

# **Docker Compose Setup for Flask, MongoDB, and Nginx**

This KB provides step-by-step instructions to set up a Docker Compose environment that includes a Flask application, MongoDB database, and Nginx web server.

## **Prerequisites**

- A machine with Docker and Docker Compose installed.
- Basic understanding of Docker, Docker Compose, and containerized services.

## **Step 1: Installing Docker**

**For Ubuntu-based systems:**

1. Update your packages:

---

```
sudo apt update
```

```
sudo apt upgrade
```

---

2. Install required packages:

---

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

---

3. Add Docker's official GPG key:

---

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

---

4. Add the Docker repository:

---

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

---

5. Install Docker:

---

```
sudo apt update  
sudo apt install docker-ce
```

---

6. Verify Docker installation:

---

```
docker --version
```

---

## Step 2: Installing Docker Compose

1. Download the latest Docker Compose release:

```
sudo curl -L  
"https://github.com/docker/compose/releases/download/v2.20.2/docker-compose-$(uname  
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Set permissions for Docker Compose:

```
sudo chmod +x /usr/local/bin/docker-compose
```

3.  

```
sudo chmod +x /usr/local/bin/docker-compose
```

4. Verify Docker Compose installation:

```
docker-compose --version
```

## Step 3: Setting Up the Docker Compose File

Now that Docker and Docker Compose are installed, let's set up the `docker-compose.yml` file.

1. Create the Project Directory:

```
mkdir myapp  
cd myapp
```

## 2. Directory Structure:

---

```
myapp/
├── app/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── app.py
│   └── wsgi.py
├── nginx/conf.d
│   └── app.conf
└── docker-compose.yml
```

---

## 3. Create the docker-compose.yml file: Use the following content in the docker-compose.yml file:

---

```
version: '3'
services:

  flask:
    build:
      context: app
      dockerfile: Dockerfile
    container_name: flask
    image: digitalocean.com/flask-python:3.6
    restart: unless-stopped
    environment:
      APP_ENV: "prod"
      APP_DEBUG: "true"
      APP_PORT: 5000
      FLASK_ENV: "development"
      MONGODB_DATABASE: flaskdb
      MONGODB_USERNAME: flaskuser
      MONGODB_PASSWORD: "qazxsw"
      MONGODB_HOSTNAME: mongodb
    volumes:
      - appdata:/var/www
    depends_on:
      - mongodb
    networks:
```

- frontend
- backend

mongodb:

image: mongo:4.0.8  
container\_name: mongodb  
restart: unless-stopped  
command: mongod --auth  
environment:  
MONGO\_INITDB\_ROOT\_USERNAME: mongoduser  
MONGO\_INITDB\_ROOT\_PASSWORD: your\_mongodb\_root\_password  
MONGO\_INITDB\_DATABASE: flaskdb  
MONGODB\_DATA\_DIR: /data/db  
MONGODB\_LOG\_DIR: /dev/null  
volumes:  
- mongoddata:/data/db  
networks:  
- backend  
ports:  
- "27017:27017"

webserver:

image: nginx:latest  
container\_name: webserver  
restart: unless-stopped  
ports:  
- "80:80"  
- "443:443"  
volumes:  
- ./nginx/conf.d:/etc/nginx/conf.d  
- nginxdata:/var/log/nginx  
depends\_on:  
- flask  
networks:  
- frontend

networks:

frontend:

driver: bridge

backend:

driver: bridge

volumes:

mongoddata:

```
    driver: local
  appdata:
    driver: local
  nginxdata:
    driver: local
```

---

## Step 4: Writing the Dockerfiles and Nginx configuration File

Flask Dockerfile ([app/Dockerfile](#)):

---

```
FROM python:3.6.8-alpine3.9

ENV GROUP_ID=1000 \
    USER_ID=1000

WORKDIR /var/www/

ADD . /var/www/
RUN pip install -r requirements.txt
RUN pip install gunicorn

RUN addgroup -g $GROUP_ID www
RUN adduser -D -u $USER_ID -G www www -s /bin/sh

USER www

EXPOSE 5000

CMD [ "gunicorn", "-w", "4", "--bind", "0.0.0.0:5000", "wsgi"]
```

---

Nginx Configuration file ([nginx/conf.d/app.conf](#)):

---

```

upstream app_server {
    server flask:5000;
}

server {
    listen 80;
    server_name _;
    error_log /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    client_max_body_size 64M;

    location / {
        try_files $uri @proxy_to_app;
    }

    location @proxy_to_app {
        gzip_static on;

        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $http_host;
        proxy_buffering off;
        proxy_redirect off;
        proxy_pass http://app_server;
    }
}

```

---

## Application codes

`app/requirements.txt`

```

-----
Flask==1.0.2
Flask-PyMongo==2.2.0
requests==2.20.1
-----

```

app/app.py

```
-----

import os
from flask import Flask, request, jsonify
from flask_pymongo import PyMongo

application = Flask(__name__)

application.config["MONGO_URI"] = 'mongodb://' +
os.environ['MONGODB_USERNAME'] + ':' + os.environ['MONGODB_PASSWORD']
+ '@' + os.environ['MONGODB_HOSTNAME'] + ':27017/' +
os.environ['MONGODB_DATABASE']

mongo = PyMongo(application)
db = mongo.db

@app.route('/')
def index():
    return jsonify(
        status=True,
        message='Welcome to the Dockerized Flask MongoDB app!'
    )

@app.route('/todo')
def todo():
    _todos = db.todo.find()

    item = {}
    data = []
    for todo in _todos:
        item = {
            'id': str(todo['_id']),
            'todo': todo['todo']
        }
        data.append(item)

    return jsonify(
```



```

        status=True,
        data=data
    )

@app.application.route('/todo', methods=['POST'])
def createTodo():
    data = request.get_json(force=True)
    item = {
        'todo': data['todo']
    }
    db.todo.insert_one(item)

    return jsonify(
        status=True,
        message='To-do saved successfully!'
    ), 201

if __name__ == "__main__":
    ENVIRONMENT_DEBUG = os.environ.get("APP_DEBUG", True)
    ENVIRONMENT_PORT = os.environ.get("APP_PORT", 5000)
    application.run(host='0.0.0.0', port=ENVIRONMENT_PORT,
debug=ENVIRONMENT_DEBUG)

```

-----

app/wsgi.py

-----

```

from app import application

if __name__ == "__main__":
    application.run()

```

-----

## Step 5: Running Docker Compose

1. Build and Start the Containers:

Run the following command in the root of your project directory:

---

```
docker-compose up --build
```

---

2. Verify Containers Are Running:

To ensure all services are up and running:

---

```
docker-compose ps
```

---

3. **Access the Application:**

Flask application: <http://localhost:5000>

Webserver: <http://localhost> for HTTP

---

## Step 6: Managing the Environment

To Stop the Containers:

---

```
docker-compose down
```

---

To Restart the Containers:

---

```
docker-compose restart
```

---

## Step 7: Persistent Data with Volumes

The volumes in the `docker-compose.yml` file ensure data persistence across container restarts. These volumes store the database data and application data:

- **MongoDB Volume:** `/data/db` is where MongoDB stores its data.
- **App Volume:** `/var/www` contains the application files.
- **Nginx Volume:** `/var/log/nginx` stores the Nginx logs.

