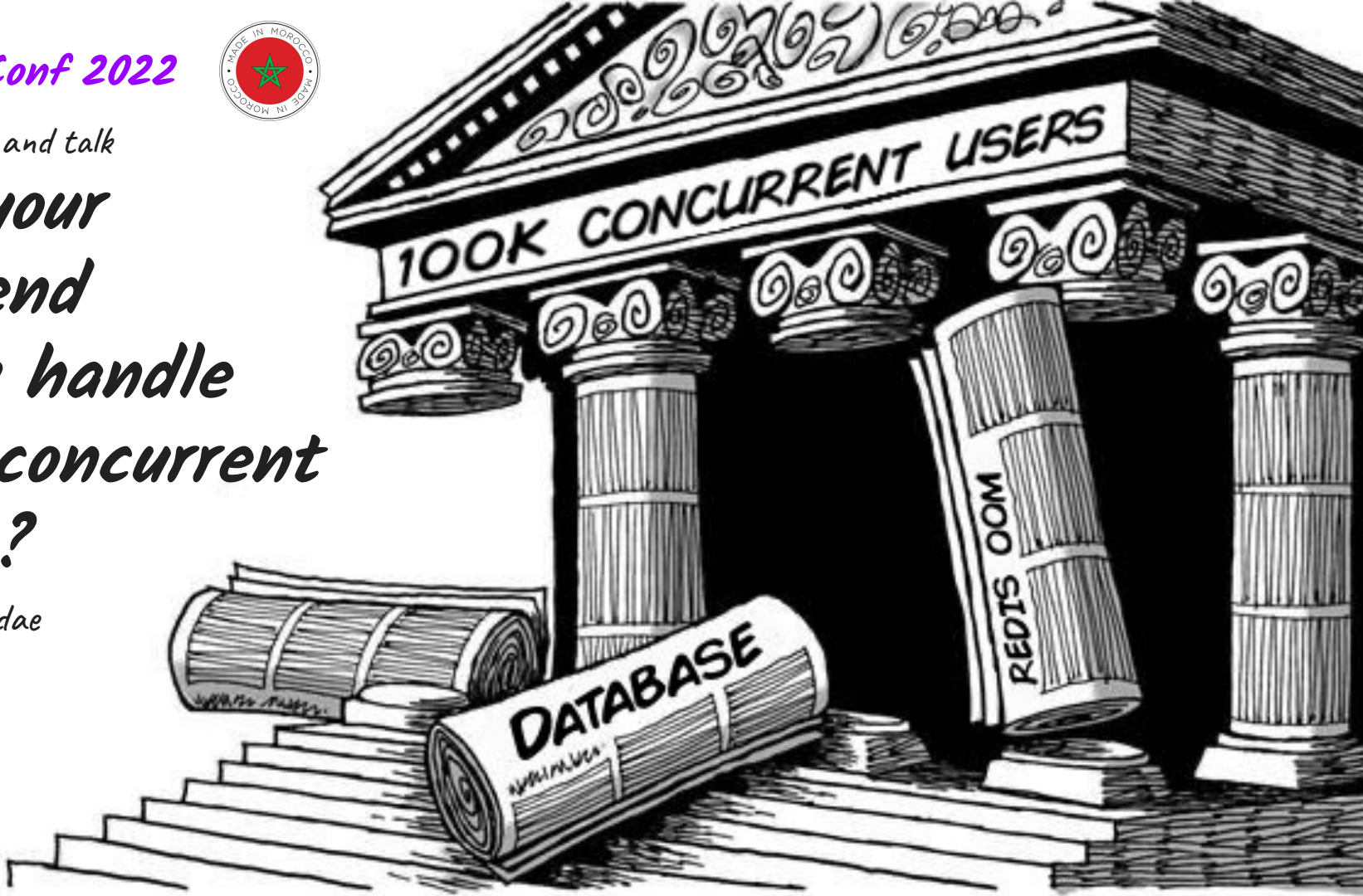
#BlablaConf 2022

Let's cruise and talk

Can your backend really handle 100K concurrent users?

By @Kaizendae



مرحبا

$ whoami

~ Abdelati Elasri

~ Full-time error correction mechanism

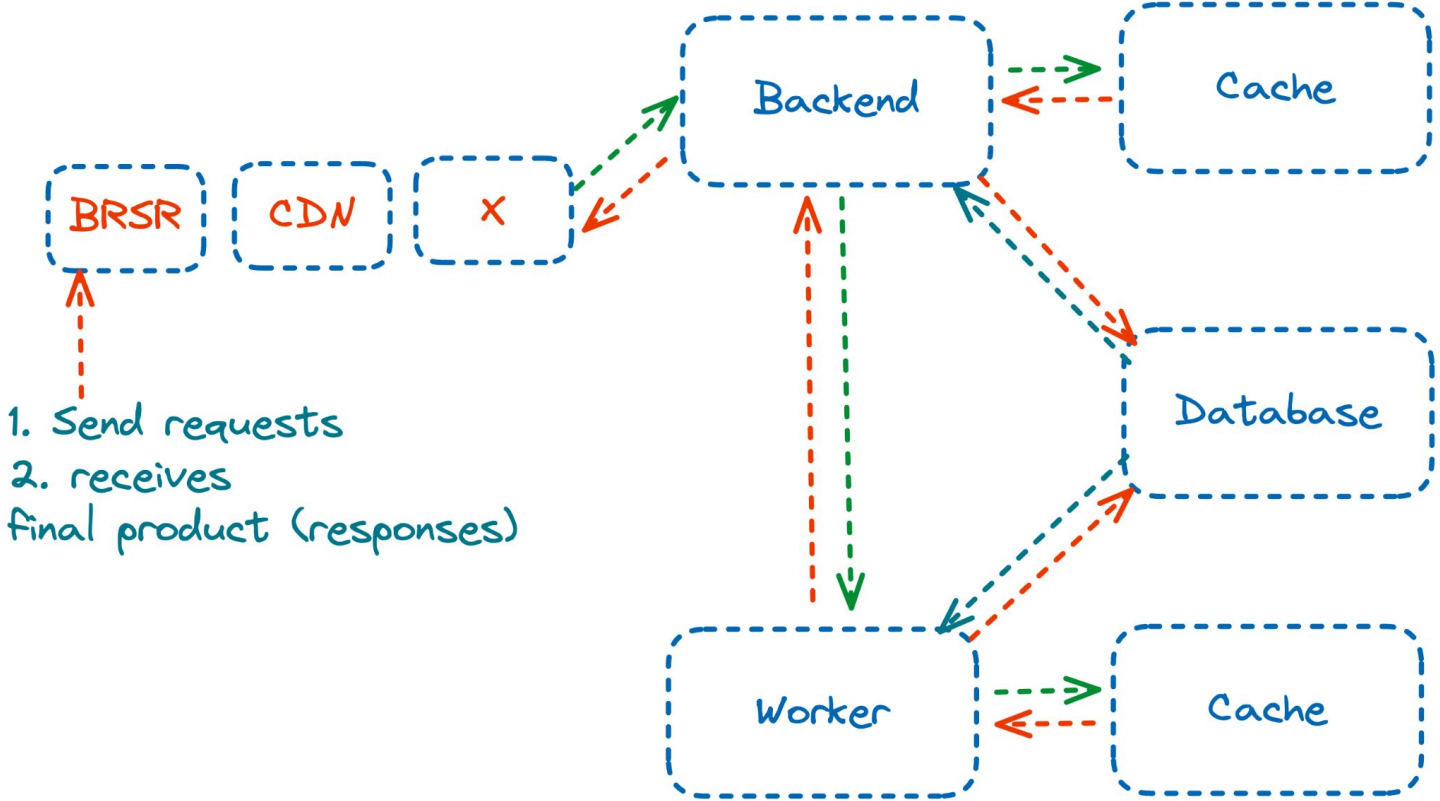
~ Draws and Reads

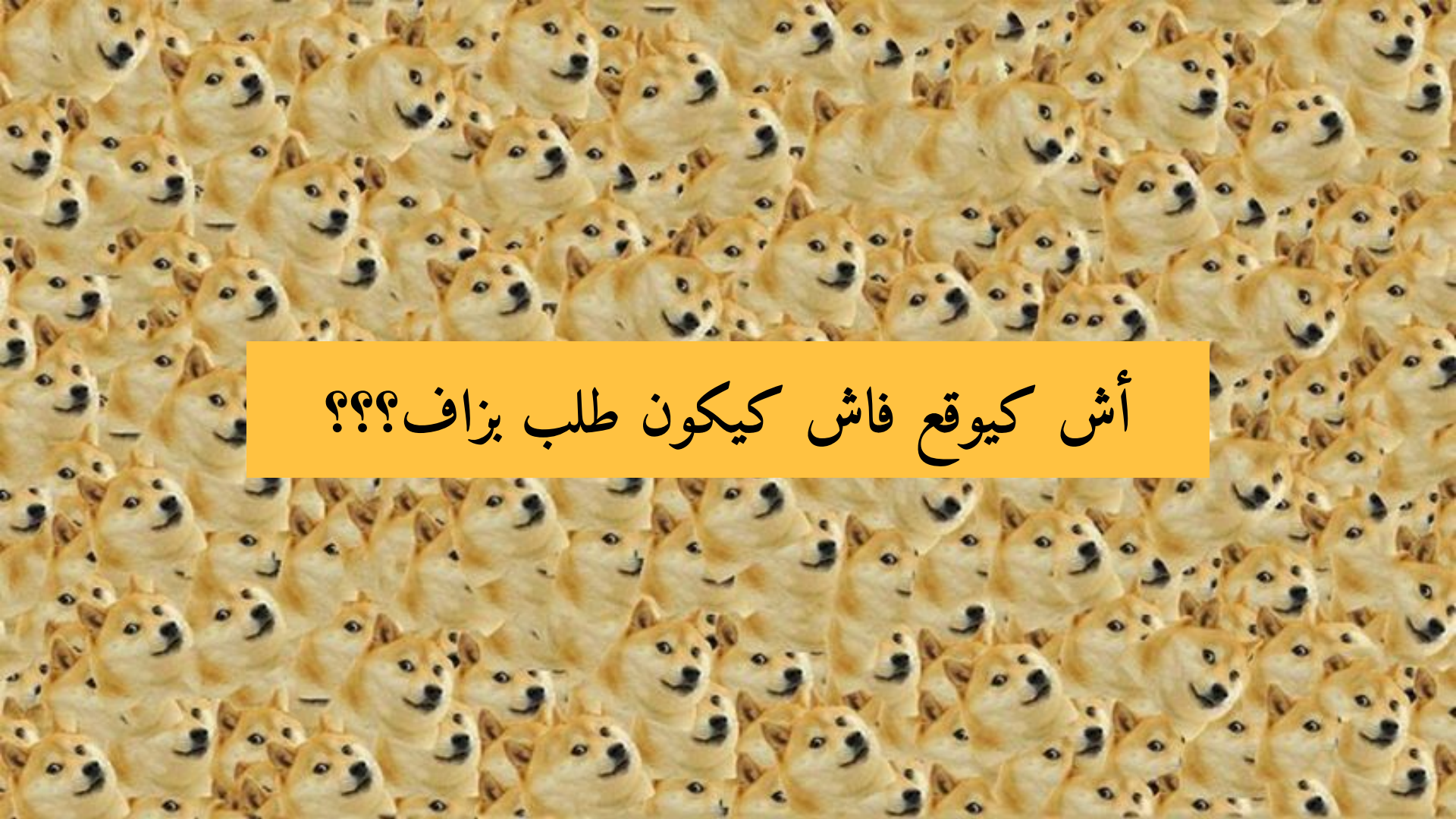
~ @Kaizendae a7ssan compte f twitter

$ whatis this

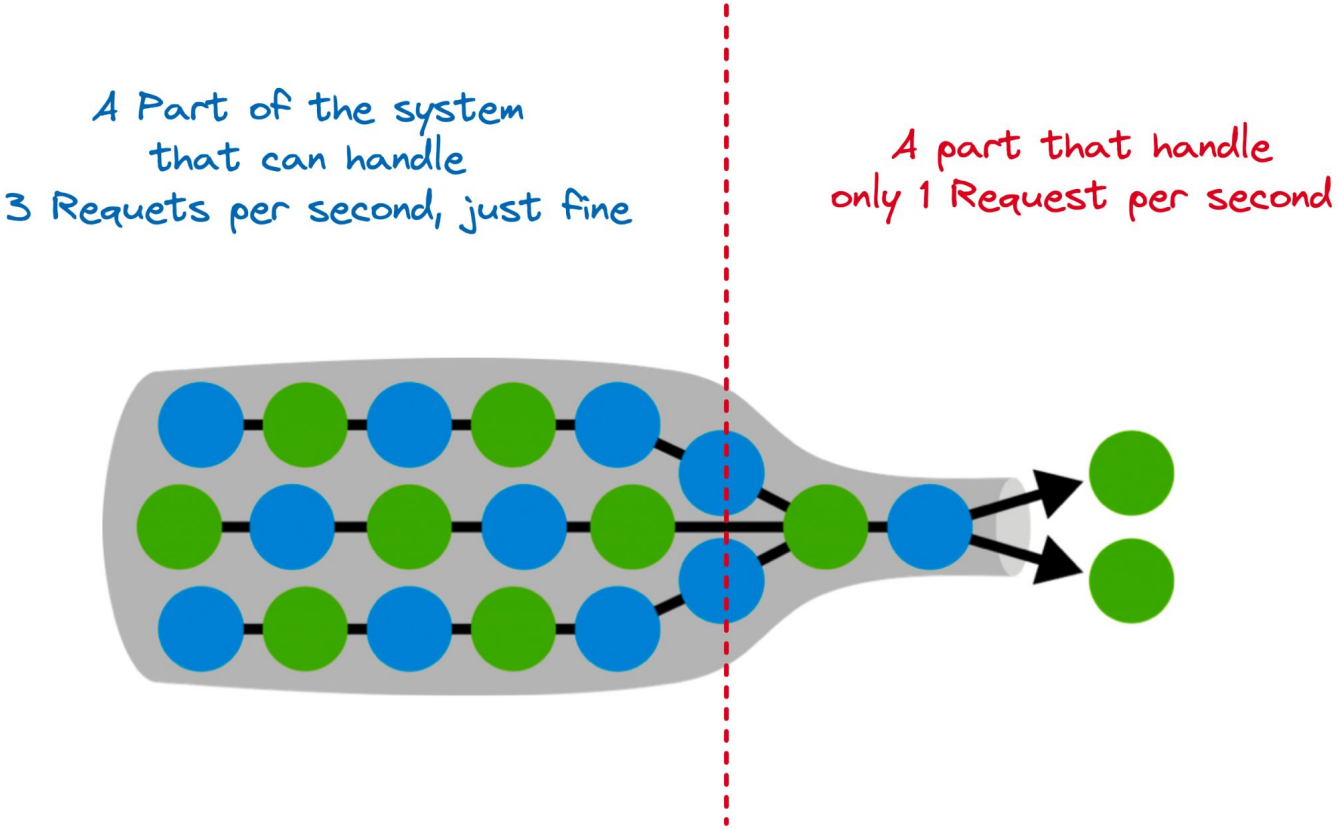
- Context.
- Tool.
- Journey(Acts and Lessons)
- Demo.
- Outro.

backend ~= Supply Chain

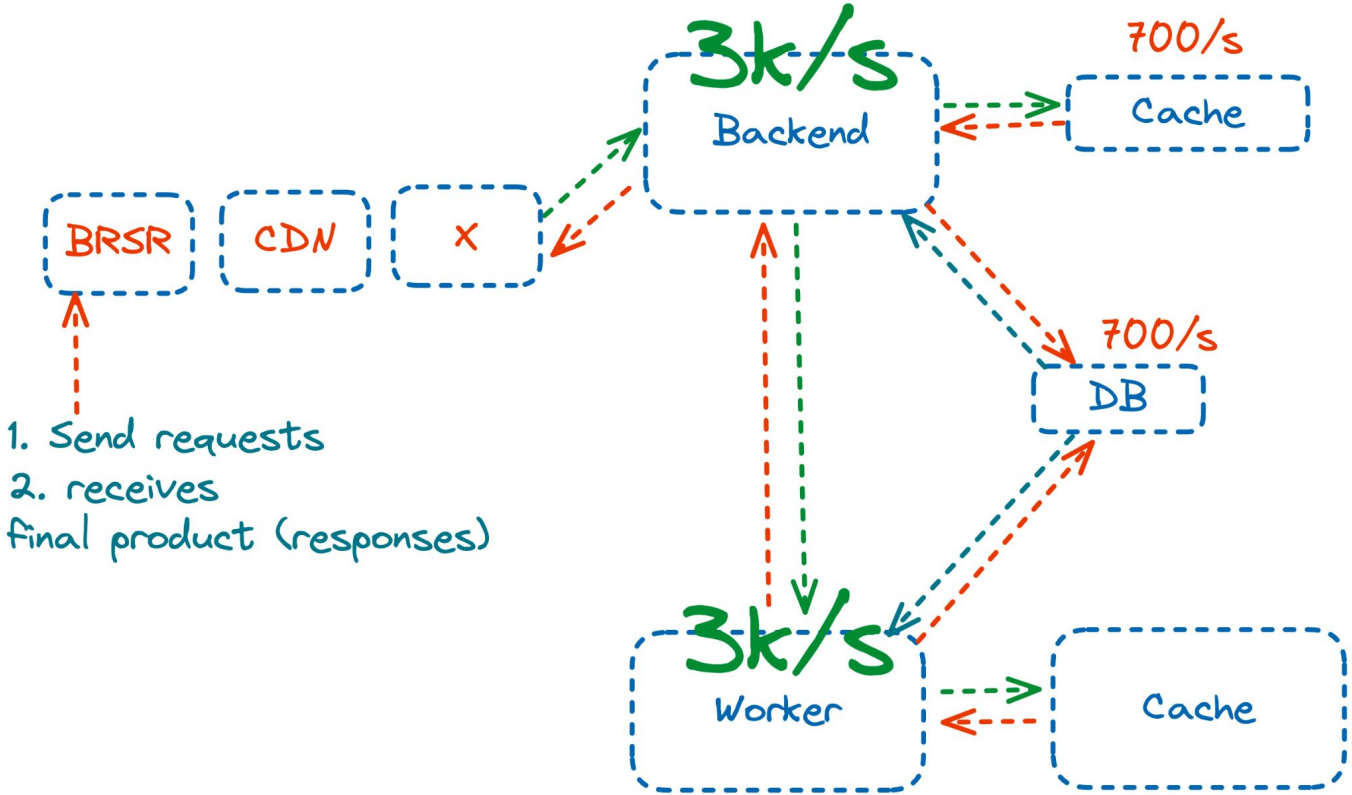



أش كيوقع فاش كيكون طلب بزاف؟؟؟

Bottlenecks happen.



Webapps ~= Supply Chain



ما هي الأسباب المحتملة

Misconfiguration
DB, Cache, Webserver

Capacity

Bottlenecks

Memory
Leaks

Autoscaling

لي غايطرا راه جاي ~ قانون مورفي

Two ways of finding out.

REACTIVE

VS

PRO-ACTIVE

الوقاية _____ العلاج

البراغماتية فْ الإختيار

التحضير للمحنة يخفف من حدتها

PRO-ACTIVE

VS

الوقاية خير من العلاج، ليس دائما

REACTIVE

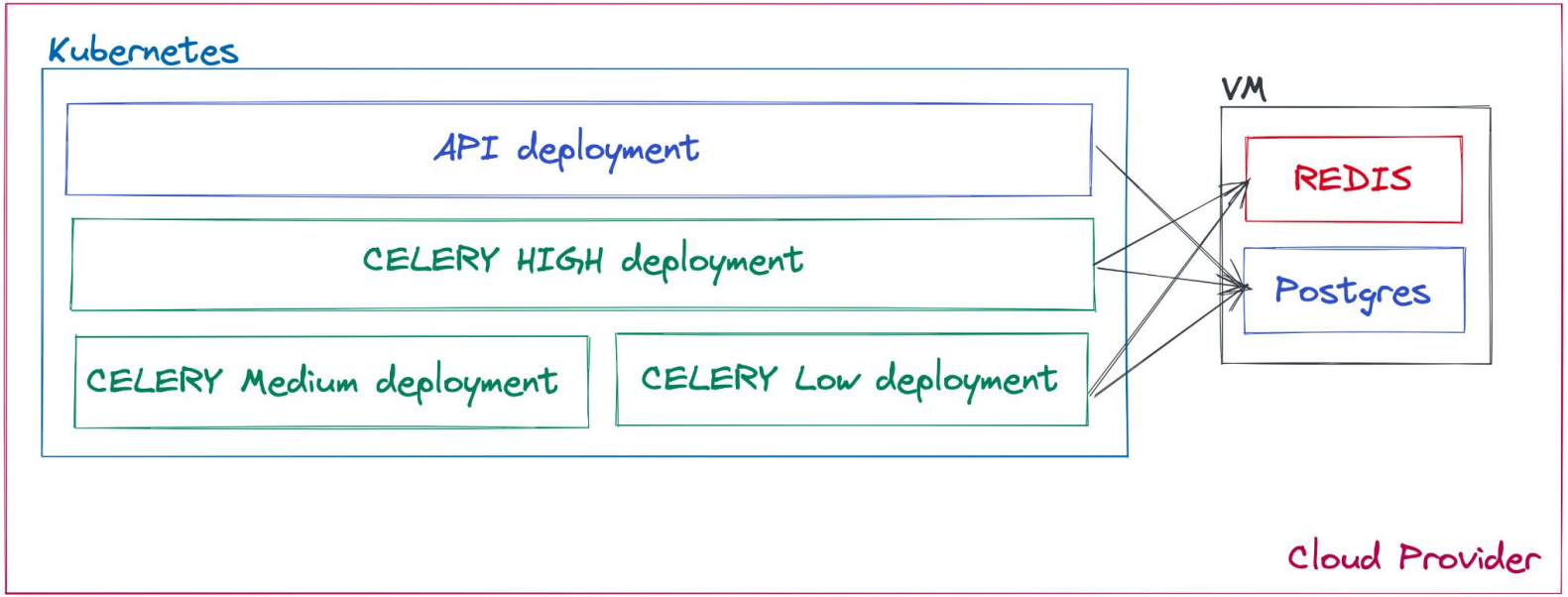
ProActive == Performance Testing

Proactively identify **bottlenecks** and (**slow response times**, high CPU and memory usage, and **low throughput**)

Ensure that the **system can handle the expected workload** in a production environment.

Action, Take note and fix any issues before users face it.

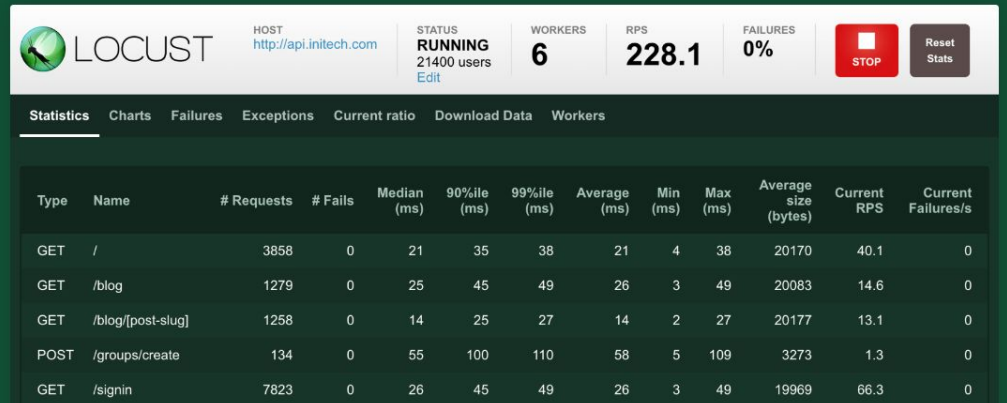
The_Context/ Proactive, expecting 100k concurrent users because of a new client.



Disclaimer

*Prod**

Quick_Locust_Intro/



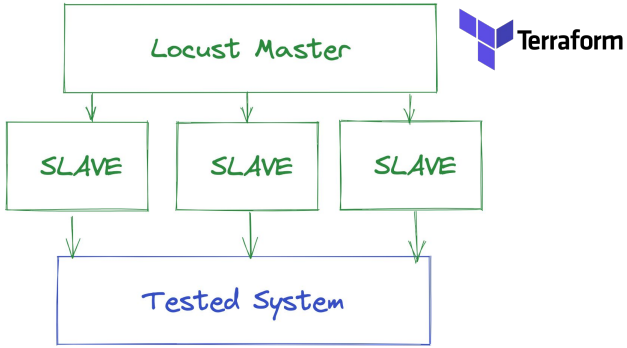
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	3858	0	21	35	38	21	4	38	20170	40.1	0
GET	/blog	1279	0	25	45	49	26	3	49	20083	14.6	0
GET	/blog/[post-slug]	1258	0	14	25	27	14	2	27	20177	13.1	0
POST	/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

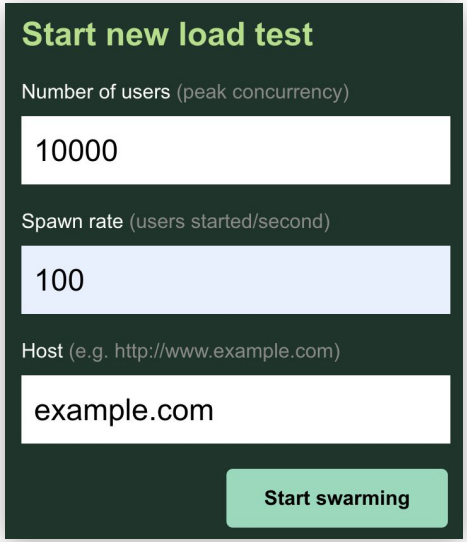
Quick_Locust_Intro/

locustfile.py

```python
from locust import HttpUser, between, task

class WebsiteUser(HttpUser):
    wait_time = between(5, 15)

    def on_start(self):
        self.client.post("/login", {
            "username": "test_user",
            "password": ""
        })

    @task
    def index(self):
        self.client.get("/")
        self.client.get("/static/assets.js")

    @task
    def about(self):
        self.client.get("/about/")
```

```
$ locust -f locustfile.py
```





LOCUST

HOST
http://api.initech.com

STATUS
RUNNING
21400 users
Edit

SLAVES
6

RPS
240

FAILURES
0%

STOP

Reset Stats

Statistics Charts Failures Exceptions Download Data Slaves

Type	Name	X	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
GET	/	V1	5416	0	21	21	4	38	20336	44.1
GET	/blog	V2	1745	0	27	26	3	49	20370	13.7
GET	/blog/[post-slug]	V1	1824	0	15	15	2	27	19943	15.9
POST	/groups/create	V1	185	0	57	55	5	108	3273	1.9
GET	/signin	V3	10266	0	26	26	3	49	19949	66.6
POST	/signin	V1	10266	0	82	82	45	120	20030	66.6
GET	/users/[username]	V2	1802	0	31	31	6	55	20194	15
POST	/users/[username]	V3	186	0	73	73	14	120	11178	1
GET	/v1/users/	V2	1791	0	26	26	3	49	19806	15.2
	Total		**33481**	**0**	**34**	**42**	**2**	**120**	**19923**	**240**

The Journey & Protagonist

المشهد الأول

تمهيدي

1- Write load test.

2. launch the swarming of the system

3- observe response time

locustfile.py

```python
from locust import HttpUser, between, task


class WebsiteUser(HttpUser):
    wait_time = between(5, 15)

    def on_start(self):
        self.client.post("/login", {
            "username": "test_user",
            "password": ""
        })

    @task
    def index(self):
        self.client.get("/")
        self.client.get("/static/assets.js")

    @task
    def about(self):
        self.client.get("/about/")
```

```
$ locust -f locustfile.py
```

Your env looks this way,

but...

What's actually happening

العشوائية ماشي إستراتيجية مزيانة

Applications | slack | Roadmap | Docs | Blog | Switch Org

Sirius (Production)

30 min ending Sep 3 10:44AM | Compare To...

Overview | Web Endpoints | Background Jobs | Database | Traces | Alerts | Reports | Settings

Last Updated: 11:00AM

750 ms | 500 ms | 250 ms | 0 ms

30 rpm | 20 rpm | 10 rpm | 0 rpm

10:16 am | 10:18 am | 10:20 am | 10:22 am | 10:24 am | 10:26 am | 10:28 am | 10:30 am | 10:32 am | 10:42 am

Throughput | Middleware | SQL | Controller

Sep 03 10:40 am - 10:41 am

Throughput	15 rpm
Middleware	0.4 ms
SQL	88.3 ms
Controller	312.5 ms
Total Response Time (Mean)	401.2 ms

Drag to view details

RESPONSE TIME - MEAN	RESPONSE TIME - 95TH	THROUGHPUT	ERRORS	APDEX
111.4 ms	744.5 ms	15.2 / min	0.0 / min	1.0

N+1 Insights 2 | Slow Query Insights 0 | Errors 2 NEW

n+1 Insights

Showing insights over the past 24 hours.

Scout analyzes your application for N+1 queries consuming over 150 ms. Click on an item to view a transaction trace where the N+1 query occurred.

App\Http\Controllers\Roles\RoleController@store

10:23:00 AM · first occured 37 minutes ago

6 queries taking 1,949 ms

App\Http\Controllers\Roles\RoleController@update

10:13:00 AM · first occured about 1 hour ago

6 queries taking 1,232 ms

Show More Insights

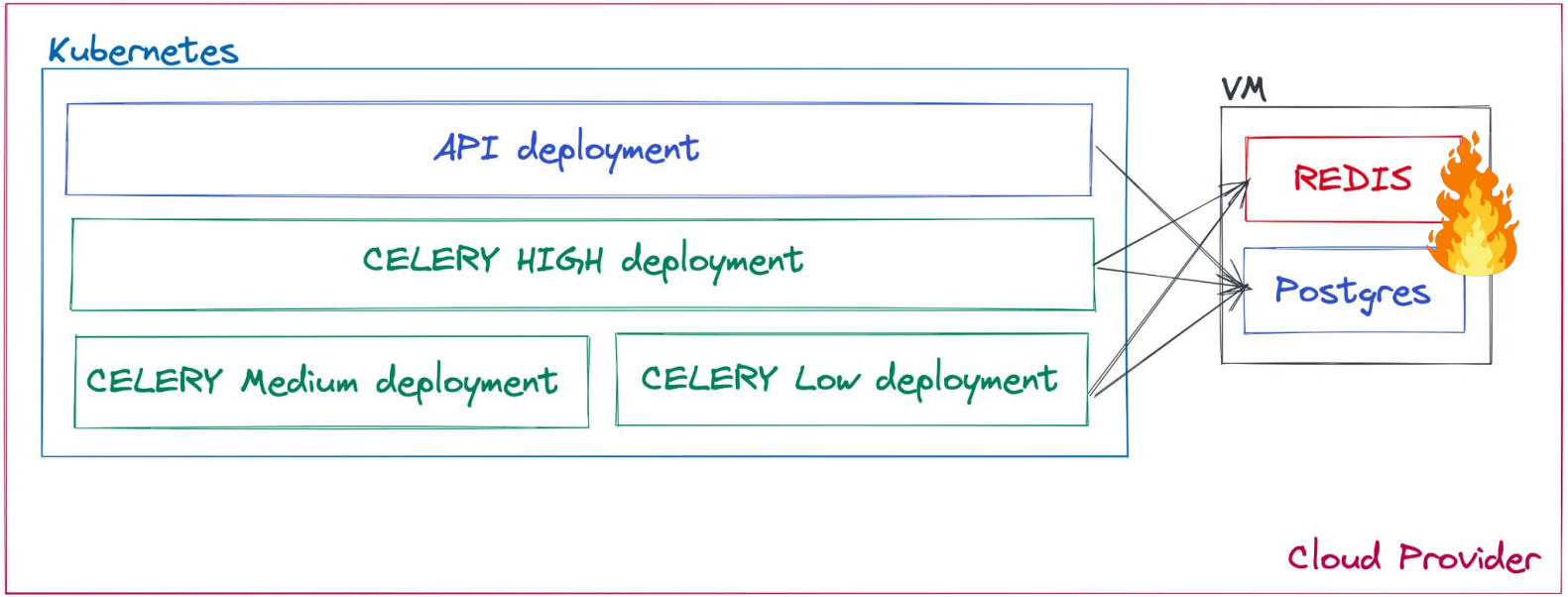

Docker Containers
Last 15 minutes
10s
CPU Load
4.08%
CPU Cores
2
Memory Load
38%
Used Memory
717.62 MiB
Storage Load
N/A
Used Storage
53.01 GiB
Running Containers
System Load
I/O Usage
Container CPU Usage
min max avg
cloud-status_app_1 0.30% 0.38% 0.33%
cloud-status_db_1 0.06% 0.09% 0.07%
portainer 0.00% 0.02% 0.01%
Container Memory Usage
min max avg
cloud-status_app_1 52.05 MiB 52.05 MiB 52.05 MiB
cloud-status_db_1 44.56 MiB 44.56 MiB 44.56 MiB
portainer 28.95 MiB 28.97 MiB 28.95 MiB
Container Cached Memory Usage
min max avg
cloud-status_app_1 15.74 MiB 15.74 MiB 15.74 MiB
cloud-status_db_1 8.21 MiB 8.21 MiB 8.21 MiB
portainer 19.70 MiB 19.70 MiB 19.70 MiB
Container Network Input
min max avg
cloud-status_app_1 0 B 0 B 0 B
cloud-status_db_1 0 B 0 B 0 B
portainer 0 B 0 B 0 B
Calendar
Container Network Output
min max avg
cloud-status_app_1 0 B 0 B 0 B
cloud-status_db_1 0 B 0 B 0 B
portainer 0 B 0 B 0 B

دابا نقدر نماركي

المشهد الثاني

Chedira launched the load testing again...

After swarming the backend...



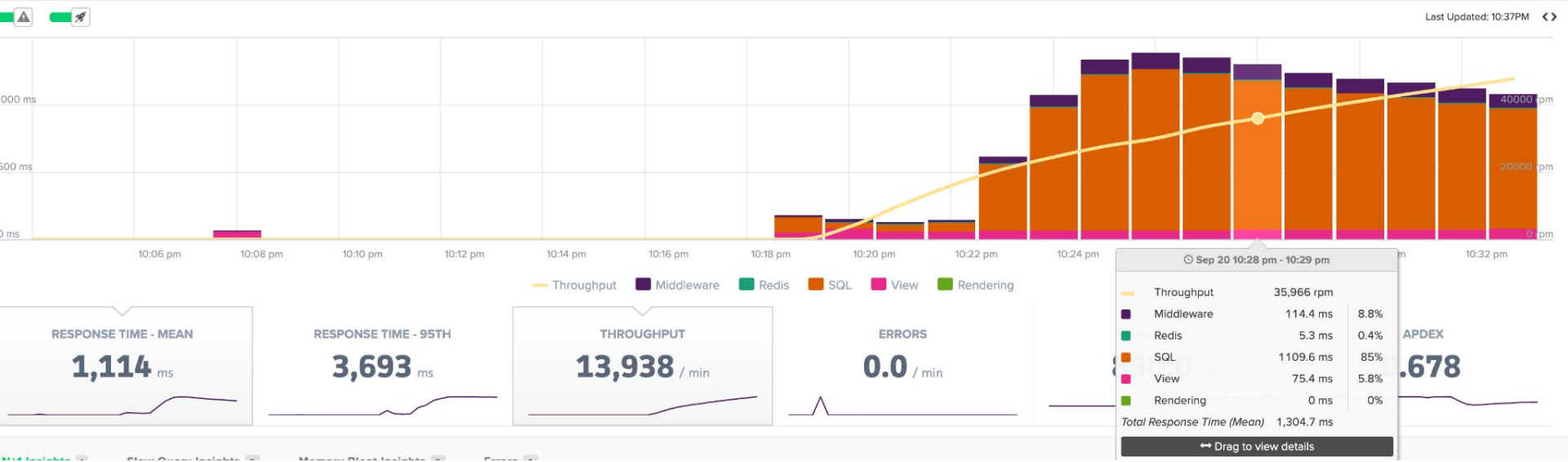
Temporary solution, validate theory, go Next...

Lessons

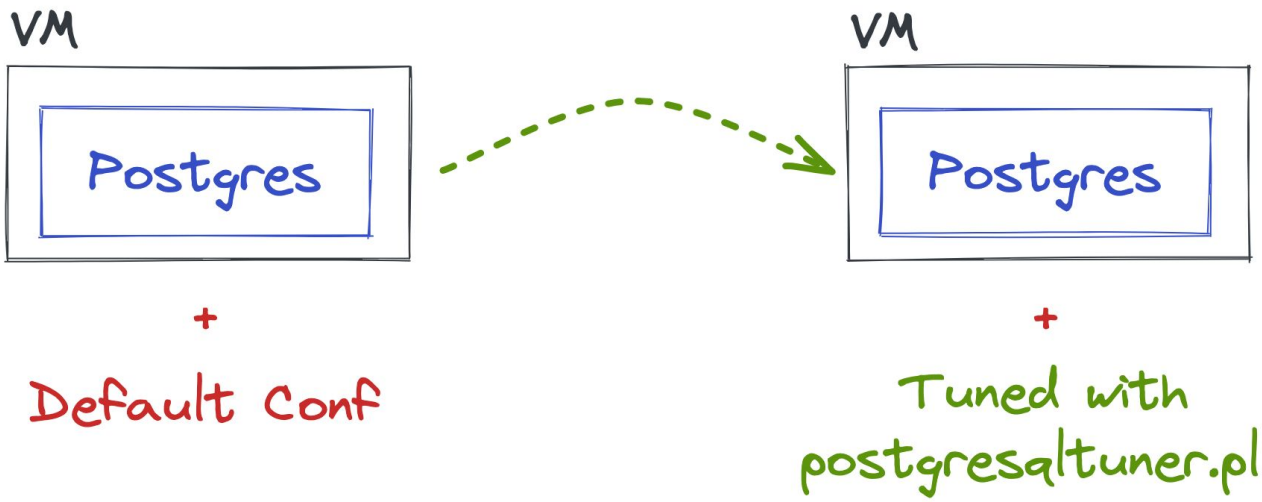
- Use Resource quotas
- Pay attention to keys TTL
- Separate DBs in redis

المشهد الثالث

Launch - Terrible response time - SQL.



The vicious loop



Lessons

- Don't trust defaults
- Use managed services(it depends)

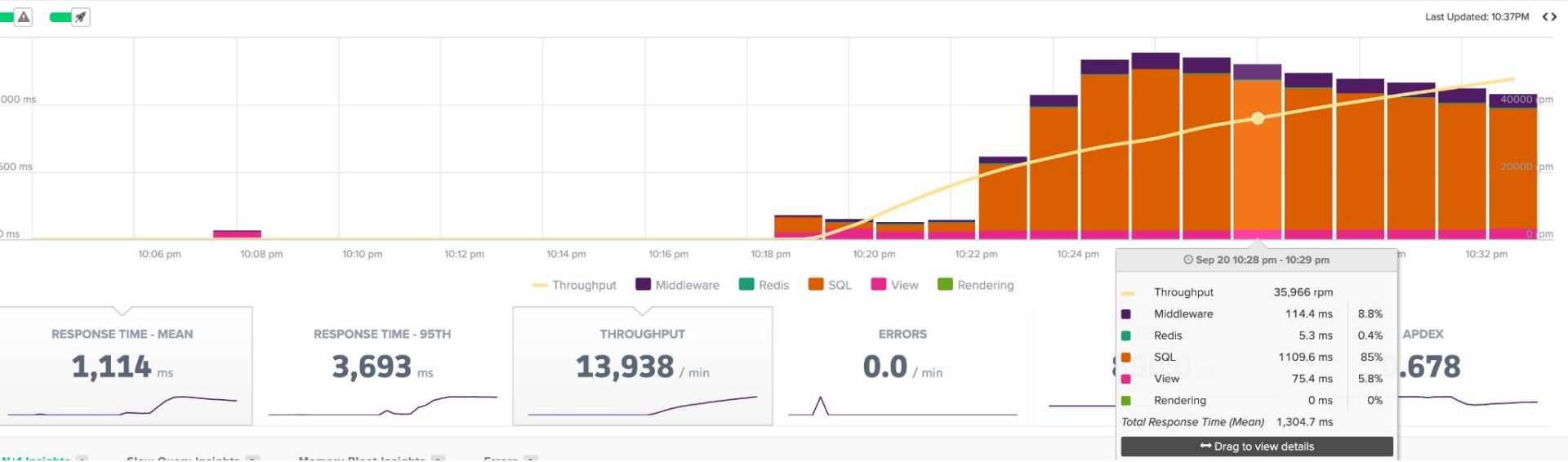
واخا الناس كيعرفو يطيبو
نهار العرس كيجيبو طباخة پرو (Managed Service)

المشهد الرابع

Scaled the Database to Extra++++ resources and tuned it,

Launched the swarming again, and...

...The Database still is bottleneck



Beyond 700R/s vertically scaling the database does nothing.

Only an expert can read these mixed signals

Beyond 700 RPS vertically scaling the database does nothing.

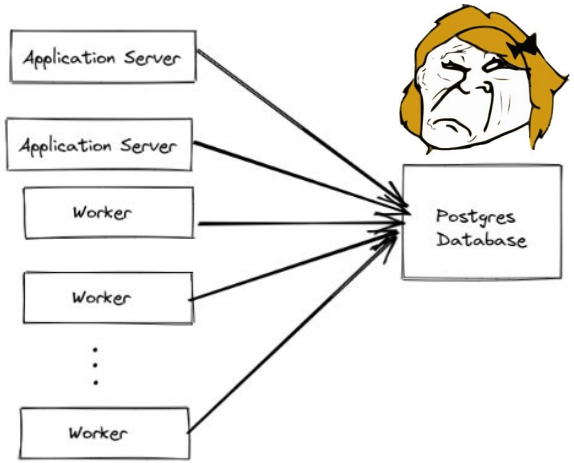
Postgres suffers on many many connections

- In modern web applications, clients tend to open a lot of connections.

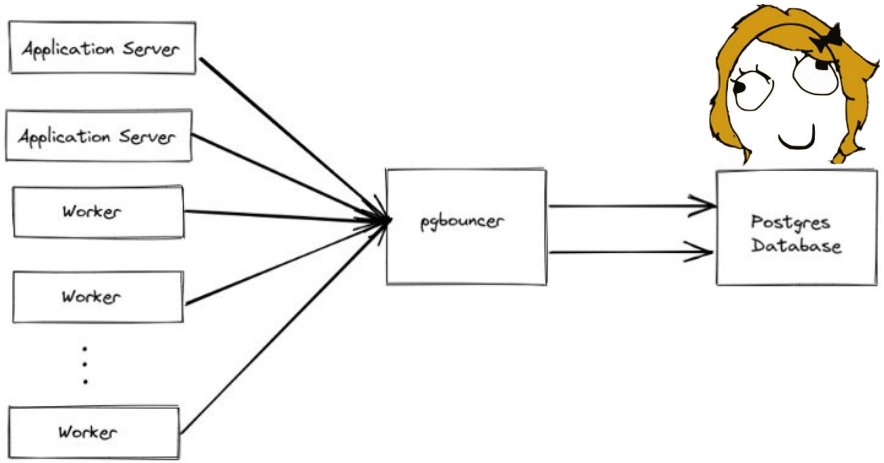
- "Open a connection as late as possible, close a connection as soon as possible".

- forking a process becomes expensive when transactions are very short.

Postgres suffers on many many connections



Without connection pooling



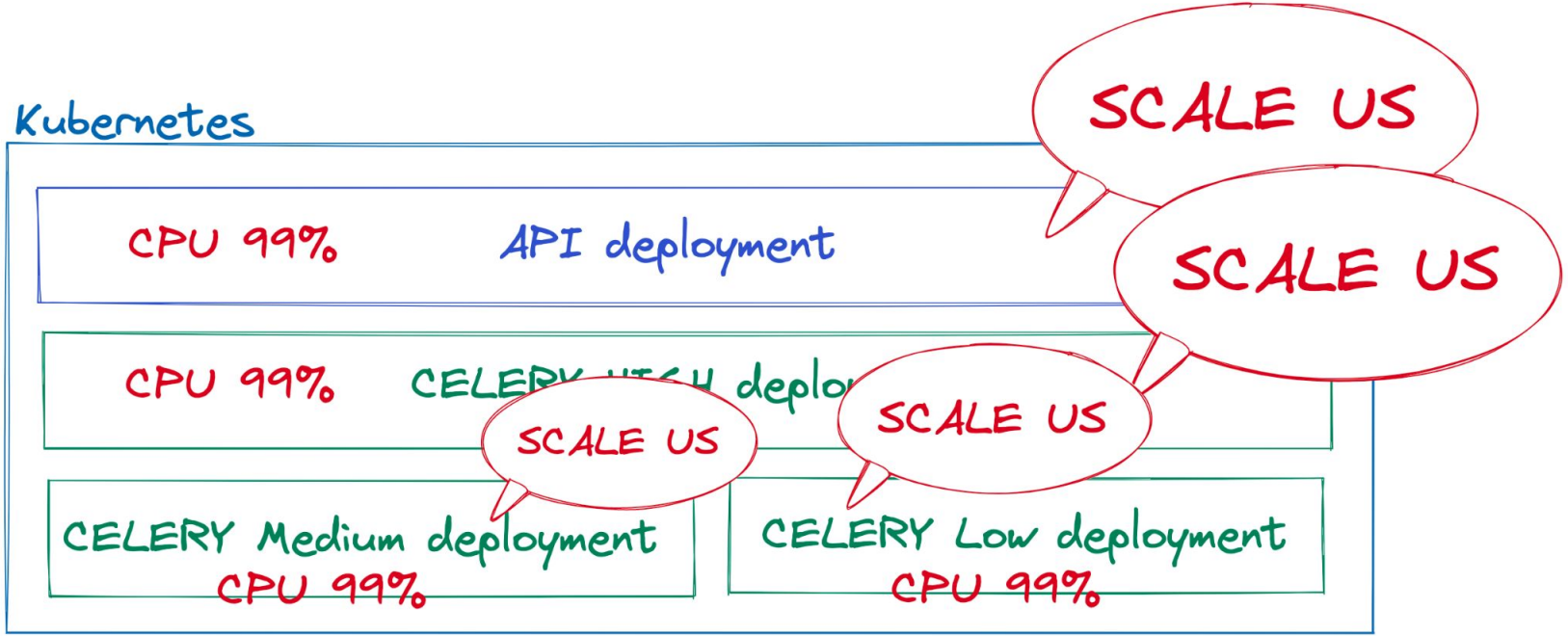
With connection pooling

Lessons

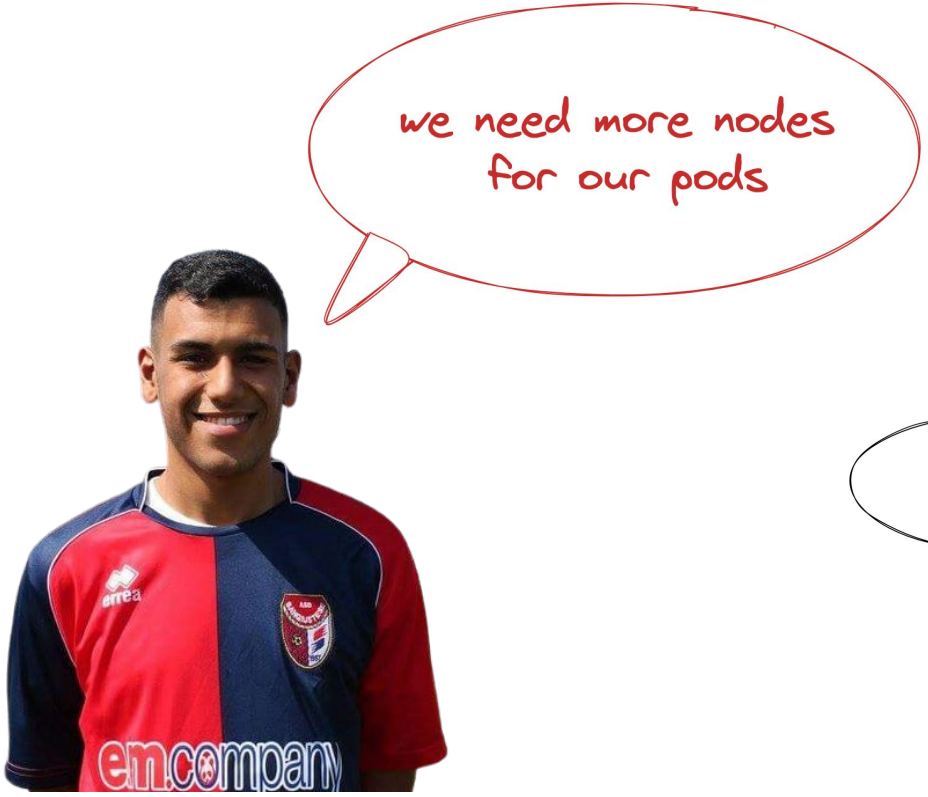
- Use a Connection Pooler
- Monitor the connection Pooler
- Experiment with the pool-size

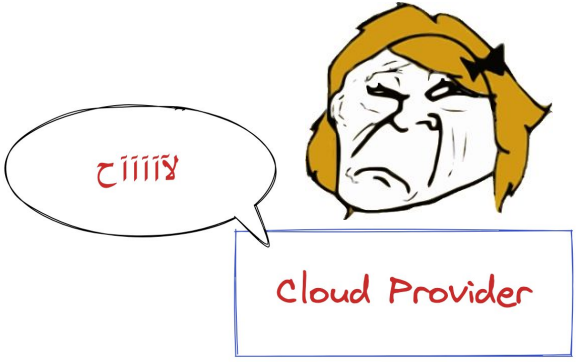
المشهد الخامس

We swarm again, and this happens...



We wanna scale up our deployments to handle more BUT





Lessons

- Trust Elasticity but verify
  but verify your quotas.

Lessons

gunicorn.conf.py | views.py | models.py | todos/urls.py

```python
from os import environ

bind = '0.0.0.0:' + environ.get('PORT', '5000')
workers = environ.get('WORKERS', 8)
loglevel = 'info'
graceful_timeout = 300
timeout = environ.get('GUNICORN_TIMEOUT', 120)
backlog = environ.get('BACKLOG', 200)
```

- Externalize configuration
- Understand your workload (IO vs Computation heavy)

Demo

Outro.

مسك الختام، لنتذكر

- Monitor.
- Use Managed services(P)
- Don't settle for the defaults(P)
- Don't take Elasticity for granted.
- Externalize configuration.
- Trust, But load test.
- Be Pragmatic.
- Take Notes and improve.

شكراً

□□□□□□□