Project FastAPI

Create A New GitHub Repository

> GitHub Repository

Setup Venv

> *python -m venv venv*
> *venv\Scripts\activate.bat*

Setup Project FastAPI

> - pip install fastapi
>
> *->from fastapi import FastAPI*

> - create app folder
> - create `__init__.py` in app folder
> - move main.py to app folder
> - Terminal: uvicorn app.main:app --reload

Using .env to Store the Scecrte Variables

```
pip install python-dotenv
import os
from dotenv import load_dotenv

load_dotenv()
os.getenv('ENV_VAR')
```

Connecting to Postgres

> - pip install psycopg2

> -> import psycopg2
>
> - Connect to an existing database
>
> -> conn = psycopg2.connect("dbname=test user=postgres")
>
> - Open a cursor to perform database operations
>
> -> cur = conn.cursor()

- Execute a command: this creates a new table

-> cur.execute("CREATE TABLE test (id serial PRIMARY KEY, num integer, data varchar);")

- Pass data to fill a query placeholders and let Psycopg perform

-> cur.execute("INSERT INTO test (num, data) VALUES (%s, %s)", (100, "abc'def"))

- Query the database and obtain data as Python objects

-> cur.execute("SELECT * FROM test;")
-> cur.fetchone()
-> (1, 100, "abc'def")

- Make the changes to the database persistent

-> conn.commit()

- Close communication with the database

-> cur.close()
-> conn.close()

model: SQLALCHEMY

- pip install sqlalchemy
- setup database.py in app folder

-> from sqlalchemy import create_engine # connect to database
-> from sqlalchemy.ext.declarative import declarative_base
-> from sqlalchemy.orm import sessionmaker

```
DATABASE_HOSTNAME=localhost
DATABASE_PORT=5433
DATABASE_NAME=fastapi
DATABASE_USERNAME=postgres

engine = create_engine(SQLALCHEMY_DATABASE_URL)
Base = declarative_base()
SessionLocal = sessionmaker(autocommit=False,
autoflush=False, bind=engine)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

create table

- setup models.py in the app folder

-> from sqlalchemy import Column -> from .database import Base

- setup class Post(Base) in models.py
- To main.py

-> from . import models -> from .database import engine

-models.Base.metadata.create_all(bind=engine)

schema: Pydantic

- pip install pydantic

-> from pydantic import BaseModel
-> from typing import Optional

- setup routers folder in app folder
- setup router in routers folder

Hashing

- User Model, Schema

-> from pydantic import EmailStr

Hah password

- `pip install passlib[bcrypt]`

-> setup utils.py in app folder
-> from passlib.context import CryptContext
-> def hash(password: str):
-> def verify(plain_password, hashed_password):

Creating Token

- `pip install python-jose[cryptography]`
- create oauth2.py in app folder

-> from jose import jwt, JWTError
-> from datetime import datetime, timedelta
-> from . import schemas
-> from fastapi import Depends, status, HTTPException
-> from fastapi.security import OAuth2PasswordBearer
-> oauth2_scheme = Oauth2PasswordBearer(tokenUrl='login')

Joins in SqlAlchemy

> - in post.py
>
> -> from sqlalchemy import func

Database migration tool - alembic

> - pip install alembic
> - alembic init alembic
> - creat alembic folder & alembic.ini

CORS Policy

> - in main.py
>
> -> from fastapi.middleware.cors import CORSMiddleware
>
> ```
> origins = ["*"]
> app.add_middleware(
>     CORSMiddleware,
>     allow_origins=origins,
>     allow_credentials=True,
>     allow_methods=["*"],
>     allow_headers=["*"],
> )
> ```

Project Pytest

> - `Keep Test Case Independent`
> - `Use dev-database to interact with database`
> - `Design Good Test Case`

```
================================================
Testing - Pytest
================================================
pip install pytest
setup tests folder
create __init__.py in tests folder
create test_XXX.py for pytest
setup def test_XXX func for pytest
CMD:pytest
CMD:pytest -v
CMD:pytest -v -s
================================================
import pytest in test_XXX.py
```

```
@pytest.fixture()

@pytest.mark.parametrize(
    "x, y, z",
    [
        (x, y, z),
        (x, y, z),
        (x, y, z),
    ]
)
def XXX(x, y, z):
    assert ....

================================================
Testing - TestClient from FastAPI
================================================
# create test_users.py in tests folder

from fastapi.testclient import TestClient
from app.main import app
from app import schemas

client = TestClient(app)

for cleaning, pytest --disable-warnings

for fail stop, pytest -v -x

================================================
Setup testing database
================================================
# in test_users.py
import pytest
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.config import settings
from app.database import get_db, Base
from app import models, schemas

SQLALCHEMY_DATABASE_URL =
f'postgresql://{settings.database_username}:
{settings.database_password}@
{settings.database_hostname}:
{settings.database_port}/{settings.database_name}_test'

engine = create_engine(SQLALCHEMY_DATABASE_URL) # create table in
fastapi_test database

Base.metadata.create_all(bind=engine)

TestingSessionLocal = sessionmaker(autocommit=False,
autoflush=False, bind=engine)

def override_get_db():
    db = TestingSessionLocal()
    try:
        yield db
```

```
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db

================================================
Create & destroy database after each test
================================================
@pytest.fixture
def client():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    yield TestClient(app)

def test_root(client):
    res = client.get('/')
    assert res.json().get('message') == 'Welcome to FastAPI again'

def test_create_user(client):
    res = client.post('/users/', json={'email':
"hey123@gmail.com", 'password': "password"})
    new_user = schemas.UserOut(**res.json())
    #print(new_user)
    assert new_user.email == "hey123@gmail.com"
    assert res.status_code == 201

================================================
More Fixtures to handle database interaction
================================================

SQLALCHEMY_DATABASE_URL =
f'postgresql://{settings.database_username}:
{settings.database_password}@{settin
database_hostname}:{settings.database_port}/
{settings.database_name}_test'

engine = create_engine(SQLALCHEMY_DATABASE_URL)

TestingSessionLocal = sessionmaker(autocommit=False,
autoflush=False, bind=engine)


@pytest.fixture
def session():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()


@pytest.fixture
def client(session):
    def override_get_db():
        try:
```

```
            yield session
        finally:
            session.close()

    app.dependency_overrides[get_db] = override_get_db
    yield TestClient(app)
```

```
================================================
create database.py in tests folder
================================================
def test_user_login(client):
    res = client.post('/login', data={'username':
"hello123@gmail.com", 'password':
"password"})
    print(res.json())
    assert res.status_code == 200
================================================
Fixture scope

@pytest.fixture(scope="module")
def session():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()


@pytest.fixture(scope="module")
def client(session):
    def override_get_db():
        try:
            yield session
        finally:
            session.close()

    app.dependency_overrides[get_db] = override_get_db
    yield TestClient(app)
================================================
Keep test independent
Test user fixture
================================================

from jose import jwt
from app.config import settings

@pytest.fixture
def test_user(client):
    user_data = {"email": "hellow123@gmail.com",
                 "password": "password"}
    res = client.post("/users/", json=user_data)

    new_user = res.json()
    new_user['password'] = user_data['password']
    return new_user
```

```
def test_login_user(client, test_user):
    res = client.post('/login', data={'username':
test_user['email'], 'password': test_user['password']})
    print(res.json())
    res_login = schemas.Token(**res.json())
    print(res_login)


    payload = jwt.decode(res_login.access_token,
settings.secret_key, algorithms=[settings.algorithm])
    id = payload.get("user_id")
    assert id == test_user['id']
    assert res_login.token_type == "bearer"
    assert res.status_code == 200


def test_create_user(client):
    res = client.post('/users/', json={'email':
"hello123@gmail.com", 'password': "password"})
    new_user = schemas.UserOut(**res.json())
    assert new_user.email == "hello123@gmail.com"
    assert res.status_code == 201


================================================
# Define fixture
tests/database.py => tests/conftest.py
conftesst.py [all pytest.fixture merge to conftest.py]
================================================
@pytest.mark.parametrize(
    "email, password, status_code",
    [
        ("hello123@gmail.com", "password", 403),
        ('wrongemail@gmail.com', 'password123', 403),
        ('sanjeev@gmail.com', 'wrongpassword', 403),
        ('wrongemail@gmail.com', 'wrongpassword', 403),
        (None, 'password123', 422),
        ('sanjeev@gmail.com', None, 422),
    ]
)
def test_incorrect_login(test_user, client, email, password,
status_code):
    res = client.post(
        '/login',
        data={"username": email, "password": password,
status_code: status_code}
    )

    assert res.status_code == status_code


================================================
test_posts
in tests/conftest.py
================================================
```

```
@pytest.fixture
def token(test_user):
    return create_access_token({"user_id": test_user['id']})


@pytest.fixture
def authorized_client(client, token):
    client.headers = {
        **client.headers,
        "Authorization": f"Bearer {token}"
    }

    return client
```

Deployment

Push to Github Repository

> - pip freeze > requirements.txt
> - pip install -r requirements.txt(for others)

```
-> git init
-> git config --global user.email "you@example.com"
-> git config --global user.name "your name"
-> git add --all
-> git commit -m "initial commit"
-> git branch -M main
-> git remote add origin
`https://github.com/ericarthuang/CS_WD_PythonFastAPI.git`
-> git push -u origin main
```

> - How to fix 'fatal: remote origin already exists
>
> -> git remote -v -> git remote remove "name"

Heroku Deploy

> -setup Procfile
> -> web: uvicorn app.main:app --host=0.0.0.0 --port=${PORT:-5000}

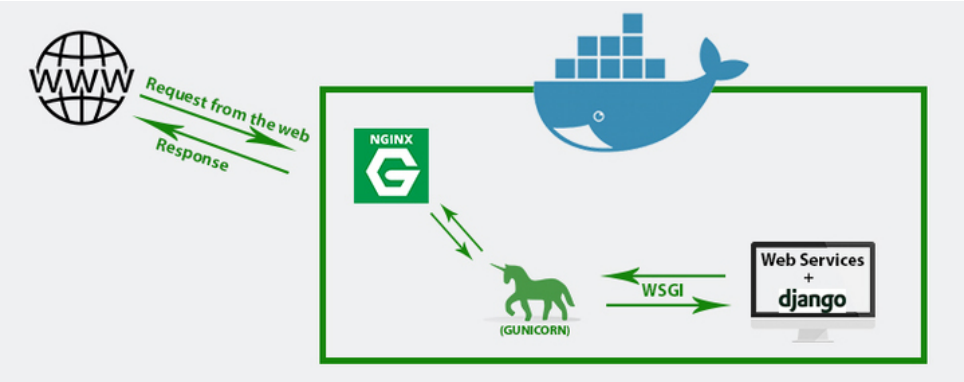
> - Heroku Postgres
> - Adding a Postgres database in Terminal
>
> heroku addons:create heroku-postgresql:<PLAN_NAME>

> `heroku --version`
> `heroku login`
> `heroku create [name of app]`

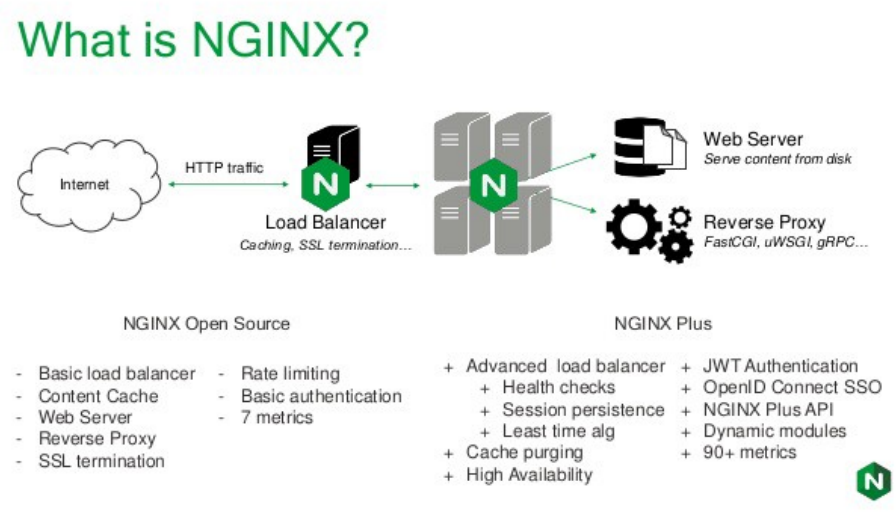
> `git remote -v` `git push heroku main`

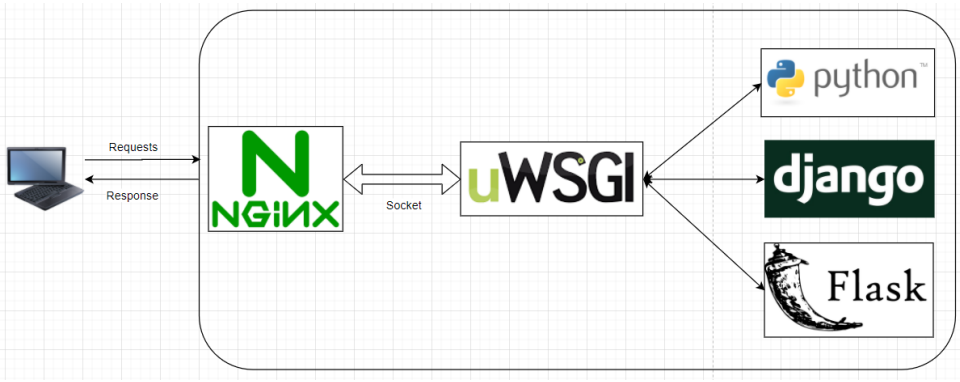
Gunicorn

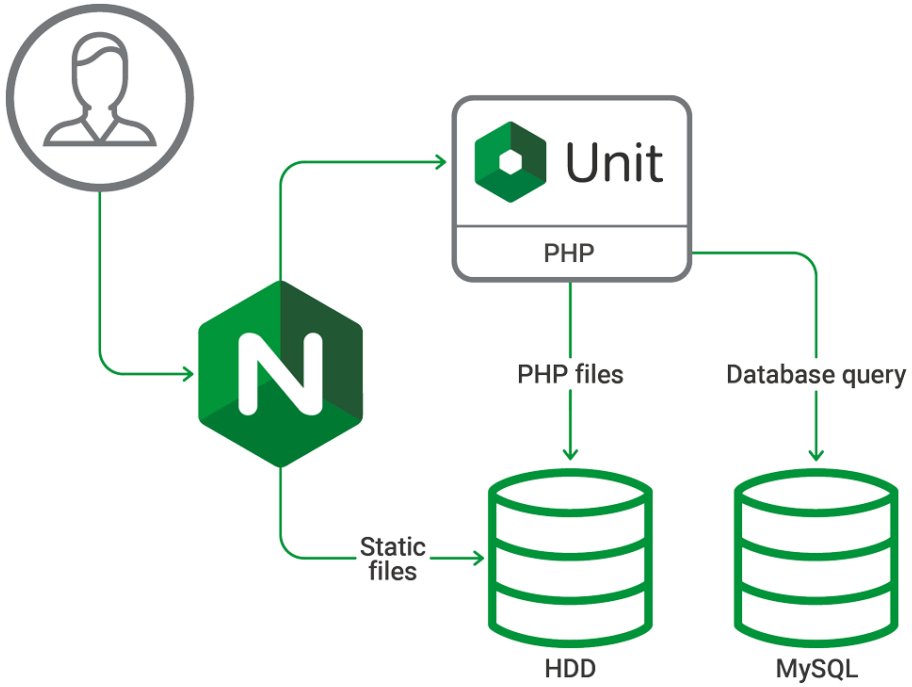
> - pip install gunicron
> - pip install httptools
> - python.exe -m pip install --upgrade pip
> - pip install uvloop
> - gunicron -w 4 -k uvicorn.workers.UvicornWorker app.main:app --bind 0.0.0.0

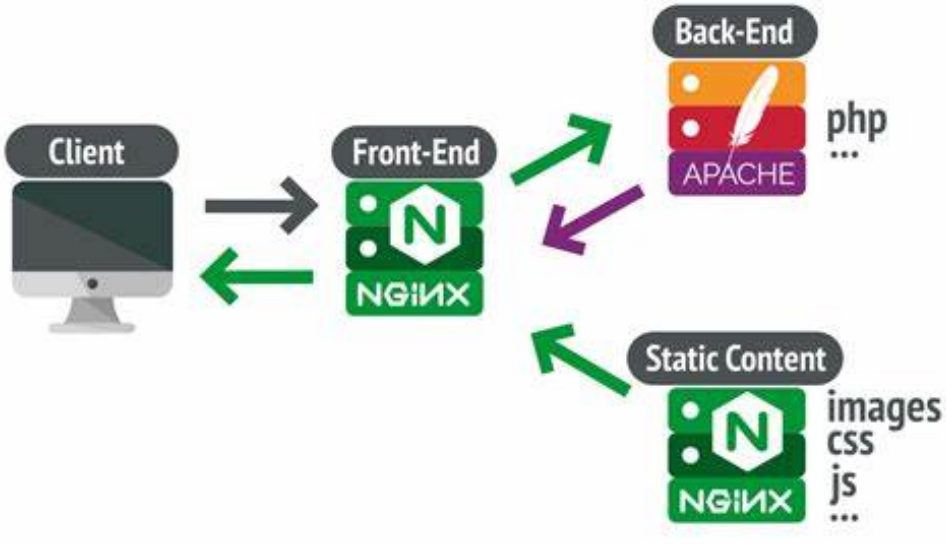


Nginx - Webserver: Act as Proxy / Handle SSL Ternimation

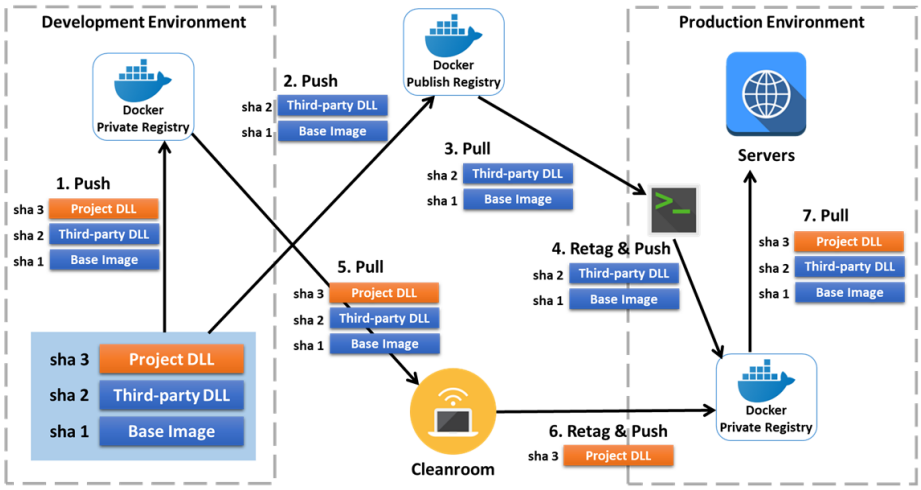


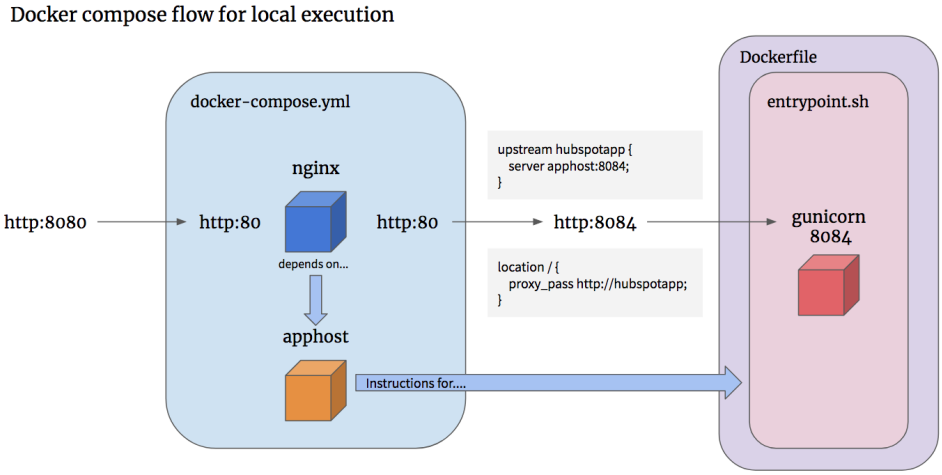

Requests
Response
NGINX
Socket
uWSGI
python
django
Flask


Unit
PHP
PHP files
Database query
Static files
HDD
MySQL


Client
Front-End
NGINX
Back-End
APACHE
php
Static Content
NGINX
images
css
js

Docker





- Creaate Dockerfile

-> CMD: docker build -t fastapi .

- Creaate docker-compose

```
-> CMD: docker-compose up -d
-> CMD: docker-compose down
```

- Create docker-compose-prod.yml
- Change docker-compose.yml to docker-compose-dev.yml

- CMD: docker-compose -f docker-compose-dev up -d
- CMD: docker-compose -f docker-compose-dev down

CI/CD Config - Github Actions

- Create `.github/workflows` in root directory
- Create `bulid-deploy.yml`

> - Link `GitHub Actions` with `Docker Hub Repository`

-- Memo End --