WTF is culture in cloud native?

Holly Cummins
IBM Garage
@holly_cummins

IBM

WTF, culture **IS** cloud native!

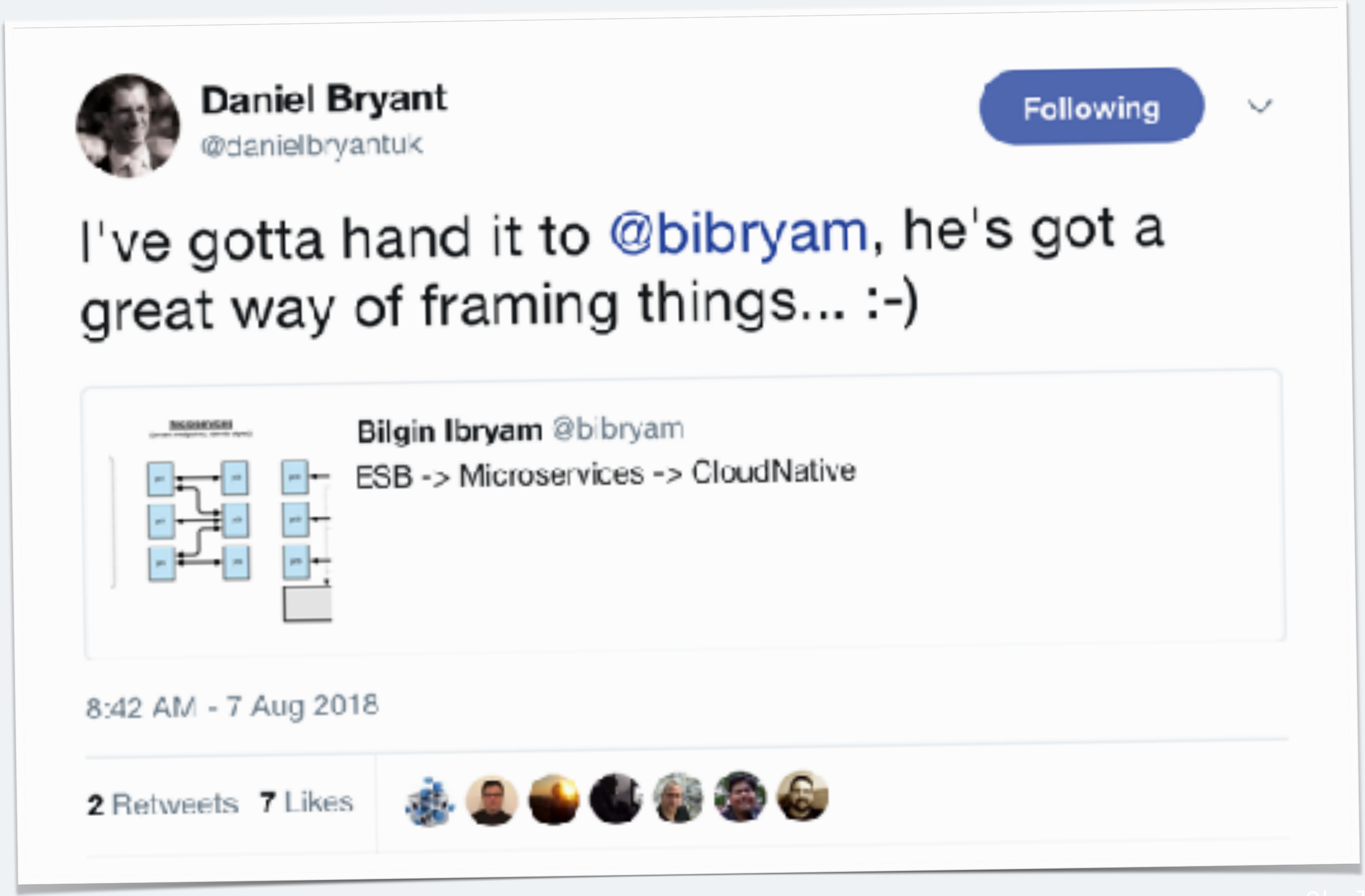
Holly Cummins
IBM Garage
@holly_cummins

WTF, culture **IS** cloud native!

(fixed it for you)

Holly Cummins
IBM Garage
@holly_cummins

what **is** cloud native?



Daniel Bryant
@danielbryantuk

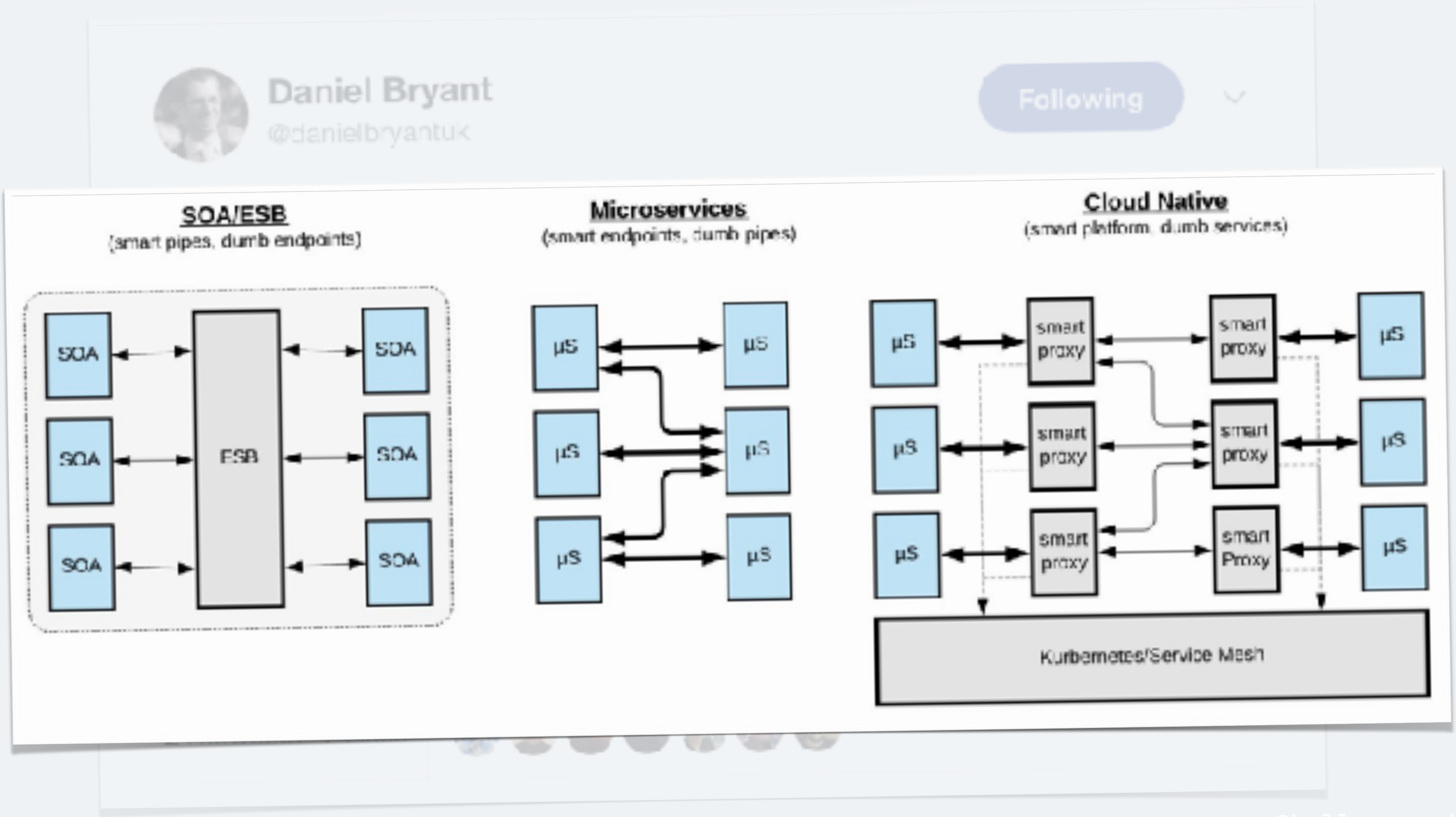
I've gotta hand it to @bibryam, he's got a great way of framing things... :-)

Bilgin Ibryam @bibryam
ESB -> Microservices -> CloudNative

8:42 AM - 7 Aug 2018

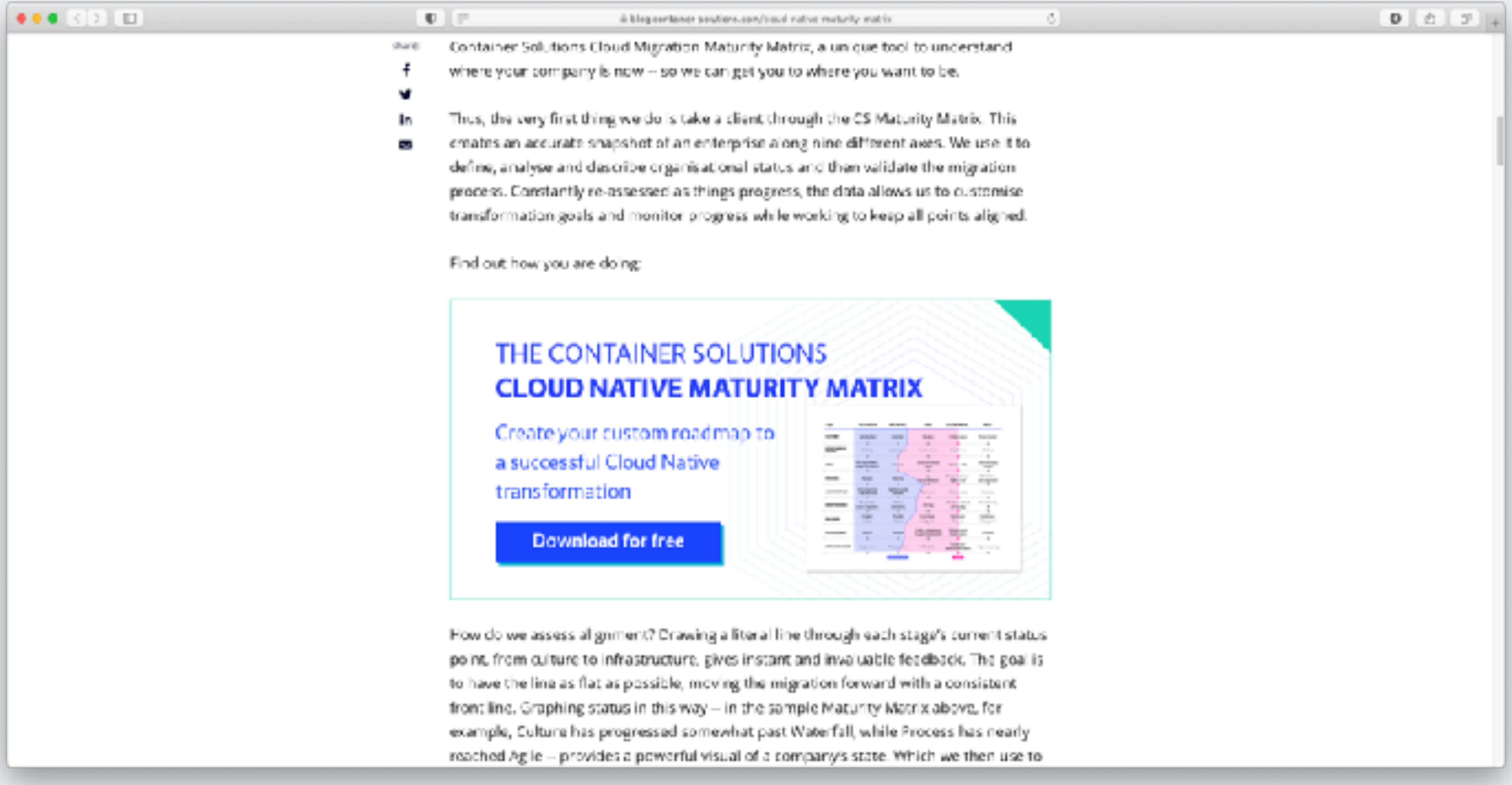
2 Retweets 7 Likes

Credit to @bibryam



Daniel Bryant

@danielbryantuk



Container Solutions Cloud Migration Maturity Matrix, a unique tool to understand where your company is now – so we can get you to where you want to be.

Thus, the very first thing we do is take a client through the CS Maturity Matrix. This creates an accurate snapshot of an enterprise along nine different axes. We use it to define, analyse and describe organisational status and then validate the migration process. Constantly re-assessed as things progress, the data allows us to customise transformation goals and monitor progress while working to keep all points aligned.

Find out how you are doing:

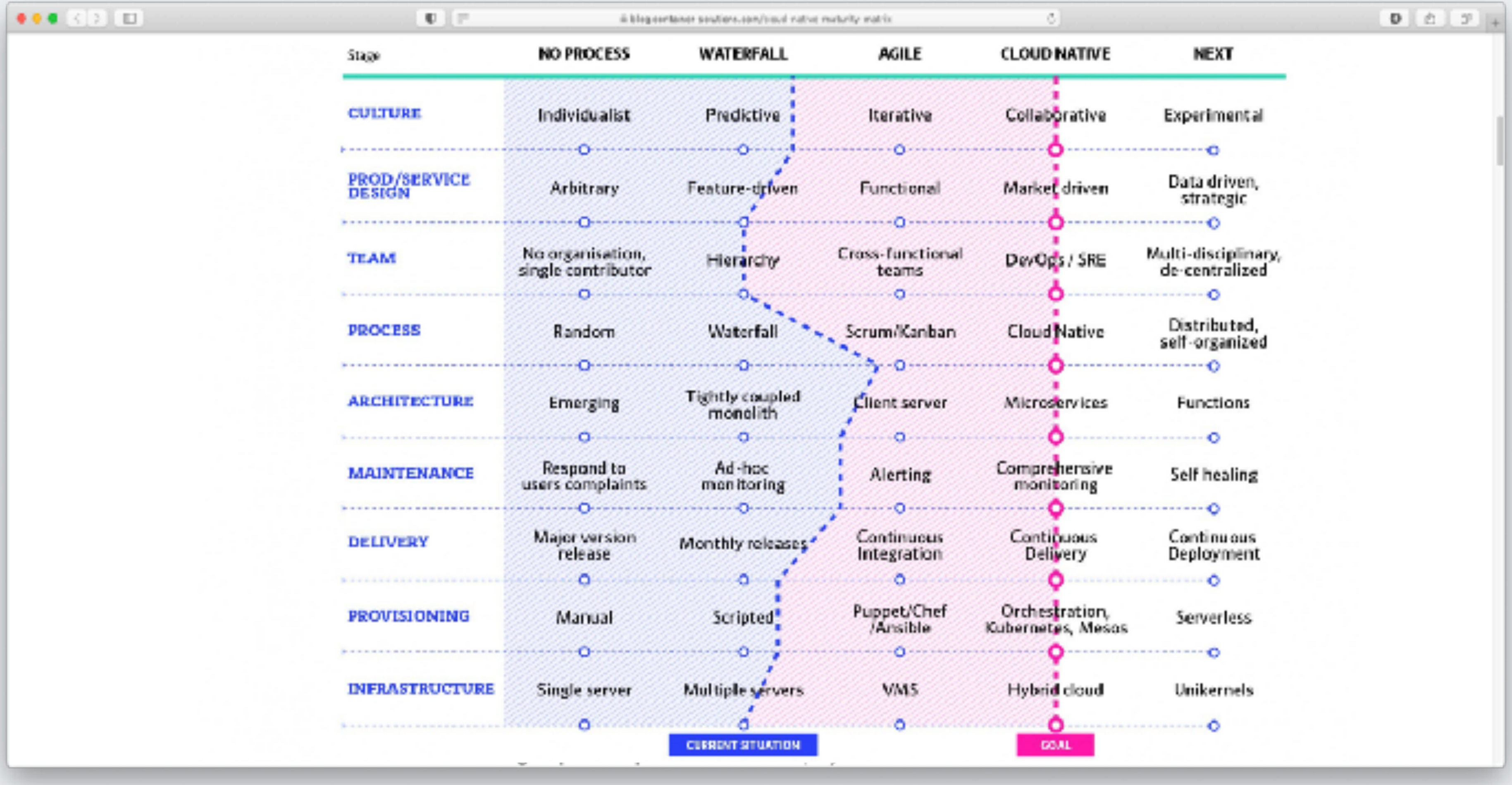
THE CONTAINER SOLUTIONS

CLOUD NATIVE MATURITY MATRIX

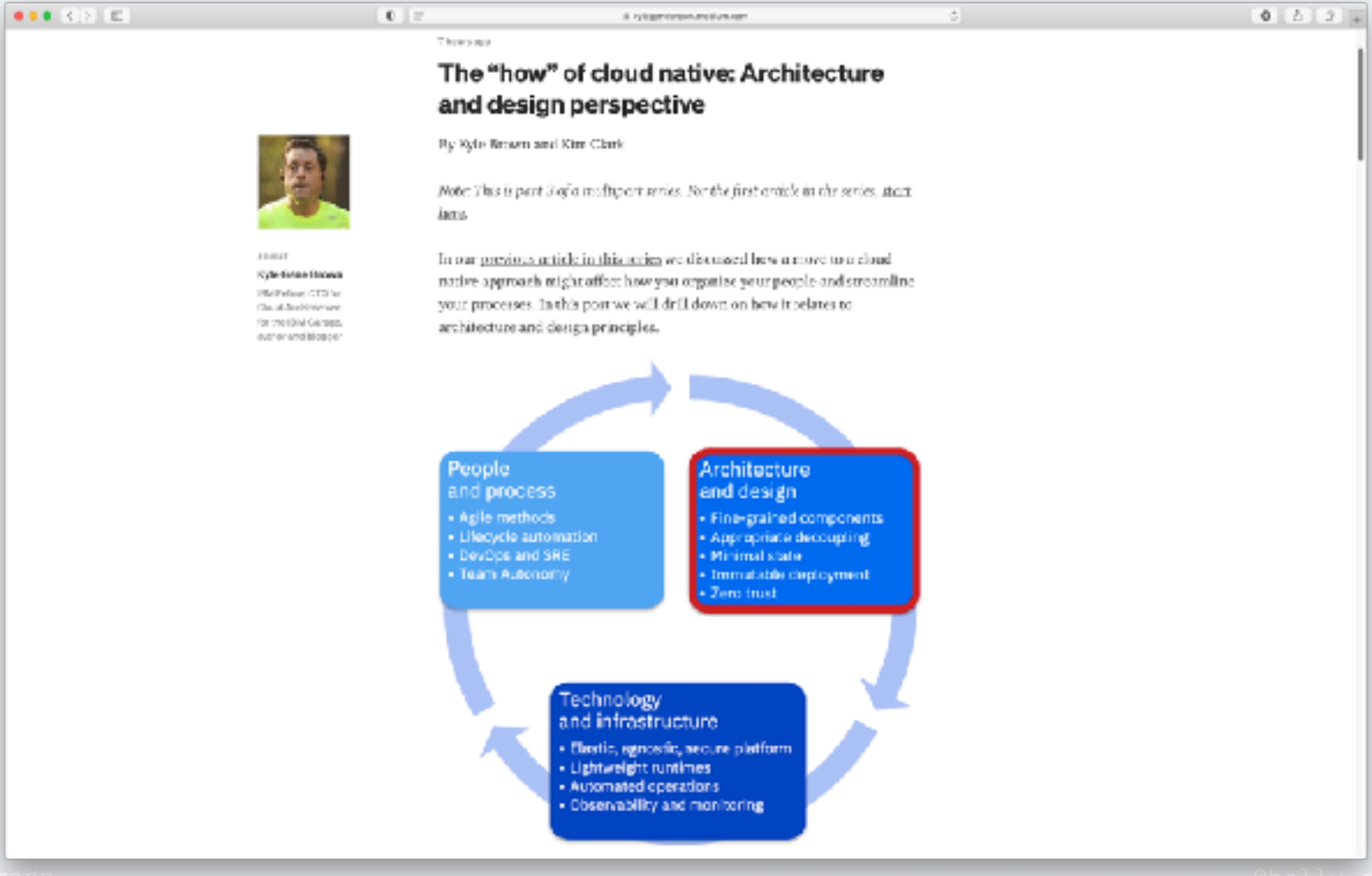
Create your custom roadmap to a successful Cloud Native transformation

Download for free

How do we assess alignment? Drawing a literal line through each stage's current status point, from culture to infrastructure, gives instant and invaluable feedback. The goal is to have the line as flat as possible, moving the migration forward with a consistent front line. Graphing status in this way – in the sample Maturity Matrix above, for example, Culture has progressed somewhat past Waterfall, while Process has nearly reached Agile – provides a powerful visual of a company's state. Which we then use to



Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
CULTURE	Individualist	Predictive	Iterative	Collaborative	Experimental
PROD/SERVICE DESIGN	Arbitrary	Feature-driven	Functional	Market driven	Data driven, strategic
TEAM	No organisation, single contributor	Hierarchy	Cross-functional teams	DevOps / SRE	Multi-disciplinary, de-centralized
PROCESS	Random	Waterfall	Scrum/Kanban	Cloud Native	Distributed, self-organized
ARCHITECTURE	Emerging	Tightly coupled monolith	Client server	Microservices	Functions
MAINTENANCE	Respond to users complaints	Ad-hoc monitoring	Alerting	Comprehensive monitoring	Self healing
DELIVERY	Major version release	Monthly releases	Continuous Integration	Continuous Delivery	Continuous Deployment
PROVISIONING	Manual	Scripted	Puppet/Chef /Ansible	Orchestration, Kubernetes, Mesos	Serverless
INFRASTRUCTURE	Single server	Multiple servers	VMS	Hybrid cloud	Unikernels



7 hours ago

The “how” of cloud native: Architecture and design perspective

By Kyle Brown and Kim Clark

Note: This is part 3 of a multipart series. For the first article in the series, start here.

In our previous article in this series we discussed how a move to a cloud native approach might affect how you organize your people and streamline your processes. In this post we will drill down on how it relates to architecture and design principles.

ABOUT

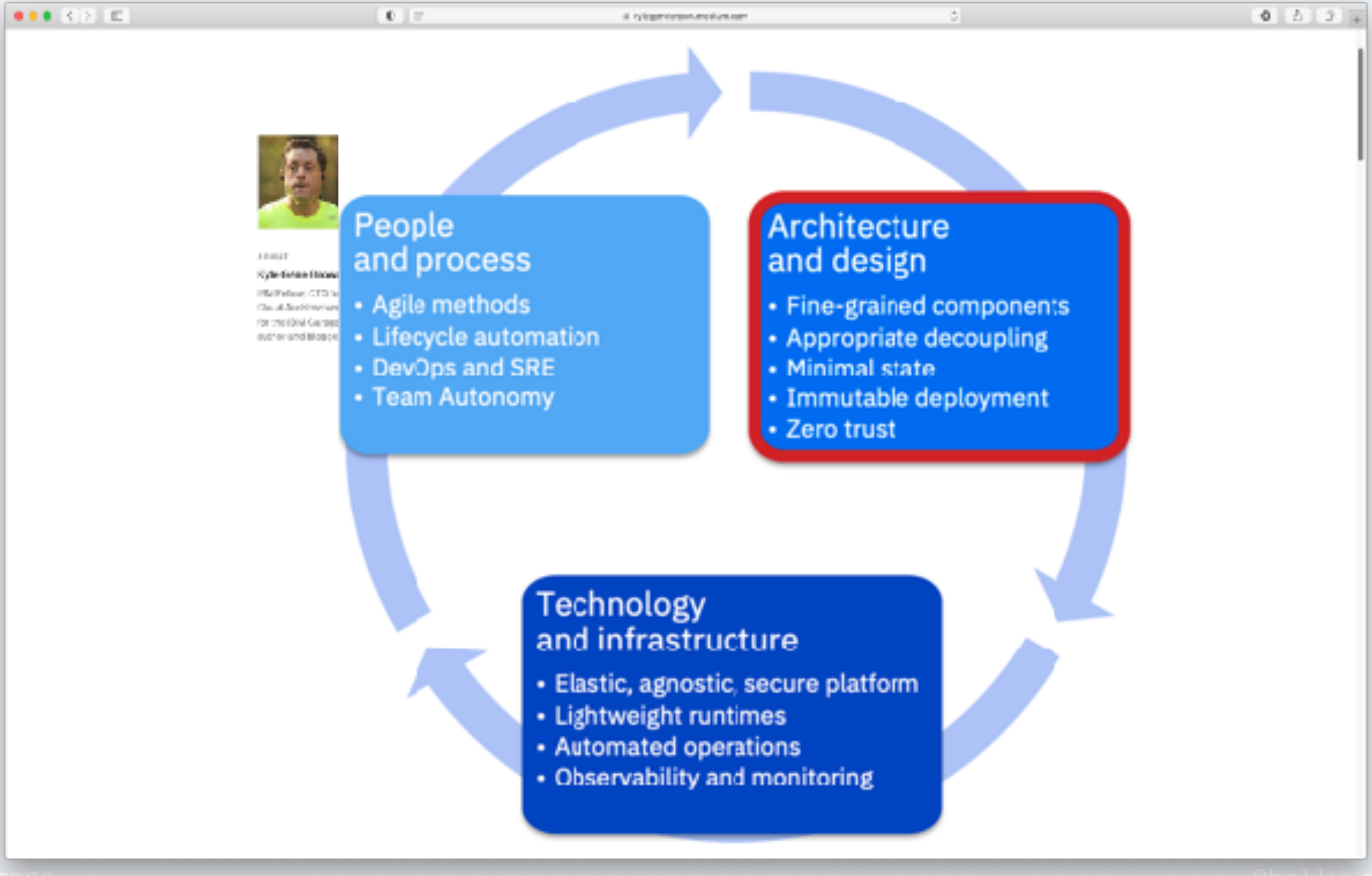
Kyle Gene Brown

IBM Fellow, CTO for Cloud Architecture for the IBM Garage, author and blogger

People and process
- Agile methods
- Lifecycle automation
- DevOps and SRE
- Team Autonomy

Architecture and design
- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment
- Zero trust

Technology and infrastructure
- Elastic, agnostic, secure platform
- Lightweight runtimes
- Automated operations
- Observability and monitoring



People and process

- Agile methods
- Lifecycle automation
- DevOps and SRE
- Team Autonomy

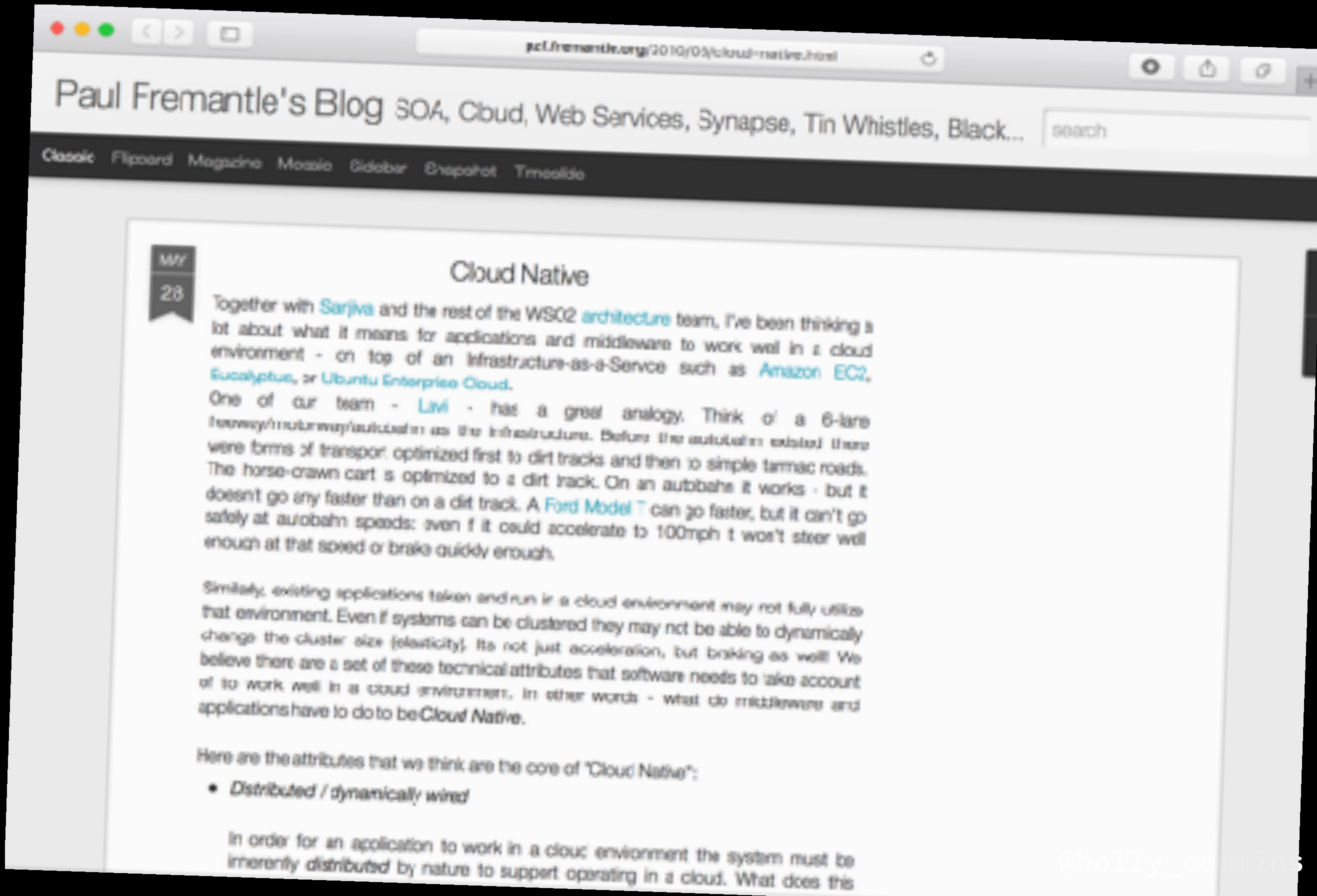
Architecture and design

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment
- Zero trust

Technology and infrastructure

- Elastic, agnostic, secure platform
- Lightweight runtimes
- Automated operations
- Observability and monitoring

2010
the dawn of cloud native



pzf.fremantle.org/2010/05/cloud-native.html

Paul Fremantle's Blog SOA, Cloud, Web Services, Synapse, Tin Whistles, Black...

Cloud Native

Together with Sanjiva and the rest of the WSO2 architecture team, I've been thinking a lot about what it means for applications and middleware to work well in a cloud environment - on top of an Infrastructure-as-a-Service such as Amazon EC2, Eucalyptus, or Ubuntu Enterprise Cloud.

One of our team - Lavi - has a great analogy. Think of a 6-lane freeway/motorway/autobahn as the infrastructure. Before the autobahn existed there were forms of transport optimized first to dirt tracks and then to simple tarmac roads. The horse-drawn cart is optimized to a dirt track. On an autobahn it works - but it doesn't go any faster than on a dirt track. A Ford Model T can go faster, but it can't go safely at autobahn speeds: even if it could accelerate to 100mph it won't steer well enough at that speed or brake quickly enough.

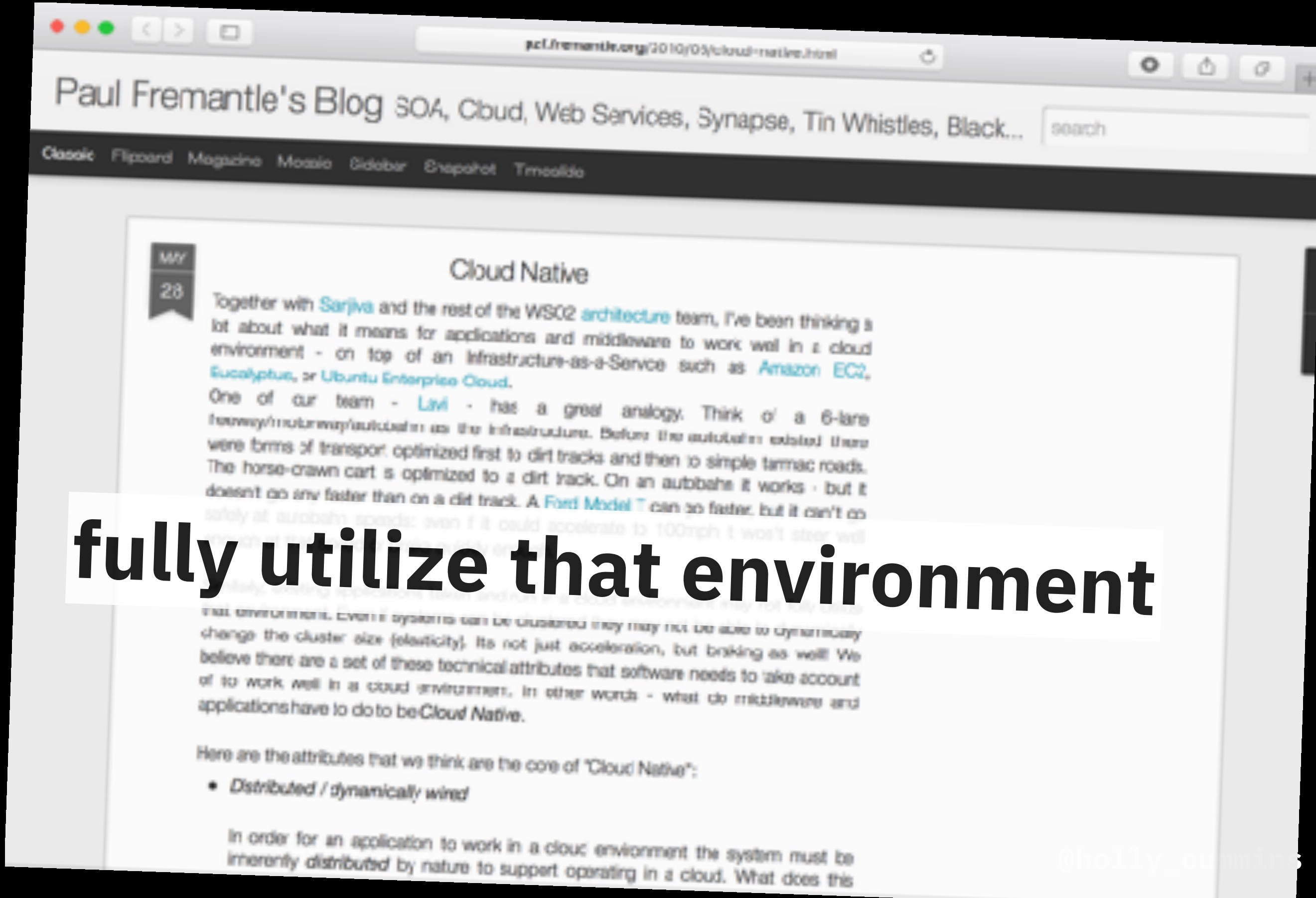
Similarly, existing applications taken and run in a cloud environment may not fully utilize that environment. Even if systems can be clustered they may not be able to dynamically change the cluster size (elasticity). Its not just acceleration, but braking as well! We believe there are a set of these technical attributes that software needs to take account of to work well in a cloud environment. In other words - what do middleware and applications have to do to be *Cloud Native*.

Here are the attributes that we think are the core of "Cloud Native":

- *Distributed / dynamically wired*

In order for an application to work in a cloud environment the system must be inherently *distributed* by nature to support operating in a cloud. What does this

2010 the dawn of cloud native


pcf.fremantle.org/2010/05/cloud-native.html
Paul Fremantle's Blog SOA, Cloud, Web Services, Synapse, Tin Whistles, Black...
search
Classic
Flipcard
Magazine
Mosaic
Sidebar
Snapshot
Timeslide
MAY
28

Cloud Native

Together with Sanjiva and the rest of the WSO2 architecture team, I've been thinking a lot about what it means for applications and middleware to work well in a cloud environment - on top of an Infrastructure-as-a-Service such as Amazon EC2, Eucalyptus, or Ubuntu Enterprise Cloud.

One of our team - Lavi - has a great analogy. Think of a 6-lane freeway/motorway/autobahn as the infrastructure. Before the autobahn existed there were forms of transport optimized first to dirt tracks and then to simple tarmac roads. The horse-drawn cart is optimized to a dirt track. On an autobahn it works - but it doesn't go any faster than on a dirt track. A Ford Model T can go faster, but it can't go safely at autobahn speeds: even if it could accelerate to 100mph it won't steer well enough at that speed or brake quickly enough.

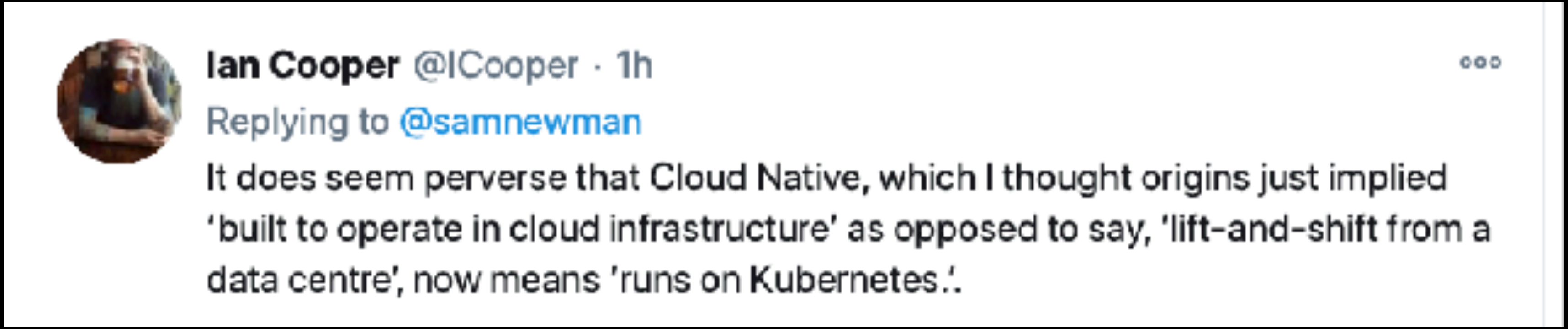
Similarly, existing applications taken and run in a cloud environment may not fully utilize that environment. Even if systems can be clustered they may not be able to dynamically change the cluster size (elasticity). Its not just acceleration, but braking as well! We believe there are a set of these technical attributes that software needs to take account of to work well in a cloud environment. In other words - what do middleware and applications have to do to be *Cloud Native*.

Here are the attributes that we think are the core of "Cloud Native":

- *Distributed / dynamically wired*

  In order for an application to work in a cloud environment the system must be inherently *distributed* by nature to support operating in a cloud. What does this

fully utilize that environment



Ian Cooper @ICooper · 1h

Replying to @samnewman

It does seem perverse that Cloud Native, which I thought origins just implied ‘built to operate in cloud infrastructure’ as opposed to say, 'lift-and-shift from a data centre’, now means ’runs on Kubernetes.’.

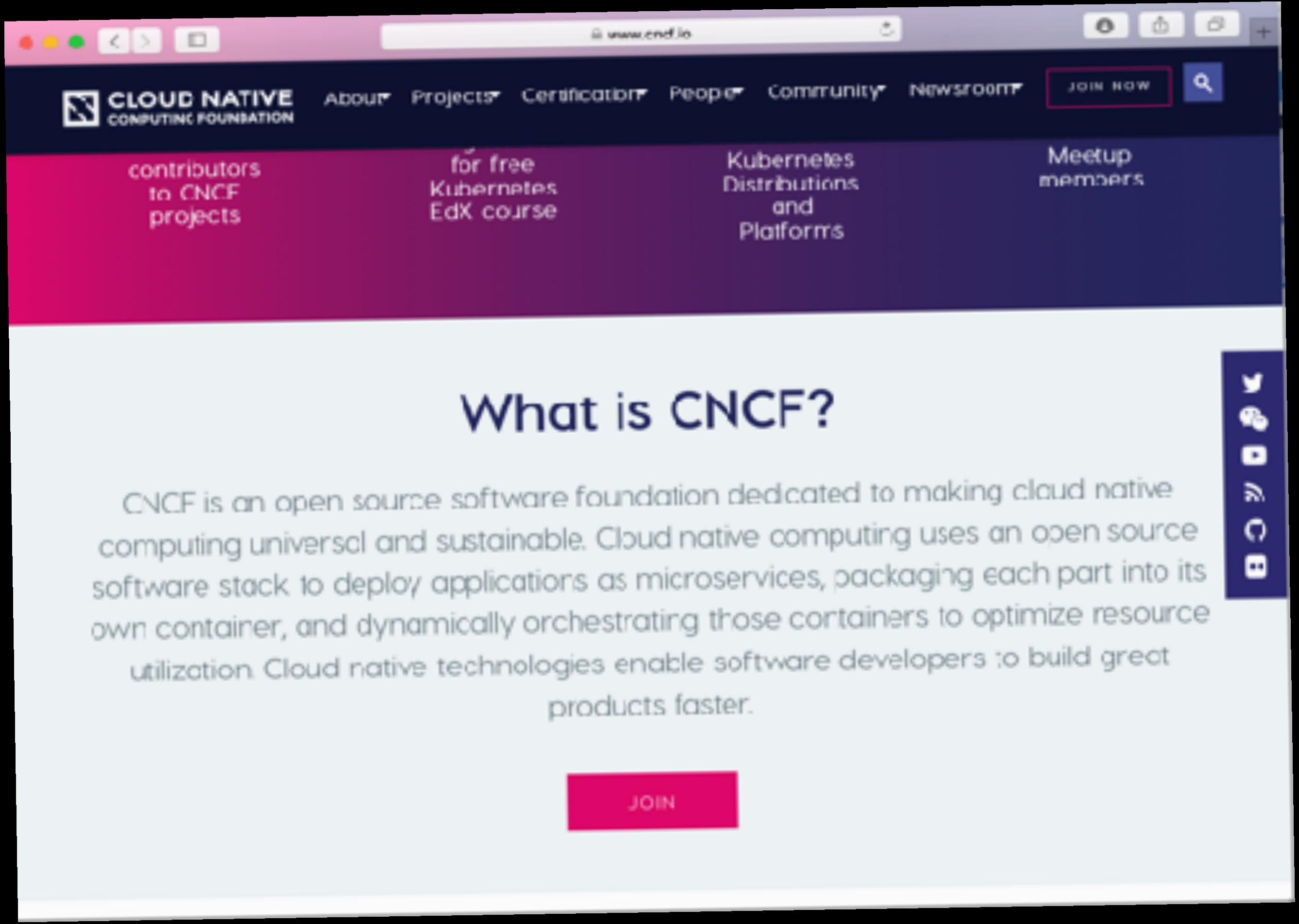
Justin Cormack @justincormack · 10m

The fog has remained. Are we cloud native now?

Justin Cormack @justincormack · Dec 6

Its foggy again

Show this thread



contributors to CNCF projects

for free Kubernetes EdX course

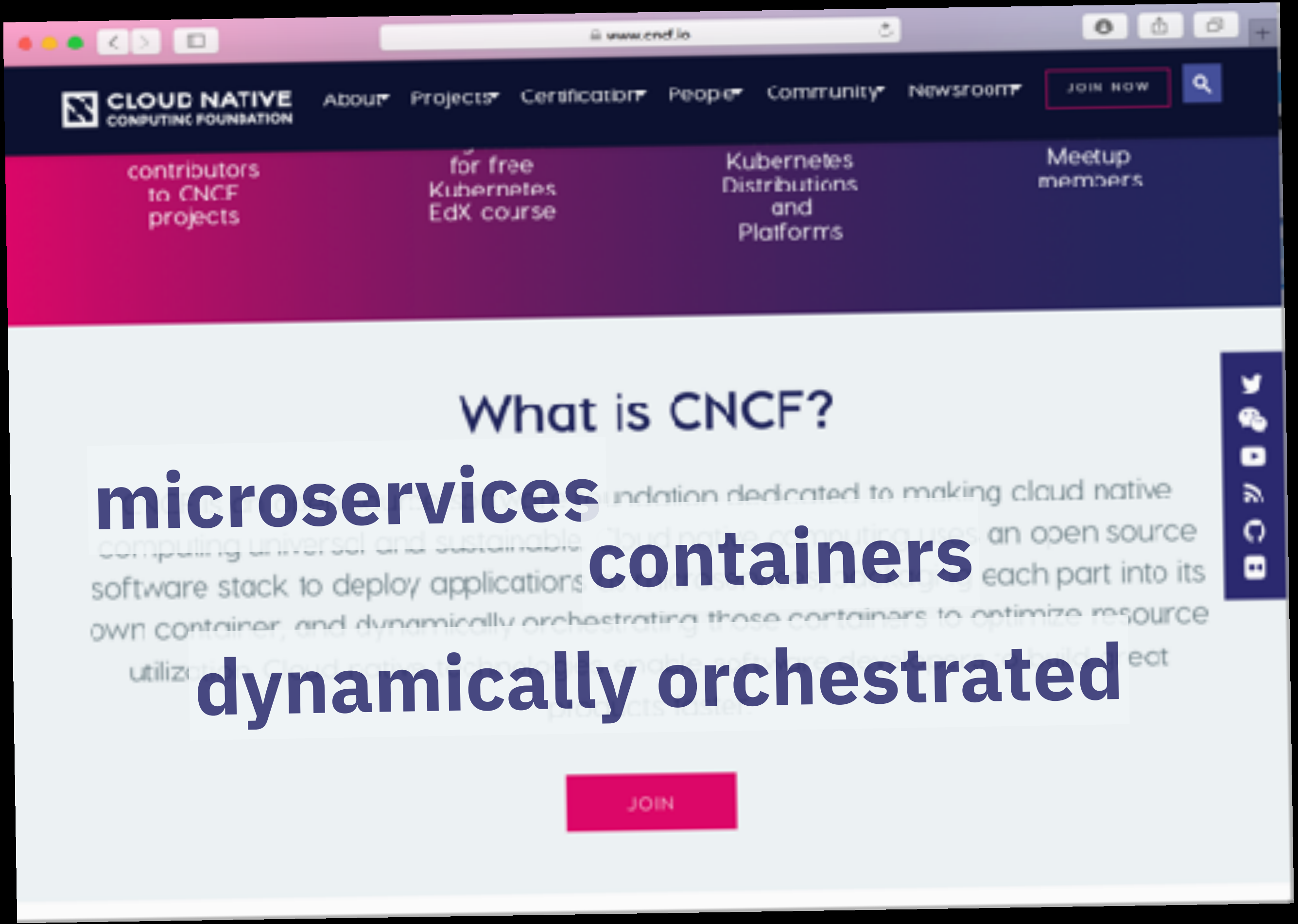
Kubernetes Distributions and Platforms

Meetup members

What is CNCF?

CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.

2019



contributors to CNCF projects

for free Kubernetes EdX course

Kubernetes Distributions and Platforms

Meetup members

What is CNCF?

…undation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications … each part into its own container, and dynamically orchestrating those containers to optimize resource utiliz… …ect … products faster.

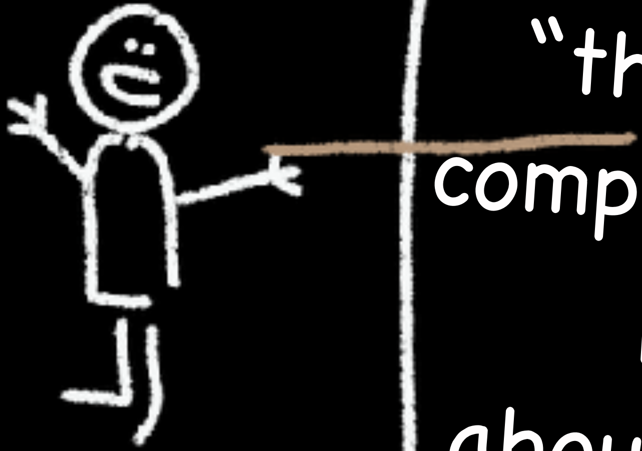
microservices

containers

dynamically orchestrated

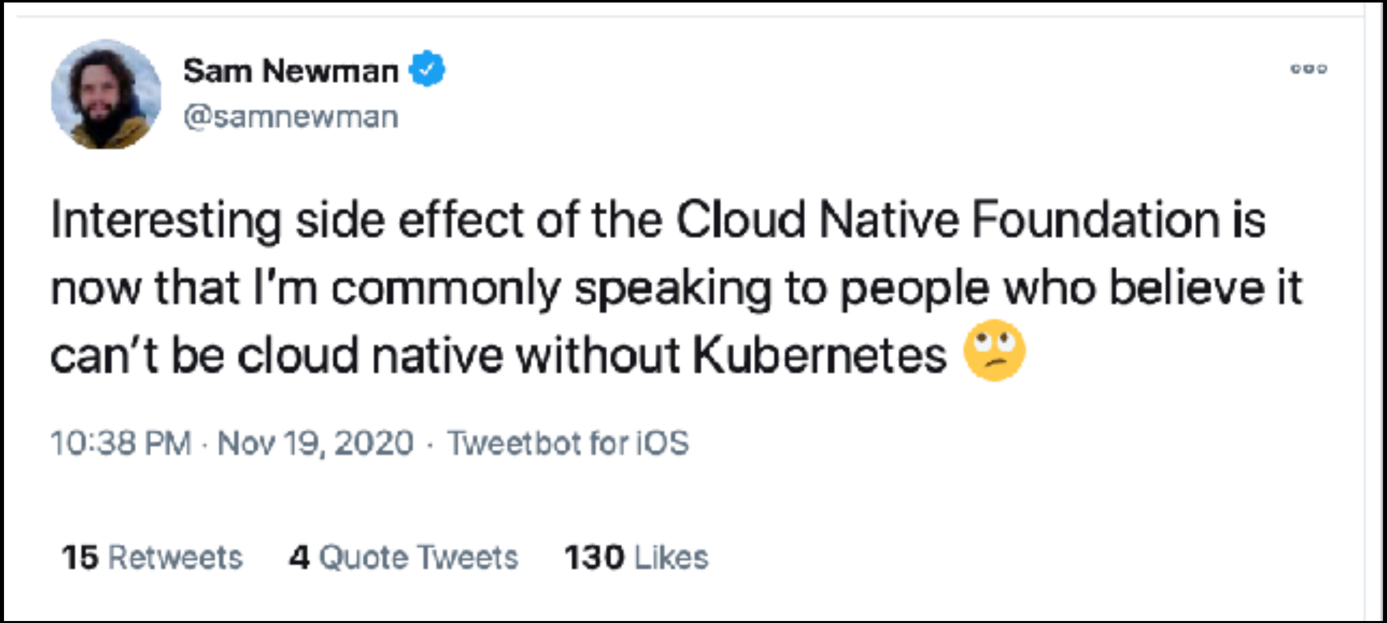
2019

WTF is CLOUD NATIVE



Dr Holly

“the cloud native computing foundation is **wrong** … about cloud native.”



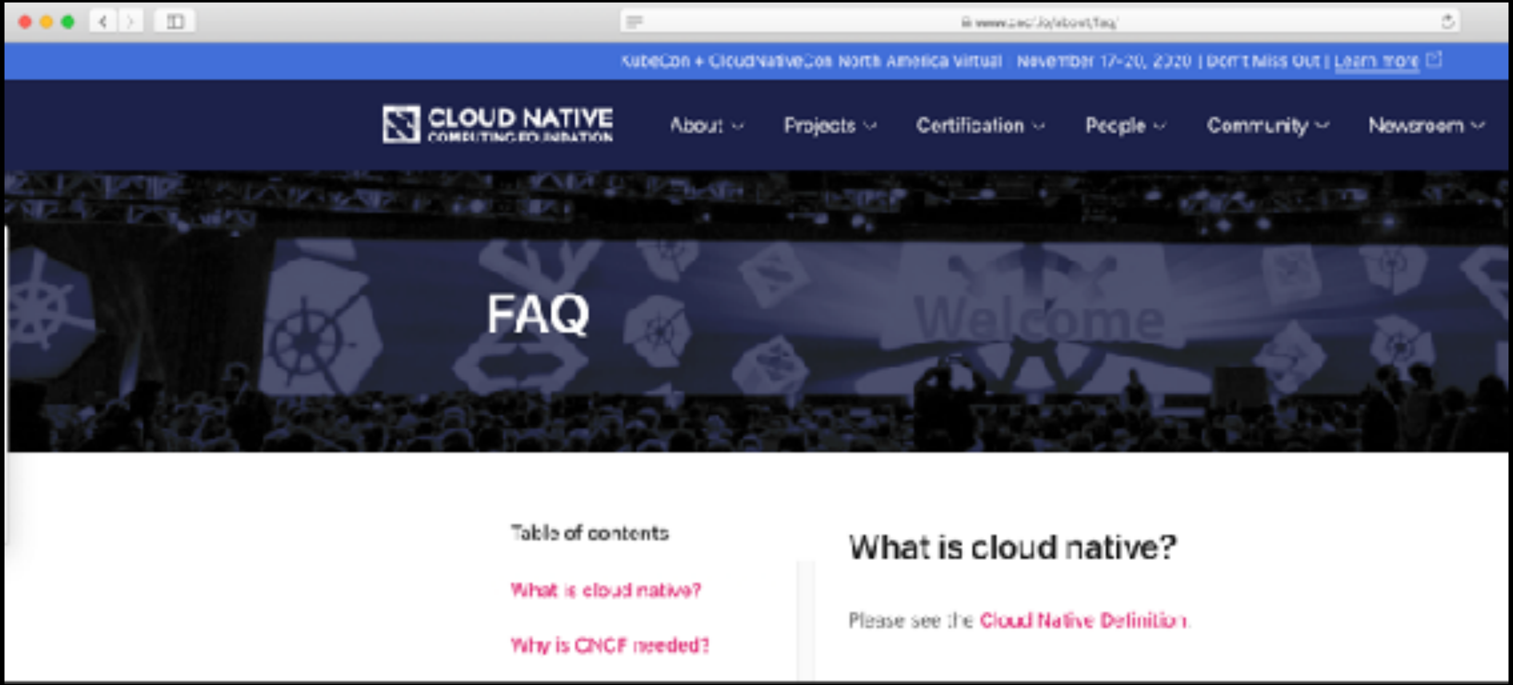
Sam Newman

@samnewman

Interesting side effect of the Cloud Native Foundation is now that I'm commonly speaking to people who believe it can't be cloud native without Kubernetes 🙄

10:38 PM · Nov 19, 2020 · Tweetbot for iOS

15 Retweets **4** Quote Tweets **130** Likes



FAQ

Table of contents

What is cloud native?

Why is CNCF needed?

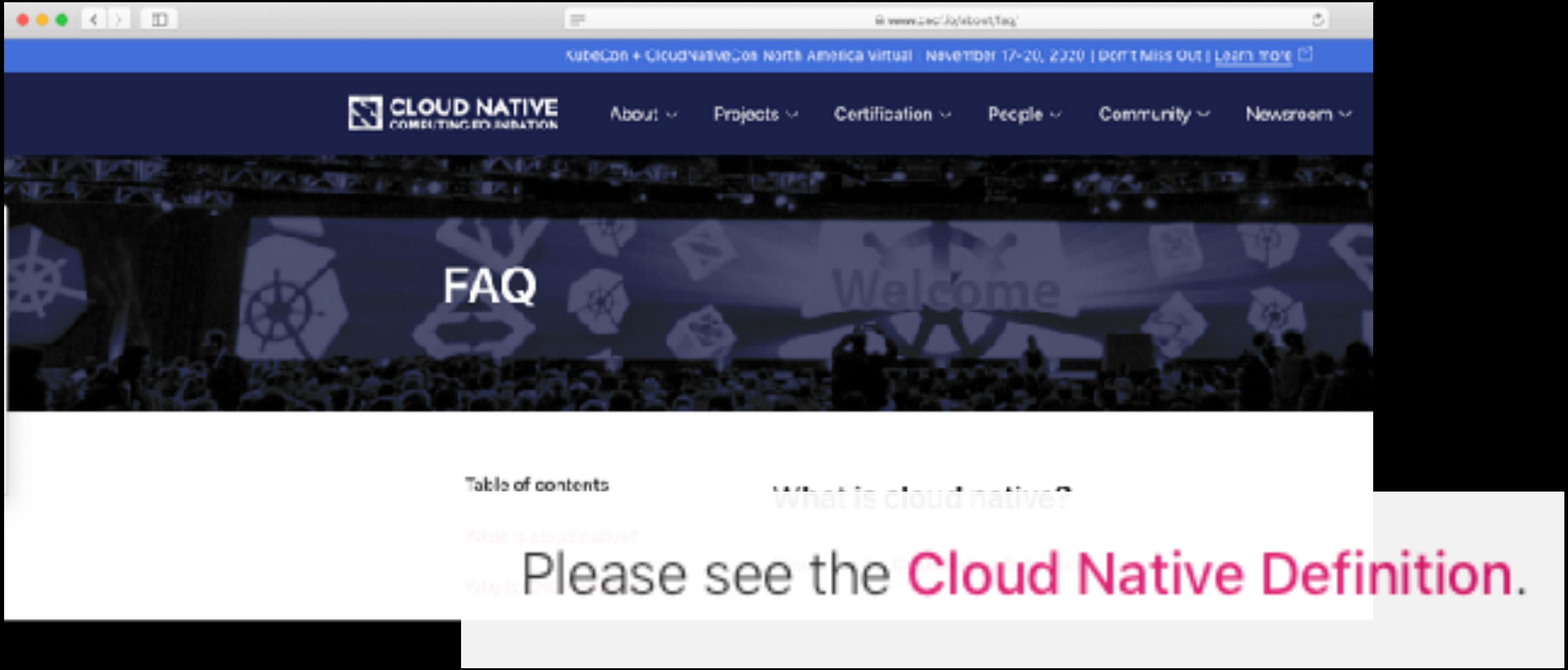
What is cloud native?

Please see the Cloud Native Definition.

2020

IBM **Garage**

@holly_cummins



KubeCon + CloudNativeCon North America Virtual November 17-20, 2020 | Don't Miss Out | Learn more

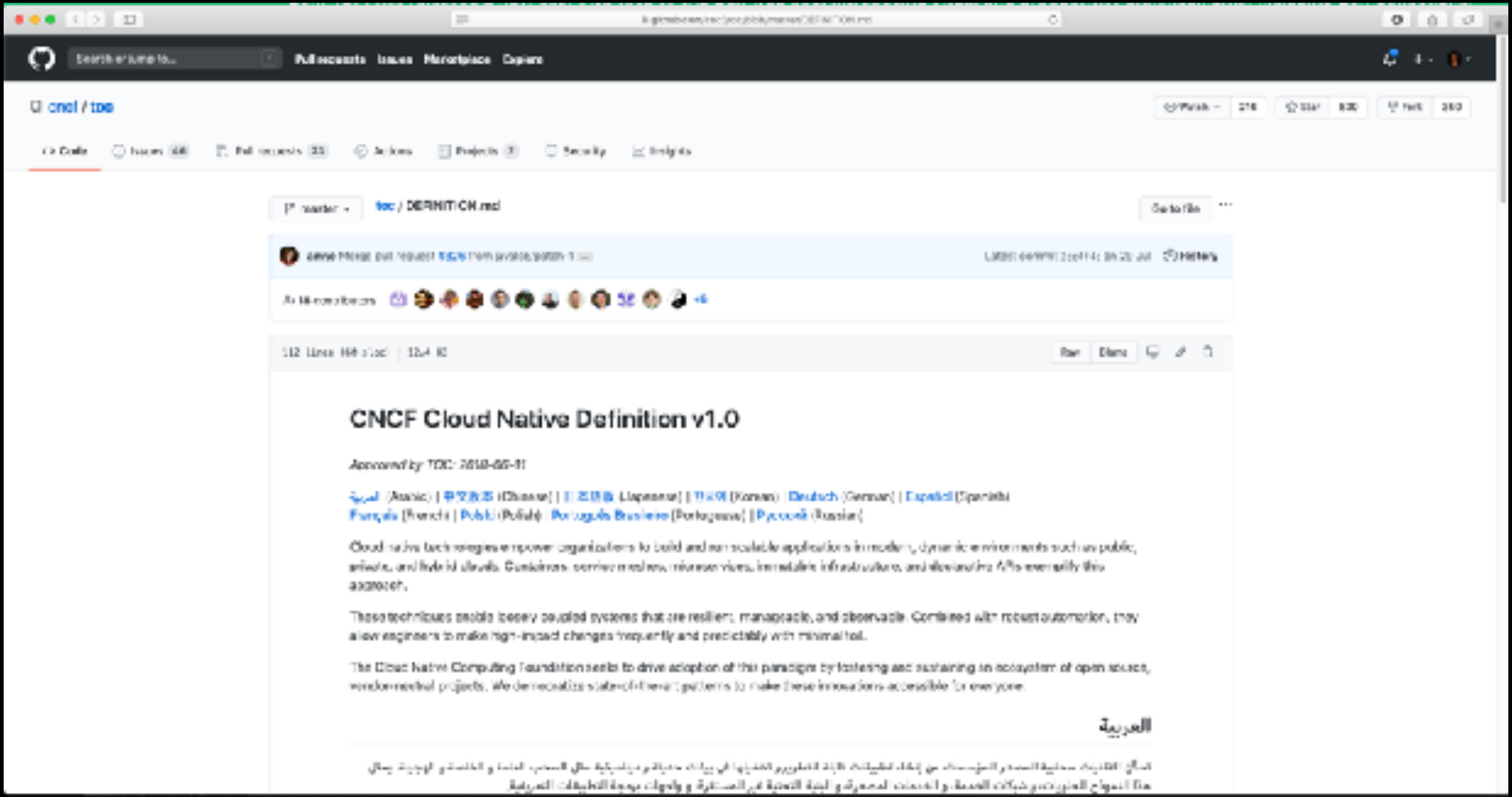
FAQ

Table of contents

What is cloud native?

Please see the **Cloud Native Definition**.

2020



CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

العربية (Arabic) | 中文版本 (Chinese) | 日本語版 (Japanese) | 한국어 (Korean) | Deutsch (German) | Español (Spanish) | Français (French) | Polski (Polish) | Português Brasileiro (Portuguese) | Русский (Russian)

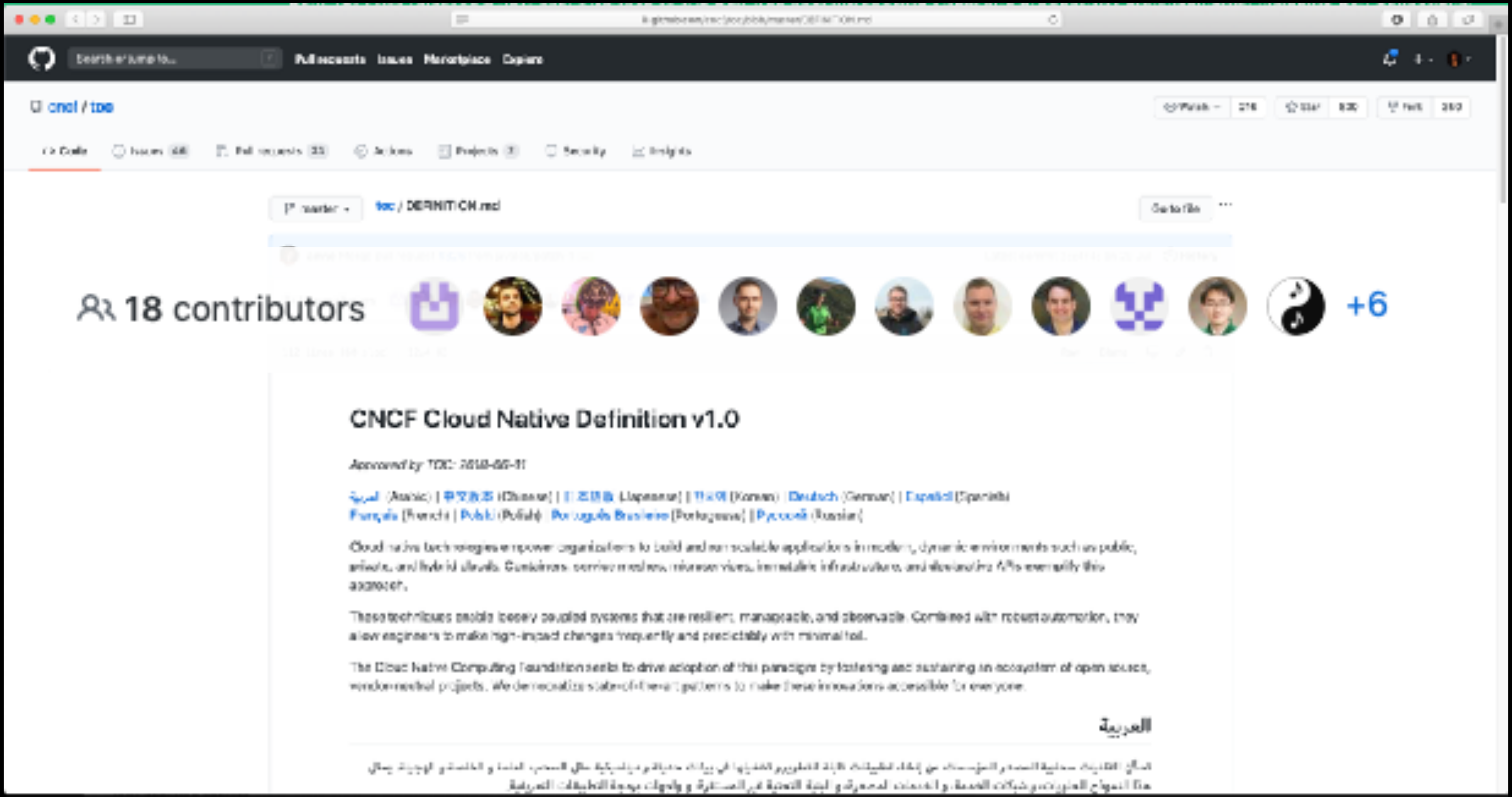
Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

العربية

[illegible]



18 contributors +6

CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

العربية (Arabic) | 中文版本 (Chinese) | 日本語版 (Japanese) | 한국어 (Korean) | Deutsch (German) | Español (Spanish) Français (French) | Polski (Polish) | Português Brasileiro (Portuguese) | Русский (Russian)

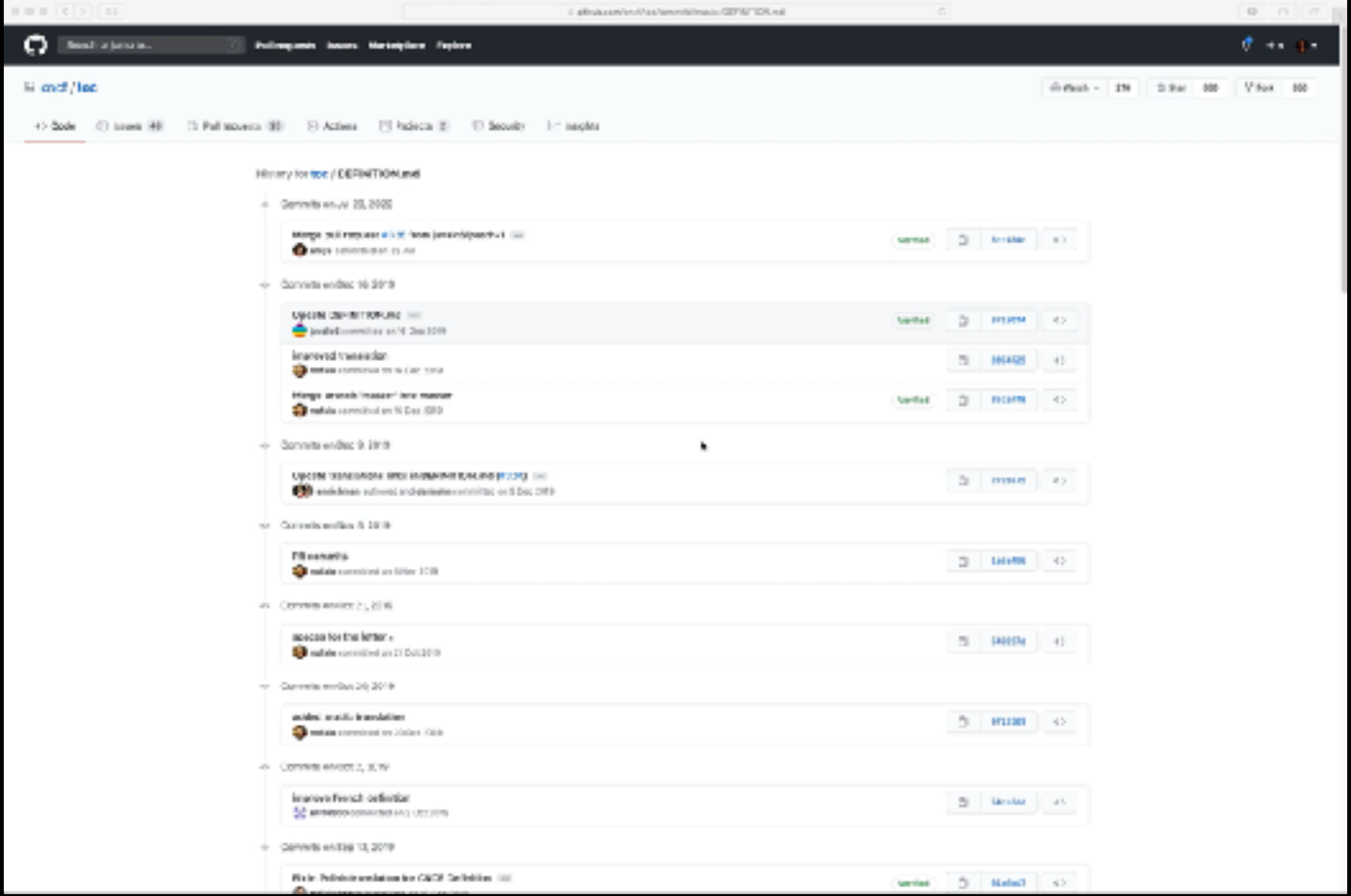
Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

العربية

[illegible]



History for toc / DEFINITION.md

Commits on Jul 28, 2020

Merge pull request [illegible] from [illegible]

Commits on Dec 16, 2019

Update DEFINITION.md

Improved translation

Merge branch 'master' into master

Commits on Dec 9, 2019

Update [illegible] (#[illegible])

Commits on Nov 5, 2019

Fill semantic

Commits on Oct 21, 2019

spaces for the letter [illegible]

Commits on Oct 20, 2019

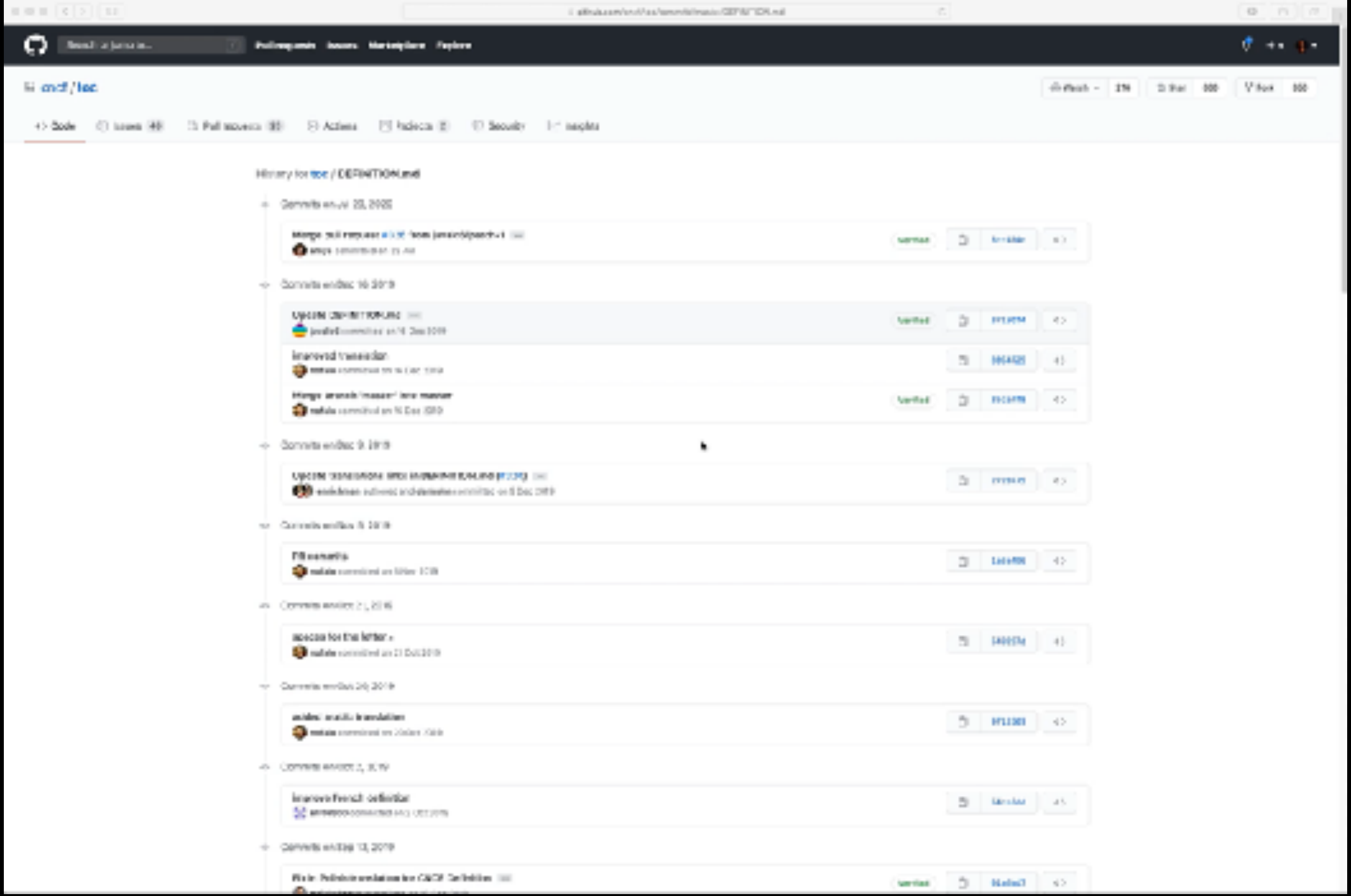
added arabic translation

Commits on Oct 2, 2019

Improve French definition

Commits on Sep 13, 2019

[illegible] translation for CNCF Definition



History for toc / DEFINITION.md

Commits on Jul 23, 2020

Merge pull request [illegible]

Commits on Dec 16, 2019

Update DEFINITION.md

Improved translation

Merge branch 'master' into master

Commits on Dec 9, 2019

Update [illegible]

Commits on Nov 5, 2019

[illegible]

Commits on Oct 21, 2019

spaces for the letter [illegible]

Commits on Oct 20, 2019

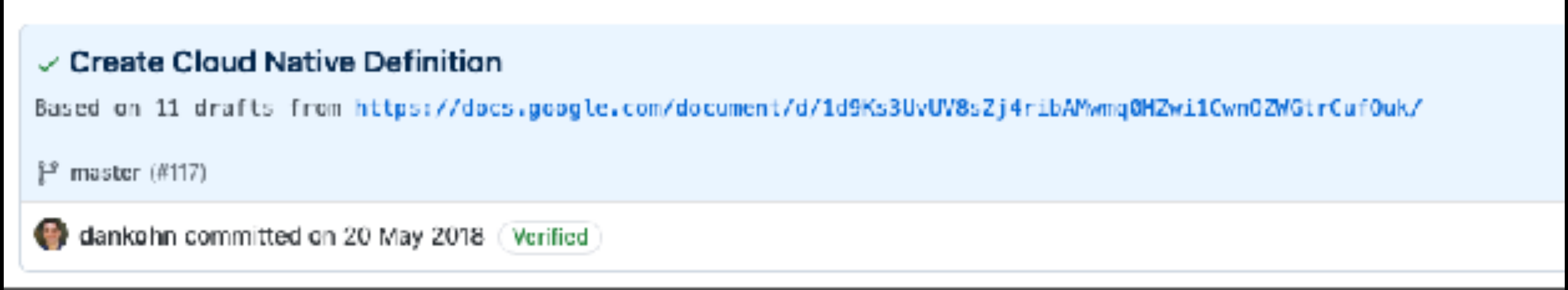
added arabic translation

Commits on Oct 2, 2019

Improve french definition

Commits on Sep 13, 2019

[illegible] CNCF Definition

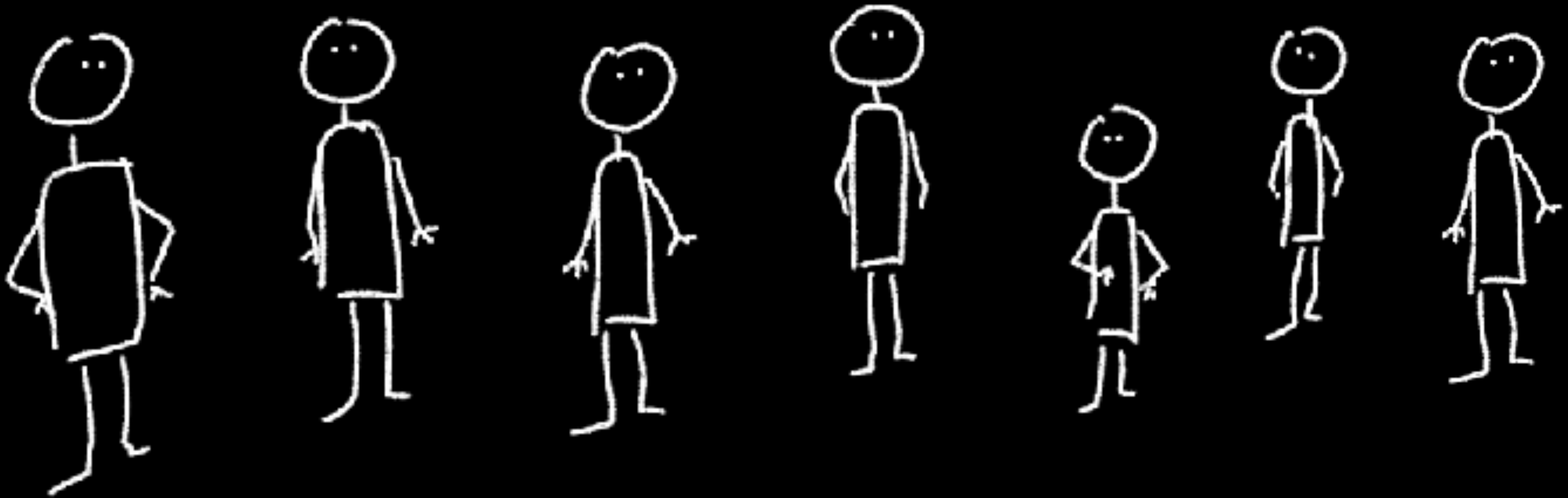


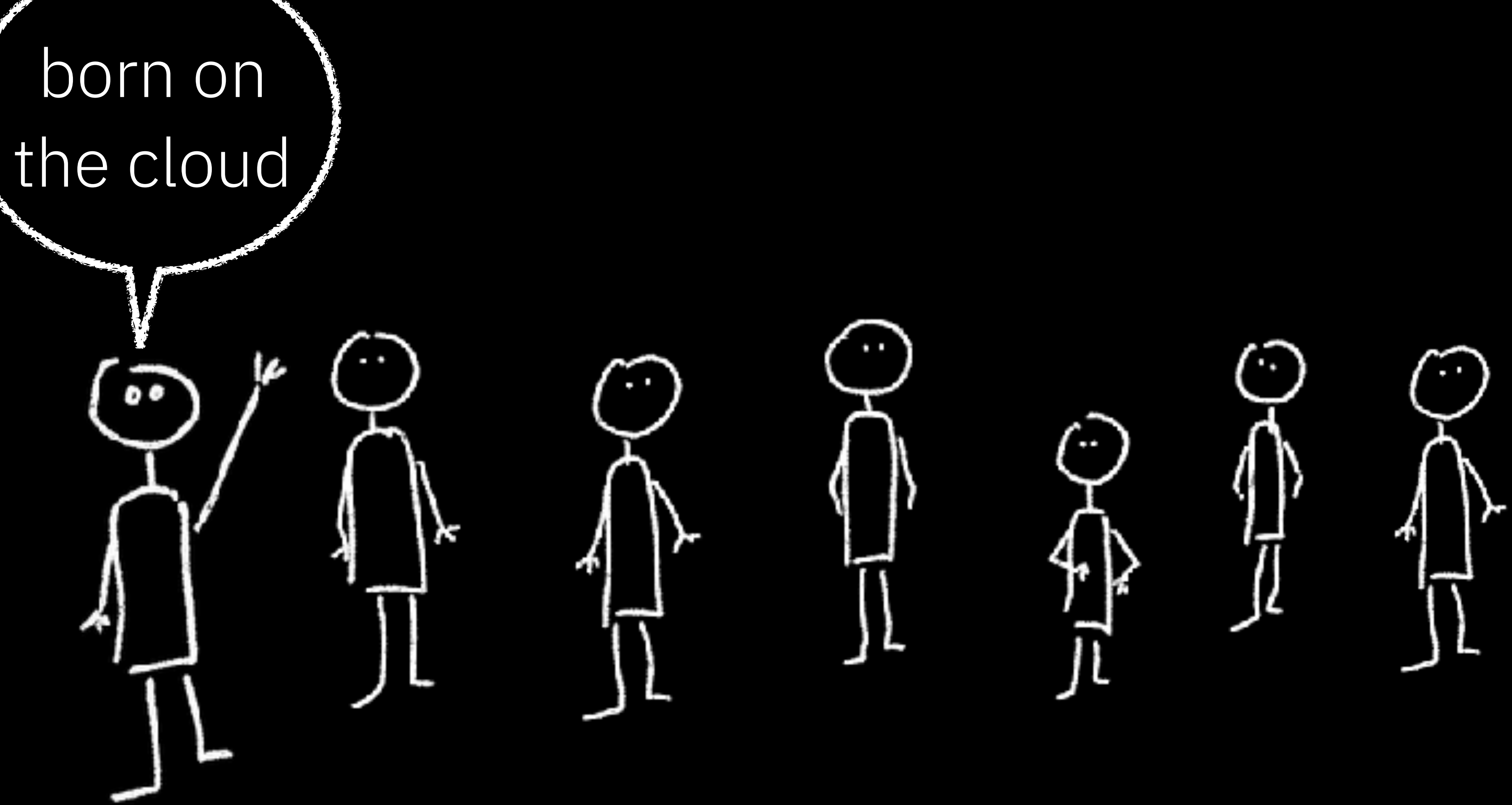
✓ Create Cloud Native Definition

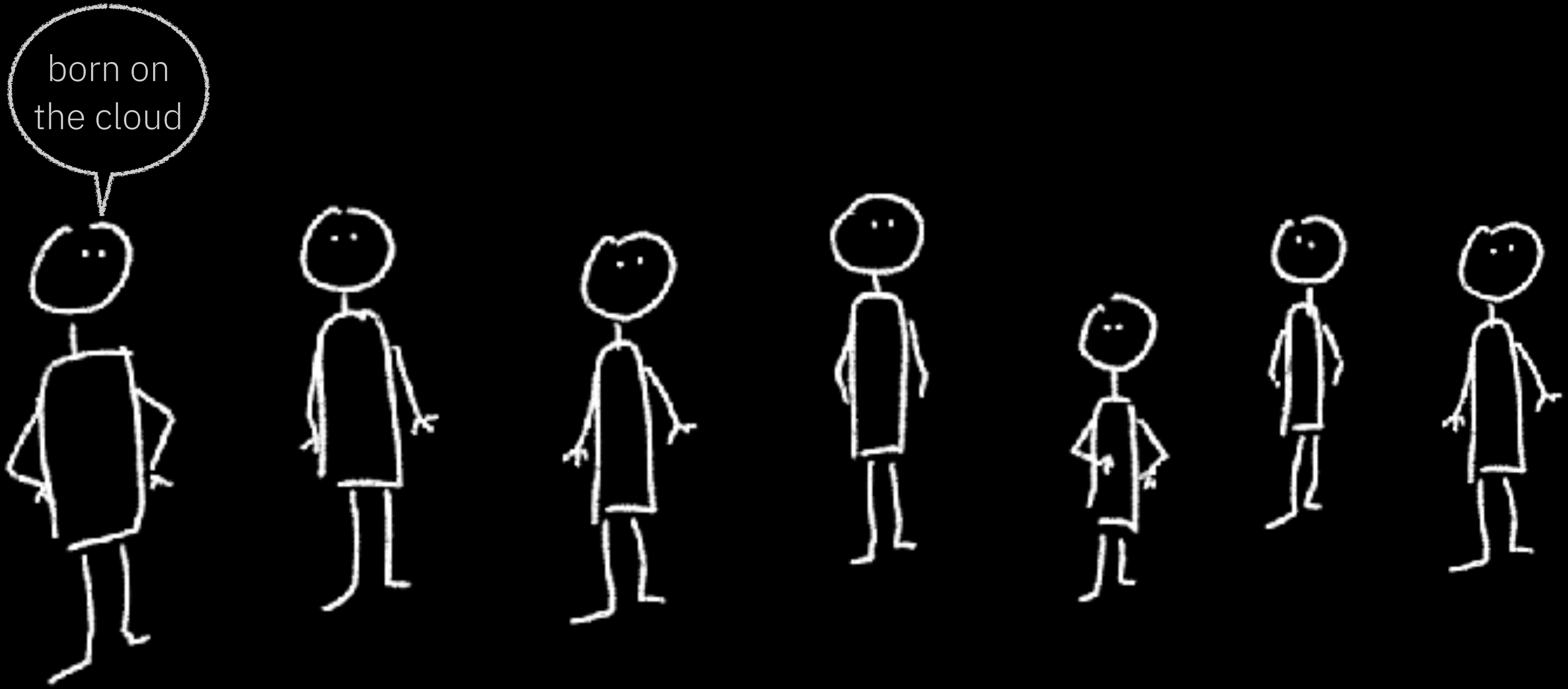
```
Based on 11 drafts from https://docs.google.com/document/d/1d9Ks3UvUV8sZj4ribAMwmq0HZwi1CwnOZWGtrCufOuk/
```

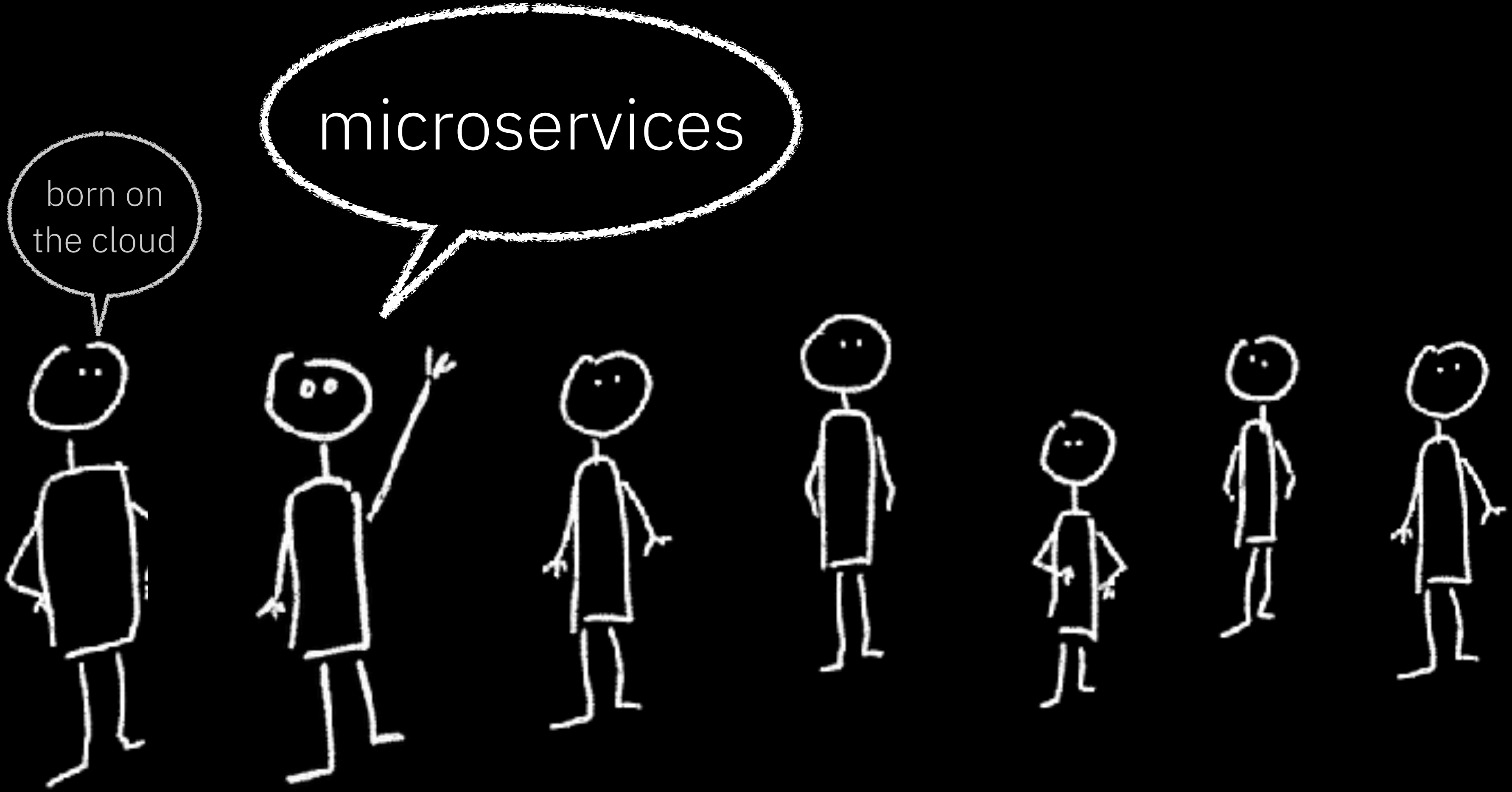
master (#117)

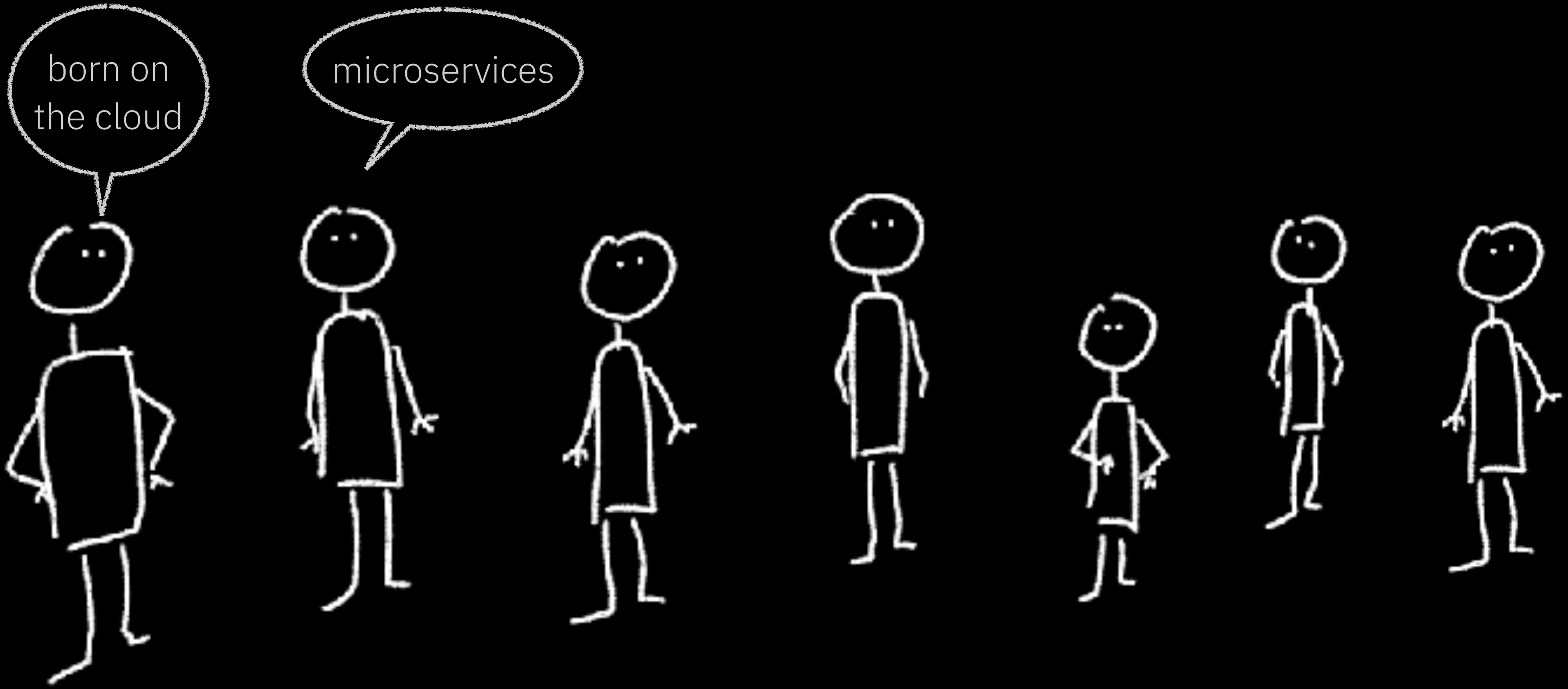
dankohn committed on 20 May 2018 Verified

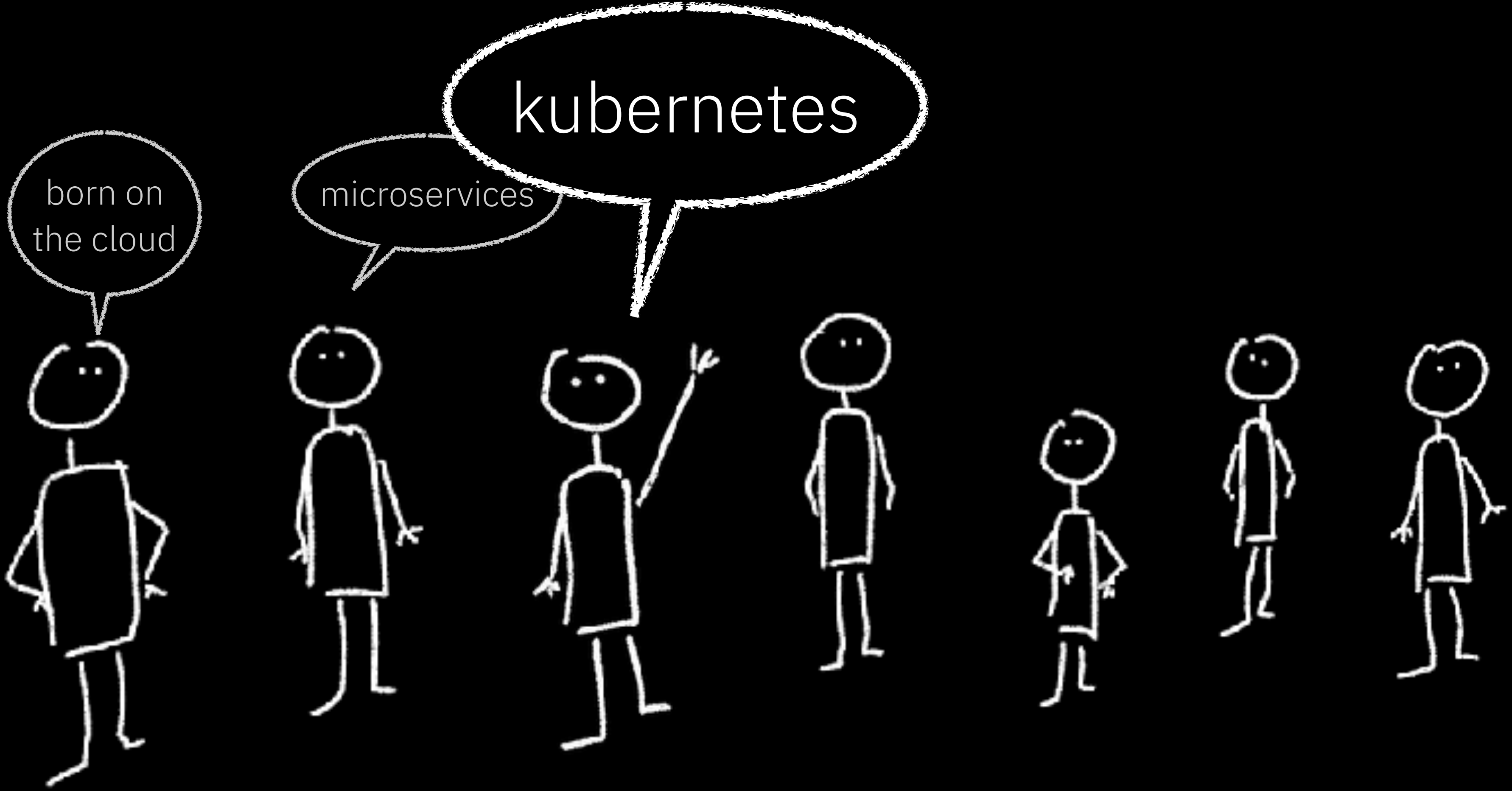


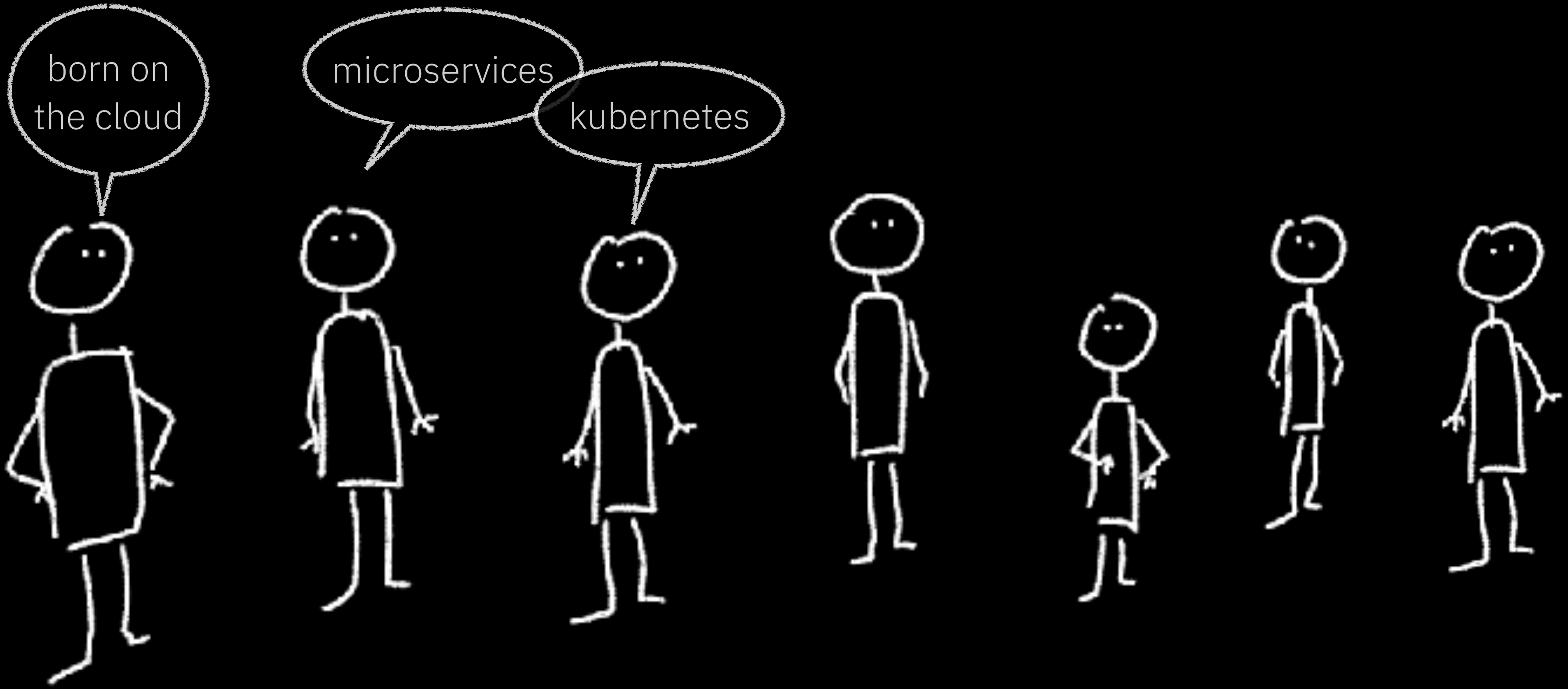

born on the cloud

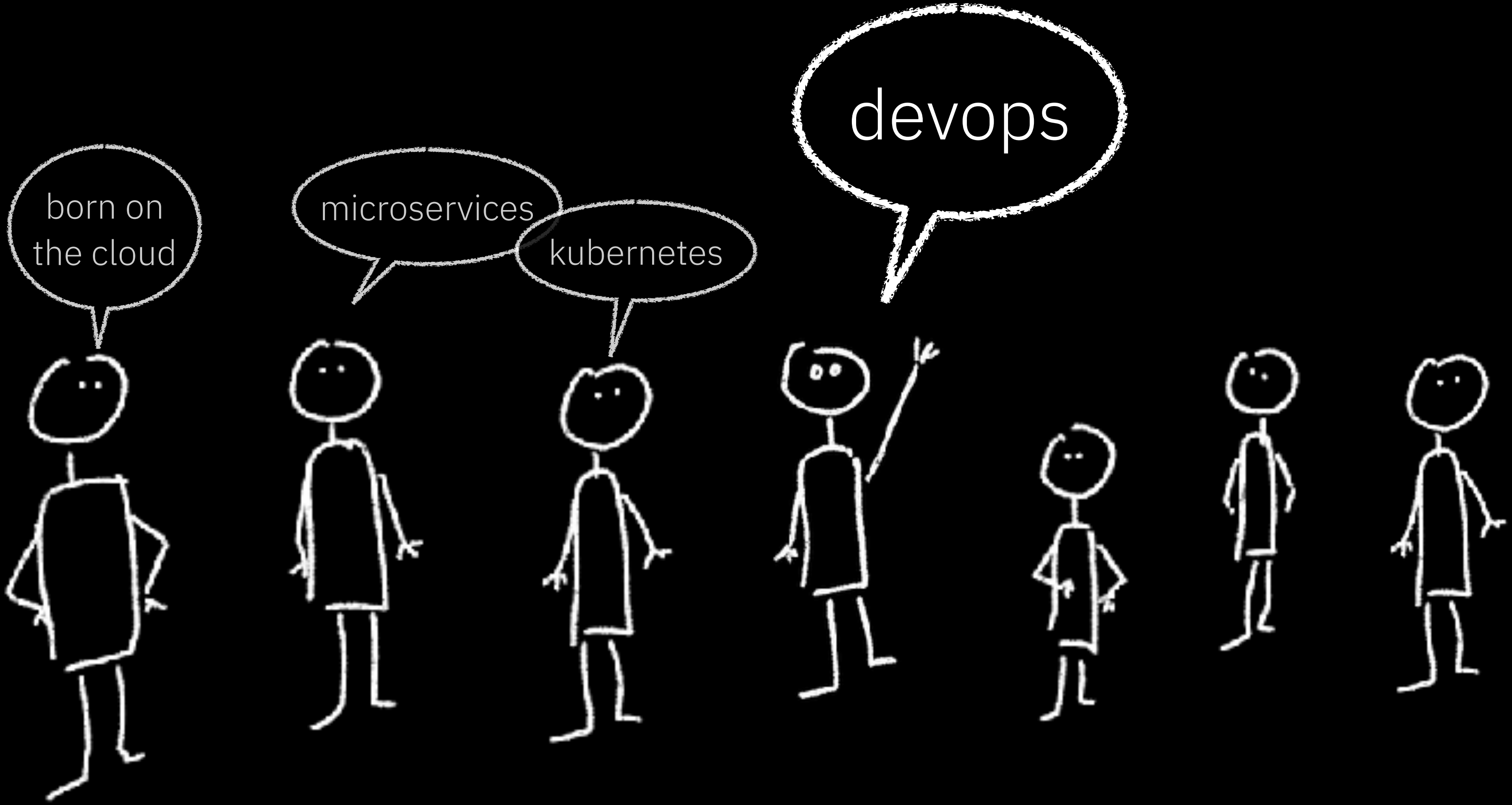

born on the cloud

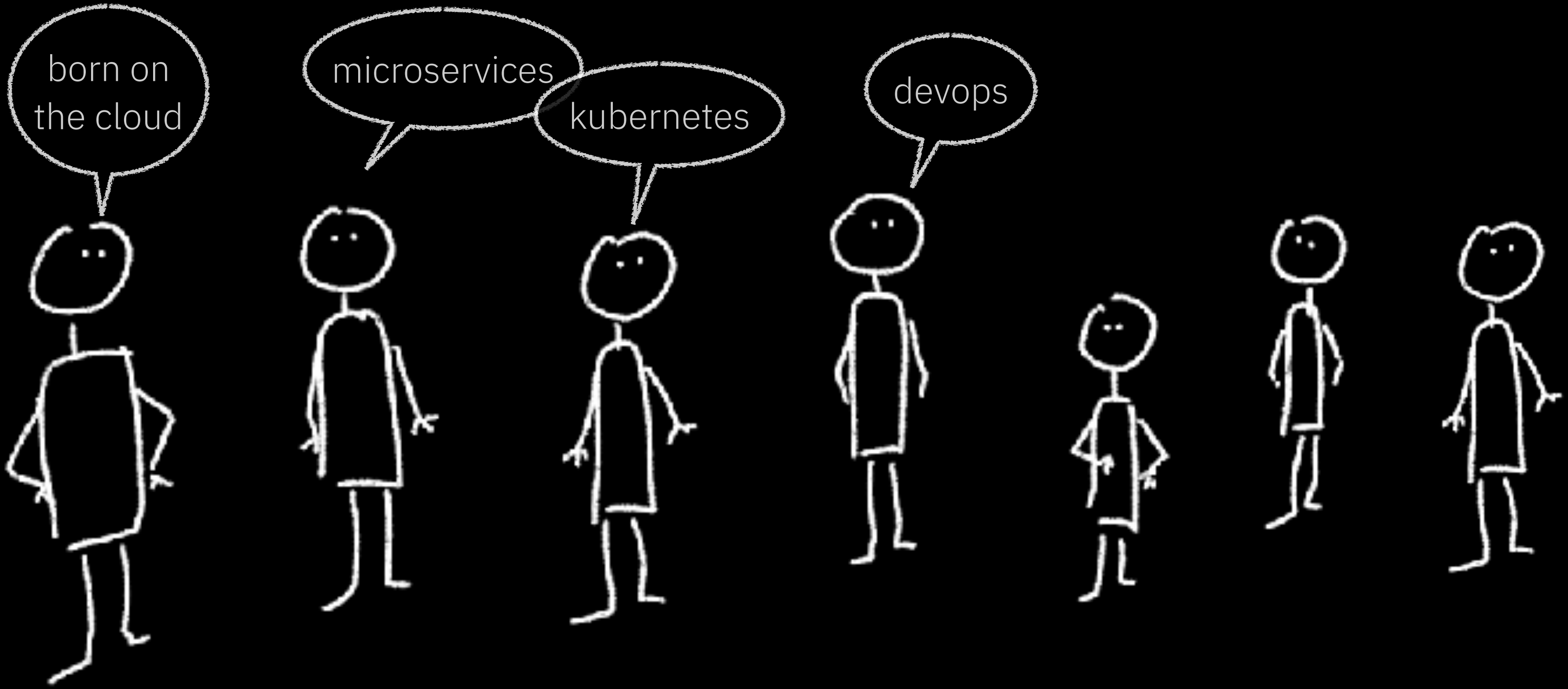

born on the cloud
microservices

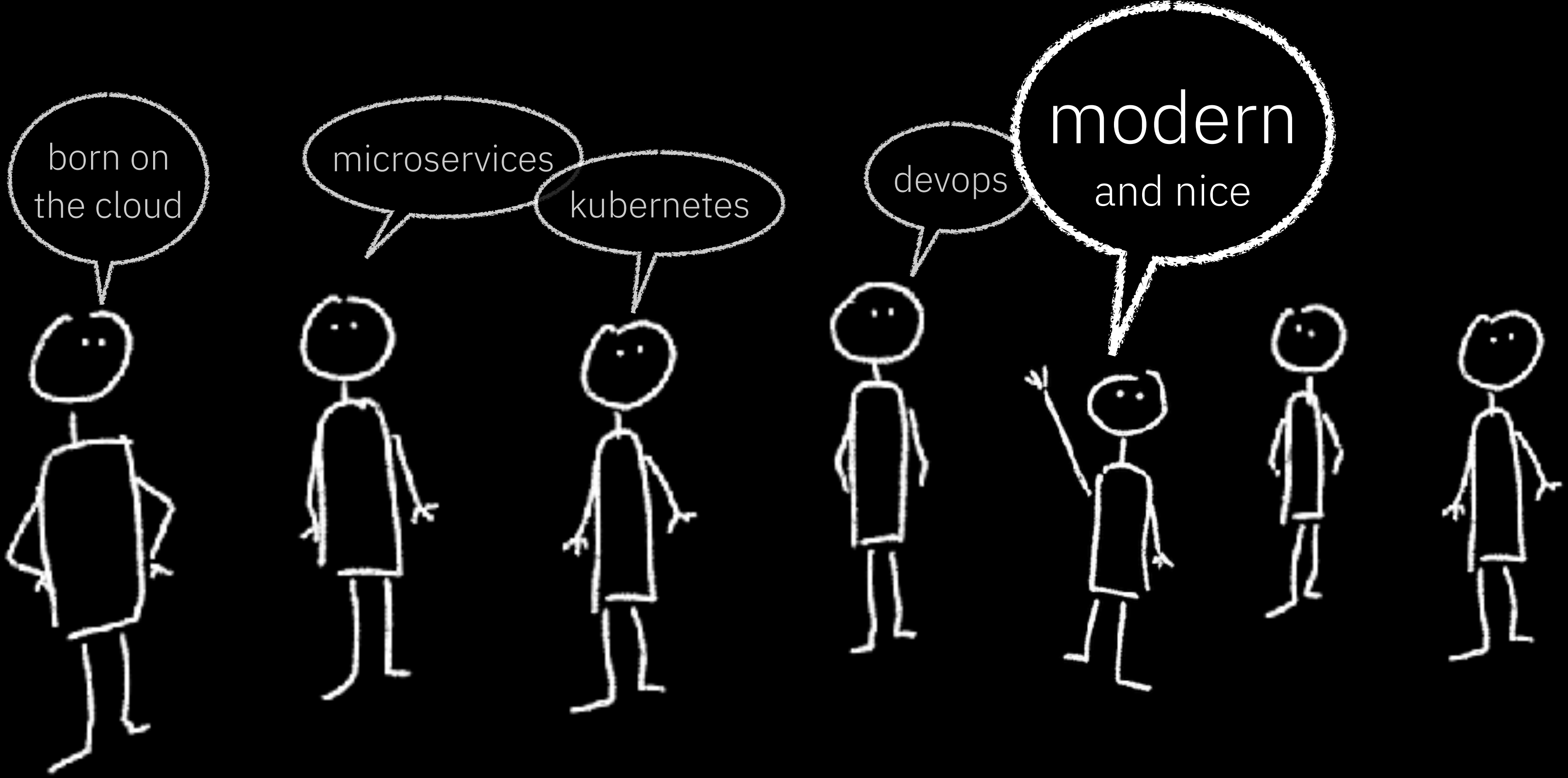

born on the cloud
microservices

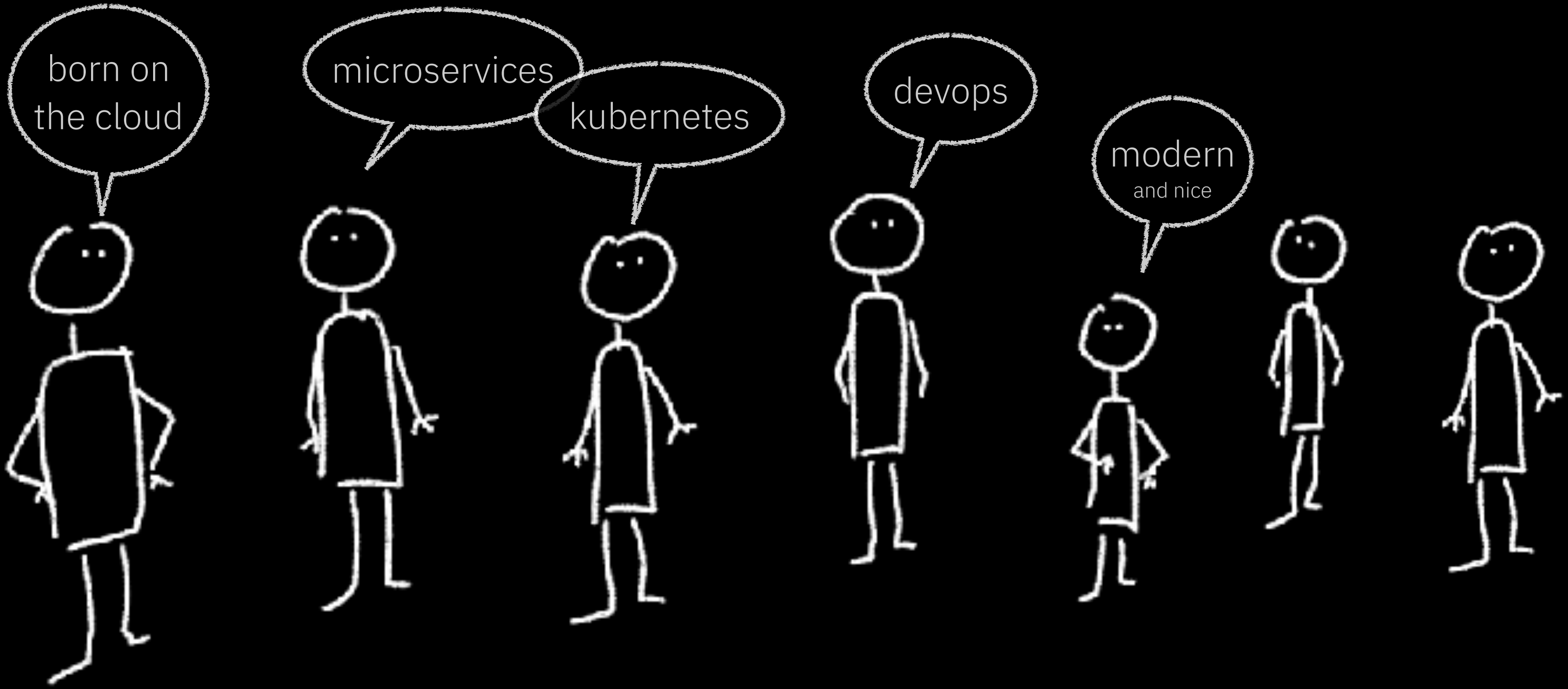

born on the cloud
microservices
kubernetes

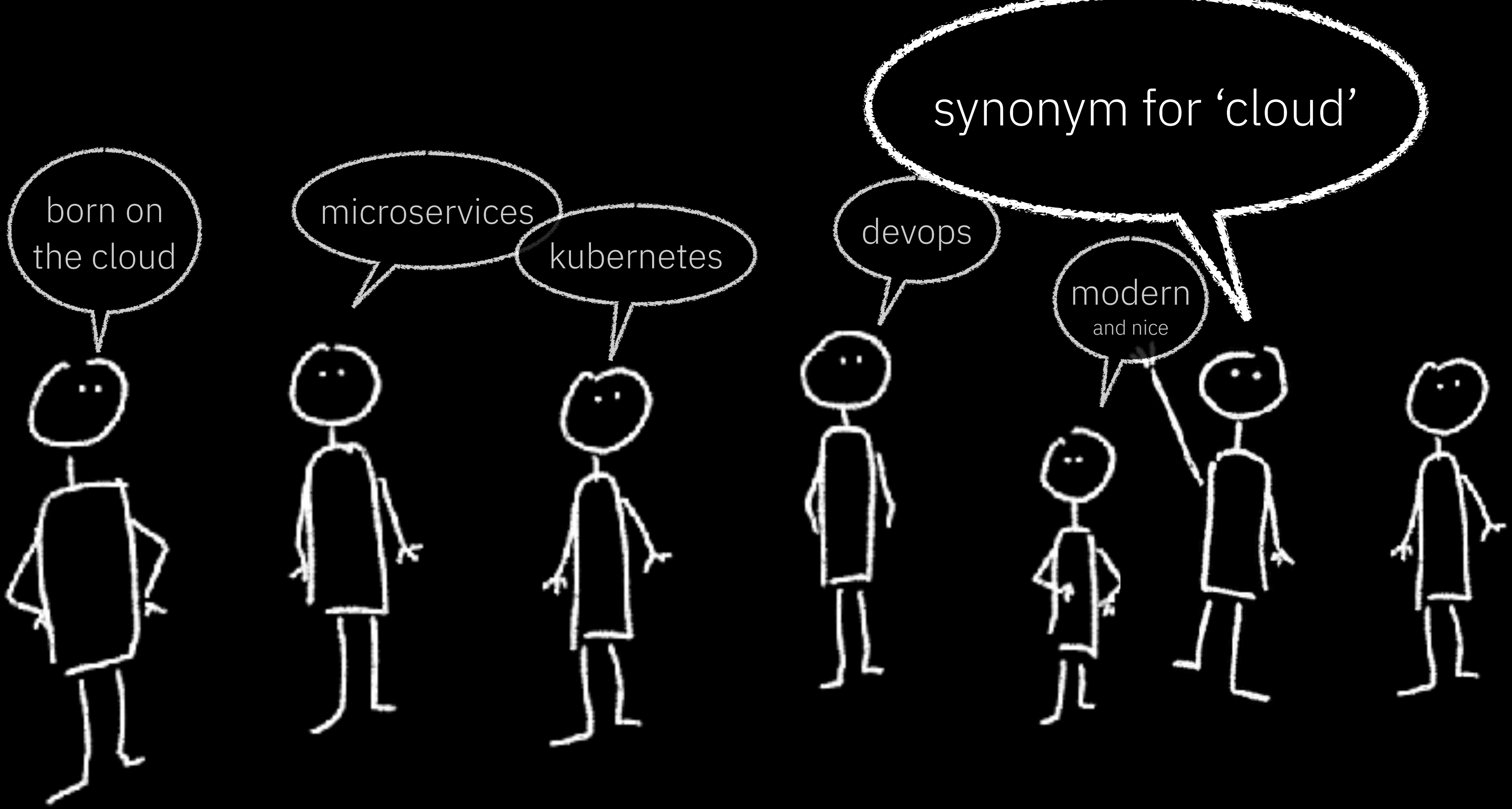

born on the cloud
microservices
kubernetes

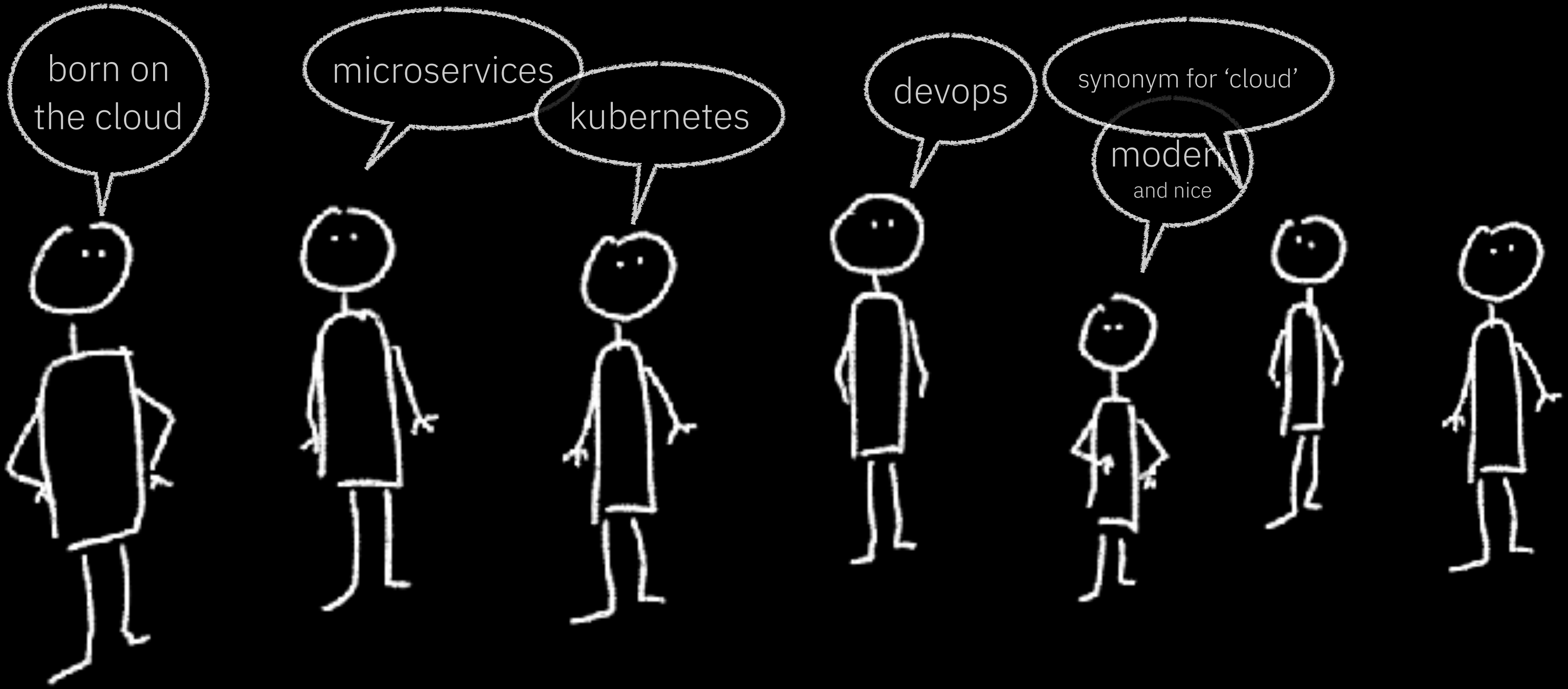

born on the cloud
microservices
kubernetes
devops

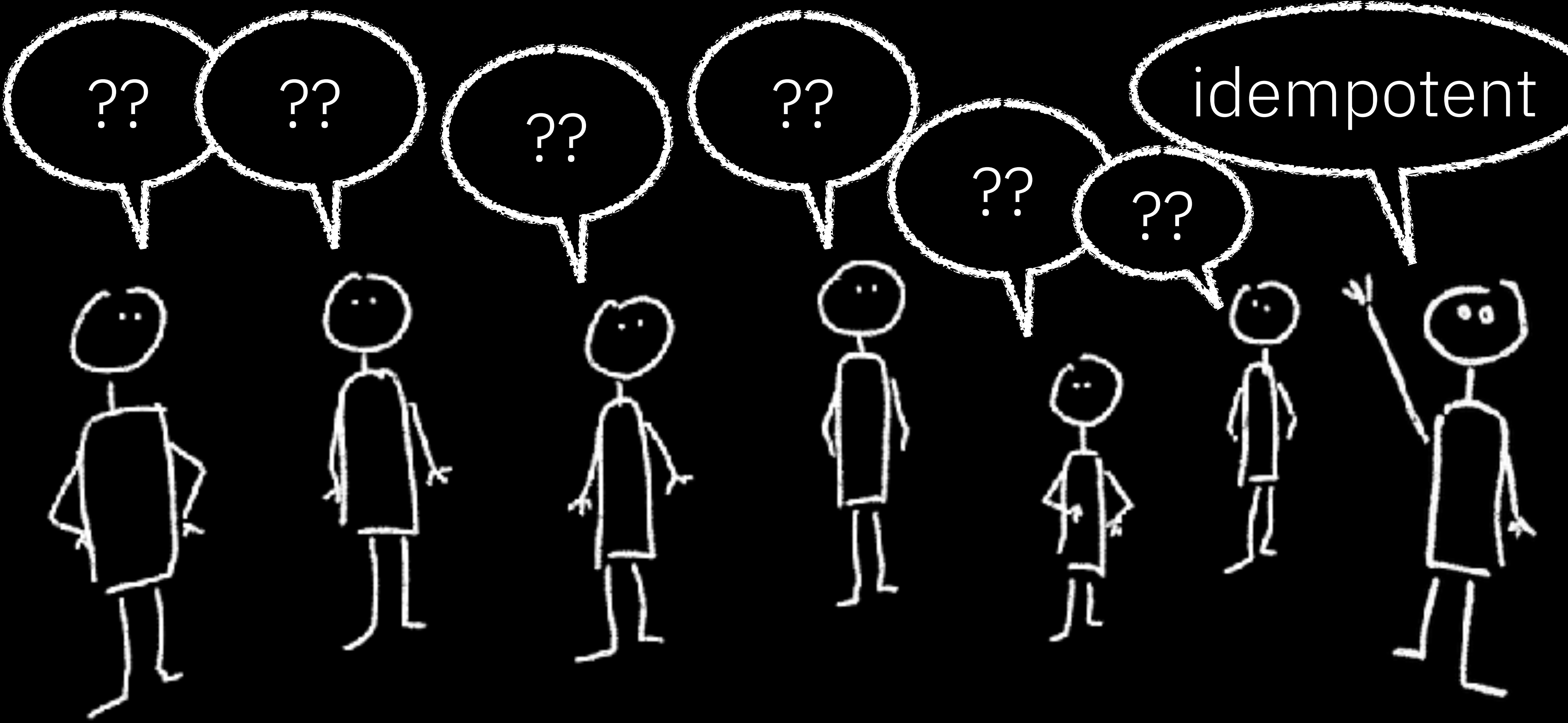

born on the cloud
microservices
kubernetes
devops

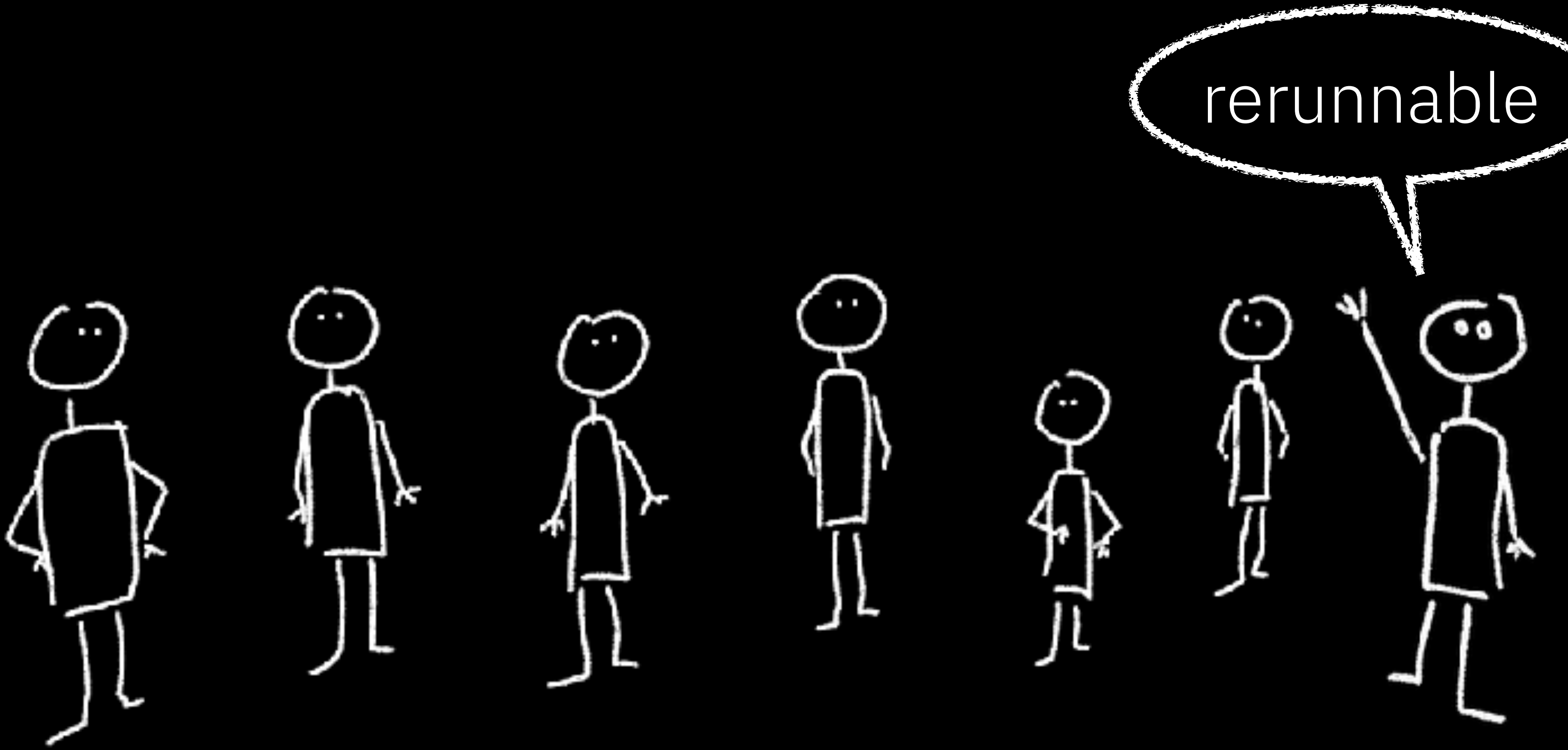

born on the cloud
microservices
kubernetes
devops
modern
and nice

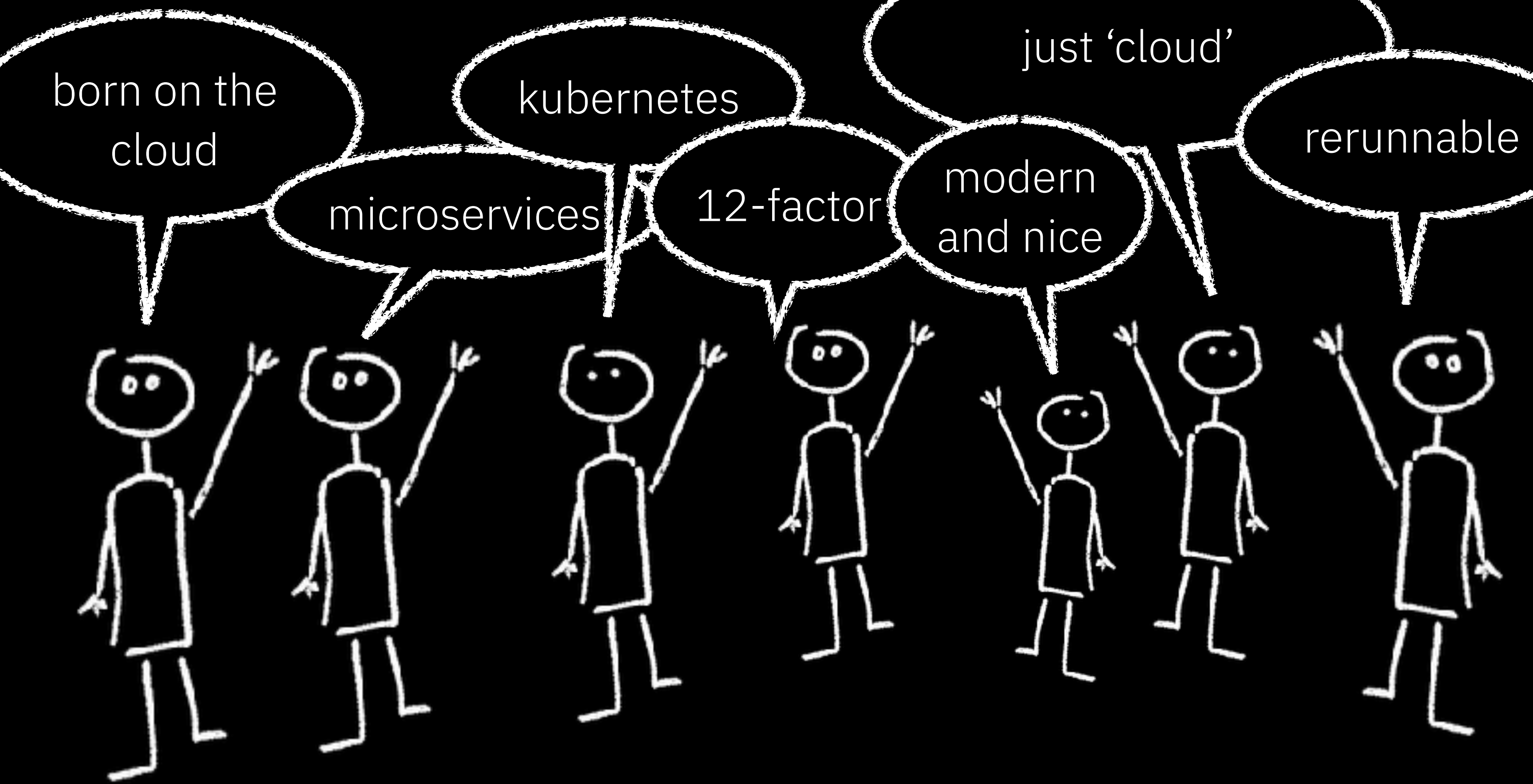

born on the cloud
microservices
kubernetes
devops
modern
and nice


born on the cloud
microservices
kubernetes
devops
modern
and nice
synonym for ‘cloud’


born on the cloud
microservices
kubernetes
devops
synonym for 'cloud'
modern
and nice

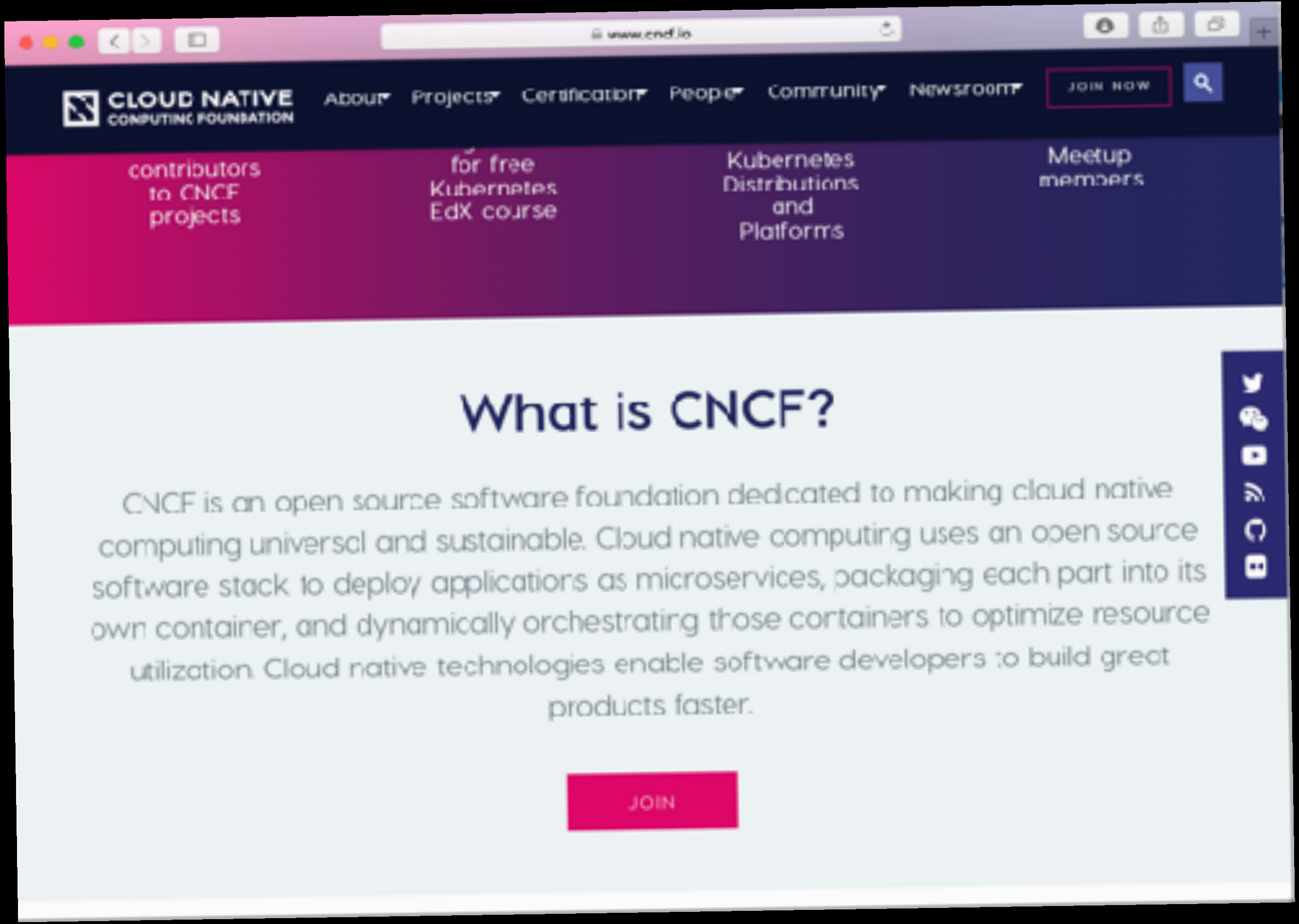

??
??
??
??
??
??
idempotent


rerunnable


born on the cloud
microservices
kubernetes
12-factor
modern and nice
just ‘cloud’
rerunnable

cloud native is **not**

cloud native is **not**
a synonym for ‘microservices’



contributors to CNCF projects

for free Kubernetes EdX course

Kubernetes Distributions and Platforms

Meetup members

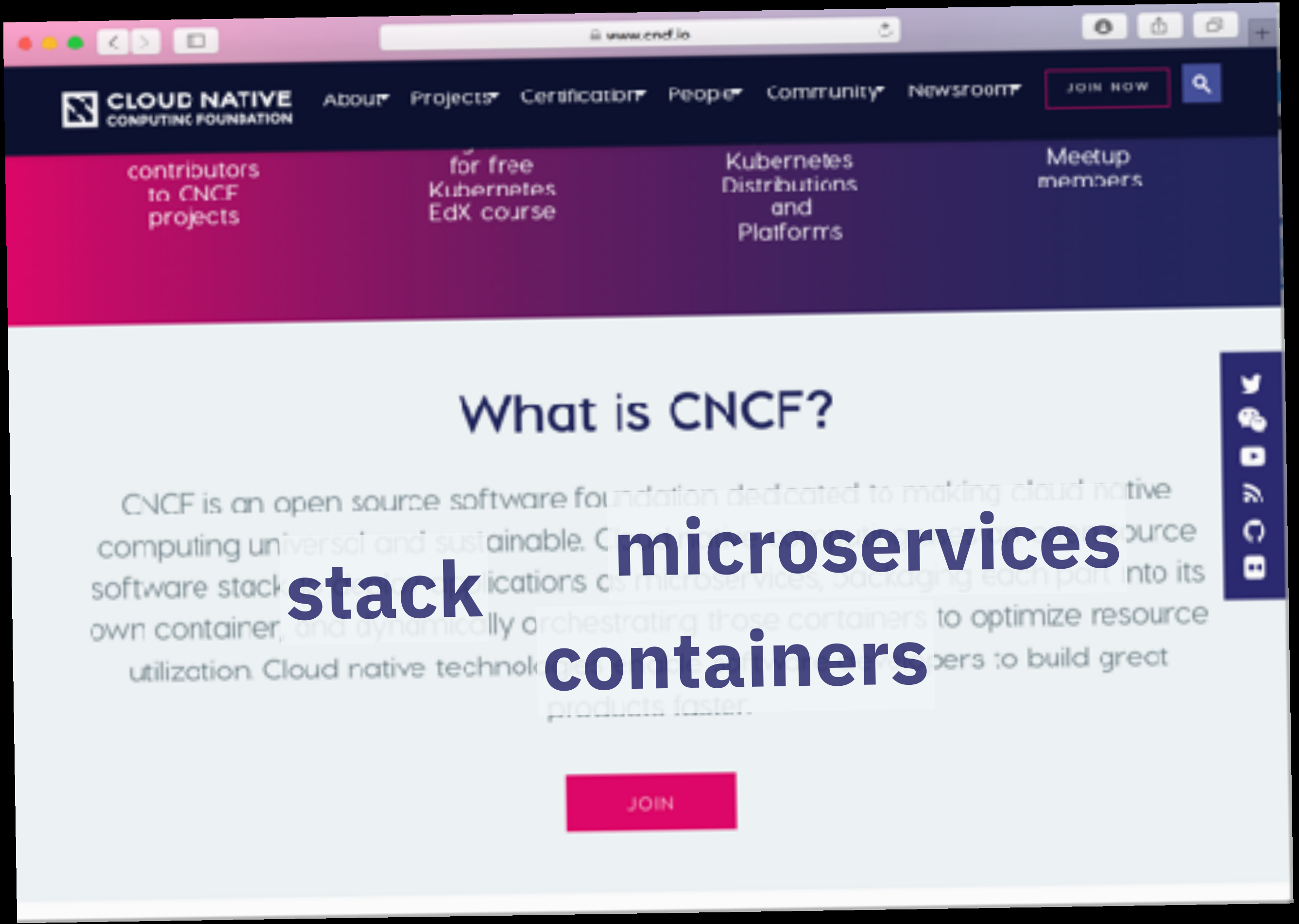
What is CNCF?

CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.

2019

#IBMGarage

@holly_cummins


www.cncf.io
CLOUD NATIVE COMPUTING FOUNDATION
About
Projects
Certification
People
Community
Newsroom
JOIN NOW
contributors to CNCF projects
for free Kubernetes EdX course
Kubernetes Distributions and Platforms
Meetup members
JOIN

What is CNCF?

CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.

stack

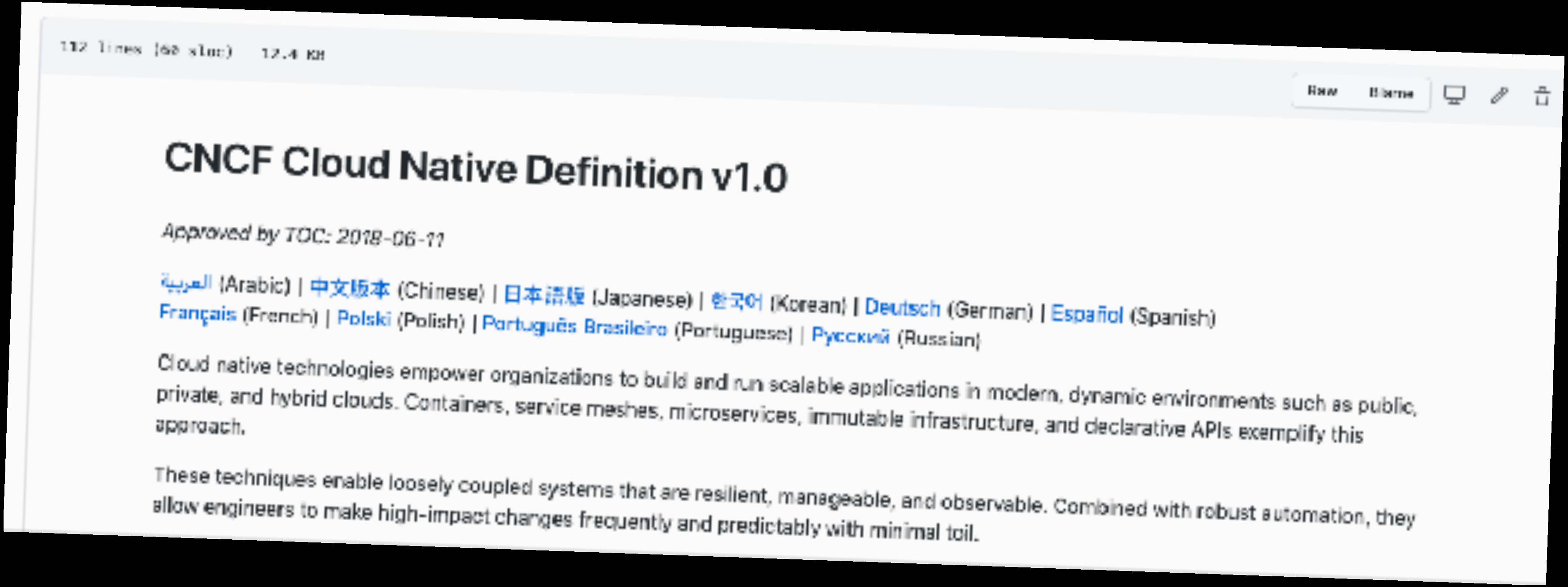
microservices

containers

2019

#IBMGarage

@holly_cummins



CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

العربية (Arabic) | 中文版本 (Chinese) | 日本語版 (Japanese) | 한국어 (Korean) | Deutsch (German) | Español (Spanish) Français (French) | Polski (Polish) | Português Brasileiro (Portuguese) | Русский (Russian)

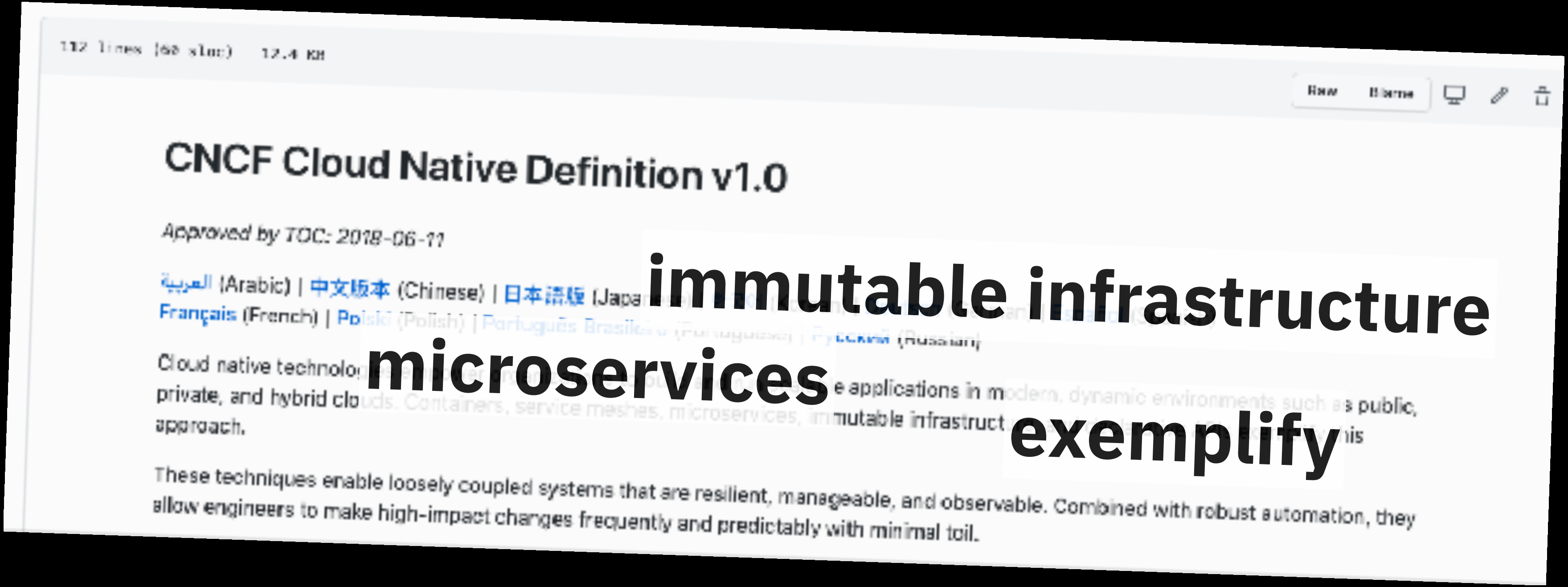
Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

2020

IBM **Garage**

@holly_cummins



CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

العربية (Arabic) | 中文版本 (Chinese) | 日本語版 (Japanese) | 한국어 (Korean) | Deutsch (German) | Español (Spanish) | Français (French) | Polski (Polish) | Português Brasileiro (Portuguese) | Русский (Russian)

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

immutable infrastructure

microservices

exemplify

2020

IBM **Garage**

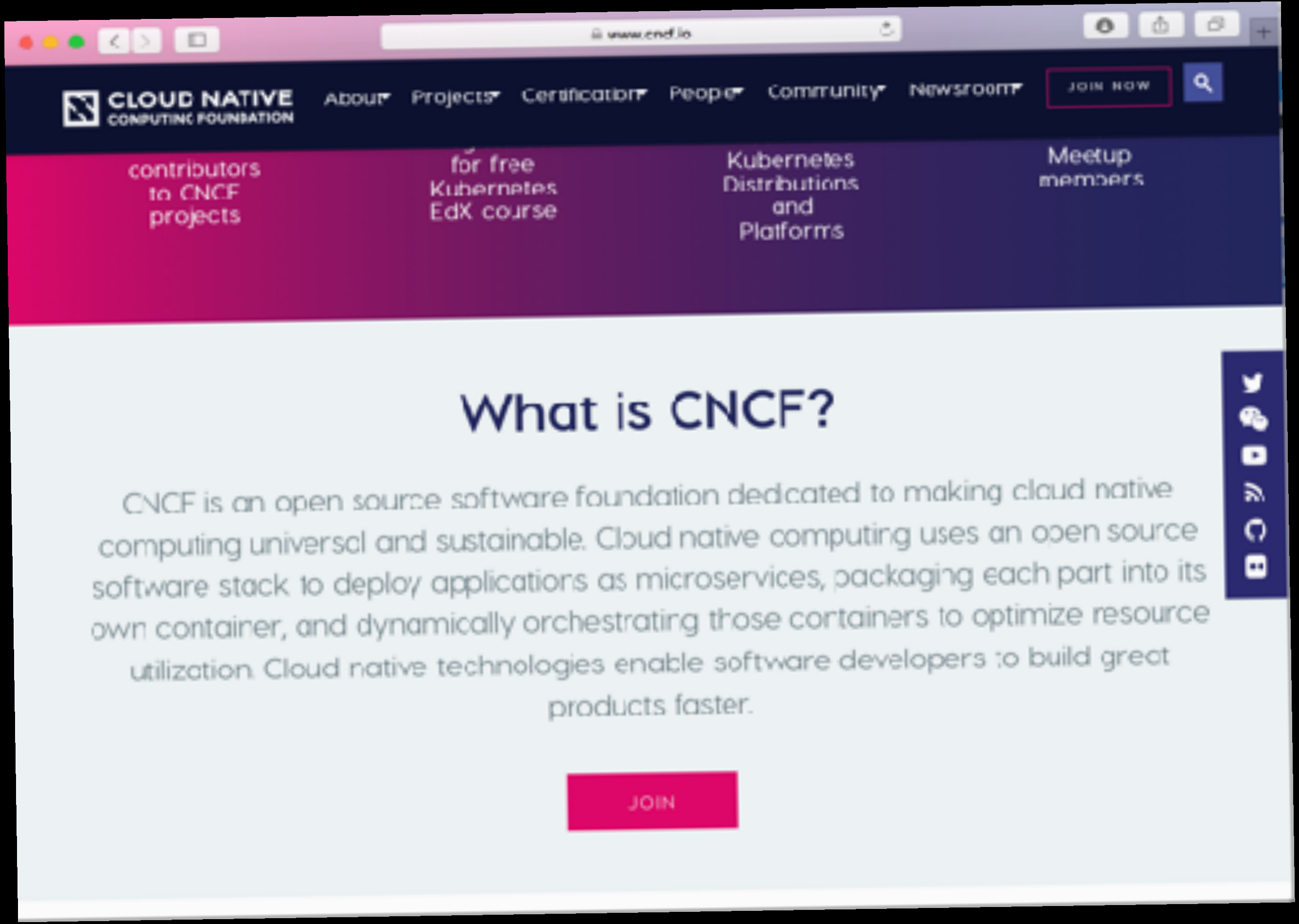
@holly_cummins

why?

what **problem** are we trying to solve?

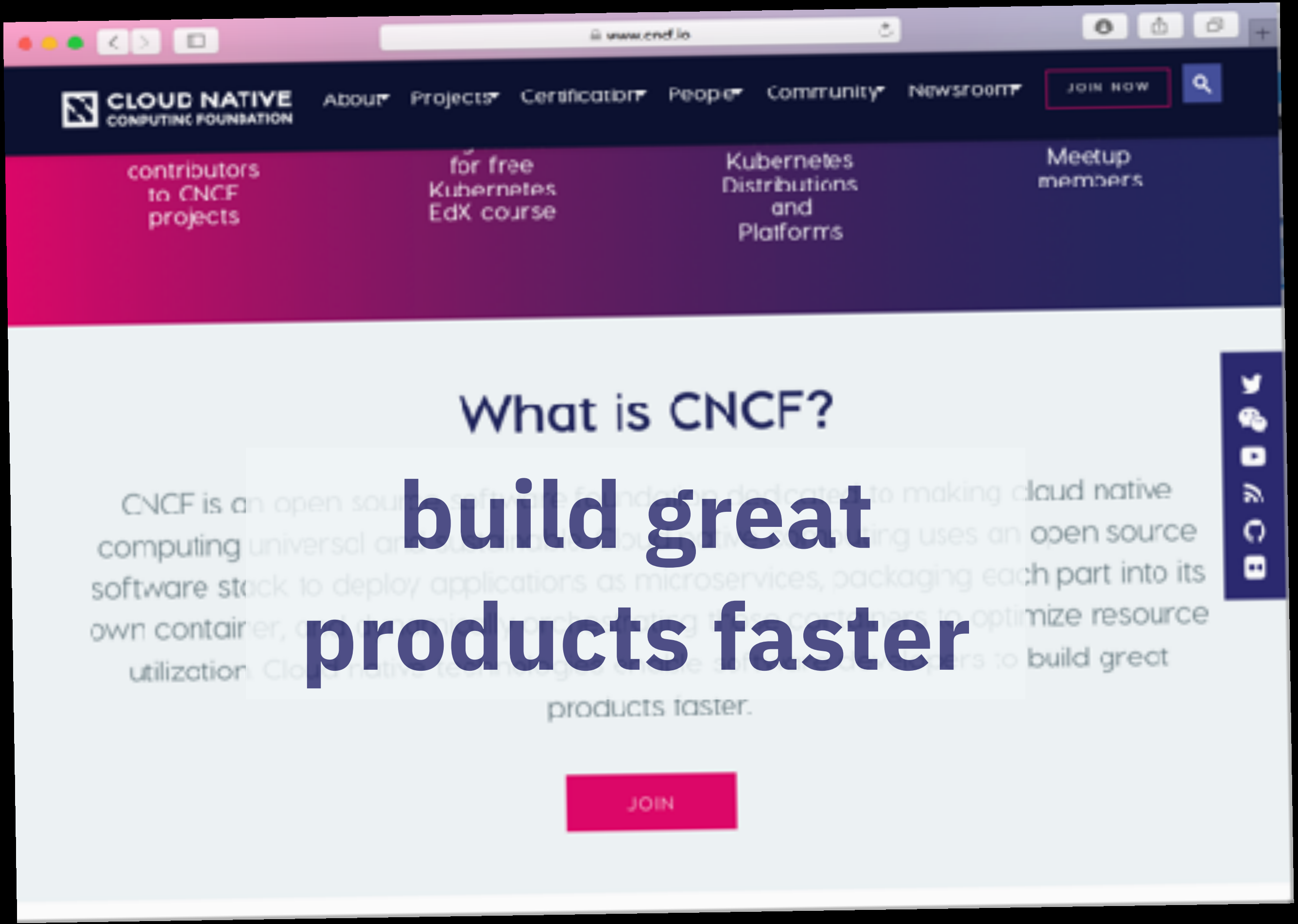
“everyone else is doing it?”

wishful mimicry



What is CNCF?

CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.



contributors to CNCF projects

for free Kubernetes EdX course

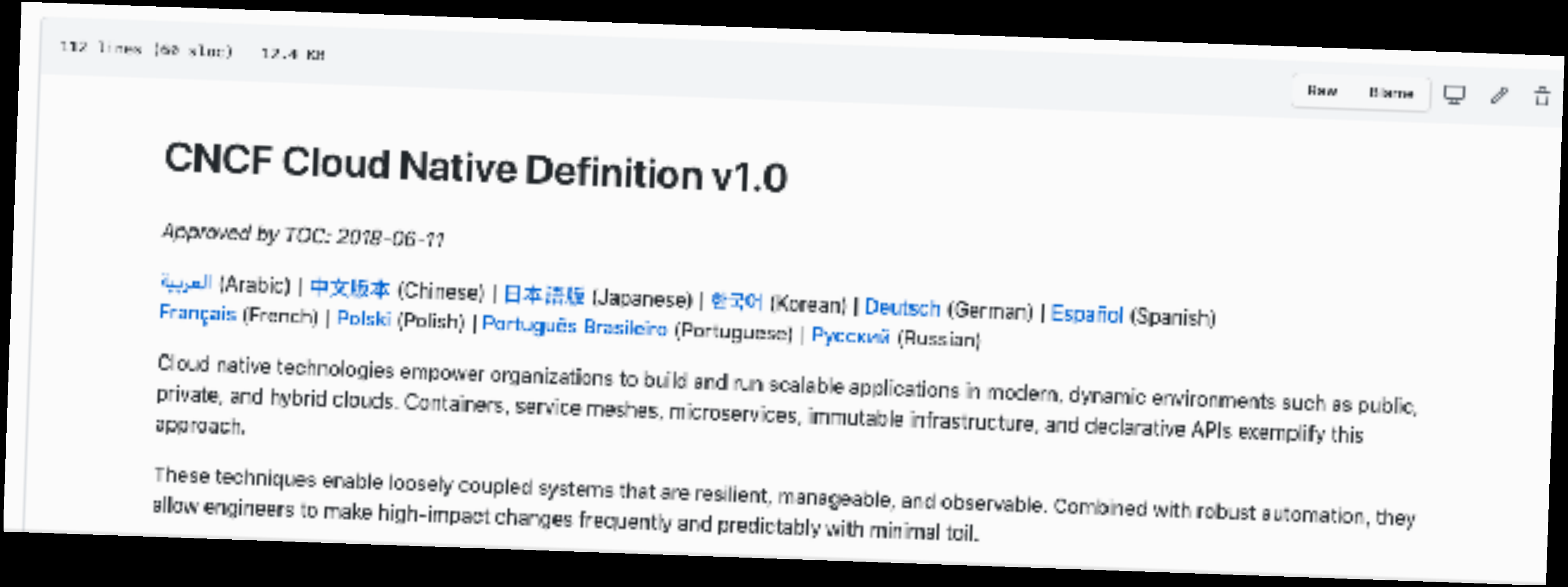
Kubernetes Distributions and Platforms

Meetup members

What is CNCF?

CNCF is an open source software foundation dedicated to making cloud native computing universal and sustainable. Cloud native computing uses an open source software stack to deploy applications as microservices, packaging each part into its own container, and dynamically orchestrating those containers to optimize resource utilization. Cloud native technologies enable software developers to build great products faster.

build great products faster



CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

العربية (Arabic) | 中文版本 (Chinese) | 日本語版 (Japanese) | 한국어 (Korean) | Deutsch (German) | Español (Spanish) | Français (French) | Polski (Polish) | Português Brasileiro (Portuguese) | Русский (Russian)

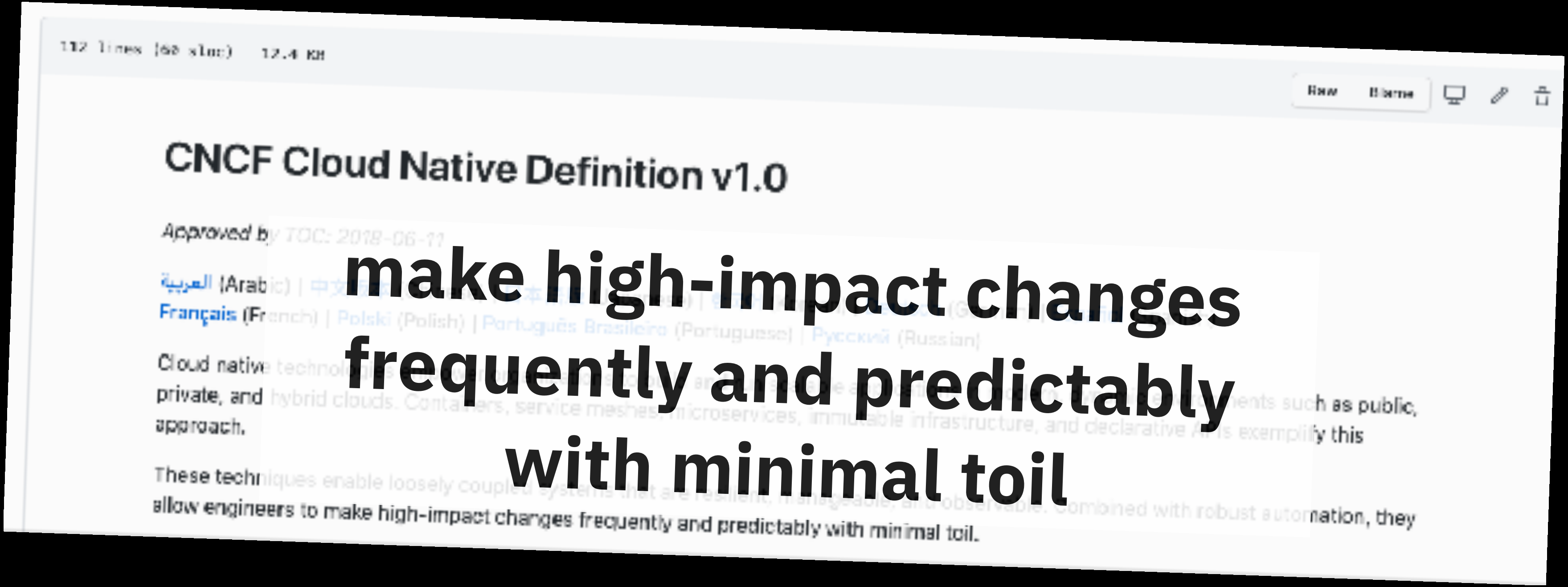
Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

2020

IBM **Garage**

@holly_cummins



CNCF Cloud Native Definition v1.0

Approved by TOC: 2018-06-11

العربية (Arabic) | 中文版本 (Chinese) | 日本語版本 (Japanese) | ... (German) | ...
Français (French) | Polski (Polish) | Português Brasileiro (Portuguese) | Русский (Russian)

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

make high-impact changes frequently and predictably with minimal toil

2020

IBM **Garage**

@holly_cummins

why cloud?

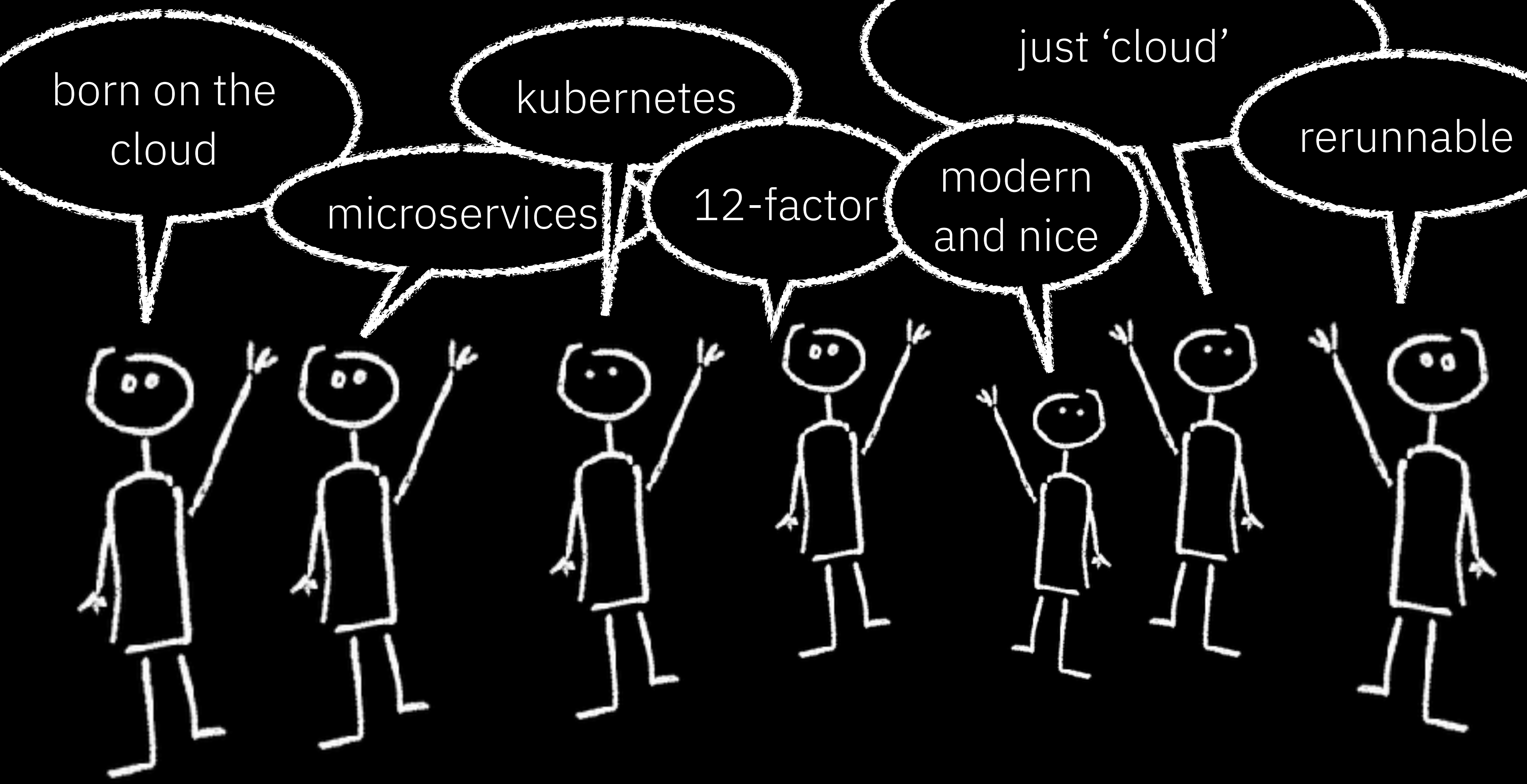
cost

elasticity

speed

exotic capabilities

why cloud native?


born on the cloud
microservices
kubernetes
12-factor
modern and nice
just ‘cloud’
rerunnable

cloud native is

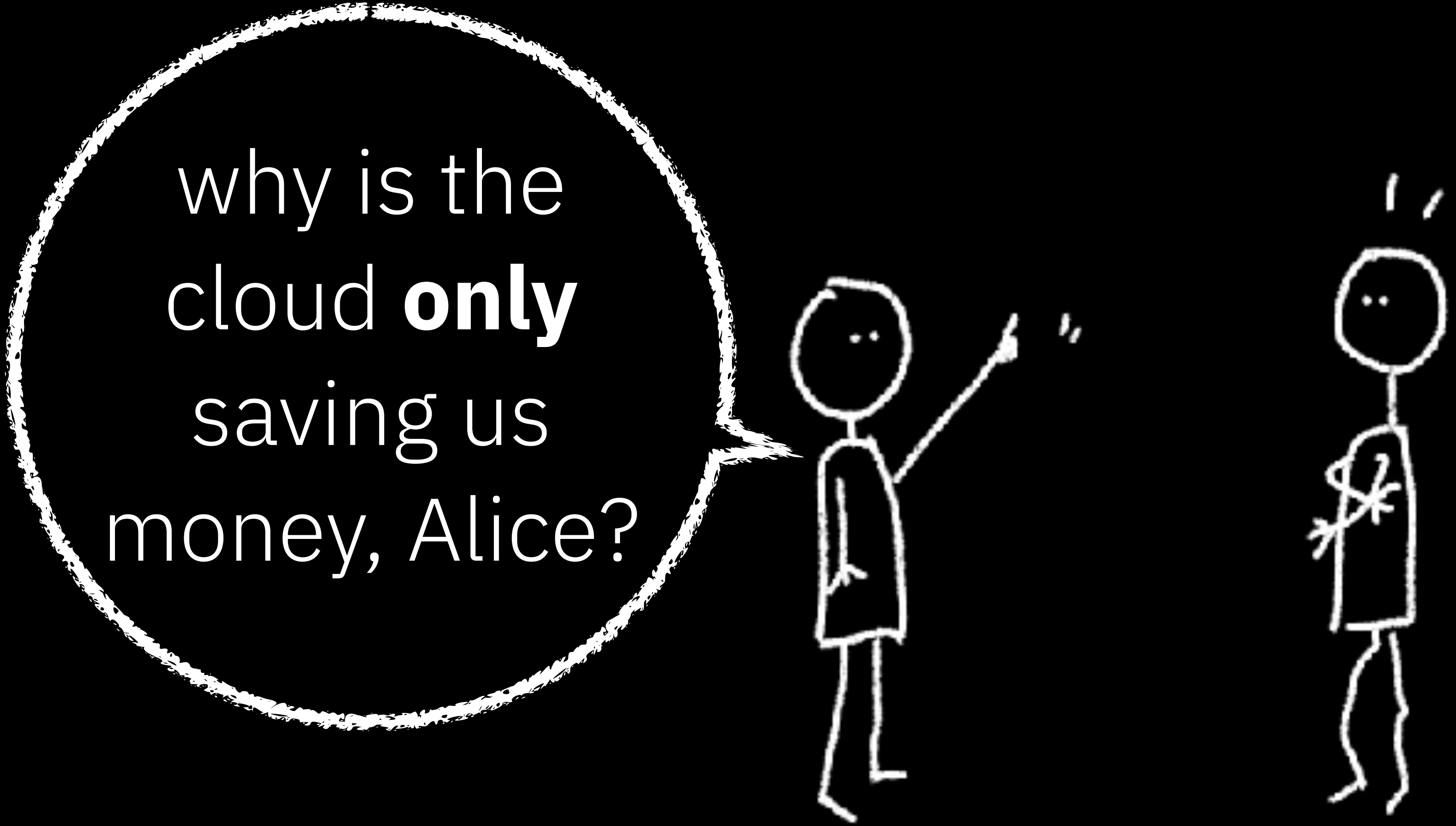
the **combination**

of culture and technology

to achieve cloud goals

are we all agreed
on the goal?

why are there no **microservices** in this cloud native app Alice?



why is the cloud **only** saving us money, Alice?

microservices
are not the **goal**

microservices
are not the **goal**

they are the **means**

“we’re going too slowly.

we need to get rid of COBOL and make microservices!”

“we’re going too slowly.

we need to get rid of COBOL
and make microservices!”

“... but our release board
only meets twice a year.”

containers are a good base

it’s not a **competition** to see how many you can have

containers are a good base

distributed monolith

distributed monolith

but without compile-time checking

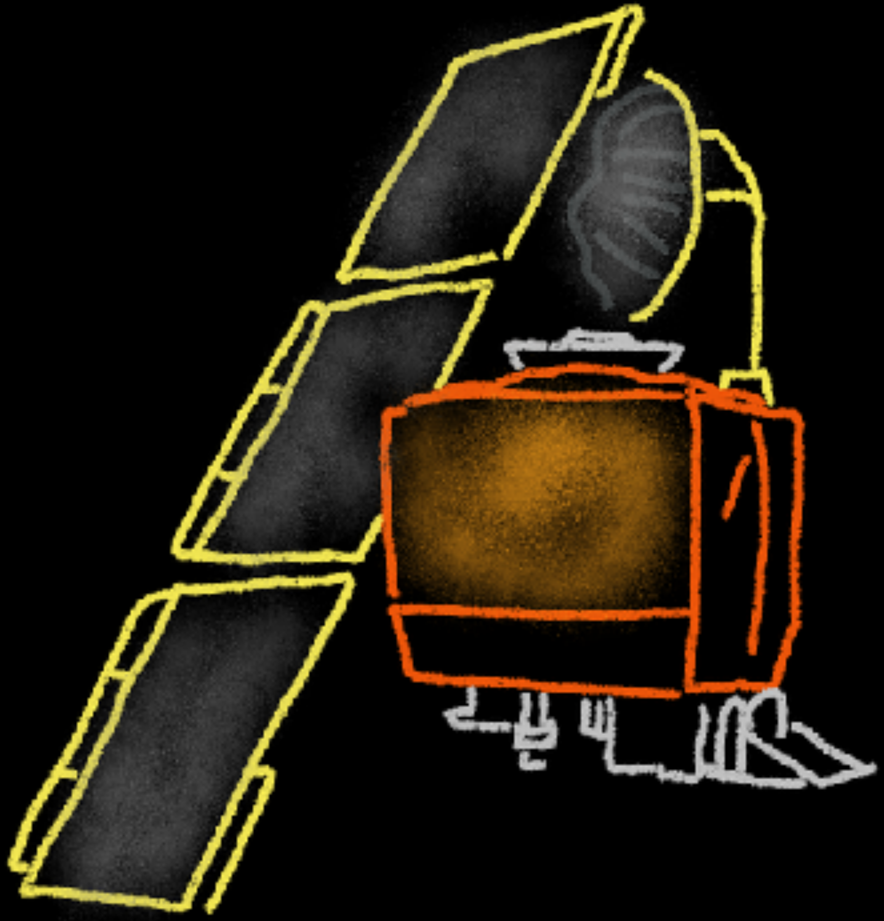
… or guaranteed function execution

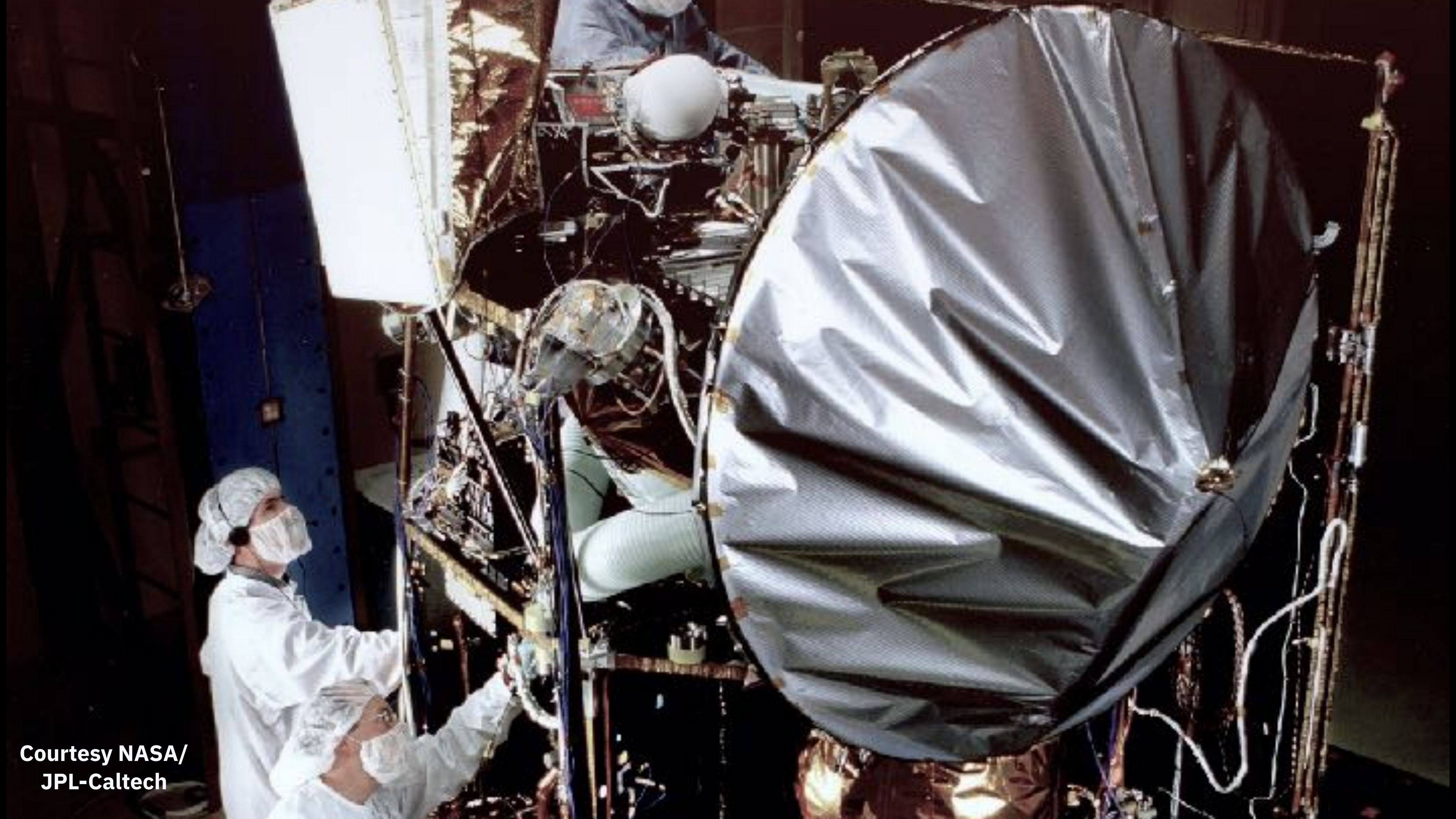
“every time we change one microservice, another breaks”

distributed != decoupled

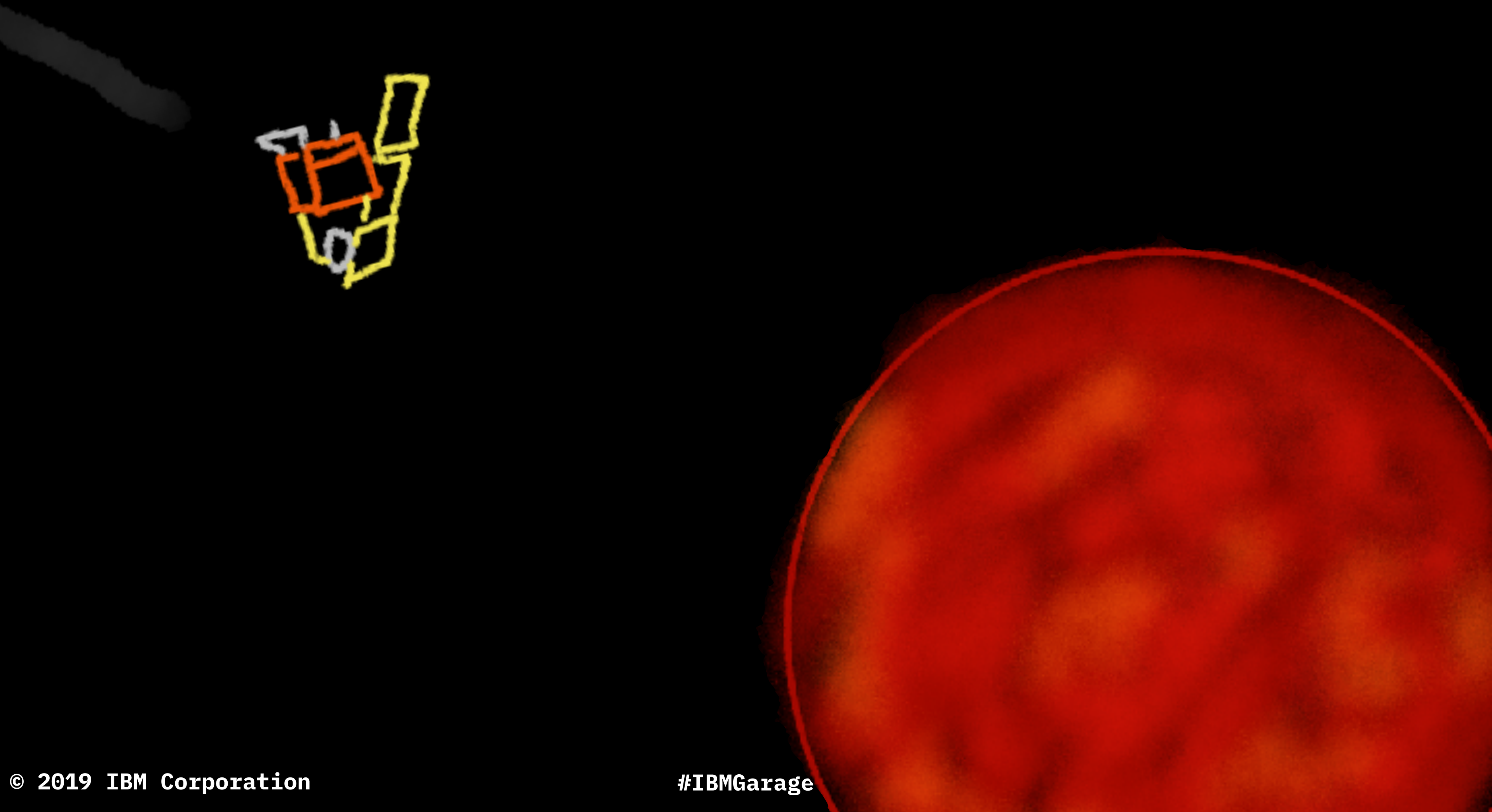
cloud-native spaghetti is still spaghetti

(Image: Cloudy with a Chance of Meatballs.)

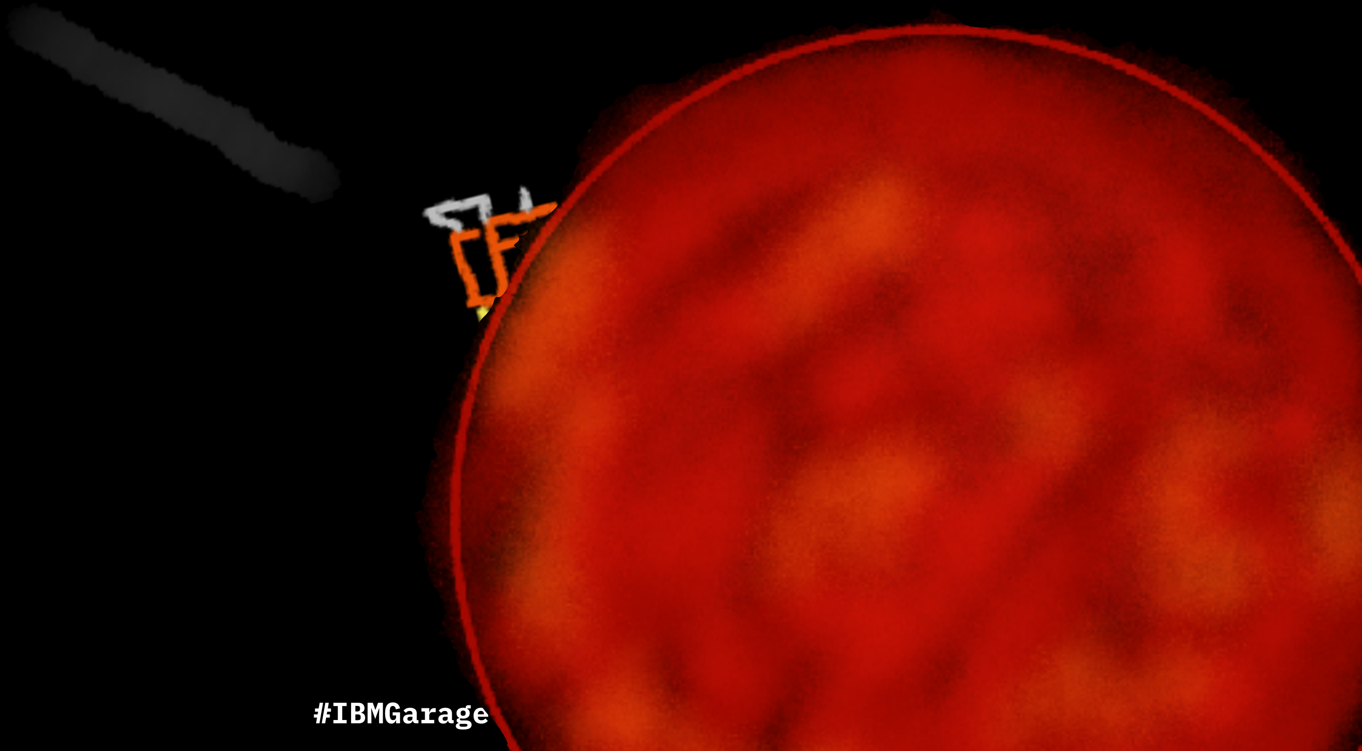


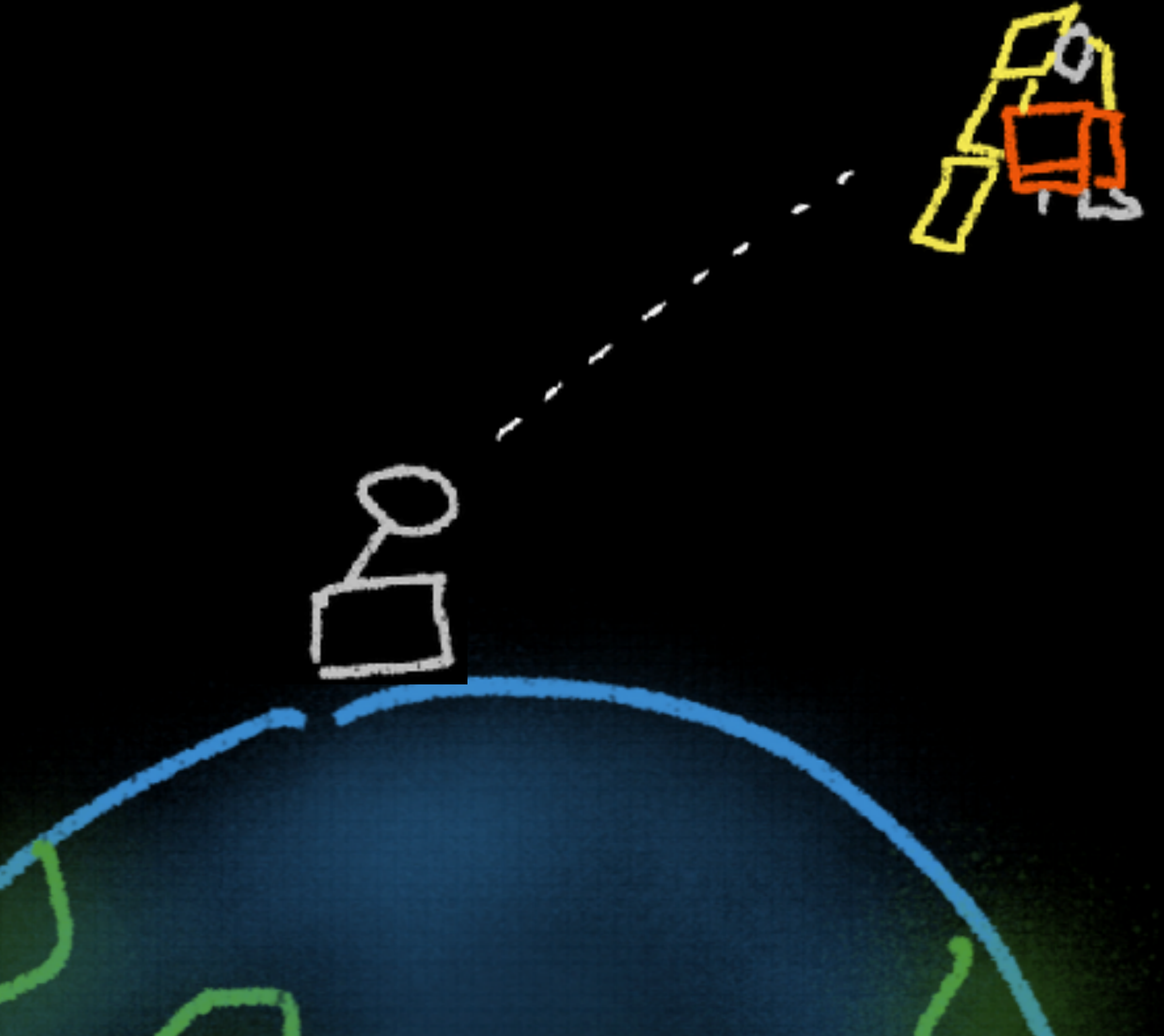


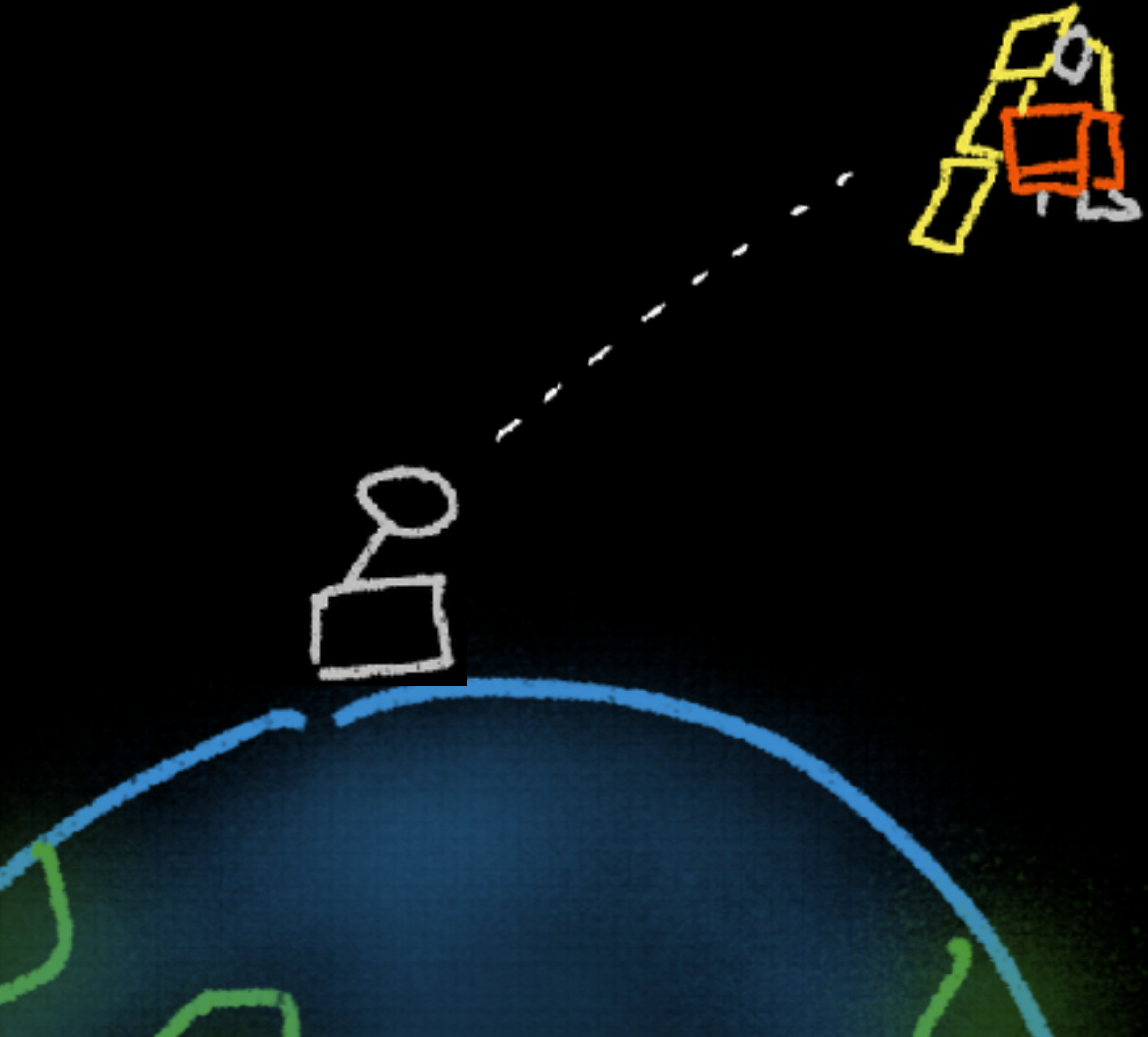
Courtesy NASA/
JPL-Caltech



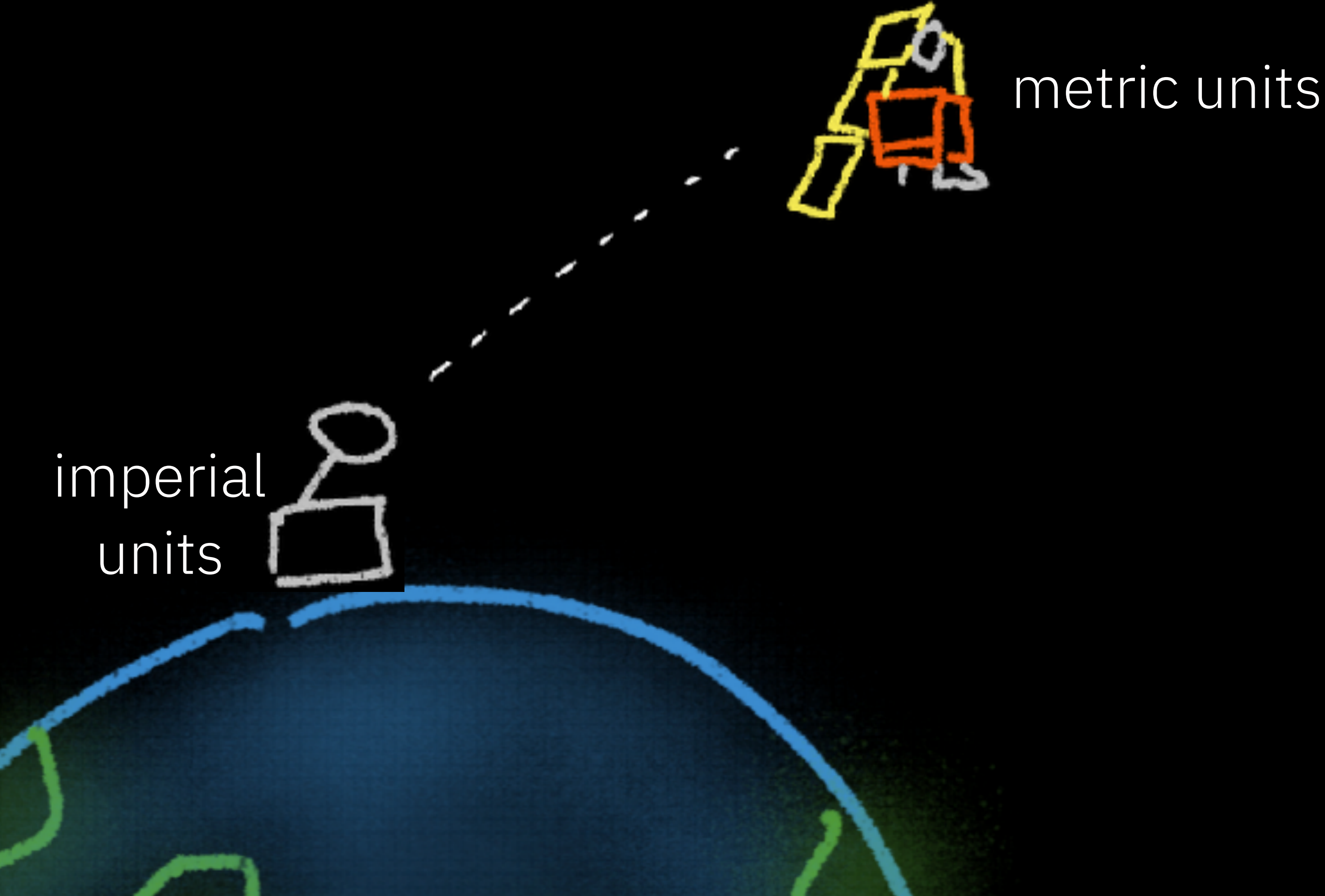
#IBMGarage





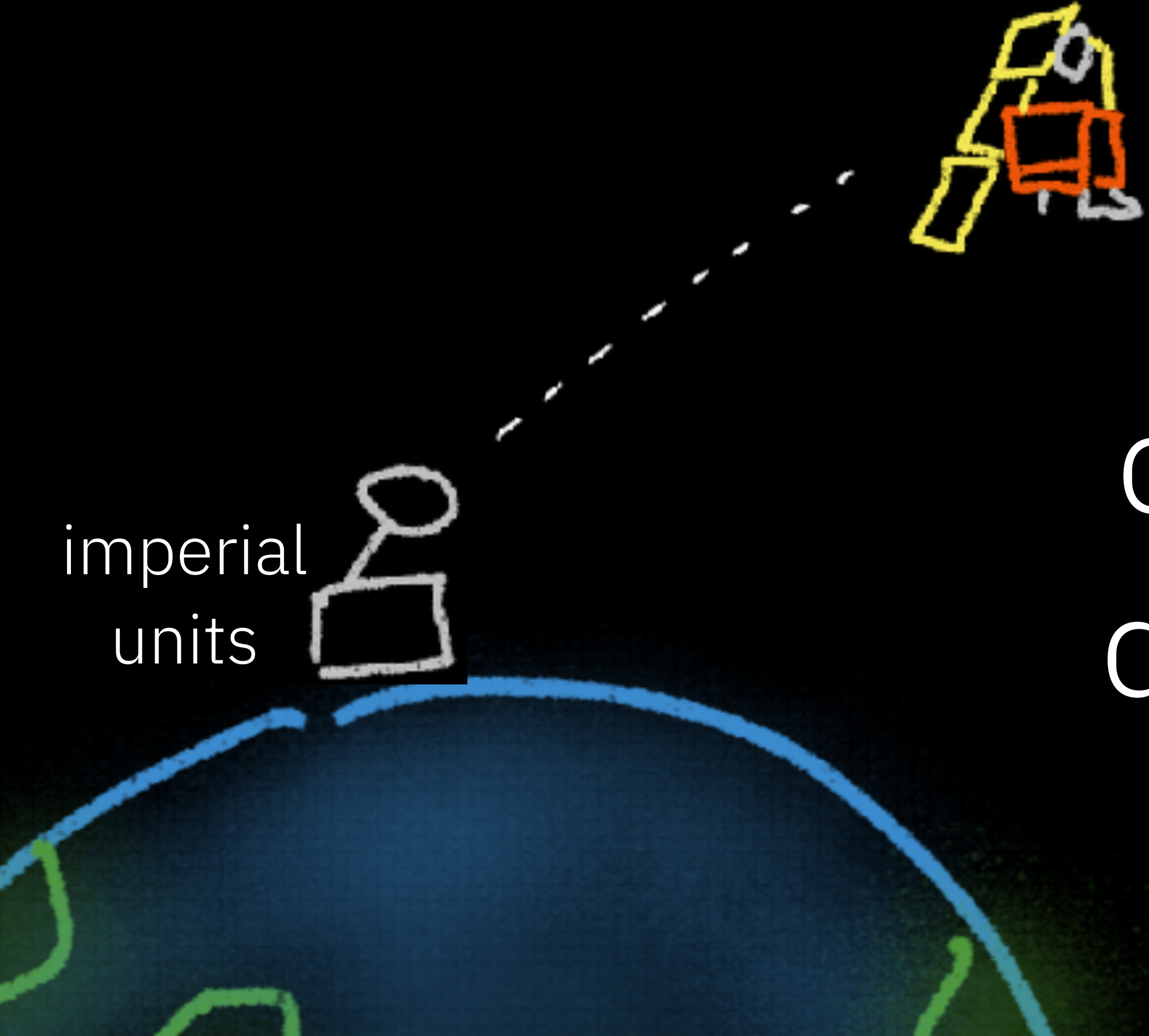


metric units



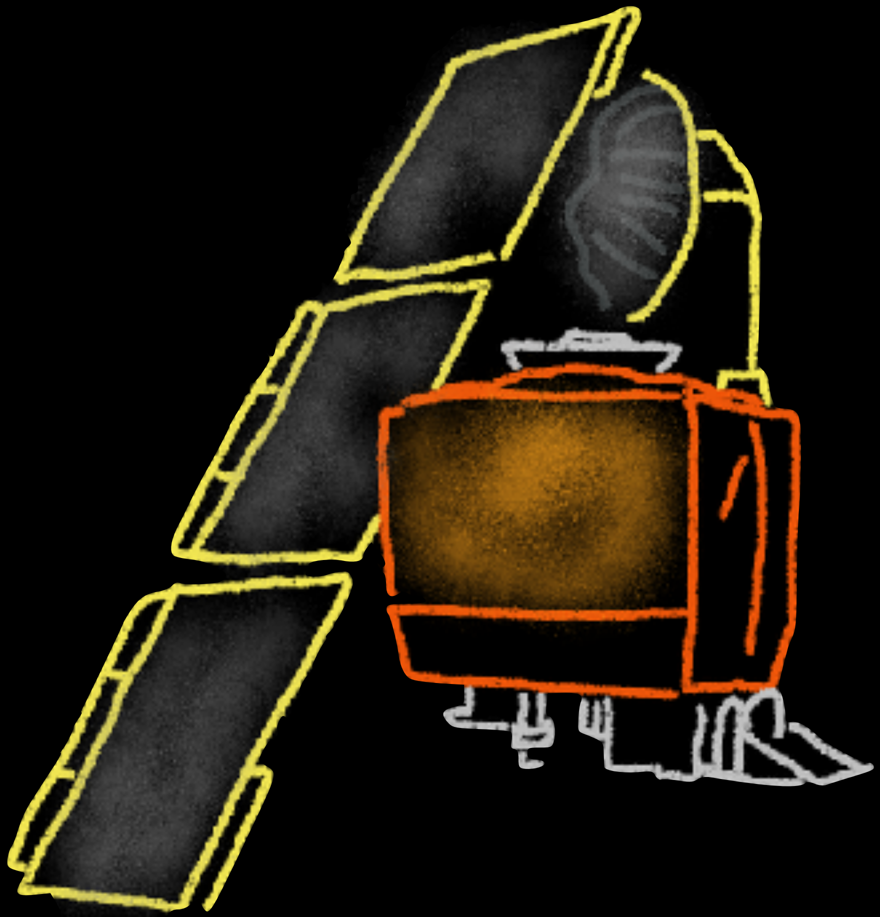
metric units

imperial units

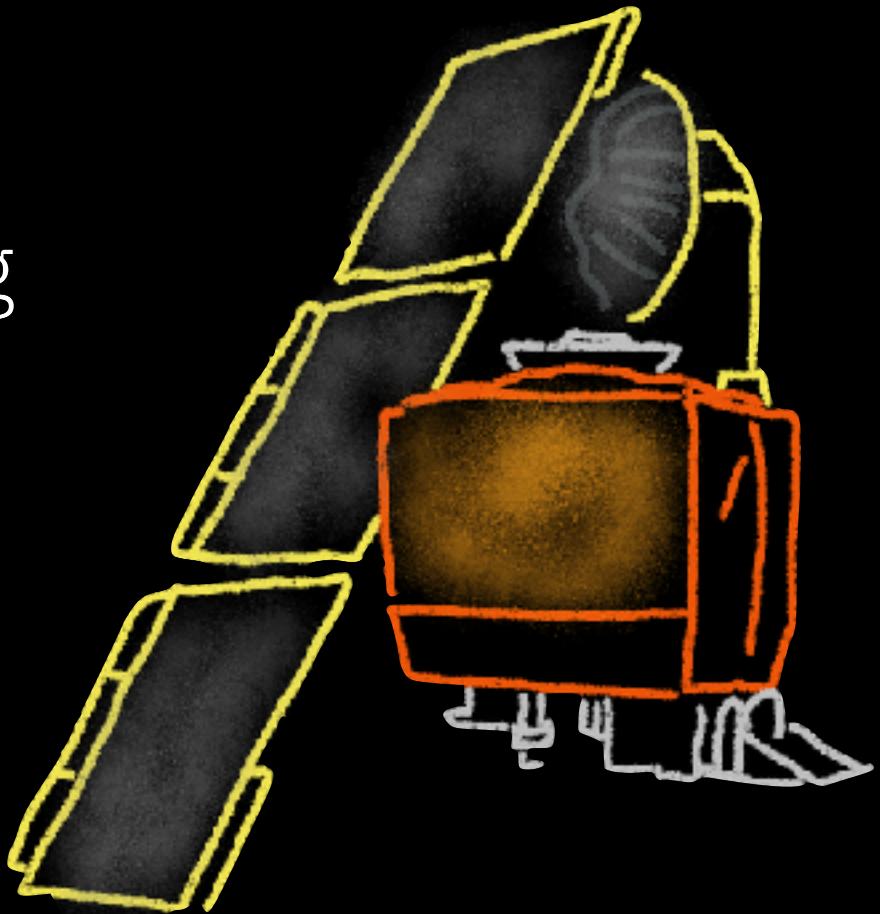


distributing did not help

microservices **need**
consumer-driven contract tests

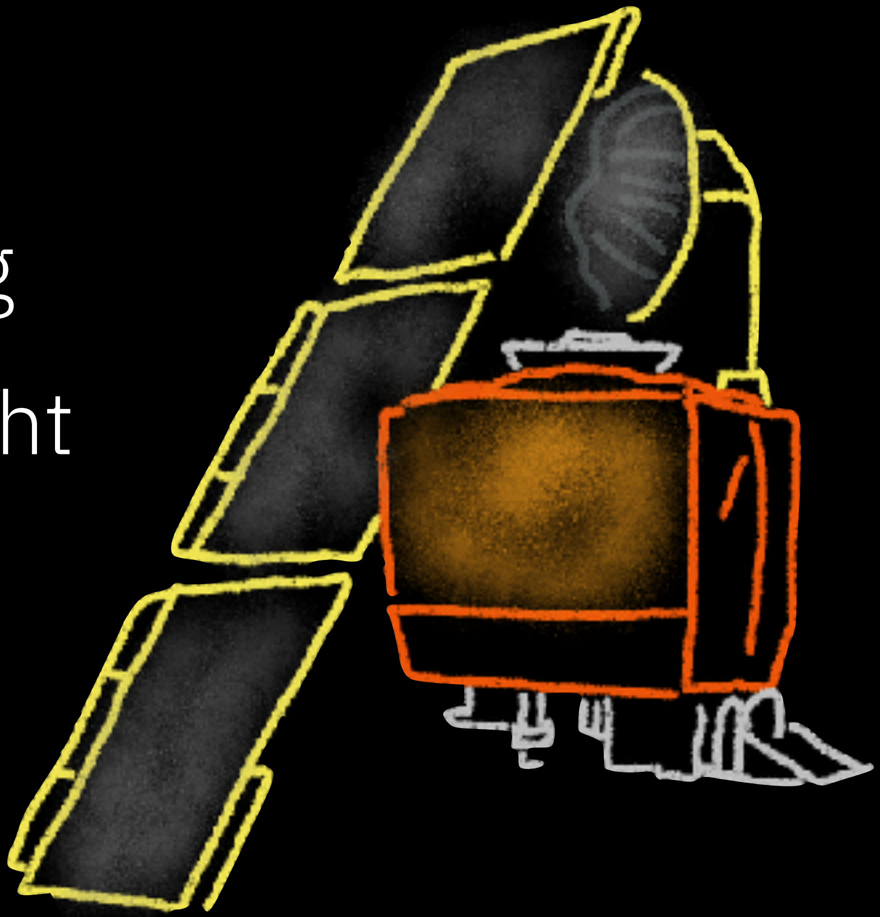


navigators warned something was wrong



navigators warned something was wrong

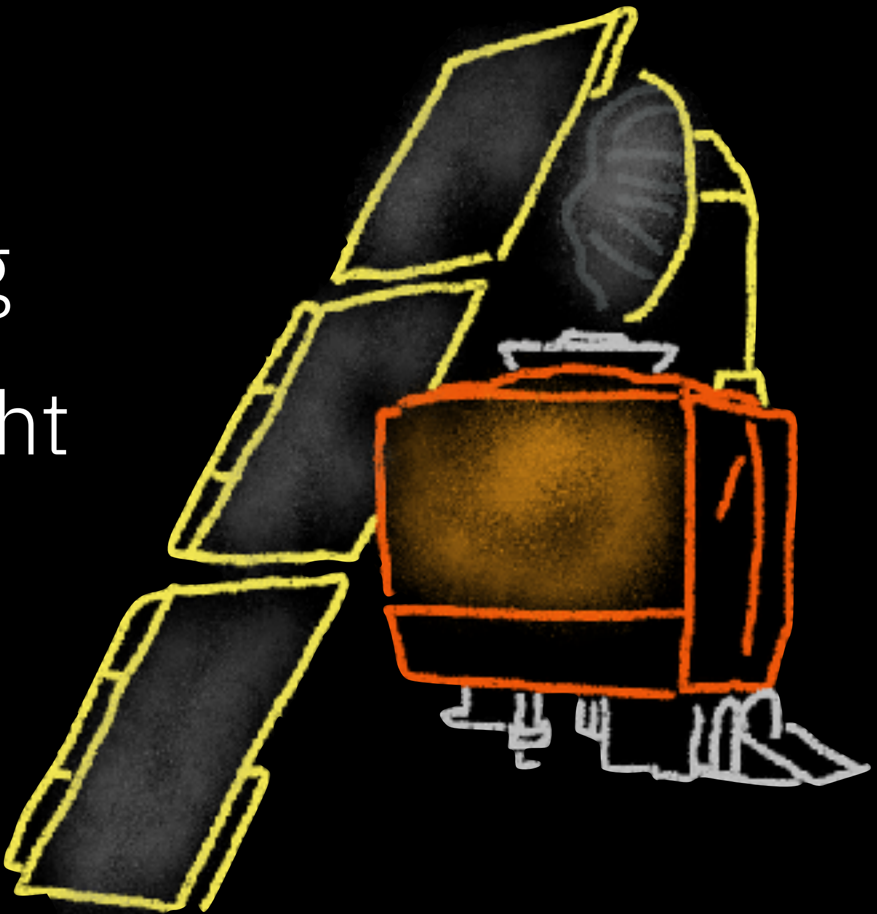
they didn’t fill in the right form



navigators warned something was wrong

they didn't fill in the right form

so nothing was done



quality theatre

the not-actually-continuous continuous integration and continuous deployment

“we have a CI/CD”

CI/CD is something you **do**
not a tool you buy

“i’ll merge my branch into our CI next week”

“CI/CD … CI/CD … CI/CD …

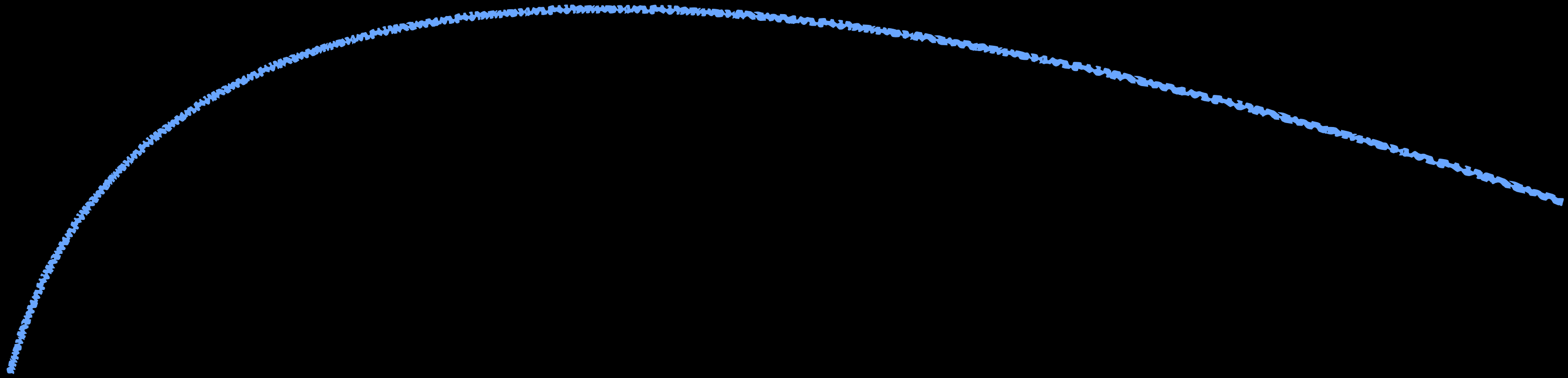
we release every six months …

CI/CD …. ”

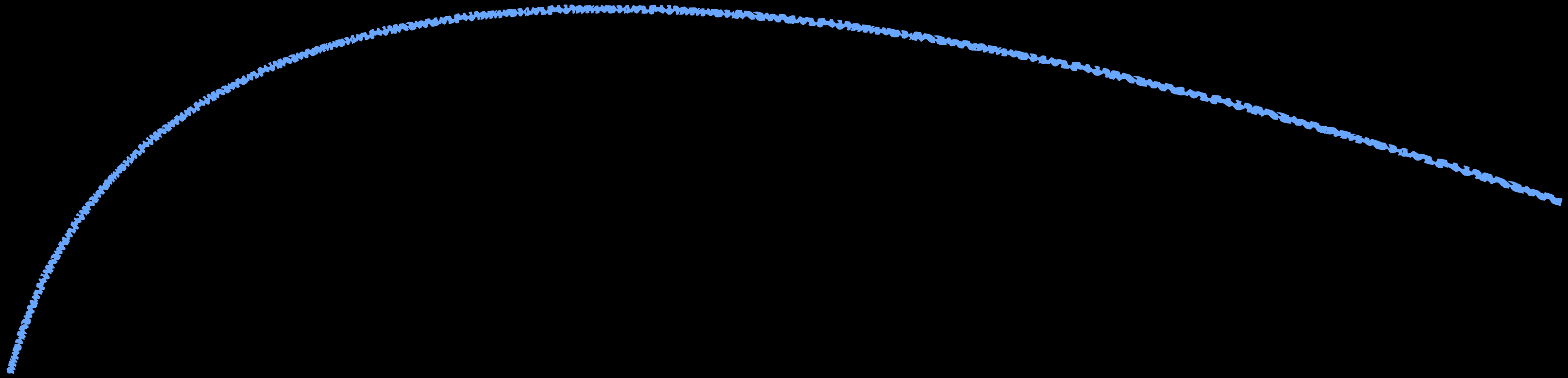
continuous.

I don’t think that word means
what you think it means.

how often should you push to master?

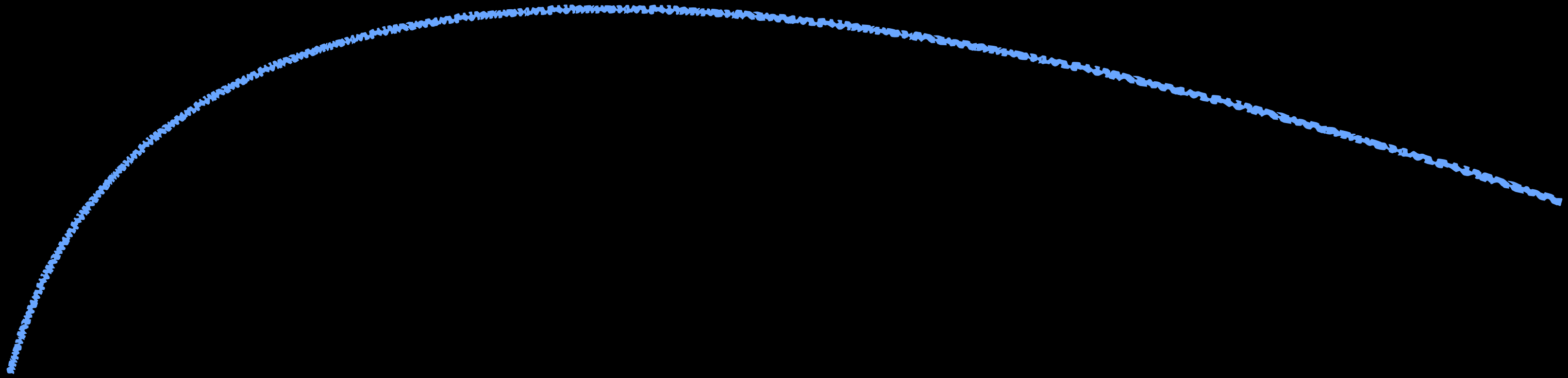


how often should you integrate?



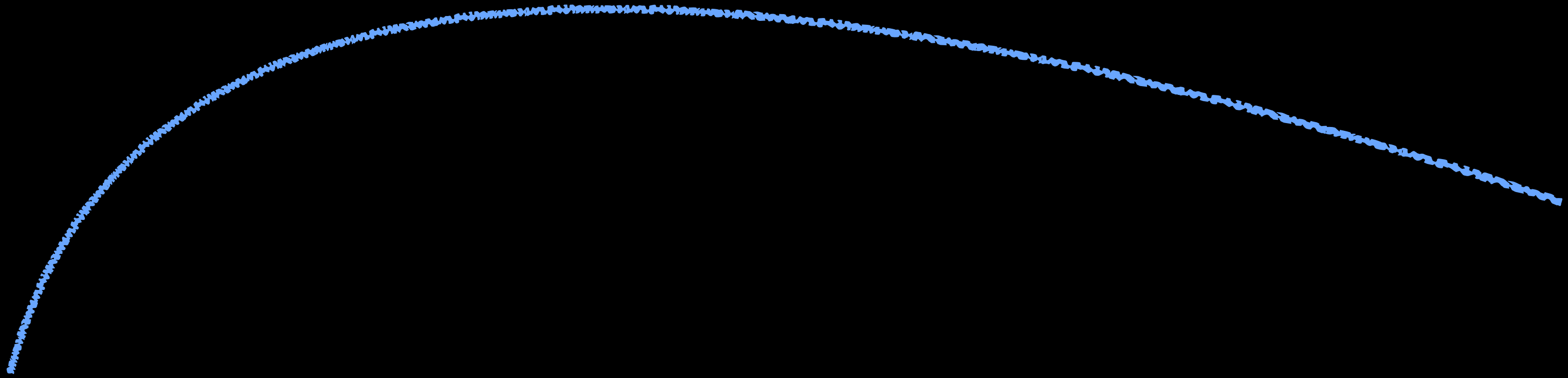
how often should you integrate?

every character



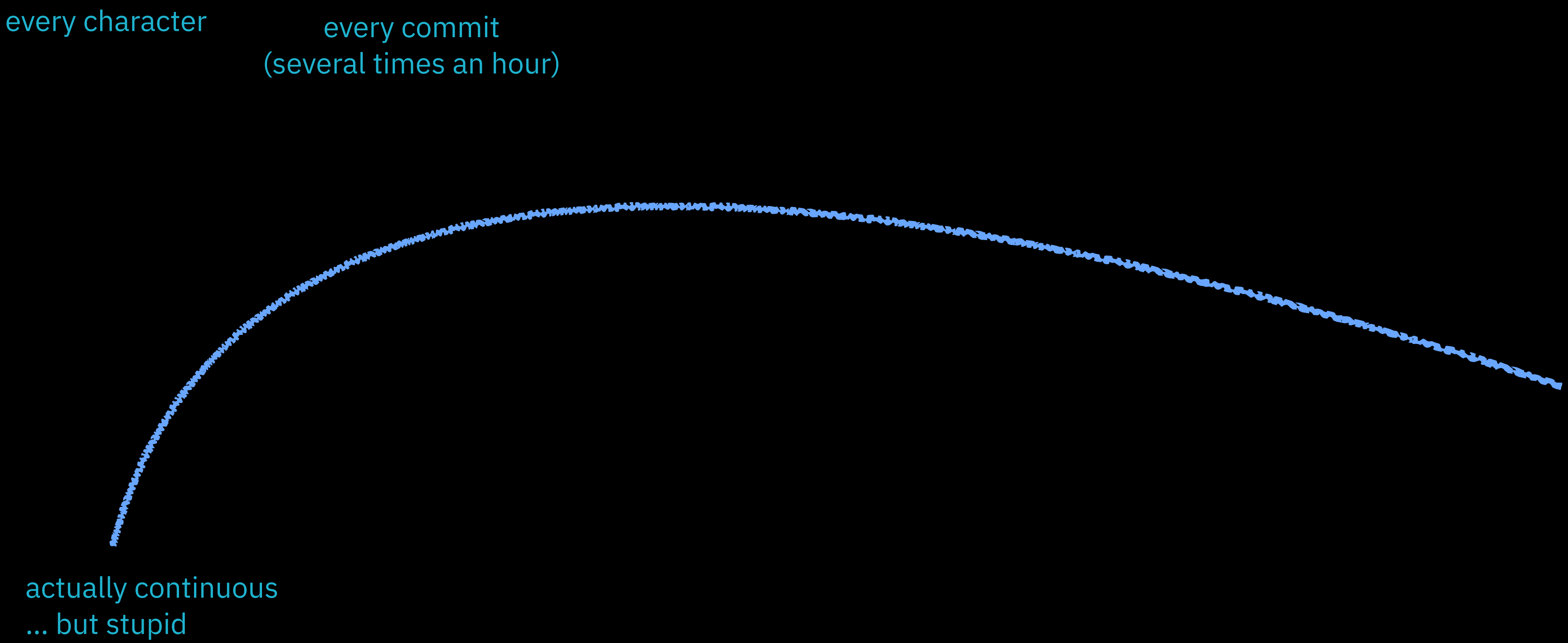
how often should you integrate?

every character

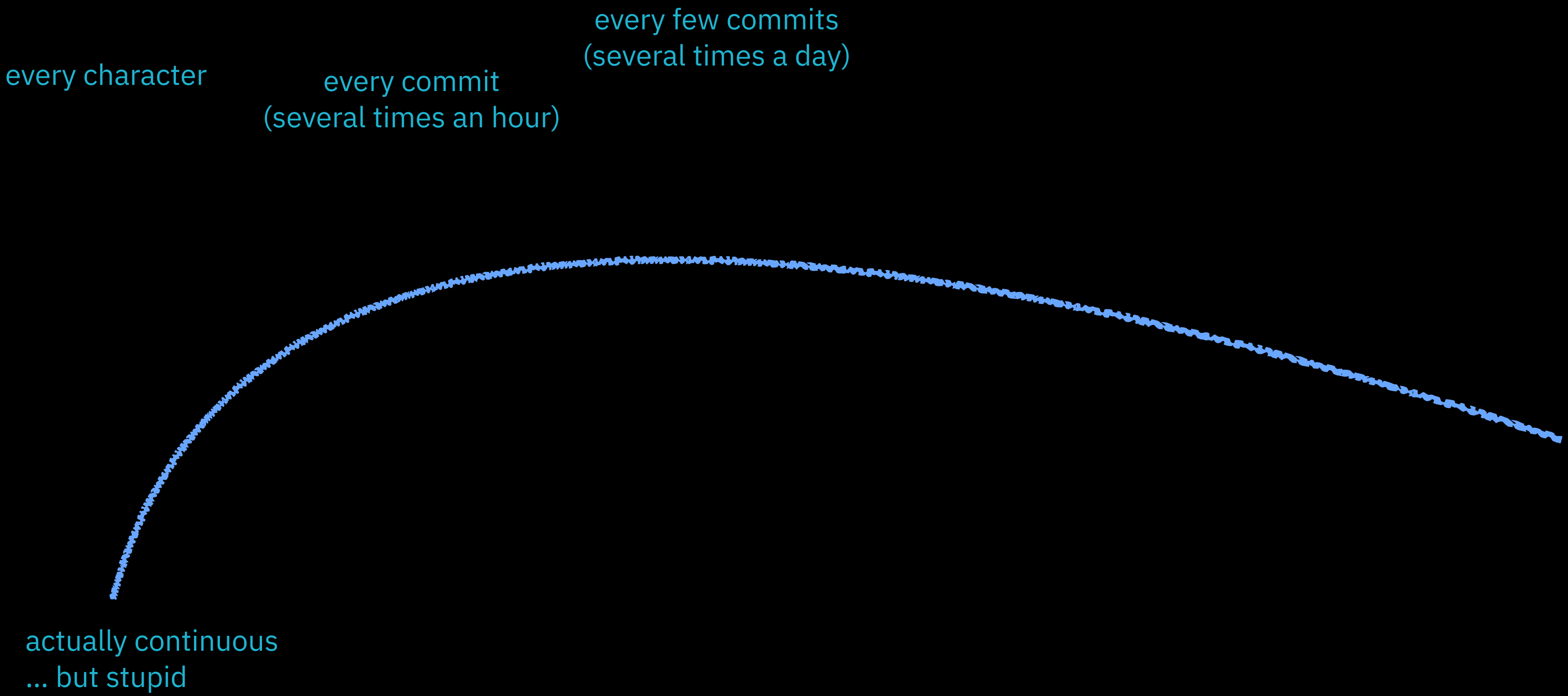


actually continuous
... but stupid

how often should you integrate?



how often should you integrate?



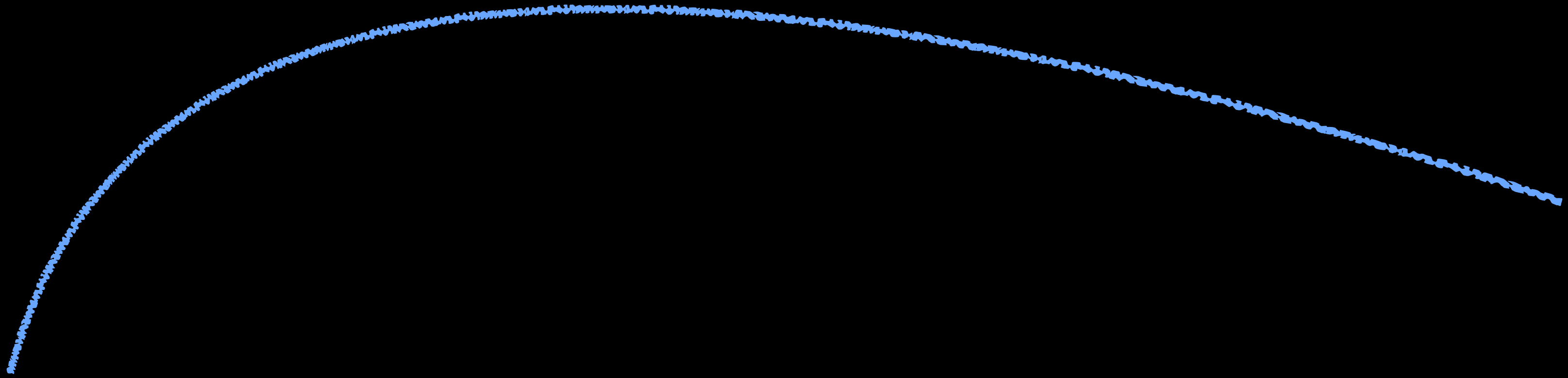
every character

every commit
(several times an hour)

every few commits
(several times a day)

actually continuous
… but stupid

how often should you integrate?



every character

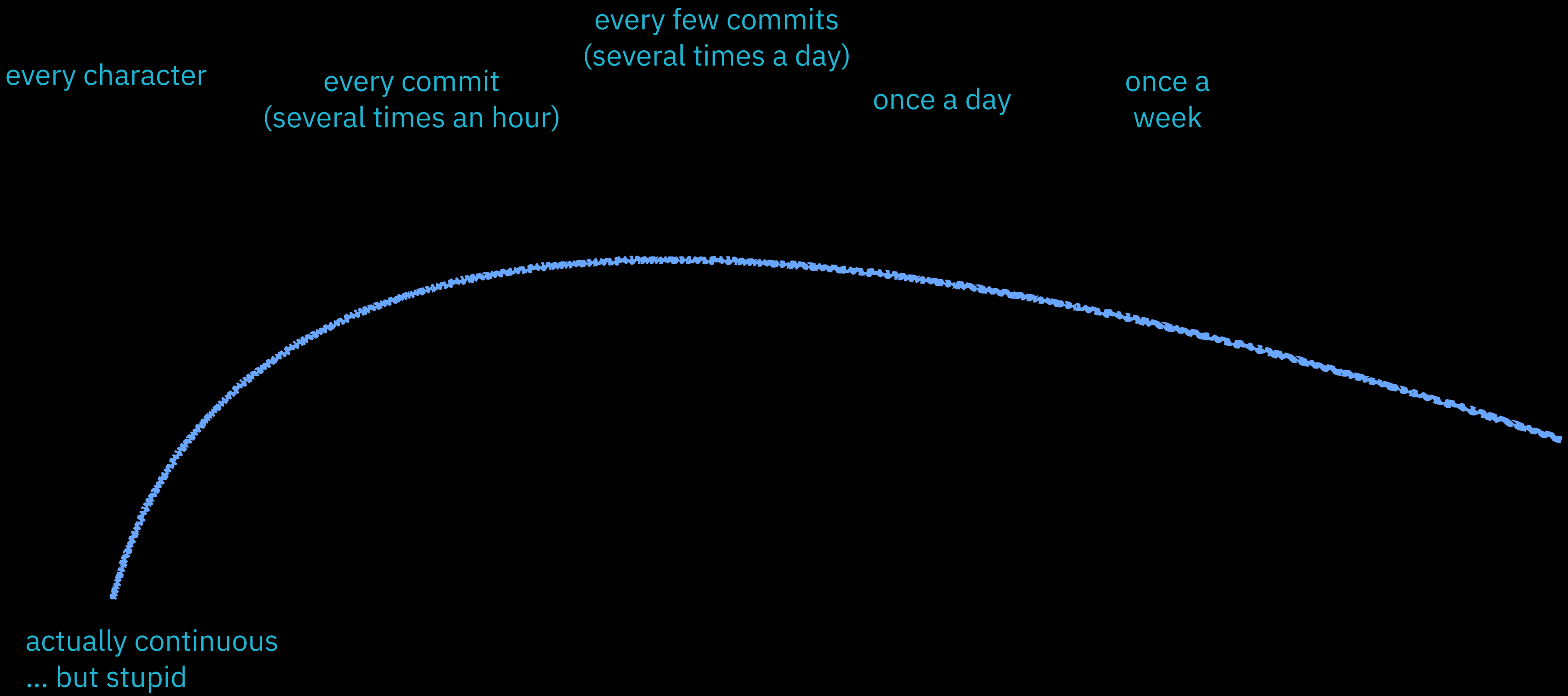
every commit
(several times an hour)

every few commits
(several times a day)

once a day

actually continuous
… but stupid

how often should you integrate?



every character

every commit
(several times an hour)

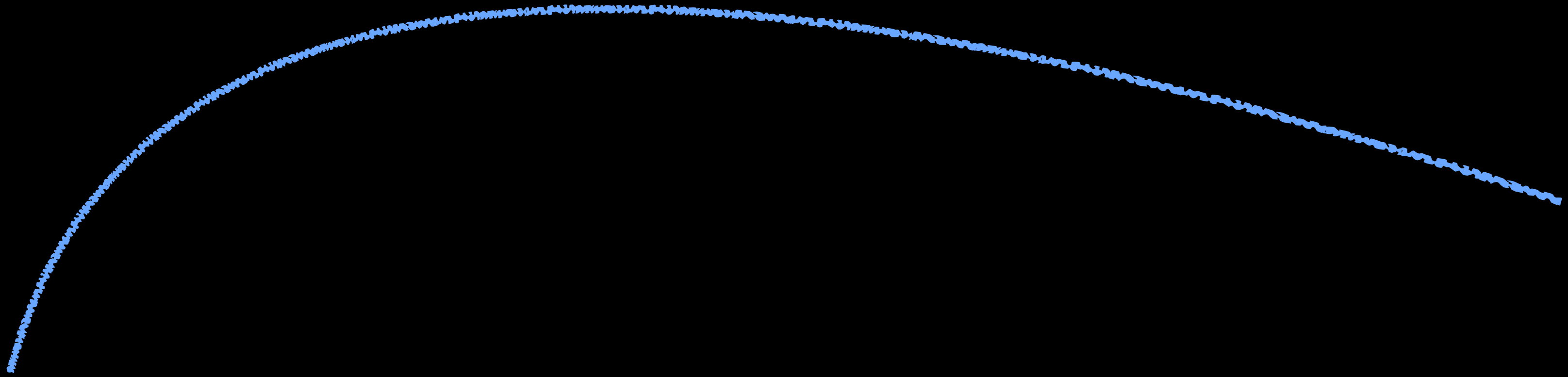
every few commits
(several times a day)

once a day

once a
week

actually continuous
… but stupid

how often should you integrate?



every character

every commit
(several times an hour)

every few commits
(several times a day)

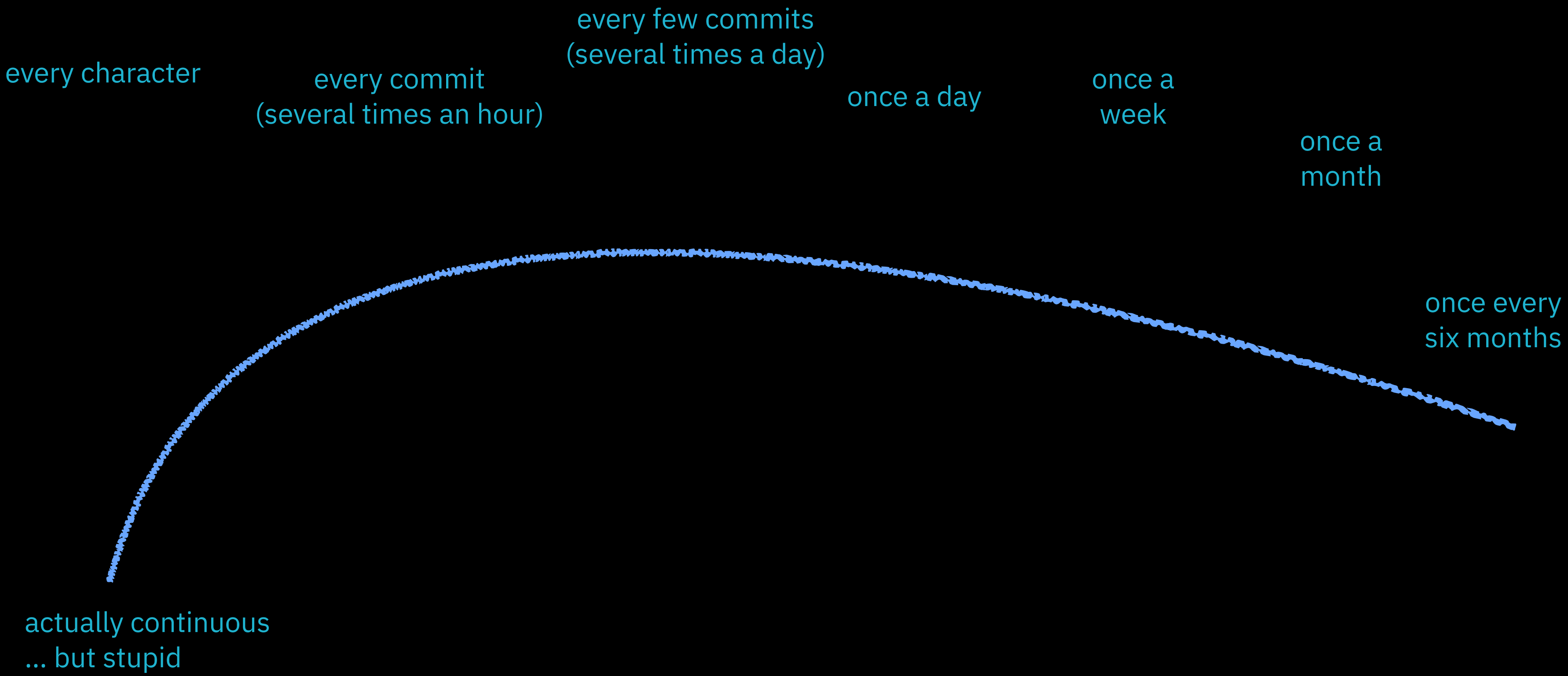
once a day

once a
week

once a
month

actually continuous
… but stupid

how often should you integrate?



every character

every commit
(several times an hour)

every few commits
(several times a day)

once a day

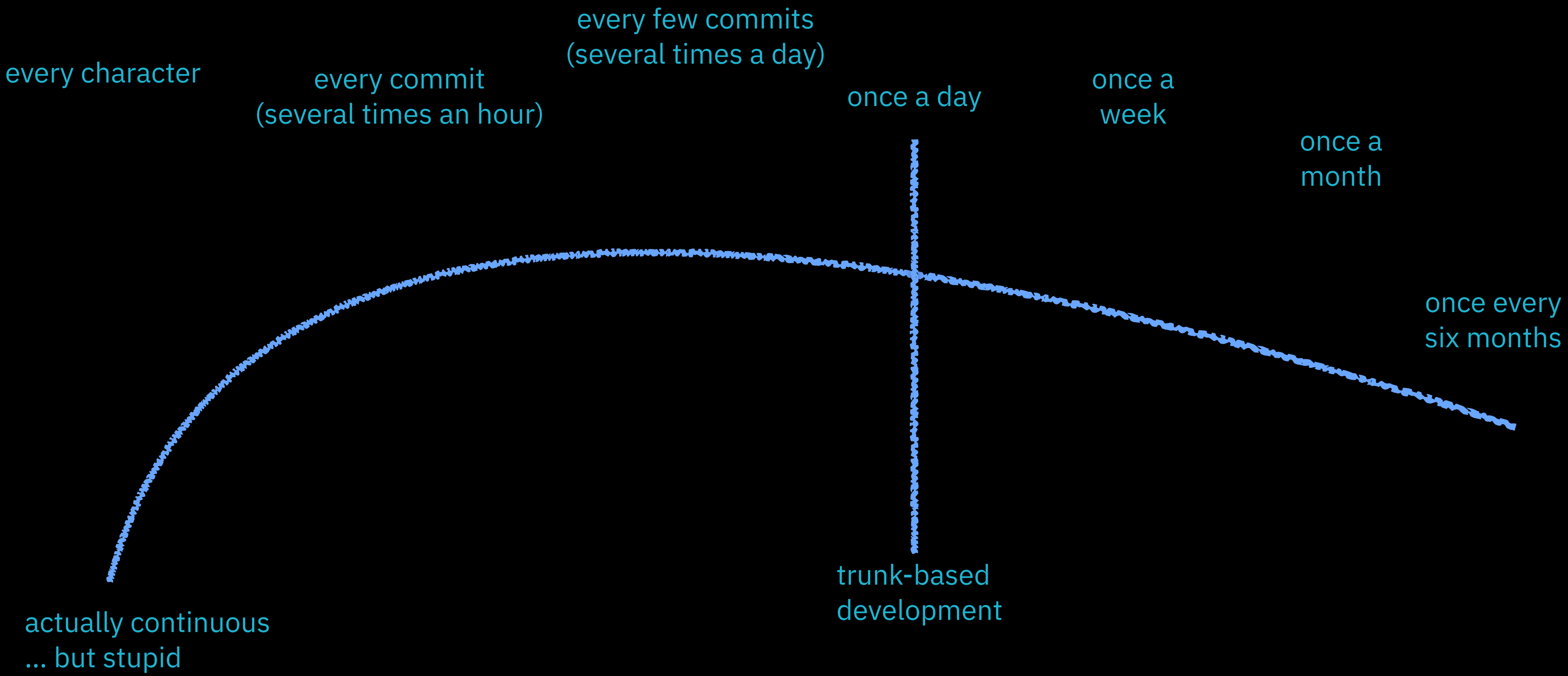
once a
week

once a
month

once every
six months

actually continuous
… but stupid

how often should you integrate?



every character

every commit
(several times an hour)

every few commits
(several times a day)

once a day

once a
week

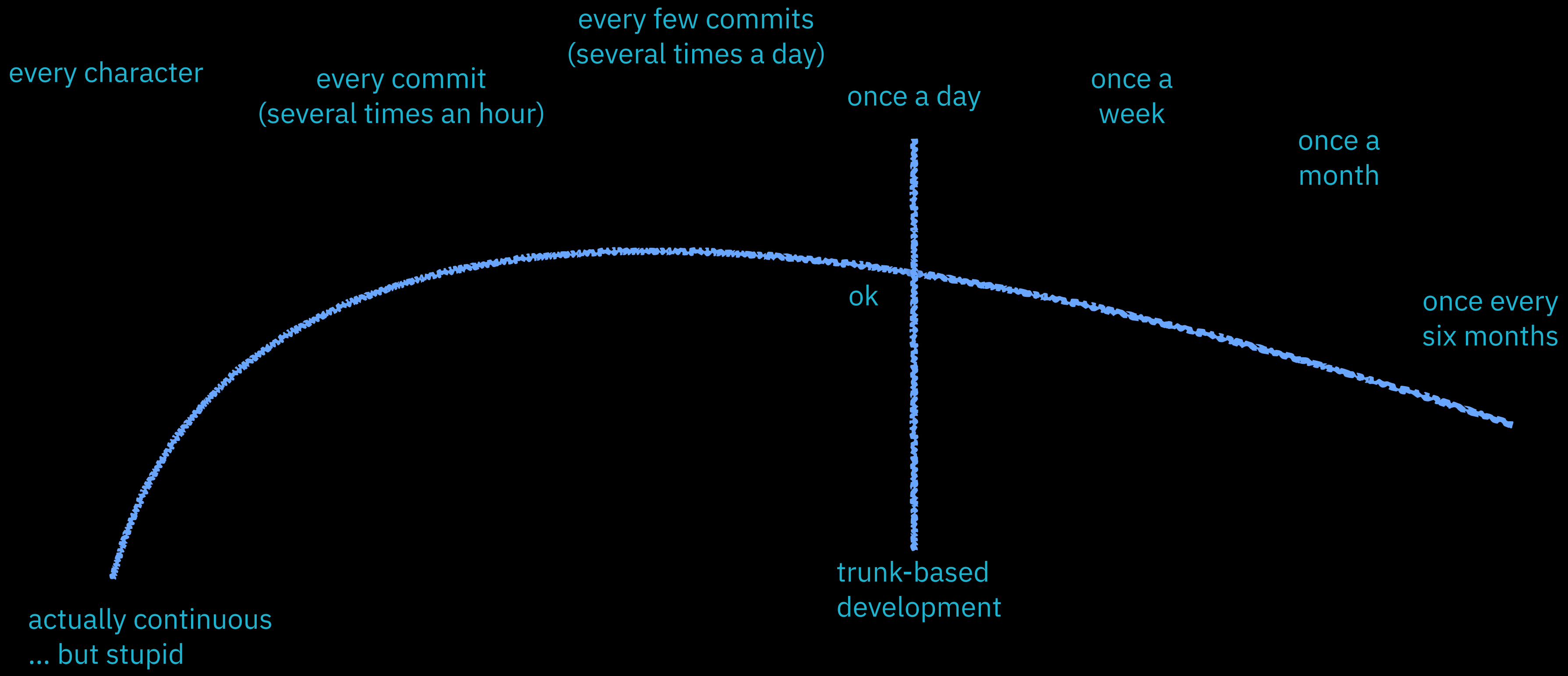
once a
month

once every
six months

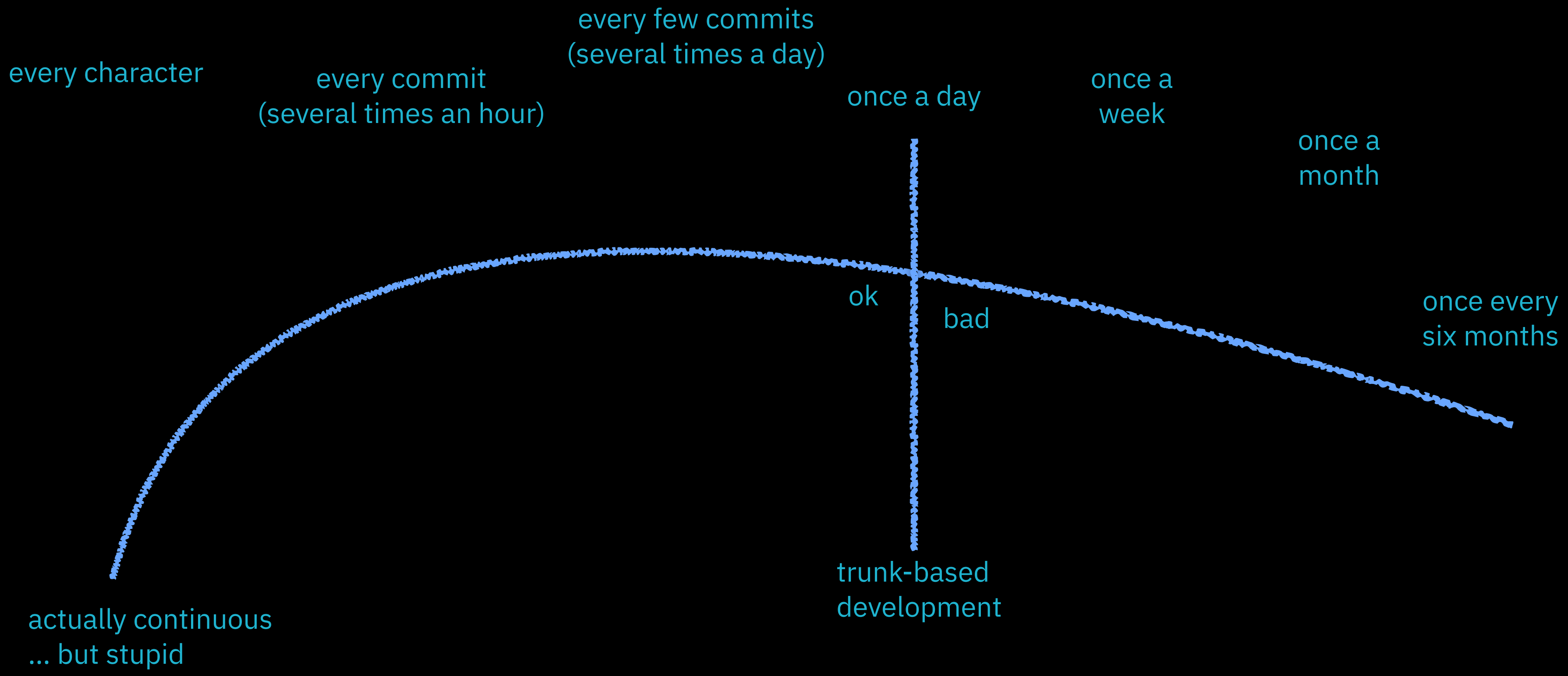
actually continuous
… but stupid

trunk-based
development

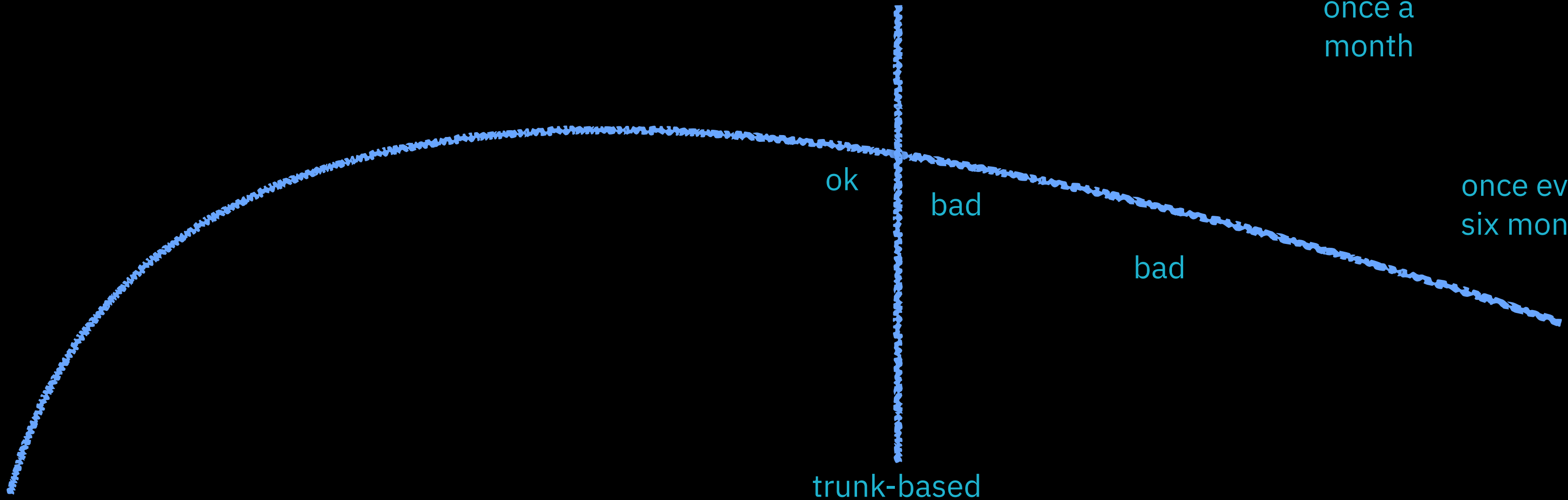
how often should you integrate?



how often should you integrate?



how often should you integrate?



every character

every commit
(several times an hour)

every few commits
(several times a day)

once a day

once a
week

once a
month

ok

bad

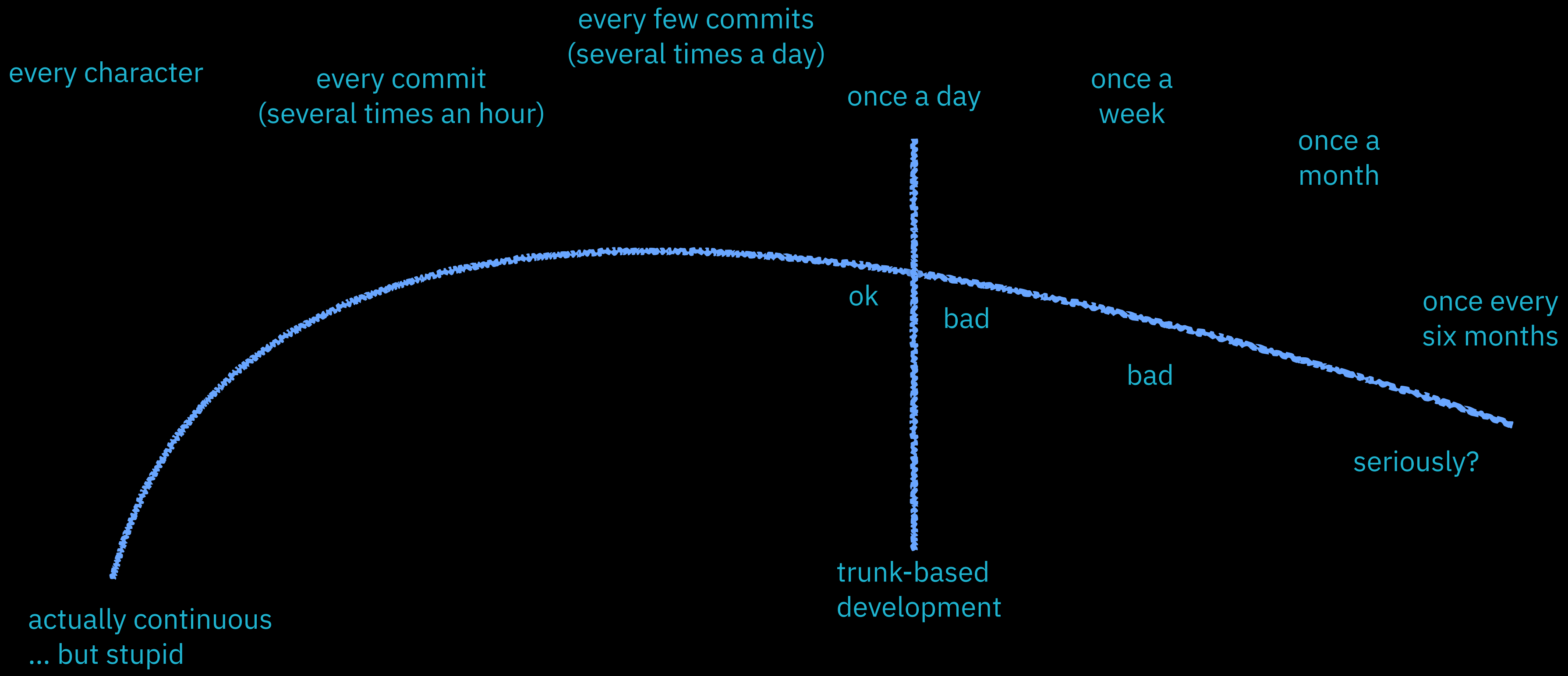
bad

once every
six months

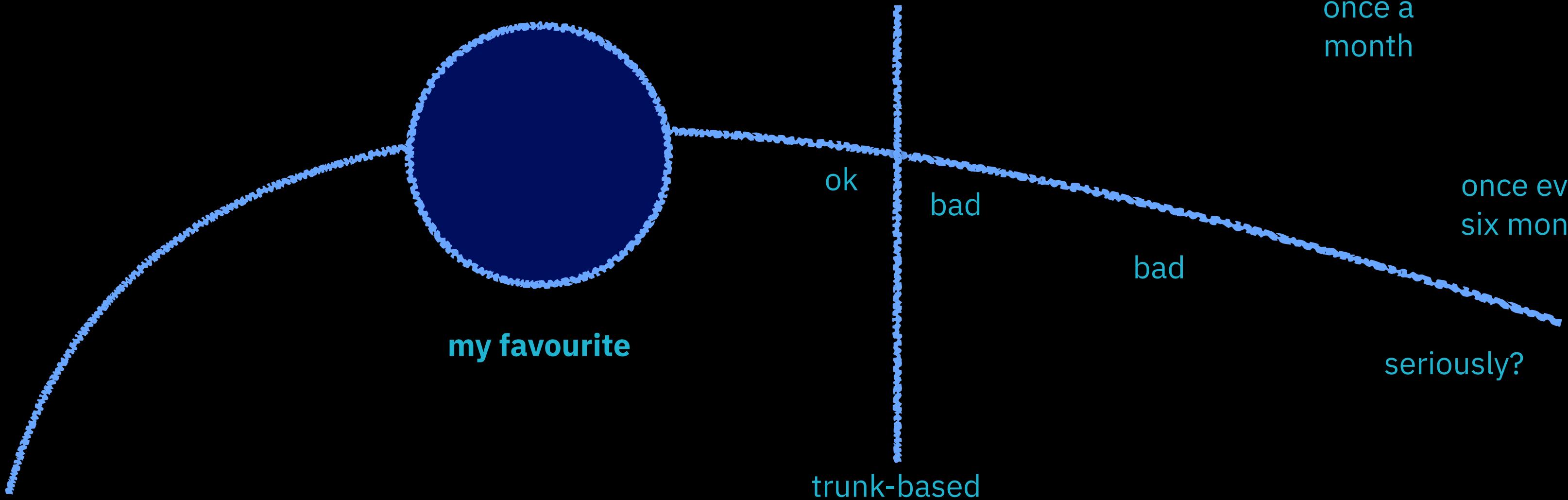
trunk-based
development

actually continuous
… but stupid

how often should you integrate?



how often should you integrate?



every character

every commit
(several times an hour)

every few commits
(several times a day)

once a day

once a
week

once a
month

once every
six months

ok

bad

bad

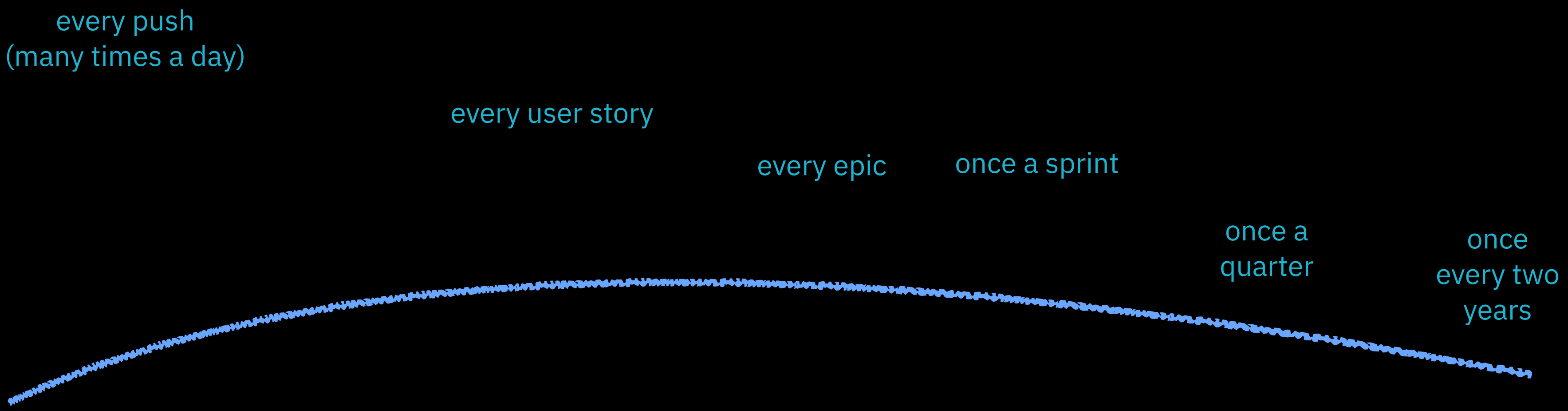
seriously?

my favourite

trunk-based
development

actually continuous
… but stupid

how often should you release?



every push
(many times a day)

every user story

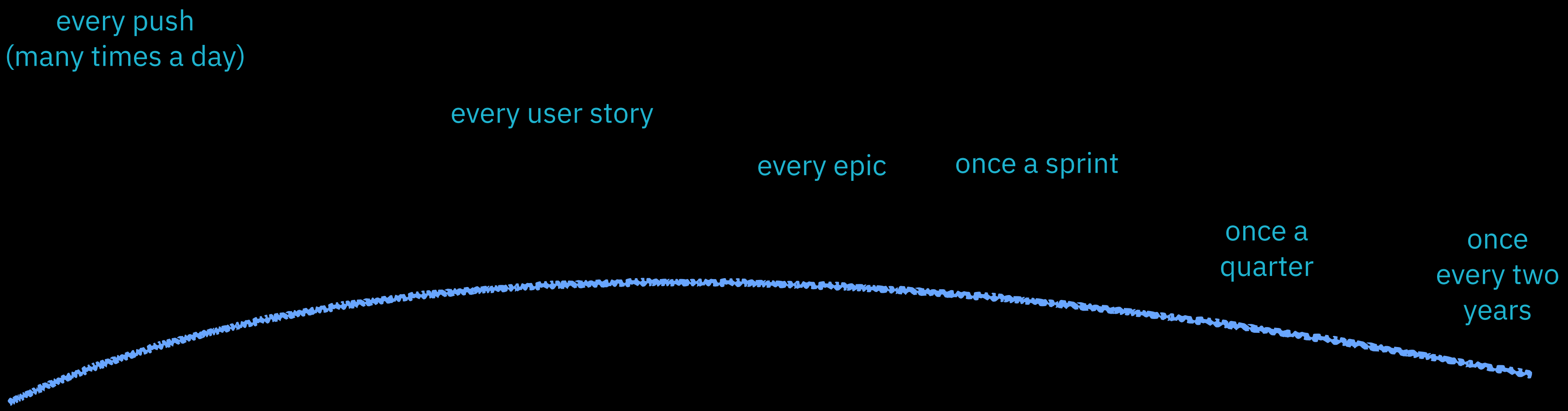
every epic

once a sprint

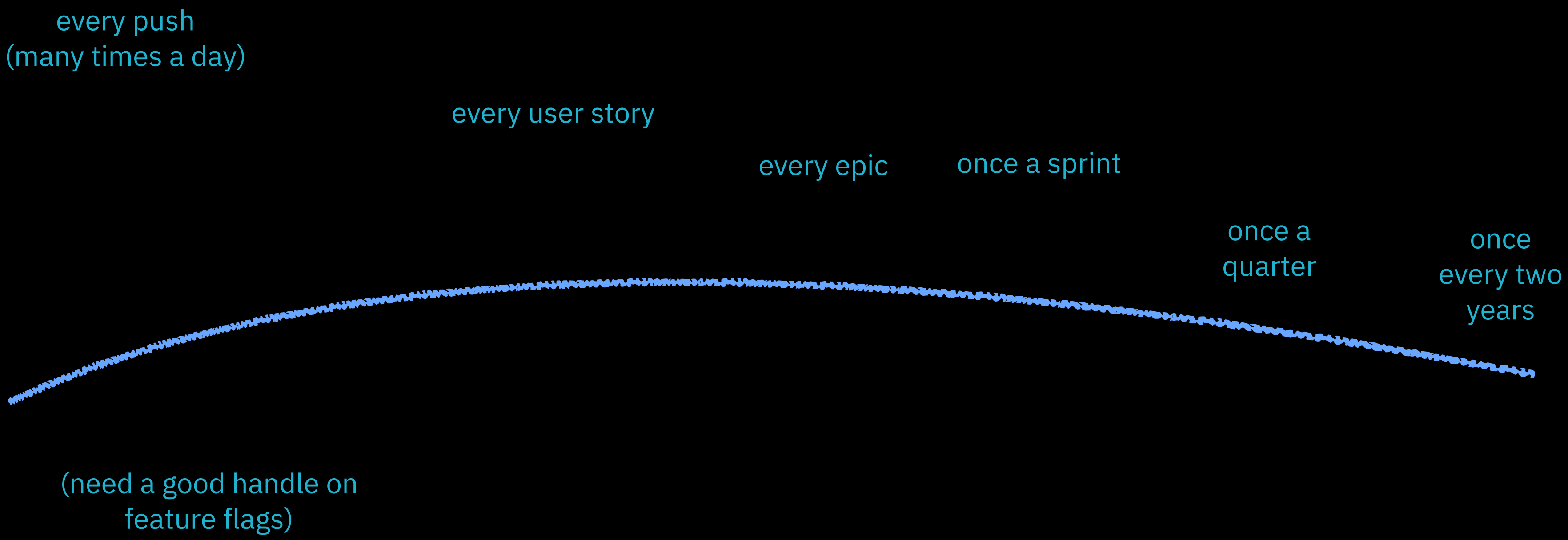
once a
quarter

once
every two
years

how often should you deploy?



how often should you deploy?



every push
(many times a day)

every user story

every epic

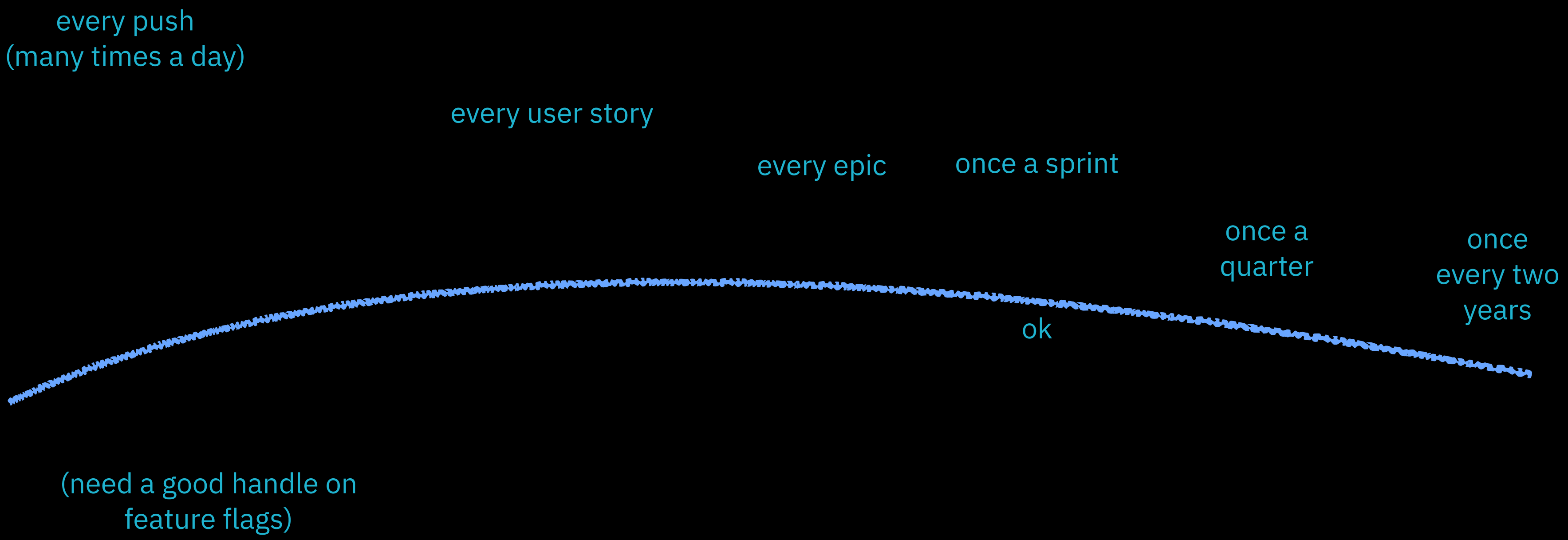
once a sprint

once a
quarter

once
every two
years

(need a good handle on
feature flags)

how often should you deploy?



every push
(many times a day)

every user story

every epic

once a sprint

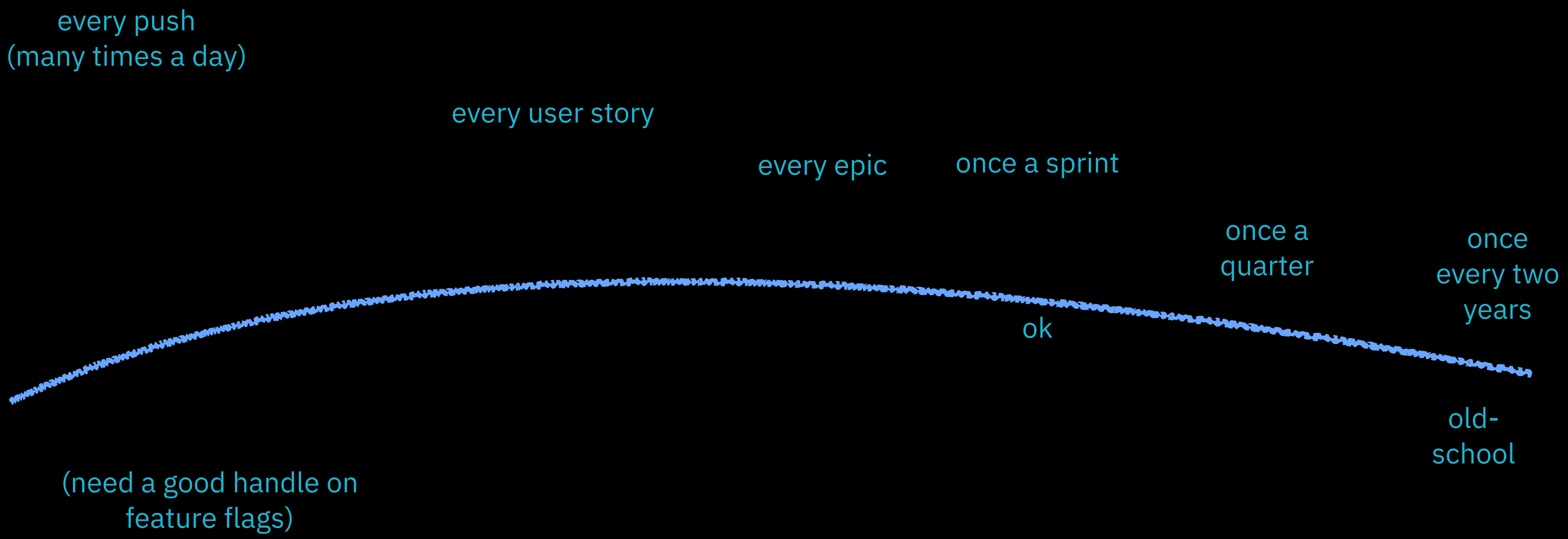
once a
quarter

once
every two
years

ok

(need a good handle on
feature flags)

how often should you deploy?



every push
(many times a day)

every user story

every epic

once a sprint

once a
quarter

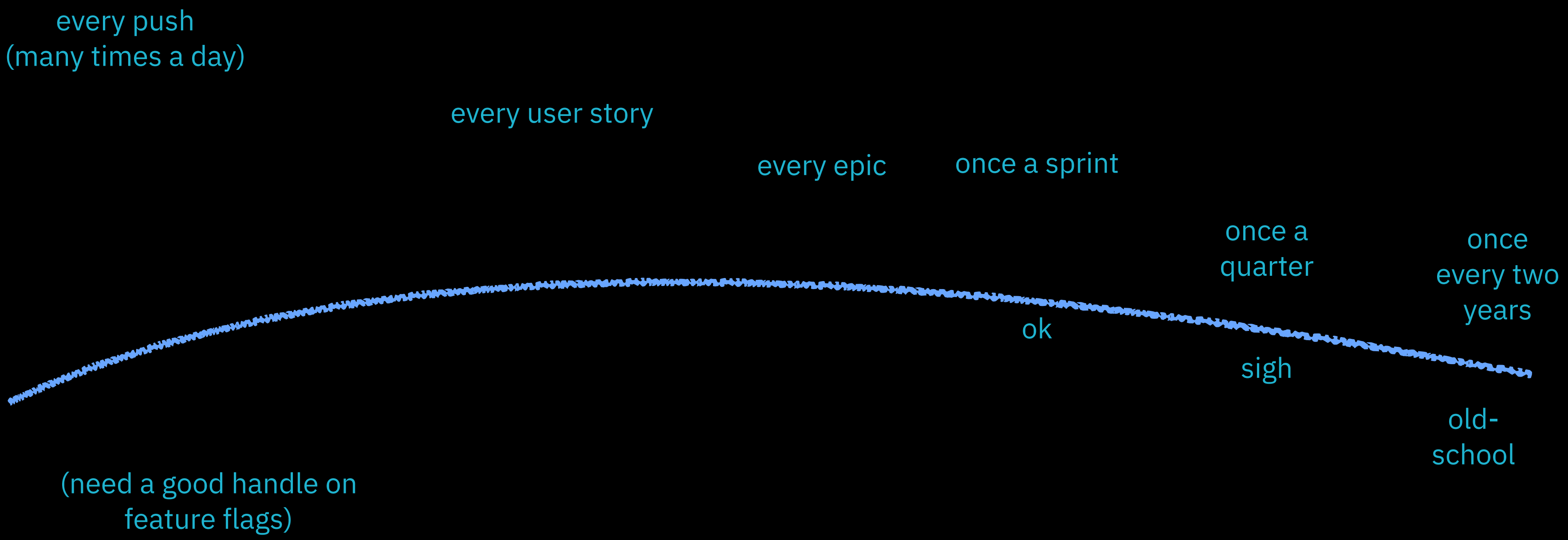
once
every two
years

ok

old-
school

(need a good handle on
feature flags)

how often should you deploy?



every push
(many times a day)

every user story

every epic

once a sprint

once a
quarter

once
every two
years

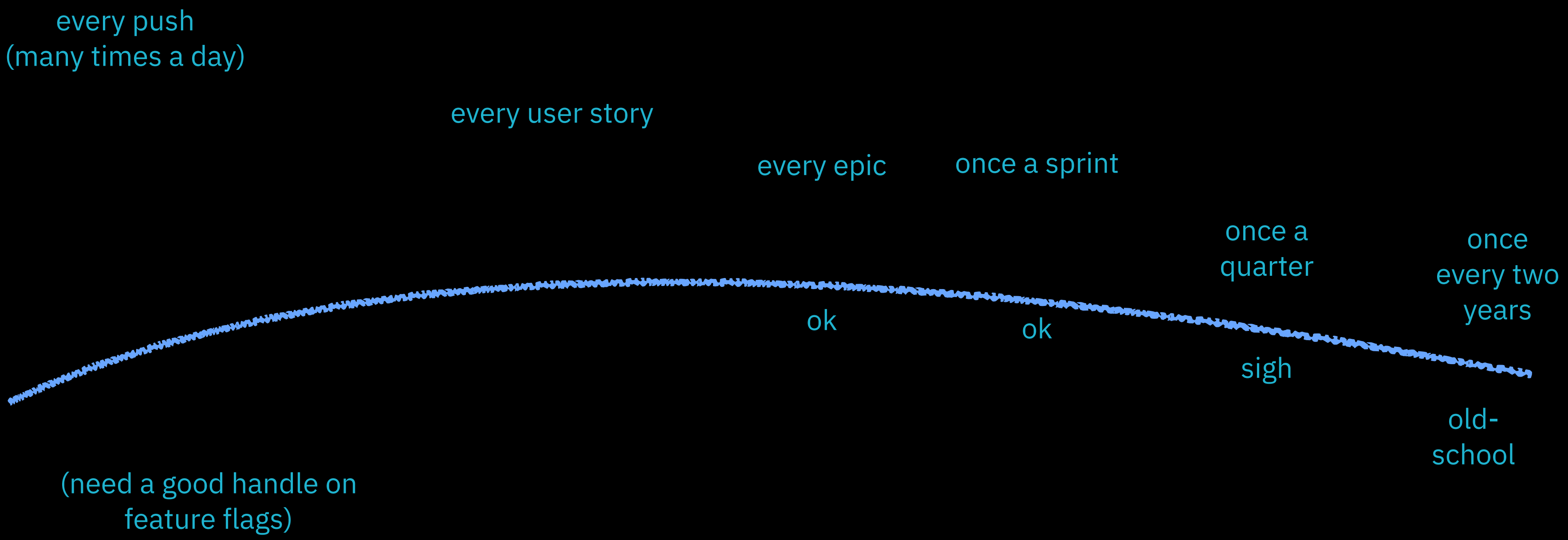
ok

sigh

old-
school

(need a good handle on
feature flags)

how often should you deploy?



every push
(many times a day)

every user story

every epic

once a sprint

once a
quarter

once
every two
years

ok

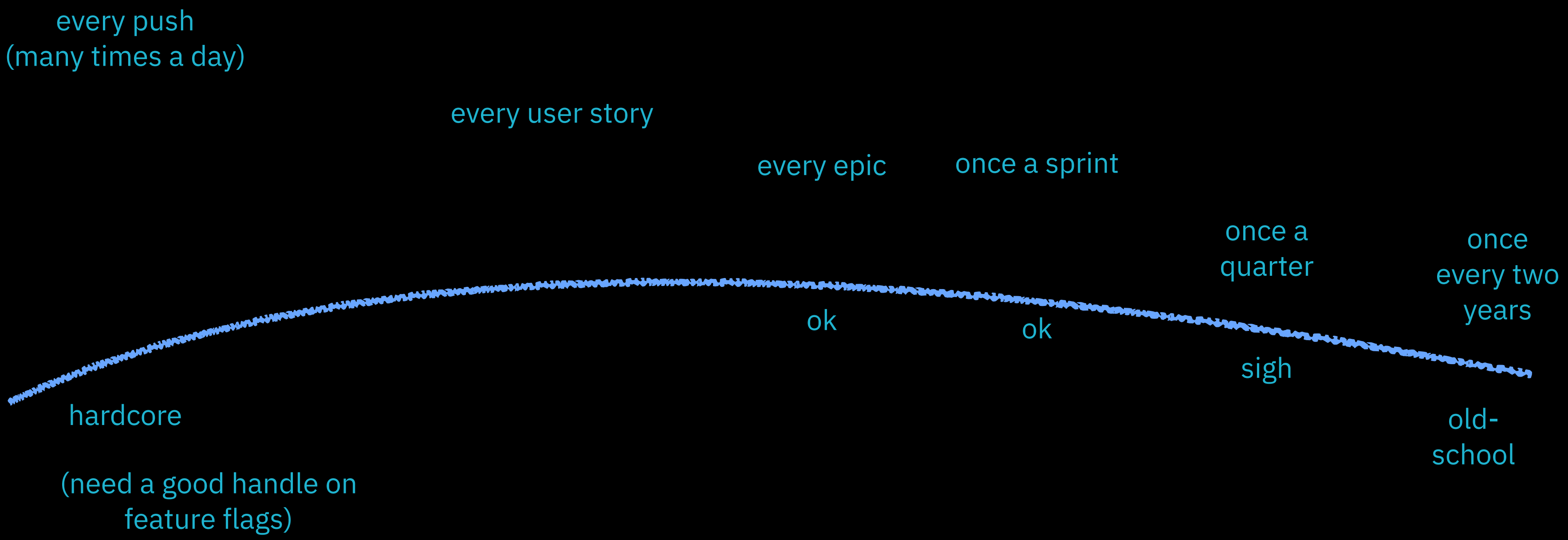
ok

sigh

old-
school

(need a good handle on
feature flags)

how often should you deploy?



every push
(many times a day)

hardcore

(need a good handle on
feature flags)

every user story

every epic

ok

once a sprint

ok

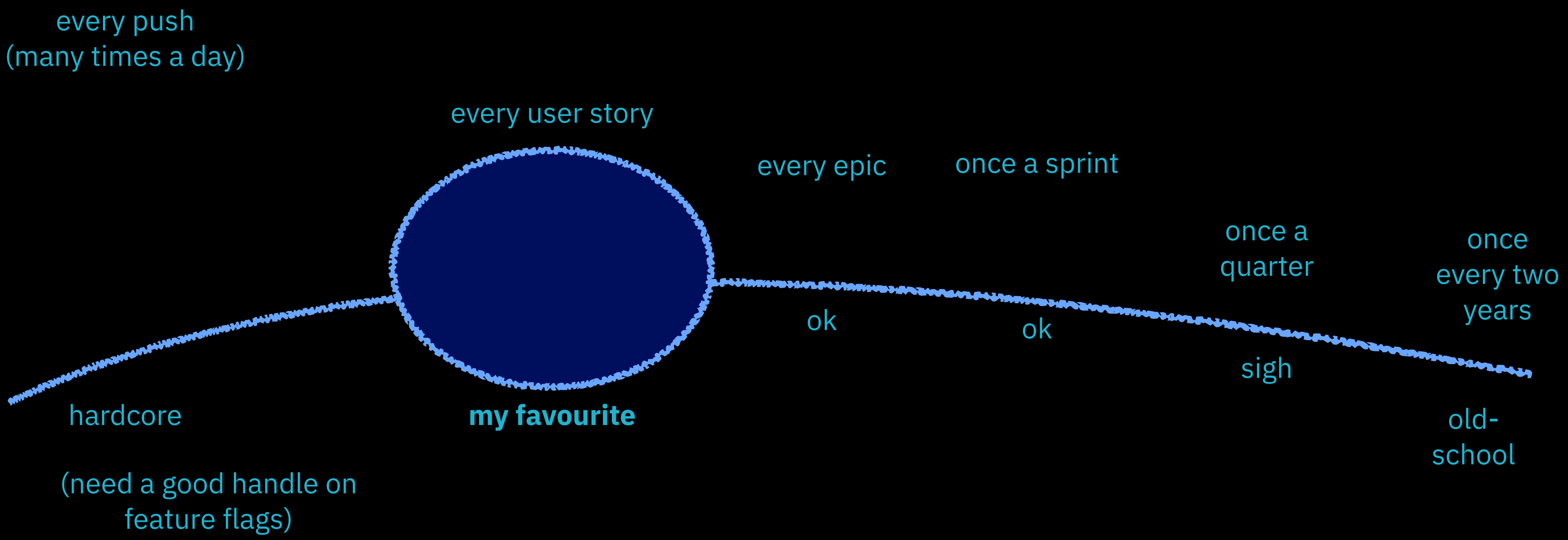
once a
quarter

sigh

once
every two
years

old-
school

how often should you deploy?

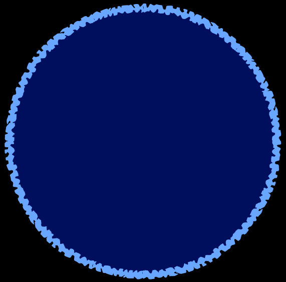


how often should you test in staging?

how often should you deliver?

how often should you deliver?

every push



my favourite

“we can’t actually **release** this.”

why?

what's stopping more frequent deploys?

“we can’t release this microservice…

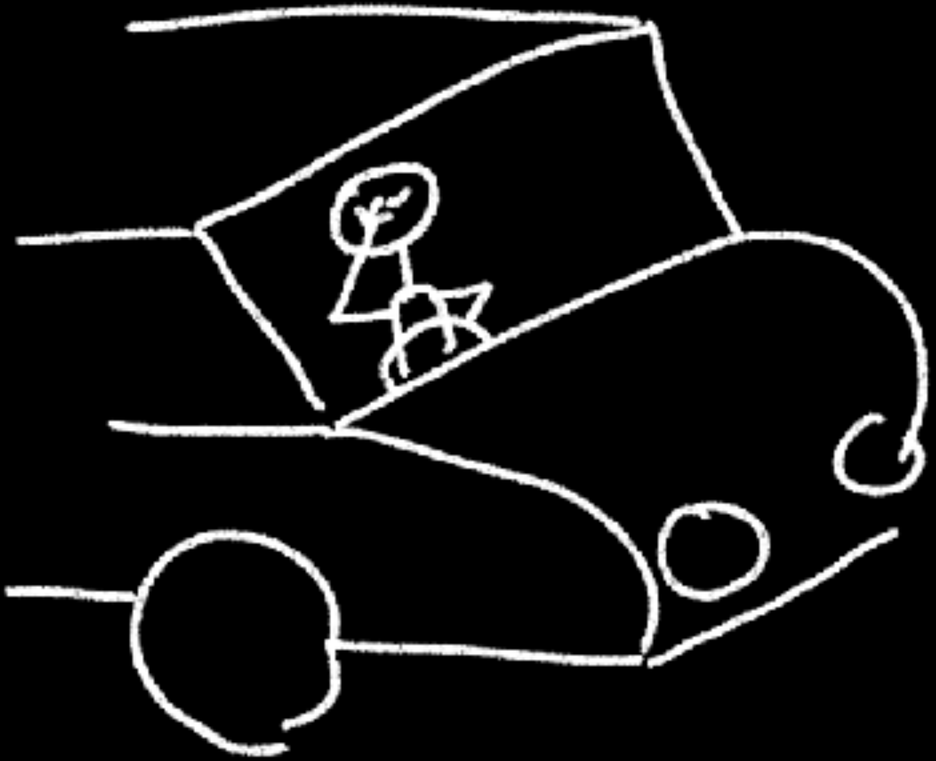
we deploy all our microservices at the same time.”

speed

speed

what’s the point of architecture that can go faster, if you don’t go faster?

what’s the point of architecture that can go faster, if you don’t go faster?



drive a car

feedback is good engineering

feedback is good business

deferred wiring

feature flags

A/B testing

canary deploys

make releases

deeply **boring**

by doing them often

@holly_cummins

make recovery

from failures

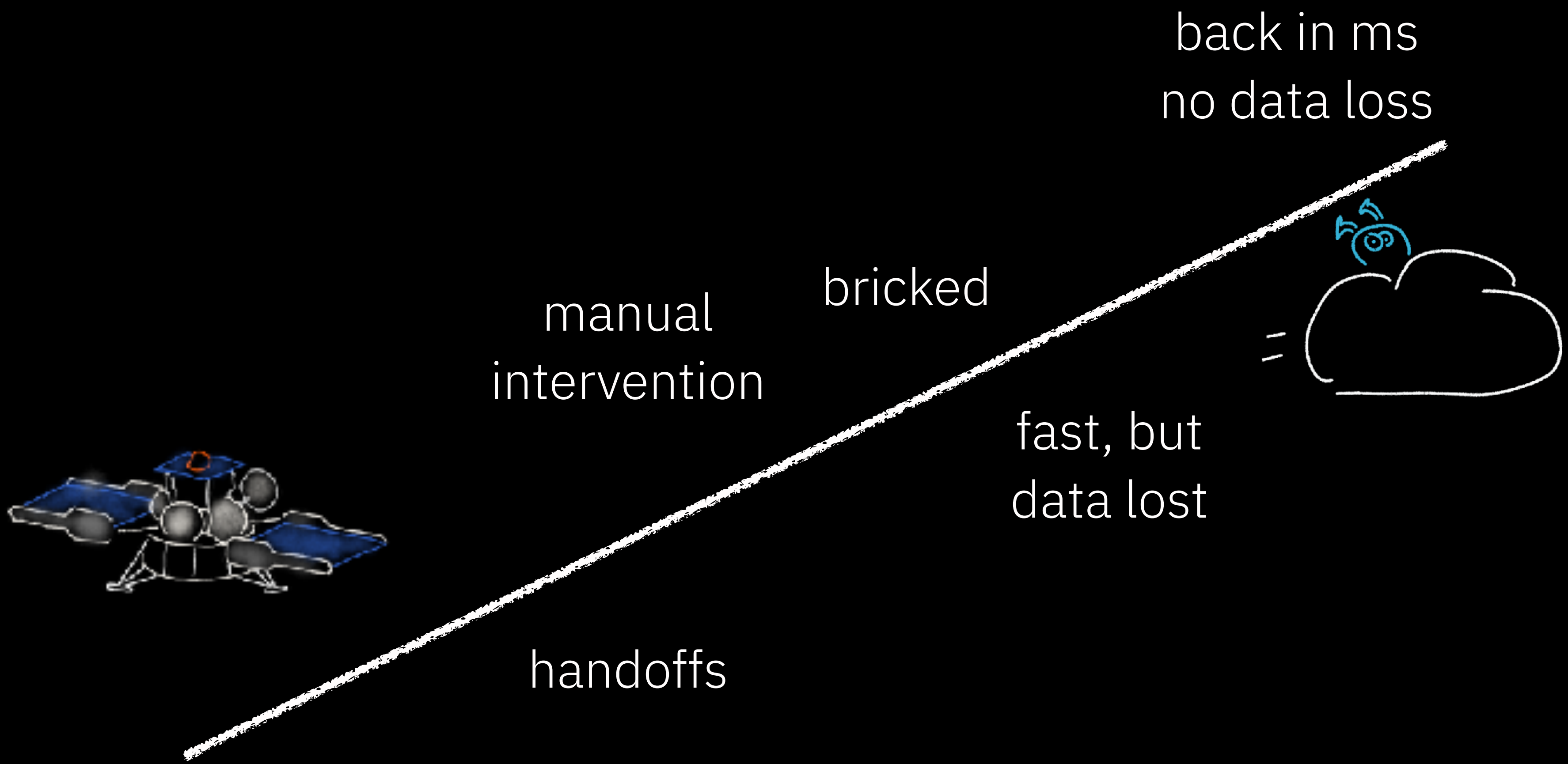
deeply **boring**

make recovery

from failures

deeply **boring**

by doing it often



back in ms
no data loss

bricked

manual
intervention

fast, but
data lost

handoffs

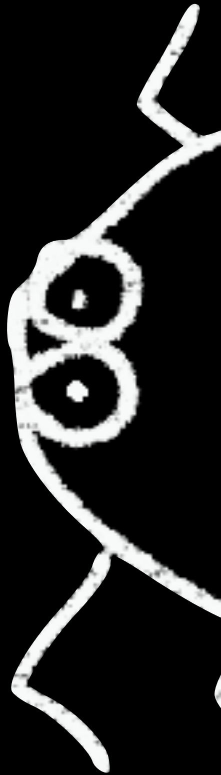
handoffs bad
automation good

automate everything

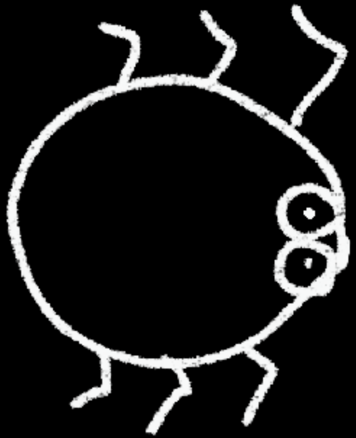
“our tests aren’t automated”

“we don’t know if our code works”

“we don’t know if our code works”



“we can’t ship until we have more confidence in the quality”



“we can’t ship until we have more confidence in the quality”

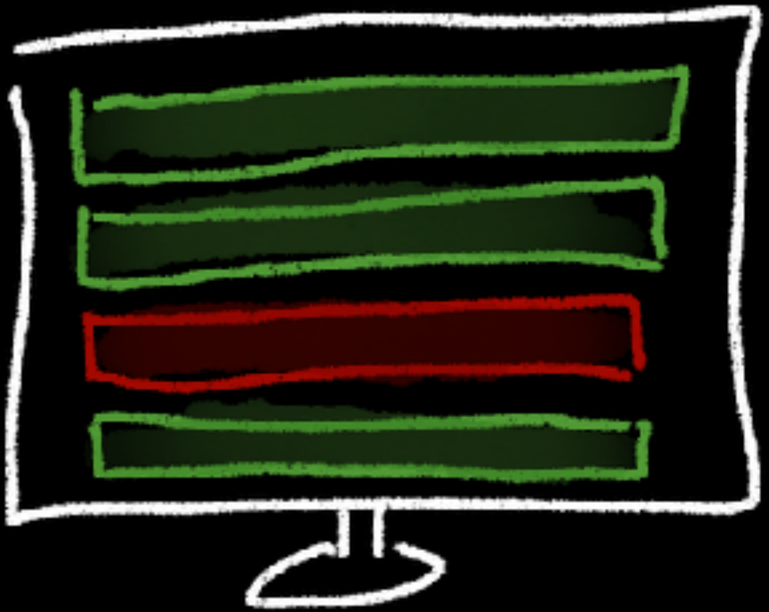
you can **fix** that

microservices **need** automated integration tests

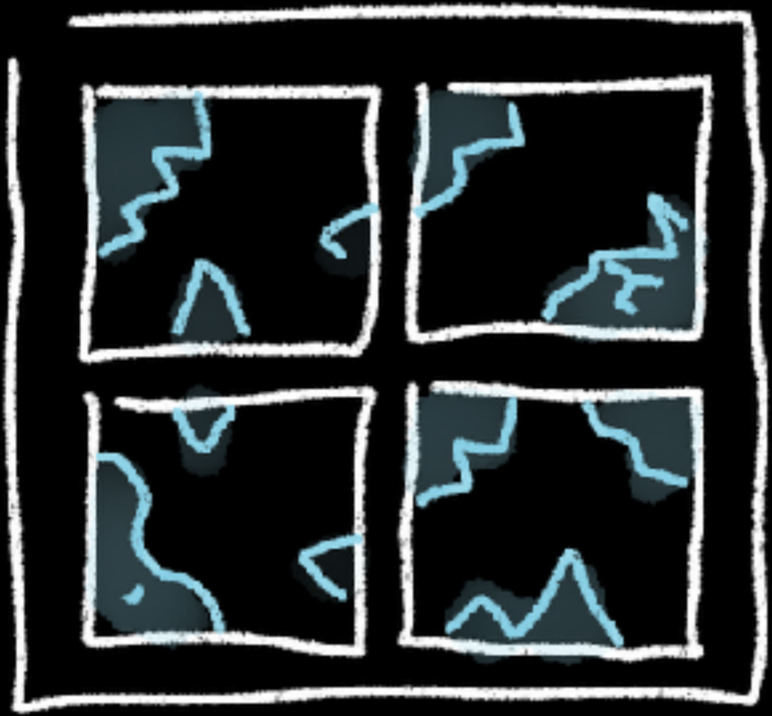
not a good CI/CD indicator

a good CI/CD indicator

“we don’t know when the build is broken”

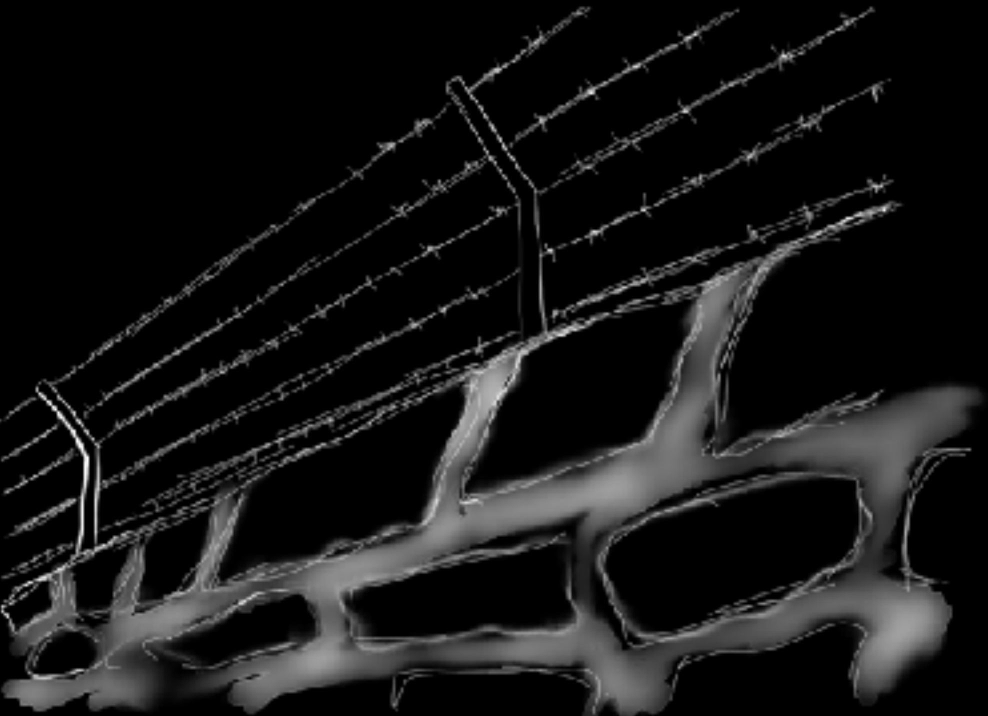


a good build radiator



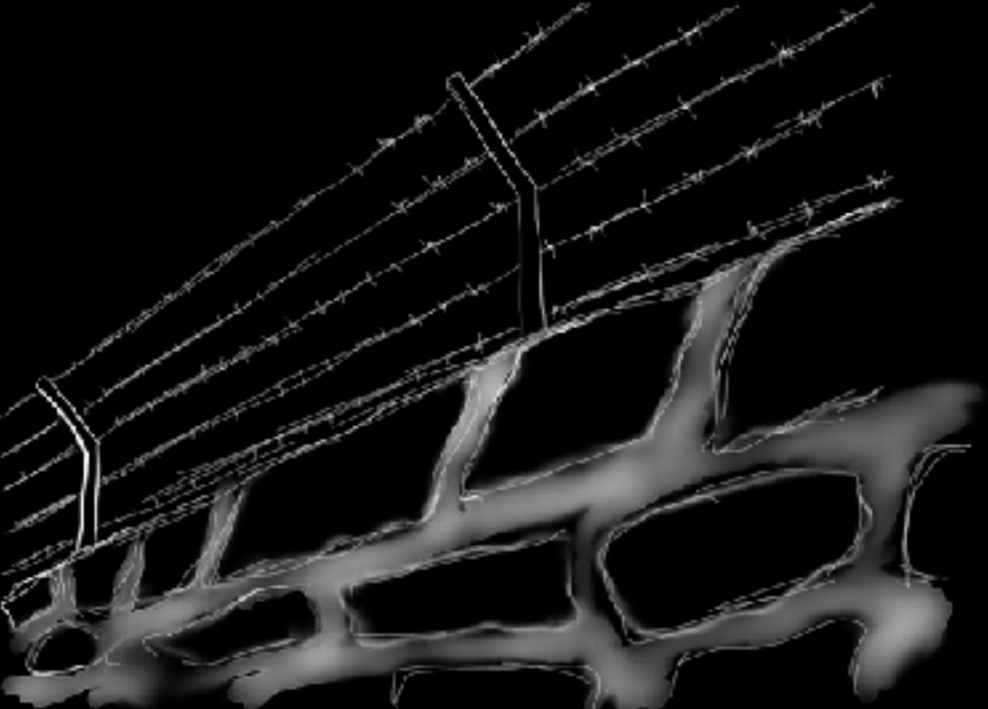
“oh yes, that build has been broken for a few weeks…”

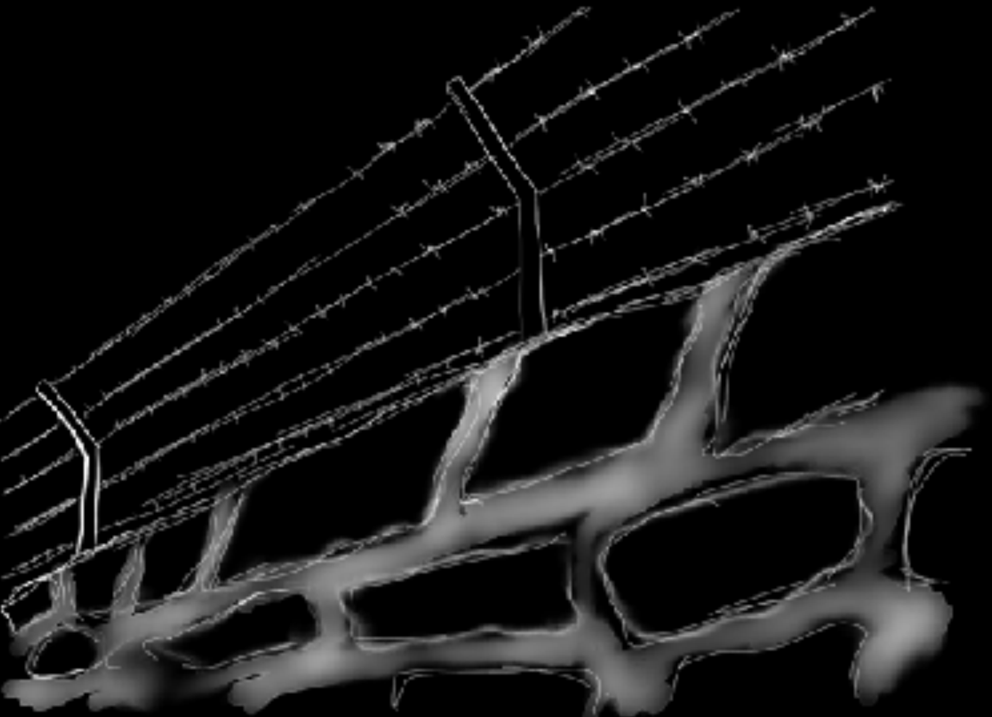
the locked-
down totally
rigid inflexible
un-cloudy cloud



@holly_cummins

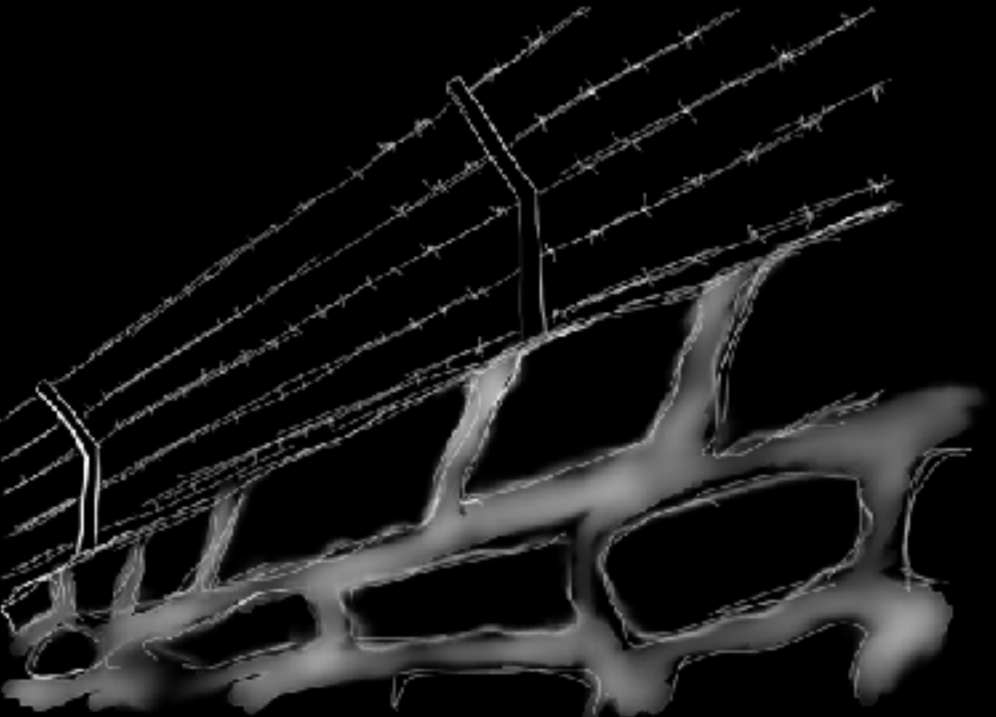
“we’ve configured our network!





“we’ve configured our network!

you can either access the cloud servers … or access jira.



“we’ve configured our network!

you can either access the cloud servers … or access jira.

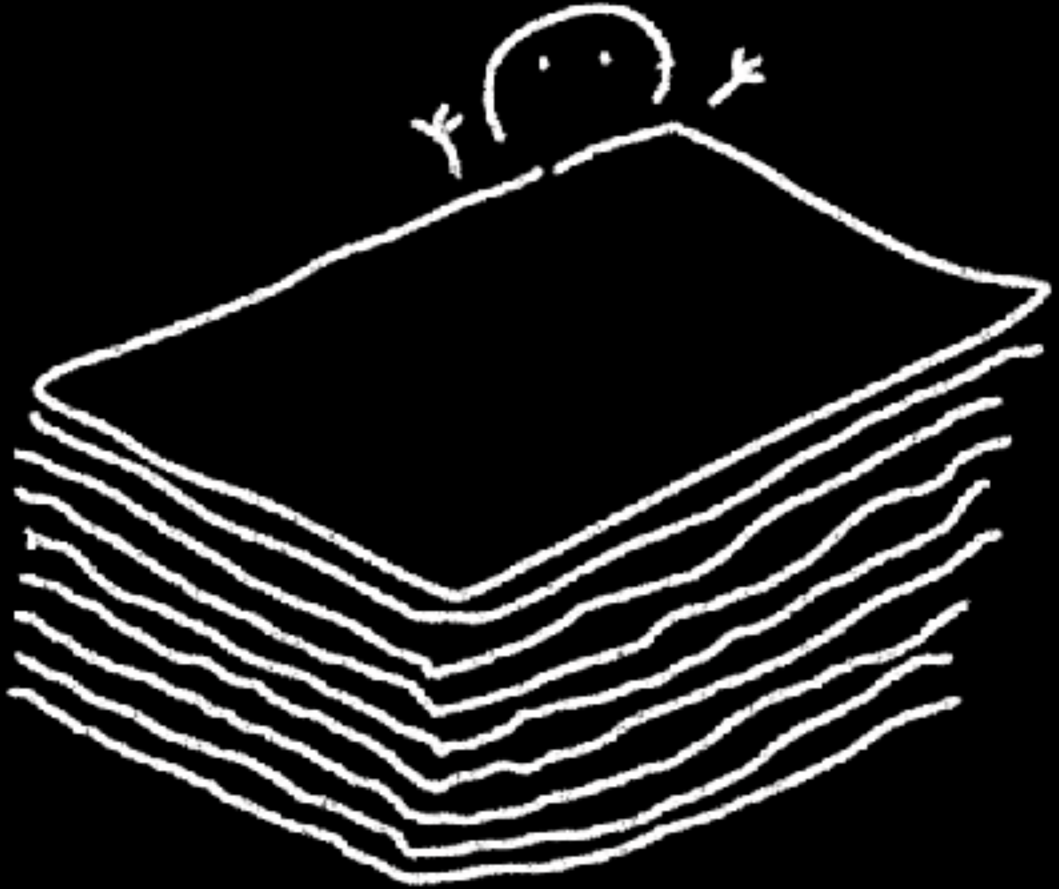
to access both you’d need two machines.”

@holly_cummins

“it takes us a week to start coding.”

“it takes us a week to start coding.”

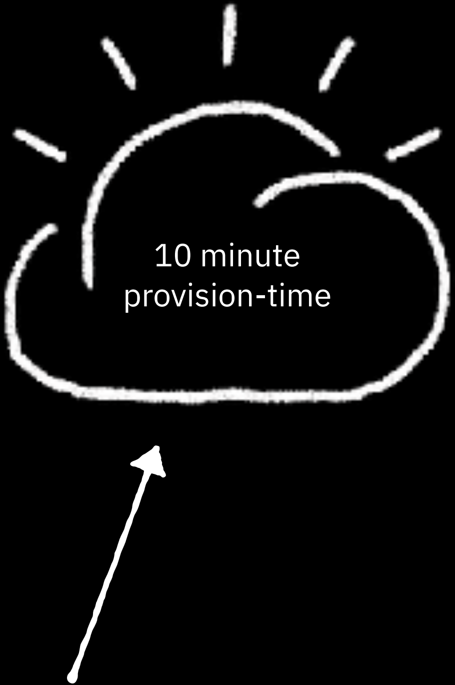
“two days to get a repo ...
two days to get a pipeline ...”



“we’ve scheduled the architecture board review for a month after the project is ready to ship”

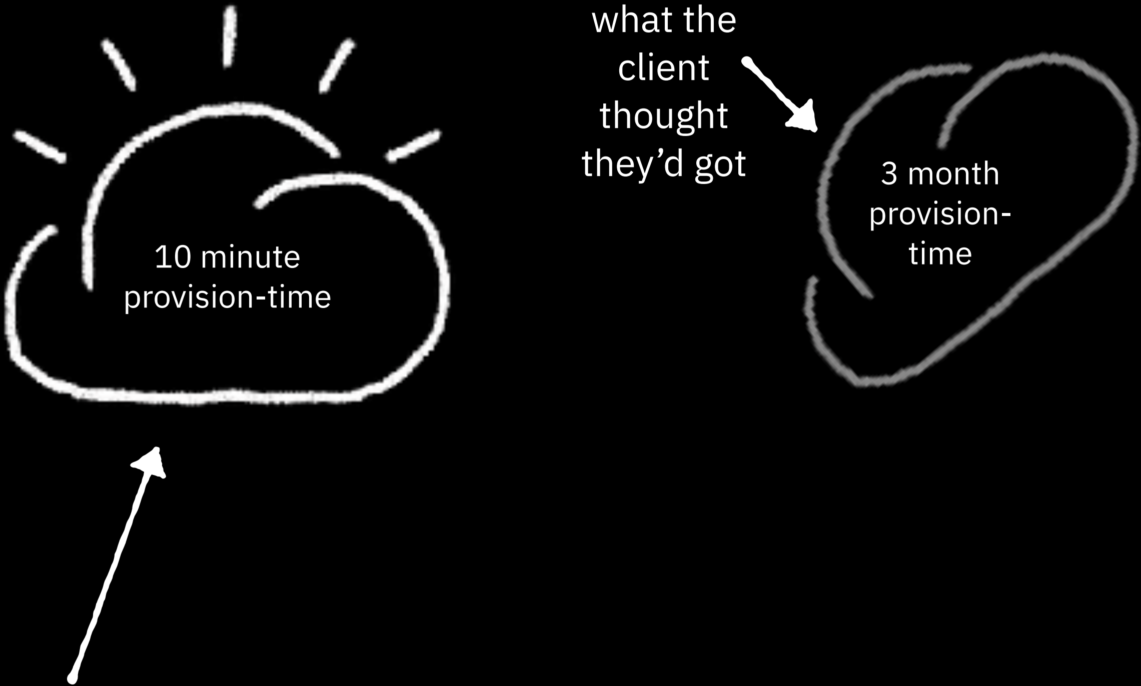
@holly_cummins

“this provisioning software is broken”



what we sold

“this provisioning software is broken”



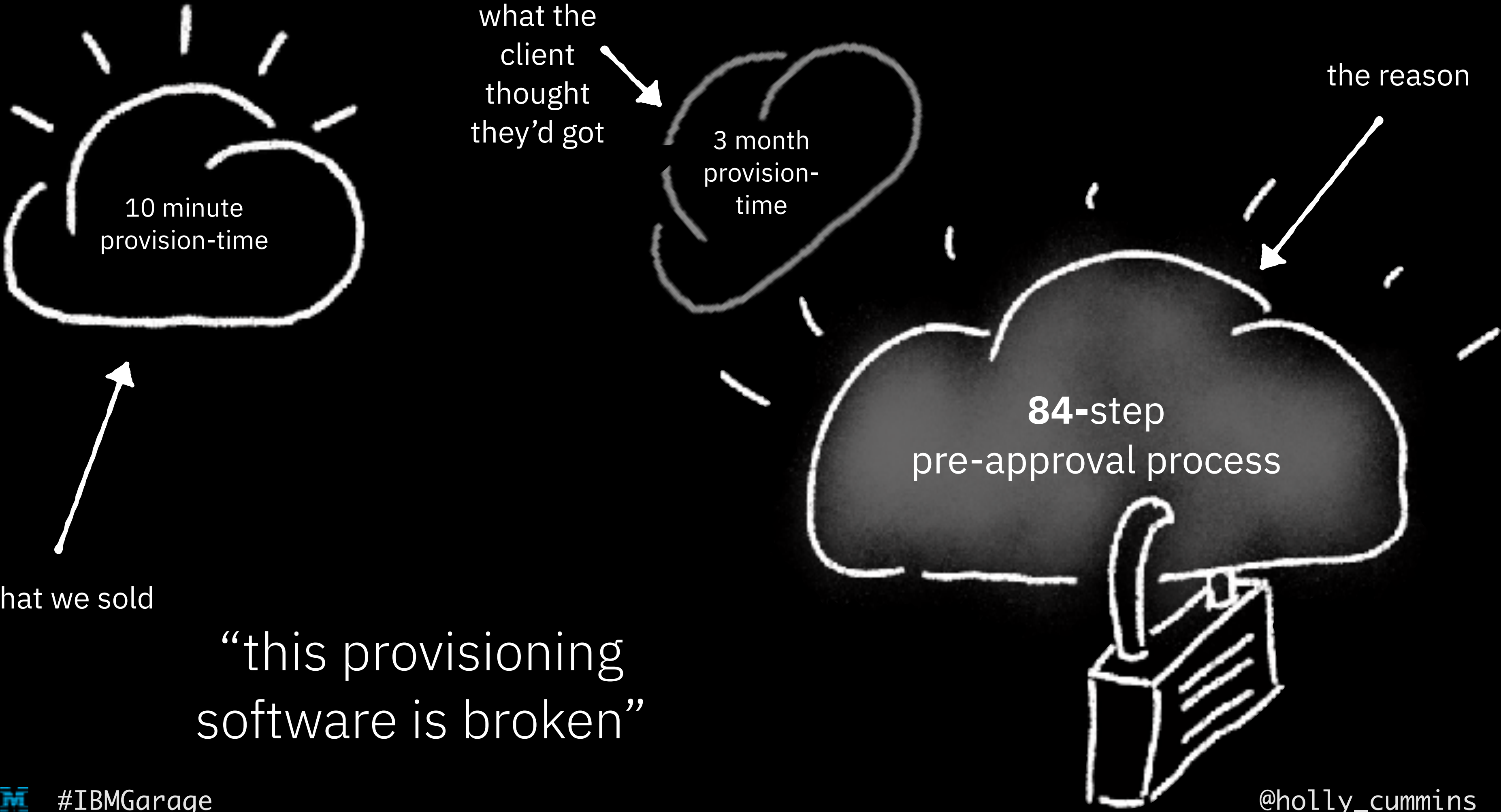
10 minute provision-time

what the client thought they'd got

3 month provision-time

what we sold

“this provisioning software is broken”



10 minute provision-time

what we sold

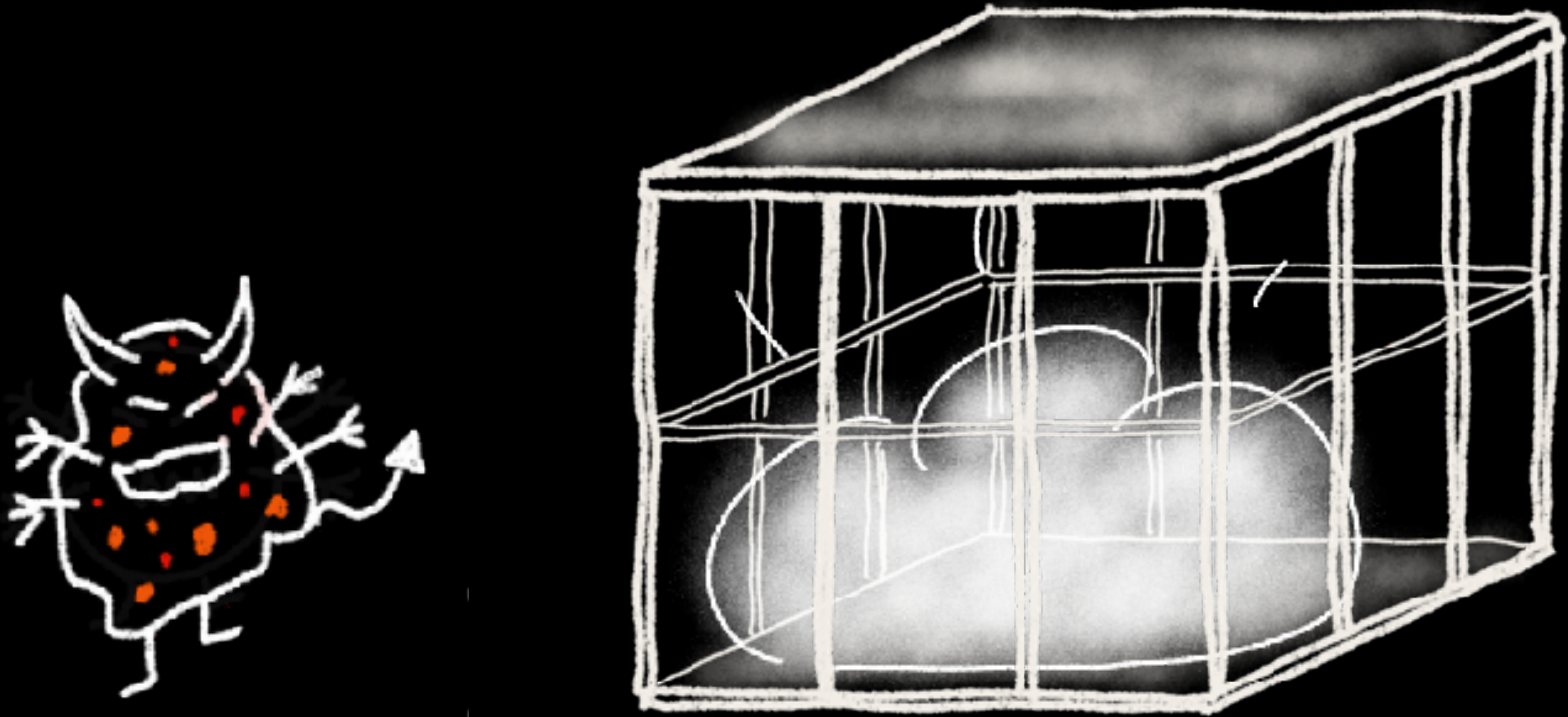
what the client thought they'd got

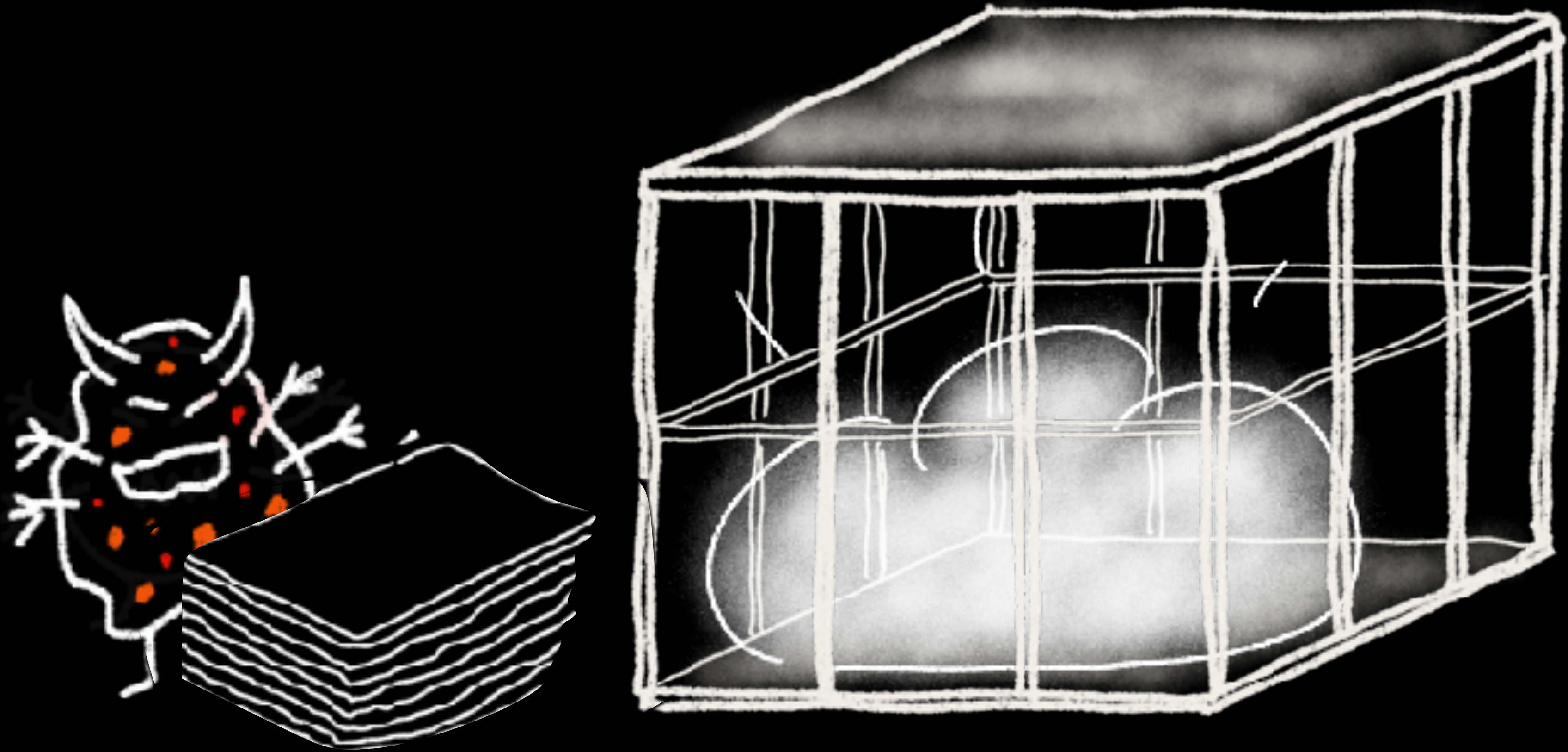
3 month provision-time

the reason

84-step pre-approval process

"this provisioning software is broken"





old-style governance isn’t going to work

Provider A

Provider A

Provider B

“we’re going to change cloud provider to fix our procurement process!”

Provider A

Provider B

“we’re going to change cloud provider to fix our procurement process!”

if the developers are the only ones changing, cloud native is not going to work there is a **cost**:

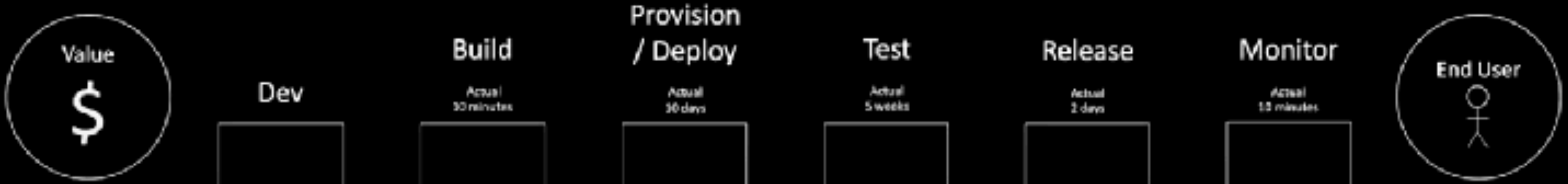
developers leave

if you automate something, change the processes around that assume that the previously manual process is expensive or error prone.

Delivering software better

The objective is to…

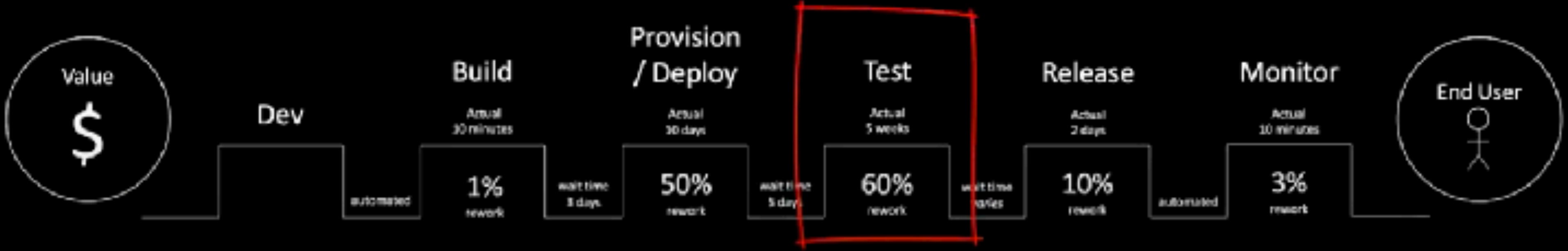
Optimize the System as a Whole



Value → User

Delivering software better

Localized optimization will not deliver desired outcomes!



good cloud
native culture

be clear on what you're trying to achieve

optimise for feedback

psychological safety

collaboration

co-creation feels awesome

@holly_cummins