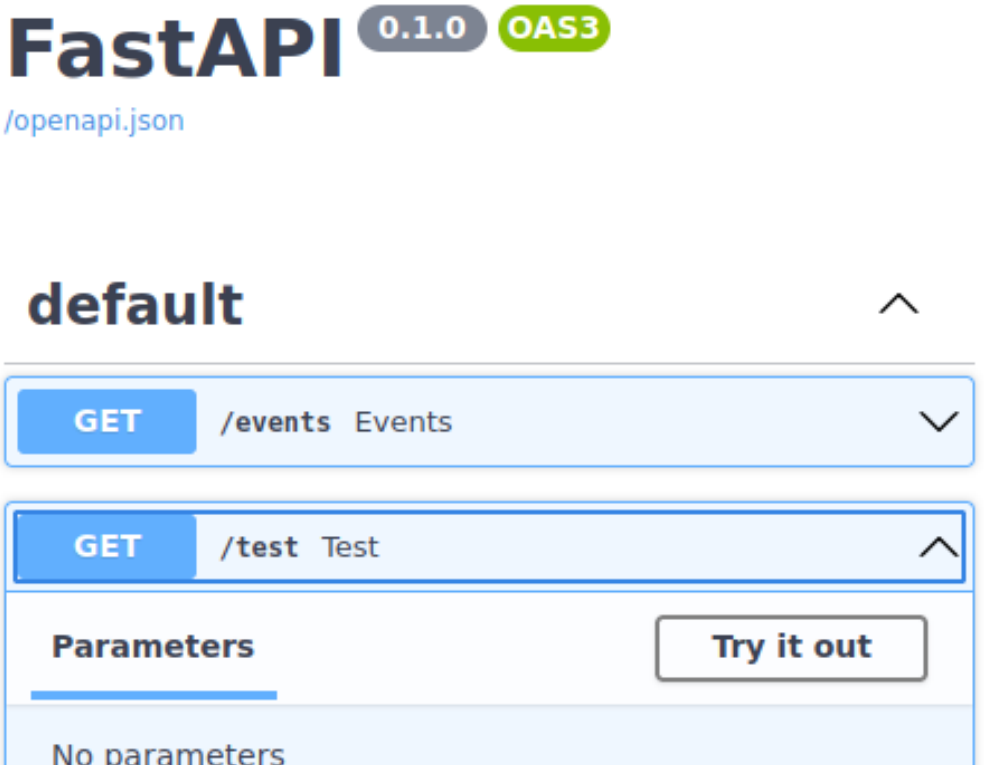
# Deploying a Python web app with docker/docker-compose and traefik+letsencrypt on a cloud VPS (*)

(*) not really production ready.. probably..

# Meet the “app”

https://pygraz.duckdns.org



# Meet the “app”

```python
from fastapi import FastAPI
import httpx

app = FastAPI(docs_url="/")


@app.get("/events")
async def events():
    async with httpx.AsyncClient() as client:
        r = await client.get(
            'https://api.meetup.com/PyGRAZ/events', timeout=10.0)
        return r.json()


@app.get("/test")
async def test():
    return {"name": "test"}

```

# Meet the “app”

- poetry init
- poetry add fastapi httpx[http2] uvicorn gunicorn

```
[tool.poetry]
name = "pygraz-server"
version = "0.1.0"
description = ""
authors = ["Christoph Reiter <reiter.christoph@gmail.com>"]

[tool.poetry.dependencies]
python = "^3.9"
fastapi = "^0.70.0"
httpx = {extras = ["http2"], version = "^0.20.0"}
uvicorn = "^0.15.0"
gunicorn = "^20.1.0"

[tool.poetry.dev-dependencies]
```

# Dockerize

```
1  FROM python:3.9-buster as base
2
3  RUN apt-get update && apt-get install -y \
4      python3 \
5      python3-pip \
6      python3-venv \
7      && rm -rf /var/lib/apt/lists/*
8
9  RUN python3 -m pip install "poetry==1.1.11"
10
11 COPY . /app
12 WORKDIR /app
13 RUN poetry install --no-dev
14
15 ENTRYPOINT ["poetry", "run", "gunicorn", "-k", \
16     "uvicorn.workers.UvicornWorker", \
17     "--bind", "0.0.0.0:80", "pygraz.main:app"]
18 EXPOSE 80
```

- docker build .
- docker run -p 8080:80 ec33ba2b264e
- http://localhost:8080

# Dockerize

- Use a good base image:
  - Updated frequently
  - Supported
  - From someone you trust

**Python WSGI HTTP Server.**
**Master process which manages workers**

```
ENTRYPOINT ["poetry", "run", "gunicorn", "-k", \
    "uvicorn.workers.UvicornWorker", \
    "--bind", "0.0.0.0:80", "py...az.main:app"]
EXPOSE 80
```

**Eventloop (uvloop)**

# Docker setup pitfalls

- docker by default doesn't log rotate (disk will get full). You need to select a different logging backend.
  - echo '{"log-driver": "local"}' > /etc/docker/daemon.json
- docker uses iptables, potentially overriding your rules: https://docs.docker.com/network/iptables/ (services that are exposed in docker-composer will be public)

# Now we have

- Python app
- Running in Docker
- Done?

**What handles the docker containers? Updating, restarting, dependencies, other containers like MariaDB etc..**

# docker-compose

```
1   version: '3.4'
2
3   services:
4
5     api:
6       restart: unless-stopped
7       build:
8         context: ./pygraz-server
9       environment:
10        WEB_CONCURRENCY: '4'
11      ports:
12        - "8080:80"
```

- docker-compose.yml
- “docker-compose up”

You can create a service from a Dockerfile, like here, or an image tag name, or a git repo with a Dockerfile in it for example.

4 worker processes

Port 80 in the container gets mapped to 8080 on the host

# Other commands

- docker-compose (ps|rm|build|logs|exec|run|start|stop|restart|….) <service>

- Similar to docker itself, but you can reference everything by service name and not by container ID etc..

# Now we have

- Python app
- Running in Docker + managed via docker-compose
- Done?

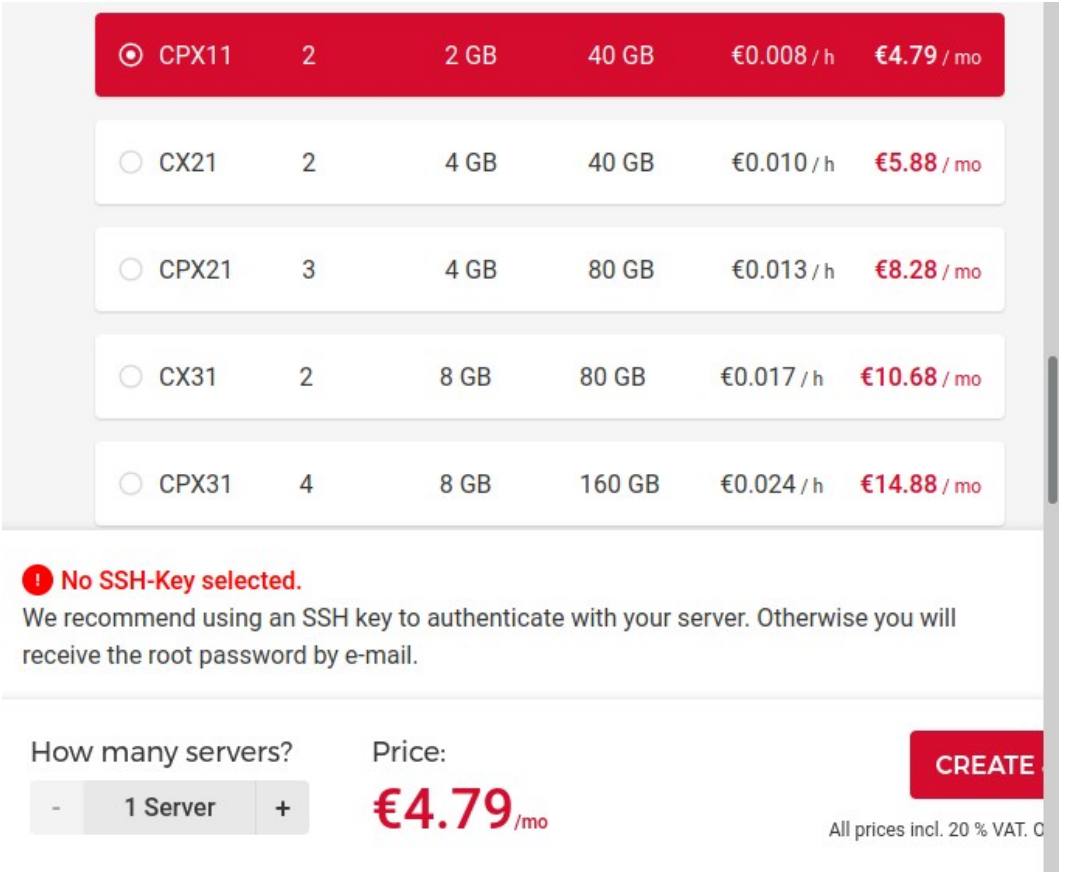
- **Letsencrypt + TLS, domain, put on server**

# Need a Domain

https://www.duckdns.org/


Like most DynDNS services blocked by facebook etc, so has its limits

# Need a VPS

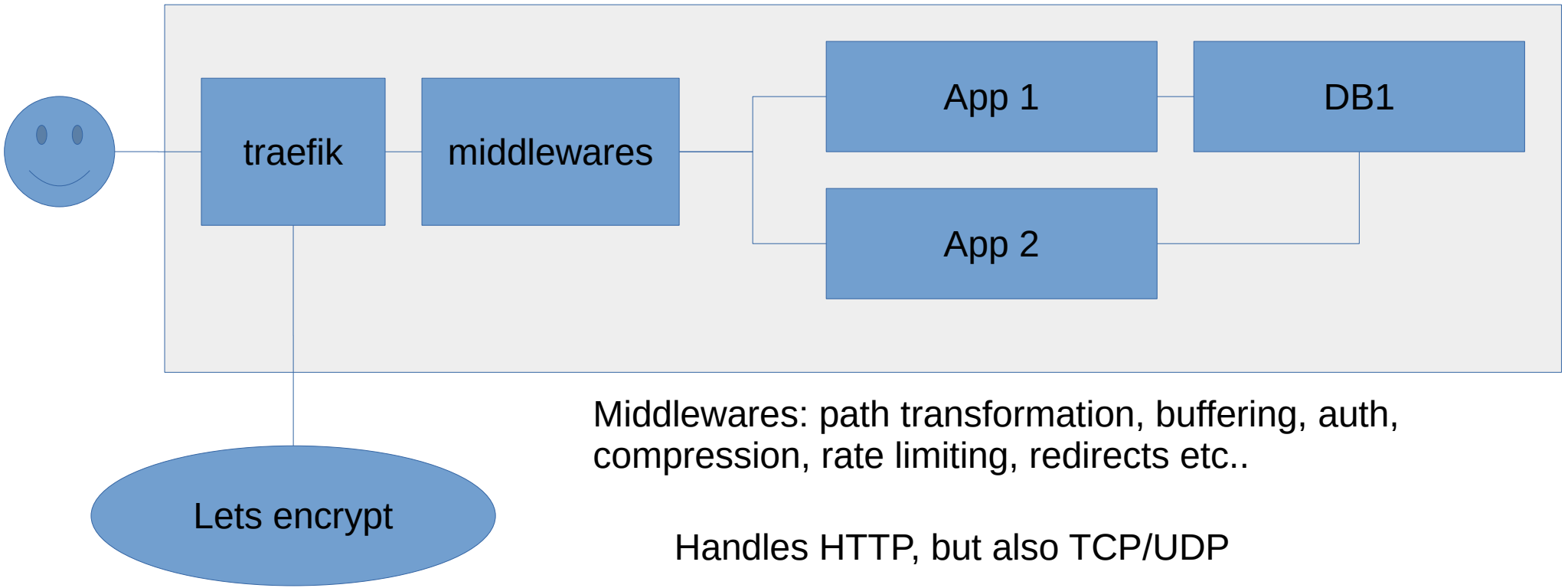



Billed hourly, so nice for testing here.

# Traefik (reverse proxy)

- https://traefik.io/

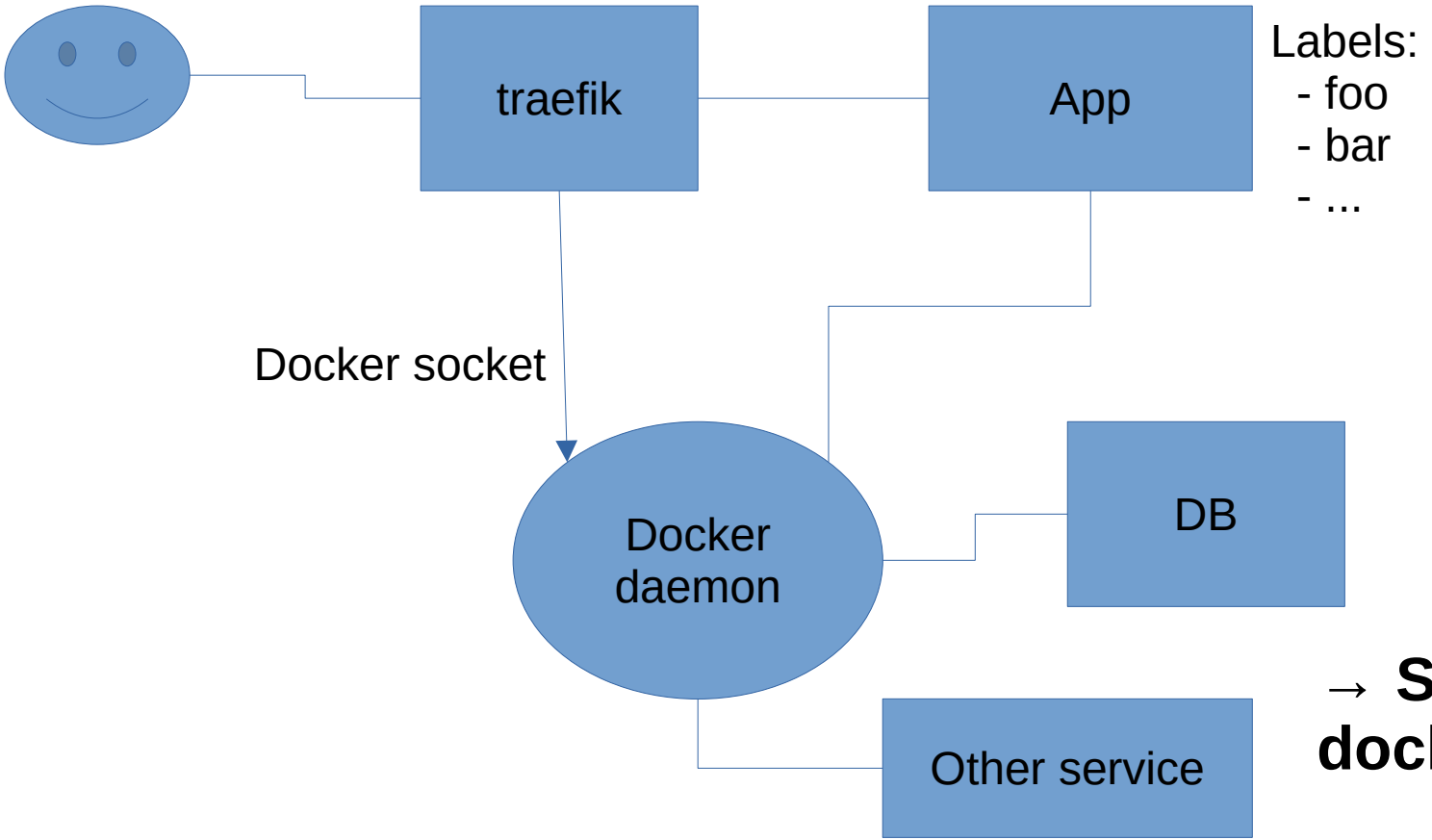
Has nice docker-compose integration




Middlewares: path transformation, buffering, auth, compression, rate limiting, redirects etc..

Handles HTTP, but also TCP/UDP

# Traefik + docker-compose




→ **SHOW docker-compose.yml**

# FORWARDED_ALLOW_IPS

- gunicorn: forwarded-allow-ips

- gunicorn: secure-scheme-headers
- Default: {'X-FORWARDED-PROTOCOL': 'ssl', 'X-FORWARDED-PROTO': 'https', 'X-FORWARDED-SSL': 'on'}

- Traefik manages them and is the only way we get a requests, so we can allow all IPs

# Security

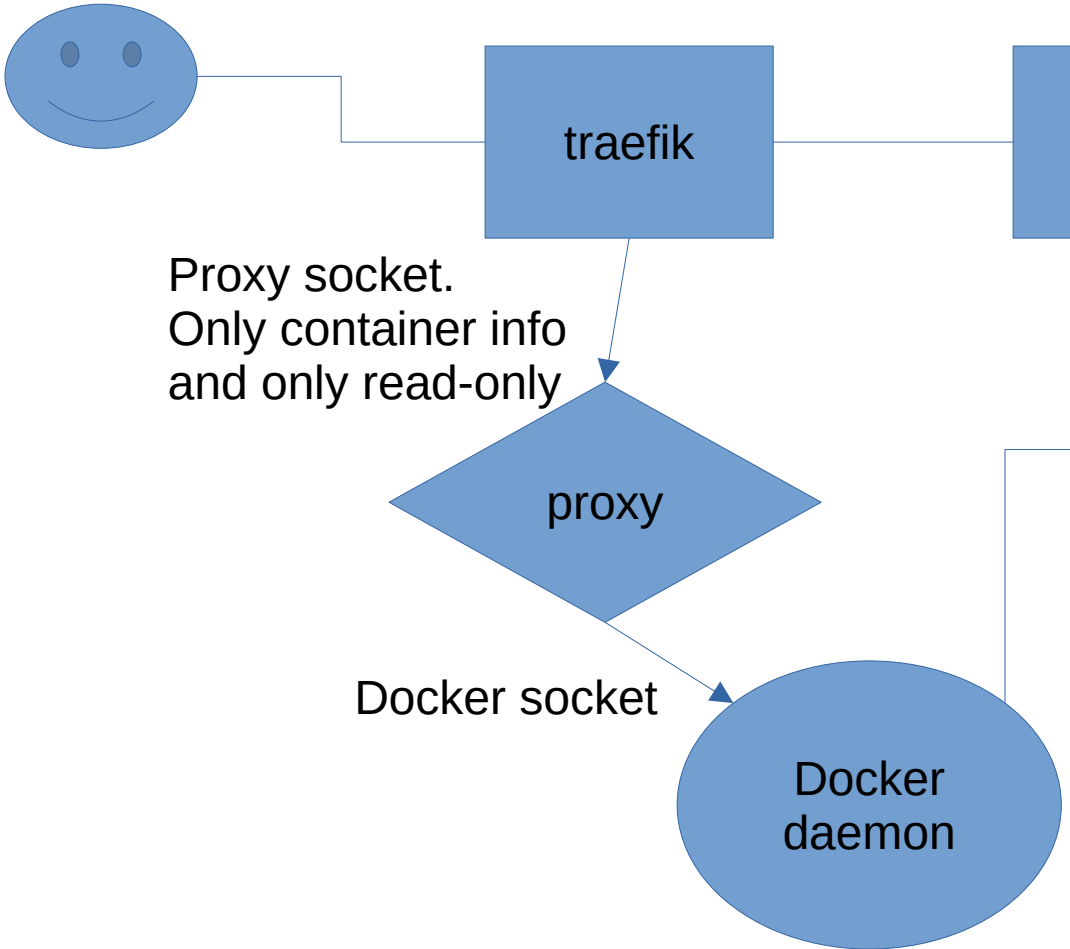
- docker+docker-compose+traefik: added complexity… more opportunities to make mistakes.

- On the other side, less fear of updating, since updating one image doesn't affect other parts of the system. Different Python/PostgreSQL etc versions. And you can easily roll back.

# Security

- Are your docker images up to date? Are the base images up to date? (in addition to your lock files)
- What is publicly accessible?
  - traefik is the only service, if it gets hacked then what?
  - has access to the docker daemon (means it can create/change docker images) you can use a docker proxy to work around that → tecnativa/docker-socket-proxy

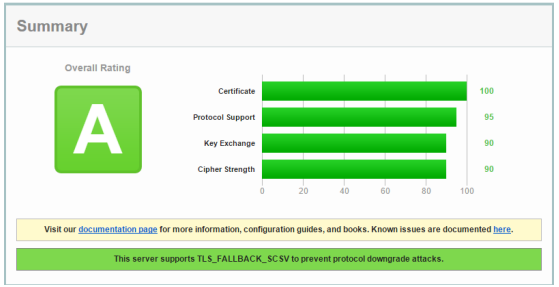
# tecnativa/docker-socket-proxy




**CONTAINER API, GET only**

→ **SHOW docker-compose.yml**

(*) but now you also have to trust tecnativa/docker-socket-proxy…




# TLS config

```
- "--certificatesResolvers.le.acme.keyType=EC256"
    - "--providers.file.directory=/configs/"
    - ./traefik/traefik-tls.yml:/configs/traefik-tls.
```

```
# https://ssl-config.mozilla.org/#server=traefik&v
tls:
  options:
    default:
      minVersion: VersionTLS12
      cipherSuites:
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305
        - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
```

- Defaults in traefik allow too much
- https://ssl-config.mozilla.org/
- Needs extra config file sadly, not via docker-compose → mount config file into container, tell traefik to look for it. Also use ECDSA cert for better compat (golang is missing cipher suites for RSA...)

ssllabs.com/ssltest

# “Deploy”

- ssh into server
- apt update && apt install git docker-compose
- git clone <repo> && cd <repoName>
- docker-compose up --detach
- …..
- profit

# Updating things

- “docker-compose pull” → pull new base images
- Since our app is build and not provided as an image we have to “docker-compose build api” also
- “docker-compose up” → rebuild+recreate+restart if something has changed

# Open Questions

- How to roll back?
- How to automate deployment/updates?
- DB migrations?
- Better logging?
- Monitoring?

# End

- Questions?

https://github.com/lazka/pygraz-traefik/