There's no magic...

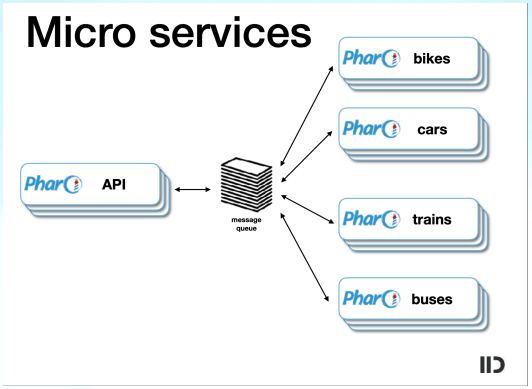
... until you talk about databases

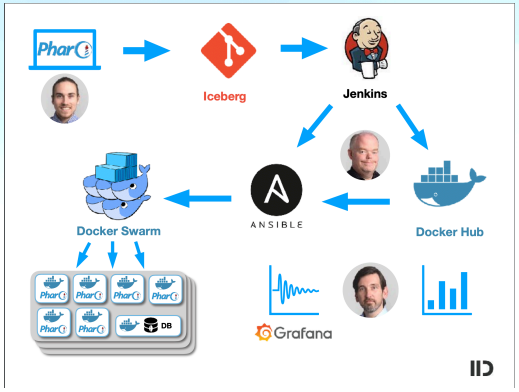
Norbert Hartl ESUG 2022

"An important thing you need to know about a rule is when you should break it"

(Norbert Hartl, ESUG 2022)

Recap: ESUG 2018





tech stack


mongoDB.
docker
ANSIBLE
RabbitMQ
HAPROXY
elasticsearch
Grafana

It
has
grown
...

It
has
grown
small

Three things that can kill a project

Three things that can kill a project

1. complexity

Three things that can kill a project

1. complexity
2. complexity

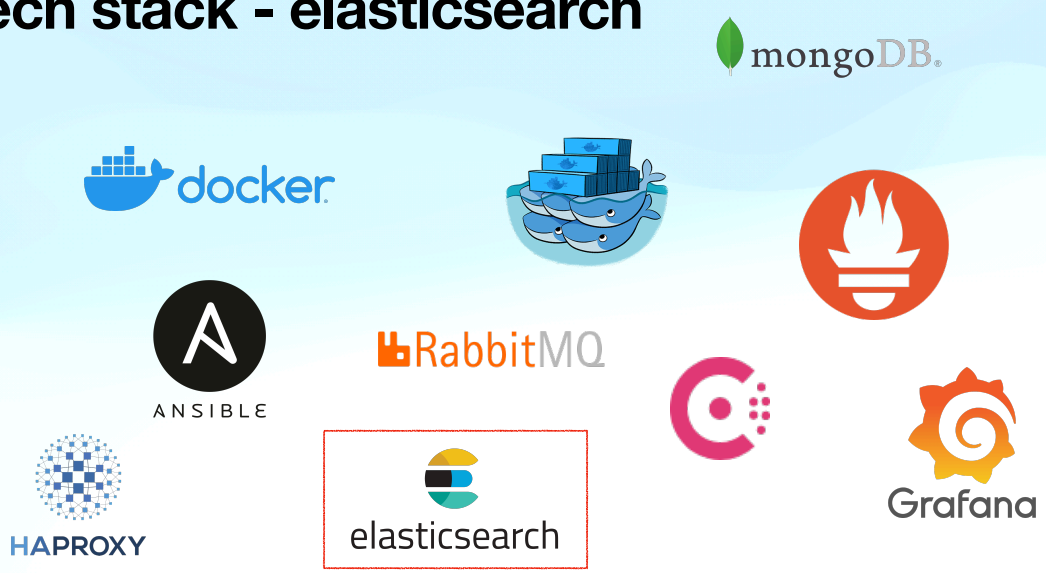
Three things that can kill a project

1. complexity
2. complexity
3. javascript

tech stack


mongoDB.
docker
ANSIBLE
RabbitMQ
HAPROXY
elasticsearch
Grafana

tech stack - elasticsearch



Events and aggregation

Everyone likes dashboards

Events and aggregation

Everyone likes dashboards

- avoid

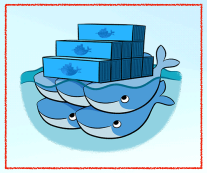
Events and aggregation

Everyone likes dashboards

- avoid
- postpone

tech stack - micro services

tech stack - orchestration



mongoDB.

docker

ANSIBLE

RabbitMQ

HAPROXY

elasticsearch

Grafana

Docker swarm & Kubernetes

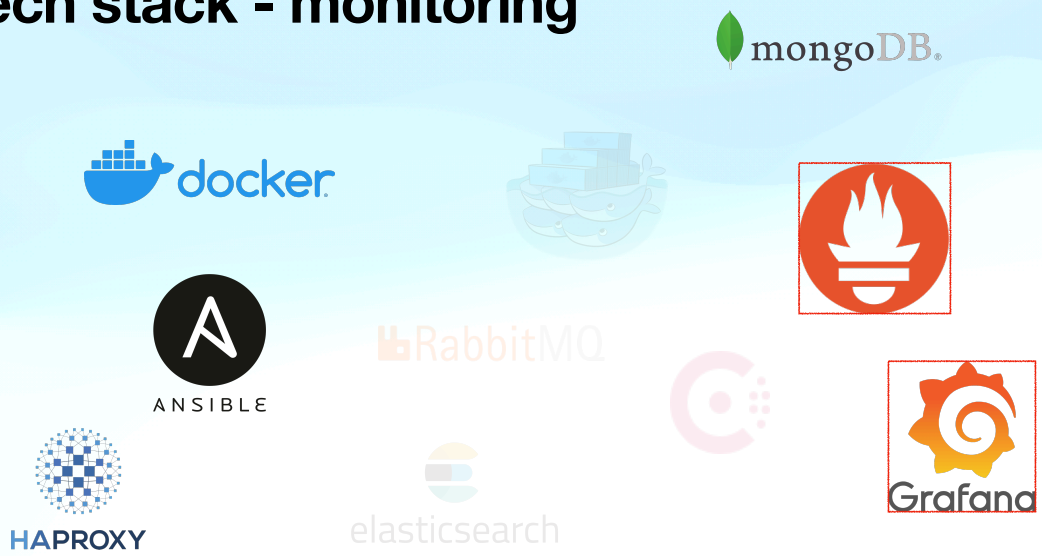
There is only one advize

Docker swarm & Kubernetes

There is only one advize

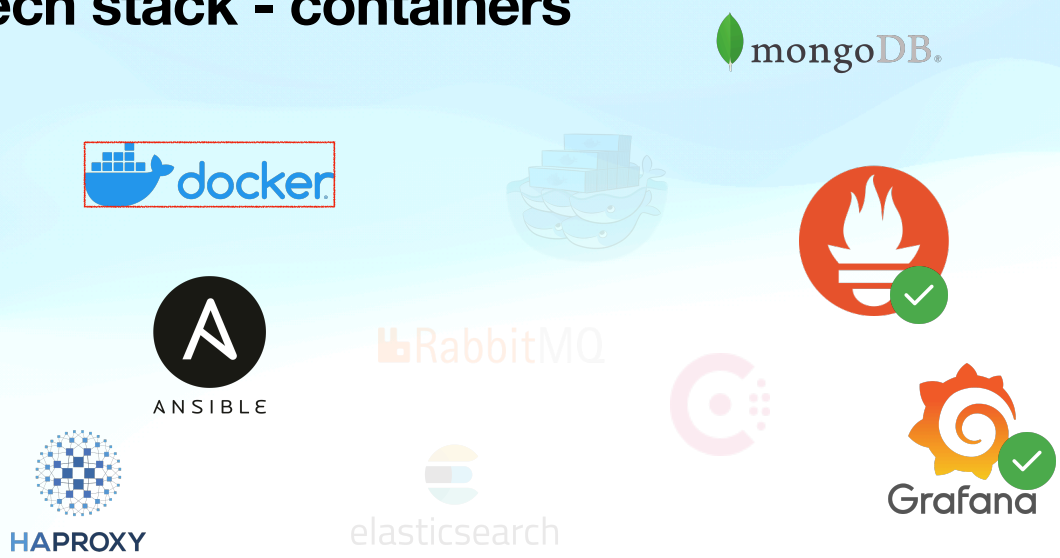
- Don't

tech stack - monitoring

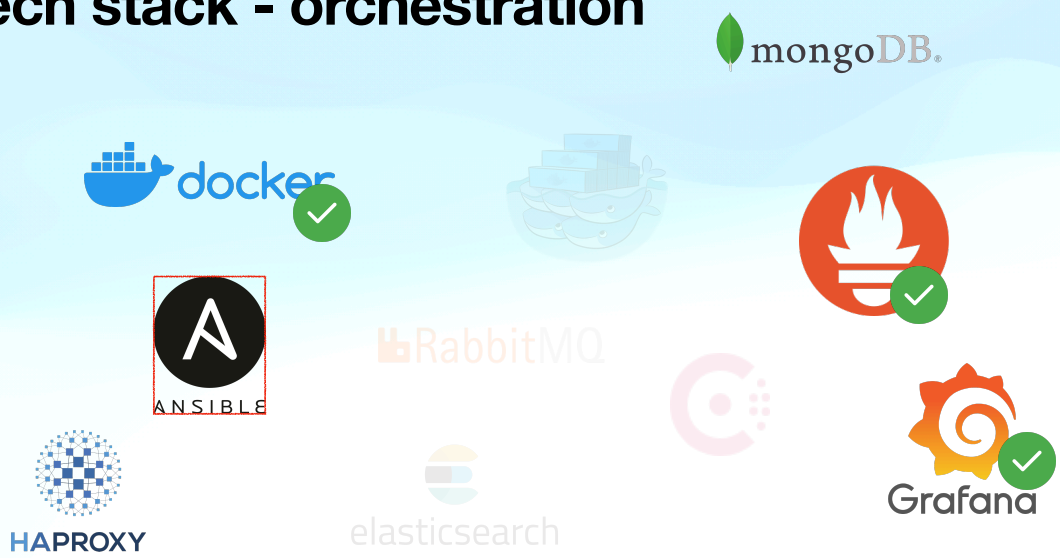


„If you have a service that is not monitored you don’t have a

tech stack - containers



tech stack - orchestration



Ansible cheat shee

inventory

```
[api-group]
apptive1
apptive2
apptive3
```

host_vars

```
hostname: apptive1
internal_ip: 10.1.2.5
apptive_api_ports:
  - 3600
  - 3601
```

play

```
- hosts: api-group
  roles:
   - apptivegrid-api
```

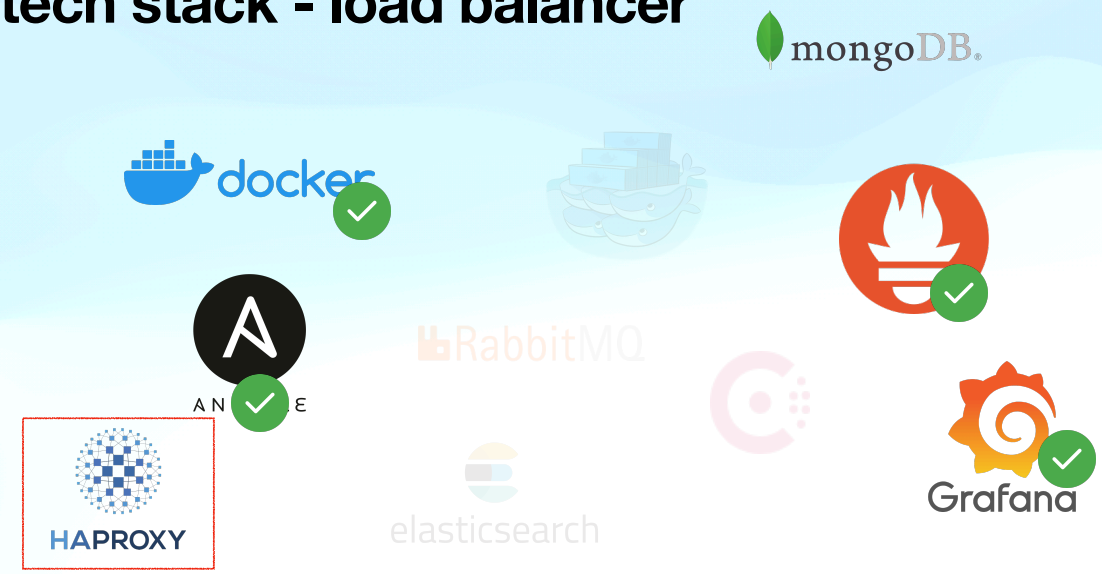
apptivegrid-api role

```
- name: Deploy apptivegrid API
  community.docker.docker_container:
    name: "apptive-api-{{ item.0+1 }}"
    image: apptivegrid-api:
{{ apptivegrid_api_version }}
    ports:
      - "{{internal_ip}}:{{ item.1 }}:3600"
    volumes:
```

haproxy role

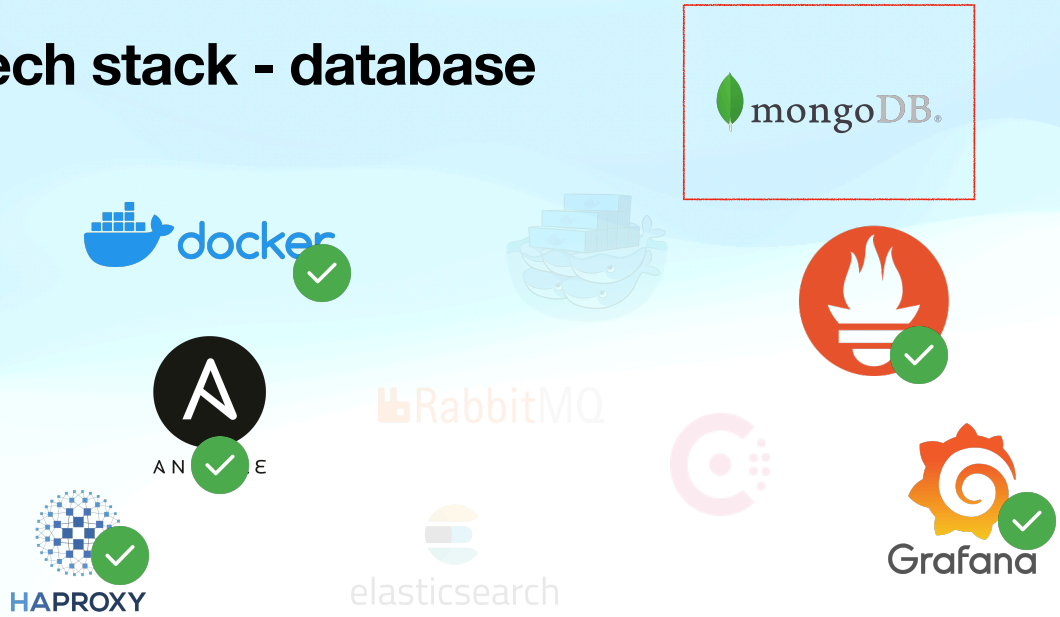
```
backend apptivegrid-api-backend
  balance leastconn
{% for apihost in groups[,api-group'] %}
{% for port in
hostvars[apihost].apptive_api_ports %}
  server api-…-{{port}} {{ ….internal_ip }}:
{{ port }} check
{% endfor %}
{% endfor %}
```

tech stack - load balancer

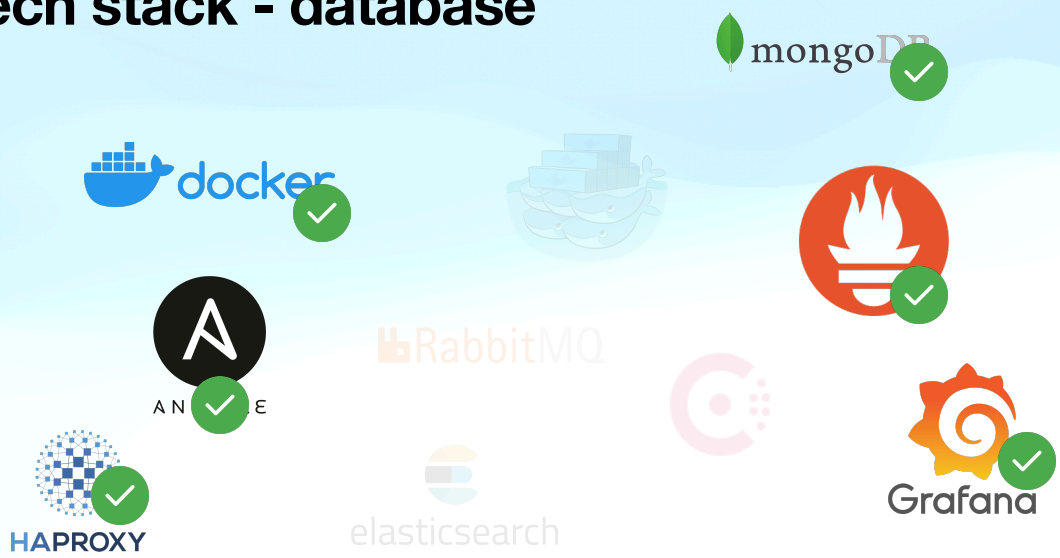


tech stack

tech stack - database

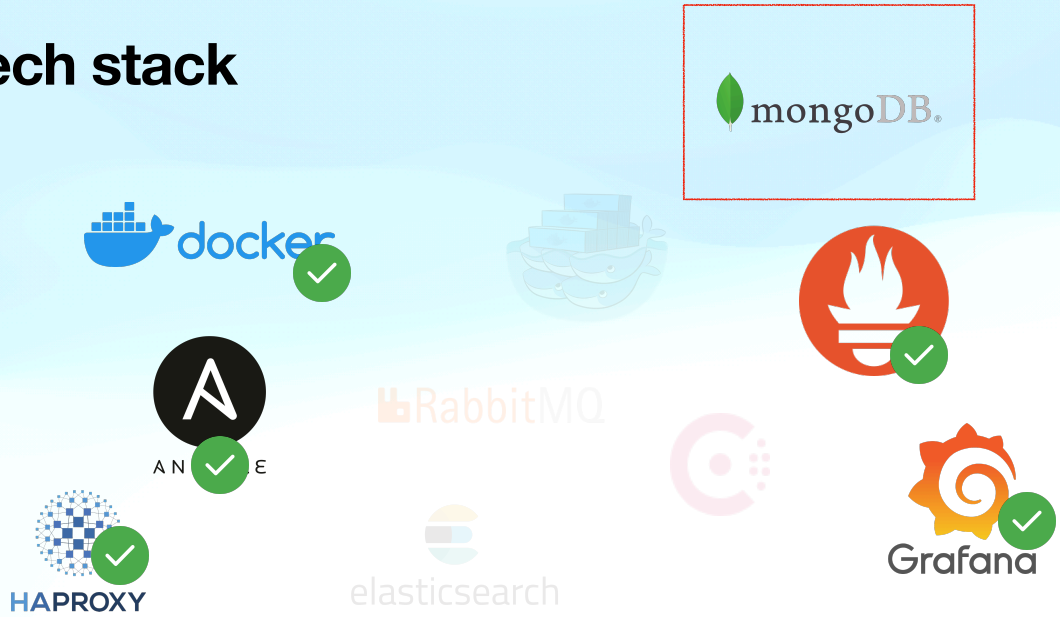


tech stack - database



wait...

tech stack



Mongo DB

The good parts

- simple document storage
- provides database cluster
- supposed to be web scale
- we have voyage for it

Mongo DB

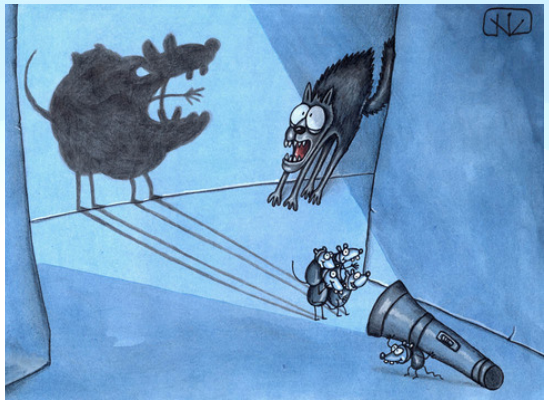
The not-so-good parts

- JSON supports 6 data types
- BSON supports a few more
- transactions are not part of mongo talk
- single writer vs. sharding
- query DSLs are a drag

Soil

What it needs to be an OO database?

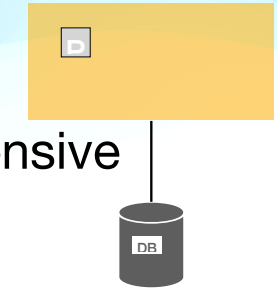
- ACID transaction (with MVCC)
- Regional file locking (row-level locking)
- serialization/materialization
- A b-tree implementation for indexing
- 100% smalltalk



How do we scale that?

Escaping the single machine

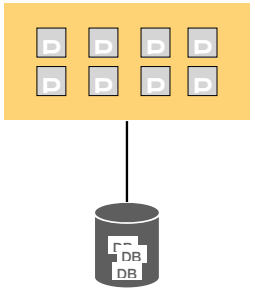
- Files are local on a machine
- Opening databases per request is expensive
- File locking enables multi-image usage
- How to scale to more than one machine?



Distribute the database

Escape step #1

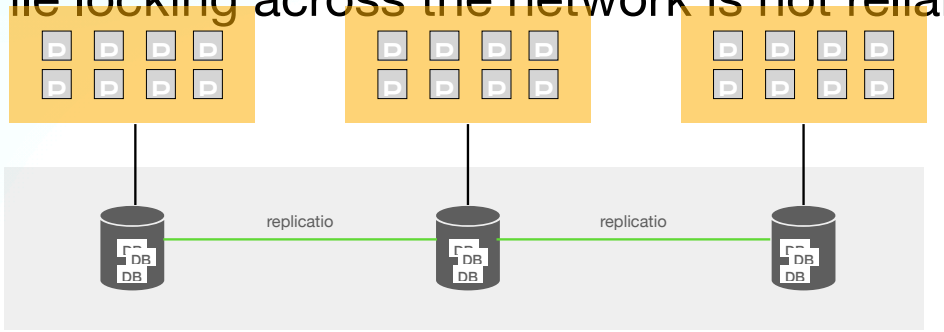
- Reduce conflict potential
- Partition the model
- Each user has its own database on disk (4kb)



Distribute the database

Escape step #2

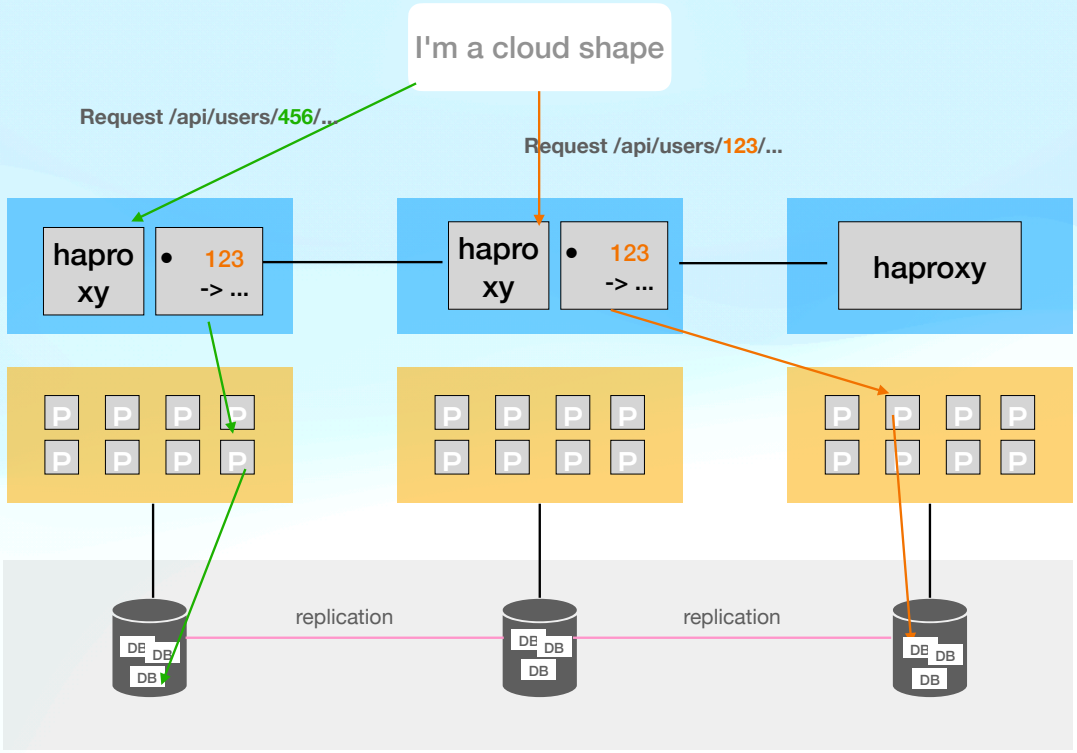
- Use a distributed filesystem (GlusterFS)
- Enables multi machine setup
- File locking across the network is not reliable



Distribute the database

Escape step #3

- stateless service
- URI contains partition criteria (/api/users/**74827492**/...)
- stick on path,word(3,/) if { path_beg /api/users/ }
- each request to the same database goes to the same image


I'm a cloud shape
Request /api/users/456/...
Request /api/users/123/...
hapro xy
123 -> ...
hapro xy
123 -> ...
haproxy
P
DB
replication
replication

Escape summary

The complete plan

- Persistence approaches are application specific
- Architecture can provide performance/scalability
- Writing local files does not need to be a blocker
- Pinning writes to one place solves a lot