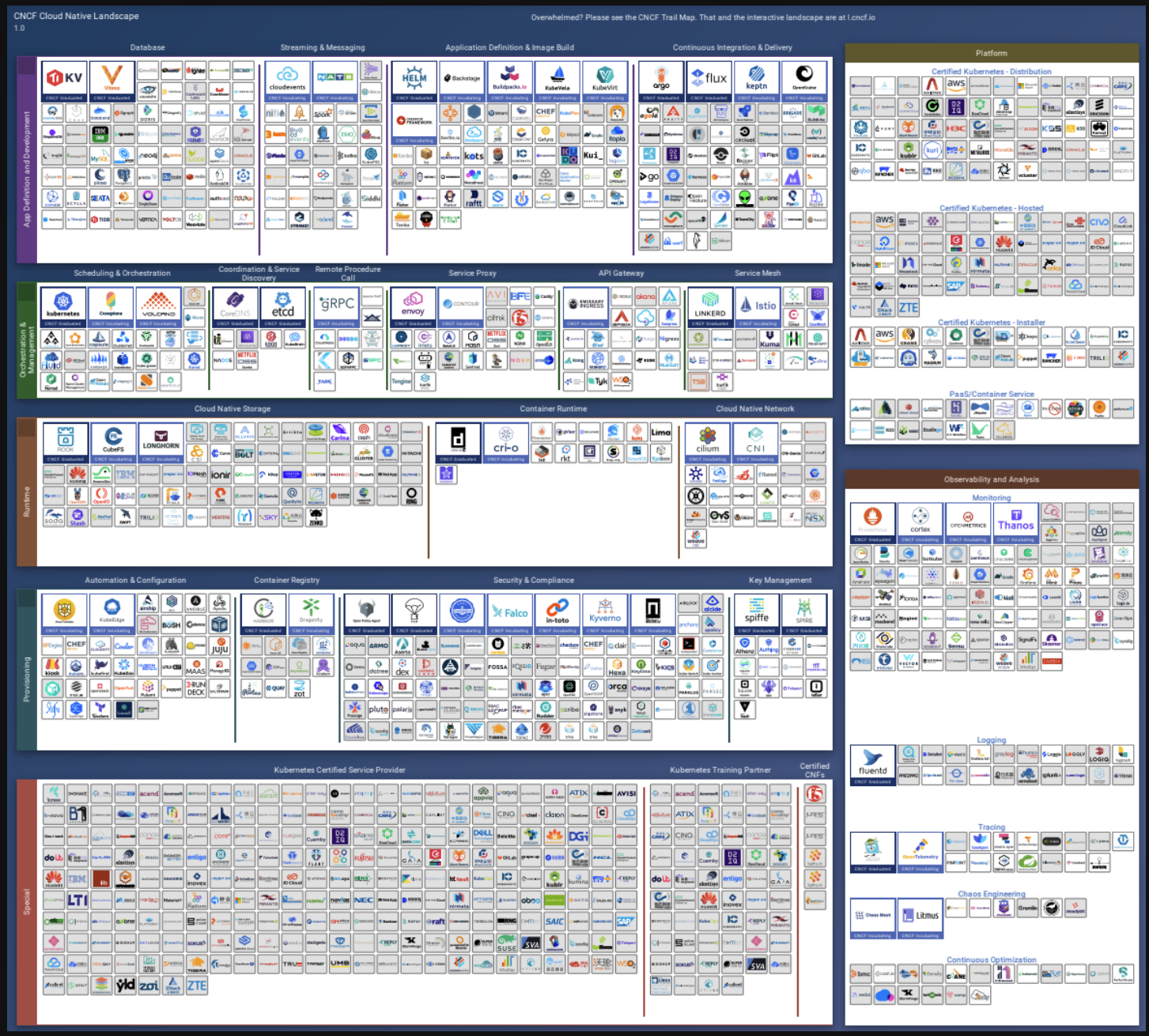
High-performing Engineering Teams, and the Holy Grail


CNCF Cloud Native Landscape
1.0
Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io
App Definition and Development
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Runtime
Cloud Native Storage
Container Runtime
Cloud Native Network
Provisioning
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Special
Kubernetes Certified Service Provider
Kubernetes Training Partner
Certified CNFs
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Continuous Optimization

Jeremy Meiss

Director, DevRel &

CircleCI

So back to the tech industry...


YOU SEEK THE HOLY GRAIL.

Forrester 2021 Total Economic Impact Study

Using best-in-class CI/CD platform can provide:

- $7.8 million saved from shorter software development cycles.
- $4.3 million recuperated in lost developer productivity.
- 50% decrease in annual infrastructure spend.
- $1.7 million estimated value of improved code quality.


2019
State of DevOps Report
puppet
circleci
splunk>
2017
State of DevOps Report
puppet
DORA
Delotte.
2018
State of DevOps Report
puppet
splunk>
2016
State of DevOps Report

Image: Consumer Choice Center

CI/CD Benchmarks for high-performing teams

Duration

Mean time to resolve

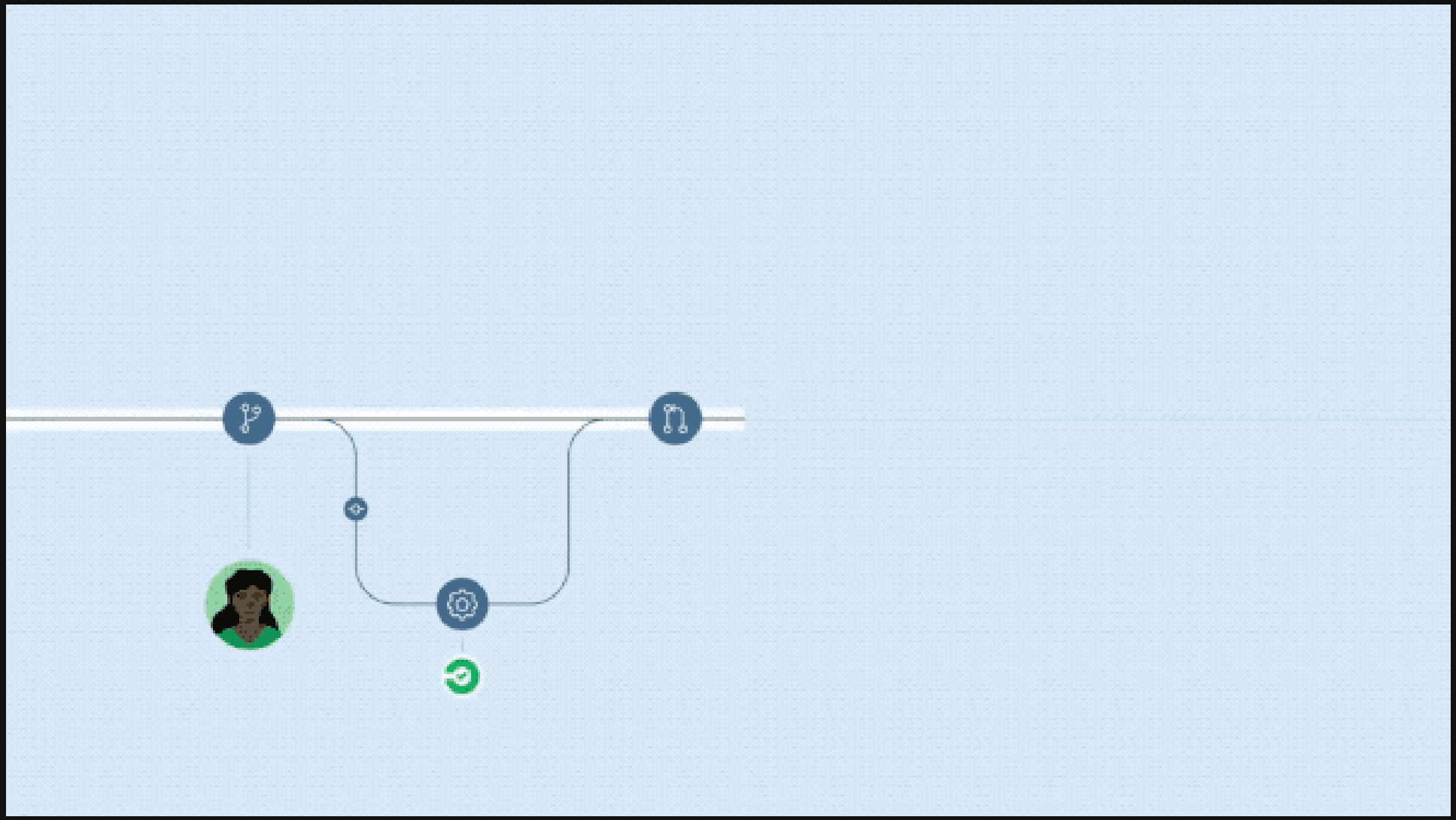
Success rate

Throughput

Duration

the foundation of software engineering velocity, measures the average time in minutes required to move a unit of work through your pipeline


And There Was Much Rejoicing



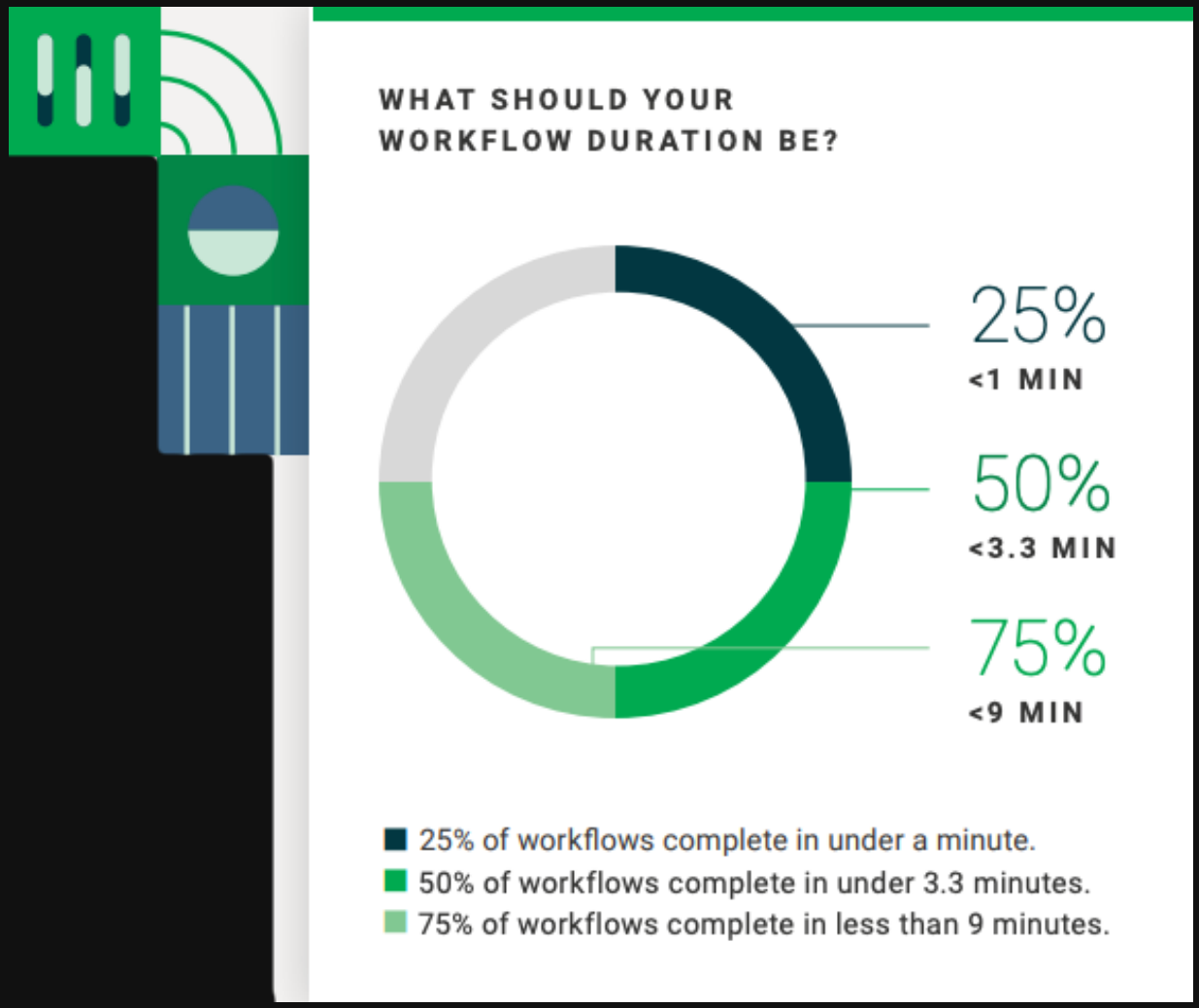
Duration Benchmark

<=10 minute builds

"a good rule of thumb is to keep your builds to no more than ten minutes. Many developers who use CI follow the practice of not moving on to the next task until their most recent check-in integrates successfully. Therefore, builds taking longer than ten minutes can interrupt their flow."

– **Paul M. Duvall (2007).** *Continuous Integration: Improving Software Quality and Reducing Risk*

Duration: What the data shows



Benchmark: 5-10mins

Improving test coverage

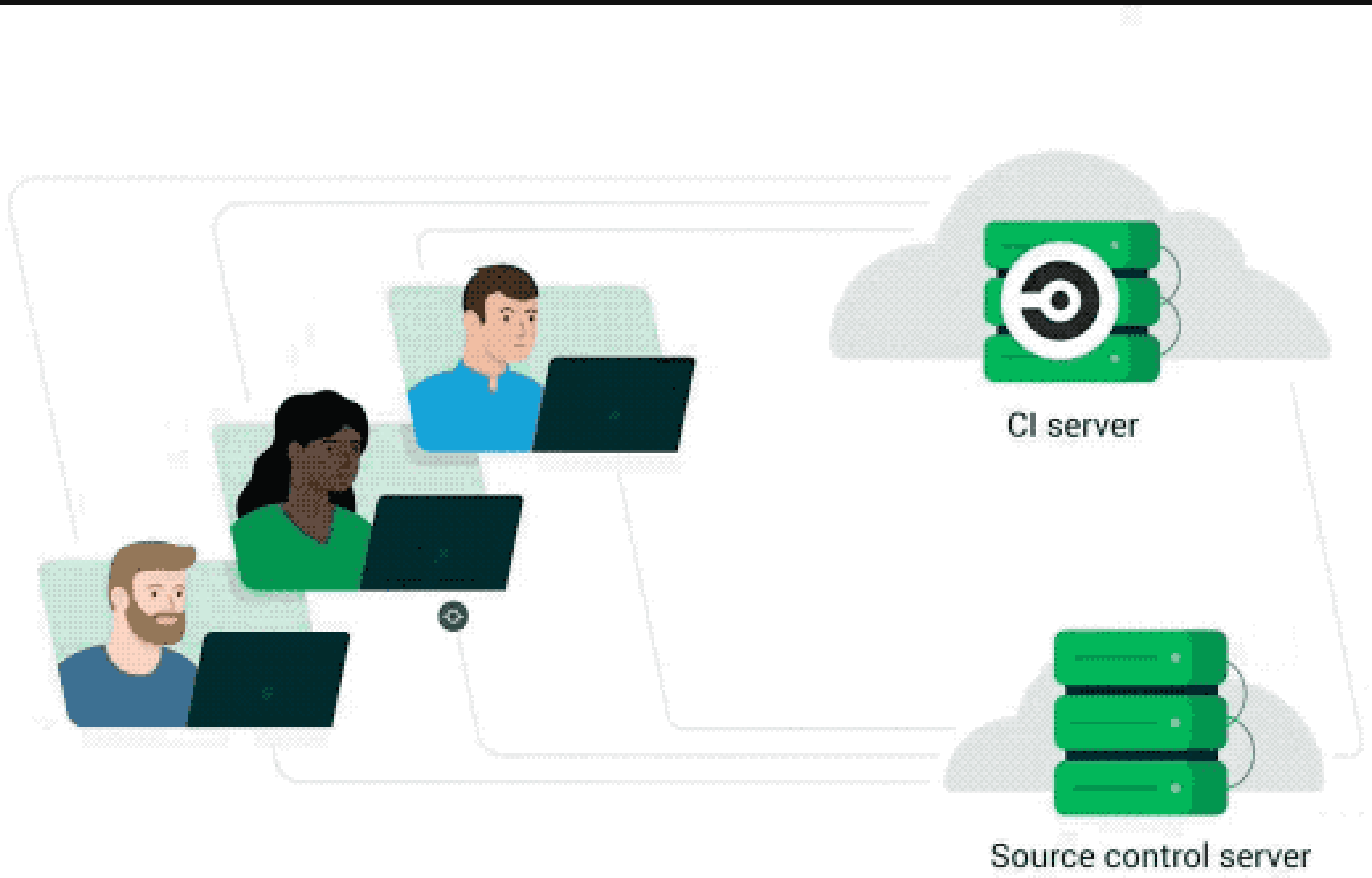
- Add unit, integration, UI, end-to-end testing across all app layers
- Add code coverage into pipelines to identify inadequate testing
- Include static and dynamic security scans to catch vulnerabilities
- Incorporate TDD practices by writing tests during design phase

Optimizing your pipelines

- Use test splitting & parallelism for simultaneous multiple tests
- Cache dependencies & data to avoid rebuilding unchanged code
- Use Docker images custom made for CI environments
- Choose the right machine size for your needs

Mean time to Recovery

the average time required to go from a failed build signal to a successful pipeline run


CI server
Source control server

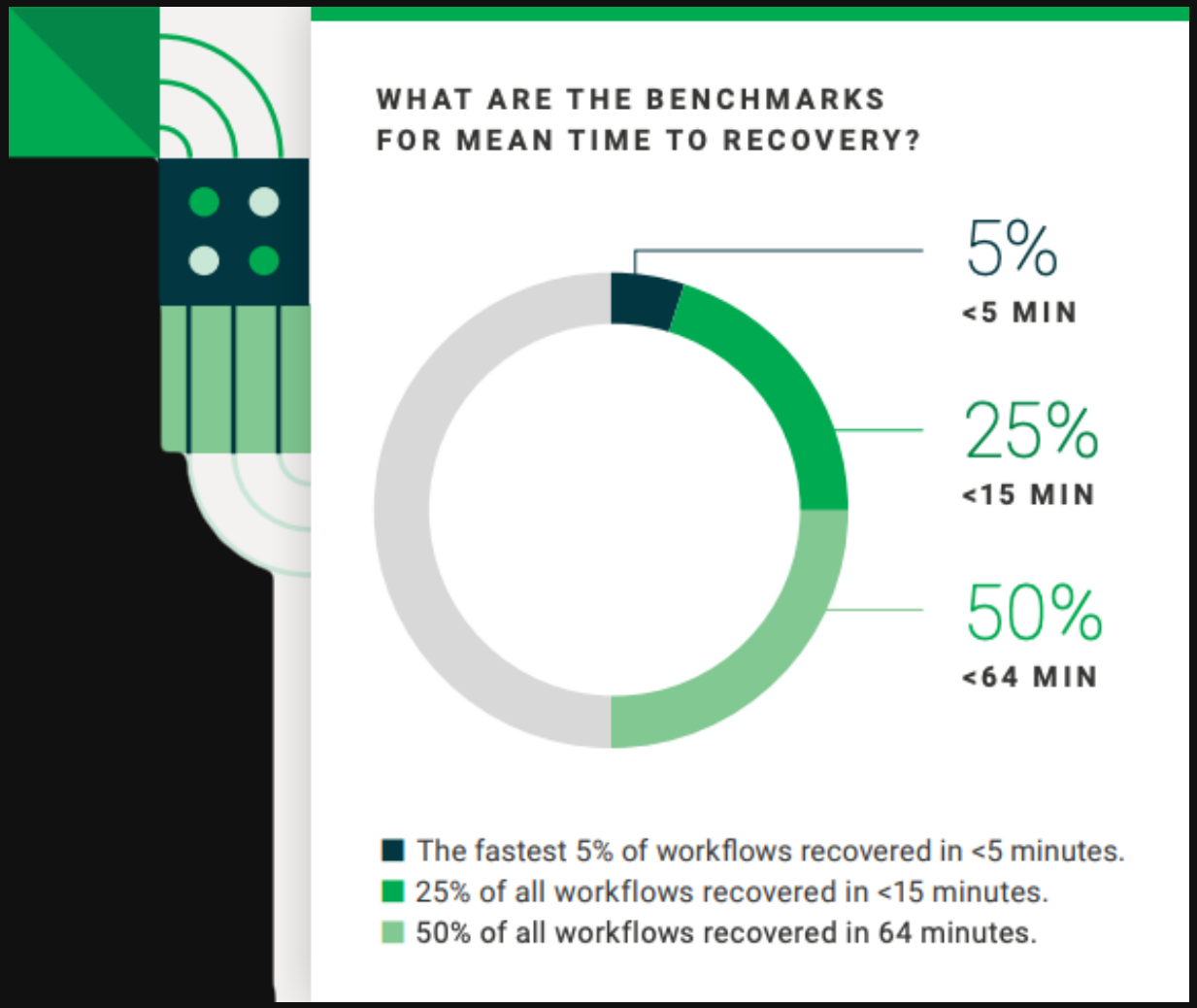
"A key part of doing a continuous build is that if the mainline build fails, it needs to be fixed right away. The whole point of working with CI is that you're always developing on a known stable base."

– **Martin Fowler (2006).** *"Continuous Integration."* Web blog post. MartinFowler.com

MTTR Benchmark

<=60min MTTR on default branches

MTTR: What the data shows



Benchmark: 60 mins

Treat your default branch as the lifeblood of your project

Getting to faster recovery times

- Treat default branch as the lifeblood of your project
- Set up instant alerts for failed builds (Slack, Pagerduty, etc.)
- Write clear, informative error messages for your tests
- SSH into the failed build machine to debug remote test env

Success rate

number of passing runs divided by the total number of runs over a period of time


now go away...
...or i will
taunt you a
second time!

Success rate benchmark

90%+ success rate on default branches

Success rate: What the data shows

Benchmark: 90%+ on default

Throughput

average number of workflow runs that an organization completes on a given project per day


“Look, in order to maintain high velocity,
your pipelines must be optimized.”

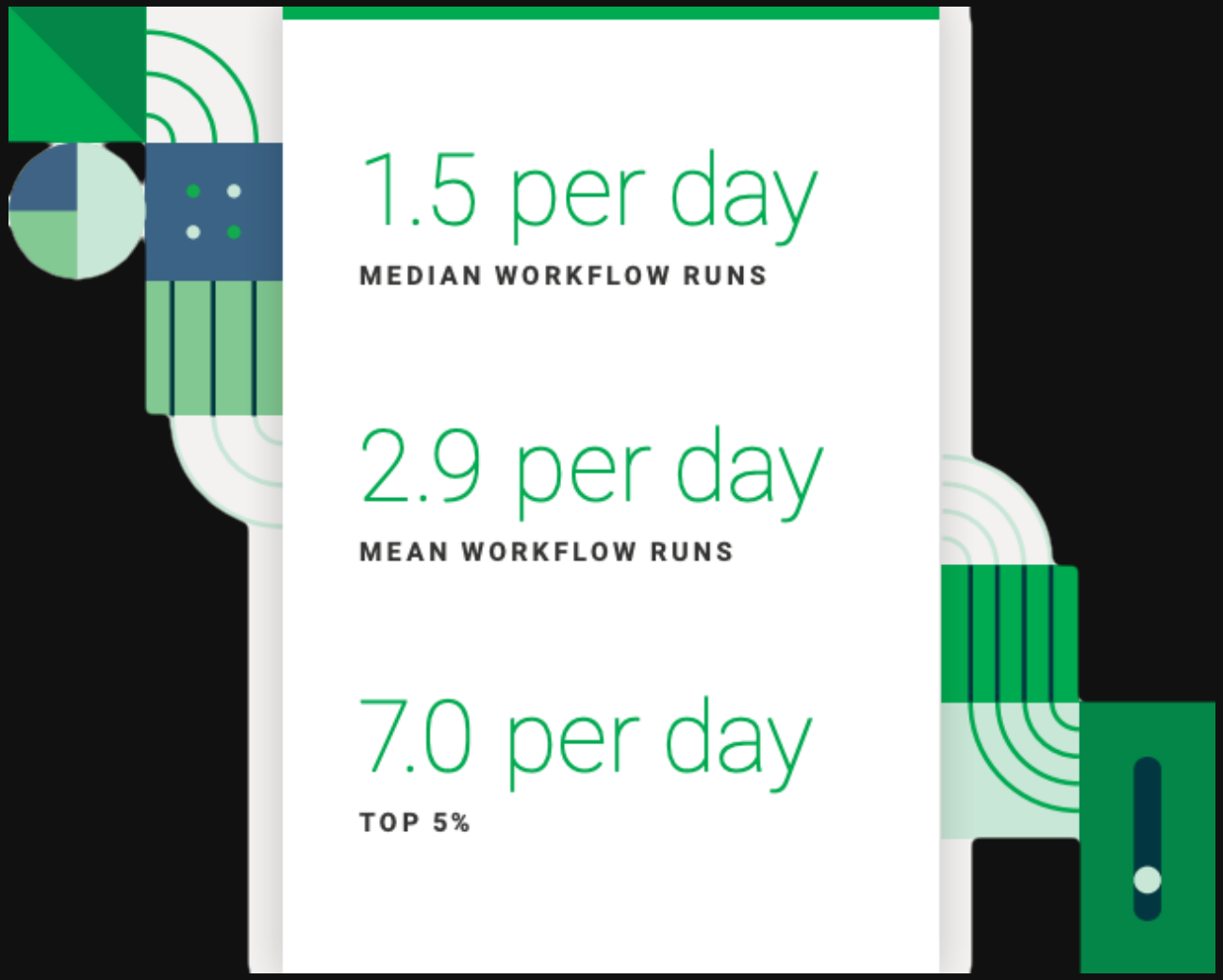

It's only a model.

Throughput benchmark

Throughput benchmark

It depends.

Throughput: What the data shows



Benchmark: at the speed of your business

Throughput is the most dependent on the other metrics

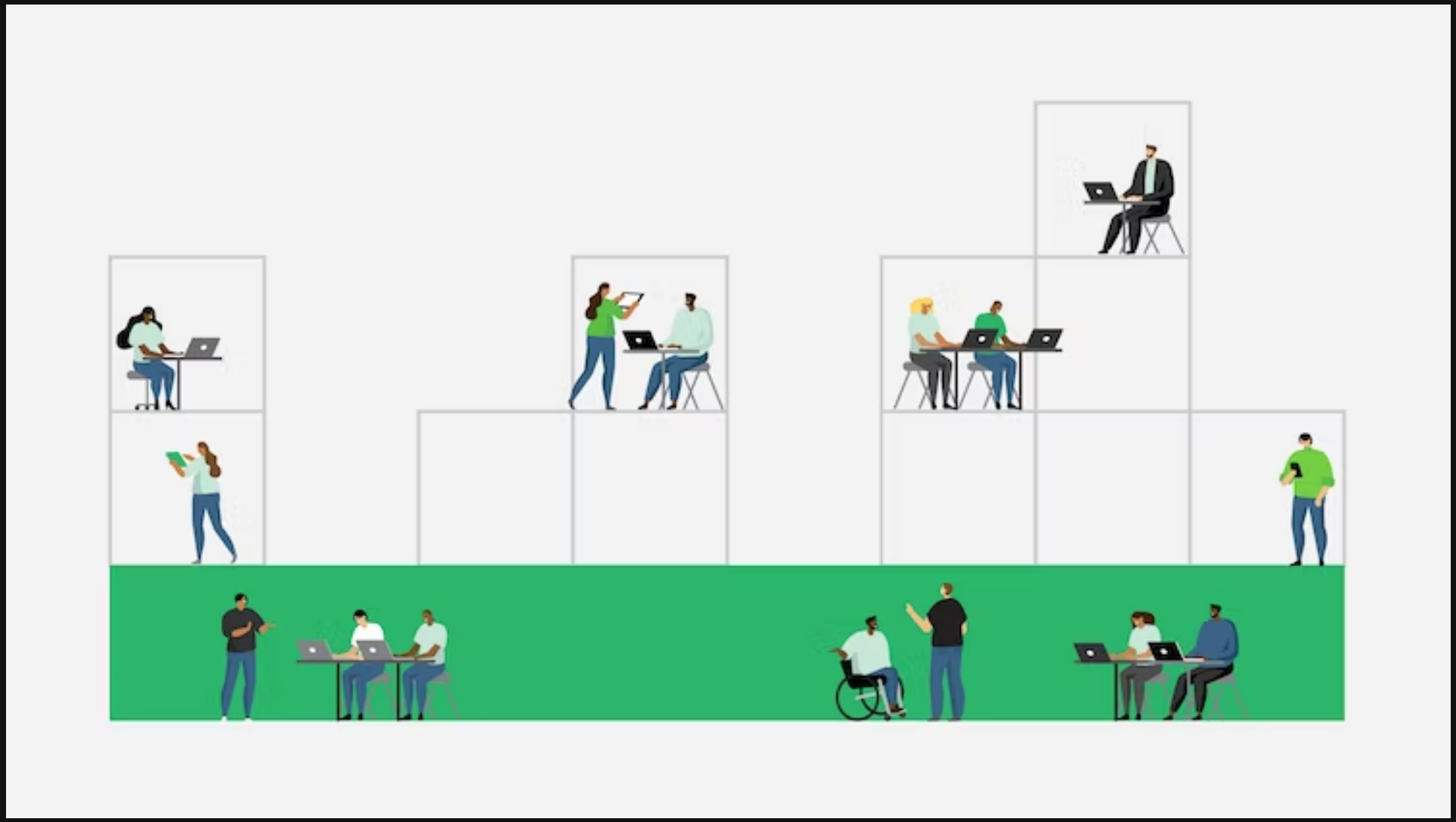
High-performing teams in 2023

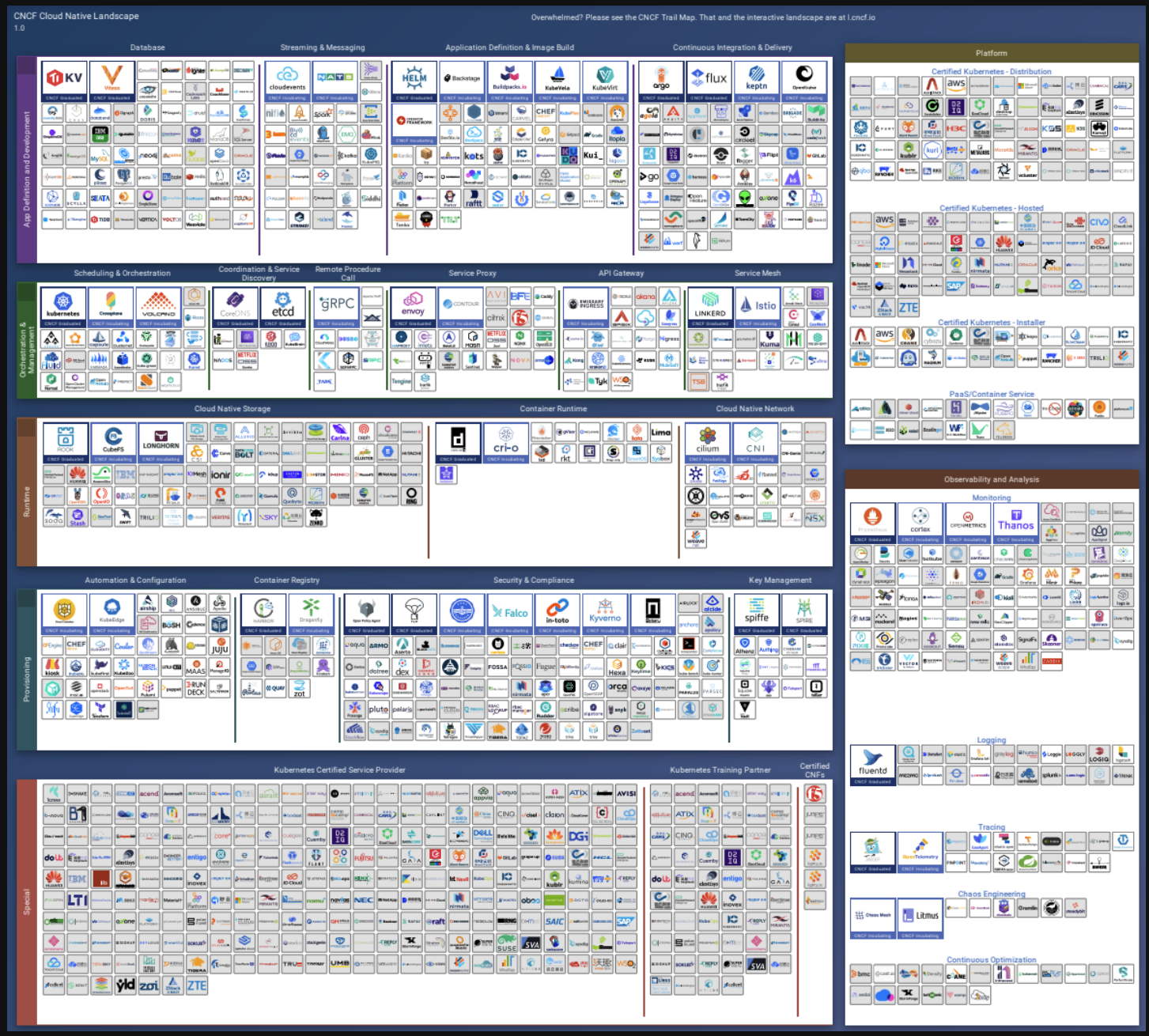
Metric	2020	2022	2023	Benchmark
Duration	4.0 minutes	3.7 minutes	3.3 minutes	10 minutes
TTR	72.9 minutes	73.6 minutes	64.3 minutes	<60 minutes
Success Rate	Avg 78% on default	Avg 77% on default	Avg 77% on default	Average >90% on default
Throughput	1.46 times per day	1.43 times per day	1.52 times per day	As often as your business requires - not a function of your tooling

The impact of Platform teams

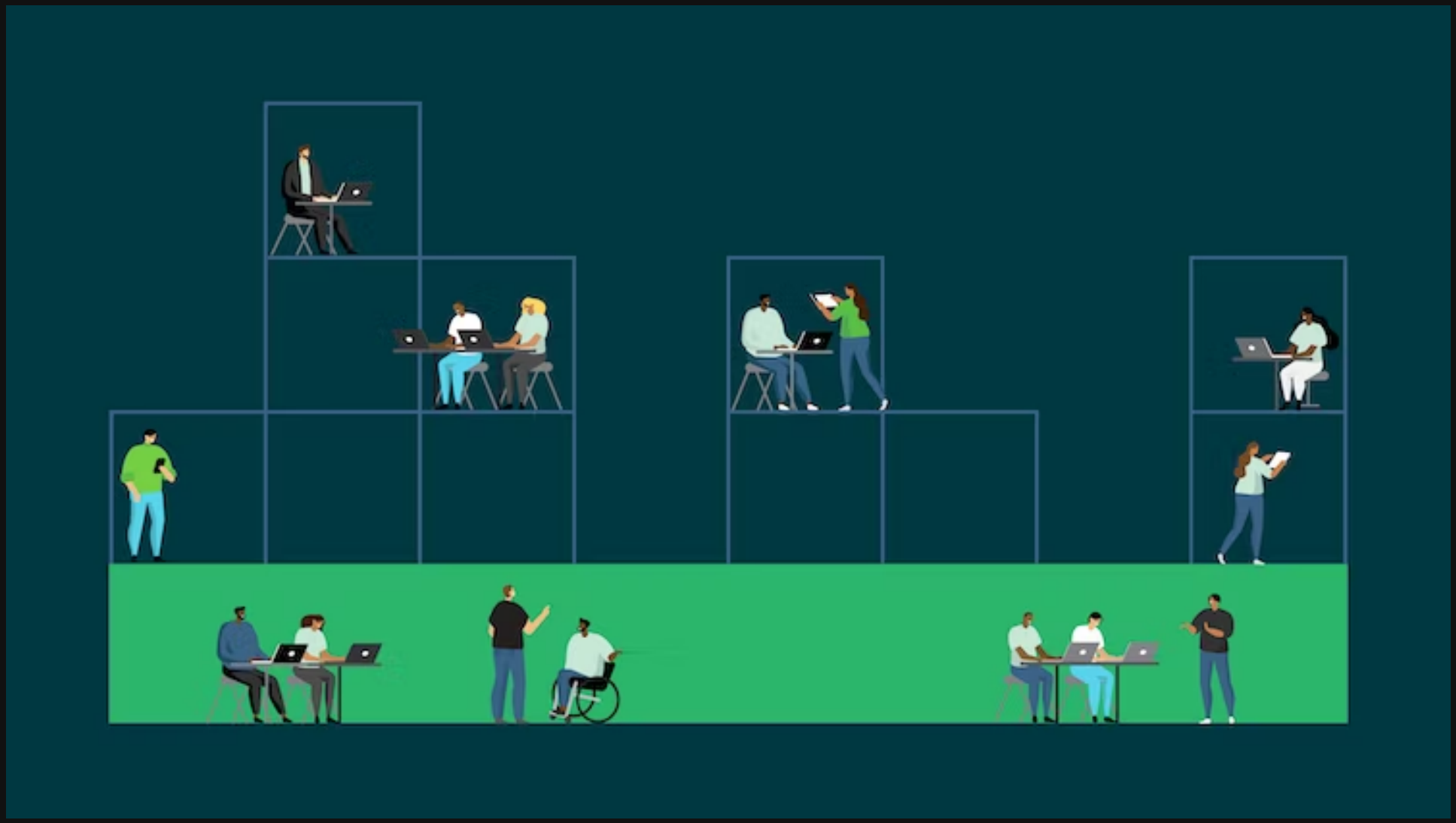
Platform Teams, DevOps, and YOU

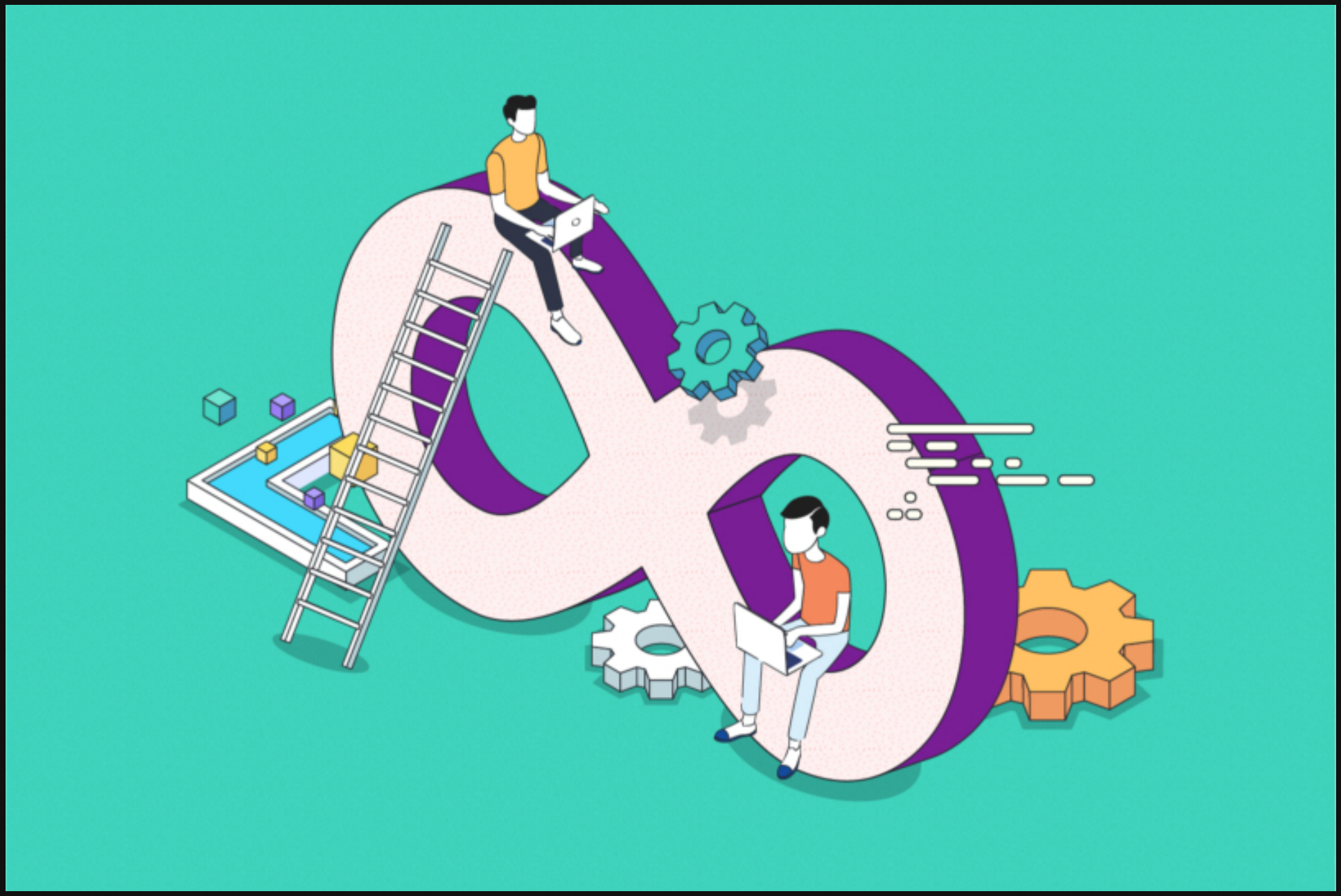
No, DevOps is not dead




CNCF Cloud Native Landscape
1.0
Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io
App Definition and Development
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Runtime
Cloud Native Storage
Container Runtime
Cloud Native Network
Provisioning
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Special
Kubernetes Certified Service Provider
Kubernetes Training Partner
Certified CNFs
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Continuous Optimization

The Rise of Platform Teams





Platform Perspective: Duration

- Identify and eliminate impediments to developer velocity
- Set guardrails and enforce quality standards across projects
- Standardize test suites & CI configs (shareable configs / policies)
- Welcome failed pipelines, i.e. fast failure
- Actively monitor, streamline, & parallelize pipelines across the org

Platform Perspective: MTTR

- Ephasise value of deploy-ready, default branches
- Set up effective monitoring & alerting systems, track recovery time
- Limit frequency & severity of broken builds w/ role-based policies
- Config- and Infrastructure-as-Code tools limit misconfig potential
- Actively monitor, streamline, & parallelize pipelines across the org

Platform Perspective: Success Rate

- With low success rates, look at MTTR & shorten recovery time first
- Set baseline success rate, aim for continuous improvement, look for flaky tests or test coverage gaps
- Be mindful of patterns & influence of external factors, i.e. decline on Fridays, holidays, etc.

Platform Perspective: Throughput

- Map goals to reality of internal & external business situations, i.e. customer expectations, competitive landscape, codebase complexity, etc.
- Capture a baseline, monitor for deviations
- Alleviate as much developer cognitive load from day-to-day work

2023 State of Software Delivery Report

circle.ci/sosdr2023

Thank You.

For feedback and swag:

circle.ci/jeremy

timeline.jerdog.me

@IAmJerdog

@jerdog

/in/jeremymeiss

@jerdog@hachyderm.io