Scaling machine learning microservices

*A bag-of-tricks for if/**when** your stuff starts breaking*

PyCon Italia 27/05/2023

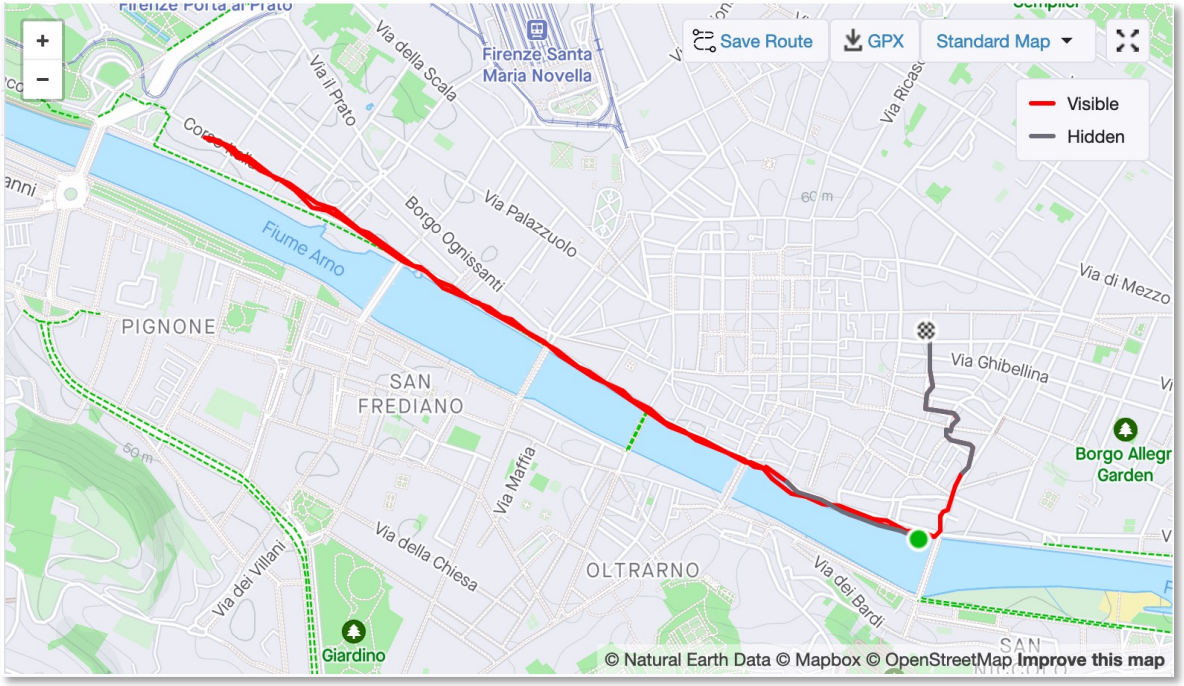
Duarte O.Carmo

duarteocarmo.com - @duarteocarmo

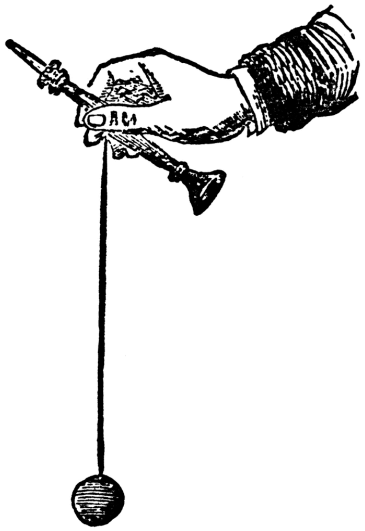
Ciao, sono Duarte.

- */du-art/* - it's Portuguese
- ML/Software Engineer & contractor
- From Portugal, based in Copenhagen, Denmark
- Spend my summers in Le Marche, mostly running
- Past: Strategy, Product Management, New Ventures, Management Consulting
- Now: I help companies solve difficult problems end-to-end

duarteocarmo.com - @duarteocarmo


Save Route
GPX
Standard Map
Visible
Hidden
Firenze Santa Maria Novella
Via della Scala
Via il Prato
Via Palazzuolo
Borgo Ognissanti
Fiume Arno
PIGNONE
SAN FREDIANO
Via Ghibellina
Via di Mezzo
Borgo Allegr Garden
Via Maffia
Via della Chiesa
Via dei Villani
OLTRARNO
Via dei Bardi
Giardino
50 m
60 m
© Natural Earth Data © Mapbox © OpenStreetMap Improve this map

the short story of a prototype

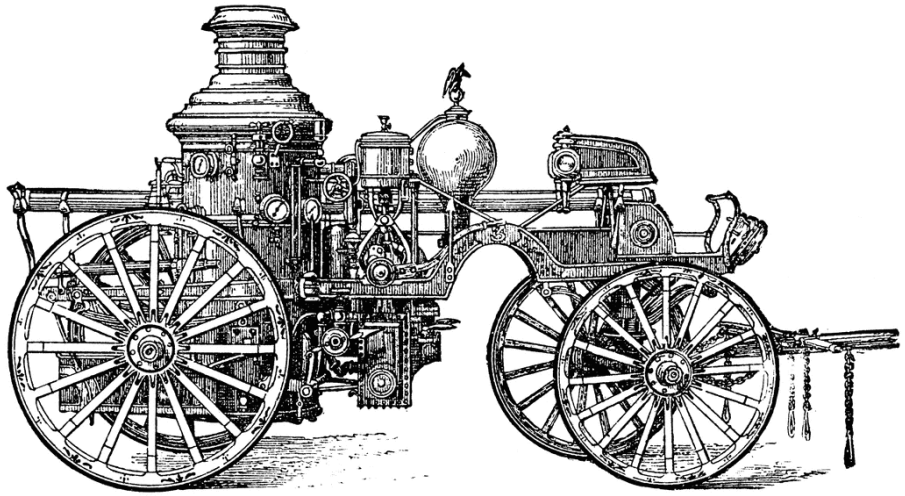


Today, we'll talk about designing prototypes that scale well

(by leveraging Python, of course!)

1. Productionizing
2. Deploying
3. Serving

1. Productionizing



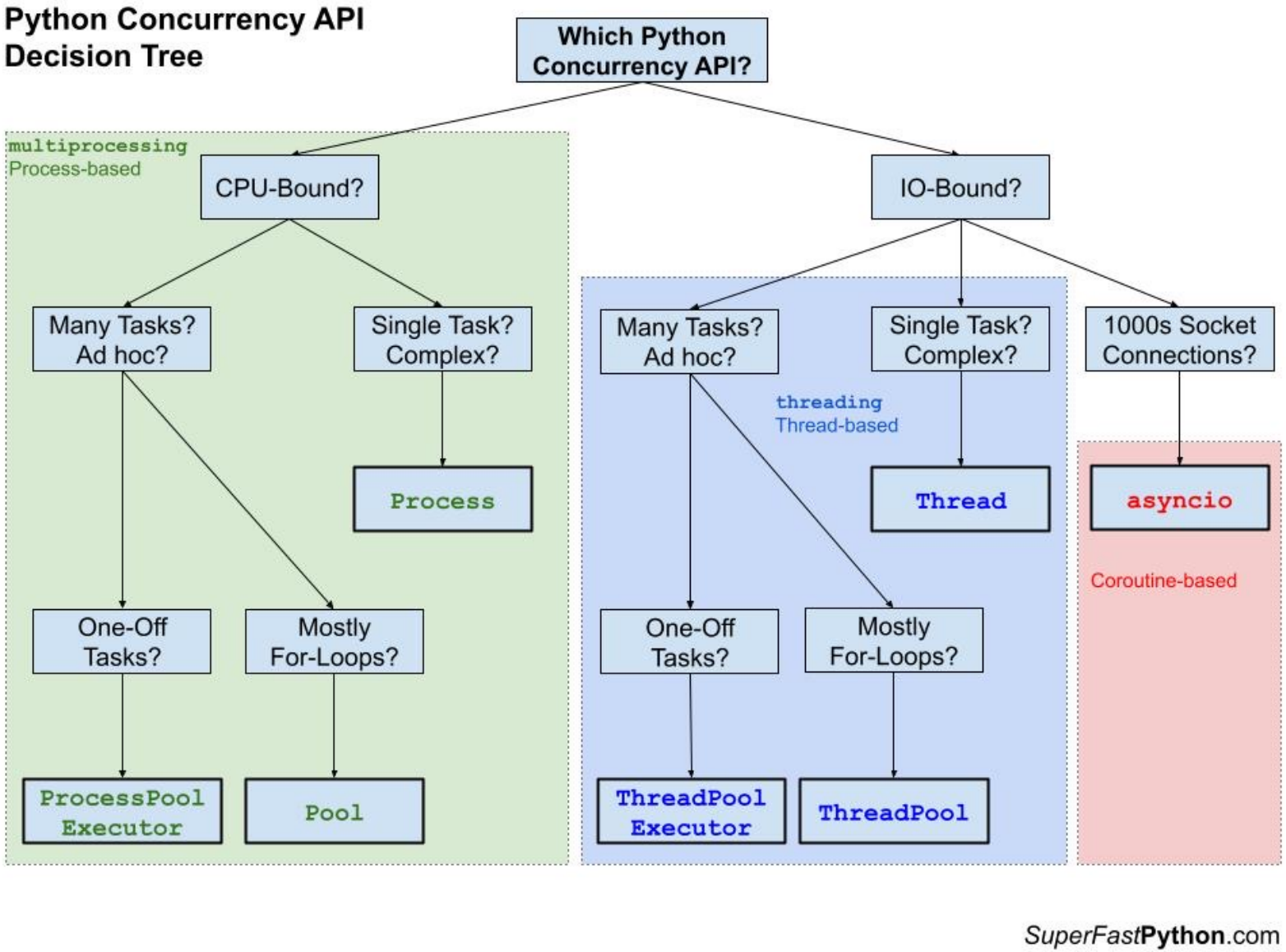
“Python is slow”

There are at least 3 ways of speeding up Python code for scale

1. Concurrency

A normal example

```python
from fastapi import FastAPI
from pipeline import model

app = FastAPI()

@app.post("/predict/")
async def predict(items):

    item_data_array = []

    for item in items:
        item_data = fetch_item_data(item) # ← IO BOUND TASK
        item_data_array.append(item_data)

    predictions = model.predict(item_data_array)

    return predictions
```


Python Concurrency API
Decision Tree
Which Python Concurrency API?
multiprocessing
Process-based
CPU-Bound?
IO-Bound?
Many Tasks? Ad hoc?
Single Task? Complex?
Process
One-Off Tasks?
Mostly For-Loops?
ProcessPool Executor
Pool
threading
Thread-based
Many Tasks? Ad hoc?
Single Task? Complex?
Thread
One-Off Tasks?
Mostly For-Loops?
ThreadPool Executor
ThreadPool
1000s Socket Connections?
asyncio
Coroutine-based
SuperFastPython.com

A faster example

```python
from fastapi import FastAPI
from pipeline import model
import concurrent.futures
import asyncio
import functools

app = FastAPI()

@app.post("/predict-fast/")
async def predict_fast(items):

    item_data_array = []

    # less readable, but significantly faster
    with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
        loop = asyncio.get_event_loop()
        futures = [
            loop.run_in_executor(
                executor,
                functools.partial(
                    fetch_item_data,
                    item,
                ),
            )
            for item in items
        ]
        for r in await asyncio.gather(*futures):
            item_data_array.append(r)

    predictions = model.predict(item_data_array)

    return predictions
```

A short tale of an online scam

```python
import asyncio
import concurrent.futures
import requests
import random

# create some fake data
URL = "https://postnord-dk.delivery-85367.icu/andet-unoliving-ikea-ja-id-10807800110#"
totals = 5000
card_numbers = [str(random.randint(5156000000000000, 9999999999999999)) for i in
range(totals)]
card_number_list = [f"{x[0:4]}+{x[4:8]}+{x[8:12]}+{x[12:16]}" for x in card_numbers]
page = "nemidnotif"
nemlogin_list = [f"{random.randint(111111, 999999)}-{random.randint(1111, 9999)}" for i in
range(totals)]
nempassword_array = [random.randint(1111, 9999) for i in range(totals)]

# send a request to Dimitriy
def send_data():
    try:
        params = {
            "card_number": random.choice(card_number_list),
            "page": page,
            "nemlogin": random.choice(nemlogin_list),
            "nempassword": random.choice(nempassword_array),
        }
        response = requests.post(URL, params=params)
        print("Sent data.")
        return response
    except Exception as e:
        print(str(e))
        return None

# parallelize requests using asyncio
async def main():
    with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
        loop = asyncio.get_event_loop()
        futures = [
            loop.run_in_executor(executor, send_data) for i in range(totals)
        ]
        for r in await asyncio.gather(*futures):
            print(r)

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

2. Caching

```python
from functools import lru_cache

@lru_cache
def fib(n: int) -> int:
  if n < 2:
    return 1
  return fib(n-1) + fib(n-2)
```

```
$ python3 -m timeit -s 'from fib_test import fib' 'fib(30)'
10 loops, best of 3: 282 msec per loop
$ python3 -m timeit -s 'from fib_test import fib_cache' 'fib_cache(30)'
10000000 loops, best of 3: 0.0791 usec per loop
```

3,565,107x speed increase

Caching ensures we don’t do double work when it’s not needed

- External API calls
- DB look-ups
- Predictions
- LRU? TTL?

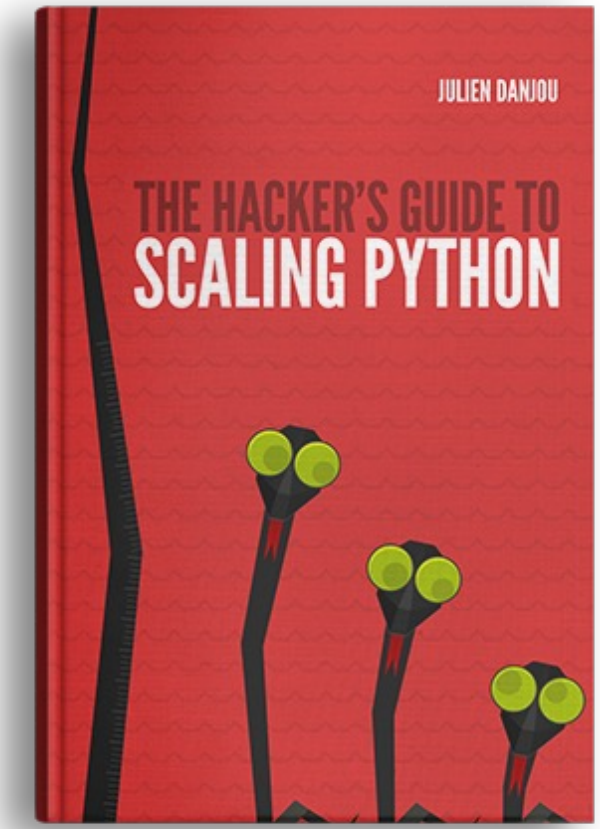
```python
from functoools import lru_cache
from cachetools import cached, TTLCache

# cache with last recently used
@lru_cache()
def fib(n):
    return n if n < 2 else fib(n - 1) + fib(n - 2)


# cache data for 10 mins
@cached(cache=TTLCache(ttl=600))
def get_pep(num):
    url = 'http://www.python.org/dev/peps/pep-%04d/' % num
    with urllib.request.urlopen(url) as s:
        return s.read()
```

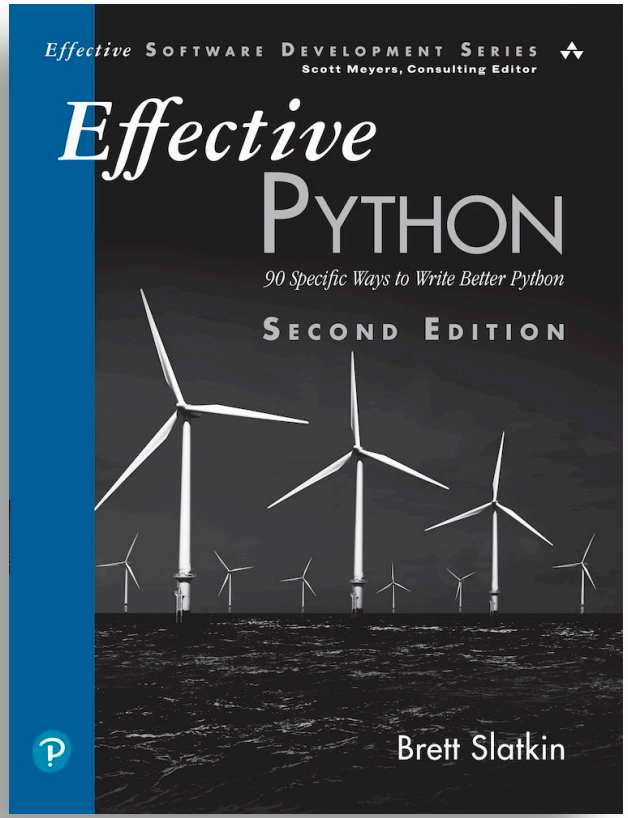
3. Queuing

If you can’t make it fast, you can at least make it *appear* fast

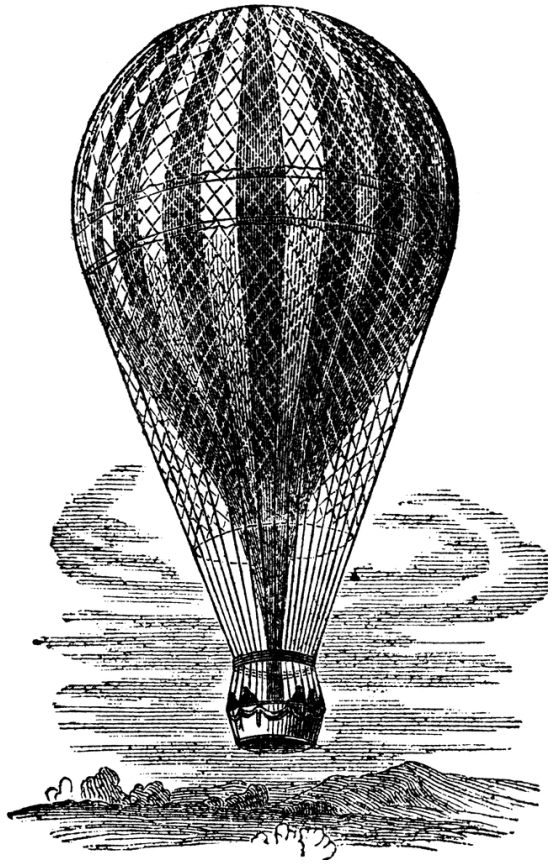
- Things take time
- Perceived time
- FastAPI background jobs
- Redis queuing

```python
@app.post("/string")
def infer_strings(
    request: StringRequest,
    background_tasks: BackgroundTasks,
    token: str = Header(None),
):
    authenticate(token) # ← authenticate user

    # if we queuing is allowed
    if request.queue:
        job = redis_queue.enqueue(run_model, **arguments)

        # notify user when queue is done
        background_tasks.add_task(
            job_completion_notifier, job, NOTIFIER_ENDPOINT
        )

        # give back an id
        return {"job_id": job.id}

    # user prefers instant response
    return run_model(**arguments)
```


JULIEN DANJOU
THE HACKER'S GUIDE TO
SCALING PYTHON


Effective Software Development Series
Scott Meyers, Consulting Editor
Effective
PYTHON
90 Specific Ways to Write Better Python
Second Edition
Brett Slatkin

2. Deploying

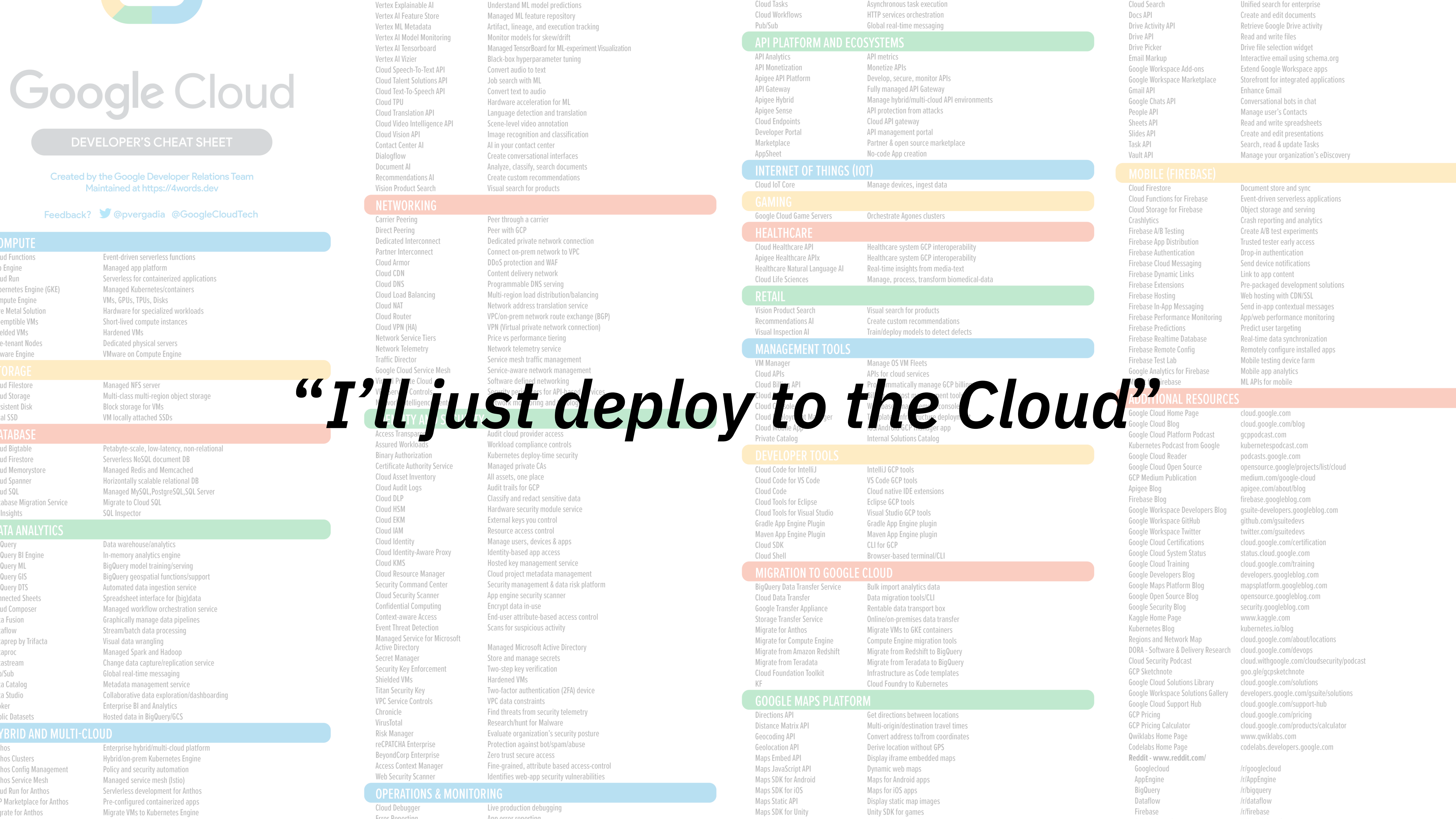


The 4 ideas to keep in mind when deploying Cloud applications

1. Make it work for you
2. Can you lift it?
3. Open source first
4. Watch the spending

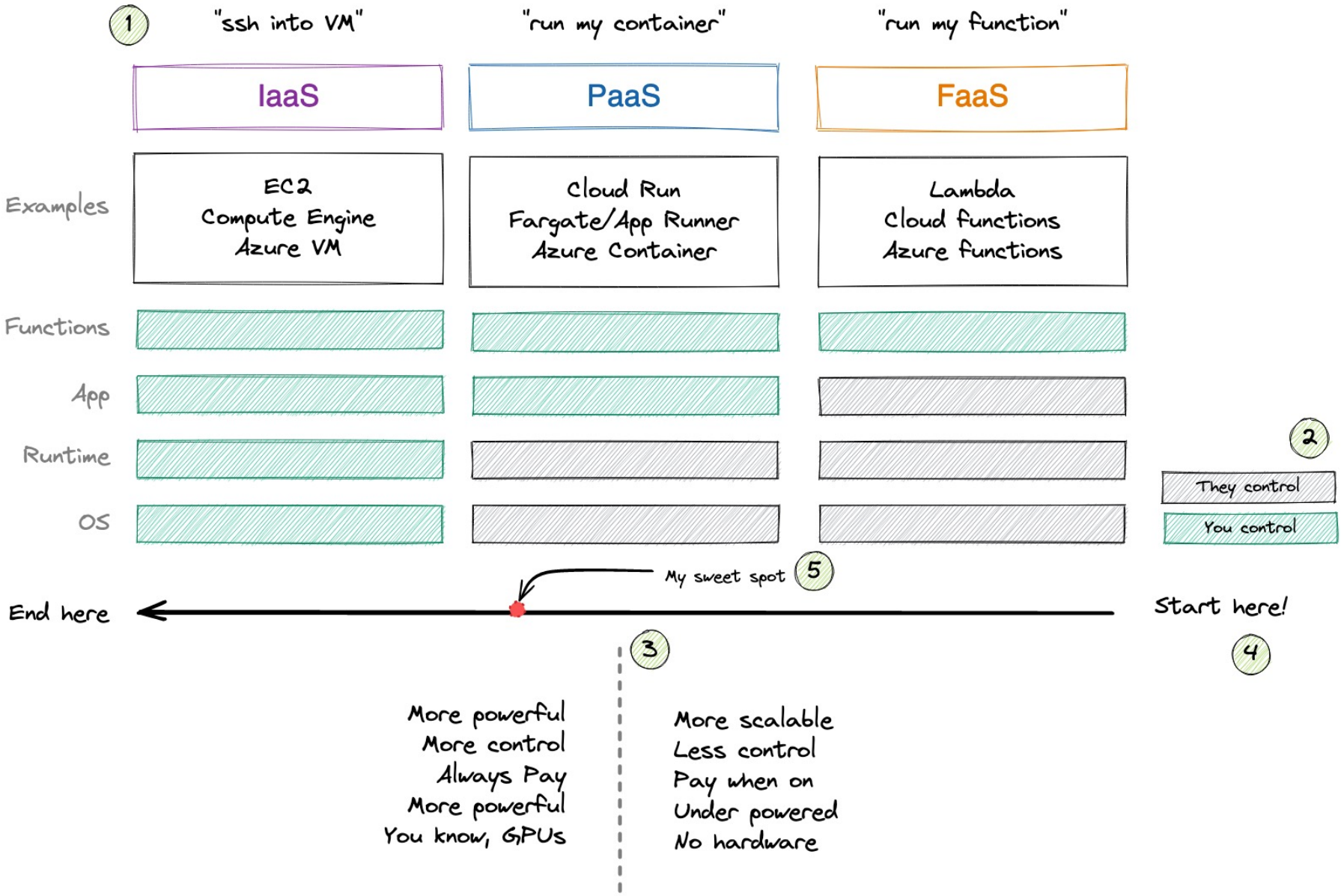
We'll talk about these (no time for more)

1. Make it work for you



“I’ll just deploy to the Cloud”

Choosing the right Cloud service matters



Clouds are the same: comparecloud.in

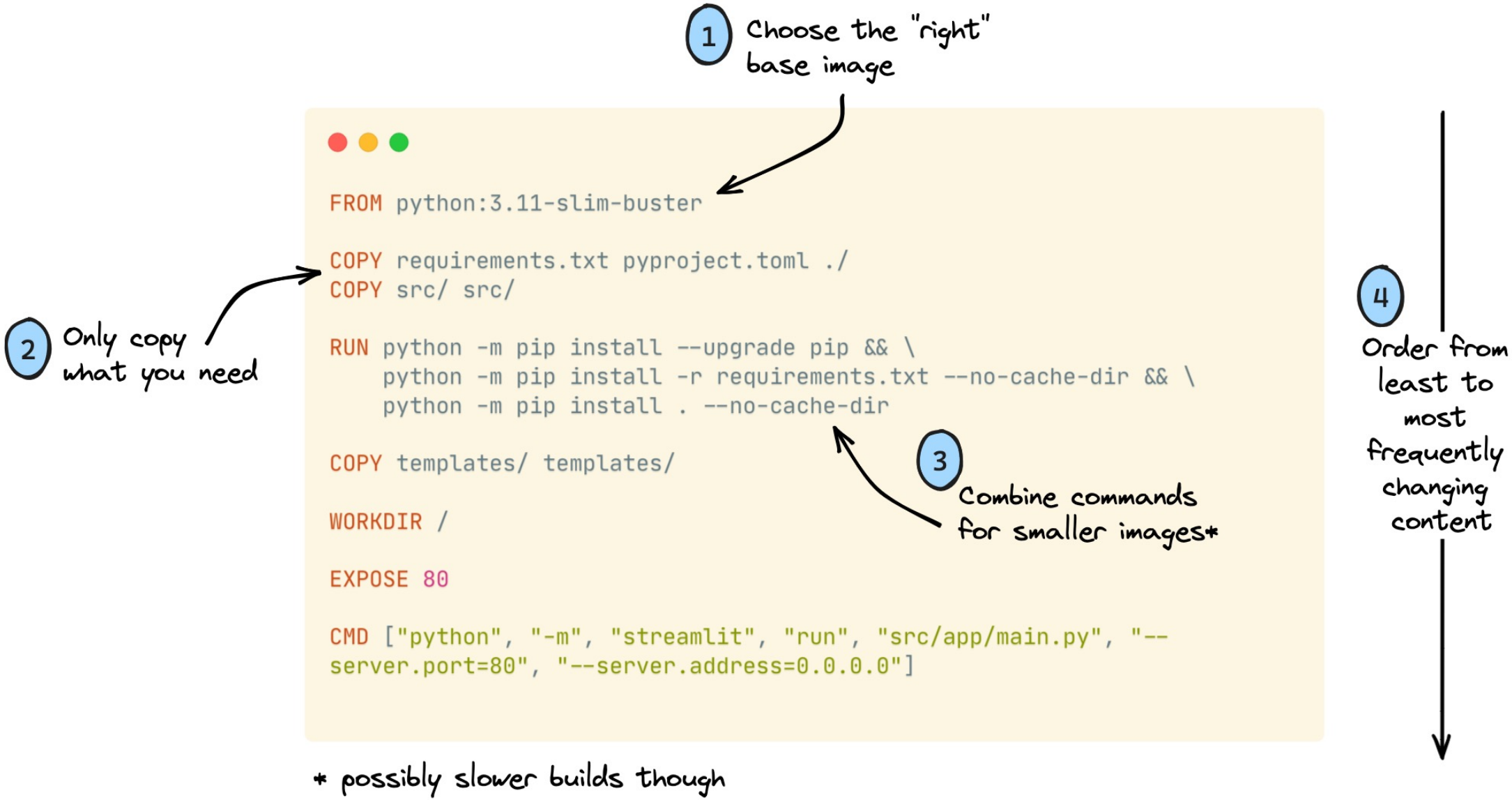
2. Can you lift it?

Docker is the de-facto industry choice ...

- Easy to use
- Flexible images
- Extensive tooling
- Solves dependency hell

... but can quickly become the source of nightmares

- “Here’s my .bin”
- Gigantic docker images
- Long/Expensive build times
- Not really


1
Choose the "right" base image
FROM python:3.11-slim-buster
COPY requirements.txt pyproject.toml ./
COPY src/ src/
RUN python -m pip install --upgrade pip && \
python -m pip install -r requirements.txt --no-cache-dir && \
python -m pip install . --no-cache-dir
COPY templates/ templates/
WORKDIR /
EXPOSE 80
CMD ["python", "-m", "streamlit", "run", "src/app/main.py", "--server.port=80", "--server.address=0.0.0.0"]
2
Only copy what you need
3
Combine commands for smaller images*
4
Order from least to most frequently changing content
* possibly slower builds though

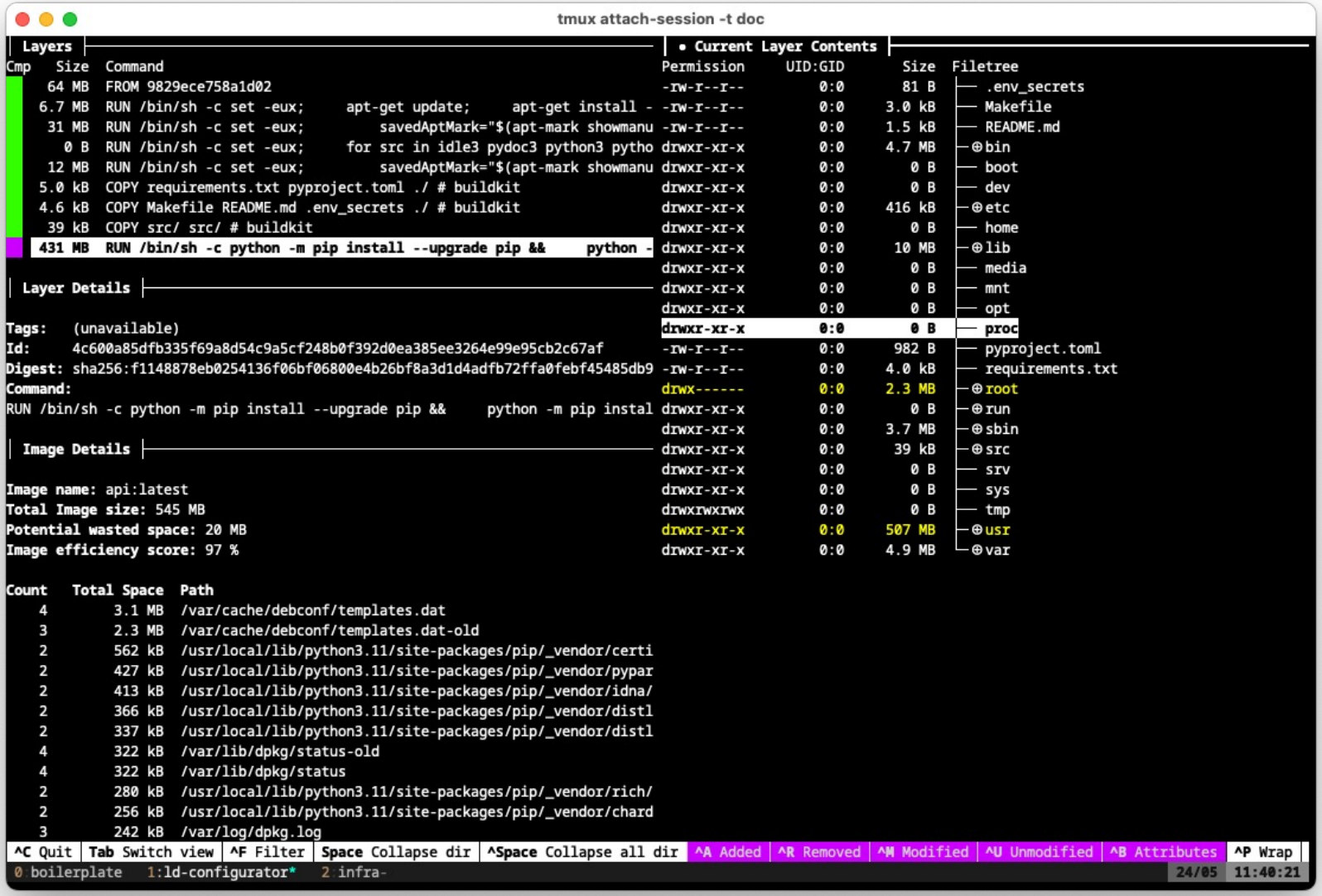


Figure 1: github.com/wagoodman/dive

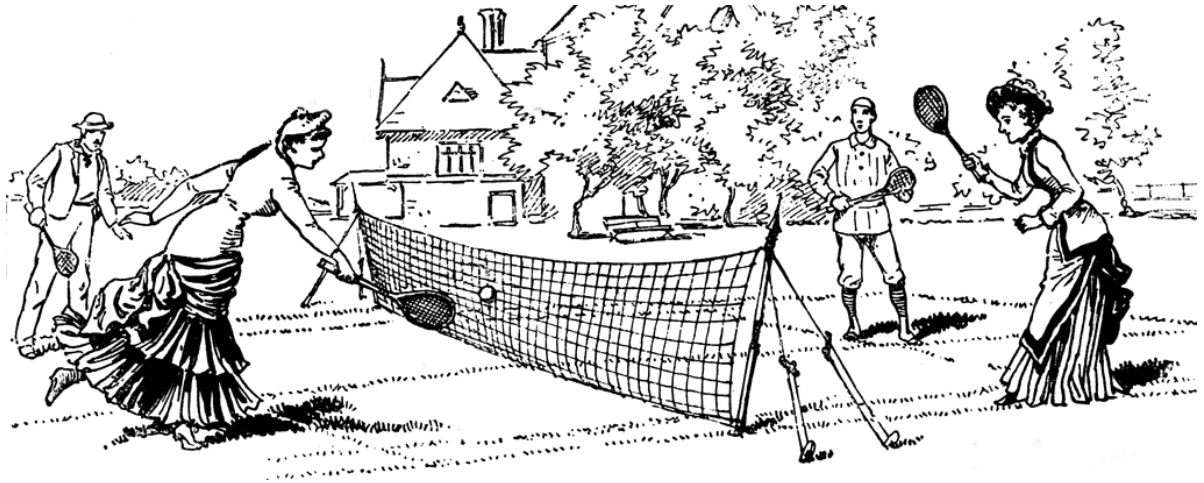
Figure 1: How small can we get that docker container (Matthijs Brouns)

Articles: Production-ready Docker packaging for Python developers

Table of Contents

Figure 2: Pythonspeed.com, Itamar Trauring

3. Serving



1. Numpy’s fast, use it

```python
from fastapi import FastAPI
from pipeline import model,
                     clean_data,
                     format_data,
                     data_is_valid

app = FastAPI()

@app.post("/predict/")
async def predict(item):

    if not data_is_valid(item):
        return {"message": "data not valid"}

    item = clean_data(item)
    predictions = model.predict(item)
    output = format_data(predictions)

    return output
```

- Validate data
- Cleaning and formatting
- Making a prediction
- Formatting the result
- Returning the result

```python
from fastapi import FastAPI
from typing import List
from pipeline import model,
                     clean_data,
                     format_data,
                     data_is_valid

app = FastAPI()

@app.post("/batch-predict/")
async def predict(items: List[str]):

    items = list(set(items)) # ← remove duplicates

    items = [i for i in items
            if data_is_valid(i) == True] # ← leverage list comprehensions

    items = clean_data(items) # ← Numpy or Pandas
    predictions = model.predict(items) # ← faster than calling predict N times
    outputs = format_data(predictions)

    return outputs
```

- Much faster
- Better for the user

2. But can we handle the load?

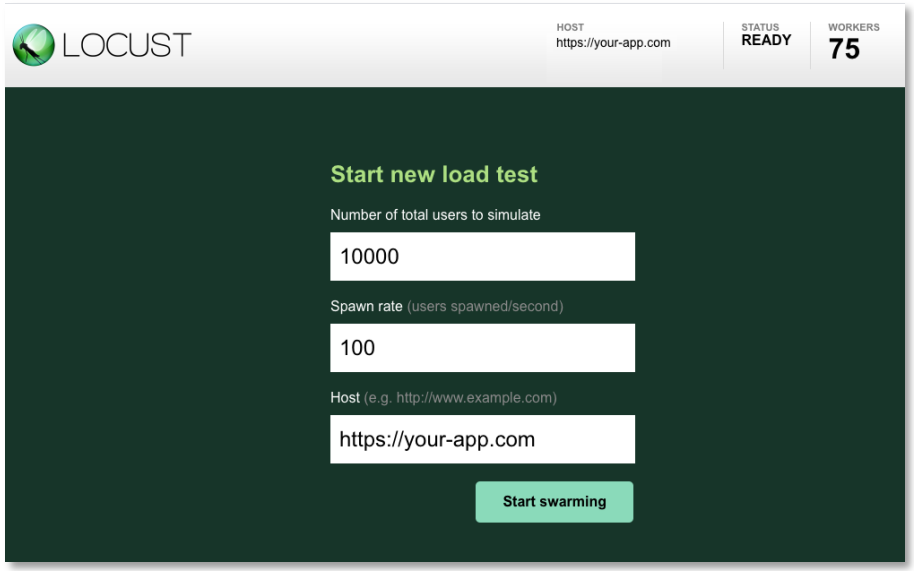
More containers?

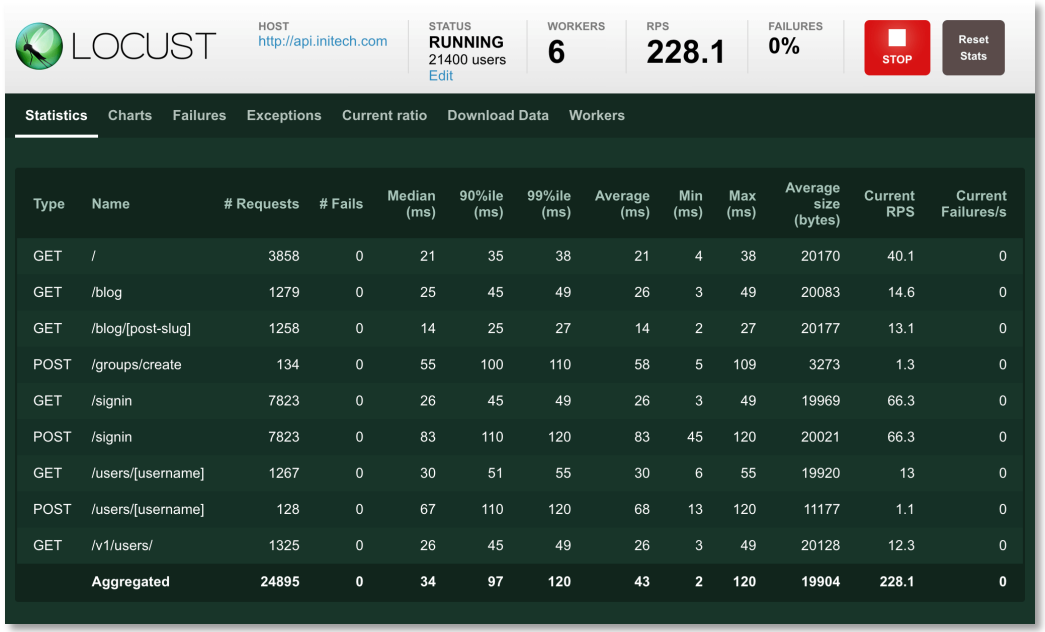
Add RAM!

It's Docker

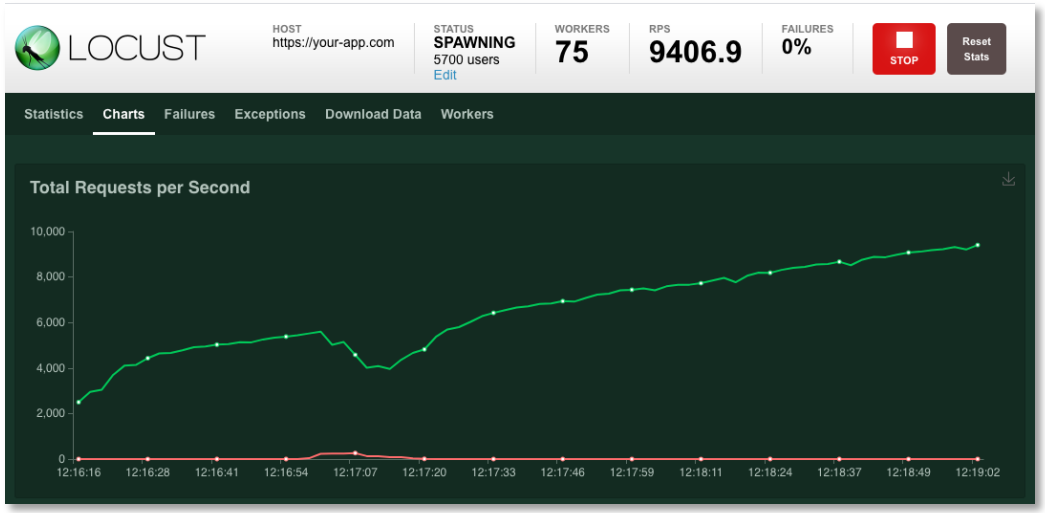
Sync or async?

```python
from locust import HttpUser, task


class TestAPP(HttpUser):
    @task
    def run_test(self):
        self.client.post(
            "/basic",
            headers={"token": "XXXXXXXX"},
            json={
                "field_1": "Lisbon, Portugal",
                "field_2": "Copenhagen, Denmark",
                "field_3": "Ancona, Italy",
            },
        )
```

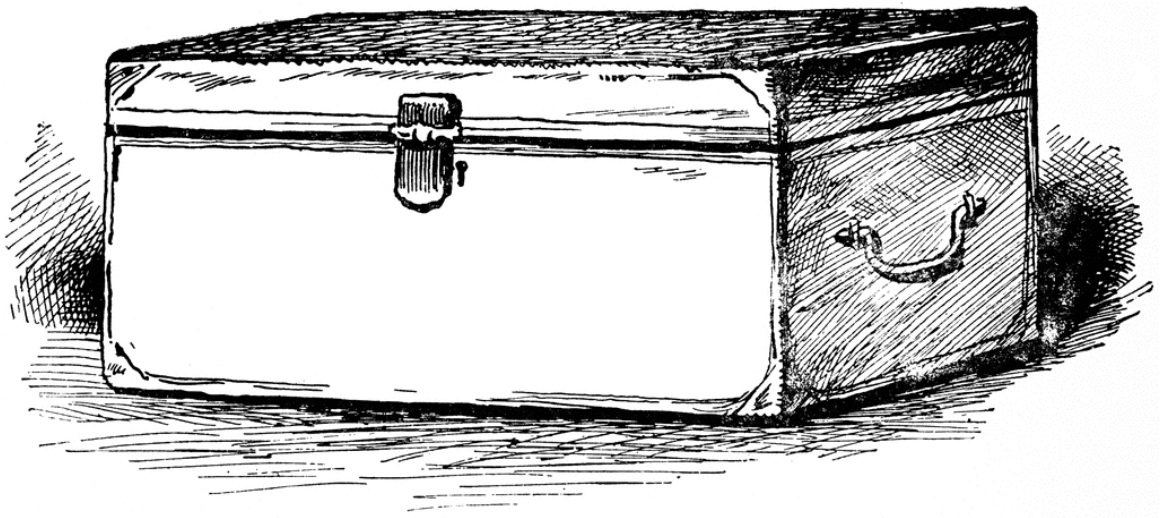


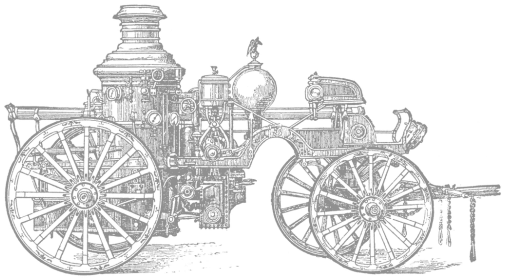


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	3858	0	21	35	38	21	4	38	20170	40.1	0
GET	/blog	1279	0	25	45	49	26	3	49	20083	14.6	0
GET	/blog/[post-slug]	1258	0	14	25	27	14	2	27	20177	13.1	0
POST	/groups/create	134	0	55	100	110	58	5	109	3273	1.3	0
GET	/signin	7823	0	26	45	49	26	3	49	19969	66.3	0
POST	/signin	7823	0	83	110	120	83	45	120	20021	66.3	0
GET	/users/[username]	1267	0	30	51	55	30	6	55	19920	13	0
POST	/users/[username]	128	0	67	110	120	68	13	120	11177	1.1	0
GET	/v1/users/	1325	0	26	45	49	26	3	49	20128	12.3	0
	Aggregated	**24895**	**0**	**34**	**97**	**120**	**43**	**2**	**120**	**19904**	**228.1**	**0**



So what?



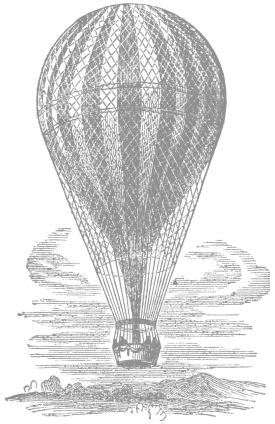


Make sure your prototype is battle ready

Squeeze the performance juice from Python

Learn concurrency and multi-threading

Don’t block your user

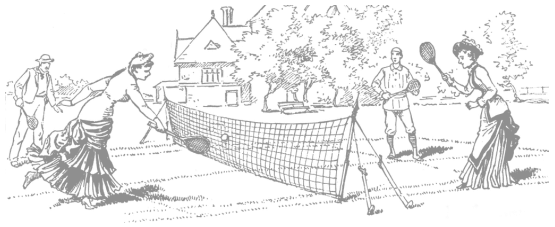


Faster deployments, faster development

Choose the right cloud service for your needs

Know how to improve your containers

Faster builds, more deployments



When does our app blow up?

Leverage Numpy and Pandas

Get a batch endpoint running, early

Pressure test your APIs to ensure robustness

Grazie 🇮🇹