Dockerized Flask Application with Gunicorn

This document explains **your complete Docker + Flask + Gunicorn setup**, line by line, so you clearly understand **what is happening, why it is needed, and how everything connects**. This is written from a **beginner-to-intermediate DevOps perspective**.

---

1. Project Overview

You have built a **production-style Flask application** that: - Runs inside a **Docker container** - Uses **Gunicorn** instead of Flask's development server - Exposes the app on a host machine using **port mapping**

This is the **correct industry approach** for running Python web apps.

---

2. Application Code (`launch.py`)

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def run():
    return "{\"message\":\"Hello World Python. This is my first docker
project\"}"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=int("5000"), debug=True)
```

Explanation

`from flask import Flask`

Imports the Flask framework, which is used to build web applications in Python.

`app = Flask(__name__)`

Creates a Flask application object. - `__name__` helps Flask locate files and resources

`@app.route("/")`

Defines a **route (URL endpoint)**. - `/` means the **root URL**

`def run():`

This function executes when someone accesses `/`.

Return statement

Returns a **JSON-style response**:

```
{"message": "Hello World Python. This is my first docker project"}
```

`if __name__ == "__main__":`

This block runs **only when the file is executed directly**. - Important for **local testing**

`host="0.0.0.0"`

- Makes the app accessible **outside the container**

`port=5000`

- Flask listens on port 5000 inside the container

⚠️In production, Flask's built-in server is **not used**. Gunicorn replaces it.

---

3. Requirements File (`requirements.txt`)

```
flask==2.2.5
gunicorn==21.2.0
```

Why this file is needed

- Lists all Python dependencies
- Ensures **consistent builds** across environments

Flask

Used to create the web application.

Gunicorn

- A **production-grade WSGI server**
- Handles multiple requests efficiently
- Much faster and safer than Flask's default server

4. Dockerfile (Heart of the Project)

```
FROM python:3.9-slim
```

Base Image

- Uses a **lightweight Python image**
- `slim` reduces image size and attack surface

```
WORKDIR /app
```

Working Directory

- Sets `/app` as the working directory inside the container
- All commands run from here

```
COPY requirements.txt .
```

Copy Dependencies First (Best Practice)

- Allows Docker **layer caching**
- Dependencies are installed only when requirements change

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

Install Dependencies

- `--no-cache-dir` keeps image size small
- Installs Flask and Gunicorn

```
COPY . .
```

Copy Application Code

- Copies `launch.py` and all other files into `/app`

```
EXPOSE 5000
```

Port Exposure

- Documents that the container listens on port **5000**
- Does NOT publish the port by itself

```
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "launch:app"]
```

Application Startup Command

Gunicorn Breakdown

- `gunicorn` → WSGI server
- `-w 4` → 4 worker processes
- `-b 0.0.0.0:5000` → Bind to all interfaces
- `launch:app` →
- `launch` = Python file
- `app` = Flask object

✅This is **production-ready configuration**.

```
HEALTHCHECK CMD curl --fail http://localhost:5000/ || exit 1
```

Health Check

- Docker periodically checks if app is alive
- Marks container as **unhealthy** if it fails

Used by: - Docker - Kubernetes - Cloud platforms

---

5. Docker Build Command

```
docker build -t sourav/first-prod-guni-python:0.0.1 .
```

Explanation

- `docker build` → Builds an image

- `-t` → Tags the image
- `sourav/first-prod-guni-python` → Image name
- `0.0.1` → Version
- `.` → Current directory as build context

---

6. Docker Run Command

```
docker container run -p 5003:5000 sourav/first-prod-guni-python:0.0.1
```

Port Mapping

- `5003` → Host machine port
- `5000` → Container port

Result

Access the app in browser:

```
http://localhost:5003
```

---

7. Complete Request Flow

```
Browser → localhost:5003
        ↓
Docker Port Mapping
        ↓
Gunicorn (5000)
        ↓
Flask App (launch.py)
        ↓
Response Returned
```

---

8. Why Gunicorn Instead of Flask Server?

Flask Dev Server	Gunicorn
Single-threaded	Multi-worker

Flask Dev Server	Gunicorn
Not secure	Production-grade
Debug only	High performance

---

9. Best Practices You Followed ✅

- ✔️Used slim base image
- ✔️Dependency caching
- ✔️Gunicorn for production
- ✔️Healthcheck added
- ✔️Versioned Docker image

---

10. Deep Dive: Gunicorn (Very Important for Knowledge & Interviews)

This section explains **everything related to Gunicorn** that is used or implied in your project. Read this carefully — this is core backend + DevOps knowledge.

10.1 What is WSGI?

WSGI stands for **Web Server Gateway Interface**.

In simple words:

> WSGI is a **standard rulebook** that allows a web server to talk to a Python web application.

Why WSGI Exists

Browsers cannot talk directly to Python code.

```
Browser → Web Server → Python Application
```

WSGI defines **how requests go in and responses come out**.

Flask, Django, FastAPI (via adapters) all follow WSGI.

10.2 Flask vs Gunicorn (Very Important)

Flask Built-in Server

- Only for **development**
- Single process
- Cannot handle high traffic
- No request management

Flask clearly says:

> Do NOT use this server in production.

Gunicorn

- Production-grade **WSGI server**
- Manages multiple worker processes
- Handles concurrency
- Stable and battle-tested

Your project correctly uses **Gunicorn instead of Flask server**.

10.3 What Exactly is Gunicorn?

Gunicorn = **Green Unicorn** ♊

It is: - A **Python WSGI HTTP Server** - Sits between client and Flask app

```
Client → Gunicorn → Flask App
```

Gunicorn: - Accepts HTTP requests - Sends them to Flask via WSGI - Returns responses back to client

---

10.4 Understanding `launch:app`

From your Dockerfile:

```
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "launch:app"]
```

Breakdown:

- `launch` → Python file name (`launch.py`)
- `app` → Flask application object

Gunicorn internally does:

```
from launch import app
```

That's how it loads your Flask app.

10.5 What are Worker Processes?

A **worker** is a separate Python process that handles requests.

```
Gunicorn Master Process
   ├── Worker 1
   ├── Worker 2
   ├── Worker 3
   └── Worker 4
```

In your project:

```
-w 4
```

means **4 worker processes**.

10.6 Why Multiple Workers Are Needed

Each worker: - Can handle **one request at a time** (sync workers)

If only 1 worker: - 1 slow request blocks others

With 4 workers: - 4 requests handled in parallel

This improves: - Performance - Reliability - Availability

10.7 How Many Workers Should Be Used?

General formula:

```
workers = (2 × CPU cores) + 1
```

Example: - 2-core CPU → 5 workers - 4-core CPU → 9 workers

In Docker containers, **4 workers** is a safe default for learning and small apps.

10.8 Gunicorn Binding (`-b` option)

```
-b 0.0.0.0:5000
```

Meaning: - `0.0.0.0` → Accept requests from anywhere - `5000` → Listening port inside container

This is **mandatory** inside Docker.

If you use `127.0.0.1`, Docker port mapping will NOT work.

10.9 Request Flow with Gunicorn (Very Important)

```
Browser
  ↓
Docker Port Mapping (5003 → 5000)
  ↓
Gunicorn Master Process
  ↓
One Worker Process
  ↓
Flask Route (/)
  ↓
Response
```

10.10 Gunicorn vs Nginx (Conceptual)

Gunicorn	Nginx
WSGI server	Web server / Reverse proxy
Runs Python code	Serves static files
Handles app logic	Handles SSL, caching

In real production:

```
Client → Nginx → Gunicorn → Flask
```

Your project is **one step before full production**, which is perfect for learning.

10.11 Gunicorn Worker Types (Extra Knowledge)

Gunicorn supports different worker types:

- `sync` (default) → Simple, safe
- `gevent` → Async
- `eventlet` → Async
- `uvicorn` → ASGI apps

Your project uses: - Default **sync workers** (best for beginners)

10.12 What Happens If a Worker Crashes?

Gunicorn master process: - Detects worker failure - Automatically restarts it

This makes Gunicorn: - Fault tolerant - Production-safe

10.13 Why Gunicorn is Perfect for Docker

- Lightweight
- No OS dependency
- Easy to configure
- Works well with Kubernetes

That’s why most Python Docker images use:

```
gunicorn app:app
```

---

11. Interview-Ready One-Liners (Gunicorn)

- Gunicorn is a production WSGI server for Python apps
- WSGI connects web servers and Python applications
- Workers allow parallel request handling
- Flask server is not suitable for production
- Gunicorn master manages worker lifecycle

---

12. Updated Final Summary

You have successfully built a **production-ready Dockerized Flask application** using **Gunicorn (WSGI server)**.

This setup is suitable for: - Cloud deployment (AWS, Azure, GCP) - Kubernetes - DevOps CI/CD pipelines

---

If you want, next we can: - Convert this into **Docker Compose** - Add **logging & env variables** - Deploy to **AWS EC2 / ECS** - Add **Nginx reverse proxy**