Deploying Django web applications in Docker containers

Jamie Hewland

PyConZA 2017

PRAEKELT.ORG

00

Hi 👋, I’m Jamie

SRE at Praekelt.org

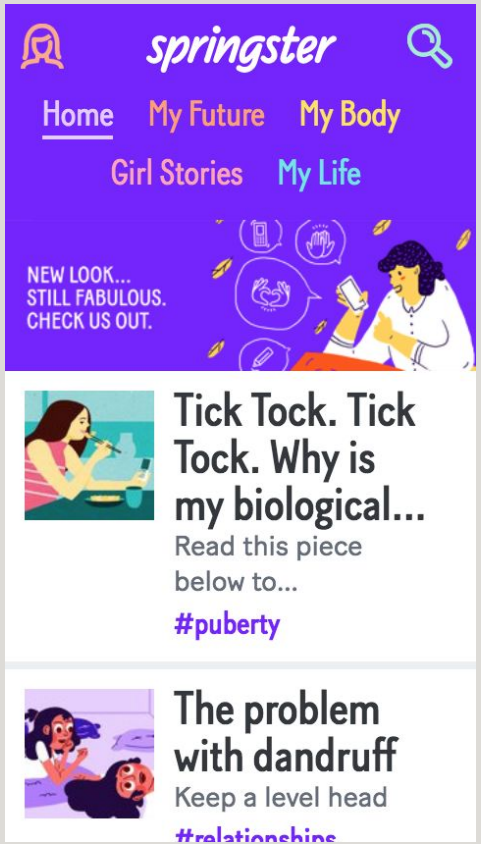
@JayH5

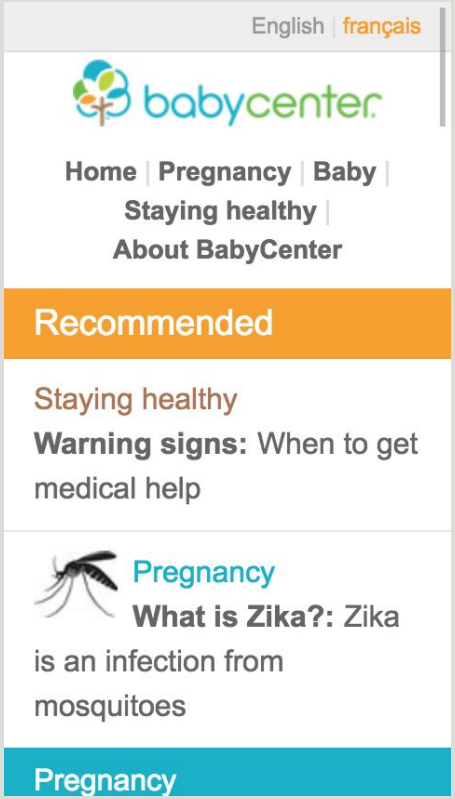
@jayhewland

/jhewland

We moved to containers because we needed to deploy a lot of sites quickly

...and manually managing Python processes across many servers wasn’t scaling.





PRAEKELT
ORG

Deploying Django web applications in *Docker containers*

Specifically, Docker containers to run under a *container orchestration* system.

- **What are Docker containers?***
- **What is container orchestration?***
- **How is Django typically deployed?**
- **How do I deploy Django in a Docker container?**

***Very roughly speaking**

01

Containers & container orchestration

It’s 🔥🔥🔥

What’s a (Linux) container?

Isolation of a process’ <u>view</u> of its operating environment (via *namespaces*)

- Process trees
- User IDs
- Networking
- Mounted file systems...

Limitation/prioritization of resources (via *`cgroups`*)

- CPU
- Memory
- Block I/O
- Networking...

Docker containers 🐳

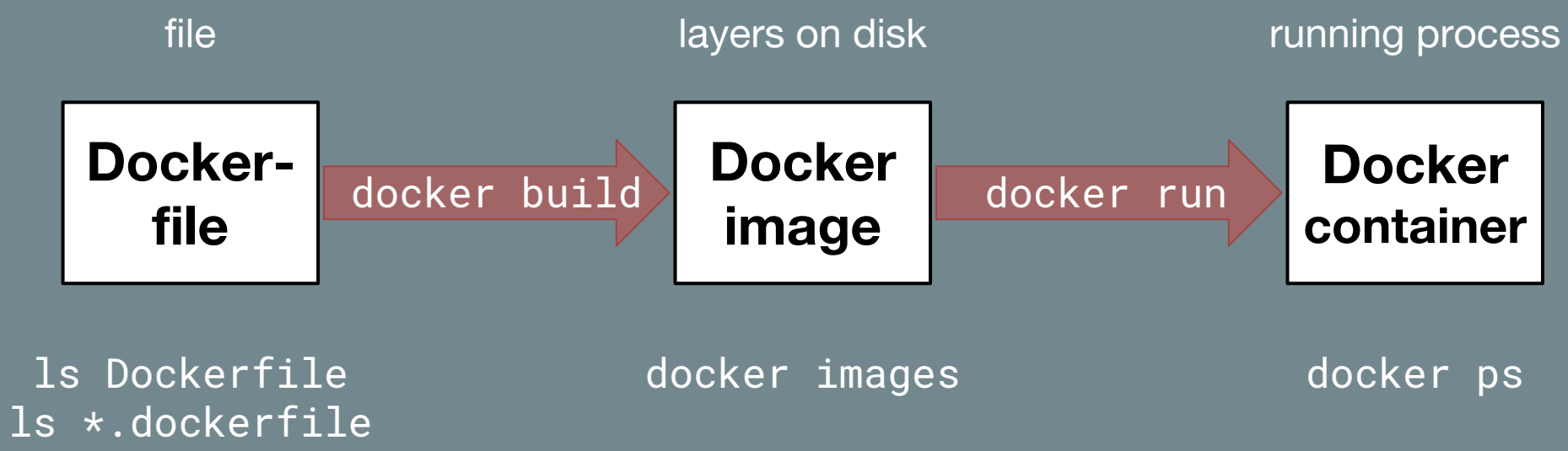
Docker is the most popular container technology.

- Batteries included
- Easy-to-use, lots of sensible defaults
- *“Copy-on-write union filesystem”*: containers can start up very quickly and share a lot of image data
- Images available for all the software you know & love

```
> $ docker run
```

ubuntu
debian
redis
rabbitmq
python
postgres
sentry
nginx
nodejs

Docker terminology



Terms *"image"* and *"container"* often conflated

Why containers?

Consistent portability

- A clean way to package software
- With (almost) everything it needs to run
- With a single, simple entry-point
- Limit access to resources
- Eliminates “but it works on *my* machine”

Container orchestration

“Container Orchestration refers to the automated arrangement, coordination, and management of software containers.”

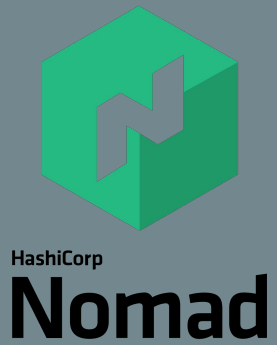
Container Orchestration with Kubernetes: An Overview - Adrian Chifor
http://bit.ly/2takgmd

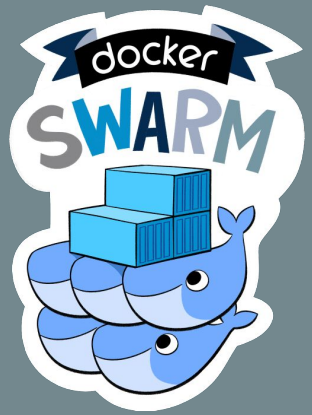
Container orchestration

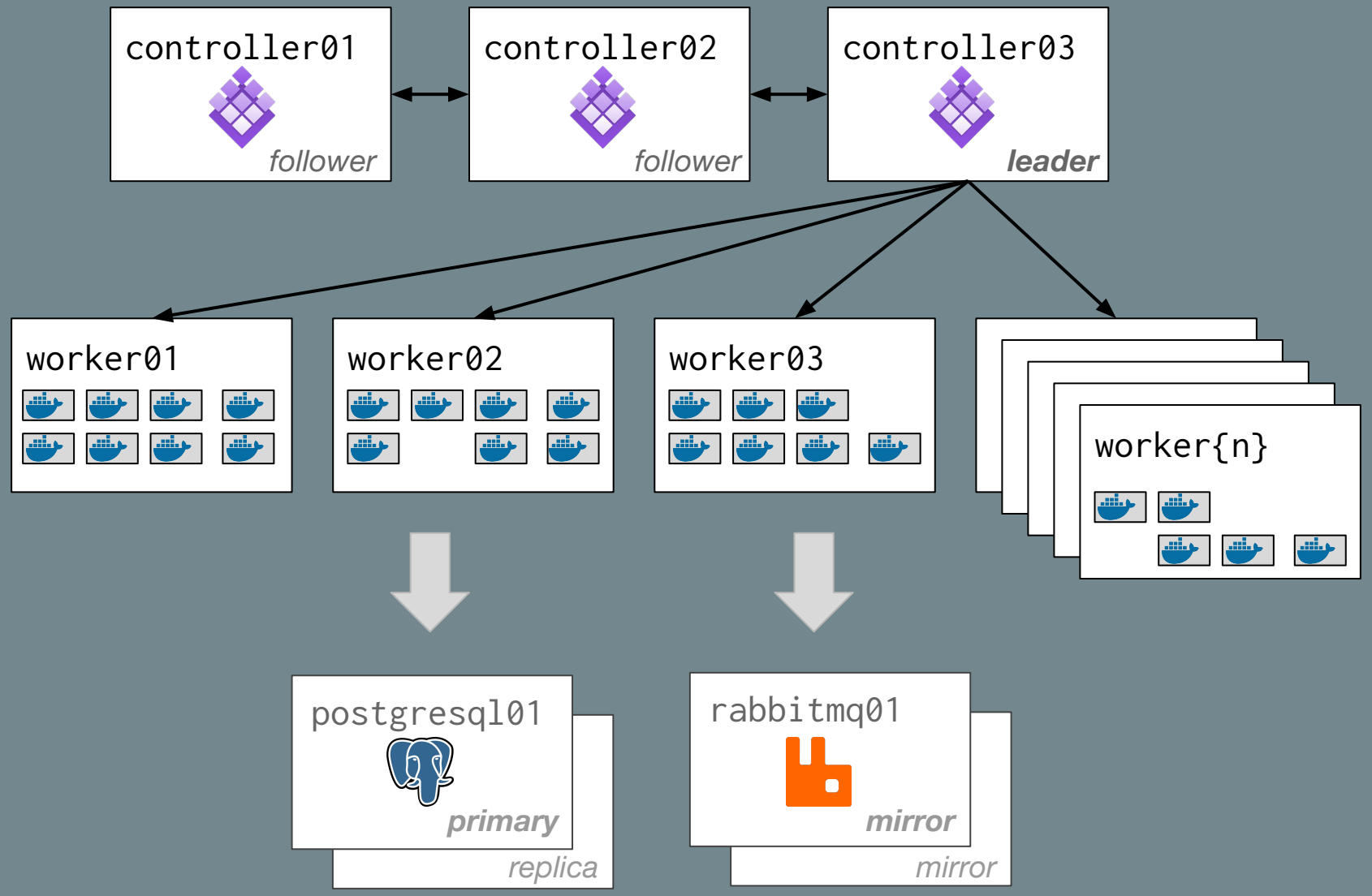
Achieved through a variety of services:

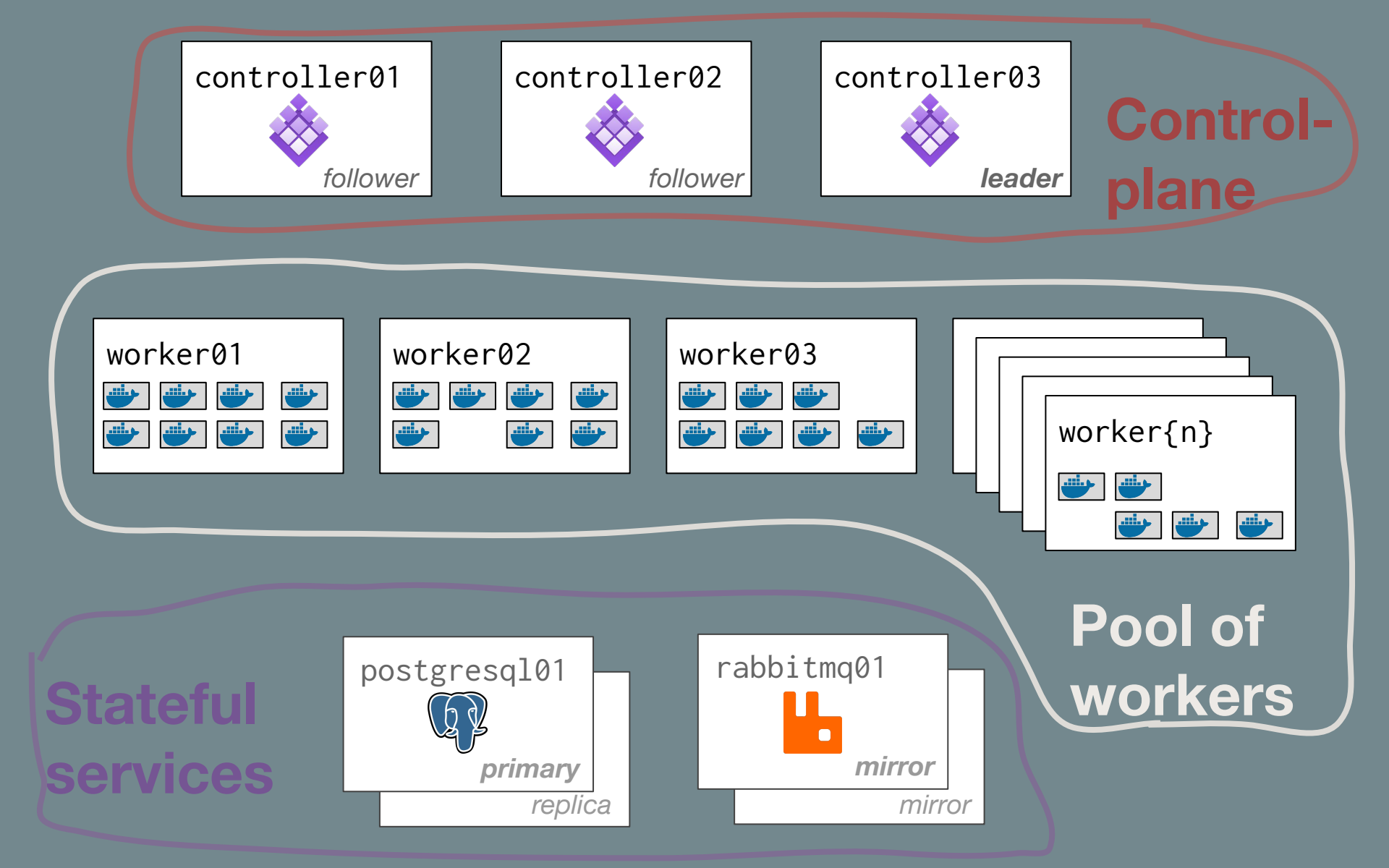
- Service discovery
- Load balancing
- Health checks
- Resource management
- More...

Container orchestration is 🔥🔥🔥



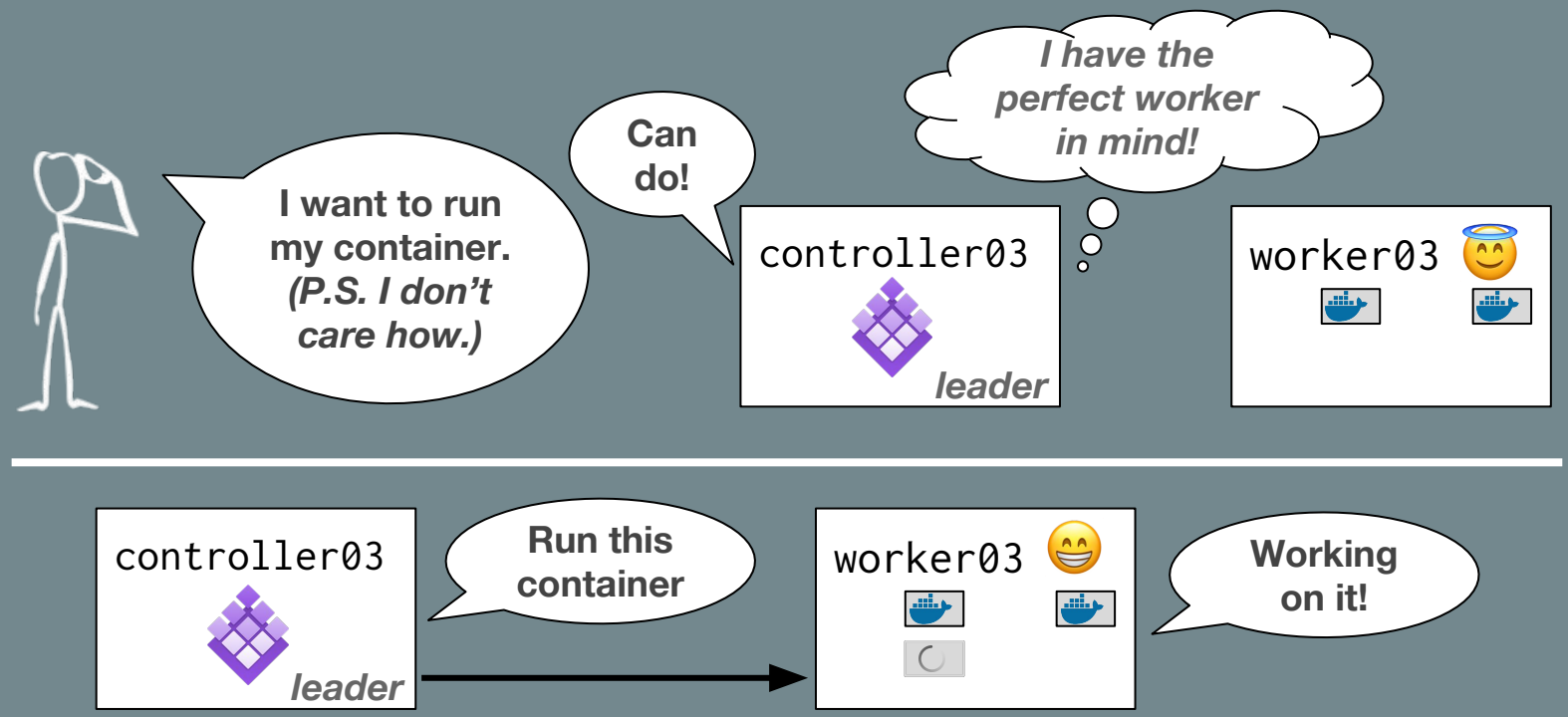



controller01
follower
controller02
follower
controller03
leader
worker01
worker02
worker03
worker{n}
postgresql01
primary
replica
rabbitmq01
mirror
mirror


controller01
follower
controller02
follower
controller03
leader
Control-plane
worker01
worker02
worker03
worker{n}
Pool of workers
Stateful services
postgresql01
primary
replica
rabbitmq01
mirror
mirror

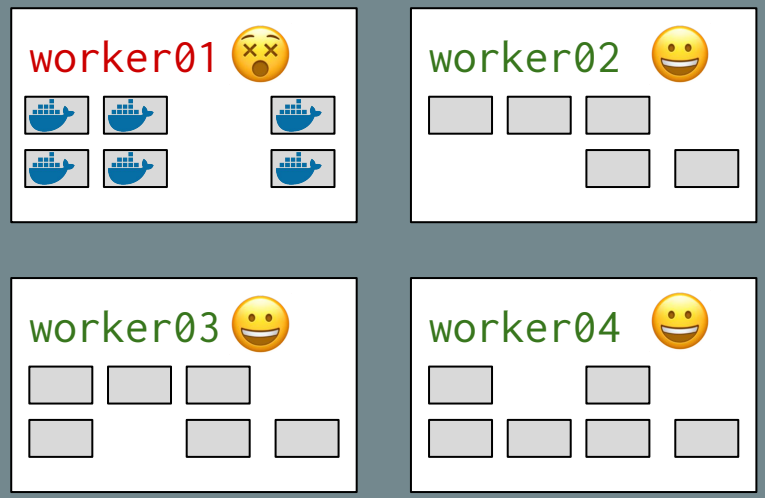
Advantage 1: Deployments

Don’t need to choose how/where to run apps

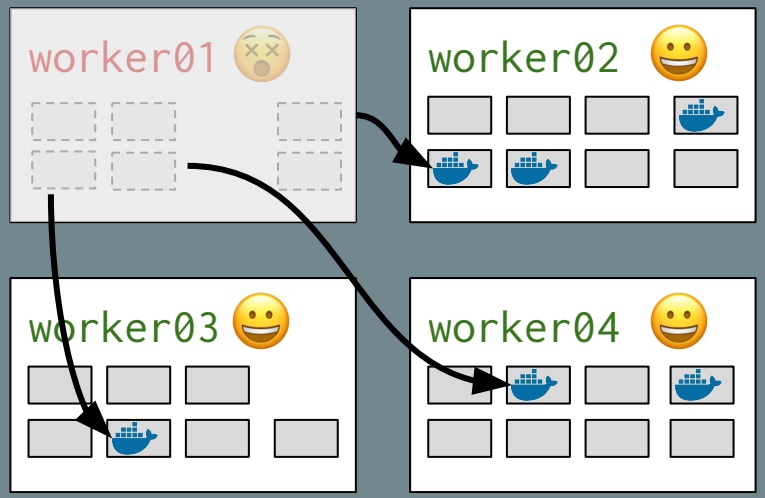


Advantage 2: Failover

Containers get moved off unhealthy worker hosts



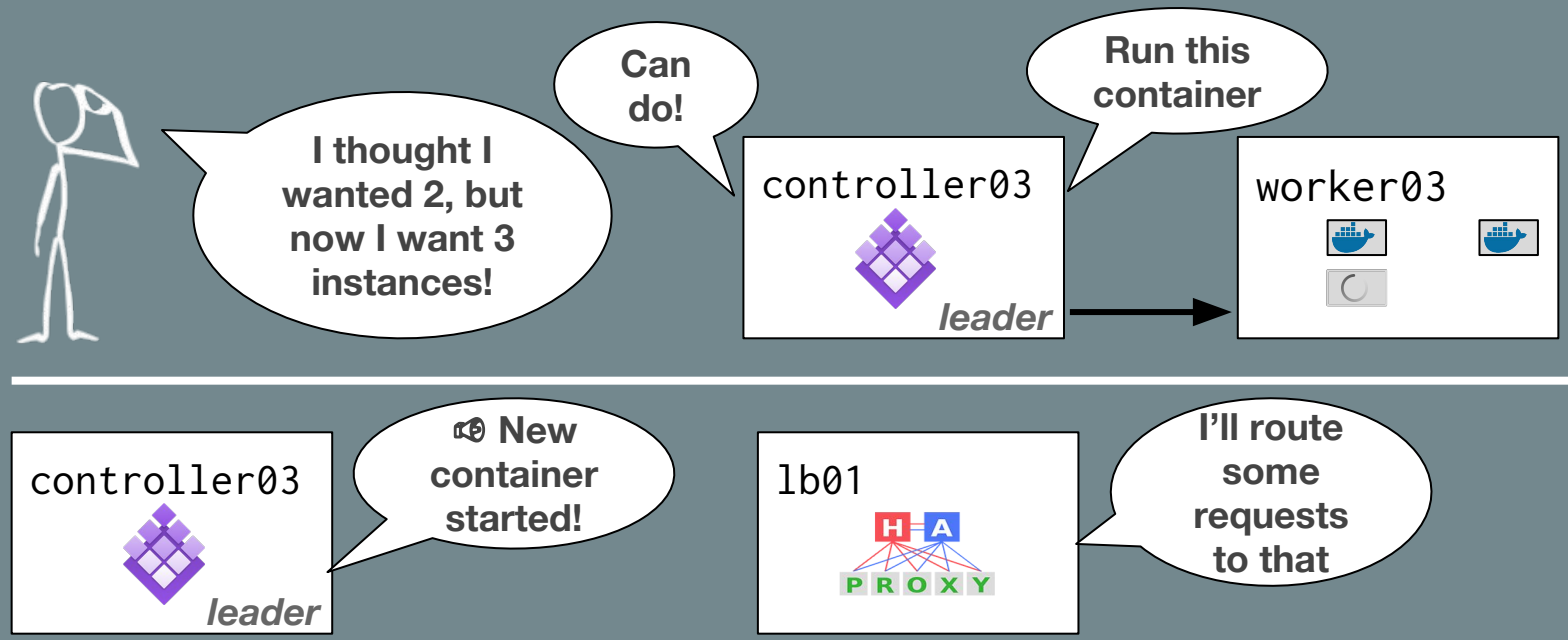
1. `worker01` is unhealthy



2. Containers migrated off `worker01`

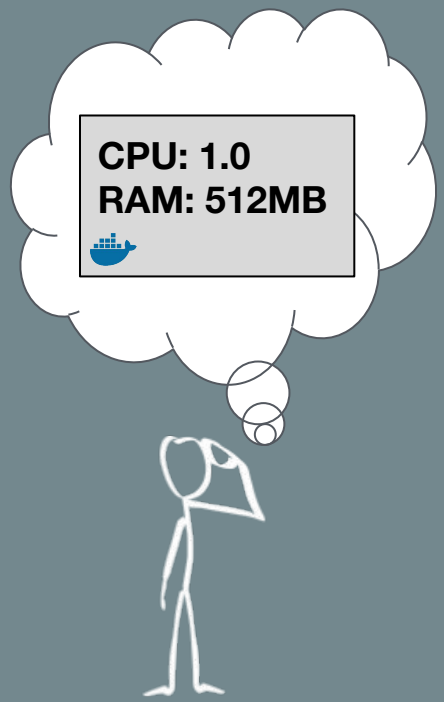
Advantage 3: Scaling

Can scale the number of running containers



Advantage 4: Resource utilisation

Orchestration systems can pack containers efficiently



worker03

- CPU: 0.4, RAM: 512MB
- CPU: 1.0, RAM: 768MB
- CPU: 0.5, RAM: 128MB

CPU: 1.9/3.0
RAM: 1.375/2.0GB

worker05

- CPU: 1.0, RAM: 768MB
- CPU: 1.0, RAM: 512MB
- CPU: 0.5, RAM: 256MB

CPU: 2.5/3.0
RAM: 1.5/2.0GB

02

Deploying Django web applications

“Webservers”

django

- Open source web framework
- Very popular
- Roughly model-view-controller (MVC)
- Featureful: Caching, i18n, middleware…
- Lots of 3rd party extensions

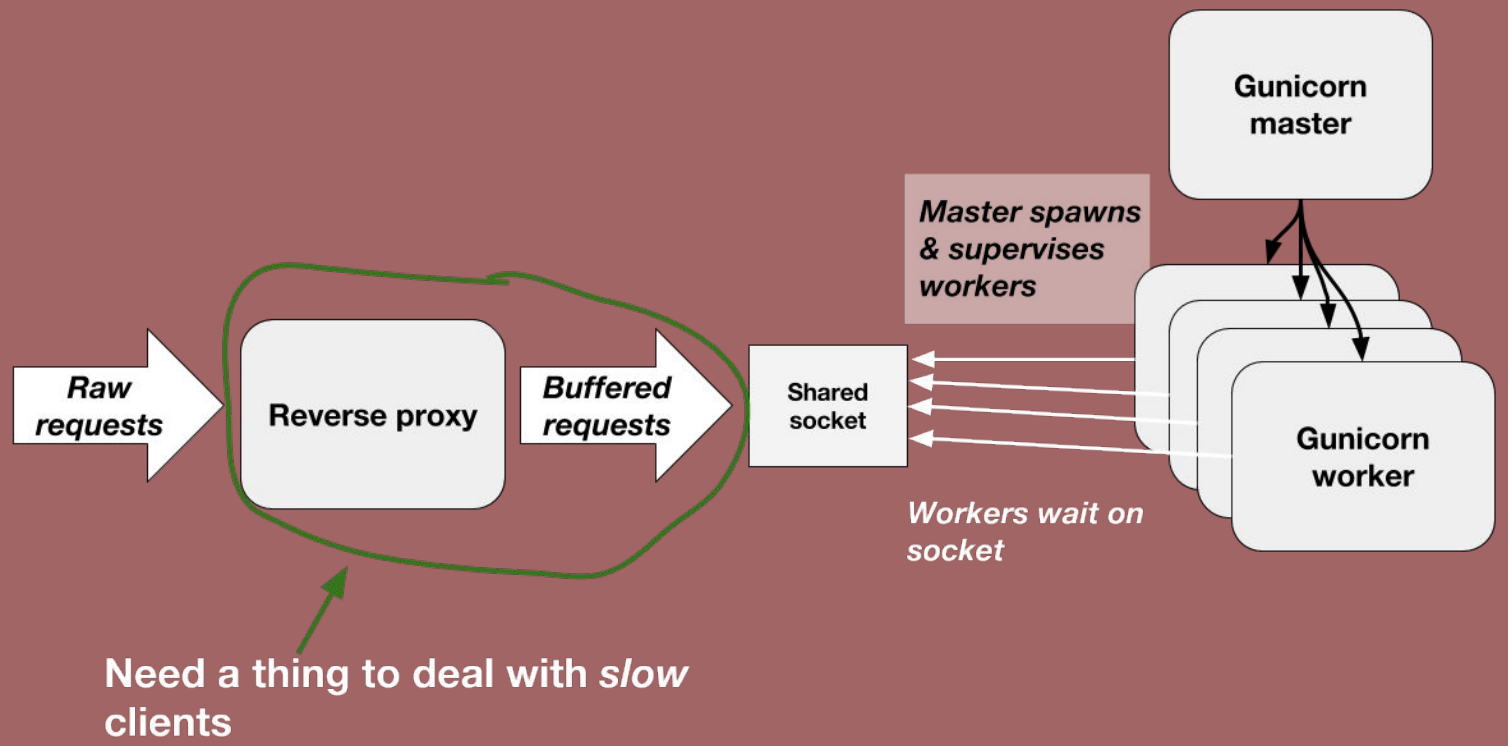
Running Django

- Django applications typically served using a WSGI server
- Web Server Gateway Interface (WSGI): PEP 3333
- Two sides: *“server”/“gateway”* and *“application”/“framework”*
- Server calls application once per request with data from request

- **Gunicorn is a WSGI server**
- **Based on *Unicorn* (Ruby software)**
- **Pre-fork worker model:**
  - One master process spawns one or more worker processes
  - Master terminates workers if they take too long
  - Workers = 2-4 x number of cores
- **<u>Designed to only serve fast clients*</u>**

With its default worker implementation

How it fits together

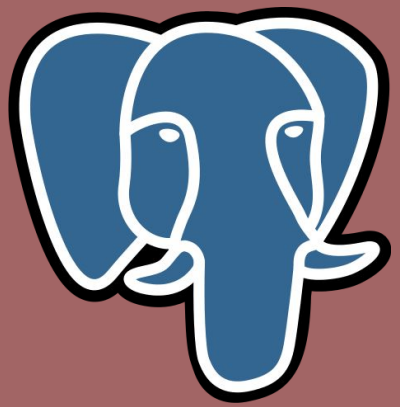


NGINX

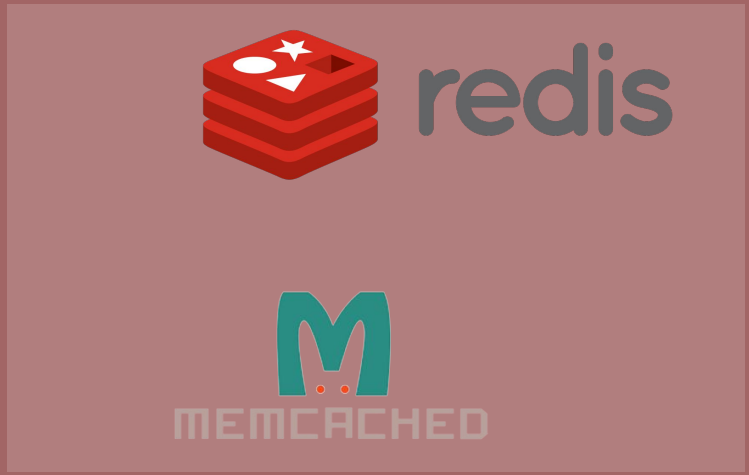
- **Reverse proxy most often used with Gunicorn is *Nginx***
- **Very fast and battle-tested: “shields” our Python code from the outside**
- **Can use it to do other useful stuff:**
  - Serve static files (CSS, JS, fonts...)
  - Caching
  - *Compression, SSL, more...*

- Django under Gunicorn runs only in response to requests
- What about long-running and/or periodic tasks?
- Celery: Distributed Task Queue
- Integrates with Django
- *But now we need a message broker*

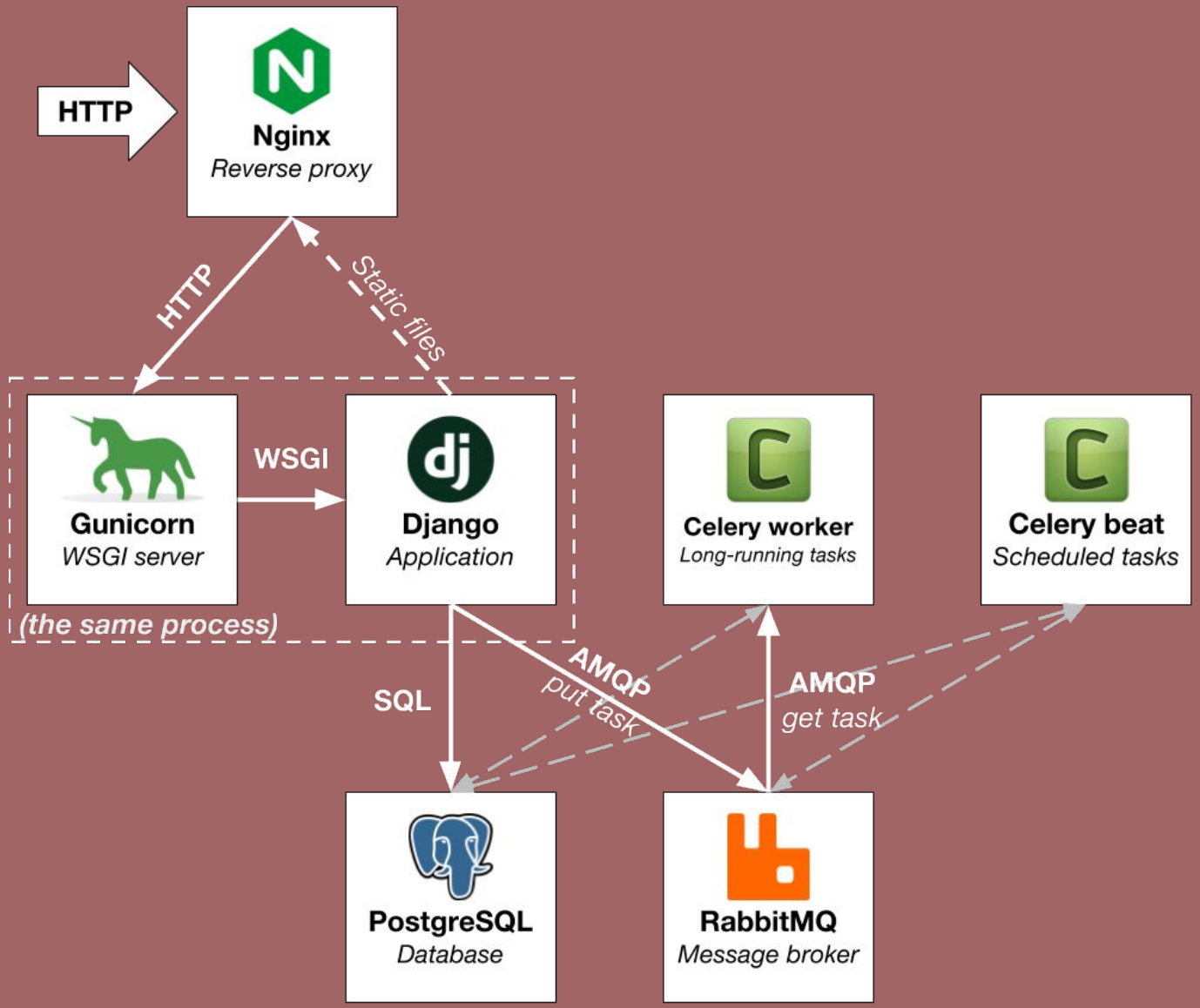
Other things needed for Django



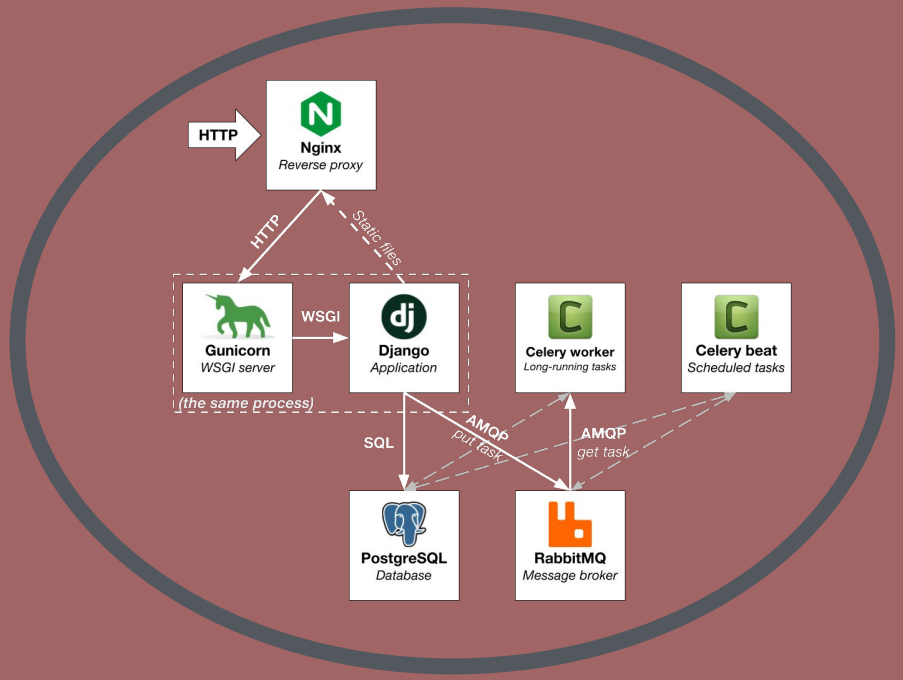
A database:
probably PostgreSQL

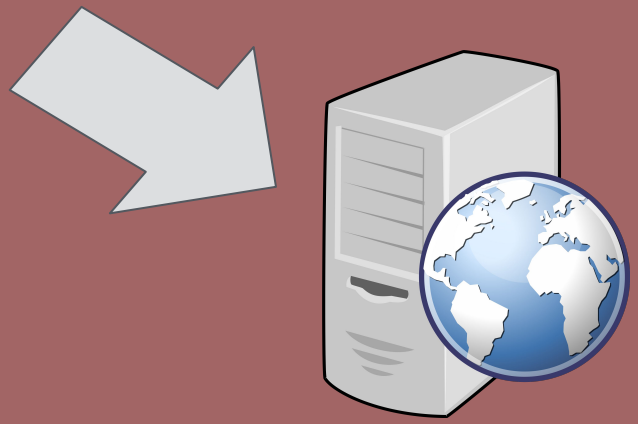


(Optional) caching:
Redis or Memcached


HTTP
Nginx
Reverse proxy
HTTP
Static files
Gunicorn
WSGI server
WSGI
Django
Application
(the same process)
Celery worker
Long-running tasks
Celery beat
Scheduled tasks
SQL
AMQP
put task
AMQP
get task
PostgreSQL
Database
RabbitMQ
Message broker

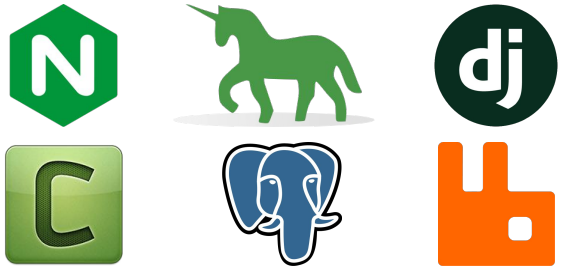
Take all of that and...



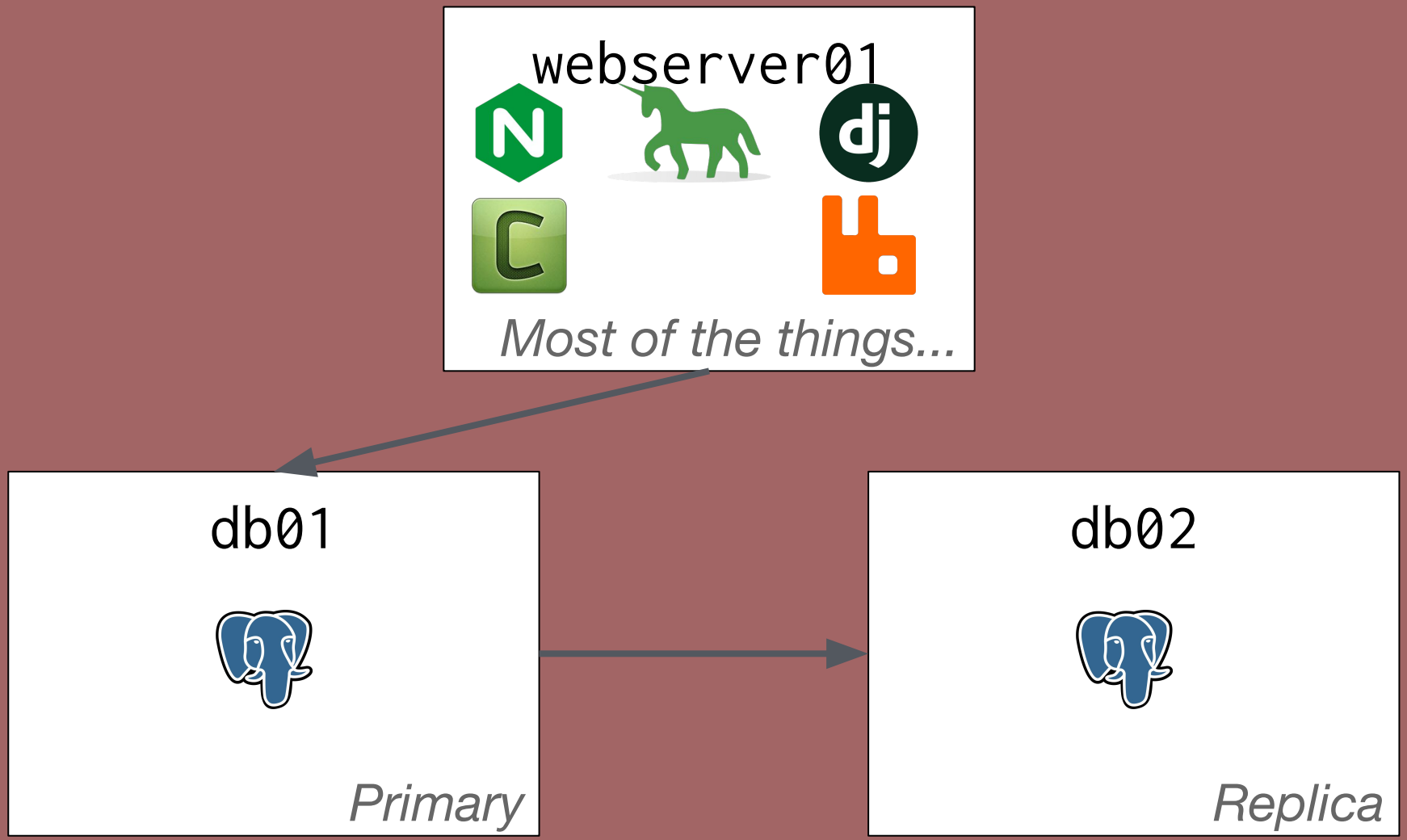


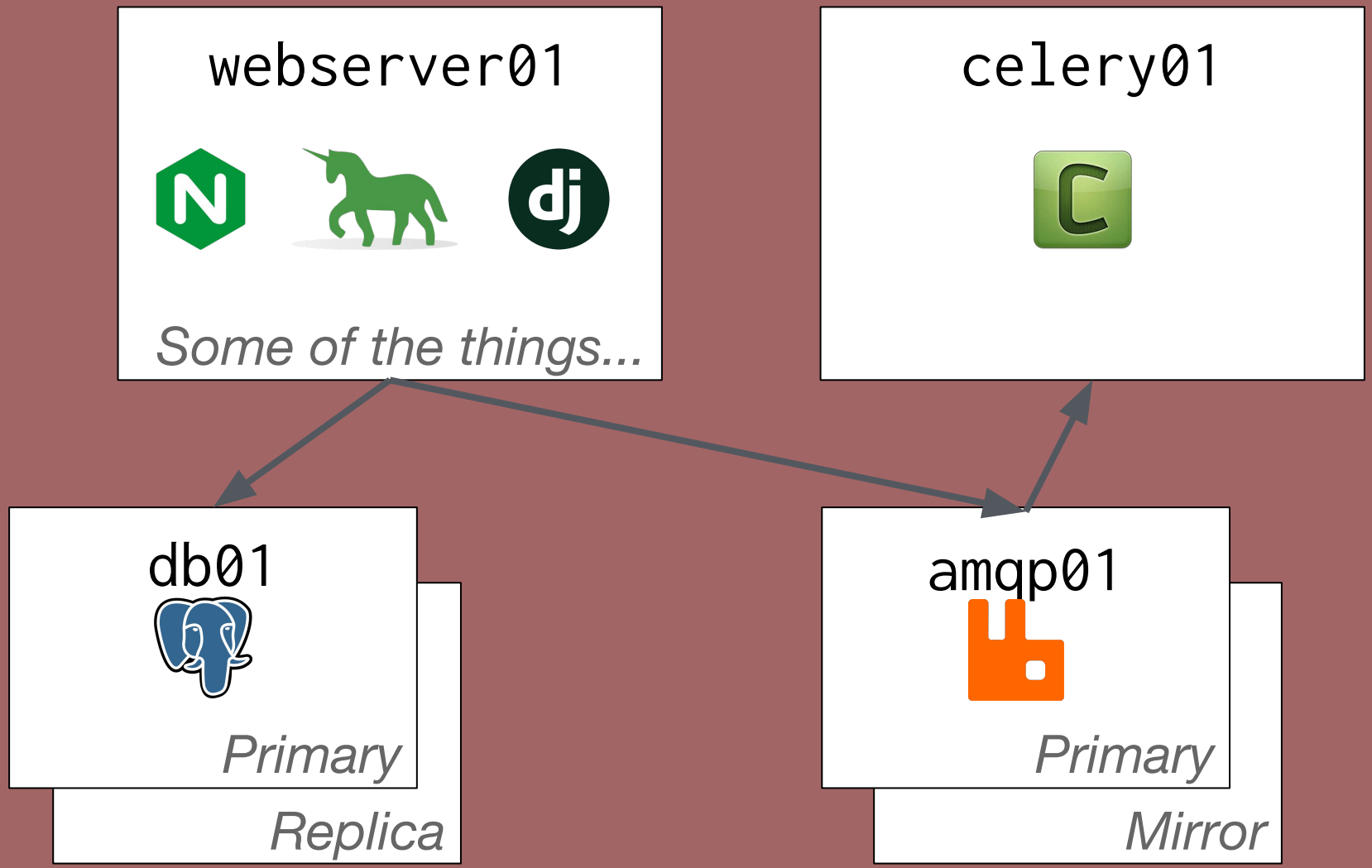
...just chuck it on a server

webserver01

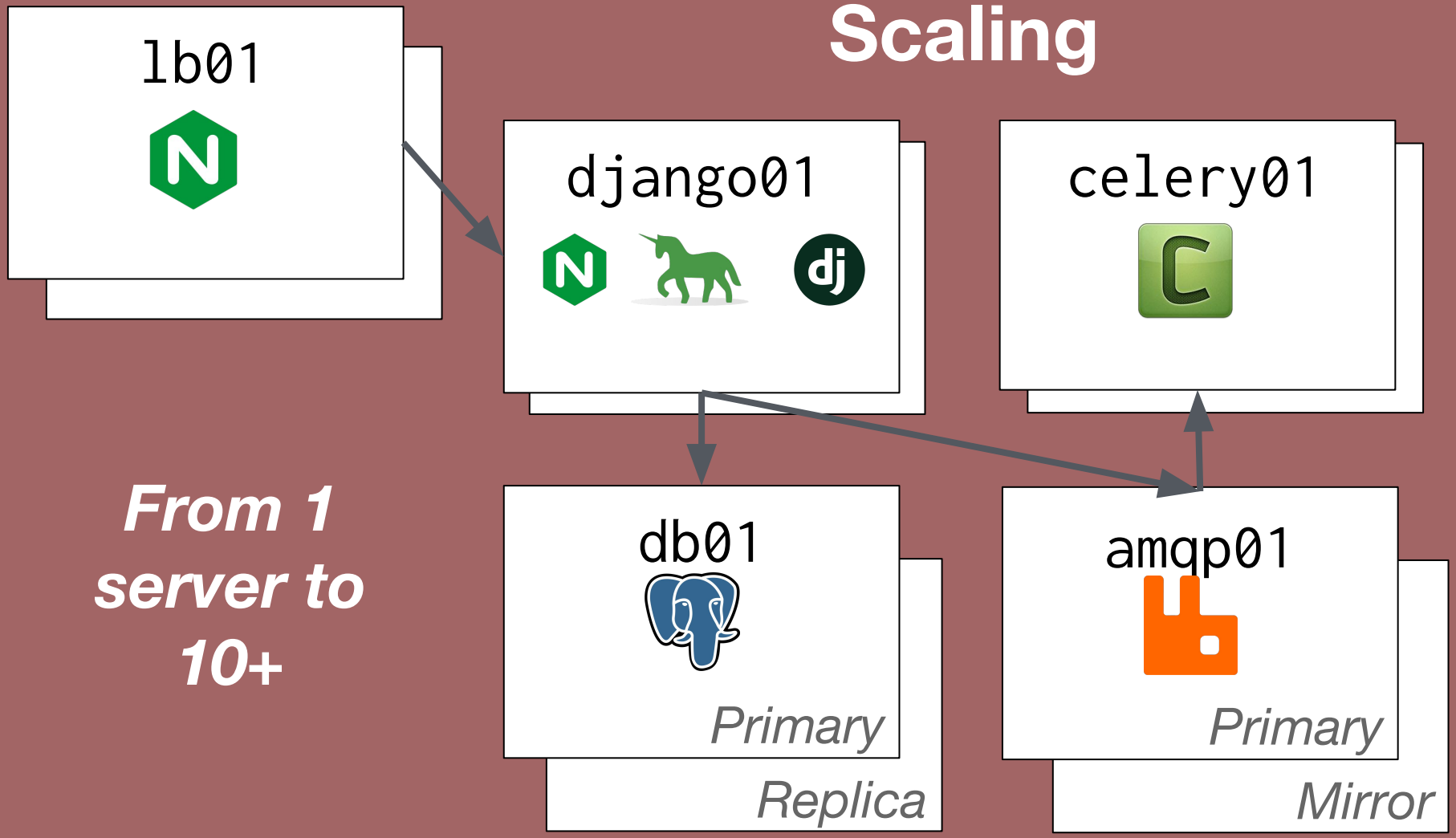


All the things...


webserver01
Most of the things...
db01
Primary
db02
Replica


webserver01
Some of the things...
celery01
db01
Primary
Replica
amqp01
Primary
Mirror

Scaling



From 1 server to 10+

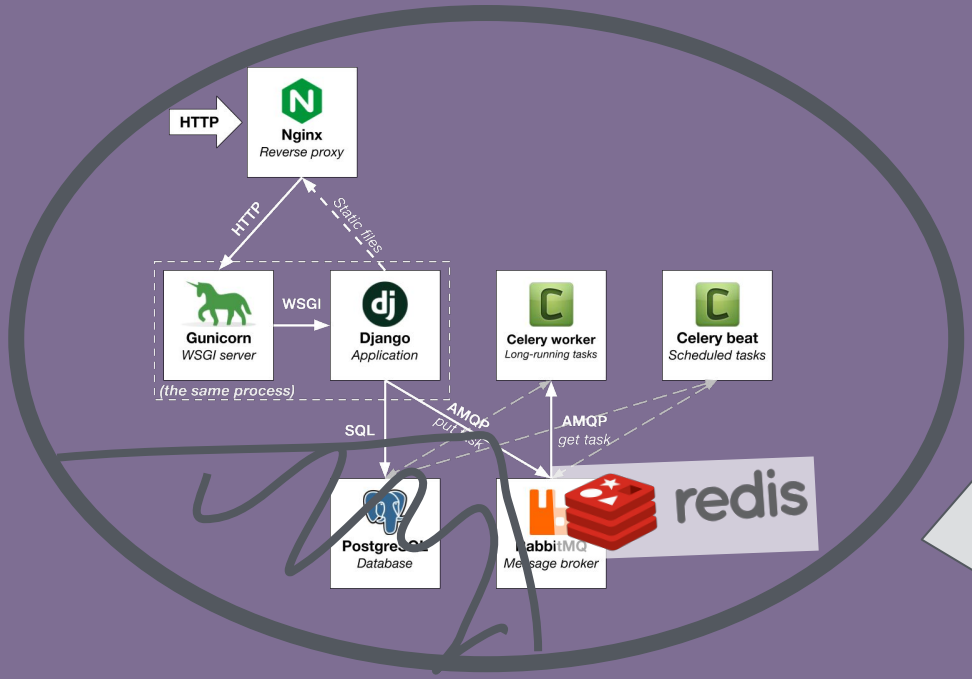
03

Containerizing Django

Making it fit

You’ll never guess what we tried first...

Took <u>most</u> of that and...

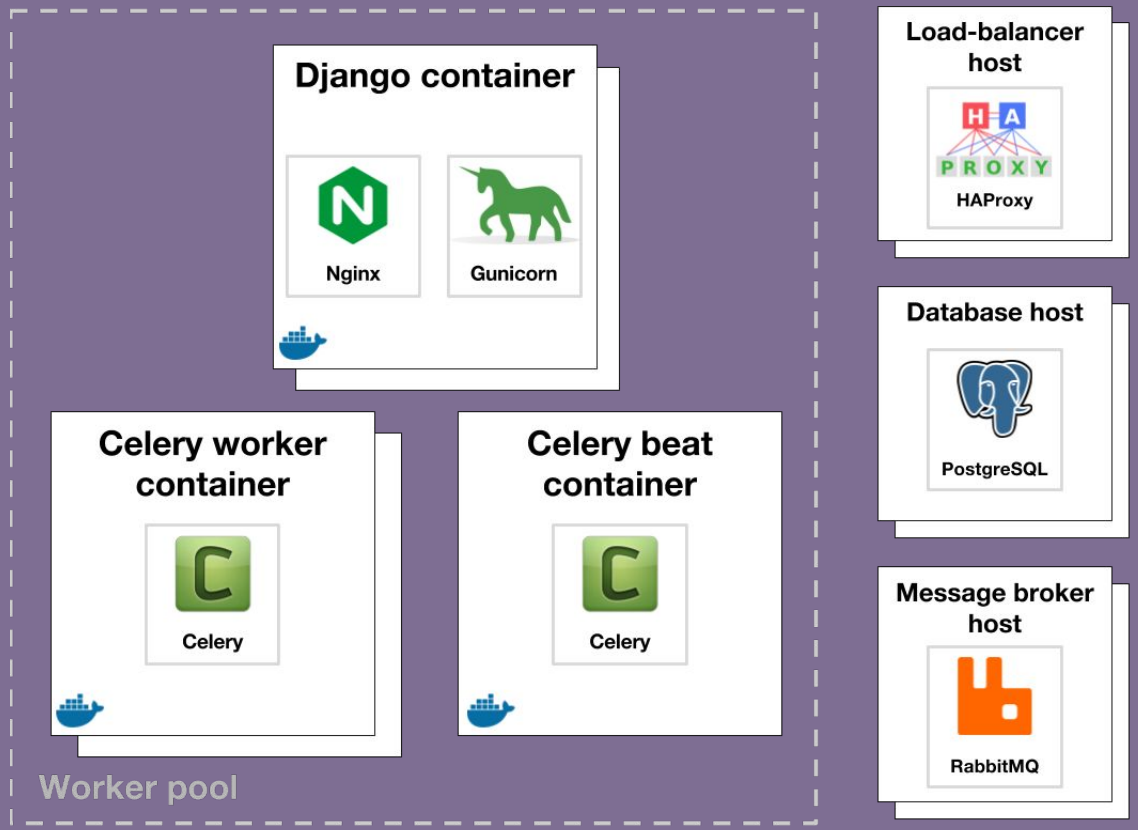


...just chucked it in a container

Do not do this

- Docker containers are *not* mini-VMs
- Isolate processes, not services
- Good health checks very difficult
- No programs to manage multiple processes (no init system)
- Container orchestration systems expect containers to be ephemeral

What we've settled on



Configuration

- ***Don't*** **want to build a Docker image with all the config files inside it**
  - Not flexible, slow reconfiguration
  - Secrets in Docker image
- **Difficult to move configuration files around with containers**
- **Solution: Django settings module reads config from environment variables**

django-environ

DATABASE_URL=postgres://user:pass@db01/dbname

CACHE_URL=memcache://mem01:11211,mem02:11211

EMAIL_URL=smtp+tls://user:pass@smtp01:465

```python
import environ

env = environ.Env()

# Raises ImproperlyConfigured exception if SECRET_KEY not in os.environ
SECRET_KEY = env("SECRET_KEY")
DEBUG = env("DEBUG", default=False)

DATABASES = {"default": env.db(default="sqlite:////tmp/db.sqlite3")}
CACHES = {"default": env.cache(default="locmemcache://")}
EMAIL_CONFIG = env.email_url(default="consolemail://")
```

Startup (entrypoint) scripts

When the container starts we need to do some things...

- Run database migrations
- Create a superuser account on first run
- Set some default Gunicorn arguments
- Switch to a non-root user
- More… (but hopefully not)

Logging

Since containers are ephemeral, so are their log files

- Log everything to stdout/stderr
- Container orchestrators will collect this
- Even more important that only one thing runs in a container
- Bonus points: make your logs machine-readable

User-uploaded files

User-uploaded files in Django can be stored in a “media” directory

- Don’t do this (containers or not)
- Extra hard if you have to manage networked storage
- Use `django-storages`, store in S3

`django-bootstrap` image

(Not to be confused with CSS Bootstrap)

- Standardized base image for all our Django deployments
- Nginx configuration optimised for Django in a container
- Startup scripts for Django & Celery
- Thoroughly tested with example app

`praekeltfoundation/docker-django-bootstrap`

`django-bootstrap` Dockerfile

```
1   FROM praekeltfoundation/django-bootstrap
2
3   # Copy in our source code and install it
4   COPY . .
5   RUN pip install -e .
6
7   # Tell Django & Celery where our config & app is
8   ENV DJANGO_SETTINGS_MODULE cake_service.settings
9   ENV CELERY_APP cake_service
10
11  # Collect static files so that they can be served
12  RUN django-admin collectstatic --noinput
13
14  # Pass arguments to Gunicorn: where our WSGI app is
15  CMD ["cake_service.wsgi:application"]
```

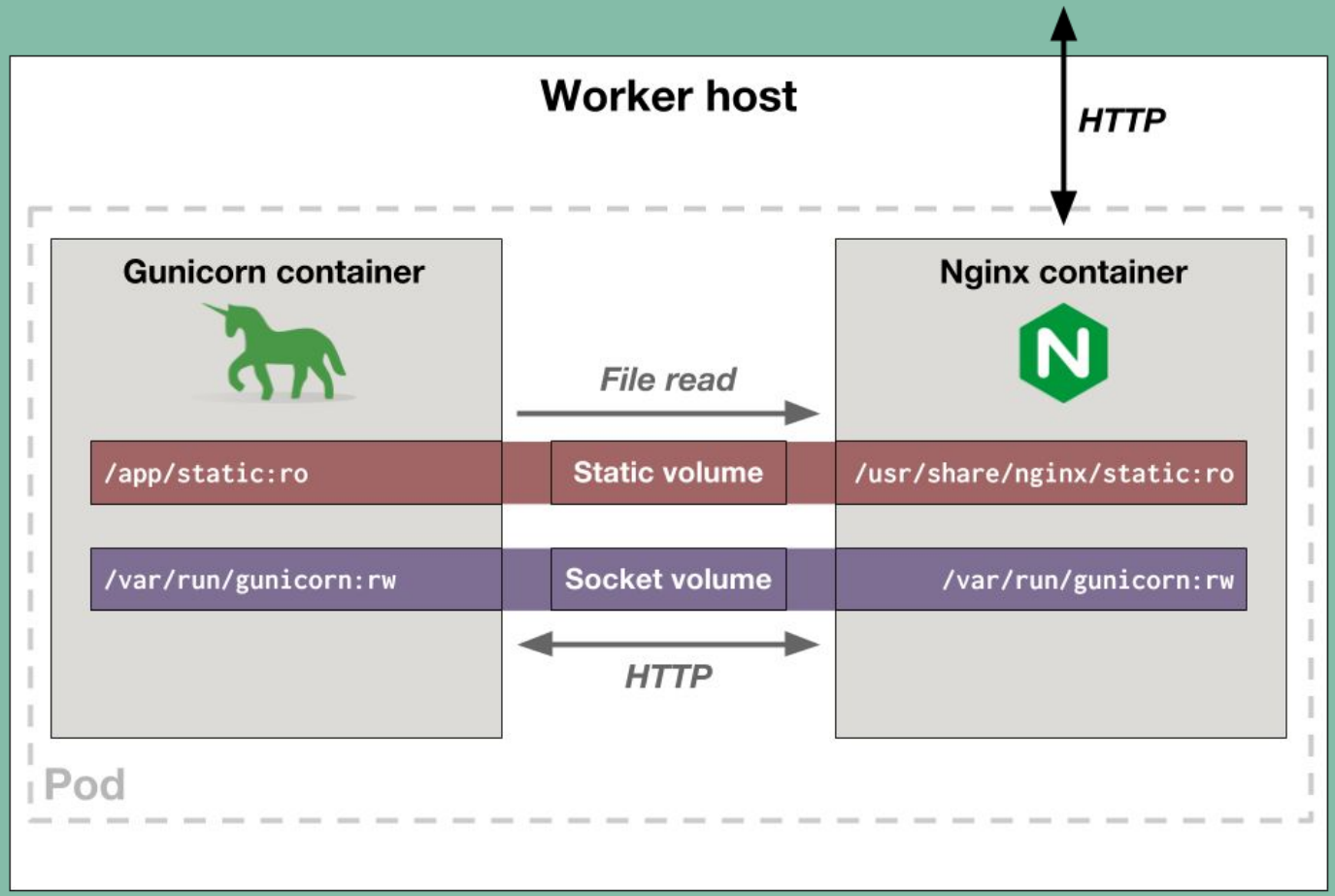
04

Where to from here?

Further improvements

But you're still running more than one thing in a container?

Deploying as a *pod*



Configuration secrets

- We're now putting passwords in environment variables
- But env vars are quite easy to leak
- Container orchestration platforms have tools for storing secret data securely
- Dynamic credential management (Hashicorp Vault): credentials only valid as long as the container exists

Metrics via nginx-lua-prometheus

```
# HELP nginx_http_connections Number of HTTP connections
# TYPE nginx_http_connections gauge
nginx_http_connections{state="reading"} 0
nginx_http_connections{state="waiting"} 0
nginx_http_connections{state="writing"} 1

# HELP nginx_http_request_duration_seconds HTTP request latency
# TYPE nginx_http_request_duration_seconds histogram
nginx_http_request_duration_seconds_bucket{host="localhost",le="00.005"} 3
nginx_http_request_duration_seconds_bucket{host="localhost",le="00.010"} 3
...
nginx_http_request_duration_seconds_bucket{host="localhost",le="10.000"} 3
nginx_http_request_duration_seconds_bucket{host="localhost",le="+Inf"} 3
nginx_http_request_duration_seconds_count{host="localhost"} 3
nginx_http_request_duration_seconds_sum{host="localhost"} 0.002000093460083

# HELP nginx_http_requests_total Number of HTTP requests
# TYPE nginx_http_requests_total counter
nginx_http_requests_total{host="localhost",status="200"} 1
nginx_http_requests_total{host="localhost",status="404"} 1
```

Metrics via `nginx-lua-prometheus`

- Prometheus can poll container orchestrator to know where to scrape
- Get Django-specific metrics from Nginx (e.g. all requests not to `/static/`)
- “Free” metrics for all apps with same base image
- But you *should* properly instrument your Django application...

05

In conclusion

Containers are cool, but...

Containers/container orchestration can seem complicated...

- Persistent storage difficult*
- No config files*
- No log files*
- Can only run one thing per container*
- Can’t SSH in*
- Distributed system 😰

**Roughly speaking*

...but they have some big advantages

- Easy deployments
- Easy scaling
- Efficient resource usage
- Generally increased automation
- Consistently packaged apps

Containers for Django

A common base image can provide...

- Tested and optimised server (Nginx) config
- Encapsulate best practices for containers & container orchestration platform
- Consistent platform for deploying apps
- Potential for adding new features

Questions?

praekeltfoundation/docker-django-bootstrap

PRAEKELT·ORG

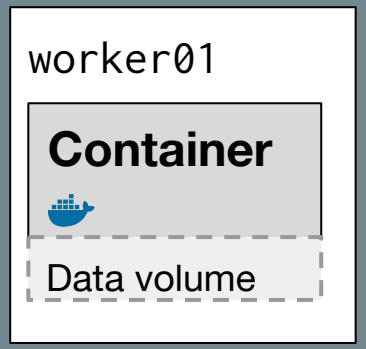
Thank you.

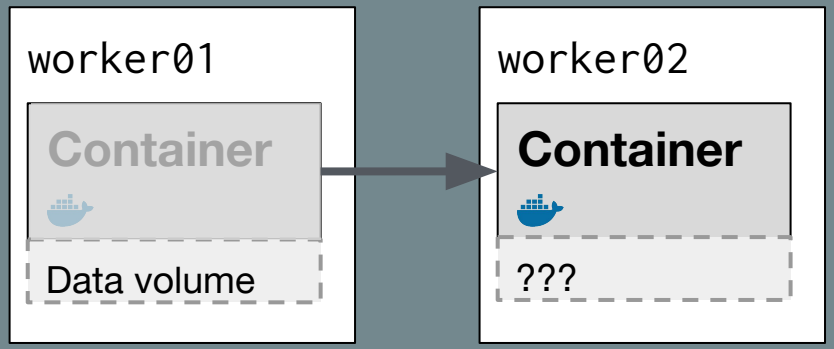
Special thanks to Jeremy Thurgood (@jerith) for reviewing my code.
Go see his talk later!

PRAEKELT·ORG

Complication 1: Persistent storage

Moving data is harder than moving a container





1. The container needs to be moved

2. But its data needs to move with it

Complication 2: Networking

Things move around and thus have weird addresses

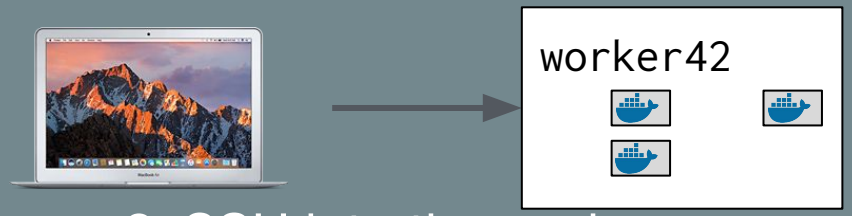
Complication 3: Debugging

It's hard to just "SSH into" a container

```
curl controller01:8080/v2/apps ...
```

1. Find which worker the container is on

```
ssh -t public01 ssh worker42
```



2. SSH into the worker

```
docker ps | grep cake-service
```

3. Find the container ID

```
docker exec -it 981681d291ab bash
```

```
root@981681d291ab:~#
```

4. Run Bash in the container