October 12, 2023

How we made our app scalable

yodeck

Who am I

- Spyros Panagiotopoulos
- Software/DevOps engineer
- Curious about tech
- Problem Solver

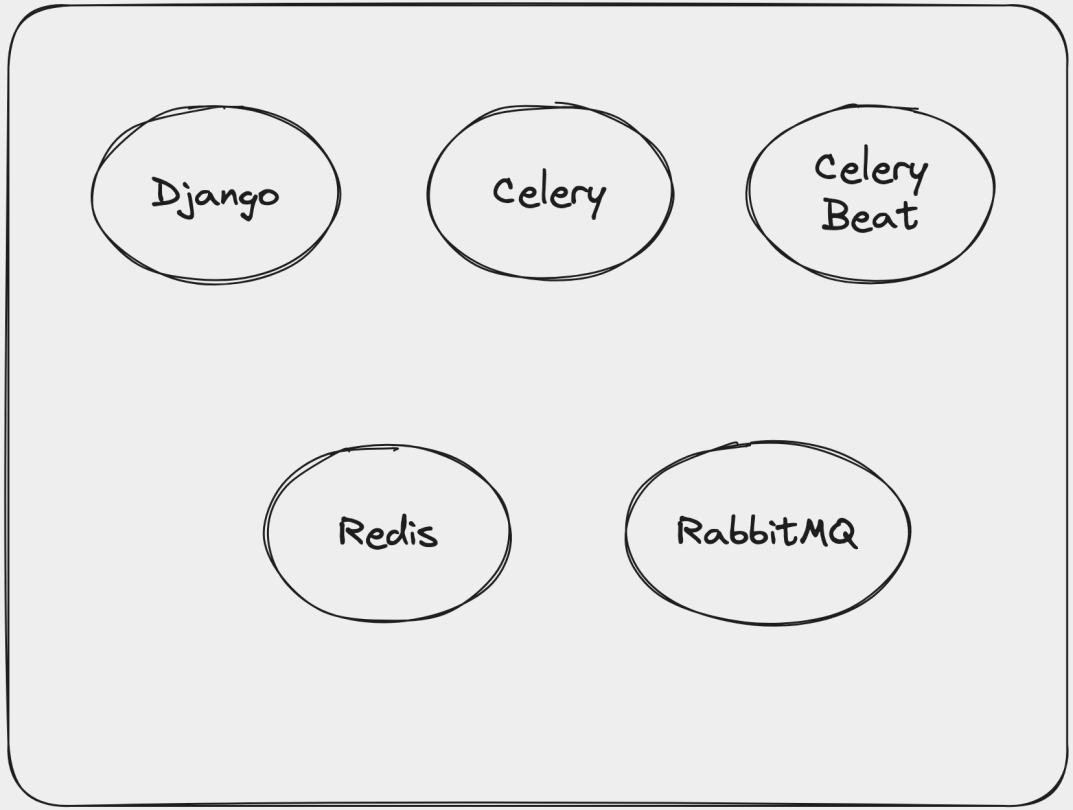
Stateful vs Stateless

- Maintains and relies on the state or data between interactions.
- Uses memory or files to store and retrieve information.
- Examples include traditional web applications that rely on session cookies to remember user data.

- Does not store session-specific data between interactions.
- Each request treated as an independent and self-contained transaction.
    - Required data along with request.
- Highly scalable.

yodeck

Our CMS instance before



- R5.2xlarge
  - 8 vCPUs
  - 64GB RAM

How we scale?

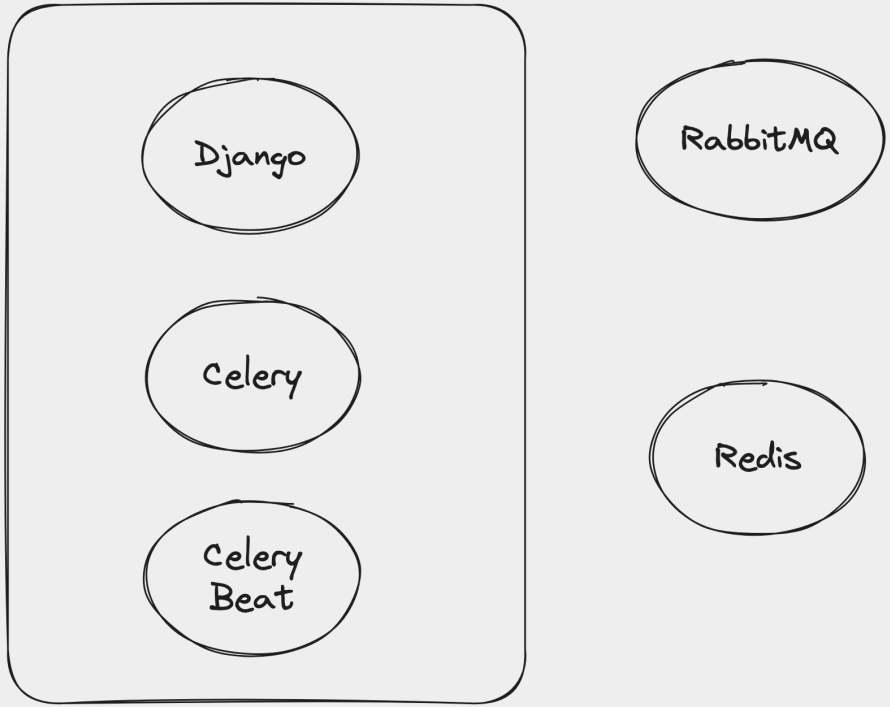
Monitoring

If you can’t measure it
you can’t improve it

Stand on the shoulders of giants

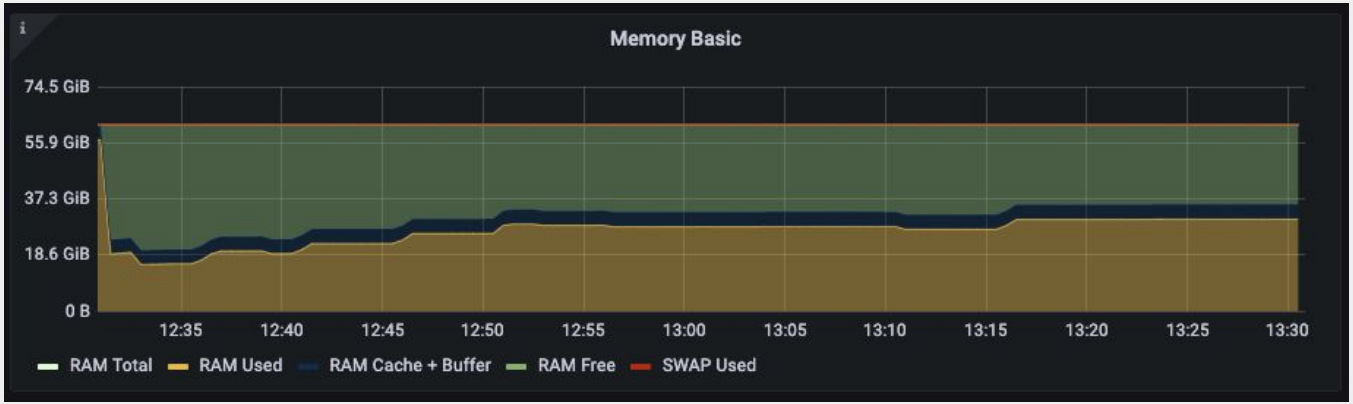
Consider migration to managed services

- Redis and RabbitMQ with the application server.
  - Makes sense when cost sensitive.
  - Might require scaling along with the application.
- With migration:
  - Pay a little extra
  - Get rid of maintenance



Utilize the --max-tasks-per-child flag in Celery

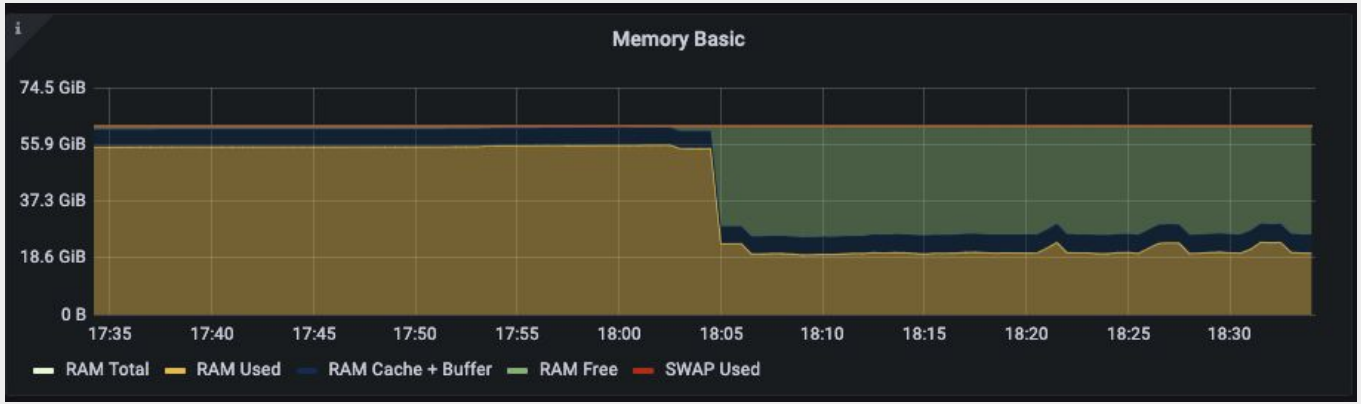
- Celery has issues with memory leaks
  - External libraries make things worse



yodeck

Utilize the --max-tasks-per-child flag in Celery

- Built in flag that kills workers after X tasks
- Find the right tradeoff between memory balance and CPU overhead
    - 400 tasks works well for us



yodeck

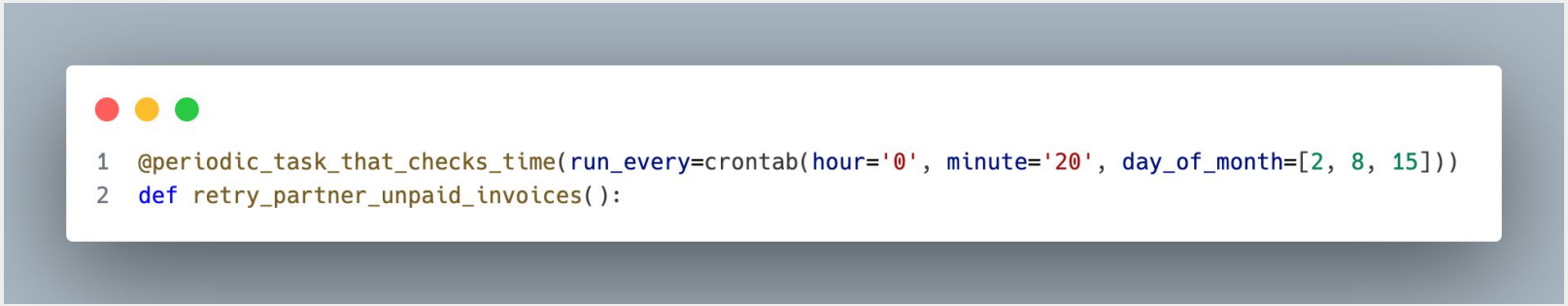
Use a database for Celery Beat

- Celery beat stores tasks metadata in a file by default
  - (Used to be) easily corruptible shelve database

```
@periodic_task_that_checks_time(run_every=crontab(hour='0', minute='20', day_of_month=[2, 8, 15]))
def retry_partner_unpaid_invoices():
```

Use a database for Celery Beat

- Experimented with `django-celery-beat`

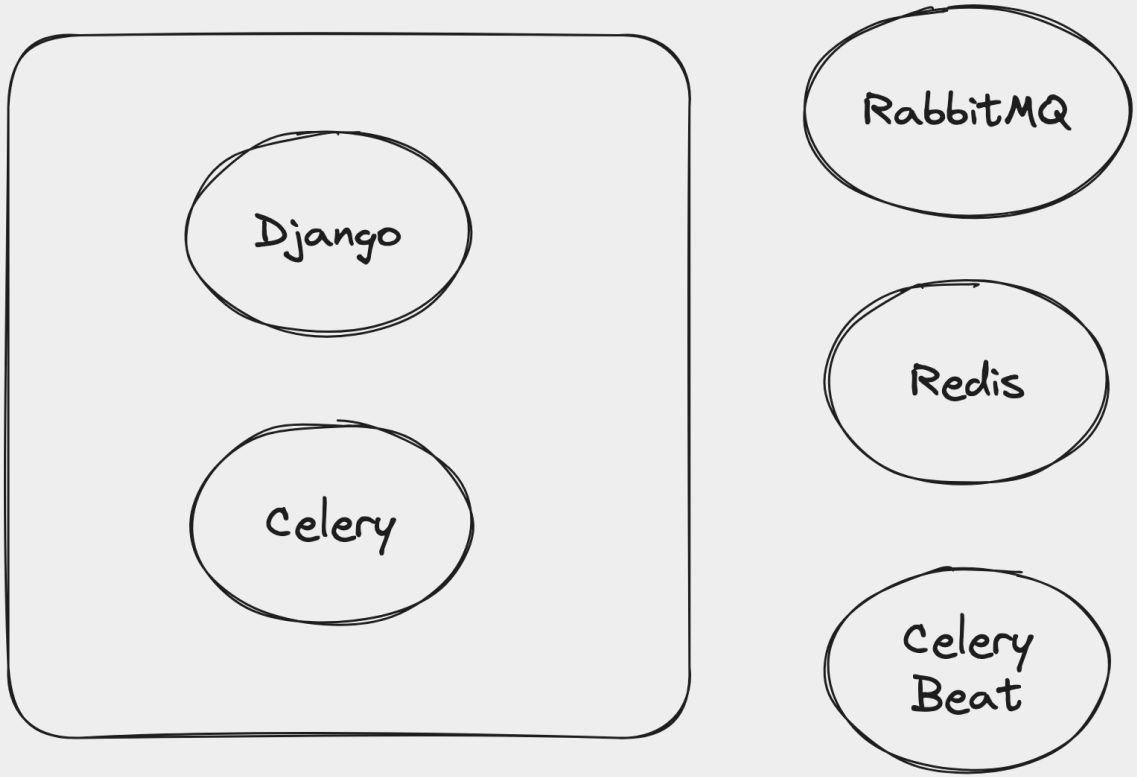


```python
@periodic_task_that_checks_time(run_every=crontab(hour='0', minute='20', day_of_month=[2, 8, 15]))
def retry_partner_unpaid_invoices():
```

- Ended up with `redbeat`

```
celery -A app beat --detach \
    --scheduler=redbeat.RedBeatScheduler
```

yodeck


Django
Celery
RabbitMQ
Redis
Celery
Beat

Avoid Writing Data to Files

- State should live outside of the application or its infrastructure
- We stored audit logs in files
  - Rotated each day
  - A task separated logs of each account
- When scaled logs would be missed

Avoid reading data from files

- Don’t rely that files generated by your application code would be always there
    - Don’t create them in the first place
- Zoho SDK does this by default
    - Persists authentication token in a file
    - Thankfully, base class can be extended

Store sessions in the database

- Web requests are stateless
- To prove identity clients send a piece of information every time
- 2 ways of storing this information
  - Data inside a cookie
  - Data on server, tied with a random string, called Session ID
- Web Servers prefer Session IDs
  - Harder to tamper with
  - Total control of session expiration
  - Store more data than cookie
  - Session data in memory
  - What happens when requests are spread across servers

Store sessions in the database

- Django has an easy way of storing sessions in database
  - Also has an easy way to utilize both database & cache

```
INSTALLED_APPS = (
    'django.contrib.sessions',
    )
MIDDLEWARE = (
    'django.contrib.sessions.middleware.SessionMiddleware',
)
SESSION_ENGINE = "django.contrib.sessions.backends.cached_db"
```

Zero downtime rolling updates with uWSGI

yodeck

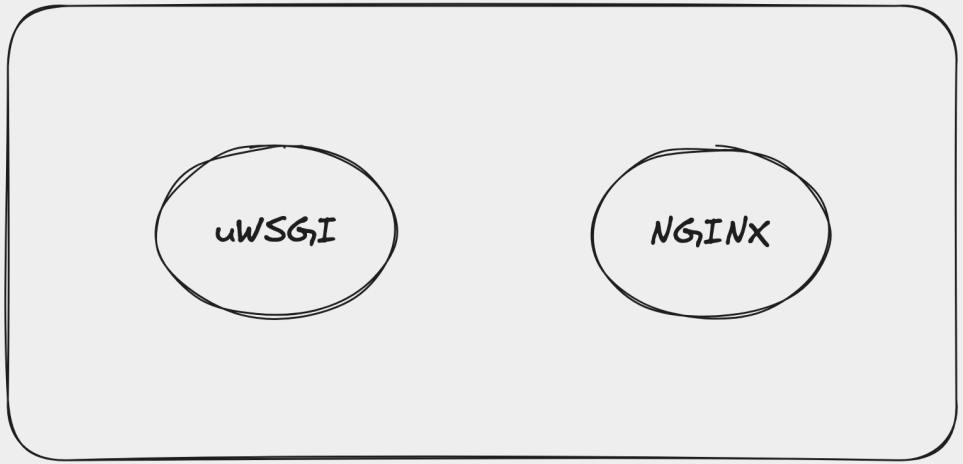
uWSGI

- Application server and protocol
  - Serve Python applications
- Single server
  - Pull the code
  - Perform migrations
  - Gracefully reload workers

yodeck

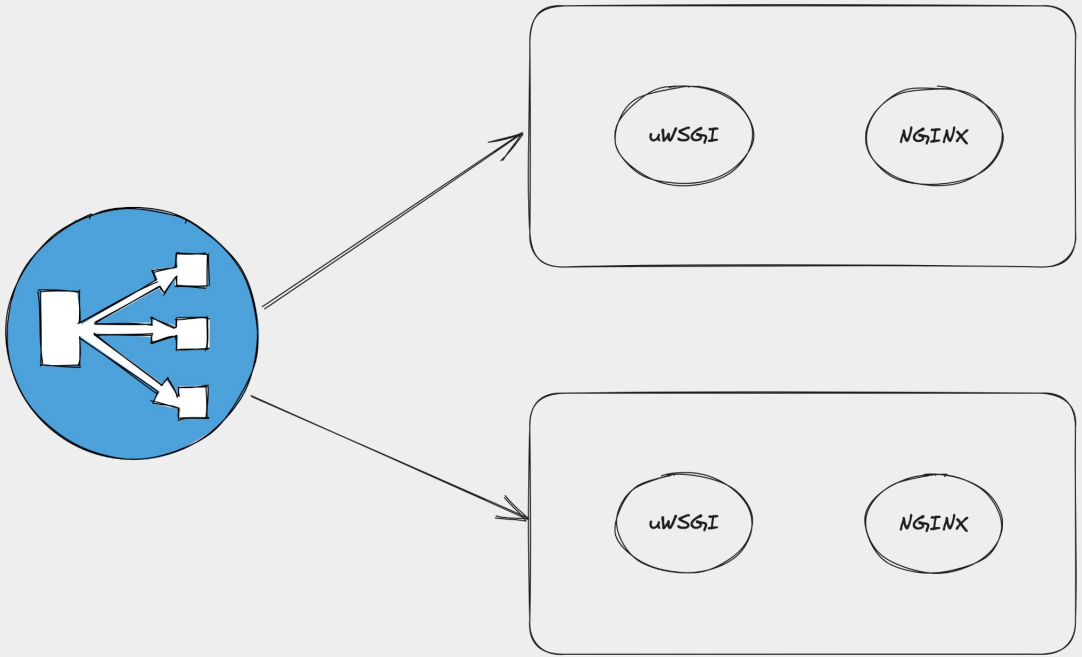
uWSGI and Kubernetes

- 2 containers in our pod
- 2 readiness probes
  - Custom nginx endpoint that returns 200
  - GET to our login page



uWSGI and Kubernetes

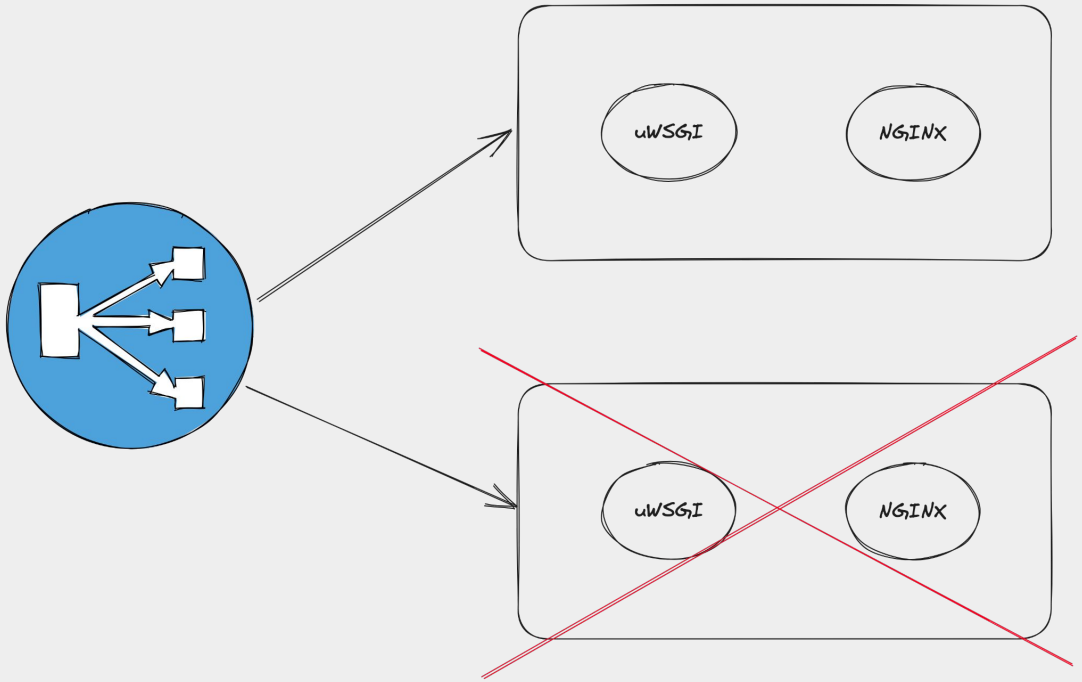
- A new pod gets created
- When it's ready, it's added in the Kubernetes service
- Starts serving requests



yodeck

uWSGI and Kubernetes

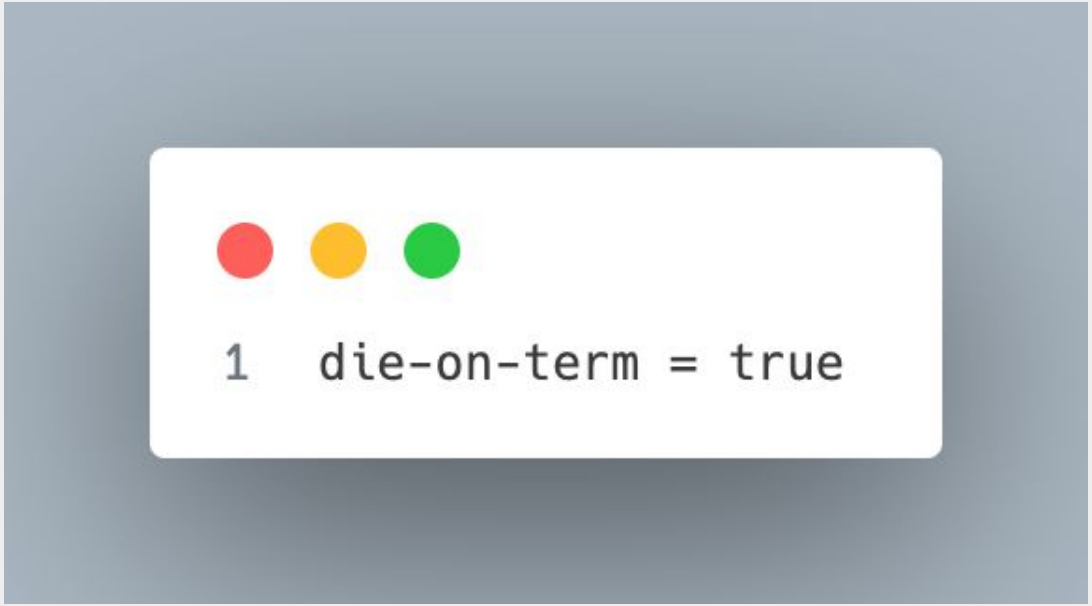
- What happens when a pod gets terminated?
- Kubernetes sends a SIGTERM signal
    - SIGTERM by default brutally reloads the uWSGI workers



yodeck

uWSGI and Kubernetes

- uWSGI docs recommend to use the `die-on-term` flag to respect the SIGTERM signal

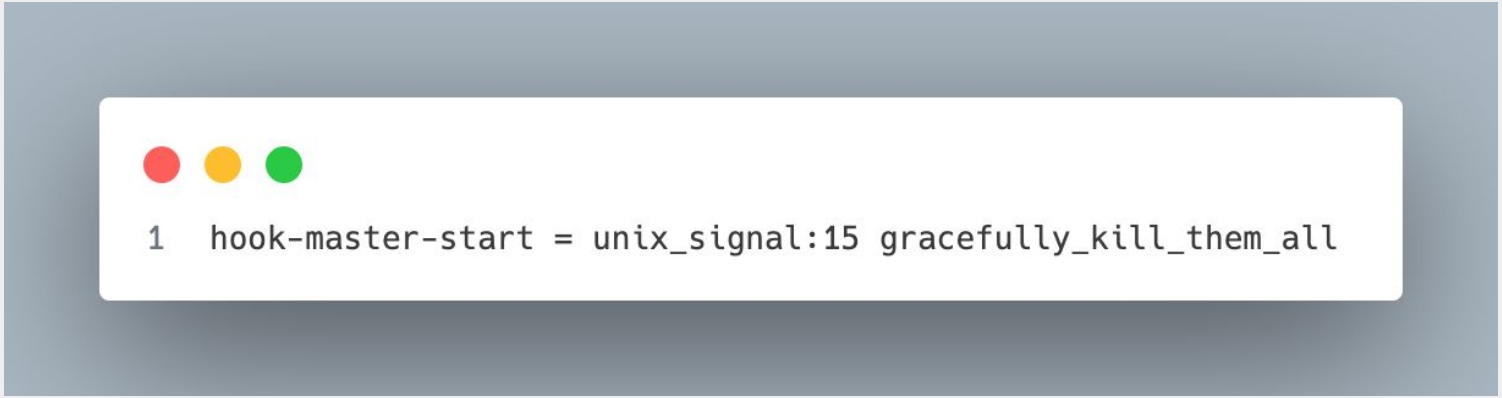


- But this brutally kills all the workers

yodeck

uWSGI and Kubernetes

- SIGTERM signal can be trapped with the following configuration



```
1  hook-master-start = unix_signal:15 gracefully_kill_them_all
```

uWSGI and Kubernetes

- uWSGI exposes a FIFO to interact with it
  - Reload logs
  - Reload the code
  - Gracefully kill workers

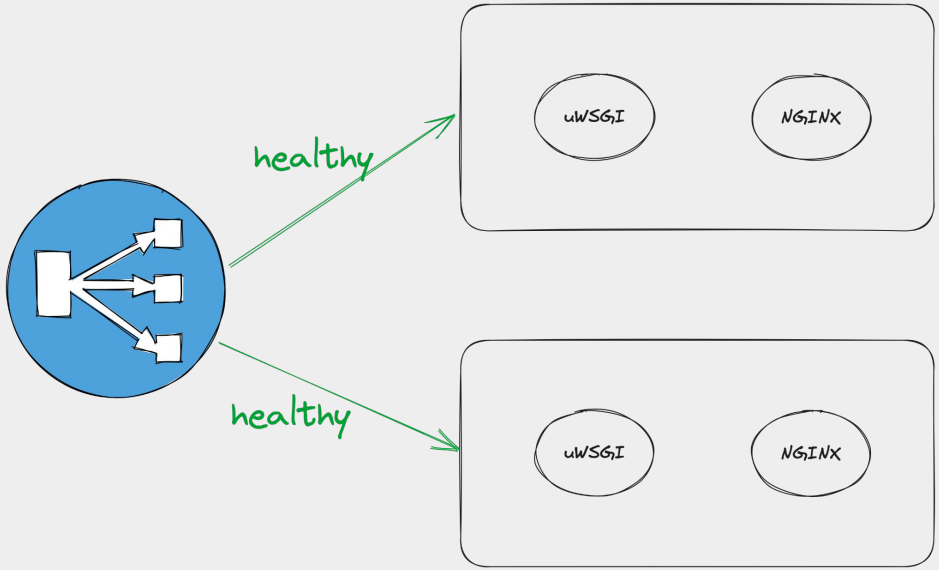
```yaml
lifecycle:
  preStop:
    exec:
      command:
      - "/bin/sh"
      - "-c"
      - "echo q > /run/uwsgi/uwsgififo"
```

- This should work but we still received 502 responses

yodeck

uWSGI and Kubernetes

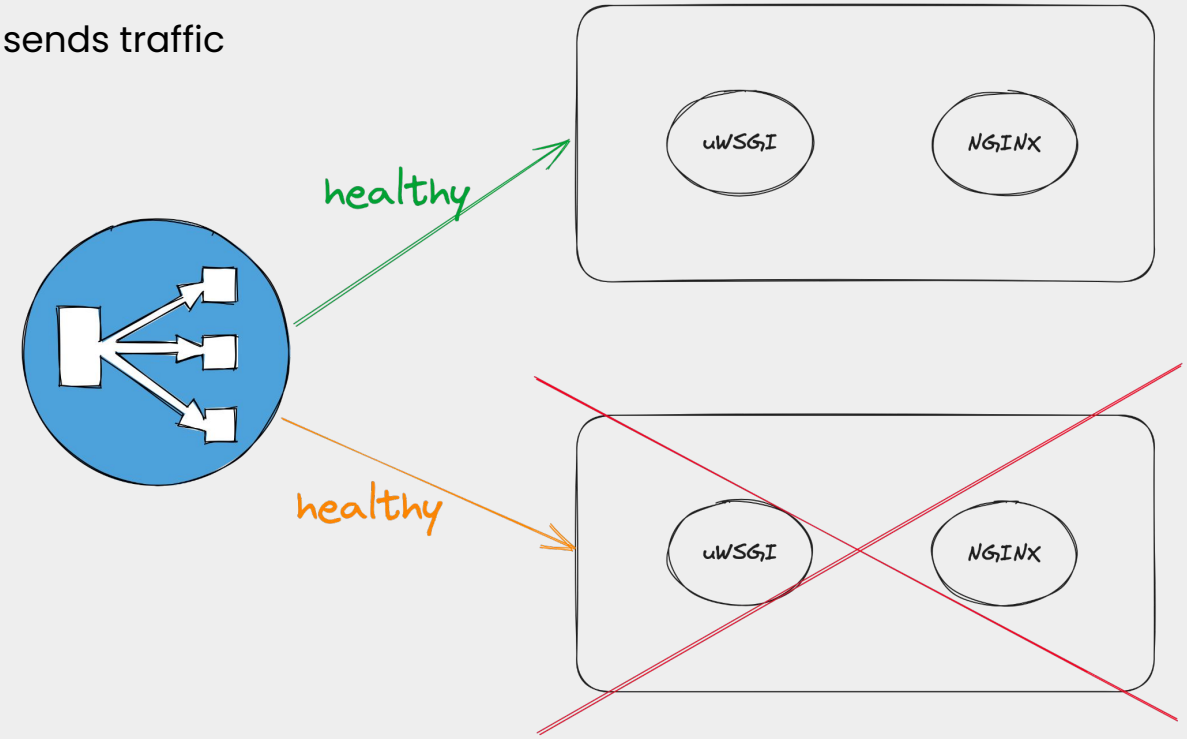
- We use Application Load Balancer from AWS
- When a pod is created and targeted by a service, it is added to the ALB target group
    - We have a health check in ALB as well
- When the pod is ready, ALB sends traffic to it



yodeck

uWSGI and Kubernetes

- When a pod is terminated, it takes some time to deregister from ALB
- But ALB sends traffic



yodeck

uWSGI and Kubernetes

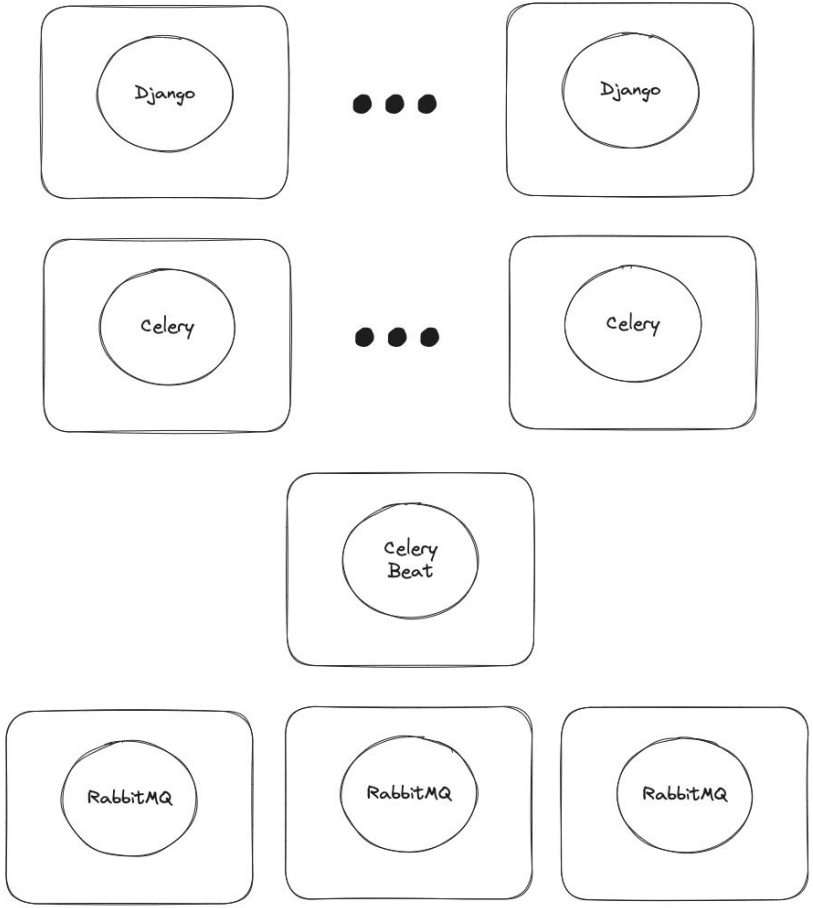
- We can use **Readiness Gate**
- Injects extra data to the pod
- Kubernetes knows the ALB registration status

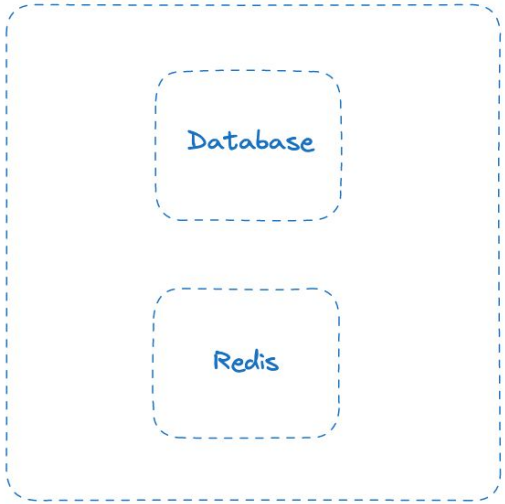
```
$ kubectl label namespace readiness elbv2.k8s.aws/pod-readiness-gate-inject=enabled
namespace/readiness labeled
```

yodeck

Today

yodeck


Django
Django
Celery
Celery
Celery Beat
RabbitMQ
RabbitMQ
RabbitMQ


Database
Redis

Our CMS now

Questions?

Thank you

yodeck