VM instance]**

Setting Requirements

- Installing the Components from the Ubuntu Repositories
  - **sudo apt update**
  - **sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev python3-setuptools**
- Move your app files to server
- Install requirement.txt
  - **cd myapp**
  - **pip install -r requirements.txt**
- Install uwsgi and nginx
  - **sudo apt install uwsgi-core**
  - **sudo apt-get install nginx**
- Check nginx from server IP address
  - **http://[IP]**

Serve Flask Applications with uWSGI and Nginx on VM Machine

What is NGINX?

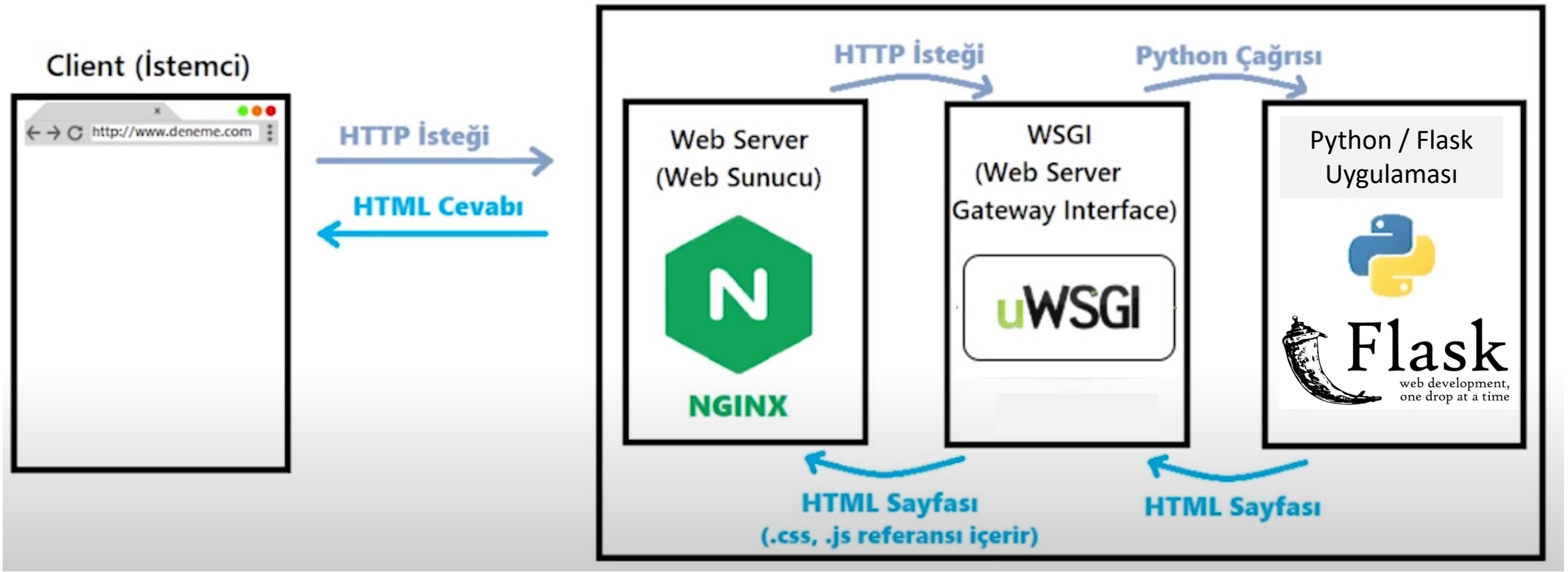


- Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache.
- It can handle high http traffic.
- It is good at security and cashing.

What is uWSGI?

- uWSGI is a Web Server Gateway Interface (WSGI) server implementation that is typically used to run Python web applications.
- The uWSGI server is responsible for loading your Flask application using the WSGI interface.

.


Client (İstemci)
http://www.deneme.com
HTTP İsteği
HTML Cevabı
HTTP İsteği
Python Çağrısı
Web Server
(Web Sunucu)
NGINX
WSGI
(Web Server
Gateway Interface)
uWSGI
Python / Flask
Uygulaması
Flask
web development,
one drop at a time
HTML Sayfası
(.css, .js referansı içerir)
HTML Sayfası

Creating a uWSGI Configuration File

- Create outside of app folder named myapp.ini
  - [uwsgi]
  - wsgi-file = run.py
  - callable = myapp
  - socket = 127.0.0.1:4242
  - processes = 4
  - master = true
  - chmod-socket = 660
  - vacuum = true
  - die-on-term = true

Configuring Nginx to Proxy Requests

- cd /etc/nginx/
- cd sites-enabled
- sudo nano default
  - location / {
    include uwsgi_params;
    uwsgi_pass 127.0.0.1:4242; }
- Test
  - **sudo nginx -t**
- sudo systemctl restart nginx

Run app

- Add log file
  - touch file.log
- Run app on background
  - nano myapp.ini
  - daemonize = /home/duyguay/file.log
- uwsgi myapp.ini
  - Check server! Congrats ☺

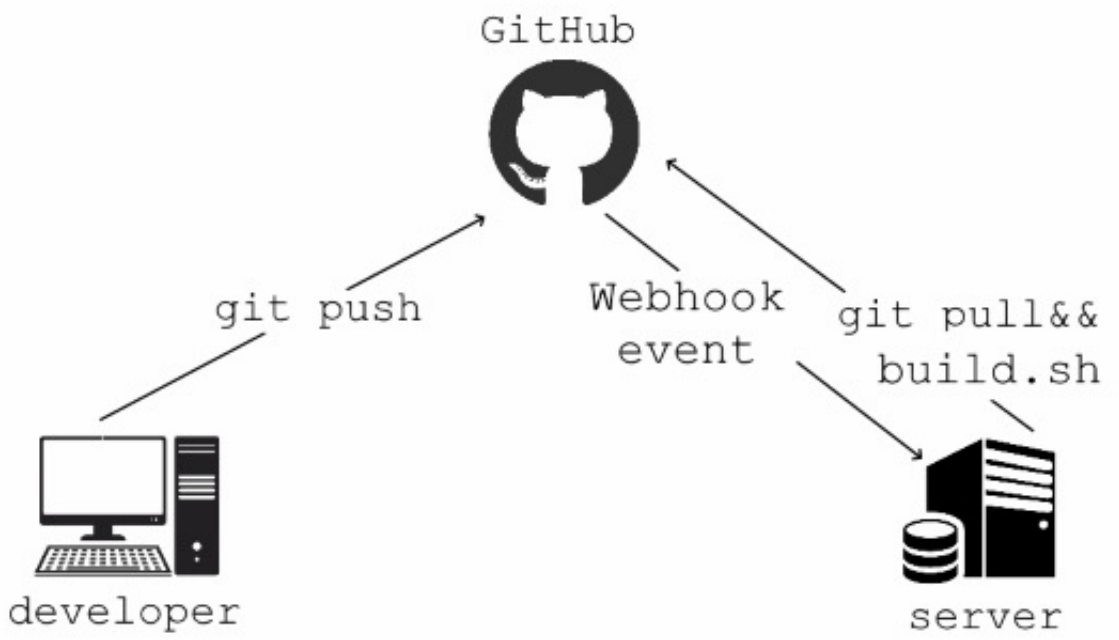
Push the Project to GitHub

- Create myapp.ini file also in your local
- Creating a repo in Github
- Check for Git Version, If not installed download the git according to OS of your system.
  - **git --version**
- Clone the repo you created with ssh link
  - **git clone “repo link”**
- Copy your files to cloned empty repo or move
- Committing files into the git repo.
  - **cd ‘your repo name’**
  - **git add .**
  - **git commit -m "First Commit”**
- Final push
  - **git push origin ‘branch_name’**

https://hackernoon.com/step-by-step-guide-to-push-your-first-project-on-github-fec1dce574f

Continuous Integration of your Code and App with Git

- Everytime you push your code to github from your local, you will see the change in your server directly.



Webhooks

- **Webhooks:** A webhook is a way for an app to provide other applications with real-time information.
- A webhook delivers data to other applications as it happens, meaning you get data immediately.
- Whenever you trigger an event in your github repo, it will send the information to the http address.
- Repo/settings/webhooks/Add webhook
- Write **payload url**:
  - http://[IP Adress]/update
  - Change content type to json

Adding Webhook Route to Flask

- **import time**
- **import uwsgi**
- **import git**

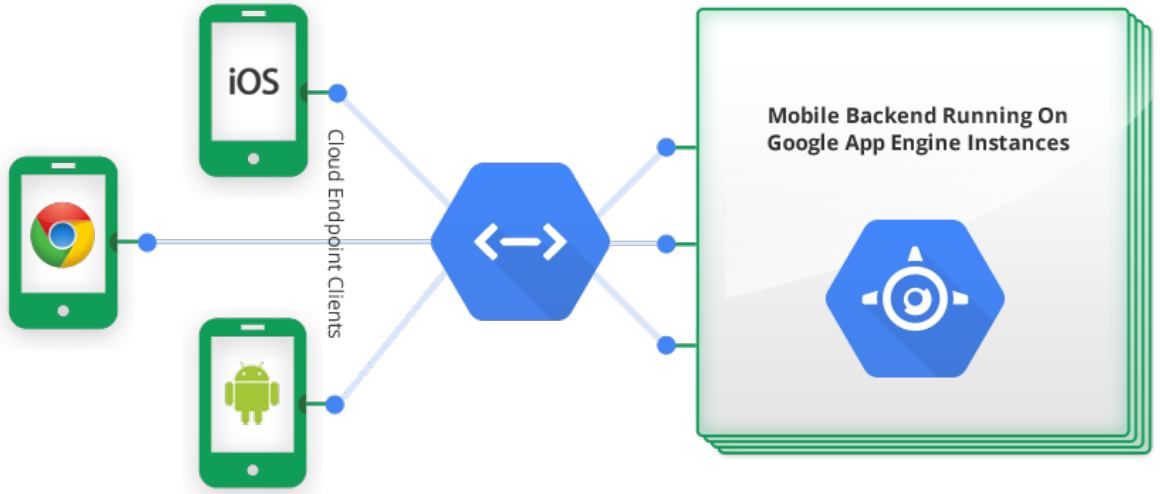
```
@myapp.route("/update", methods = ["POST"])
def update():
    if request.method == 'POST':
        g = git.cmd.Git('/home/duyguay/deployment')
        g.pull()
        time.sleep(5)
        uwsgi.reload()
        return ''
```

- **Change model path in main.py**
- **Commit and push**

- **clone git repo to server with http link**
- **pip install GitPython**
- **cd repo - uwsgi myapp.ini**
- **Change background color on your local and push**
- **Check server IP**

Model Deployment with App Engine on GCP

- App Engine is **a fully managed, serverless platform for developing and hosting web applications at scale**.
- Popular languages and framework
- Focus on your code
- Multiple storage options
- Powerful built-in services
- Familiar development tools
- Deploy at google scale



File Structure

- Make sure your **flask** file named as **main.py**
- Make sure to use **app.** in main.py
- Create a file called **app.yaml** to configure runtime
  - **runtime: python38**

Installing Google Cloud SDK

- Orchestrate virtual machine instances directly from your command line
- https://cloud.google.com/sdk/docs/install
- **./google-cloud-sdk/install.sh**

Initialize Gcloud

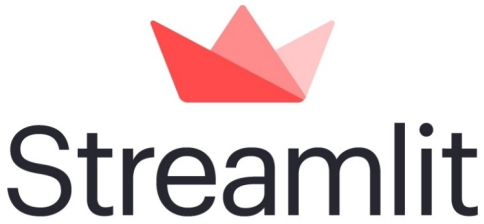
- New terminal
- cd ’your project folder path’
- gcloud init
- Enable Cloud Build API

Deploying App

- **gcloud app deploy app.yaml --project “project name”**
- Don’t forget to disable app engine!
  - App engine > settings > disable application

Streamlit

- Streamlit is **an open-source python framework for building web apps for Machine Learning and Data Science**.
- The best thing about Streamlit is it doesn’t require any knowledge of web development. If you know Python, you’re good to go!
- A few demos in streamlit
  - pip install streamlit
    - streamlit hello



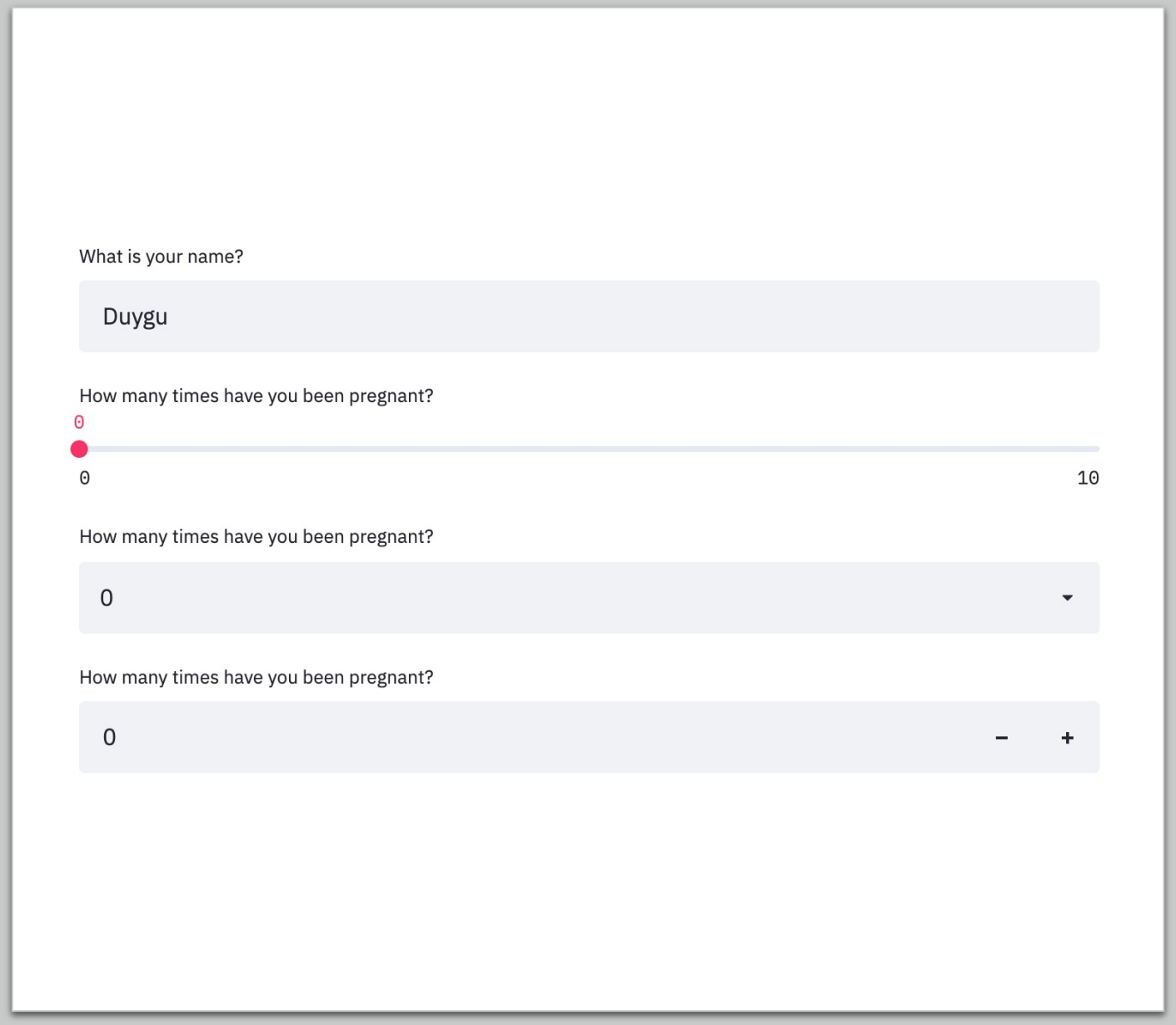
Creating Containers

```
import streamlit as st

siteHeader = st.beta_container()

with siteHeader:
    st.title('Diabetes Prediction')
    st.text('Diabetes is a disease that occurs when your blood glucose, also called blood sugar, is too high. ')
```

streamlit run main.py

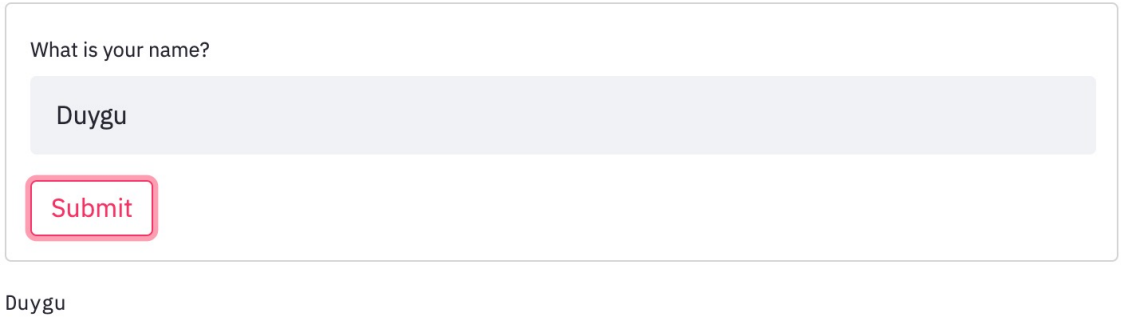
Collecting User Input

- Text input
- Slider
- Drop down menu
- Number input



Creating Forms

```
with st.form(key='columns_in_form'):
  input_feature = st.text_input('What is your name?')
  submitted = st.form_submit_button('Submit')

if submitted:
  st.text(input_feature)
```



Optimizing App's Run Time

- Every time the user changes an input, the whole application runs from the beginning.
- If your application includes a lot of calculations, works with big amounts of data or does calls to a database to retrieve data, your application will quickly become too slow to interact with.
- As a solution, the streamlit team developed "caching".

```
@st.cache  # 👈 This function will be cached
def my_slow_function(arg1, arg2):
    # Do something really slow in here!
    return the_output
```

Styling

```
st.markdown(
    """
    <style>
    .main {
    background-color: #FA6B6D;

    }

    </style>
    """,
    unsafe_allow_html=True
)
```

Deploying a Streamlit App

- Getting an account from streamlit.io/sharing.
- Sign up.
- Get invite in the next 1-2 business days.
- Create a github repo.
  - Make sure it is a public repo and it has a requirement.txt file.
- Sign in to streamlit.io
  - Make sure you sign up with e-mail in your github account.
- New app > Deploy

Thank you!