ML-service deployment & observability

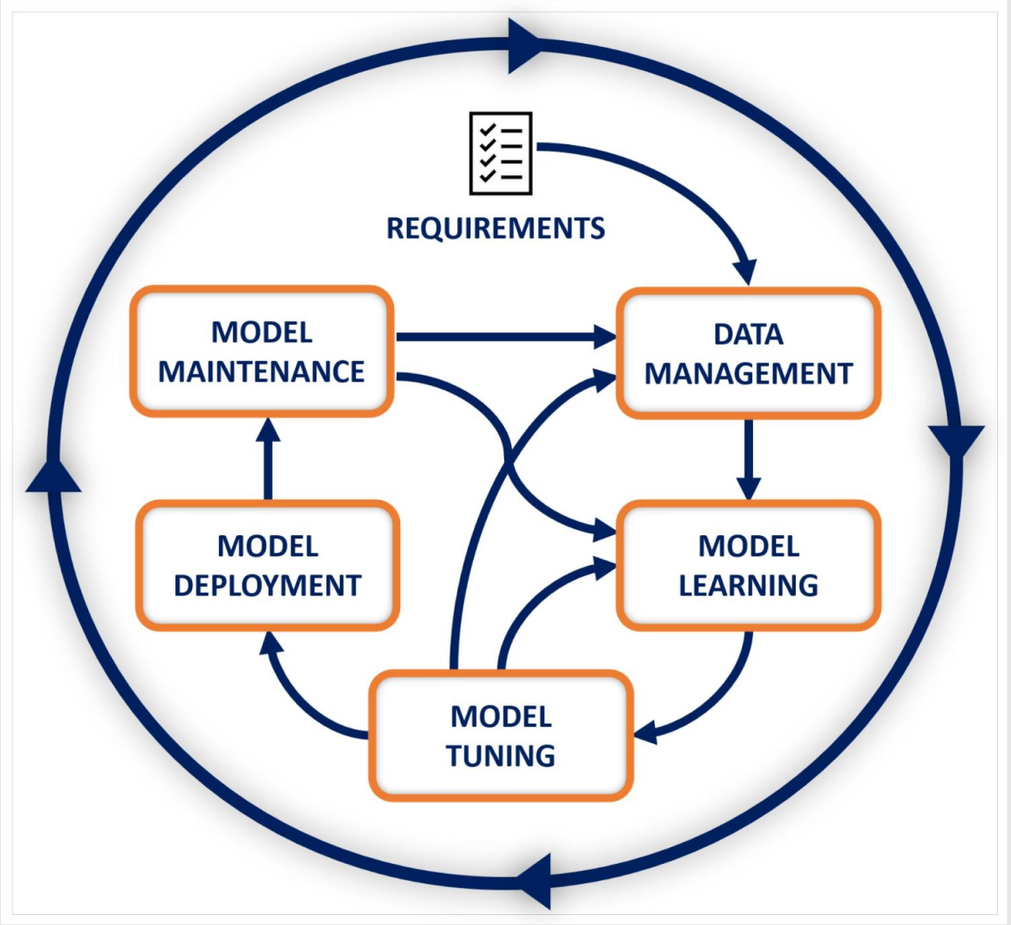
Why and how to move your model take out of local machine? machine?

Problem

- When your laptop goes to sleep, the service dies
- A single-use ngrok tunnel URL is not production-ready

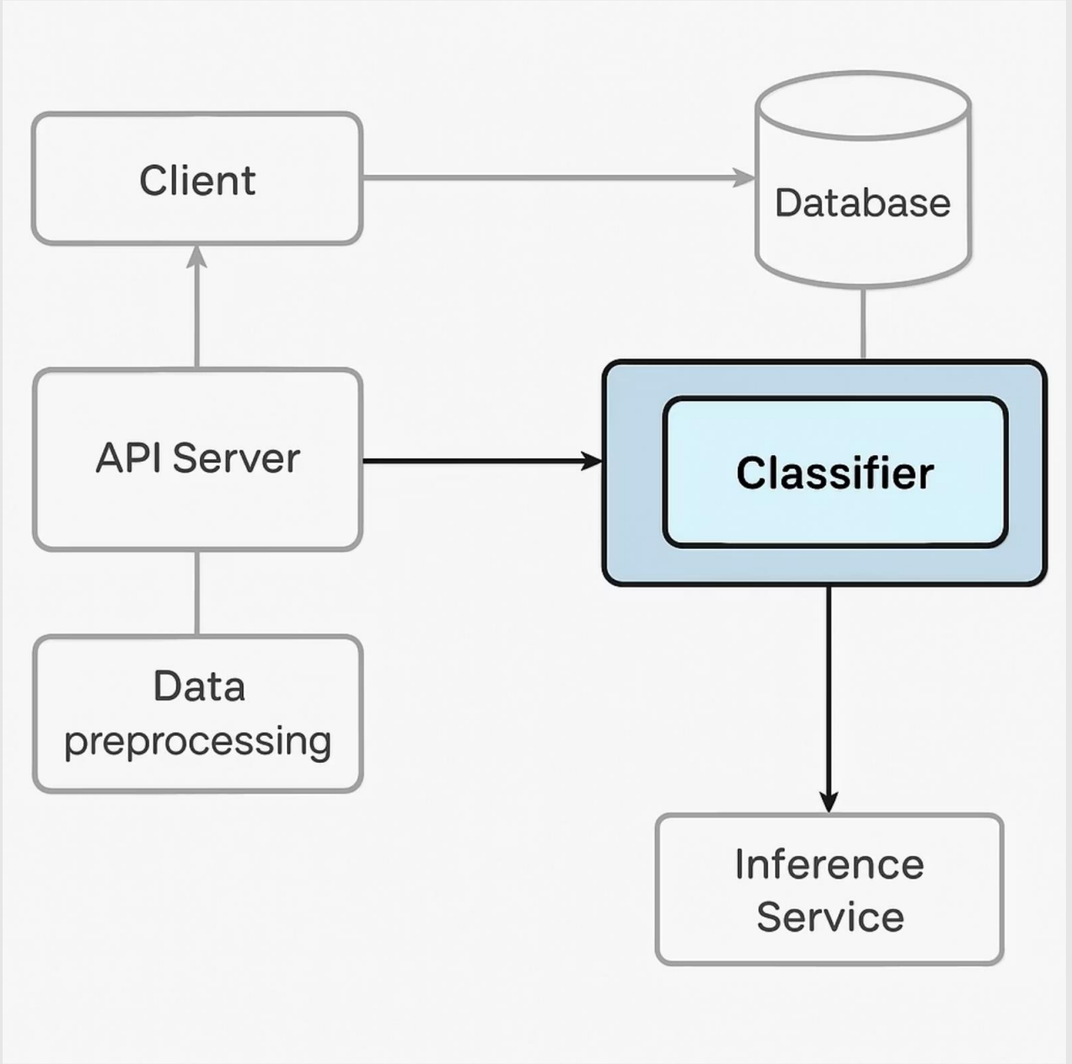
Goal of this session

- Launch model 24 × 7
- Get permanent IP/URL
- Verify it stays alive even after shutdown



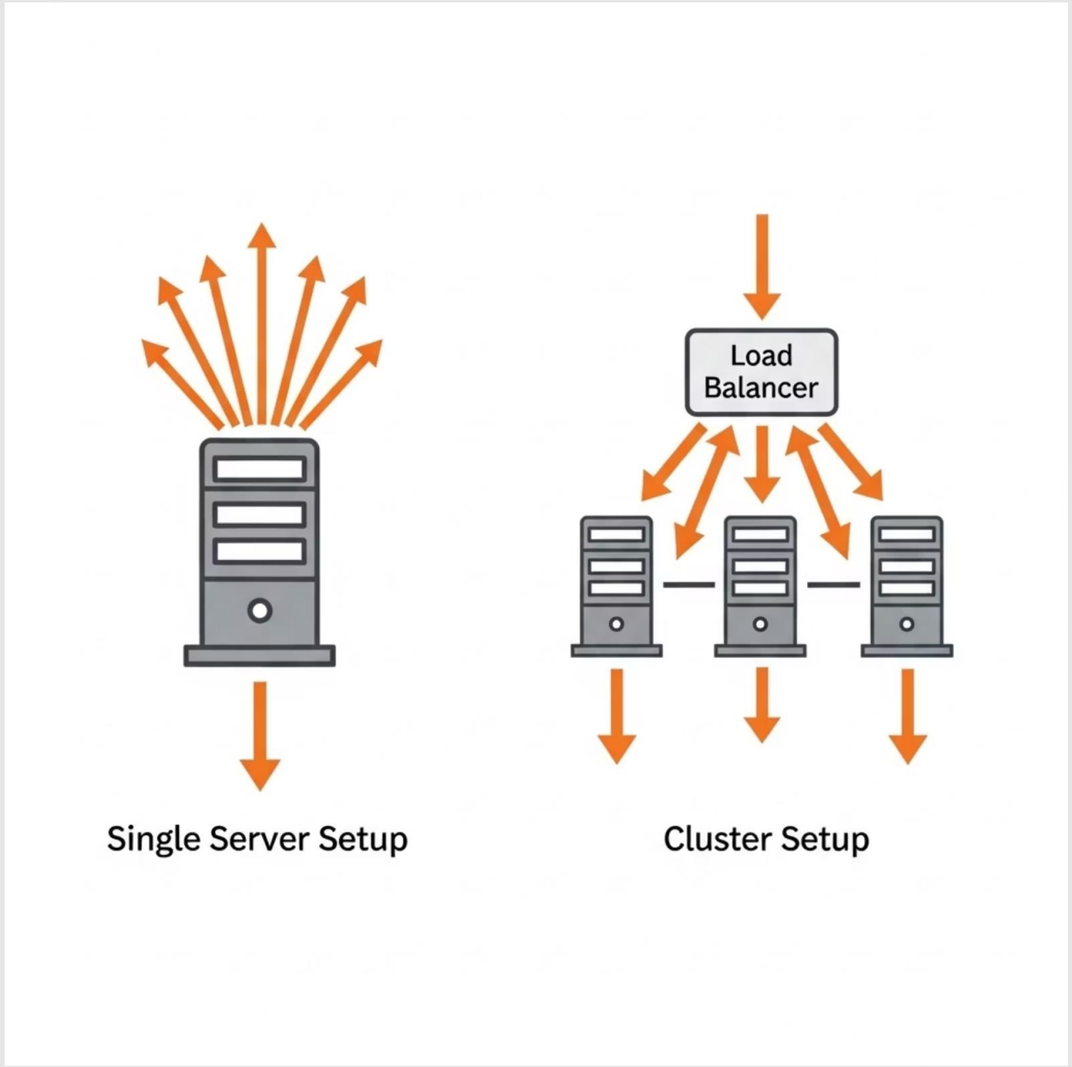
Our current setup: what remains after refactoring

- Previously had 4 services in docker-compose (MLflow, classifier, LLM, API)
- For our first production step, we only need the **classifier container**
- MLflow remains an internal repository; LLM and orchestrator can be added later as separate applications



Docker-compose ≠ Production

- 🔴 Single host → Single Point of Failure
- 🔴 No autoscaling or health-checks
- 🔴 Manual restarts after crashes/updates
- 🟢 Good for R&D and demonstrations

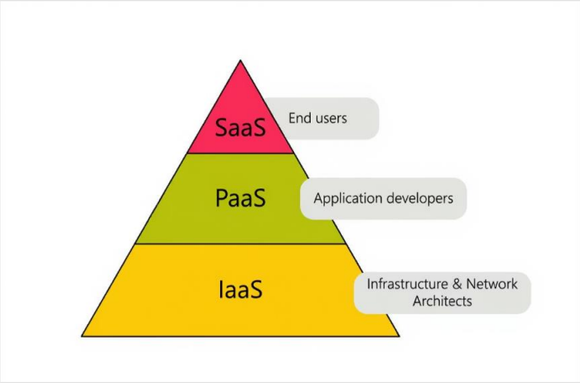


Four levels of "cloudiness" for ML services

Compute Models — Who is Responsible for What?

Understanding the division of labor between you and the cloud provider.

Level	Briefly	OS/Patches	Docker Config	Scaling
VM / IaaS	"Bare" VM (EC2, Yandex Compute)	**You**	**You**	**You**
CaaS	Container cluster w/o K8s management (AWS ECS, GCP Cloud Run)	Cloud Provider	**You**	Cloud Provider
PaaS	Full runtime, often buildpacks (Heroku, Render)	Cloud Provider	Auto-buildpacks	Cloud Provider
Edge Container	Lightweight VM near user (Fly.io, Cloudflare Workers+)	Cloud Provider	**You** (Dockerfile)	Cloud Provider



⚡ **Key takeaway:** The further right you go, the less DevOps burden, but also less flexibility.

Cloud Selection Checklist (3 questions = answer)

Navigate the options for ML service deployment with these key questions:

1 Do you need GPU/TPU acceleration?

Yes: Opt for **VM / IaaS** or **CaaS** offerings like AWS ECS GPU or Google Vertex AI.

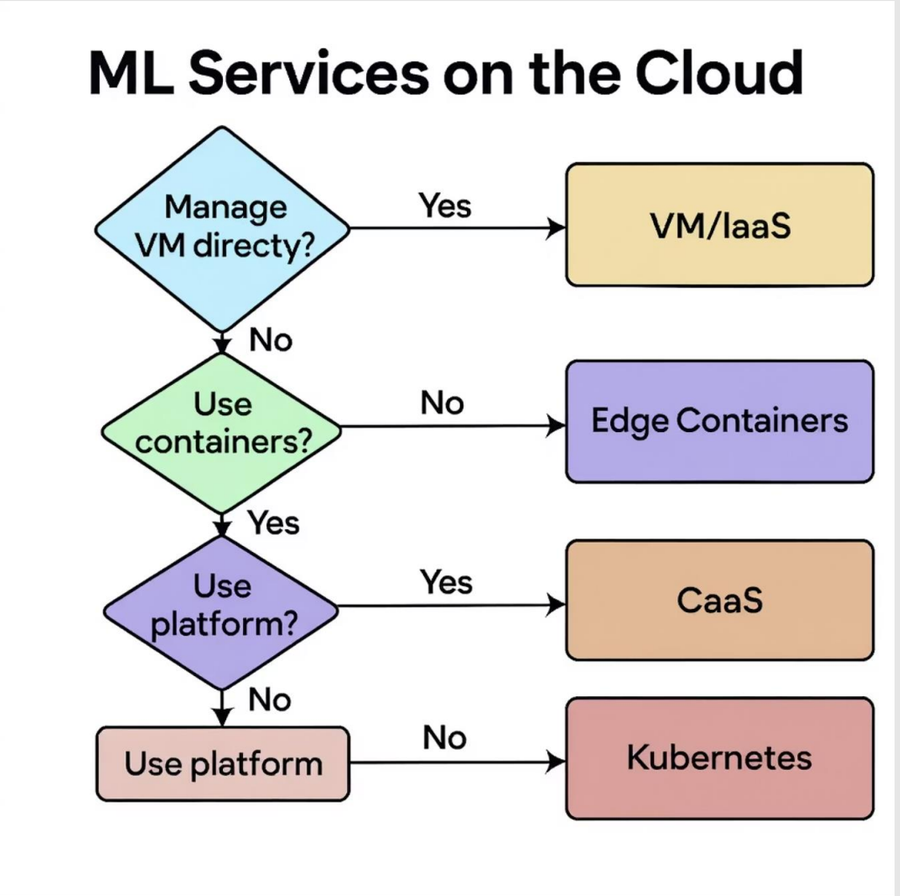
2 Is your budget currently $0?

Yes: Begin with **Edge Container** platforms such as Fly.io or Render's Free tier.

3 Do you require >1,000 RPS and multi-region deployment?

Yes: Consider advanced **CaaS** solutions like Google Cloud Run or a full Kubernetes Kubernetes setup.

Tip: Start simple with a free Edge Container, then migrate to more robust robust solutions as your load and requirements grow.



Cloud options for ML-services — quick reference

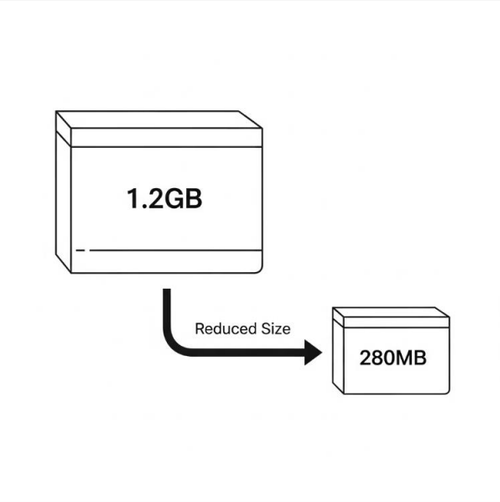
Quick Reference Table

IaaS	AWS EC2, YC VM	~45 sec	❌ Manual	❌	–	GPU inference, Root access
CaaS	AWS ECS Fargate, GCP Cloud Run	300-800 ms	✅	✅	180k Gi-ms / month (Cloud Run)	Burst-traffic API
PaaS	Heroku Eco, Render	5-15 sec	✅	⚠️ (sleep)	Render Free dyno	MVP with minimal DevOps
Edge	**Fly.io**, Cloudflare Workers, Vercel Edge	< 1 sec	✅	✅	Fly ≈ 3 GB-hr, 256 MB RAM	Latency-sensitive ML API

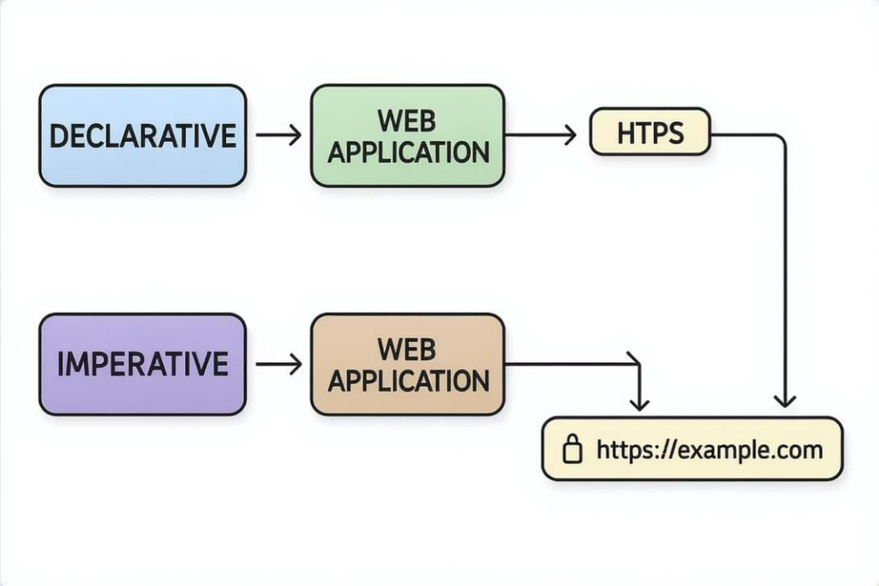
Dockerfile: fast, lightweight, reproducible

Checklist

- Multistage build → *size < 300 MB*
- python:3.11-slim-bookworm
- .dockerignore (__pycache__, .git/*, *.ipynb)
- Non-root user app
- CMD ["uvicorn","app:app","--host","0.0.0.0","--port","8000","--workers","1"]




```
# --- build stage
FROM python:3.11-slim-bookworm AS builder
COPY pyproject.toml poetry.lock ./
RUN pip install --no-cache-dir poetry
&& \ poetry export -f requirements.txt --output reqs.txt --
without-hashes#

--- runtime stage
FROM python:3.11-slim-bookworm
WORKDIR /app
COPY --from=builder /reqs.txt ./reqs.txt
RUN pip install --no-cache-dir -r reqs.txt
COPY src/ ./src/
USER 1001
CMD ["uvicorn","src.main:app","--host","0.0.0.0","--port","8000"]
```



2 paths to a working URL

Option A (Declarative)

First, use **flyctl launch** to configure your application, followed by **fly deploy** to publish it and get your live URL.

Option B (Imperative)

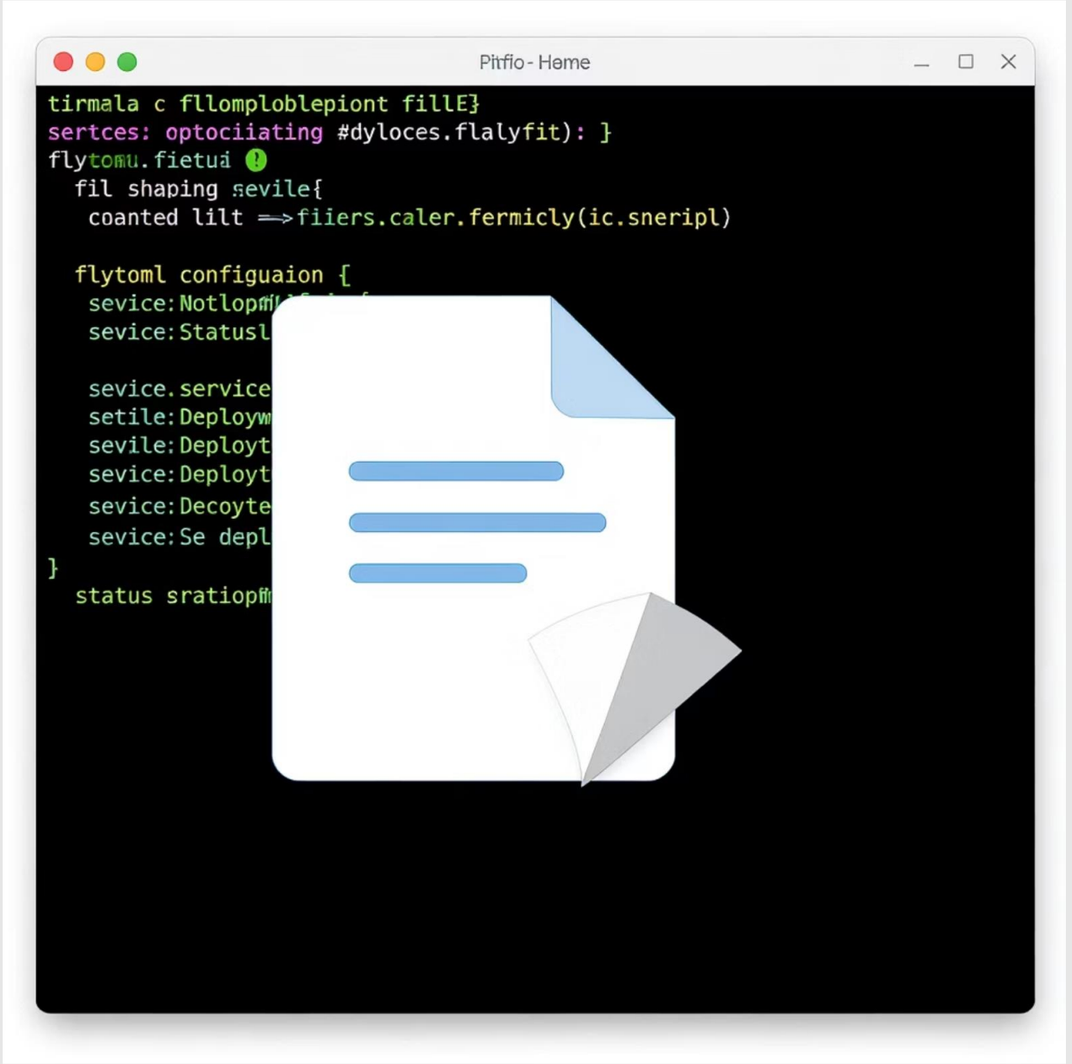
Begin by using **docker push** to upload your image, then simply use **fly machine run** to launch a virtual machine and obtain your URL.

⏱ We'll demonstrate both in ~8 min; starting with A, then B

Ė BŅÕŘ ŇÑŐÕÕŘ Ą ŅÕŘĖPÖÖ Õ

```
# 1. generate config and Anycast-IP

flyctl launch --name msg-cls --dockerfile Dockerfile --no-deploy

# 2. check/adjust fly.toml
# 3. deploy servicefly
deploy --remote-only
```

- Configuration stored in Git → review, rollback
- Fly creates/updates **Machines** automatically
- Rollbacks and release history out of the box



B. fly machine run — launch VM with one command

```
# with ready image in registry.fly.iofly auth dockerdocker push registry.fly.io/msg-cls:v1fly
machine run registry.fly.io/msg-cls:v1 \ -a msg-cls --region fra \ --port 8000 --autostop stop
--restart always \ -v data:/data --cpus 1 --memory 512
```

- Creates VM → mounts **data** volume
- Process starts in < 1 sec; URL provided immediately
- All config in command → suitable for ad-hoc jobs and migrations

Fly deploy or Fly machine run?

	fly deploy	fly machine run
Config in Git	✅	-
Speed "edit→URL"	medium (build)	**instant**
Scaling	scale count in TOML	manual machine clone
Use-case	API-service 24/7	batch job, PoC

- Need one stable API → **deploy**
- Need experiment "right now" → **machine run**
- Can combine: API via deploy, background tasks via machine run

Understanding fly.toml

```
app = "msg-cls"primary_region = "ams"[env]MODEL_NAME =
"msg_cls"MODEL_ALIAS = "champion"[[services]] internal_port = 8000
protocol = "tcp" [[services.ports]] handlers = ["http"] port = 80
[[services.ports]] handlers = ["tls", "http"] port = 443[checks]
[checks.alive] grace_period = "10s" path = "/alive" type = "http"
```

app	name
primary_region	WHERE to deploy
env	secrets/variables
services	routing
checks	health

/alive → health-check

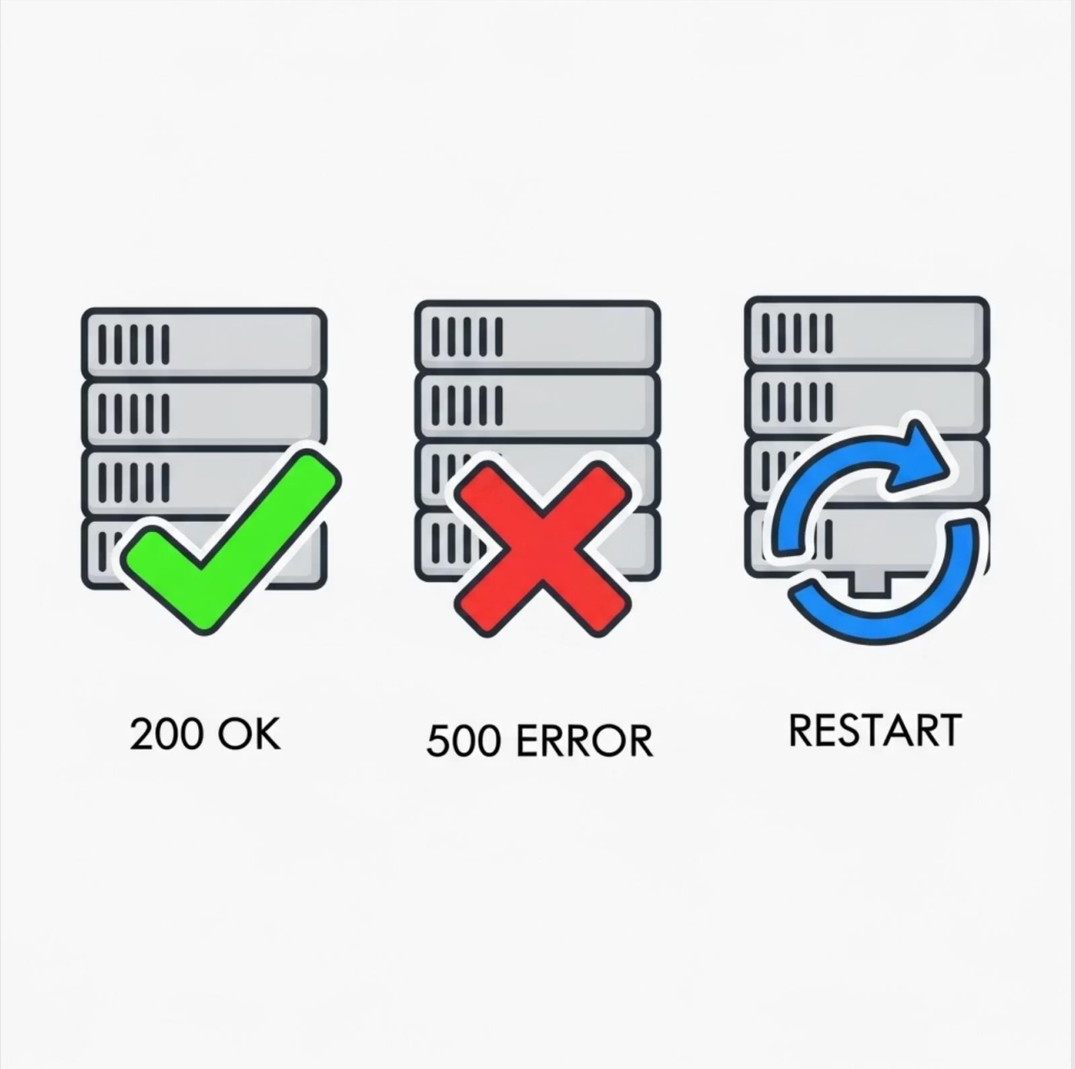
FastAPI

```
@app.get("/alive")def alive(): return {"status": "ok"}
```

fly.toml (fragment)

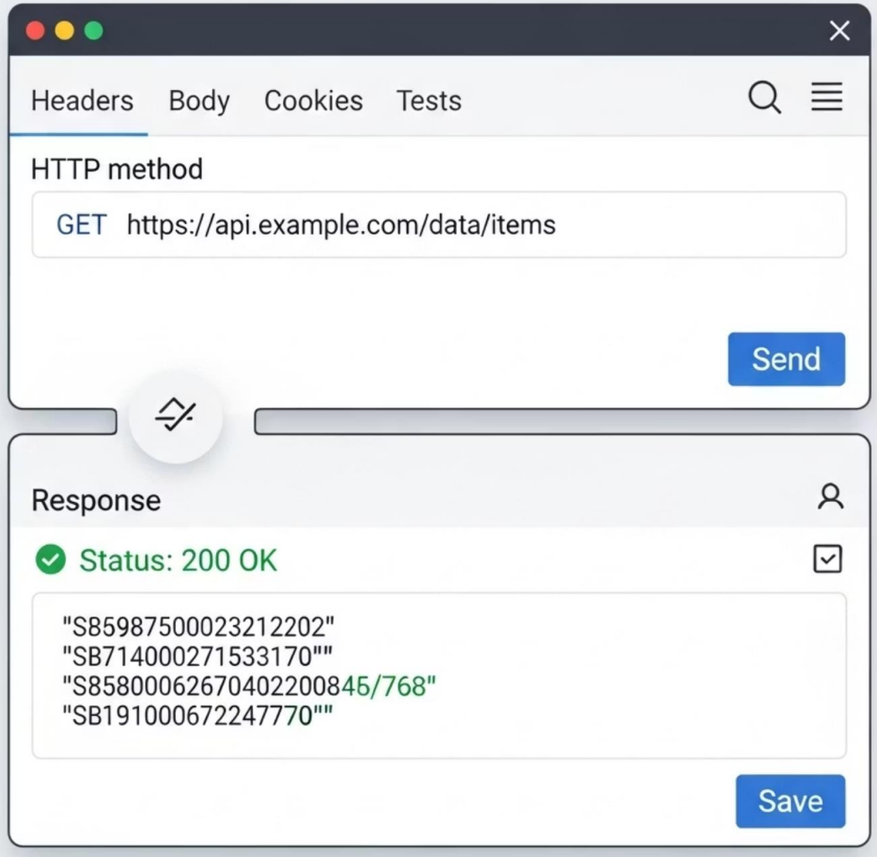
```
[checks] [checks.alive] path = "/alive" interval = "15s"
timeout = "5s"
```

🚑 2 consecutive errors → Fly restarts the VM.



Smoke-test /predict

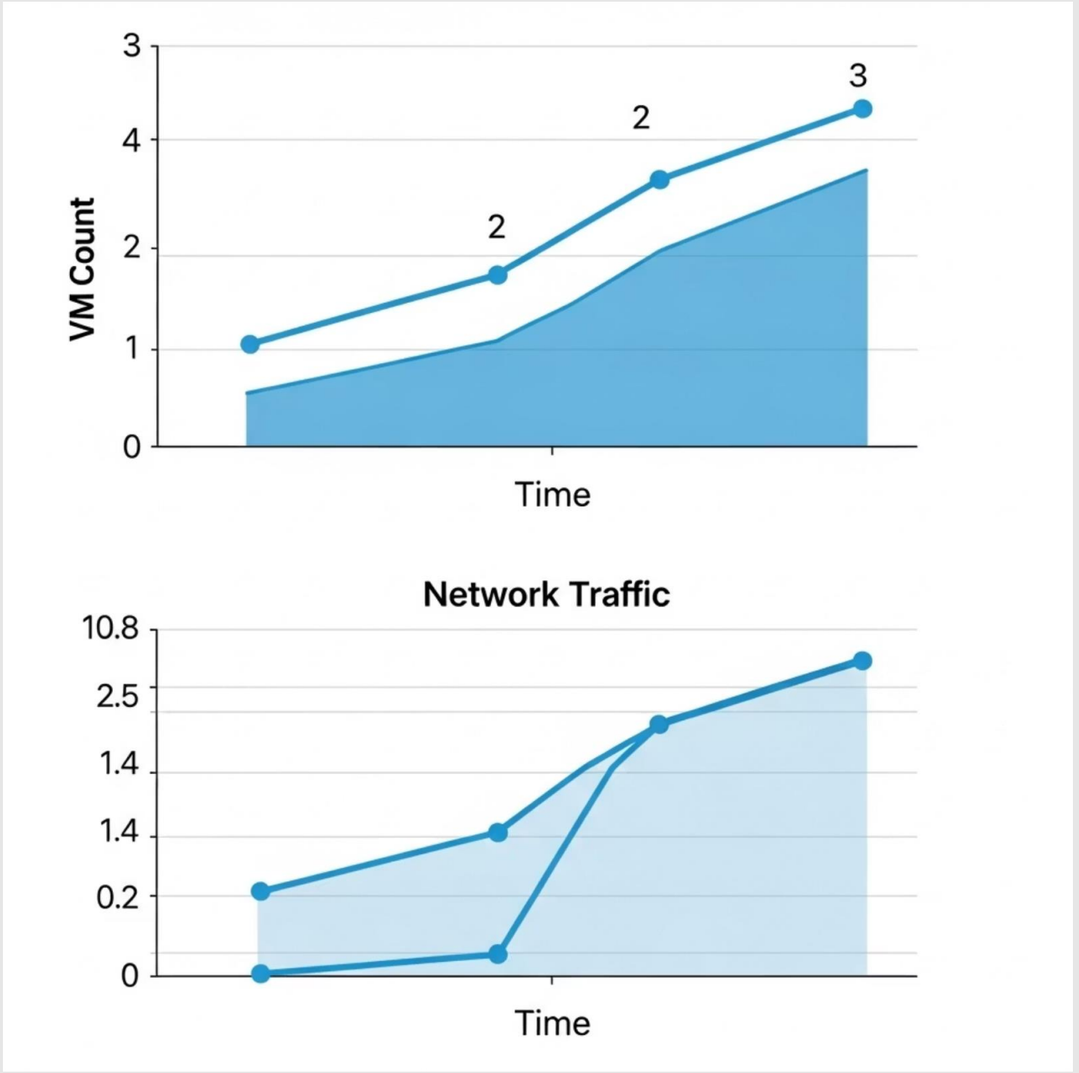
```
curl -X POST https://.fly.dev/predict \ -H "Content-Type:
application/json" \ -d '{"text": "Buy cheap
iPhone"}'Response: {"label": "spam", "proba": 0.97}
```



Response includes label + probability

Auto-scale & Auto-stop

- **Auto-start** — first request brings up VM in < 1 sec
- **Auto-stop** — idle > 15 min → VM = 0 (no credit usage)
- CPU-based scaling: fly scale count min=1 max=3
- Load analysis command: fly app status --all



💰 $0 when idle

Important Fly.io limitations

✕ One **volume** is tied to a region — don't use SQLite in multi-region

✕ RAM 256 MB free-tier → large BERT weights may not fit

✕ Cold-start if VM = 0 — ~800 ms

✓ Can set up a second app for LLM-assist

- LiteFS = distributed SQLite (beta)
- shared-cpu-1x → 256 MB

Other Cloud Platforms

How to choose

Í ÑÑŇ ĠĨ ¿ Ċİ Ĩ ¿ Ē	$0 Budget?	>1,000 RPS?
👉 CaaS / IaaS	👉 PaaS Free tiers	👉 CaaS autoscale

Google Cloud Run

Fully-managed CaaS

Ã 🚀 Ė Þ PŎŒŃMŎÑ ĆП Í ŎŃŎBŒŃMŎŃÁPŎÁŖŇǾ

Ã 🛡️ ĠŎŎNMŎĢ İ İ Ĩ Ĭ ŎÞ PÁŎŅÁPOŇÁNŎŔ

- 💰 Pay-per-CPU-ms billing

AWS ECS Fargate

Serverless containers

Ã ☁️ Í Ŏ CĚŃǾŅŇǾĽŐMPŃOŎŊ PŎ Ö MŌMŊÑ

- 🔗 Integrates with AWS ALB & IAM
- 🎯 Granular per-task pricing

CI/CD — concept, not configuration

- Pipeline stored alongside code → reproducibility
- Tests break the build — protect production
- Same template as for web applications — ML service = regular service

Why observability?

Problem → Goal

🔴 *"200 OK ≠ everything is fine"* – without metrics you won't notice that latency increased 10x overnight

4 Golden Signals

1. **Latency** – response time
2. **Traffic** – request volume
3. **Errors** – 4xx/5xx ratio
4. **Saturation** – resource load

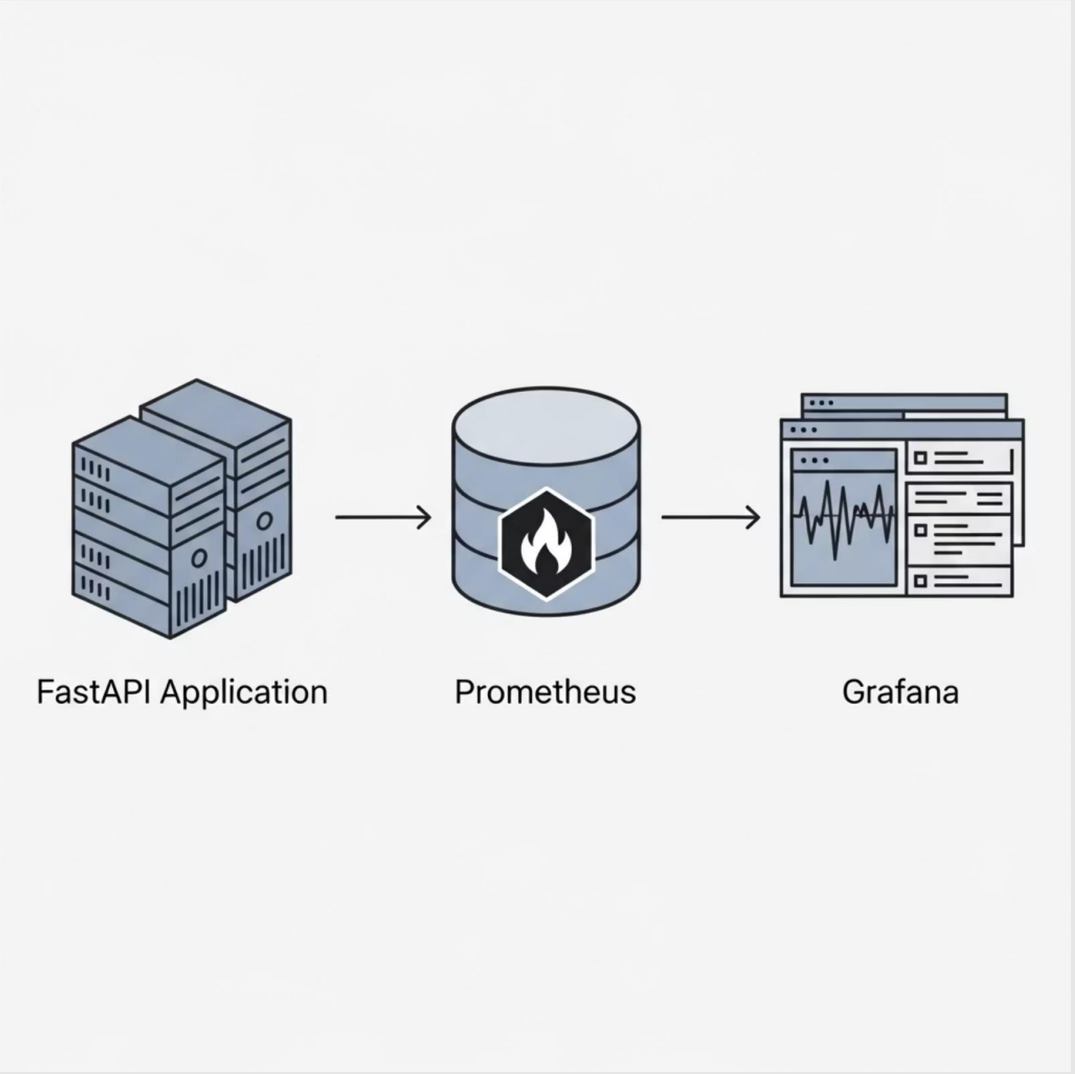
🟢 Goal: collect at minimum **Latency P95 and Error Rate** before your first your first production incident

Lightweight stack for practice: Prometheus + Grafana

docker-compose.yml (fragment)

```
version: "3.8»

services:
        Prometheus:
                image: prom/prometheus:v2.52
                volumes:
["./prometheus.yml:/etc/prometheus/prometheus.yml"]
                ports: ["9090:9090"]
        grafana:
                image: grafana/grafana-oss:10.4
                ports: ["3000:3000"]
```

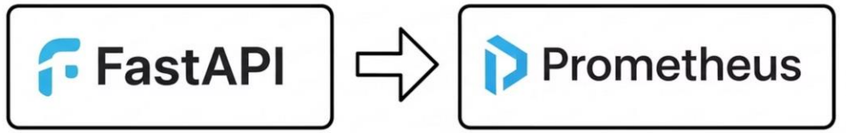
- Just two containers → run docker-compose up -d
- Prometheus scrapes **http://host.docker.internal:8000/metrics**
- Grafana admin / admin (change password!)



FastAPI → Prometheus

```
from prometheus_fastapi_instrumentator import Instrumentator

app = FastAPI()
Instrumentator().instrument(app).expose(app)
```

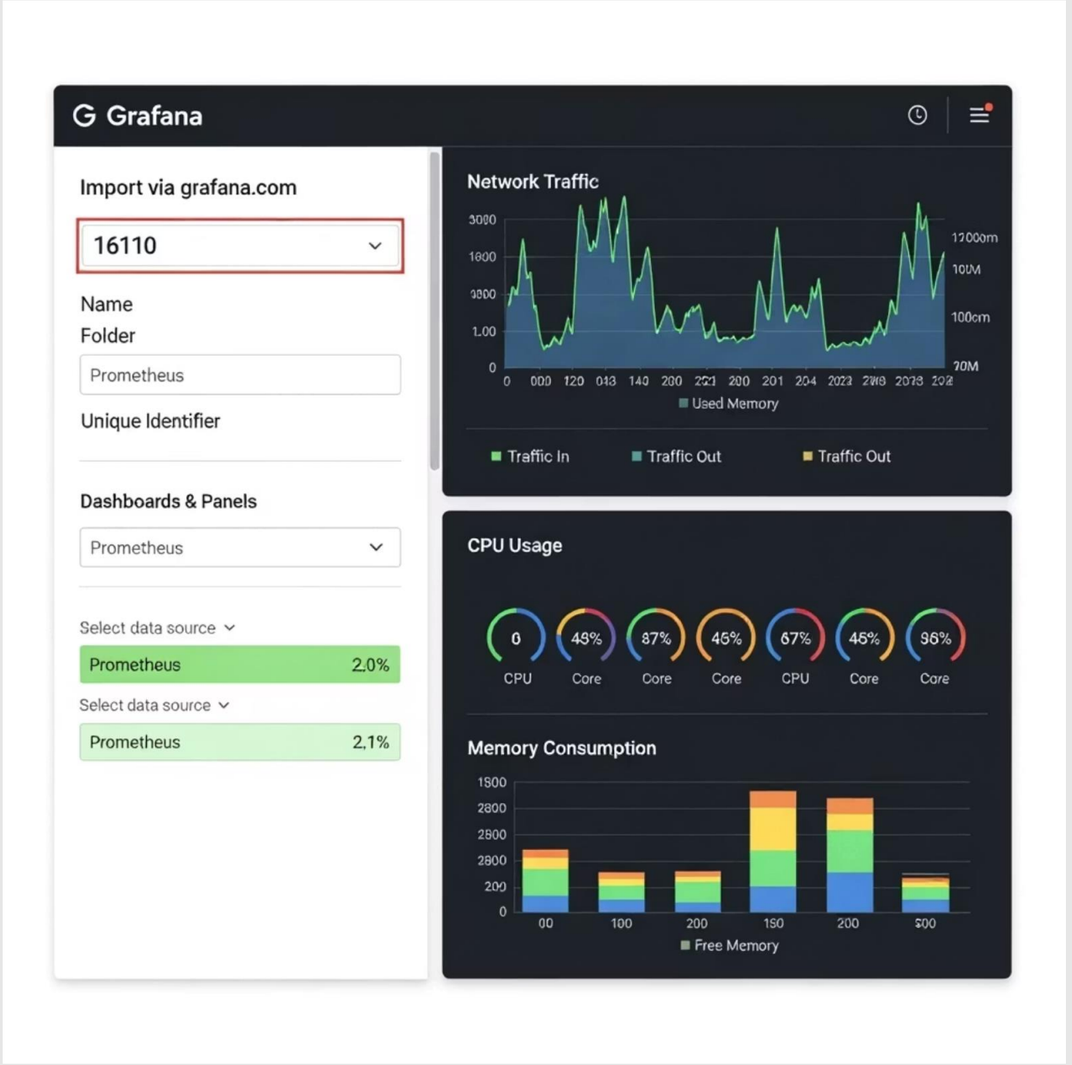
- pip install prometheus-fastapi-instrumentator
- Metrics available at /metrics
- Seamless integration — no manual middleware



Import ready-made dashboard into Grafana

1. Open Grafana → Dashboards → *Import*
2. ID **16110** (FastAPI Observability)
3. Datasource → *Prometheus*

Get panels: Latency P95, RPS, Error rate, CPU usage



What and how we log

Category	Example field	PII?	Retention
Access logs	method, status, latency	no	30 days
Business	user_id (hash), amount	partially	90 days
ML payload	text (anonymized), proba	may be	14 days
Exceptions	stacktrace	no	90 days

⚠️ Any unfiltered user text is potentially PII! We must filter it.

Online ML metrics: quality matters too

- Online Precision/Recall = calculated when delayed labels are available (example: anti-fraud)
- Data drift – feature averages, PSI
- Code example:

```
from prometheus_client import Counter, Summary

pred_counter = Counter().labels(model_version="v1")
pred_latency = Summary().labels(version="v1")
```

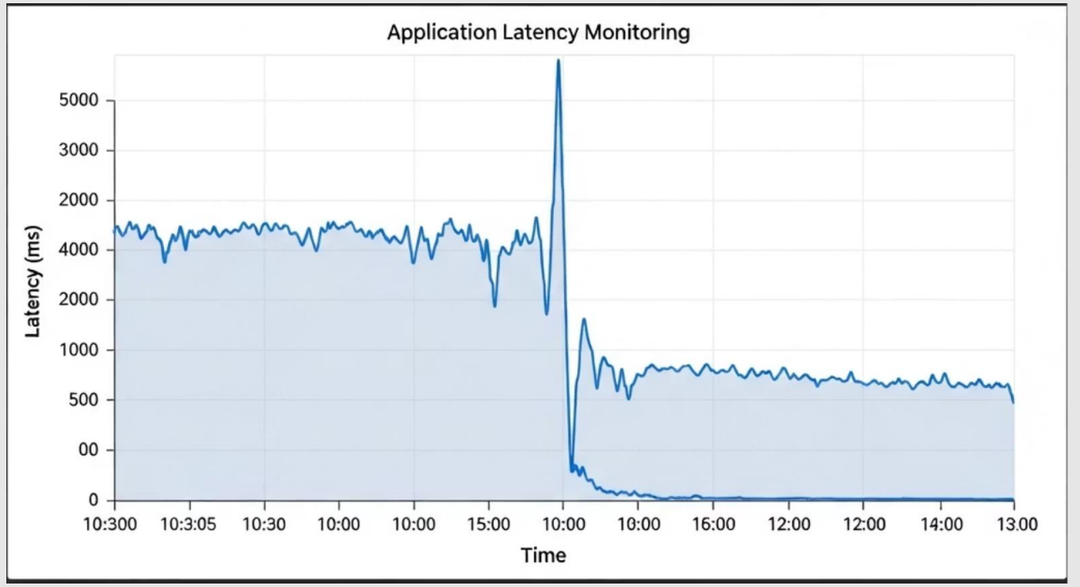
→ add to the same /metrics, build on the same dashboard

Demo — monitoring

- pip install prometheus-fastapi-instrumentator
- Add 3 lines to main.py:

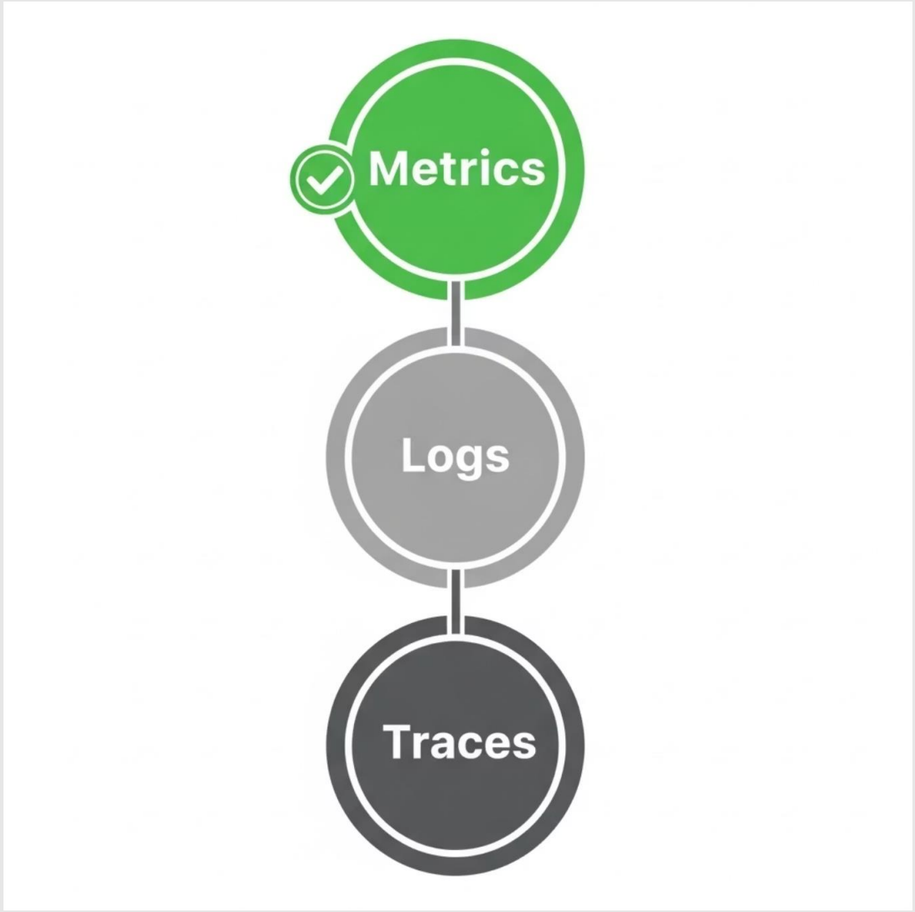
```
from prometheus_fastapi_instrumentator
import Instrumentator

Instrumentator().instrument(app).expose(app)
```

- Re-fly deploy
- Submit: Latency P95 graph screenshot from Grafana **or** Prometheus JSON dump from browser



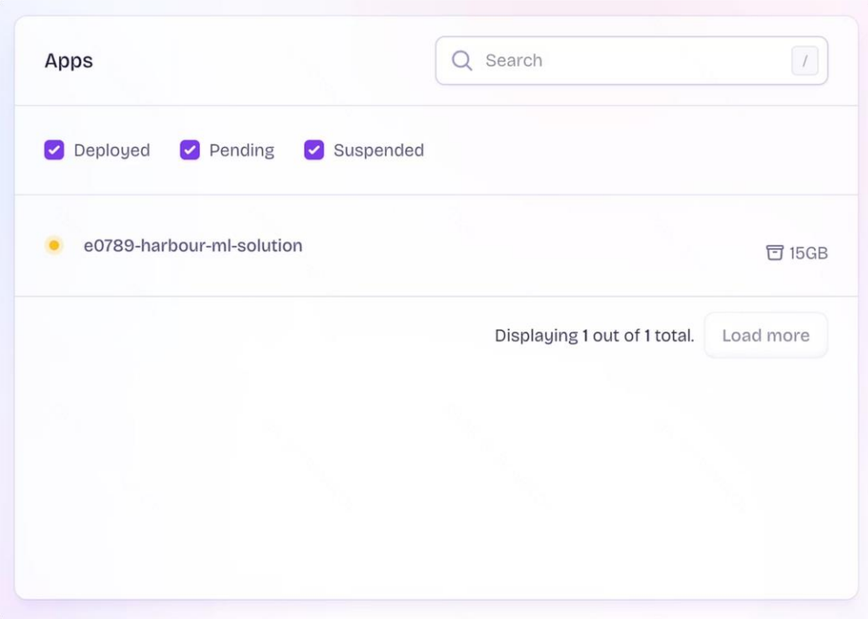
Observability summary

- ✅ 4 Golden Signals
- ✅ Prometheus + Grafana in 5 minutes
- ✅ /metrics integration in 3 lines
- ✅ Business metrics & ML metrics can be displayed side by side side



Demo — deploy classifier

1. flyctl launch --copy-config — create app
2. fly deploy — deploy it
3. Check /alive and /predict → 200 OK



Common deployment error checklist

Symptom	Cause	Fix
502 Bad Gateway	mixed up internal_port and CMD	set both to 8000
❌ No machines available	VM = 0 + cold-start disabled	fly scale count 1
OOM kill	BERT-model > 400 MB RAM	fly scale memory 512 or use Distil-BERT

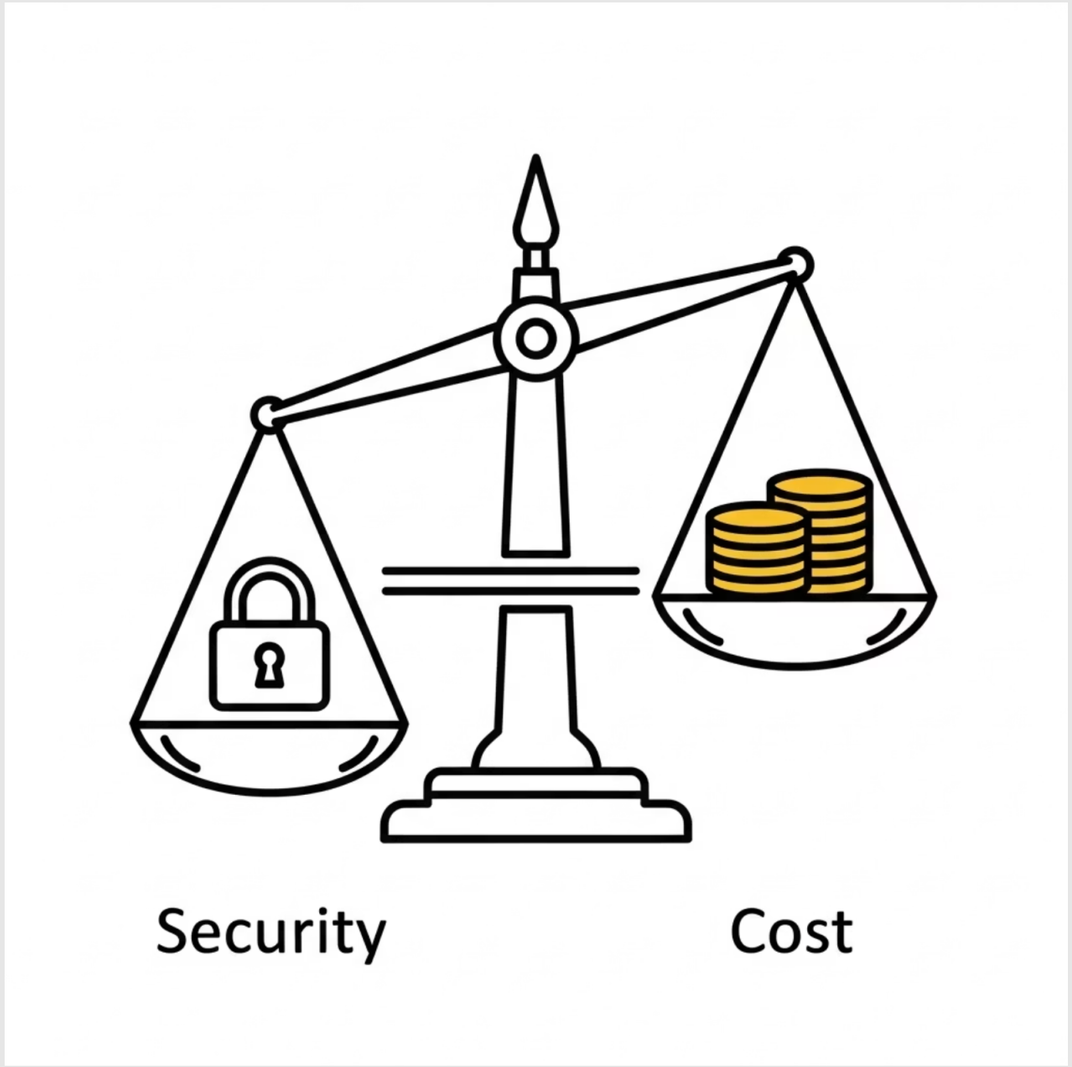
Security & Cost hints

🔒 Secrets:

- flyctl secrets set OPENAI_KEY=… — don't store in git!
- --region ams → local laws

💸 Cost savings:

- fly scale count 0 – stop overnight
- fly scale vm shared-cpu-1x --memory 512 – minimal resources
- Autostop > 15 min idle — free mode



What's next?

1

Blue-Green / Canary

fly deploy --strategy rolling

2

Traces

OpenTelemetry SDK + Grafana Tempo

3

Complete CI/CD

GitHub Actions, push to main → deploy prod

4

A/B online-evaluation

Quality metrics in Prometheus

Summary

Docker image → cloud (Fly & alternatives)

/health, /predict, Anycast URL

Prometheus /metrics + Grafana dashboard

Error checklists, secrets, cost optimization

Assignment

1. Publish your model in fly.io
    - /health and /predict return 200
    - register a public address on youare.bot