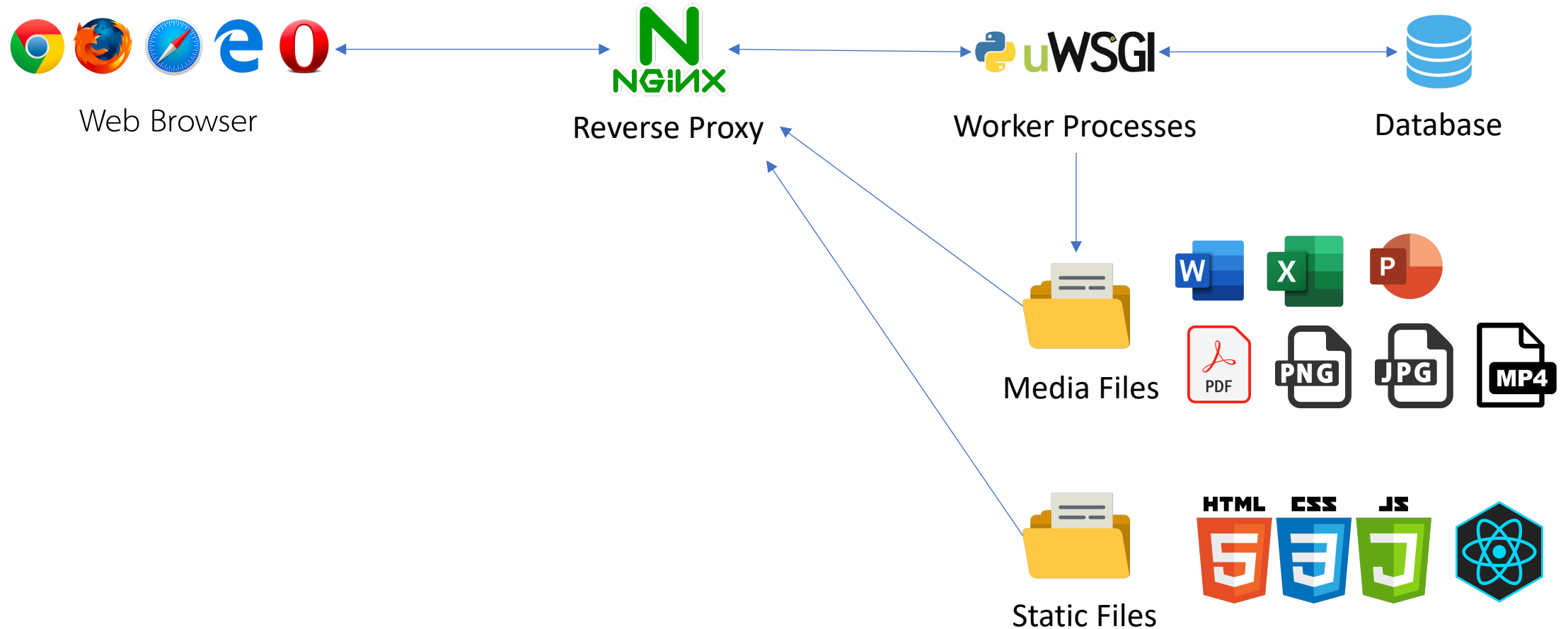


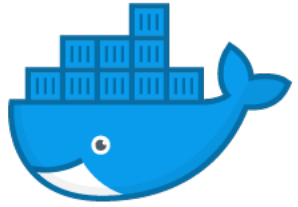
Django Deployment

<https://github.com/WasinTh/tutorial-docker-deploy>

<https://github.com/WasinTh/tutorial-docker-build>

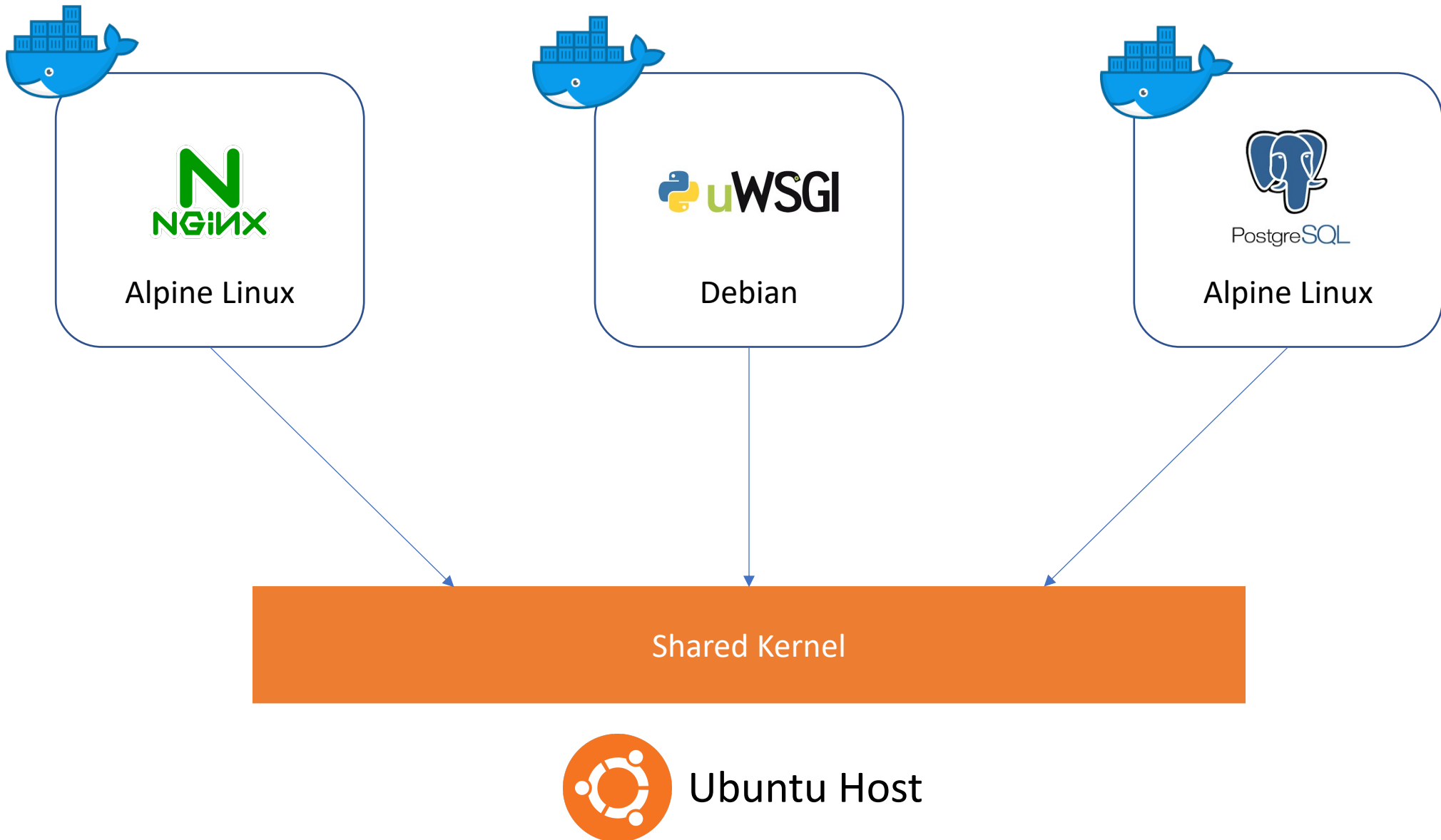
Django Deployment Architecture



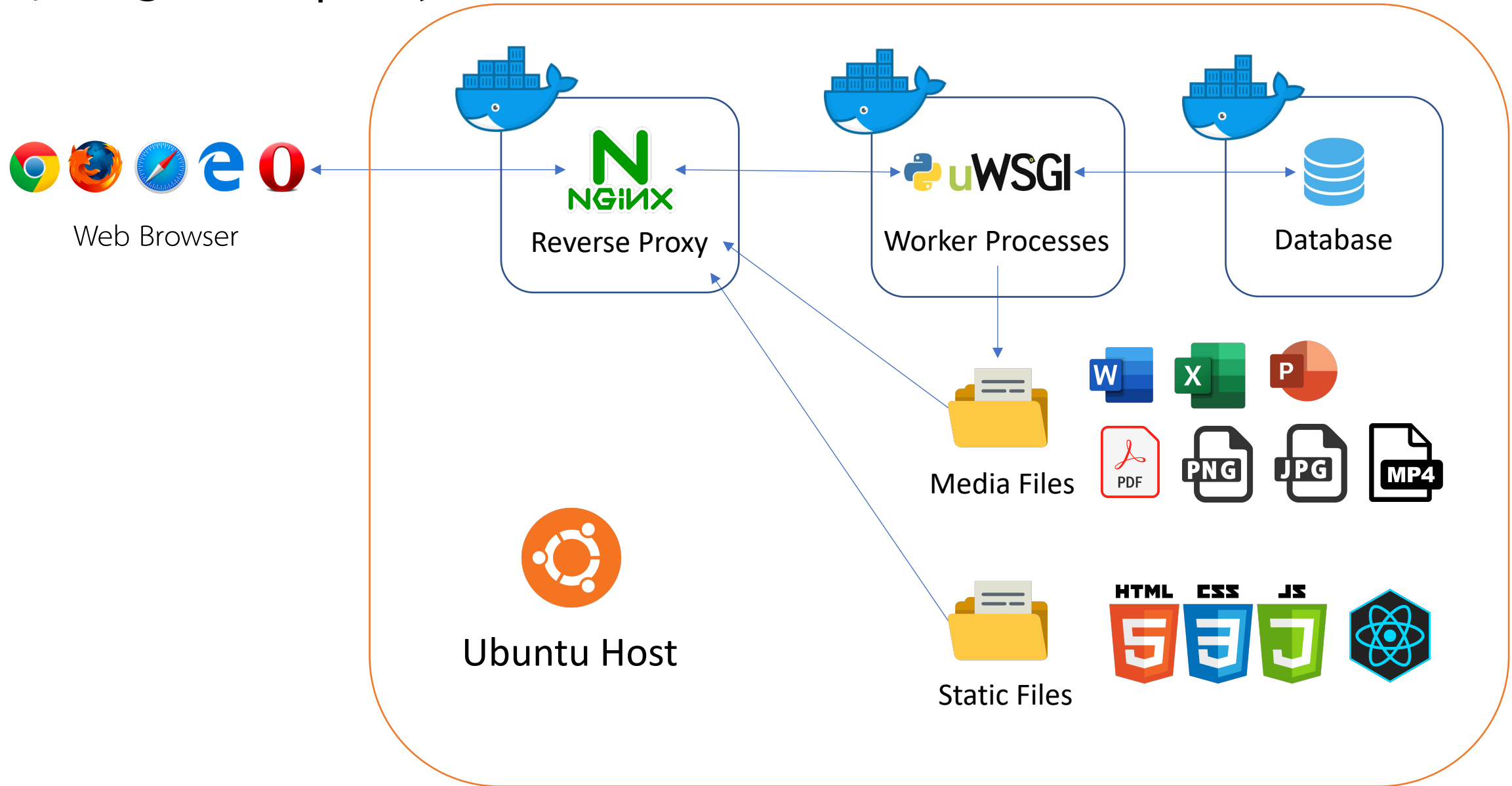


Introduction to Docker

- Docker คือเทคโนโลยี Container ชนิดหนึ่ง
 - Container คือกลุ่มของ process ที่ทำงานร่วมกัน
 - ในแต่ละ Container จะประกอบไปด้วย OS และ process ภายในของตนเอง
 - Container ทุกตัวจะ share host resource เดียวกัน



Django Deployment Architecture (Docker)



ขั้นตอนการใช้งาน Docker

- Development Server
 - เขียน Dockerfile เพื่อกำหนด step การสร้าง container
 - Build container image จาก Dockerfile
 - นำ container image ไปฝากไว้ที่ Registry Server เช่น Docker Hub
- Production Server
 - ติดตั้ง Docker
 - สร้างไฟล์ docker-compose.py
 - สำหรับ Pull Container Image ที่ต้องการและการตั้งค่า container ต่าง ๆ
 - สั่งรันไฟล์ docker-compose

สร้าง Docker Container Image

ปรับ Django Project ให้พร้อมใช้งาน Docker

- ทำให้ Django Project สามารถ configure ได้ผ่านทาง environment variable
 - การตั้งค่า Project ของ docker ต้องทำผ่าน Environment Variable เท่านั้น
- ขั้นตอนการตั้งค่า Django Project
 - ติดตั้ง package Django-environ
 - ปรับแต่งไฟล์ settings.py ให้ใช้ค่าจาก environment variable

```
pip install django-environ
```




ปรับ Django Project ให้พร้อมใช้งาน Docker (2)

personal_finance/settings.py

```
import os
import environ
from pathlib import Path

# ...

env_base = environ.Path(__file__) - 2
environ.Env.read_env(env_file=env_base('.env'))
env = environ.Env(
    DEBUG=(bool, True),
)

DEBUG = env('DEBUG')

ALLOWED_HOSTS = tuple(env.list('ALLOWED_HOSTS', default=[]))

# ...

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATICFILES_DIRS = [
    os.path.normpath(os.path.join(BASE_DIR, "staticfiles")),
]
```

สร้างโฟลเดอร์ staticfiles

ปรับ React ให้พร้อมใช้งาน Docker

- ทำให้ React Application ตั้งค่าผ่านทาง Environment Variable
- React สามารถ Load Environment Variable ผ่านทาง คำสั่ง

```
let myVar = process.env.REACT_APP_MY_VAR
```

- ปัญหาหลักของ React คือ URL Endpoint เนื่องจาก
 - เมื่อรันโปรแกรมในเครื่องที่ใช้พัฒนาจะต้องตั้ง URL ไปที่ localhost:8000
 - ตอนอยู่บน production ต้องตั้ง URL ไปที่ IP จริง



ปรับ React ให้พร้อมใช้งาน Docker (2)

- สร้างไฟล์ src/Utils.js เพื่อสร้าง Method สำหรับ generate URL

src/Utils.js

```
export function createURL(endPoint) {  
  let baseUrl = process.env.REACT_APP_BASE_URL;  
  console.log(baseUrl);  
  return baseUrl != null ? `${baseUrl}${endPoint}` : endPoint;  
}
```

- ปรับคำสั่ง Axios ทั้งหมดให้ใช้ URL ใหม่ และใช้ Utils.js เข้าช่วย เช่น

```
axios.post(createURL('/api/api-token-auth/'), {...})
```



สร้าง Django Container Image

- React

- Build react project ด้วยคำสั่งต่อไปนี้ จะได้ไฟล์ทั้งหมดอยู่ใน folder “build”

```
npm run build
```

- Copy โฟลเดอร์ build/static/js, build/static/css, build/static/media ออกมาอยู่ที่ build/js, build/css และ build/media ตามลำดับ

- Django

- แก้ไข personal_finance/urls.py ให้เพิ่ม /api/ ไว้ด้านหน้า URL ทั้งหมด (เพื่อความง่ายในการตั้งค่า Nginx)
- Download Folder docker_conf ลงมาเก็บไว้ในโฟลเดอร์ของ Django
- สร้าง folder ชื่อ staticfiles
- Copy ไฟล์จากโฟลเดอร์ <React>/build/ ทั้งหมดมาไว้ในโฟลเดอร์ <Django>/staticfiles



สร้าง Django Container Image (2) – แก้ไข urls.py

personal_finance/urls.py

```
# ...

urlpatterns = [
    path('api/admin/', admin.site.urls),
    path('api/account/current-balance/', current_balance_view),
    path('api/account/transaction-list/', transaction_list),
    path('api/account/transaction/', TransactionView.as_view()),
    path('api/account/customer/', CustomerView.as_view()),
    path('api/api-token-auth/', obtain_jwt_token),
    path('api/', include(router.urls)),
    path('api/swagger/', schema_view.with_ui('swagger', cache_timeout=0)),
]
```



สร้าง Django Container Image (3) – สร้าง Dockerfile

```
FROM python:3.9-buster
```

```
RUN apt-get update && \
    apt-get install -y \
    vim-tiny \
    supervisor \
    pkg-config && \
    rm -rf /var/lib/apt/lists/*
```

```
RUN mkdir -p /var/log/uwsgi && \
    mkdir -p /var/log/supervisor && \
    mkdir -p /var/run/uwsgi && \
    mkdir -p /usr/src/django_app
```

```
COPY docker_conf/uwsgi.ini /usr/src/uwsgi.ini
COPY docker_conf/supervisor.conf /etc/supervisor.conf
COPY docker_conf/docker-entrypoint.sh /usr/src/django_app
COPY docker_conf/requirements.txt /usr/src/requirements.txt
```

```
WORKDIR /usr/src
```

```
RUN pip install -r requirements.txt --no-cache-dir
```

```
COPY . /usr/src/django_app
```

```
WORKDIR /usr/src/django_app
```

```
ENTRYPOINT ["/usr/src/django_app/docker-entrypoint.sh"]
```

```
EXPOSE 8000 8001
```

```
CMD ["supervisord", "-c", "/etc/supervisor.conf"]
```

Dockerfile

<https://github.com/WasinTh/tutorial-docker-build>



สร้าง Django Container Image (3)

- สร้าง Account ใหม่ และสร้าง Repository ใหม่ที่ <https://hub.docker.com/>

The image displays two screenshots of the Docker Hub interface. The left screenshot shows the 'Create Repository' page. The username 'wasinth' is entered in the dropdown, and the repository name 'finance-demo' is typed. The 'Public' visibility option is selected. The right screenshot shows the newly created repository page for 'wasinth / finance-demo'. It includes a 'General' tab, a description field, and a 'Public View' button. The 'Last pushed' status is 'never'.






สร้าง Django Container Image (4)

```
docker build -t <repository_name> .
```


```
docker login -u "myusername" -p "mypassword" docker.io
```



```
docker push <repository_name>
```

 **wasinth / finance-demo**
This repository does not have a description 
 Last pushed: 35 minutes ago

Docker commands [Public View](#)
To push a new tag to this repository,

```
docker push wasinth/finance-demo:tagname
```

Tags and Scans  **VULNERABILITY SCANNING - DISABLED** [Enable](#)
This repository contains 1 tag(s).

TAG	OS	PULLED	PUSHED
 latest		35 minutes ago	35 minutes ago

[See all](#)

Recent builds
Link a source provider and run a build to see build results here.

Deploy Docker Container Image



ติดตั้ง Docker บน Production

- Remote เข้าไปยังเครื่อง Server จากนั้น Run คำสั่งต่อไปนี้

```
sudo apt-get update
sudo apt-get remove docker docker-engine docker.io
sudo apt install docker.io docker-compose -y
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER
```

- Download docker_conf Folder มาบน server
 - <https://github.com/WasinTh/tutorial-docker-deploy>



สร้างไฟล์ docker-compose.yml

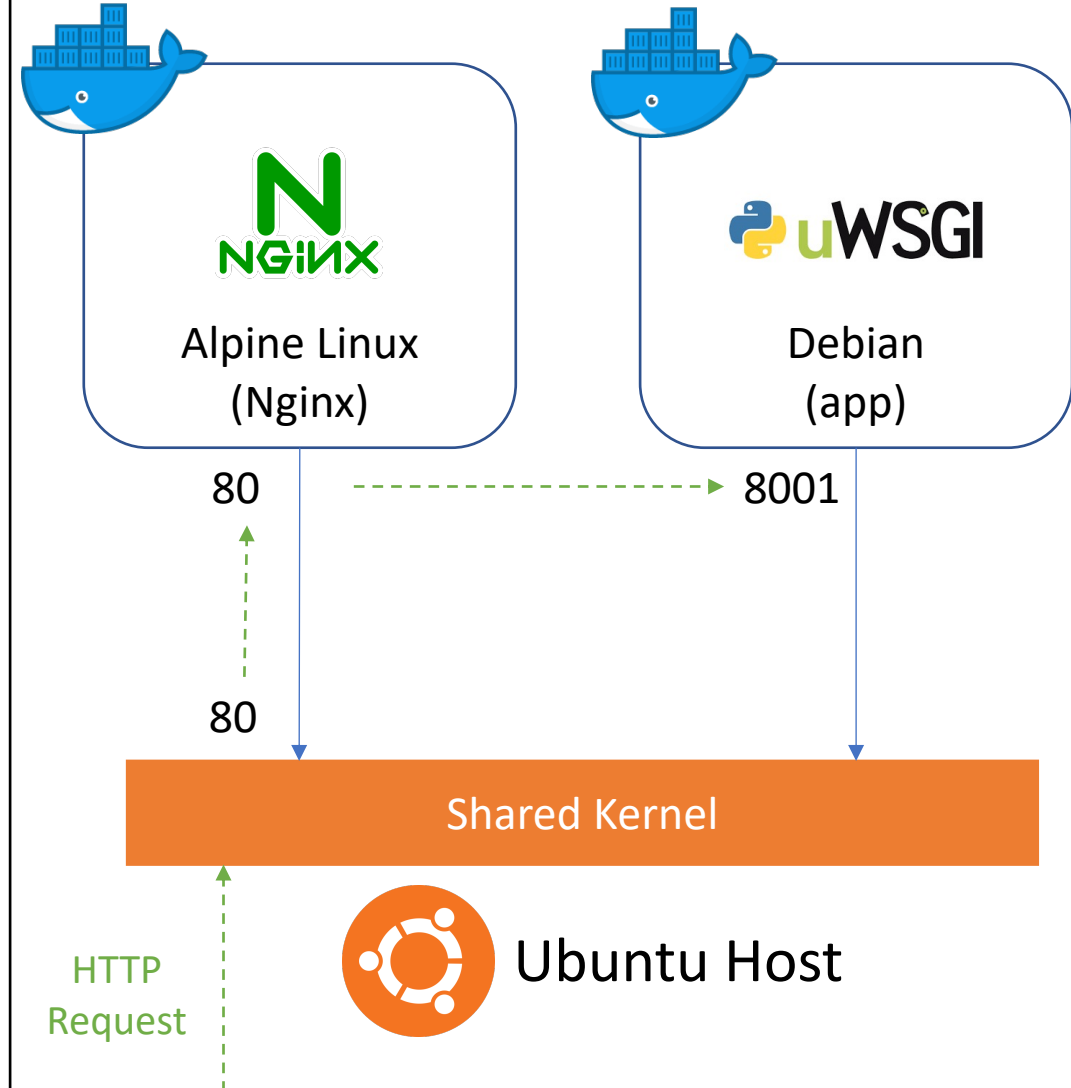
```
version: '2'

services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - 80:80
    links:
      - app
    volumes:
      - /etc/localtime:/etc/localtime:ro # Set container timezone
      - static-volume:/web-static
      - ./docker_conf/nginx.conf:/src/nginx.conf
      - ./docker_conf:/src
    command: ./src/run_nginx.sh
    environment:
      - DOMAIN_NAME=${DOMAIN_NAME}

  app:
    restart: always
    container_name: app
    environment:
      - PROJECT=${PROJECT_NAME}
      - WORKER_PROCESS=${WORKER_PROCESS}
      - DEBUG=${DEBUG}
      - ALLOWED_HOSTS=${DOMAIN_NAME}
      - DJANGO_MANAGEPY_MIGRATE=${DJANGO_MANAGEPY_MIGRATE}
    image: ${DOCKER_APP_IMAGE}
    volumes:
      - /etc/localtime:/etc/localtime:ro # Set container timezone
      - static-volume:/usr/src/django_app/static
    expose:
      - 8001

volumes:
  static-volume:
```

docker-compose.yml





Start Server

- เริ่มการทำงานของ Server ด้วยคำสั่ง

```
>> docker-compose pull  
>> docker-compose up -d
```

```
>> docker-compose pull && docker-compose up -d
```

คำสั่ง Docker ที่น่าสนใจ

- แสดง Logs ของ Server

```
>> docker-compose logs <container_name>
```

```
>> docker-compose logs -f --tail 100 nginx
```

- Restart Server – ใช้กรณีแก้ไข config ไฟล์ เช่น nginx.conf

```
>> docker-compose restart nginx
```

- เข้าสู่ Shell ของ Container

```
>> docker-compose exec nginx /bin/bash
```