

A million containers
isn't cool

You know what's
cool?

A hundred
containers

A million containers isn't cool

You know what's cool? A hundred containers.

@ChrisSinjo



GO CARDLESS

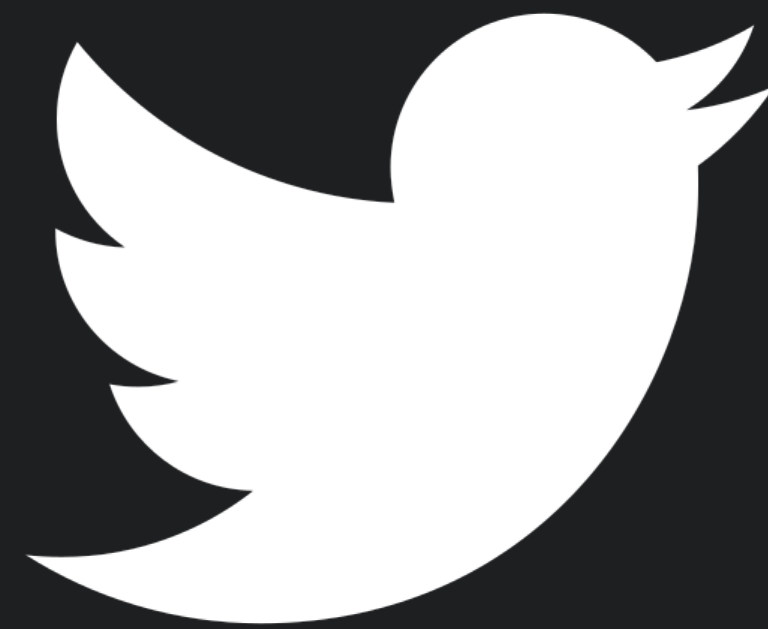
Google

Google

amazon

Google

amazon



We aren't
#webscale

(#sorrynotsorry)

So why do **we** care
about containers?

POST /cash/monies HTTP/1.1

{ amount: 100 }



High  per-request

Reliability is 

Deploying software reliably

Deploying software reliably

How containers can help

Deploying software reliably

How containers can help

Other options

First things first:
deployment artifacts

Source code



Something you can
put on a server

A .jar file

A statically linked binary

An OS package (.deb, .rpm)

Some languages
start on the back foot



Capistrano:
a *typical* Ruby flow

On each server:

On each server:

- Clone source

On each server:

- Clone source
- Build dependencies

On each server:

- Clone source
- Build dependencies
- Run schema migrations

On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets

On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets
- SIGHUP

What's wrong here?

Hope

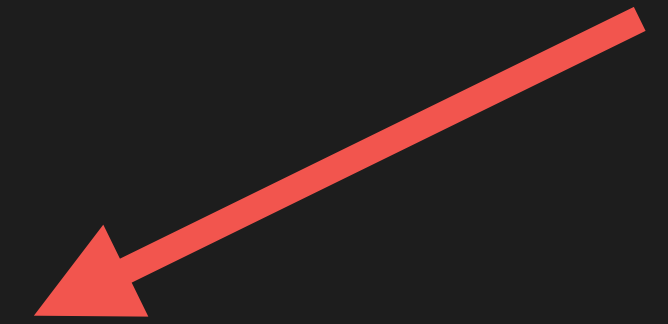
On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets
- SIGHUP

On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets
- SIGHUP

Hope




```
$ bundle install
```

```
...
```

```
Building nokogiri using system libraries.
```

```
Gem::Ext::BuildError: ERROR: Failed to build  
gem native extension.
```

On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets
- SIGHUP

Hope



On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets
- SIGHUP

Hope



Hope



On each server:

- Clone source
- Build dependencies
- Run schema migrations
- Build static assets
- SIGHUP

Hope



Hope



Hope



“Hope is not a strategy.”

– *Traditional SRE saying*

<https://landing.google.com/sre/book.html>

There's something
else

Applications

don't

run in a

vacuum

Ruby app

Ruby app



Ruby dependencies

Ruby app



Ruby dependencies



Native libraries

Ruby app



Ruby dependencies



Native libraries

Ruby app

```
graph TD; A[Ruby app] --> B[Nokogiri]; A --> C[Ruby dependencies]; B --> D[libxml2]; C --> E[Native libraries];
```

Nokogiri

Ruby dependencies

libxml2

Native libraries

Ruby app

```
graph TD; A[Ruby app] --> B[Nokogiri]; A --> C[Ruby dependencies]; B --> D[libxml2]; C --> E[Native libraries];
```

Nokogiri

Ruby dependencies

libxml2

Native libraries

Ruby app

```
graph TD; A[Ruby app] --> B[Nokogiri]; A --> C[Ruby dependencies]; B --> D[libxml2]; C --> E[Native libraries];
```

Nokogiri

Ruby dependencies

libxml2

Native libraries

How do we install
software?

Nokogiri



libxml2

Nokogiri



libxml2

```
$ bundle install
```

Nokogiri



libxml2

```
$ bundle install
```

```
$ apt-get install libxml2
```

Nokogiri

App's source repository



libxml2

Chef or whatever

That seems
inconvenient...

Container images:
totally a thing

Nokogiri

App's source repository



libxml2

Chef or whatever

Nokogiri



App's source repository

libxml2

App's source repository

This is why **most**
people care about
Docker

namespaces

cgroups

images

~~namespaces~~

~~egroups~~

images



Benji Weber

@benjiweber

Following



OH: @ChrisSinjo "Docker is a fat-jar for people not on the JVM"

RETWEETS

7

LIKES

11



6:06 PM - 29 Aug 2016

<https://twitter.com/benjiweber/status/770306615555854336>

Deploying software reliably

How containers can help

Other options

Deploying software reliably

How containers can help

Other options

So what did **we**
care about?

Uniform deployment

Uniform deployment

Based around an artifact

Uniform deployment

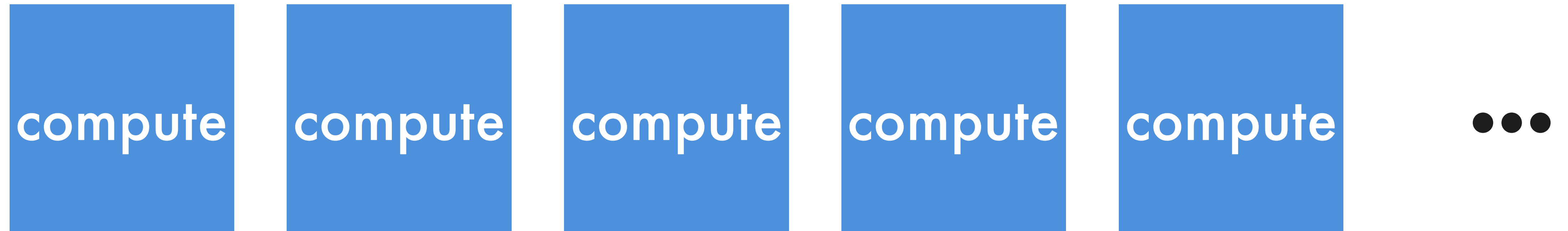
Based around an artifact

Fail early

And what didn't we
care about?

Know what your
aims aren't

Distributed schedulers



Scheduler

compute

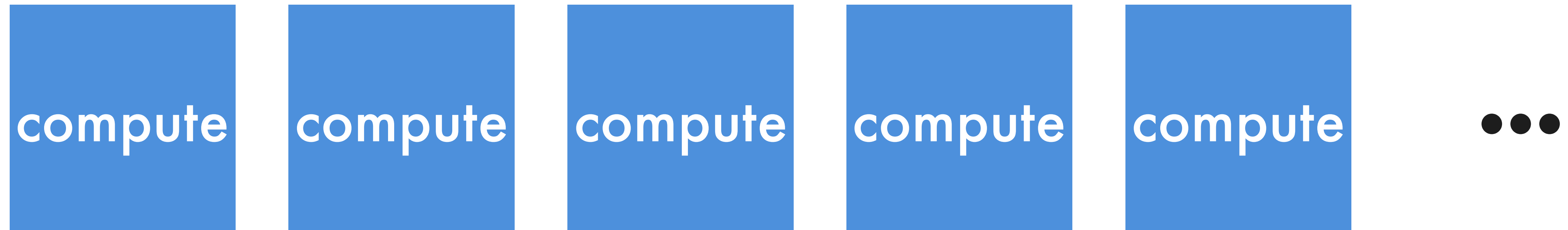
compute

compute

compute

compute

...



Scheduler

App

compute

App

compute

compute

compute

App

compute

...

Scheduler

compute

App
compute

compute

App
compute

App
compute

...

Nothing comes for
free

Kubernetes means:

Kubernetes means:

- a distributed scheduler

Kubernetes means:

- a distributed scheduler
- cluster DNS

Kubernetes means:

- a distributed scheduler
- cluster DNS
- etcd

Kubernetes means:

- a distributed scheduler
- cluster DNS
- etcd
- ...

Nothing comes for
free

We aren't
#webscale

(#sorrynotsorry)

Distributed schedulers

Distributed schedulers

So what did
we build?

3 parts...

Service definitions

A service:

A service:

— an image

A service:

- an image
- environment config

A service:

- an image
- environment config
- command to run

A service:

- an image
- environment config
- command to run
- limits (memory, CPU)

A service:

- an image
- environment config
- command to run
- limits (memory, CPU)
- ...

This is

config management

So we
used Chef

Chef

Service A

Service B

Service C

Chef

Service A

Service B

Service C

Compute 1

Compute 2

Compute 3

Chef

Service A

Service B

Service C

config

Compute 1

Service A

Service B

Compute 2

Compute 3

Chef

Service A

Service B

Service C

config

Compute 1

Service A

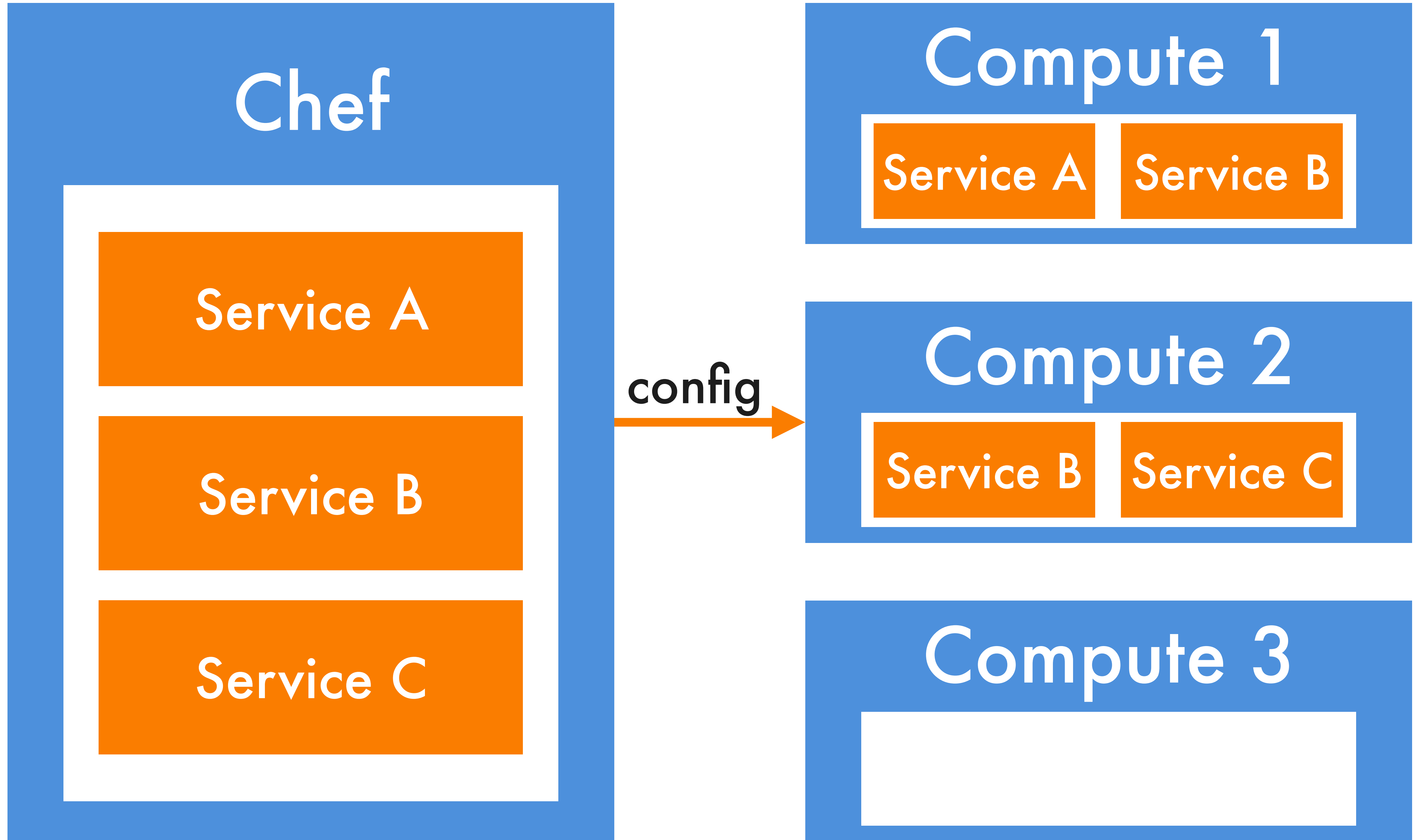
Service B

Compute 2

Service B

Service C

Compute 3



Chef

Service A

Service B

Service C

config

Compute 1

Service A

Service B

Compute 2

Service B

Service C

Compute 3

Service A

Service C

Chef

Service A

Service B

Service C

Compute 1

Service A

Service B

Compute 2

Service B

Service C

Compute 3

Service A

Service C

Service definitions

Service definitions

Single-node orchestration

Enter Conductor

```
conductor service upgrade  
  --id gocardless_app_production  
  --revision 279d903588
```



```
conductor service upgrade  
  --id gocardless_app_production  
  --revision 279d903588
```

```
conductor service upgrade  
  --id gocardless_app_production  
  --revision 279d903588
```

The flow:

The flow:

- start containers for new version

The flow:

- start containers for new version
- wait for health check

The flow:

- start containers for new version
- wait for health check
- rewrite local nginx config

The flow:

- start containers for new version
- wait for health check
- rewrite local nginx config
- reload nginx

The flow:

- start containers for new version
- wait for health check
- rewrite local nginx config
- reload nginx
- stop old containers

Conductor

nginx

Docker

Conductor

```
graph TD; Conductor[Conductor] --- nginx[nginx]; Conductor --- Old[Old]; Conductor --- Docker[Docker];
```

nginx

Old

Docker

Conductor

```
graph LR; Traffic[traffic] --> nginx; nginx -- traffic --> Old; Docker;
```

The diagram illustrates a system architecture. At the top is a large orange rectangle labeled 'Conductor'. Below it, on the left, is a blue rectangle labeled 'nginx'. Three orange arrows, each labeled 'traffic', point from the left towards the 'nginx' box. An orange arrow labeled 'traffic' points from the 'nginx' box to a smaller blue rectangle labeled 'Old'. To the right of the 'Old' box is another blue rectangle labeled 'Docker'.

traffic

nginx

traffic

Old

Docker

Conductor

```
graph TD; Traffic[traffic] --> nginx; nginx -- traffic --> Old; nginx --> New; Conductor[Conductor] -- API --> Docker;
```

The diagram illustrates a system architecture. At the top is an orange box labeled 'Conductor'. Below it, on the left, is a large blue box labeled 'nginx'. Three orange arrows labeled 'traffic' point from the left into the 'nginx' box. From the right side of the 'nginx' box, an orange arrow labeled 'traffic' points to a blue box labeled 'Old'. Above the 'Old' box is a red box labeled 'New'. To the right of the 'Old' and 'New' boxes is a large blue box labeled 'Docker'. An orange arrow labeled 'API' points from the bottom of the 'Conductor' box down to the top of the 'Docker' box.

New

nginx

traffic

Old

Docker

API

traffic

Conductor

```
graph TD; Conductor[Conductor] -- "health check" --> New[New]; nginx[nginx] -- "traffic" --> Old[Old]; Docker[Docker];
```

The diagram illustrates a system architecture for traffic management. At the top is an orange box labeled 'Conductor'. Below it, on the left, is a large blue box labeled 'nginx'. To the right of 'nginx' are two smaller blue boxes: 'New' (top) and 'Old' (bottom). Further to the right is a tall blue box labeled 'Docker'. An orange arrow labeled 'health check' points from the 'Conductor' box down to the 'New' box. Three orange arrows labeled 'traffic' point from the left into the 'nginx' box. An orange arrow labeled 'traffic' points from the 'nginx' box to the 'Old' box.

health check

New

nginx

Docker

traffic

Old

Conductor

```
graph TD; Conductor[Conductor] -- config --> nginx[nginx]; traffic1[traffic] --> nginx; traffic2[traffic] --> nginx; traffic3[traffic] --> nginx; nginx -- traffic --> Old[Old]; New[New]; Docker[Docker];
```

config

traffic

nginx

traffic

New

Old

Docker

Conductor

```
graph TD; Conductor[Conductor] -- reload --> nginx[nginx]; traffic1[traffic] --> nginx; traffic2[traffic] --> nginx; traffic3[traffic] --> nginx; nginx -- traffic --> New[New]; nginx --> Old[Old]; nginx --> Docker[Docker];
```

The diagram illustrates a traffic reload process. At the top is an orange box labeled 'Conductor'. Below it is a large blue box labeled 'nginx'. Three orange arrows labeled 'traffic' point from the left into the 'nginx' box. An orange arrow labeled 'reload' points from the 'Conductor' box down to the 'nginx' box. From the right side of the 'nginx' box, an orange arrow labeled 'traffic' points to a green box labeled 'New'. Below the 'New' box is a blue box labeled 'Old'. To the right of the 'New' and 'Old' boxes is a tall blue box labeled 'Docker'.

reload

traffic

nginx

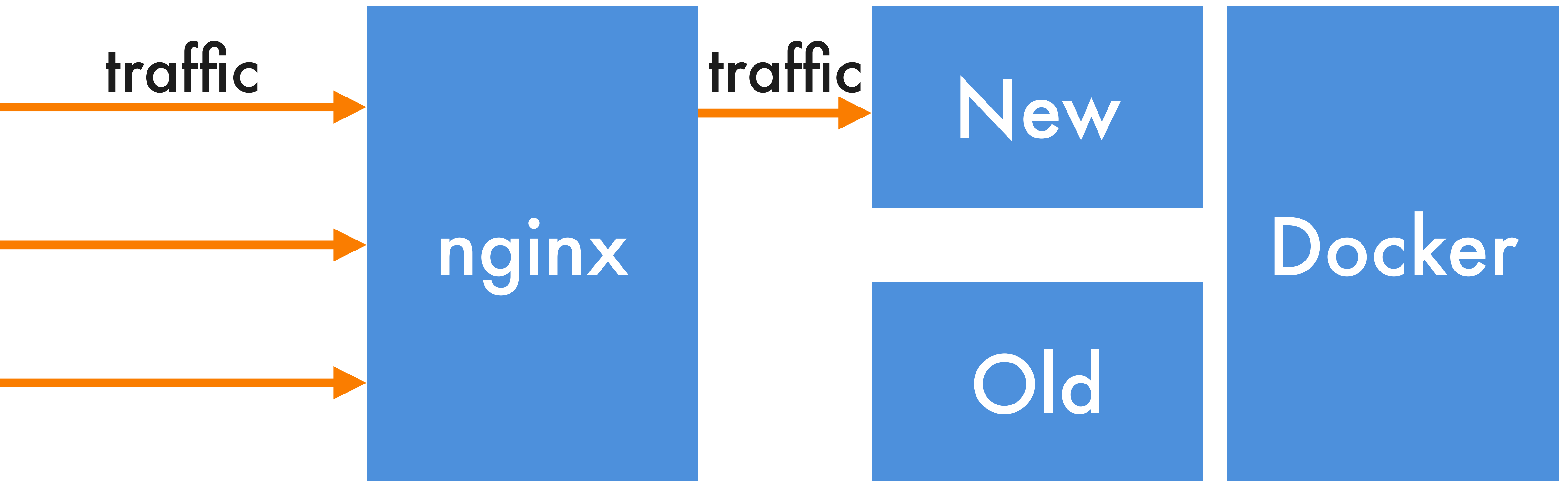
traffic

New

Old

Docker

Conductor



Conductor

```
graph TD; Traffic[traffic] --> nginx; nginx -- traffic --> New; nginx --> Old; Conductor[Conductor] -- API --> Docker;
```

The diagram illustrates a system architecture. At the top is a large orange rectangle labeled 'Conductor'. Below it, on the left, is a large blue rectangle labeled 'nginx'. Three orange arrows, each labeled 'traffic', point from the left towards the 'nginx' box. An orange arrow labeled 'traffic' points from the right side of the 'nginx' box to a blue rectangle labeled 'New'. Below the 'New' box is a red rectangle labeled 'Old'. To the right of the 'New' and 'Old' boxes is a large blue rectangle labeled 'Docker'. An orange arrow labeled 'API' points from the bottom of the 'Conductor' box down to the top of the 'Docker' box.

traffic

nginx

traffic

New

Old

Docker

API

```
graph LR; Traffic[traffic] --> nginx; nginx -- traffic --> New; Conductor -- API --> Docker;
```

Conductor

traffic

nginx

traffic

New

API

Docker

Conductor

```
graph LR; Traffic[traffic] --> nginx; nginx -- traffic --> New; Docker;
```

The diagram illustrates a system architecture. At the top is a large orange rectangle labeled 'Conductor'. Below it, on the left, is a blue rectangle labeled 'nginx'. Three orange arrows, each labeled 'traffic', point from the left towards the 'nginx' box. An orange arrow labeled 'traffic' points from the 'nginx' box to a blue rectangle labeled 'New'. To the right of the 'New' box is another blue rectangle labeled 'Docker'.

traffic

nginx

traffic

New

Docker

What about
cron jobs?

```
conductor cron generate  
  --id gocardless_cron_production  
  --revision 279d903588
```

```
conductor cron generate  
  --id gocardless_cron_production  
  --revision 279d903588
```

gocardless/

▼ app/

payment_stuff.rb

► lib/

generate-cron

```
# Clean up expired API tokens
```

```
*/30 * * * * scripts/cleanup-api-tokens
```



```
# Clean up expired API tokens
```

```
*/30 * * * * /usr/local/bin/conductor run  
  --id gocardless_cron_production  
  --revision 279d903588  
scripts/cleanup-api-tokens
```

Service definitions

Single-node orchestration

Service definitions

Single-node orchestration

A way to trigger deploys

Keep it
boring

Keep it

in Capistrano

Capistrano

deploy

Legacy
infra



Capistrano

```
graph TD; Capistrano[Capistrano] -- deploy --> Legacy[Legacy infra]; Capistrano -- deploy --> New[New infra];
```

The diagram illustrates the deployment process using Capistrano. At the top, an orange box labeled 'Capistrano' represents the deployment tool. Two orange arrows, each labeled 'deploy', point from the 'Capistrano' box to two separate blue boxes below. The left blue box is labeled 'Legacy infra' and the right blue box is labeled 'New infra', representing the target infrastructure environments.

deploy

Legacy
infra

deploy

New
infra

Help developers
do their job



1 thing
missing

“Hey, this process died.”

– a computer

```
graph TD; Supervisor[Supervisor] --- Process1[Process]; Supervisor --- Process2[Process]; Supervisor --- Process3[Process];
```

Supervisor

Process

Process

Process

```
graph TD; Supervisor[Supervisor] --- Process1[Process]; Supervisor --- Process2[Process]; Supervisor --- Process3[Process];
```

Supervisor

Process

Process

Process

Supervisor



Process

Process

Process

Supervisor

```
graph TD; Supervisor[Supervisor] -- start --> Process1[Process]; Process1[Process]; Process2[Process]; Process3[Process];
```

The diagram illustrates a supervisor process managing multiple worker processes. At the top is an orange rectangular box labeled 'Supervisor'. Below it, an orange arrow points downwards to the top of a green rectangular box labeled 'Process'. To the right of the arrow, the word 'start' is written in bold black text. Below the green box are two blue rectangular boxes, both labeled 'Process' in white text. The entire diagram is set against a light gray background.

start

Process

Process

Process

Some supervisors:

Some supervisors:

- Upstart

Some supervisors:

- Upstart

- *systemd*

Some supervisors:

- Upstart
- systemd
- runit

Those **didn't**
play well with
Docker

Docker restart policies

We didn't get
along well

Hard to stop

Hard to stop

or

Gave up entirely

We built a
process supervisor 🤖

conductor supervise

Specifically:

Specifically:

- check number of containers

Specifically:

- check number of containers
- health check each container

Specifically:

- check number of containers
- health check each container

Specifically:

- check number of containers
- health check each container
- restart if either fails

Specifically:

- check number of containers
- health check each container
- restart if either fails
- at most every 5 seconds


```
# service conductor-supervise stop
```

We don't want this
piece of software



Deploying software reliably

How containers can help

Other options

Deploying software reliably

How containers can help

Other options

systemd + rkt

Supervisor: *systemd*

Supervisor: *systemd*

Containers: *rkt*

To fit our usage:

To fit our usage:

- Conductor generates *systemd* config

To fit our usage:

- Conductor generates systemd config
- systemd manages processes

To fit our usage:

- Conductor generates systemd config
- systemd manages processes
- Delete conductor supervise

To fit our usage:

- Conductor generates systemd config
- systemd manages processes
- Delete conductor supervise
- HTTP health checks???

systemd + rkt

or

VMs + autoscaling

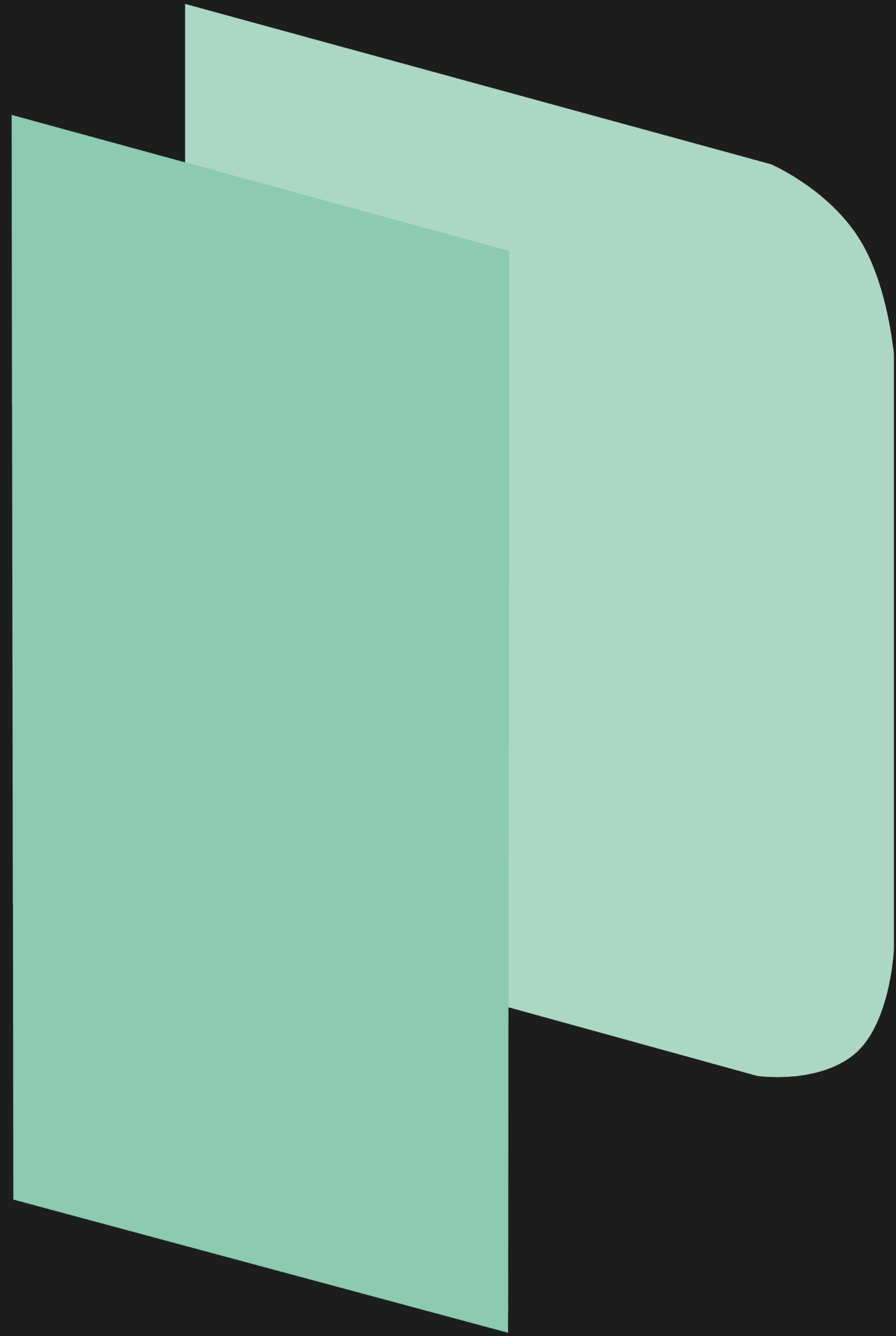
Supervisor: autoscaling

Supervisor: autoscaling

Containers → VMs

Google

amazon



Meta-thoughts

~~Meta-thoughts~~

Some reckons



Introduce
new infrastructure
where failure
is survivable

Non-critical batch jobs



Background workers



API servers

Goal state is
what matters

Everything might
change before your
next method call

The system isn't

interesting

without context

Start with why

Thank you



@ChrisSinjo

@GoCardlessEng

We're hiring



@ChrisSinjo

@GoCardlessEng

Questions?



@ChrisSinjo
@GoCardlessEng