FT FINANCIAL TIMES

Practical Cloud Native: what works, what doesn't

Sarah Wells
Technical Director, Operations & Reliability
@sarahjwells

FT FINANCIAL TIMES

Practical Cloud Native: what works, what doesn't

Sarah Wells
Technical Director, Operations & Reliability
@sarahjwells


FINANCIAL TIMES
smart arts
how to spend it
FT Weekend
Asia edition
Summer of sport
Chelsea's new designers
Flower show special
FINANCIAL TIMES
Feet of clay
Heated exchange
Fast moving
EU and Japan defy wave of protectionism
Blankfein announces departure as Goldman unveils profit surge
FINANCIAL TIMES
Trump backs down on asking citizenship status on US census
FINANCIAL TIMES
Trump blasts US-Japan defence alliance, EU ahead of G20

Cloud Native means building systems that benefit from the cloud rather than just running on it

FT

Moving to the cloud

A ‘lift and shift’ means moving things ‘as they are’

FT

Moving to the cloud

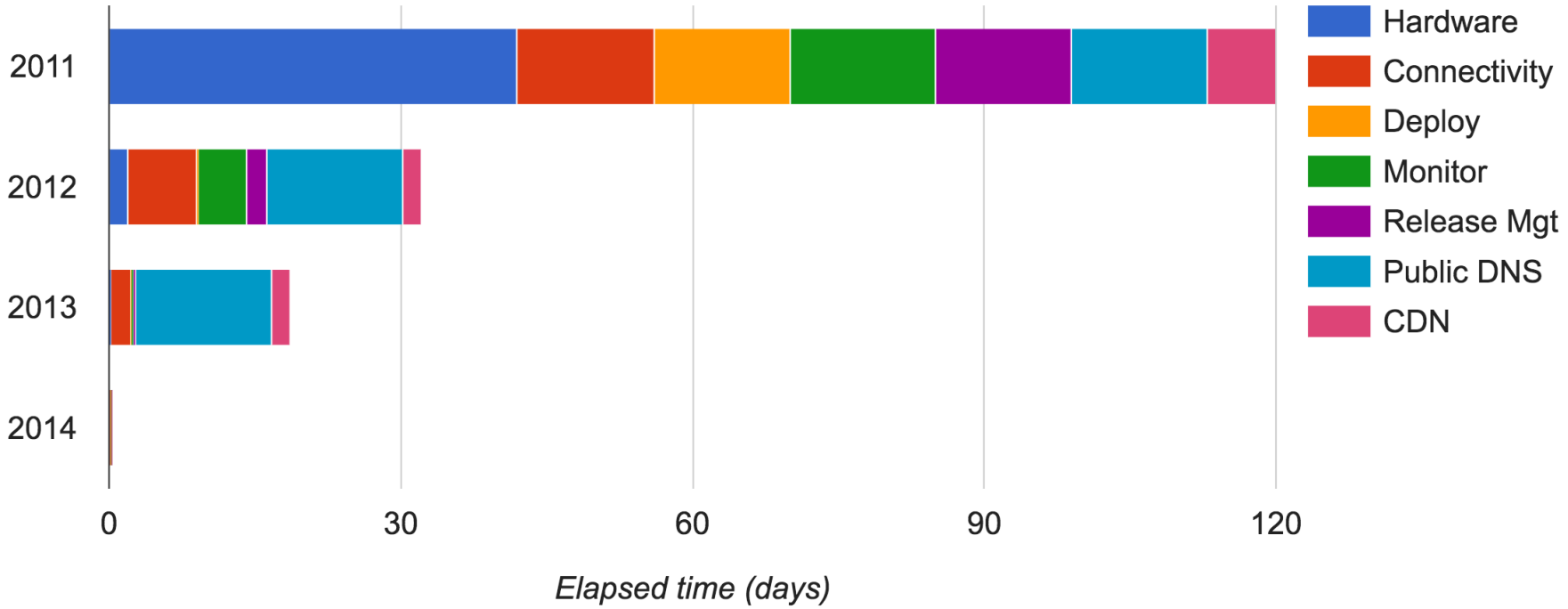
A ‘lift and shift’ gives you

- Someone else doing the infrastructure work

Moving to the cloud

A ‘lift and shift’ gives you

- Someone else doing the infrastructure work
- Quicker provisioning


2011
2012
2013
2014
0
30
60
90
120
Elapsed time (days)
Hardware
Connectivity
Deploy
Monitor
Release Mgt
Public DNS
CDN

FT

Moving to the cloud

A ‘lift and shift’ gives you

- Someone else doing the infrastructure work
- Quicker provisioning
- Easy scaling

Moving to the cloud

A ‘lift and shift’ gives you

- Someone else doing the infrastructure work
- Quicker provisioning
- Easy scaling
- More accurate cost attribution

FT

Going further gives you more

The cloud native approach

- Continuous delivery

FT

The cloud native approach

- Continuous delivery
- Microservices

The cloud native approach

- Continuous delivery
- Microservices
- Containers and orchestration

The cloud native approach

- Continuous delivery
- Microservices
- Containers and orchestration

The cloud native approach

- Continuous delivery
- Microservices
- Containers and orchestration
- X-as-a-service (PaaS, DaaS, SaaS)

FT

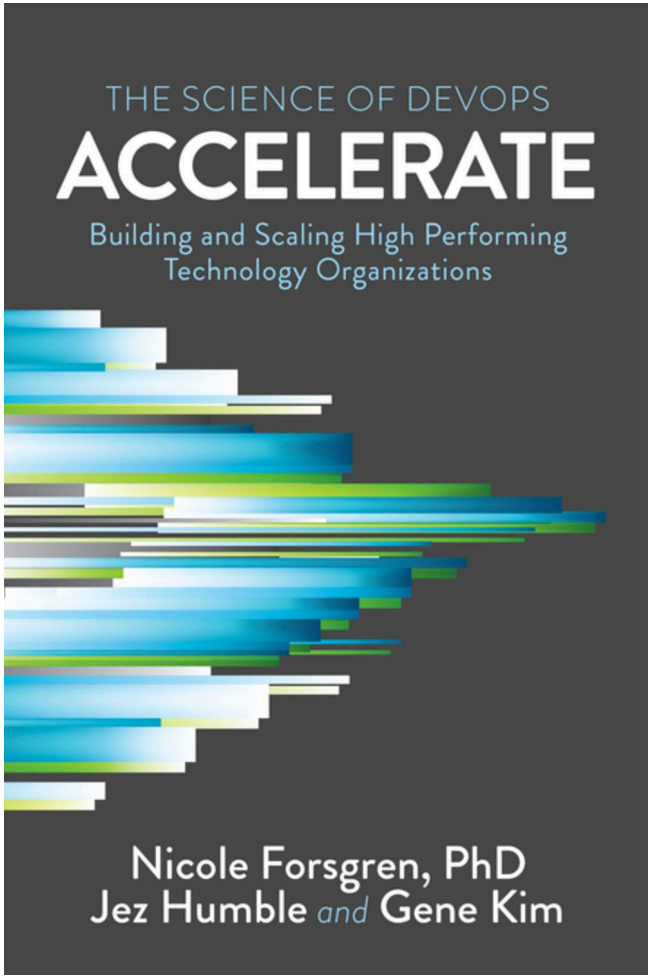
The cloud native approach

- Continuous delivery
- Microservices
- Containers and orchestration
- X-as-a-service (PaaS, DaaS, SaaS)

FT

Why cloud native?

FT



Key measures:

Delivery lead time

Deployment frequency

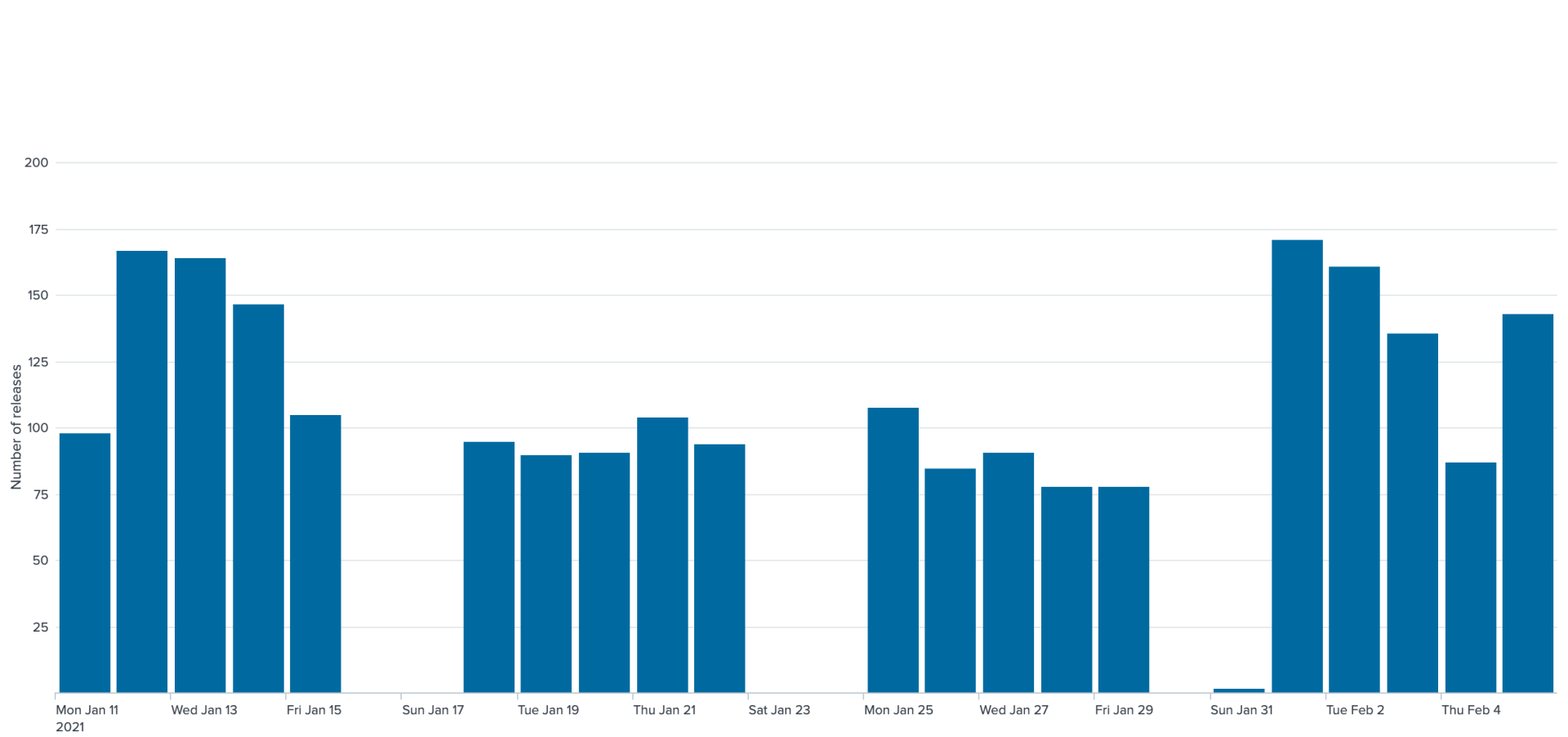
Time to restore service

Change failure rate

FT

Cloud native done properly will improve all these metrics

FT


200
175
150
125
100
75
50
25
Number of releases
Mon Jan 11
2021
Wed Jan 13
Fri Jan 15
Sun Jan 17
Tue Jan 19
Thu Jan 21
Sat Jan 23
Mon Jan 25
Wed Jan 27
Fri Jan 29
Sun Jan 31
Tue Feb 2
Thu Feb 4

But doing cloud native properly means changing a lot of things

FT

1. Optimise for loose coupling

FT

2. Empower your teams

FT

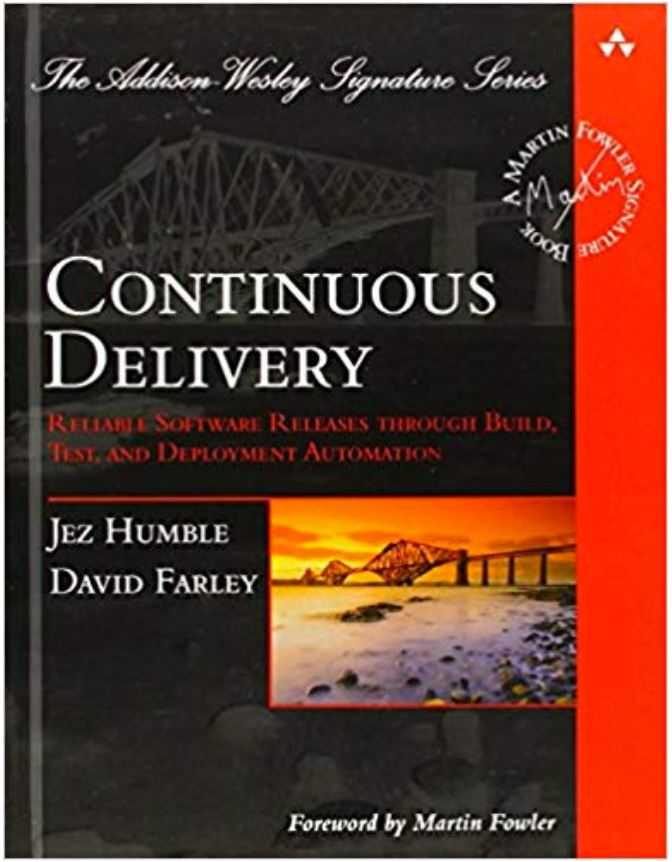
3. Understand that distributed systems have different characteristics

What changes

- Deployment
- Architecture
- How you build things
- How you test things
- How you support things
- What your organisation looks like

FT

1. Deployment



Continuous delivery requires continuous integration

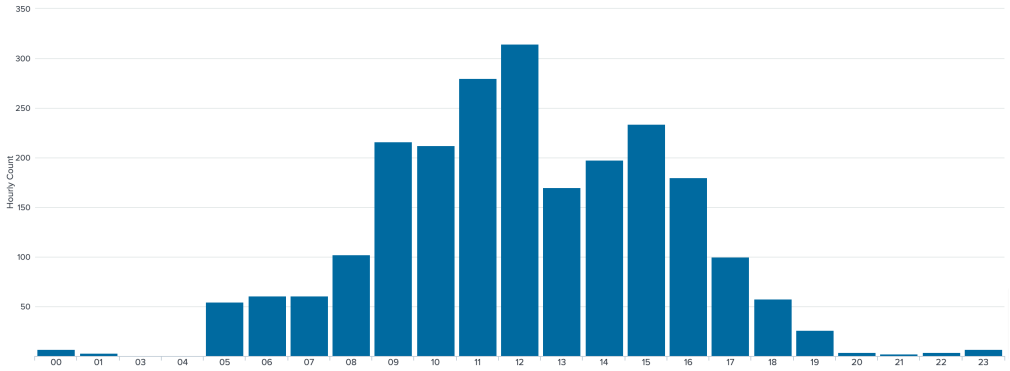
FT

No change advisory boards, no change request forms to fill in

FT

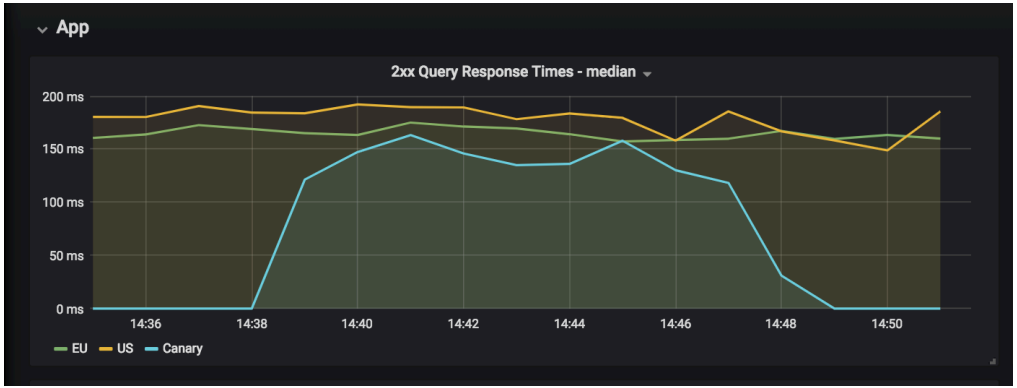
You don’t have to choose speed or stability - you get both!

FT



Zero downtime deployments

FT



Separate releasing code from releasing functionality

FT

2. Architecture

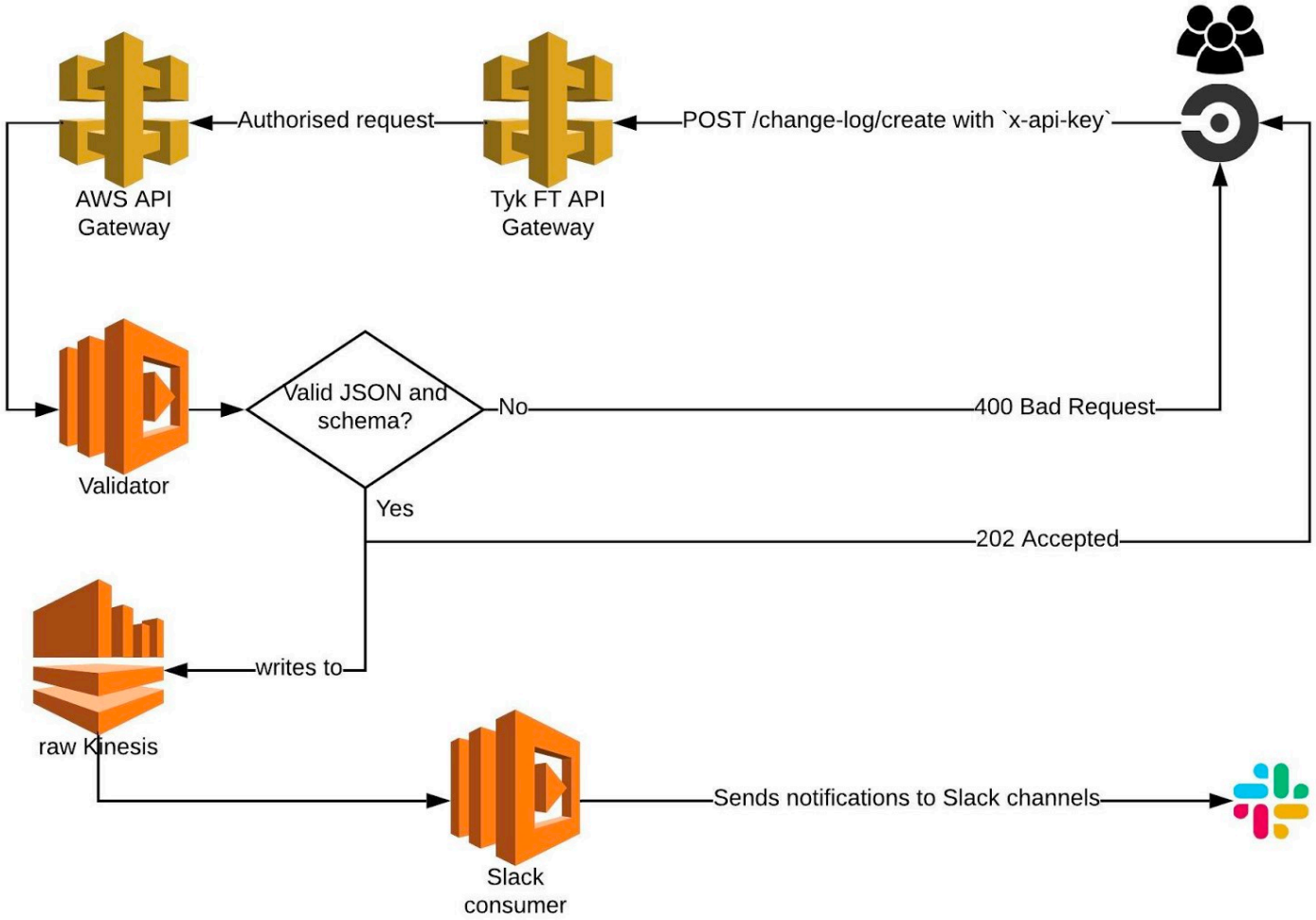
12 factor architectures

https://12factor.net

FT

Use queues where you can

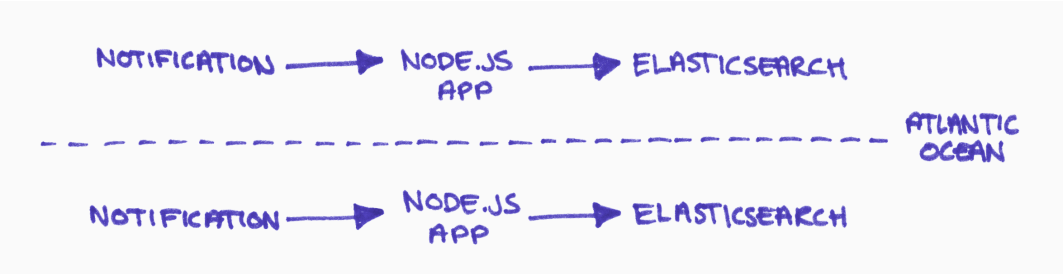
FT


AWS API Gateway
Authorised request
Tyk FT API Gateway
POST /change-log/create with `x-api-key`
Validator
Valid JSON and schema?
No
400 Bad Request
Yes
202 Accepted
writes to
raw Kinesis
Slack consumer
Sends notifications to Slack channels

Transactions can’t save you

Work towards eventual consistency

FT



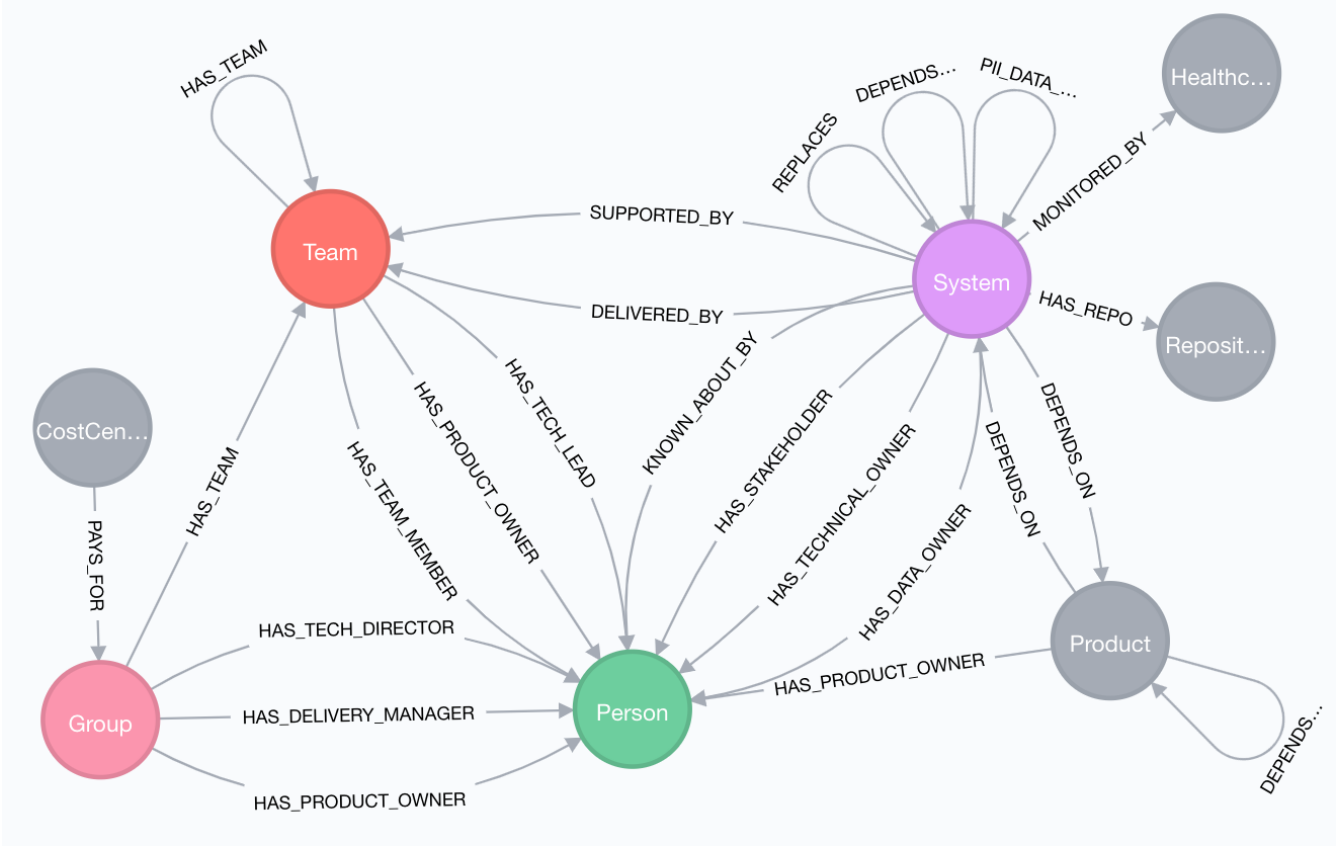
Our ft.com search instances do everything independently

https://www.matthinchliffe.dev/2021/01/27/under-engineering-is-just-fine.html

FT

Keep it simple

FT


HAS_TEAM
Team
SUPPORTED_BY
DELIVERED_BY
REPLACES
DEPENDS...
PII_DATA_...
System
MONITORED_BY
Healthc...
HAS_REPO
Reposit...
CostCen...
PAYS_FOR
HAS_TEAM
HAS_TEAM_MEMBER
HAS_PRODUCT_OWNER
HAS_TECH_LEAD
KNOWN_ABOUT_BY
HAS_STAKEHOLDER
HAS_TECHNICAL_OWNER
HAS_DATA_OWNER
DEPENDS_ON
DEPENDS_ON
Product
DEPENDS...
Group
HAS_TECH_DIRECTOR
HAS_DELIVERY_MANAGER
HAS_PRODUCT_OWNER
Person
HAS_PRODUCT_OWNER

3. How you build things

HOW COMPLEX SYSTEMS FAIL

When complex systems fail, what does this tell us about everyday work? **Richard I. Cook** explains this and more, in this classic treatise on the nature of failure, how failure is evaluated, how failure is attributed to proximate cause, and the resulting new understanding of safety.

1) Complex systems are intrinsically hazardous systems.

All of the interesting systems (e.g., transportation, healthcare, power generation) are inherently and unavoidably hazardous by the own nature. The frequency of hazard exposure can sometimes be changed but the processes involved in the system are themselves intrinsically and irreducibly hazardous. It is the presence of these hazards that drives the creation of defenses against hazard that characterize these systems.

2) Complex systems are heavily and successfully defended against failure.

The high consequences of failure lead over time to the construction of multiple layers of defense against failure. These defenses include obvious technical components (e.g. backup systems, 'safety' features of equipment) and human components (e.g., training, knowledge) but also a variety of organizational, institutional, and regulatory defenses (e.g., policies and procedures, certification, work rules, team training). The effect of these measures is to provide a series of shields that normally divert operations away from accidents.

3) Catastrophe requires multiple failures – single point failures

small, apparently innocuous failures join to create opportunity for a systemic accident. Each of these small failures is necessary to cause catastrophe but only the combination is sufficient to permit failure. Put another way, there are many more failure opportunities than overt system accidents. Most initial failure trajectories are blocked by designed system safety components. Trajectories that reach the operational level are mostly blocked, usually by practitioners.

4) Complex systems contain changing mixtures of failures latent within them.

The complexity of these systems makes it impossible for them to run without multiple flaws being present. Because these are individually insufficient to cause failure they are regarded as minor factors during operations. Eradication of all latent failures is limited primarily by economic cost but also because it is difficult before the fact to see how such failures might contribute to an accident. The failures change constantly because of changing technology, work organization, and efforts to eradicate failures.

5) Complex systems run in degraded mode.

that complex systems run as broken systems. The system continues to function because it contains so many

accidents' that nearly generated catastrophe. Arguments that these degraded conditions should have been recognized before the overt accident are usually predicated on naïve notions of system performance. System operations are dynamic, with components (organizational, human, technical) failing and being replaced continuously.

6) Catastrophe is always just around the corner.

Complex systems possess potential for catastrophic failure. Human practitioners are nearly always in close physical and temporal proximity to these potential failures – disaster can occur at any time and in nearly any place. The potential for catastrophic outcome is a hallmark of complex systems. It is impossible to eliminate the potential for such catastrophic failure; the potential for such failure is always present by the system's own nature.

7) Post-accident attribution accident to a 'root cause' is fundamentally wrong.

Because overt failure requires multiple faults, there is no isolated 'cause' of an accident. There are multiple contributors to accidents. Each of these is necessary insufficient in itself to create an accident. Only jointly are these causes sufficient to create an accident. Indeed, it is the linking of

Distributed systems are generally in a state of 'grey failure'

Richard I Cook

https://www.skybrary.aero/bookshelf/books/5926.pdf

FT

Focus on resilience and redundancy

FT

Think about what is acceptable latency

FT

Build observability into your system

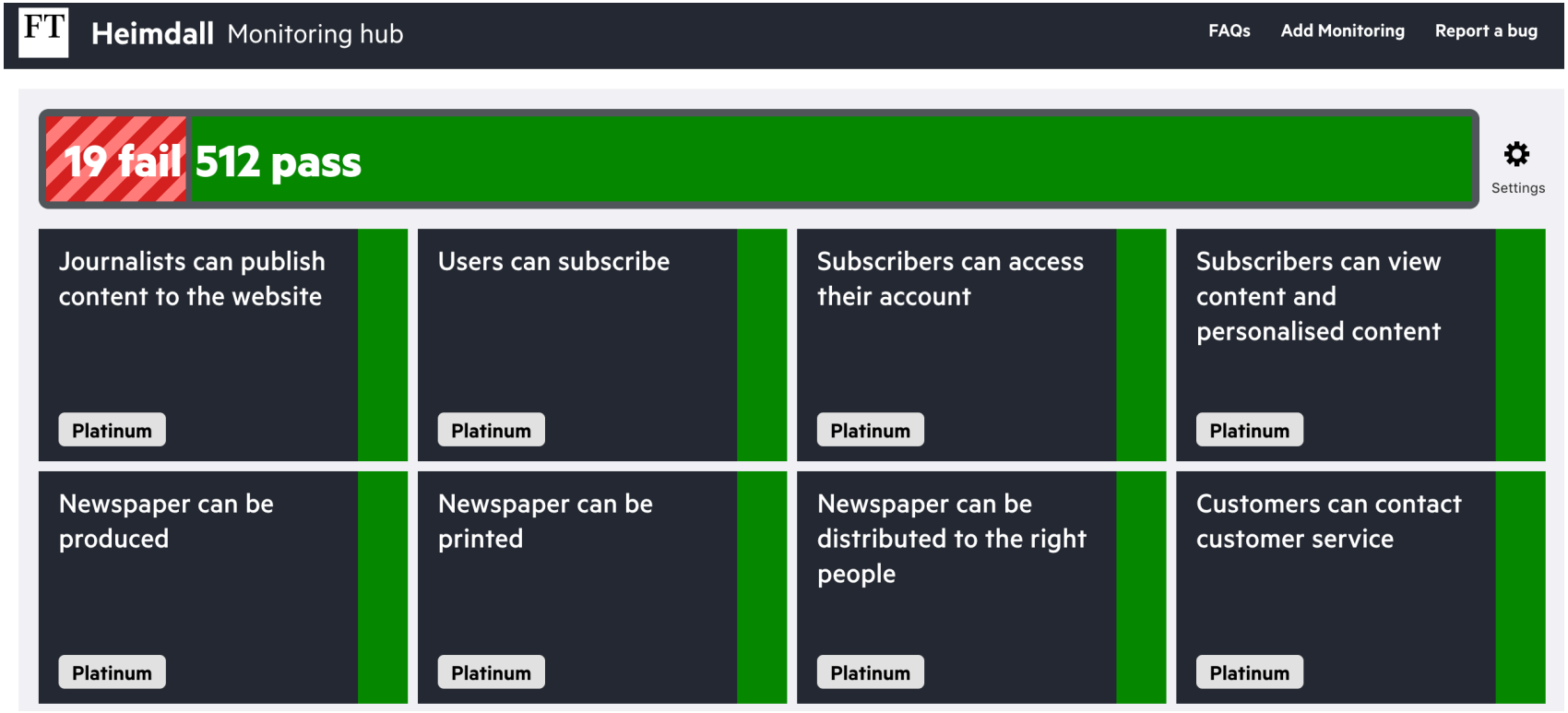
FT

Log aggregation and transaction ids

FT

Make sure you know when something is REALLY wrong

FT


FT
Heimdall Monitoring hub
FAQs
Add Monitoring
Report a bug
19 fail
512 pass
Settings
Journalists can publish content to the website
Platinum
Users can subscribe
Platinum
Subscribers can access their account
Platinum
Subscribers can view content and personalised content
Platinum
Newspaper can be produced
Platinum
Newspaper can be printed
Platinum
Newspaper can be distributed to the right people
Platinum
Customers can contact customer service
Platinum

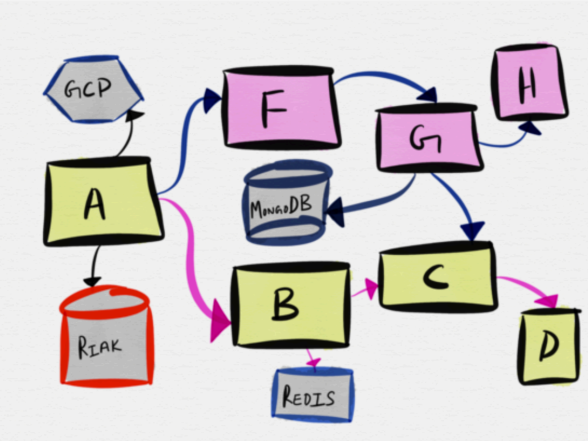
Have a plan for the inevitable migrations

4. How you test things

Every service needs automated tests

FT

Consider the following topology—a very plausible example of a microservice architecture.



Service A's interaction with service B involves service B talking to Redis and service C. However, the smallest *unit* to be tested really is ***service A's interaction with service B***, and the easiest way to test that interaction is by spinning up a fake for service B and testing service A's interaction with the

Test at the boundary of the service

Cindy Sridharan:
https://copyconstruct.medium.com/testing-microservices-the-sane-way-9bb31d158c16

FT

Acceptance tests running locally pushes developers towards a “full stack on your laptop”

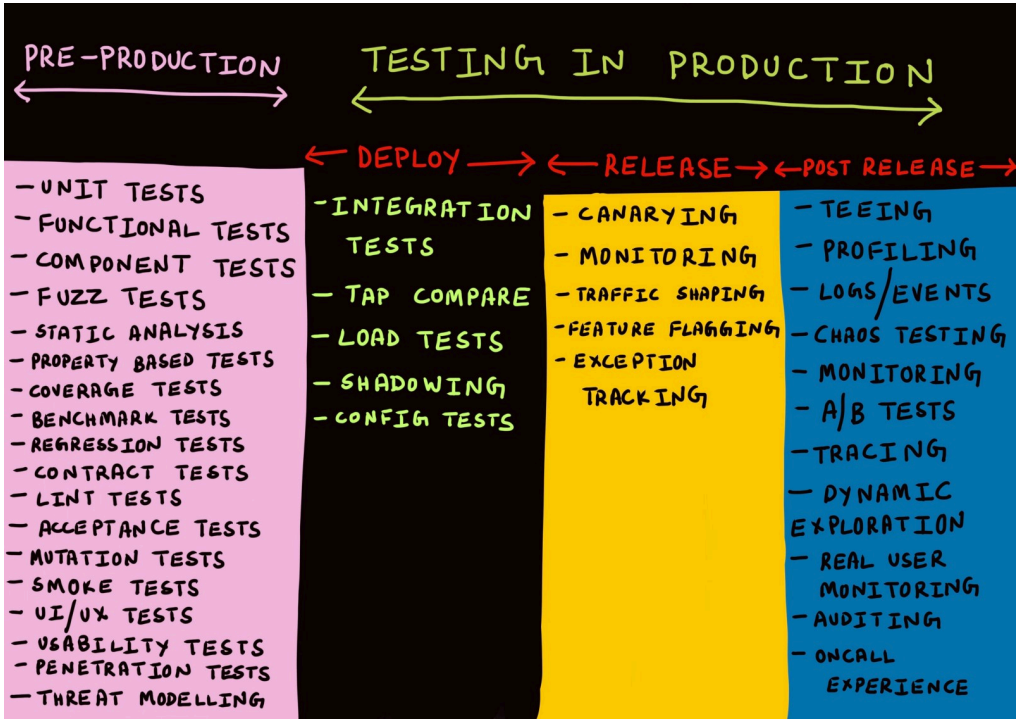
FT

You end up with a distributed monolith

FT

Test in production!

FT



Cindy Sridharan:
https://copyconstruct.medium.com/testing-in-production-the-safe-way-18ca102d0ef1

FT

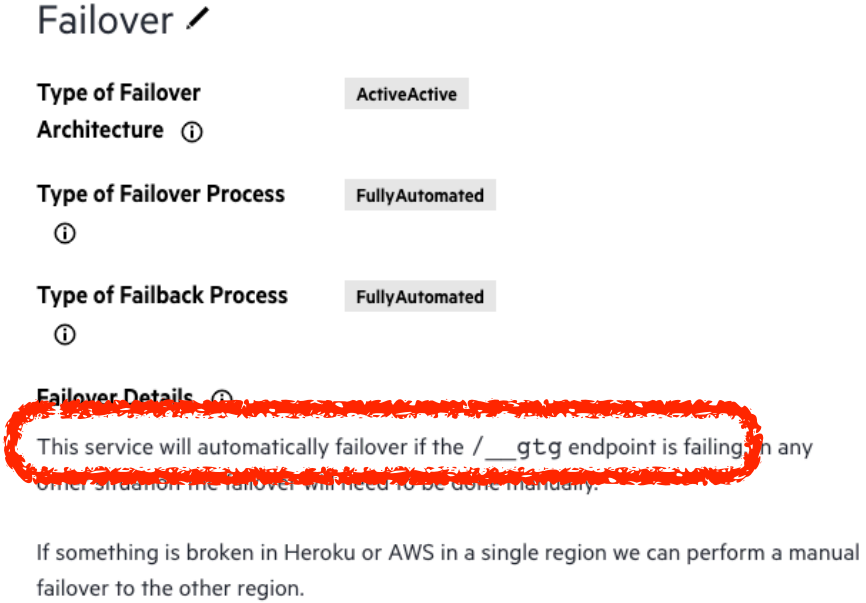
Test your infrastructure too - chaos engineering

FT

5. How you support things

The team that builds a system has to be the team that runs it too

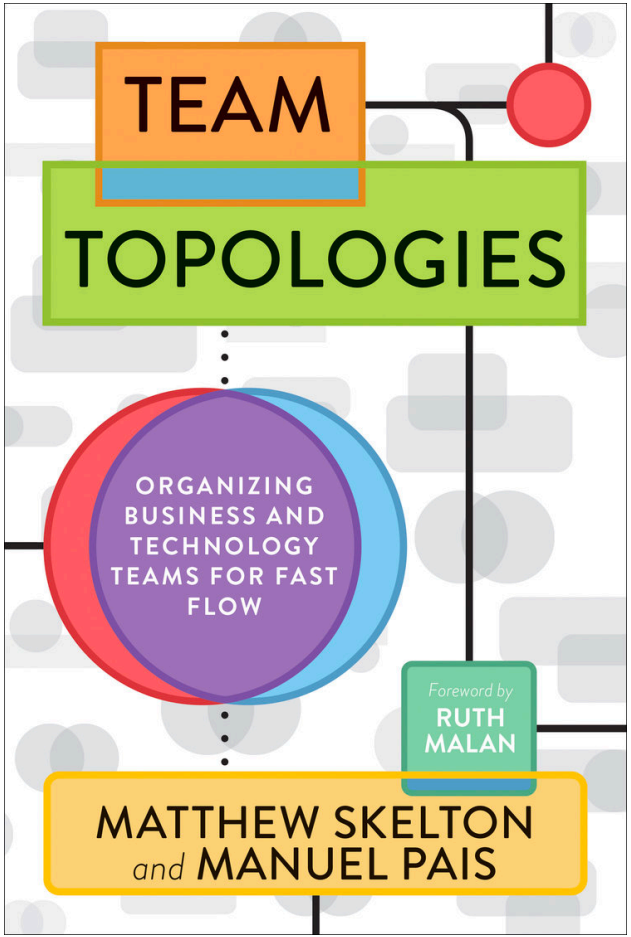
FT



You build things differently when you're the one that has to respond at 3am

FT

6. What your organisation looks like



"Organising teams for fast flow"

FT

Platform teams

Enabling teams

FT

Guardrails

These guardrails cover the things we expect a team to consider so that we build the right things and build things right. Following them ensures the things we build are safe, secure, and operable.

Use them as a checklist to make sure you are making the right decisions, or as an entry point to find out more about our principles, policies and standards.

Note that these are still being developed and added to.

1. Buy vs Build

Can you buy something to solve this problem rather than building it?

View page

2. Procurement

We need to go through a procurement process for any new relationship with a supplier, whether free or paid

View page

3. TGG Endorsement

Changes to the way the FT uses technology should be raised at the Tech Governance Group

View page

4. Adding to Biz Ops

Make sure the initial record has been created

View page

5. Security & Privacy

We need to build secure products and services

View page

6. Accessibility & Browser Support

We need to build websites that meet the needs of our customers

View page

7. Analytics, Logs & Metrics

We need to make sure we know how the things we built are used

View page

8. Change & Release Logging

All changes made at the FT must be logged

View page

Data & Storage

Choosing the right tool or platform for your data is a big decision but listed below are the solutions we use and recommend at the FT.

Filter by subject: Big Data | Databases

Amazon Athena

View page

Google BigQuery

View page

Amazon DynamoDB

View page

Elasticsearch

View page

MongoDB

View page

Neo4J

A database for working with highly interconnected data

View page

PostgreSQL

View page

Amazon S3

View page

In summary…

Going cloud native is a transformation

FT

It’s worth doing because it transforms your ability to add value to the business

FT

Thank you

- https://medium.com/ft-product-technology

FT