

I started by “dockerizing” my Flask app on top of the official `python:3.11-slim` image. In my `Dockerfile` I

1. set `PYTHONDONTWRITEBYTECODE=1` and `PYTHONUNBUFFERED=1` so no `.pyc` files are written and logs stream straight to stdout,
2. created a `/code` workdir,
3. copied in `requirements.txt` and ran `pip install --upgrade pip` && `pip install -r ...` (then wiped the pip cache),
4. copied the rest of my app into `/code`,
5. exposed port 8000, and
6. ended with `CMD ["python", "app.py"]`.
I also `.dockerignore`-ed `fly.toml`, my `.git/` folder and any local SQLite files.

Inside `app.py` I have three routes:

- `GET /` renders `index.html` via Jinja2,
- `POST /press` stamps `last_press_time = time.time()` and returns `{ status: 'ok' }`,
- `GET /status` computes `pressed = (now - last_press_time) < PRESS_DURATION` (3 sec) and returns `{ pressed: true|false }`.

My `index.html` is just a full-screen flex container with a big, glowing green button. Its `onclick` calls

```
js
CopyEdit
fetch('/press', { method: 'POST' })
  .then(r => r.json())
  .then(j => console.log(j.message));
```

so every click resets the timer on the server.

On the networking side, I run the container with `-p 8000:8000` (or on Fly.io I've set `internal_port = 8000` in `fly.toml`), so the Flask API is reachable at

cpp
CopyEdit
`http://<docker-host-ip>:8000`

Docker files(compatible fly.io)

Dockerfile:

```
ARG PYTHON_VERSION=3.11-slim

FROM python:${PYTHON_VERSION}

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

RUN mkdir -p /code

WORKDIR /code

COPY requirements.txt /tmp/requirements.txt
RUN set -ex && \
    pip install --upgrade pip && \
    pip install -r /tmp/requirements.txt && \
    rm -rf /root/.cache/

COPY . /code

EXPOSE 8000

CMD ["python", "app.py"]
```

.dockerignore:

```
fly.toml
.git/
*.sqlite3
```

app.py (endpoints):

```
# app.py
from flask import Flask, render_template, jsonify, request
import time

app = Flask(__name__)

# Instead of a boolean, store the last time (epoch seconds) the button was
pressed
last_press_time = None
PRESS_DURATION = 3.0 # seconds

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/press', methods=['POST'])
def press():
    global last_press_time
    last_press_time = time.time()
    return jsonify({'status': 'ok', 'message': 'button marked pressed'})

@app.route('/status', methods=['GET'])
def status():
    """Return True if within PRESS_DURATION of last_press_time, else
    False."""
    global last_press_time
    if last_press_time is None:
        pressed = False
    else:
        elapsed = time.time() - last_press_time
        pressed = elapsed < PRESS_DURATION

    return jsonify({'pressed': pressed})

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000)
```

Fly.toml :

```
# fly.toml app configuration file generated for flaskeryys on 2025-04-30T12:36:30-04:00
#
# See https://fly.io/docs/reference/configuration/ for information about how to use this file.
#
```

```
app = 'flaskeryys'
primary_region = 'ewr'
console_command = '/code/manage.py shell'
```

```
[build]
```

```
[env]
  PORT = '8000'
```

```
[http_service]
  internal_port = 8000
  force_https = true
  auto_stop_machines = 'stop'
  auto_start_machines = true
  min_machines_running = 0
  processes = ['app']
```

```
[[vm]]
  memory = '1gb'
  cpu_kind = 'shared'
  cpus = 1
```

```
[[statics]]
  guest_path = '/code/static'
  url_prefix = '/static/'
```

Requirements.txt :

```
Flask==3.0.2
unicorn==21.2.0
requests==2.31.0
```

./templates/index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <title>Button Press Tracker</title>
```

```
<style>
  /* Full-screen flex-centered container */
  html, body {
    margin: 0;
    padding: 0;
    height: 100%;
  }
  .container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100%;
    background: #f0f0f0;
  }

  /* Big, glowing, green-gradient button */
  button {
    font-size: 2rem;
    font-weight: bold;
    padding: 1rem 2rem;
    color: white;
    background: linear-gradient(135deg, #2ecc71, #27ae60);
    border: none;
    border-radius: 12px;
    box-shadow:
      0 0 8px rgba(46, 204, 113, 0.6),
      0 0 16px rgba(39, 174, 96, 0.8),
      inset 0 -2px 0 rgba(0, 0, 0, 0.2);
    cursor: pointer;
    transition: transform 0.1s ease, box-shadow 0.2s ease;
  }
  button:hover {
    box-shadow:
      0 0 12px rgba(46, 204, 113, 0.8),
      0 0 24px rgba(39, 174, 96, 1),
      inset 0 -2px 0 rgba(0, 0, 0, 0.2);
    transform: translateY(-2px);
  }
  button:active {
    transform: translateY(0);
  }

```

```

        box-shadow:
            0 0 6px rgba(46, 204, 113, 0.6),
            inset 0 -2px 2px rgba(0, 0, 0, 0.3);
    }
</style>
</head>
<body>
    <div class="container">
        <button id="button" onclick="sendButtonPress()">Lets Campus School
Race</button>
    </div>

    <script>
        function sendButtonPress() {
            fetch('/press', { method: 'POST' })
                .then(r => r.json())
                .then(j => console.log(j.message))
                .catch(e => console.error(e));
        }
    </script>
</body>
</html>

```

End-to-End Flow

1. Developer runs `docker build -t flask-pico .` and `docker run -d -p 8000:8000 flask-pico`.
2. Flask serves `/` and JSON endpoints on host LAN at `http://<host-ip>:8000`.
3. Browser hits `/`, renders `index.html`, user clicks button → POST `/press`.
4. `app.py` stamps `last_press_time`.
5. Pico's CircuitPython script polls `/status`

Building & Pushing the Image

- `flyctl deploy` does a remote build (via BuildKit) or can stream your local Docker context up to Fly's build farm.
- It executes your `Dockerfile` steps, producing a tagged OCI image.
- That image is pushed into Fly's private registry.

Launch & Allocation of “Machines”

- Fly spins up one or more lightweight VMs (“machines”) in your chosen region (`ewr` by default).
- Each machine:
 - Pulls your image
 - Sets up a private WireGuard-based overlay network (so all your apps—across regions—can talk to each other securely)
 - Injects your `env` vars (like `PORT=8000`)
 - Exposes the container's `internal_port` to Fly's edge routing mesh

Edge Load Balancing, TLS, & Anycast

- Fly's global Anycast network advertises your app's IP from dozens of edge PoPs.
- Incoming HTTPS requests to `https://flaskeryys.fly.dev` terminate at the nearest edge.
- The edge proxy forwards them over the WireGuard mesh into one of your running machines on port 8000.

Health Checks & Auto-Healing

- By default Fly periodically probes your HTTP service. If your container crashes or fails a health check, the control plane will automatically:
 - Reboot the machine (cold restart)
 - Re-deploy the image if needed
- You can add custom `[[services]]` “checks” in `fly.toml` (e.g. hit `/status` every 5s) so unresponsive instances are recycled instantly.

Scaling Policies

- **Horizontal scaling:** you can configure `min_machines_running` (e.g. keep at least 2 replicas in production) or enable autoscaling based on CPU/RAM/requests.
- **Vertical scaling:** specify `[[vm]]` blocks (memory, CPU) so your container has the resources to handle bursts when the Pico polls every 100 ms.

Continuous Deployment

- Every `git push` → `flyctl deploy` triggers a new build & rollout:
 - Fly streams logs from all machines (`flyctl logs`) so you can debug if `/press` or `/status` misbehaves.
 - The rollout is incremental by default—old machines are drained of traffic before being shut down, ensuring zero downtime.

Pico W Data Fetch Loop

- Your CircuitPython script points to `https://flaskeryys.fly.dev/status`: