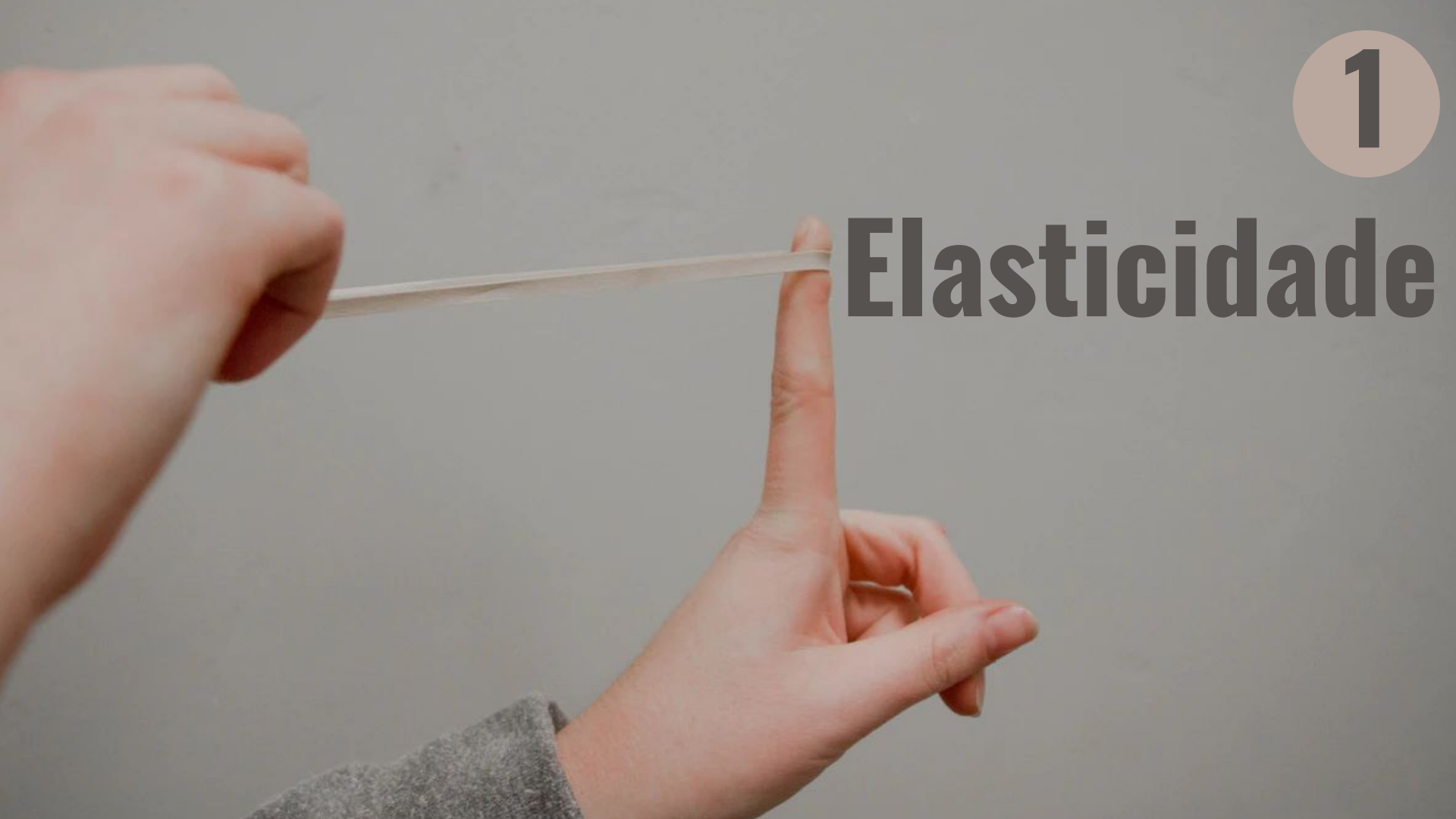
@fernandrone

Projetando containers **descartáveis**

como tratar sinais no **Docker** e **Kubernetes!**

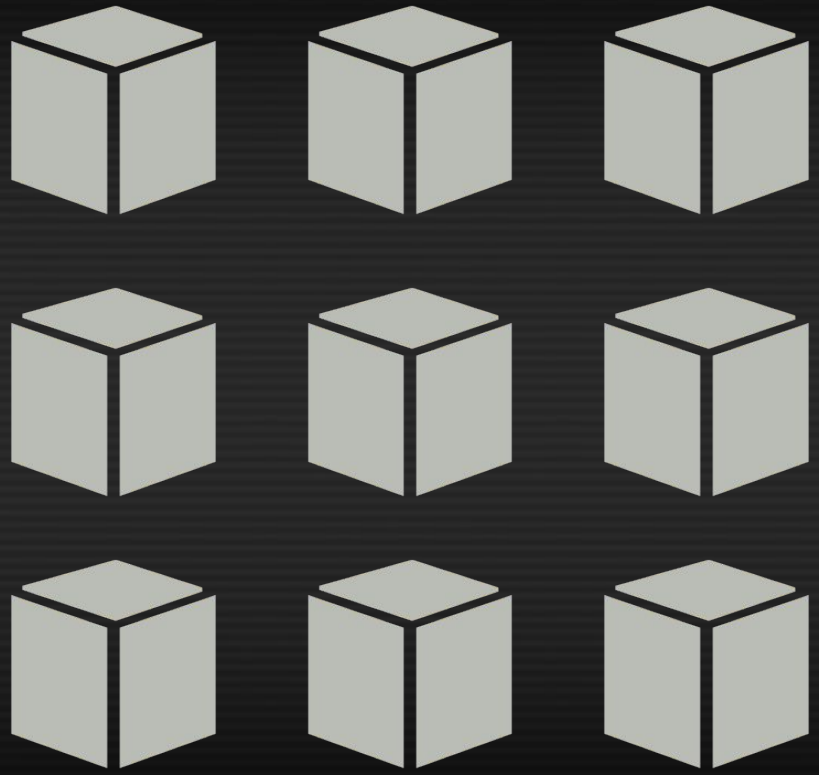
> fdr.one/sinais-code

> fdr.one/sinais-slides



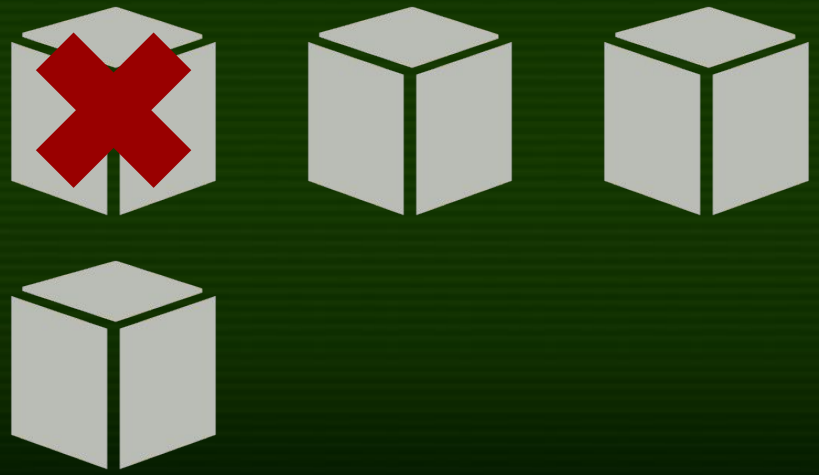
1

Elasticidade



2

Resiliência



Não existe ferramenta mágica que vai te entregar isso

Você precisa planejar sua aplicação

para que ela seja **elástica**

e **resiliente**.

// FERNANDO BARBOSA

SRE @ QuintoAndar

Programador YAML

Go | CI/CD | Kubernetes | Observability

> **@fernandrone**

> mail@fdr.one

> fernandrone.com

Você precisa planejar sua aplicação

para que ela seja **elástica**

e **resiliente**.

The Twelve-Factor App

(A aplicação doze-fatores)

by Heroku c. 2011 · https://12factor.net/pt_br/

The Twelve-Factor App

(A aplicação doze-fatores)

by Heroku c. 2011 · https://12factor.net/pt_br/

I Base de Código
II Dependências
III Configurações
IV Serviços de Apoio
V Construa, lance, execute
VI Processos
VII Vínculo de porta
VIII Concorrência
IX Descartabilidade
X Dev/prod semelhantes
XI Logs
XII Processos de Admin

The Twelve-Factor App

(A aplicação doze-fatores)

by Heroku c. 2011 · https://12factor.net/pt_br/

I Base de Código
II Dependências
III Configurações
IV Serviços de Apoio
V Construa, lance, execute
VI Processos
VII Vínculo de porta
VIII Concorrência
IX Descartabilidade
X Dev/prod semelhantes
XI Logs
XII Processos de Admin

The Twelve-Factor App

(A aplicação doze-fatores)

by Heroku c. 2011 · https://12factor.net/pt_br/

IX Descartabilidade

Os **processos** de uma aplicação doze-fatores são **descartáveis**: podem ser iniciados ou parados a qualquer momento.

Isso facilita o **escalonamento elástico**, as mudanças de configuração e a **resiliência** do código em produção.

COMO?

Aplicações devem **desligar-se graciosamente** ao receber um sinal do tipo **SIGTERM**.

Um sinal é uma **interrupção** de software.

https://www.gnu.org/software/libc/manual/html_node/Signal-Handling.html#Signal-Handling

Um sinal é uma **interrupção** de software.

Um processo pode enviar um sinal para outro processo; isso permite que um processo pai termine seus filhos, ou que dois processos se comuniquem e sincronizem.

https://www.gnu.org/software/libc/manual/html_node/Signal-Handling.html#Signal-Handling

alguns Termination Signals...

NOME	ATALHO	TRATÁVEL	DESCRIÇÃO
SIGTERM	-	**SIM**	Modo normal para pedir que um programa termine.
SIGKILL	-	**NÃO**	Término imediato do programa, não pode ser tratado.

https://www.gnu.org/software/libc/manual/html_node/Termination-Signals.html#Termination-Signals

alguns Termination Signals...

NOME	ATALHO	TRATÁVEL	DESCRIÇÃO
SIGTERM	-	**SIM**	Modo normal para pedir que um programa termine.
SIGKILL	-	**NÃO**	Término imediato do programa, não pode ser tratado.
SIGINT	CTRL+C	SIM	Gerado por interrupção do usuário, pode ser tratado ou não pelo programa.
SIGQUIT	CTRL+\	SIM	Similar a SIGINT, mas deve gerar um “core dump”.
SIGHUP	-	SIM	Deve ser utilizado para notificar que o terminal do usuário foi desconectado.

https://www.gnu.org/software/libc/manual/html_node/Termination-Signals.html#Termination-Signals

Aplicações devem **desligar-se graciosamente** ao receber um sinal do tipo **SIGTERM**.

Aplicações devem **desligar-se graciosamente** ao receber um sinal do tipo **SIGTERM**.

Aplicações devem **desligar-se graciosamente** ao receber um sinal do tipo **SIGTERM**.

Recusar novas requisições web

Aplicações devem **desligar-se graciosamente** ao receber um sinal do tipo **SIGTERM**.

Recusar novas requisições web

Esperar que requisições em andamento terminem

Aplicações devem **desligar-se graciosamente** ao receber um sinal do tipo **SIGTERM**.

Recusar novas requisições web

Esperar que requisições em andamento terminem

Retornar tarefas para filas de trabalho

Projetando containers descartáveis

como tratar sinais no Docker e Kubernetes!

```
$ cat trap.sh

#!/bin/bash
trap "handle INT" INT   # 2
trap "handle KILL" KILL # 9 - NÃO FUNCIONA
trap "handle TERM" TERM # 15

handle() {
    echo "Trapped: $1"
    echo "Encerrando o processo graciosamente..."

    sleep 2

    echo "Processo encerrado"

    exit 0 # Importante!
}

echo "Iniciando o processo (PID $$)..."
sleep infinity & # Espera para sempre e cria um novo processo
wait             # Espera o novo processo
```

```
$ ./trap.sh
Iniciando o processo (PID 9416)...
^CTrapped: INT
Encerrando o processo graciosamente...
Processo encerrado

$ ./trap.sh
Iniciando o processo (PID 9529)...
Trapped: TERM
Encerrando o processo graciosamente...
Processo encerrado
```

```
$ kill 9529
```

1

Sempre que possível trate *(handle)*
SIGTERM na sua aplicação.

docker

Why Docker? Products Use Cases Developers Company

Sign In Get Started

Debug your app, not your environment

Securely build and share any application, anywhere

Get started

READ MORE

Introducing Docker's Public Roadmap

Helping you and your development team build and ship faster.

→ View Docker Roadmap
→ Read the Blog

stackoverflow

SURVEY

Docker is a Developer Favorite

In Stack Overflow survey developers ranked Docker #1 in "Most Loved Platform", #2 "Most Wanted Platform" and #3 "Platform In Use".

→ Check out the full survey

Docker Hub

$ docker stop

Description

Stop one or more running containers

Usage

docker stop [OPTIONS] CONTAINER [CONTAINER...]

Extended description

The main process inside the container will receive **SIGTERM**, and after a grace period, **SIGKILL**.

https://docs.docker.com/engine/reference/commandline/stop/

```
$ cat Dockerfile.exec

    FROM debian:stable

    ADD trap.sh trap.sh

    ENTRYPOINT [ "./trap.sh" ]
```

```
$ docker build -t trap:exec -f Dockerfile.exec .
Sending build context to Docker daemon   7.68kB
Step 1/3 : FROM debian:stable
 ---> 5e3221e89de8
Step 2/3 : ADD trap.sh trap.sh
 ---> Using cache
 ---> 96dbeb823e76
Step 3/3 : ENTRYPOINT [ "./trap.sh" ]
 ---> Running in 208b5f6ce010
Removing intermediate container 208b5f6ce010
 ---> 995514e0f23b
Successfully built 995514e0f23b
Successfully tagged trap:exec

$ docker run -it --rm --name trap trap:exec
Iniciando o processo (PID 1)...
Trapped: TERM
Encerrando o processo graciosamente...
Processo encerrado
```

```
$ docker stop trap
exec
```

```
$ cat Dockerfile.shell

    FROM debian:stable

    ADD trap.sh trap.sh

    ENTRYPOINT "./trap.sh"
```

```
$ cat Dockerfile.shell

    FROM debian:stable

    ADD trap.sh trap.sh

    ENTRYPOINT "./trap.sh"
```

```
$ docker build -t trap:shell -f Dockerfile.shell .
Sending build context to Docker daemon  6.656kB
Step 1/3 : FROM debian:stable
 ---> 5e3221e89de8
Step 2/3 : ADD trap.sh trap.sh
 ---> Using cache
 ---> 96dbeb823e76
Step 3/3 : ENTRYPOINT "./trap.sh"
 ---> Running in e55f122d4c92
Removing intermediate container e55f122d4c92
 ---> fb6d971ed0f9
Successfully built fb6d971ed0f9
Successfully tagged trap:shell

$ docker run -it --rm --name trap trap:shell
Iniciando o processo (PID 7)...
Trapped: TERM
Encerrando o processo graciosamente...
Processo encerrado
```

```
$ docker stop trap
shell
```

ENTRYPOINT tem duas formas:

- ENTRYPOINT ["executable", "param1", "param2"] (forma "**exec**", **preferível**)
- ENTRYPOINT command param1 param2 (forma "**shell**")

Na forma "shell" seu ENTRYPOINT será iniciado como um subcomando de /bin/sh -c, que **não** passa sinais para frente.

Isso significa que o executável **não** será o PID 1 do container e **não** receberá sinais Unix - então seu executável **não** receberá o SIGTERM do **docker stop**.

https://docs.docker.com/engine/reference/builder/#entrypoint

Existe uma de contornar as limitações da forma “shell”...

usando o comando “exec”

```
FROM debian:stable

ADD trap.sh trap.sh

ENTRYPOINT exec ./trap.sh
```

Você também pode **mudar** o sinal de término enviado por **docker stop**

```
FROM debian:stable

ADD trap.sh trap.sh

STOPSIGNAL SIGQUIT

ENTRYPOINT exec ./trap.sh
```

```
FROM debian:stable

ADD trap.sh trap.sh

STOPSIGNAL SIGQUIT

ENTRYPOINT exec ./trap.sh
```

Mas (em geral) não é uma boa ideia fazer isso 😞

```
$ docker run -it trap --stop-signal SIGQUIT
```

```
$ docker run -it trap --stop-signal SIGQUIT
```

Mas (em geral) **também** não é uma boa ideia fazer isso 😞

2

Em seus Dockerfiles, utilize sempre ENTRYPOINT na forma “exec”.

Production-Grade Container Orchestration

Automated container deployment, scaling, and management

Learn Kubernetes Basics

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

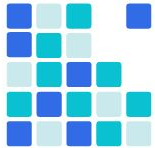
It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.

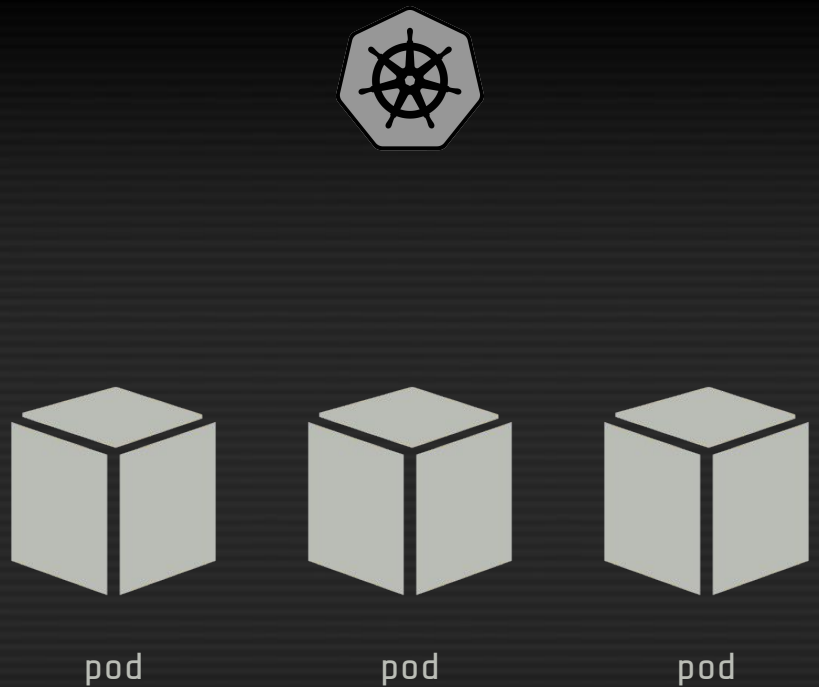
Planet Scale

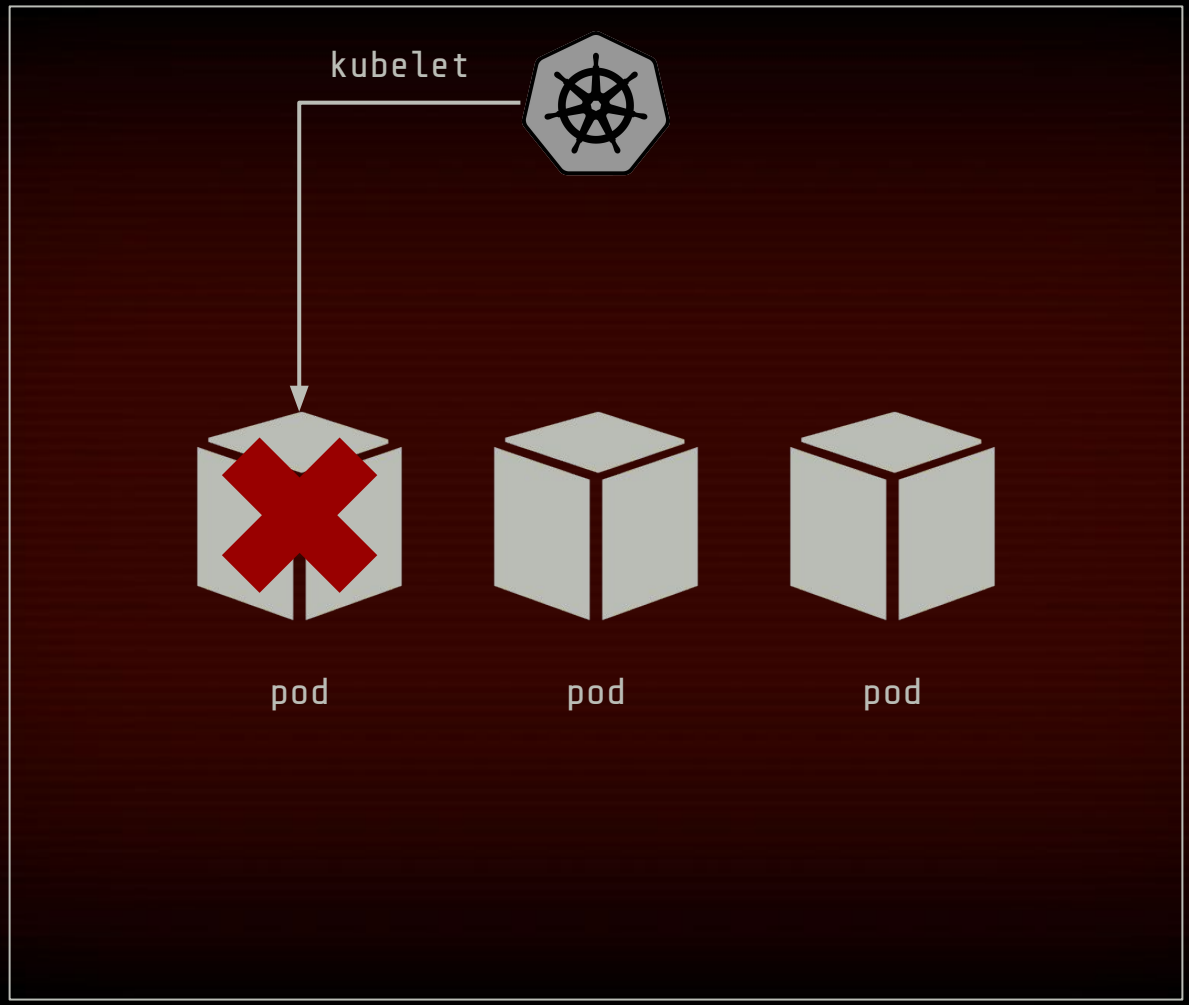
Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.

Never Outgrow

Whether testing locally or running a global enterprise, Kubernetes flexibility grows with you to deliver your applications consistently and easily no matter how complex your need is.




pod
pod
pod


kubelet
pod
pod
pod

1. Status **“Terminating”** e para de receber tráfego
2.
3.
4.
5.

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

1. Status **“Terminating”** e para de receber tráfego
2.
3.
4.
5.

Aplicações **12-factor** devem

Recusar novas requisições web

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3.
4.
5.

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

```
$ cat pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: trap
  labels:
    app: trap
spec:
  containers:
  - name: trap
    image: trap:exec
    imagePullPolicy: IfNotPresent
    command: ['./trap.sh']
    lifecycle:
      preStop:
        exec:
          command: ["echo", "Executando preStop Hook"]
```

Mais detalhes em https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3. **SIGTERM*** é enviado para todos containers do Pod
4.
5.

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3. **SIGTERM*** é enviado para todos containers do Pod
4.
5.

*SIGTERM é o padrão, mas com o comando STOPSIGNAL no Dockerfile podemos trocar o sinal utilizado!

O Kubernetes “respeita” o Dockerfile (e não está documentado).

Ver essa issue.

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3. **SIGTERM*** é enviado para todos containers do Pod
4. Kubernetes espera o **gracePeriod** (padrão é 30s)
5.

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3. **SIGTERM*** é enviado para todos containers do Pod
4. Kubernetes espera o **gracePeriod** (padrão é 30s)
5.

Aplicações **12-factor** devem

Esperar que requisições em andamento terminem

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

```
$ cat pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: trap
  labels:
    app: trap
spec:
  terminationGracePeriodSeconds: 10 # default is 30
  containers:
  - name: trap
    image: trap:exec
    imagePullPolicy: IfNotPresent
    command: ['./trap.sh']
    lifecycle:
      preStop:
        exec:
          command: ["echo", "Executando preStop Hook"]
```

Mais detalhes em https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3. **SIGTERM*** é enviado para todos containers do Pod
4. Kubernetes espera o **gracePeriod** (padrão é 30s)
5. *Se o pod ainda não terminou,* **SIGKILL** é enviado

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

1. Status **“Terminating”** e para de receber tráfego
2. **preStop Hook** é executado
3. **SIGTERM*** é enviado para todos containers do Pod
4. Kubernetes espera o **gracePeriod** (padrão é 30s)
5. *Se o pod ainda não terminou,* **SIGKILL** é enviado

Se você projetou um **container descartável** corretamente, você nunca deveria chegar no passo 5 (salvo em caso de erro).

Mais detalhes em https://kubernetes.io/docs/concepts/workloads/pods/pod/#termination-of-pods

The Twelve-Factor App

(A aplicação doze-fatores)

by Heroku c. 2011 · https://12factor.net/pt_br/

IX Descartabilidade

Os **processos** de uma aplicação doze-fatores são **descartáveis**: podem ser iniciados ou parados a qualquer momento.

Isso facilita o **escalonamento elástico**, as mudanças de configuração, e a **resiliência** do código em produção.

1

Sempre que possível trate *(handle)*
SIGTERM na sua aplicação.

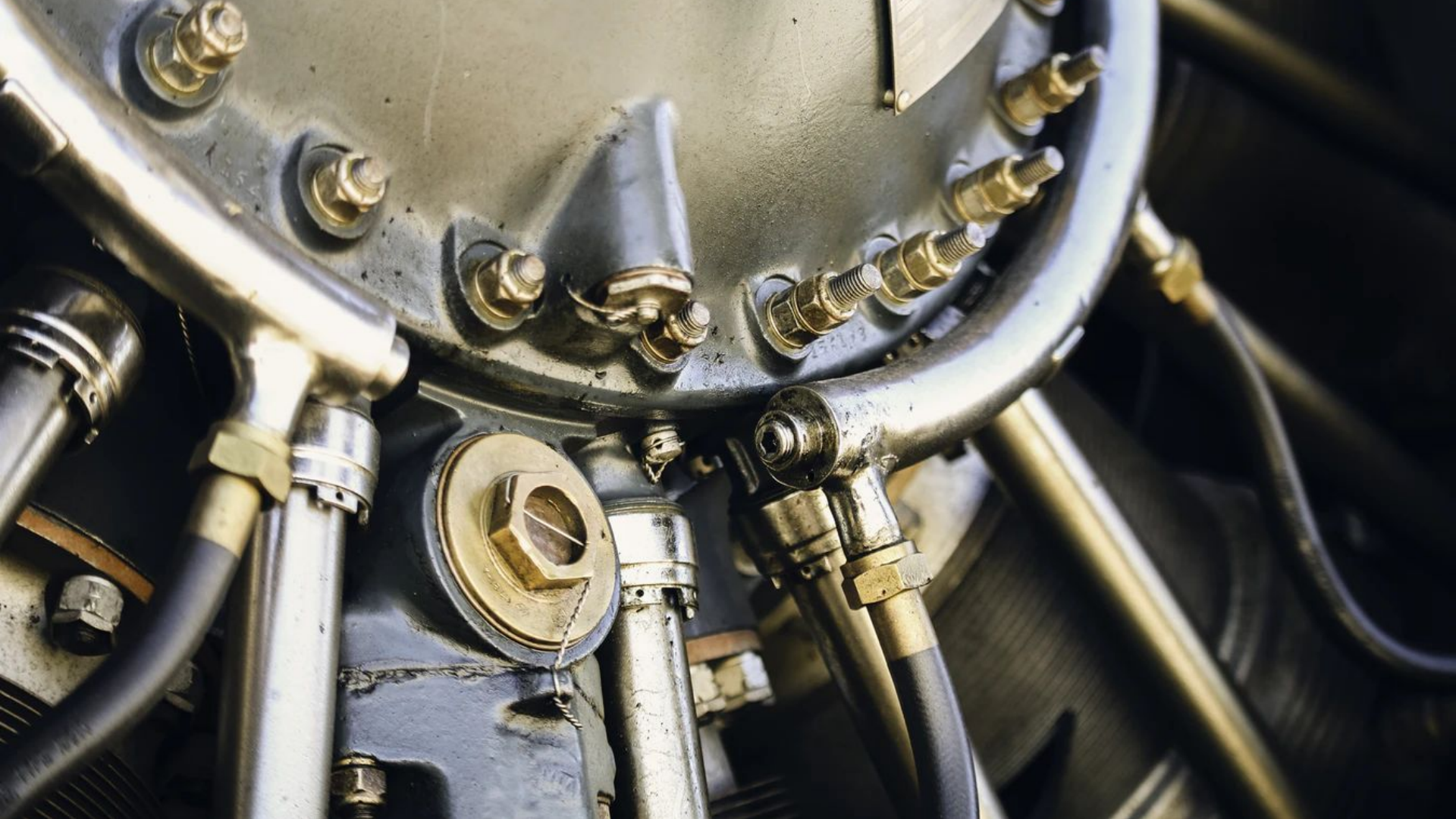
2

Em seus Dockerfiles, utilize sempre ENTRYPOINT na forma “exec”.

3

Teste suas configurações!

```
$ docker container stop ...
$ kubectl delete pod ...
```



TINI

Nem sempre podemos manipular o **código fonte** dos processos em nossos containers

https://github.com/krallin/tini

README.md

Tini - A tiny but valid `init` for containers

build passing

Tini is the simplest `init` you could think of.

All Tini does is spawn a single child (Tini is meant to be run in a container), and wait for it to exit all the while reaping zombies and performing signal forwarding.

Why Tini?

Using Tini has several benefits:

- It protects you from software that accidentally creates zombie processes, which can (over time!) starve your entire system for PIDs (and make it unusable).
- It ensures that the *default signal handlers* work for the software you run in your Docker image. For example, with Tini, `SIGTERM` properly terminates your process even if you didn't explicitly install a signal handler for it.
- It does so completely transparently! Docker images that work without Tini will work with Tini without any changes.

If you'd like more detail on why this is useful, review this issue discussion: What is advantage of Tini?.

https://github.com/krallin/tini

README.md

Tini - A tiny but valid `init` for containers

build passing

Tini is the simplest `init` you could think of.

All Tini does is spawn a single child (Tini is meant to be run in a container), and wait for it to exit all the while reaping zombies and performing signal forwarding.

Why Tini?

Using Tini has several benefits:

- It protects you from software that accidentally creates zombie processes, which can (over time!) starve your entire system for PIDs (and make it unusable).
- It ensures that the *default signal handlers* work for the software you run in your Docker image. For example, with Tini, `SIGTERM` properly terminates your process even if you didn't explicitly install a signal handler for it.
- It does so completely transparently! Docker images that work without Tini will work with Tini without any changes.

If you'd like more detail on why this is useful, review this issue discussion: What is advantage of Tini?.

```
FROM debian:stable

ADD trap.sh trap.sh

ENV TINI_VERSION v0.18.0
ADD https://github.com/krallin/tini/releases/download/${TINI_VERSION}/tini ./tini
RUN chmod +x ./tini

ENTRYPOINT ["./tini", "--"]
CMD ["./trap.sh"]
```

NODEJS

```
FROM node:latest

ADD package.json package.json
RUN npm install
ADD . .

ENTRYPOINT ["npm”]
CMD ["start”]
```

```
FROM node:latest

ADD package.json package.json
RUN npm install
ADD . .

ENTRYPOINT ["nodemon"]
CMD ["./server.js"]
```

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

```
FROM node:latest

ADD package.json package.json
RUN npm install
ADD . .

ENTRYPOINT ["node”]
CMD ["./server.js”]
```

*NPM, nodemon e outras ferramentas **em geral** não farão a gestão correta de SIGTERM, SIGINT e outros sinais. Apenas use o binário “node” diretamente.*

Um bom artigo sobre o tema em https://www.docker.com/blog/keep-nodejs-rockin-in-docker/

APACHE

Graceful Stop

Signal: WINCH

```
apachectl -k graceful-stop
```

The `WINCH` or `graceful-stop` signal causes the parent process to *advise* the children to exit after their current request (or to exit immediately if they're not serving anything). The parent will then remove its `PidFile` and cease listening on all ports. The parent will continue to run, and monitor children which are handling requests. Once all children have finalised and exited or the timeout specified by the `GracefulShutdownTimeout` has been reached, the parent will also exit. If the timeout is reached, any remaining children will be sent the `TERM` signal to force them to exit.

A `TERM` signal will immediately terminate the parent process and all children when in the "graceful" state. However as the `PidFile` will have been removed, you will not be able to use `apachectl` or `httpd` to send this signal.

O APACHE (http2) utiliza o sinal SIGWINCH (redimensionamento de janela) para *desligamento gracioso* e SIGTERM para desligamento rápido.

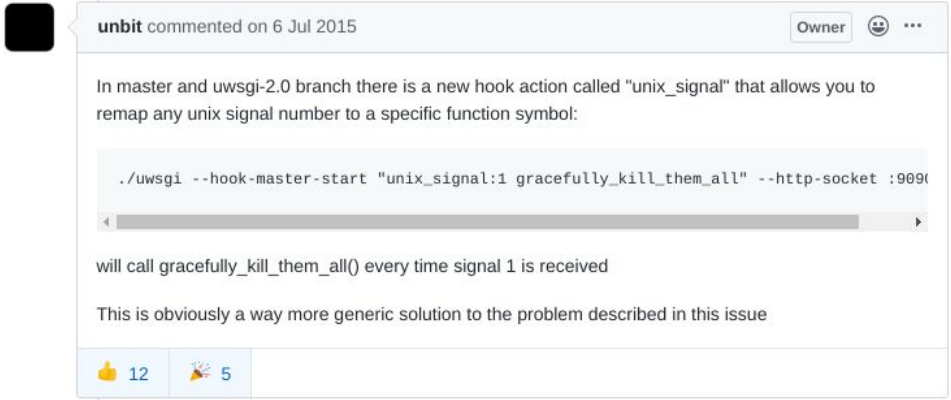
https://github.com/docker-library/php/blob/master/7.4/buster/apache/Dockerfile#L277

```
ENTRYPOINT ["docker-php-entrypoint"]

STOPSIGNAL SIGWINCH

COPY apache2-foreground /usr/local/bin/
WORKDIR /var/www/html

EXPOSE 80
CMD ["apache2-foreground"]
```

UWSGI

Signal	Description
SIGHUP	gracefully reload all the workers and the master process
SIGTERM	brutally reload all the workers and the master process
SIGINT	immediately kill the entire uWSGI stack
SIGQUIT	immediately kill the entire uWSGI stack
SIGUSR1	print statistics
SIGUSR2	print worker status or wakeup the spooler
SIGURG	restore a snapshot
SIGTSTP	pause/suspend/resume an instance
SIGWINCH	wakeup a worker blocked in a syscall (internal use)
SIGFPE	generate C traceback
SIGSEGV	generate C traceback

Fonte https://uwsgi-docs.readthedocs.io/en/latest/Management.html




```
./uwsgi --hook-master-start "unix_signal:1 gracefully_kill_them_all" --http-socket :9090 -M
```

< link >

Fonte https://uwsgi-docs.readthedocs.io/en/latest/Management.html


MUITO
OBRIGADO

// FERNANDO BARBOSA

SRE @ QuintoAndar

Programador YAML

Go | CI/CD | Kubernetes | Observability

> **@fernandrone**

> mail@fdr.one

> fernandrone.com

> fdr.one/sinais-code

> fdr.one/sinais-slides