

# High-performing engineering teams and the Holy Grail







# Jeremy Meiss



Director, DevRel & Community

 **@IAmJerdog**





So back to the tech industry....



A scene from the movie 'The Holy Grail' showing a group of knights in white surcoats and a black horse in a rocky landscape. The knights are standing in a line, facing right. A black horse is in the foreground on the left. The background is a rocky, hilly landscape under a blue sky.

**YOU SEEK THE HOLY GRAIL.**

# Forrester 2021 Total Economic Impact study

**Using best-in-class CI/CD platforms 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.



## 2016 State of DevOps Report

Presented by



## 2017 State of DevOps Report

Presented by:



puppet



DEVOPS RESEARCH & ASSESSMENT

Sponsored by:



splunk>



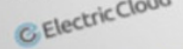
amazon



Atlassian



Deloitte



Electric Cloud

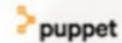


WAVEFRONT

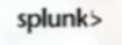
by VIMVOVO

## 2019 State of DevOps Report

Presented by



puppet



circleci



splunk>

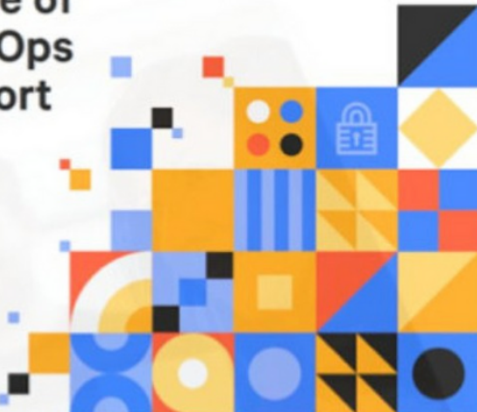
Sponsored by



ANITIAN



ServiceNow





A white rectangular card is placed on a teal, textured background that resembles denim. The card is slightly tilted and contains the text "ONE SIZE DOESN'T FIT ALL". The words "ONE SIZE" and "FIT ALL" are in black, while "DOESN'T" is in red.

**ONE SIZE  
DOESN'T  
FIT ALL**



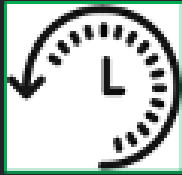


**THE HOLY HAND GRENADE FOR  
HIGH-PERFORMING ENGINEERING TEAMS**

# CI/CD Benchmarks for high-performing teams



Duration



Mean time  
to recovery



Success  
rate



Throughput





So what does the  
data say?



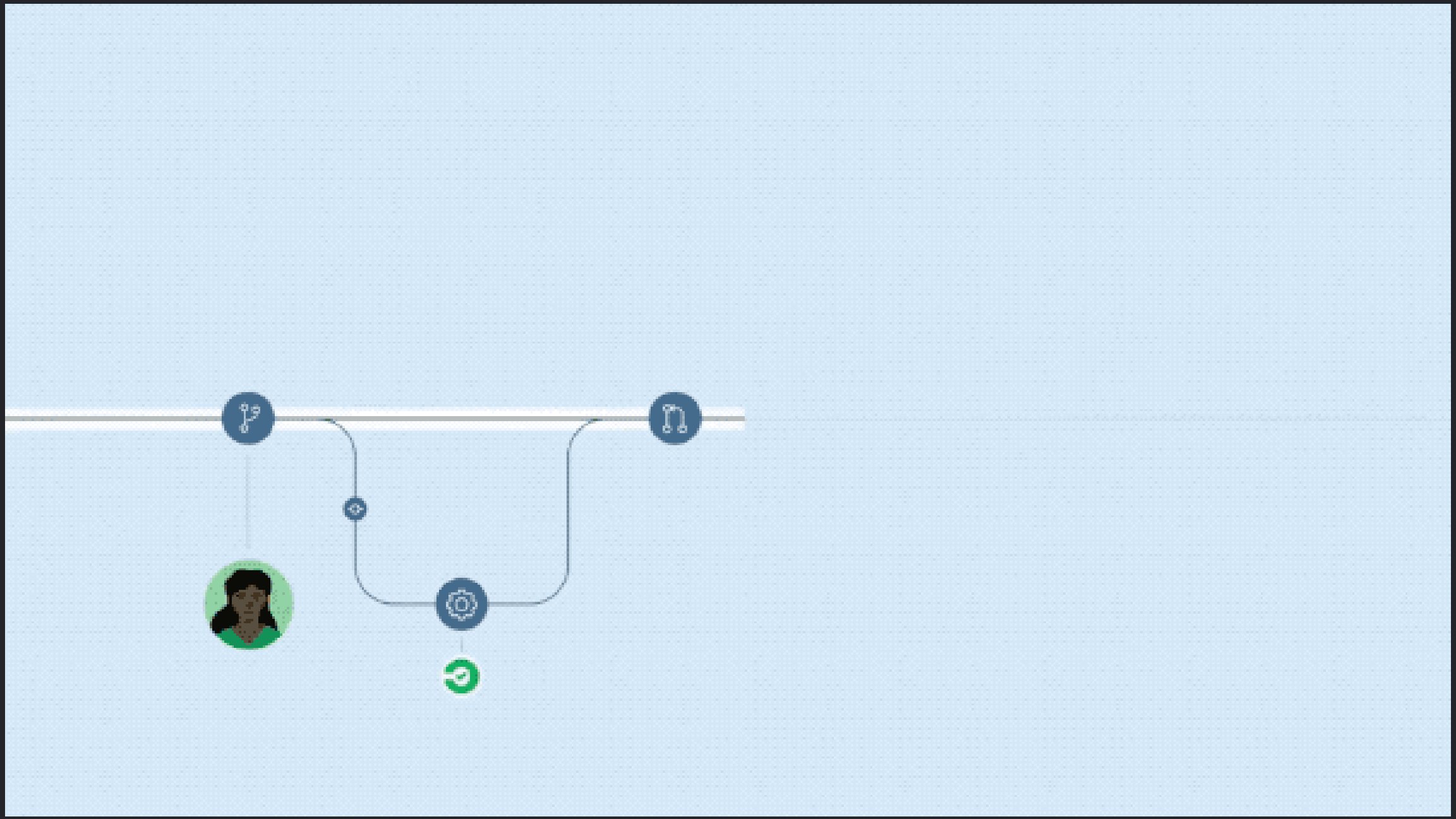
# Duration

*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

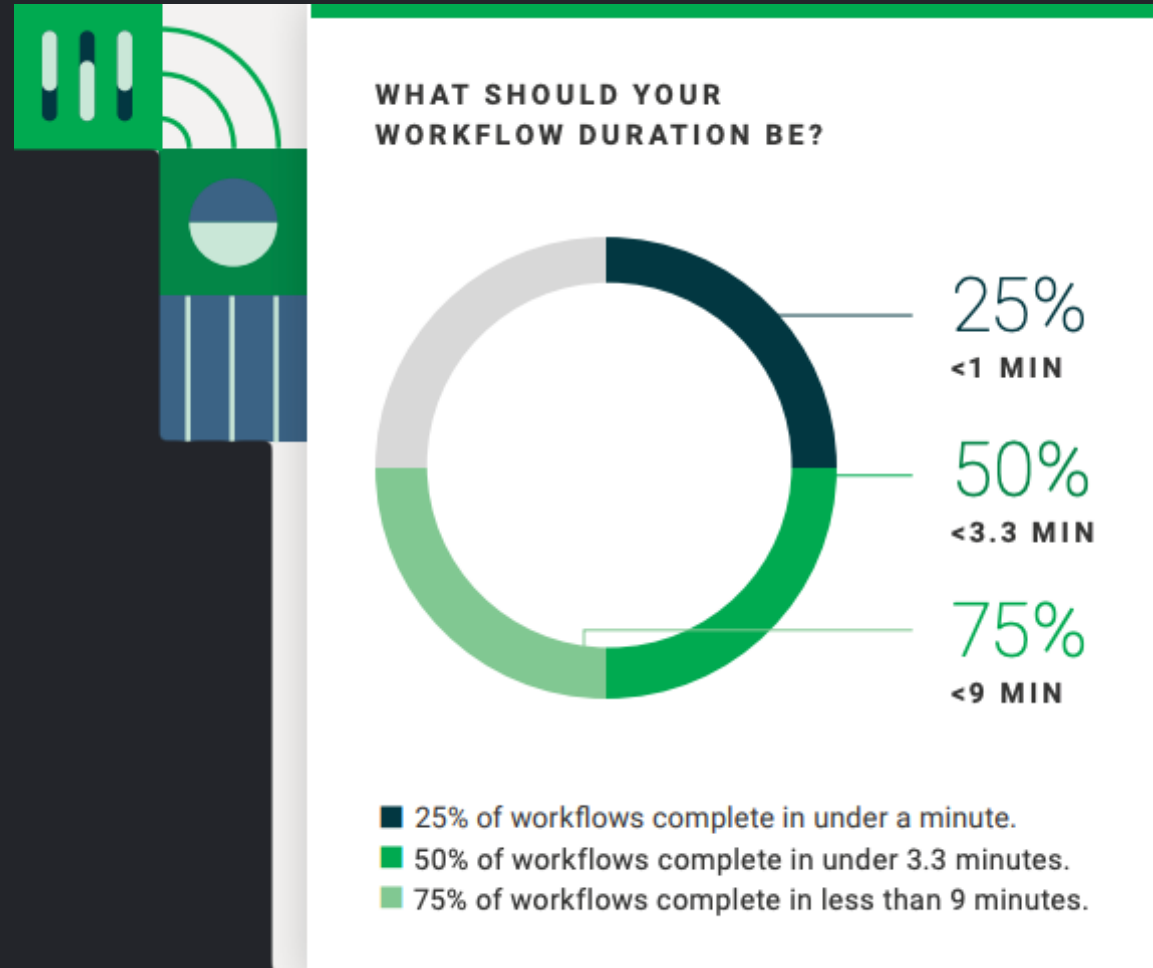
$\leq 10$  minute builds

*"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 checkin 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

”Why so much lower than  
the Duration benchmark?”

# Improving test coverage

- **Add unit, integration, UI, and end-to-end testing across all app layers**
- Incorporate code coverage tools 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** and **parallelism** to execute multiple tests simultaneously
- Cache dependencies and other data to avoid rebuilding unchanged portions
- Use Docker images custom made for CI environments
- Choose the right machine size for your needs

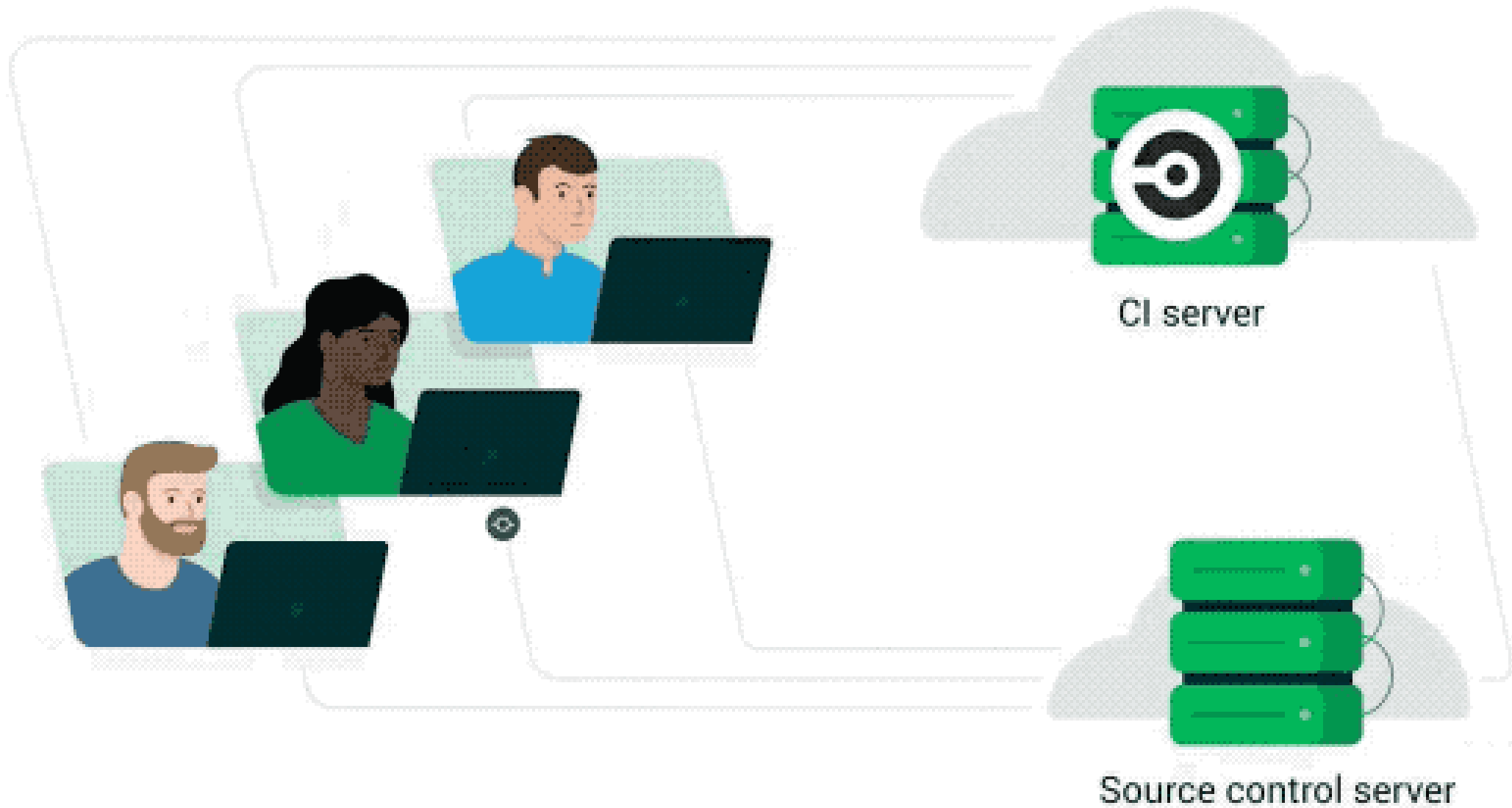




# Mean time to Recovery

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

Mean time to recovery is  
indicative of resilience



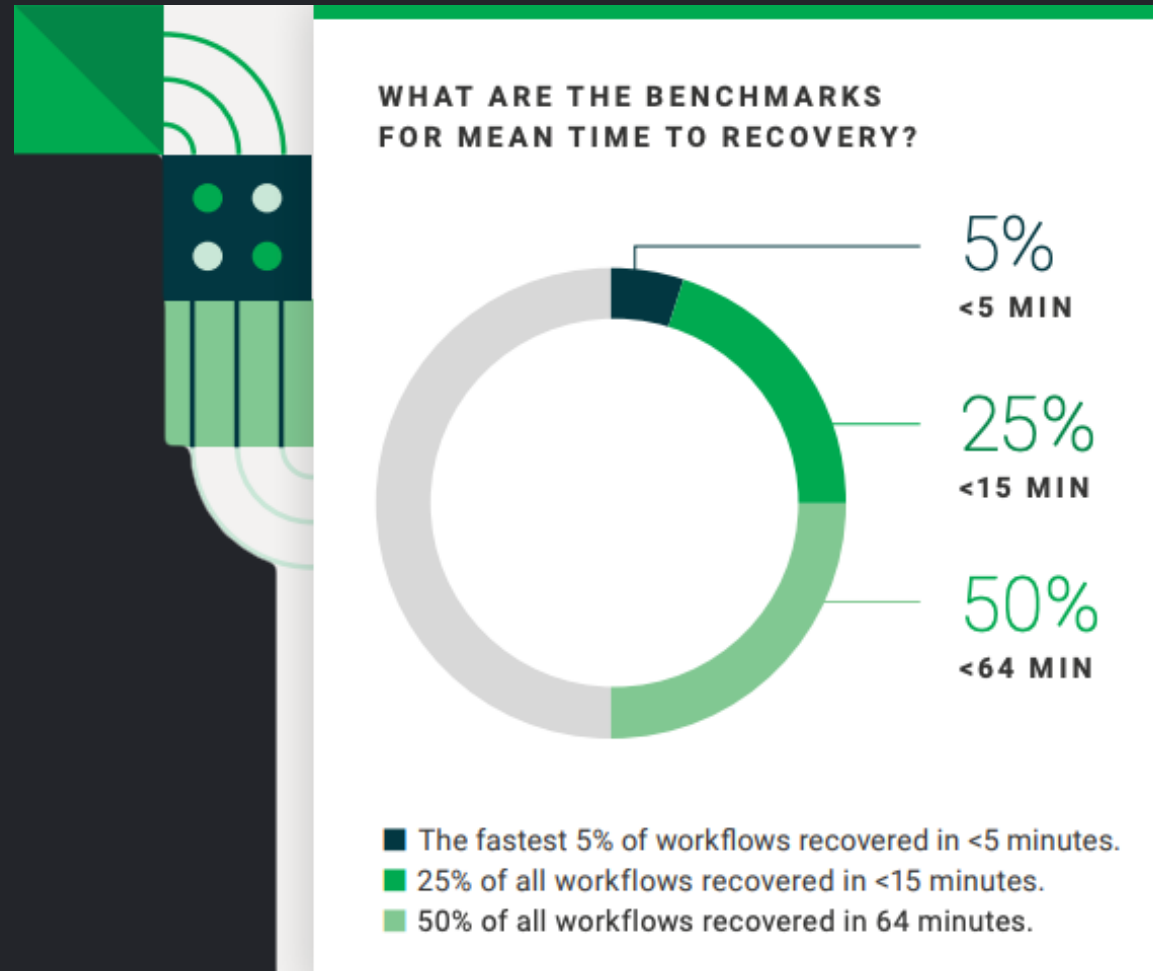
*"A key part of doing a continuous build is that if the mainline build fails, it needs to be fixed right away. The whole point of working with CI is that you're always developing on a known stable base."*

-- Fowler, Martin. "Continuous Integration." Web blog post. [MartinFowler.com](http://MartinFowler.com). 1 May 2006. Web.



$\leq 60$ min MTTR on  
default branches

# MTTR: What the data shows



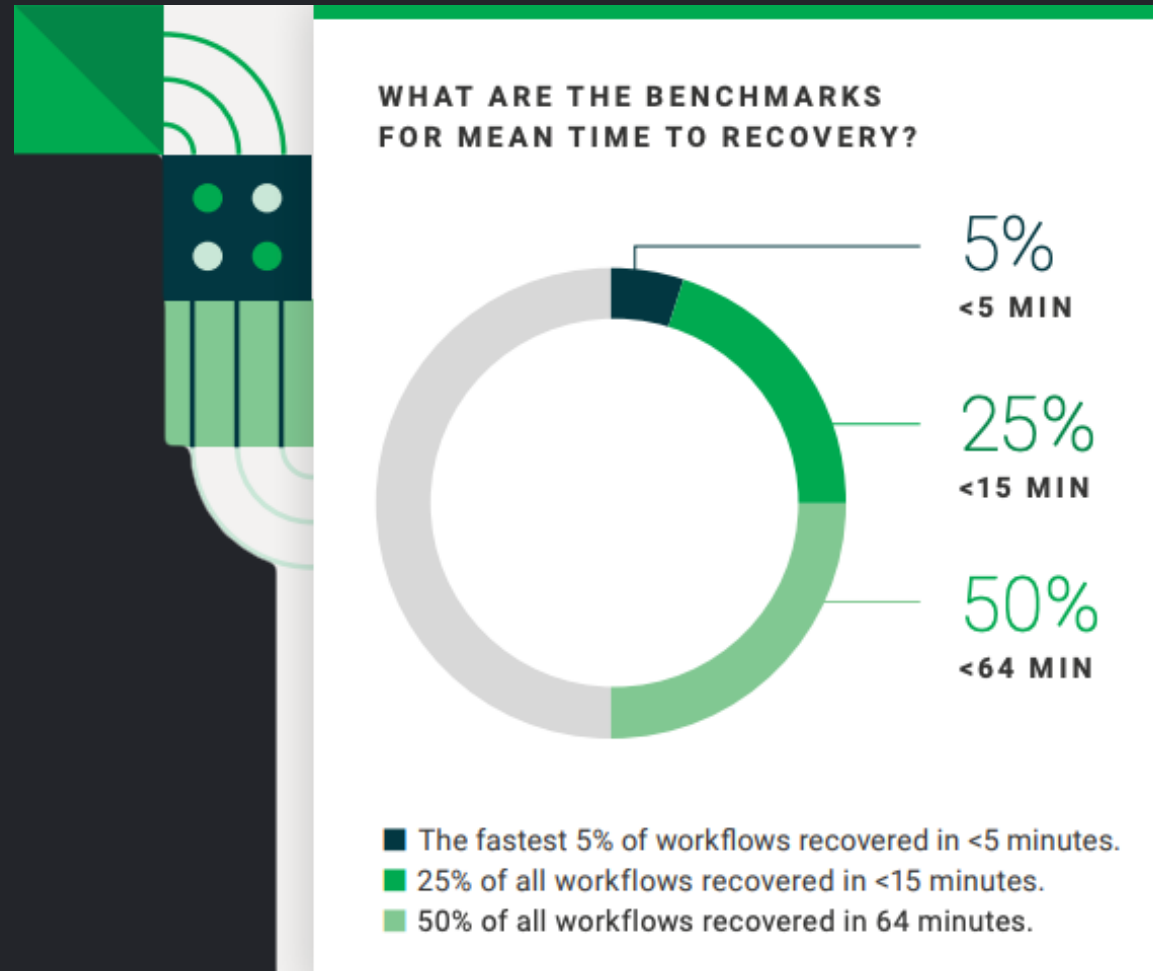
Benchmark: 60mins

# Two factors impacting reduced MTTR

- Economic pressures in the macro environment + rising competition in the micro environment, forcing teams to prioritize product stability and reliability over growth
- High performers increasingly rely on platform teams to achieve steadier and more resilient development pipelines with built-in recovery mechanisms.



# MTTR: What the data shows



Benchmark: 60mins

Treat your default branch as the  
lifeblood of your project



# Getting to faster recovery times

- Set up **instant alerts** for failed builds using services like Slack, Twilio, or Pagerduty.
- Write **clear, informative error messages** for your tests, allowing quick diagnosis
- Use **SSH into the failed build machine** to debug in the remote test environment.

# Success Rate

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

now go away...

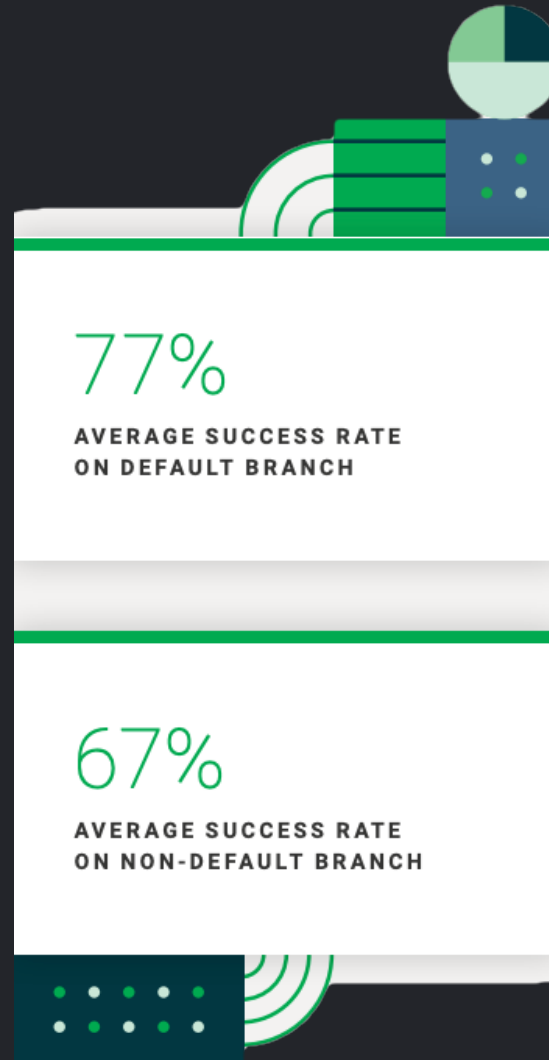


...or i will  
taunt you a  
second time!

# Success Rate Benchmark

90%+ Success rate on  
default branches

# Success rate: What the data shows



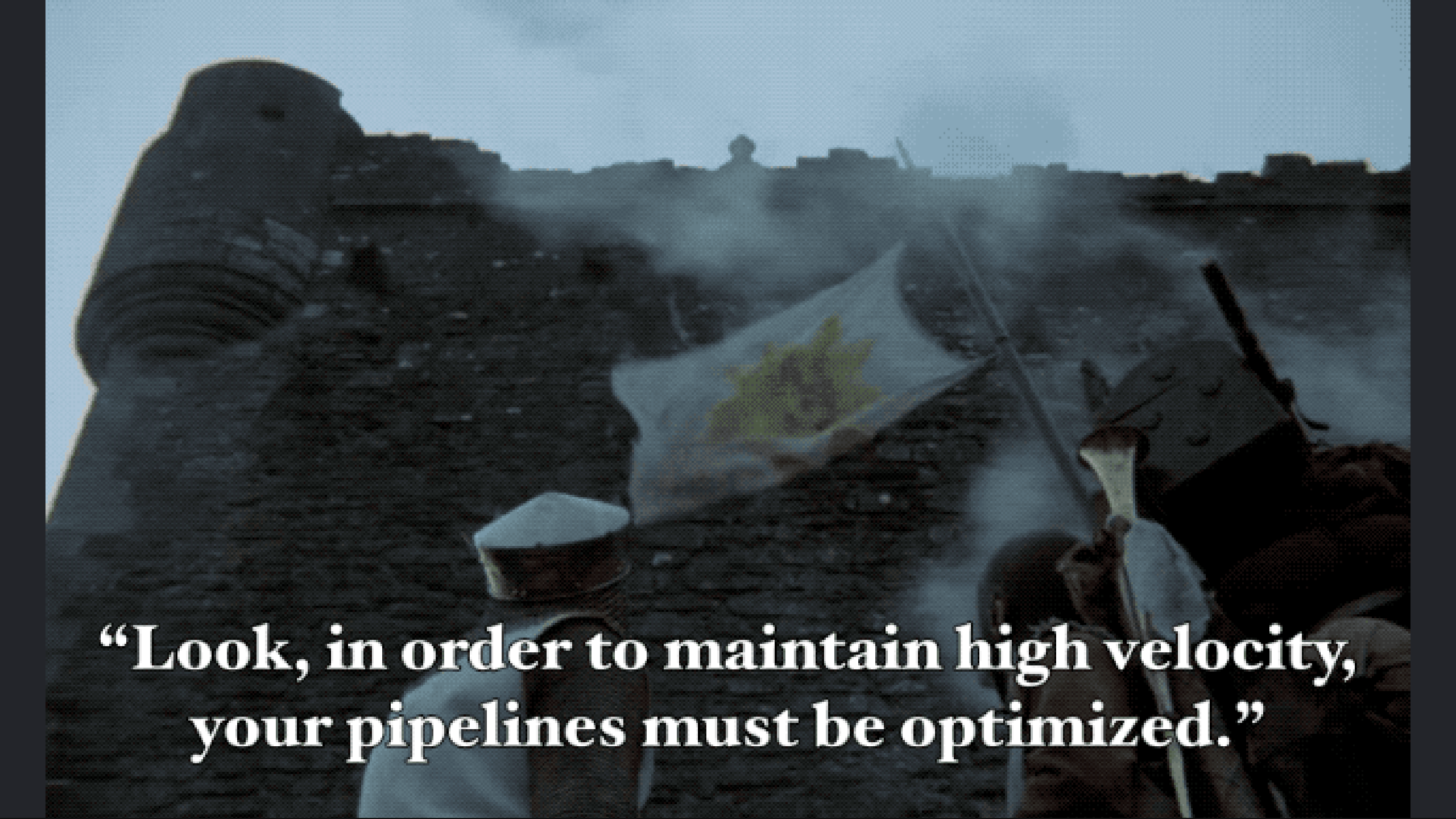
Benchmark: 90%+ on default





# Throughput

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

A historical photograph of a soldier in a trench during a battle. In the foreground, a Canadian flag with a maple leaf is visible, along with a military helmet. The soldier is wearing a white uniform and a cap. The background shows a cityscape with smoke rising from the buildings, indicating a battle scene.

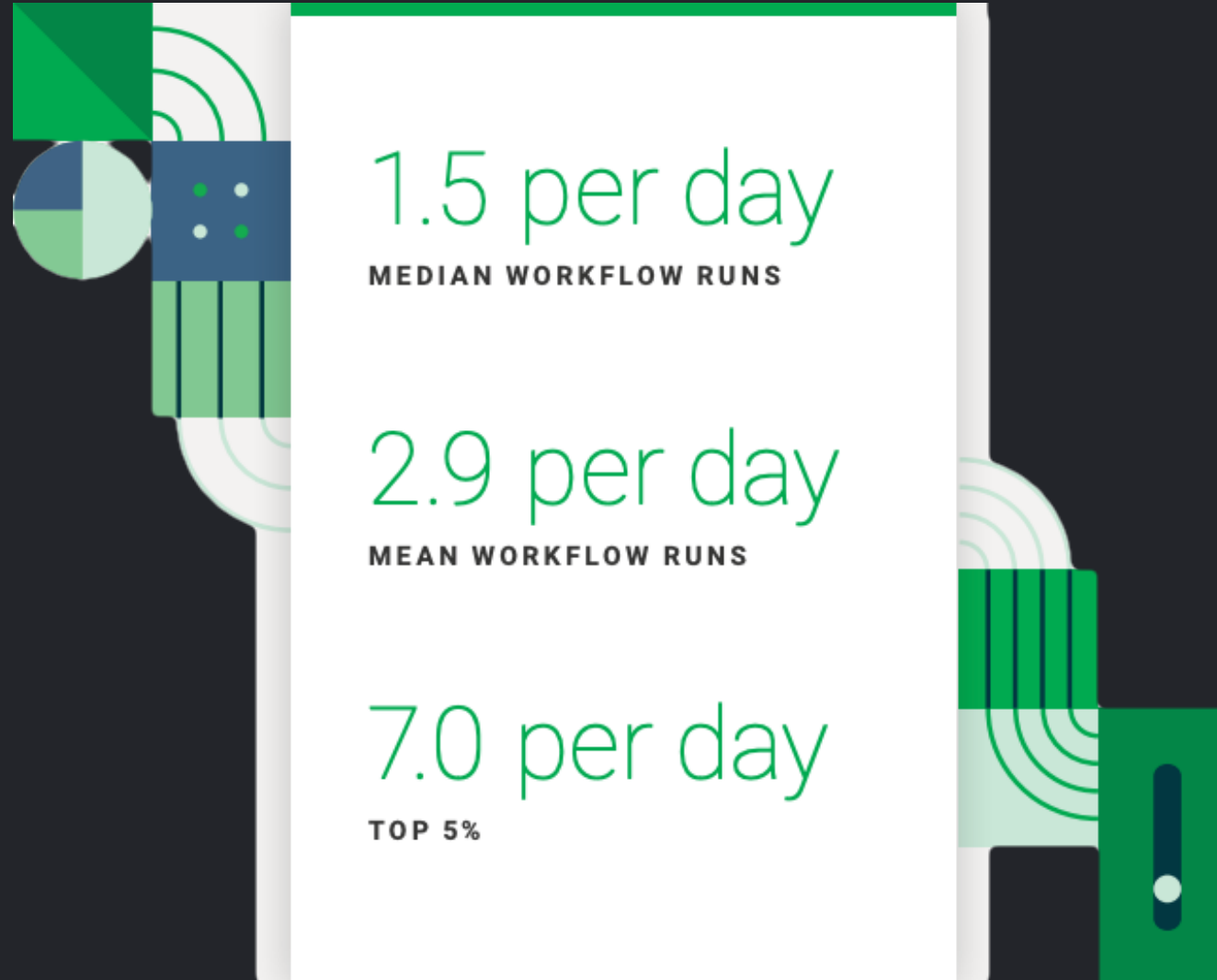
**“Look, in order to maintain high velocity,  
your pipelines must be optimized.”**



*It's only a model.*

It depends.

# Throughput: What the data shows



Benchmark: at the speed of your business





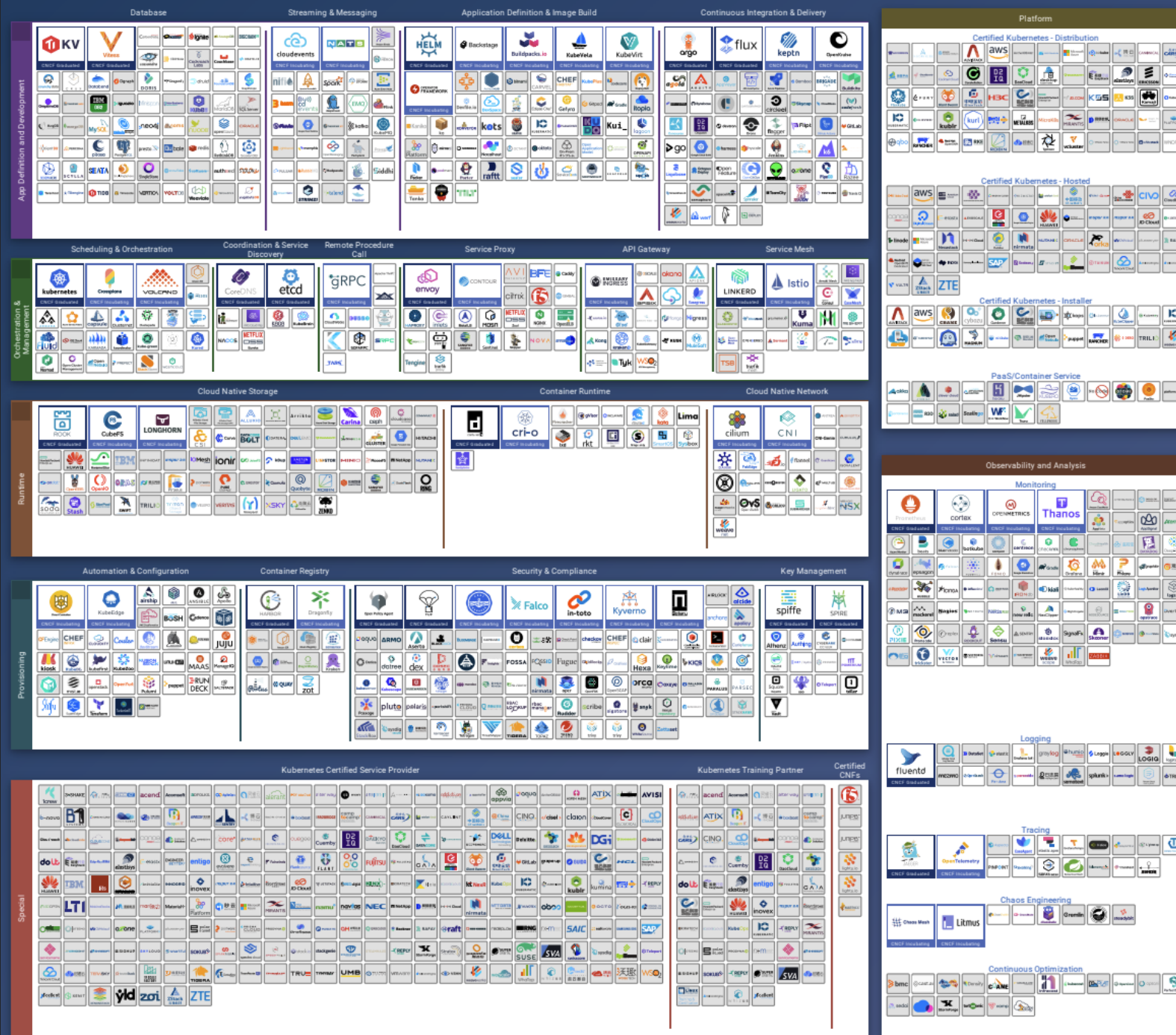
# High-Performing Teams in 2023

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

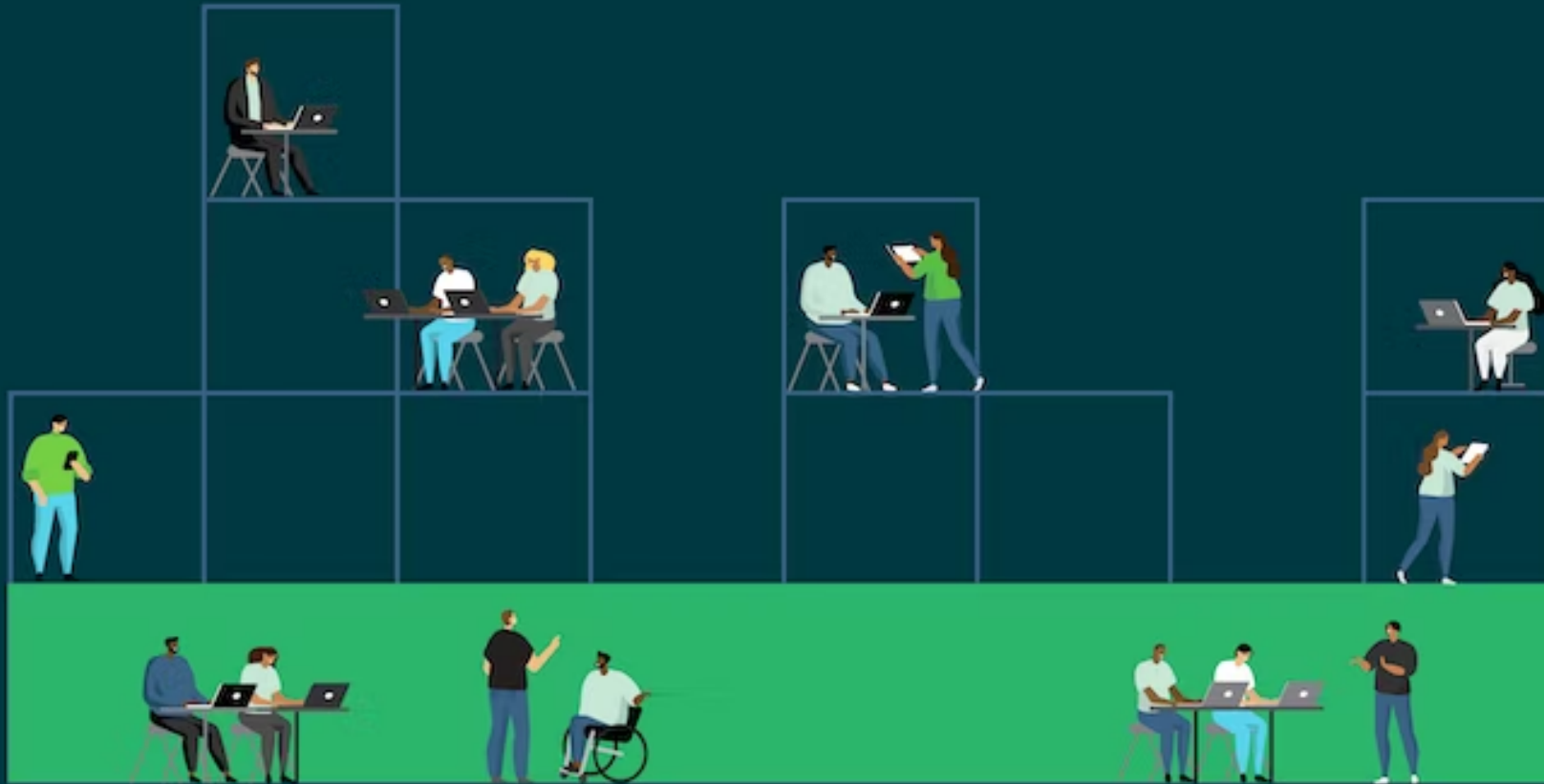


Platform Teams, DevOps, and you





# The Rise of Platform Teams



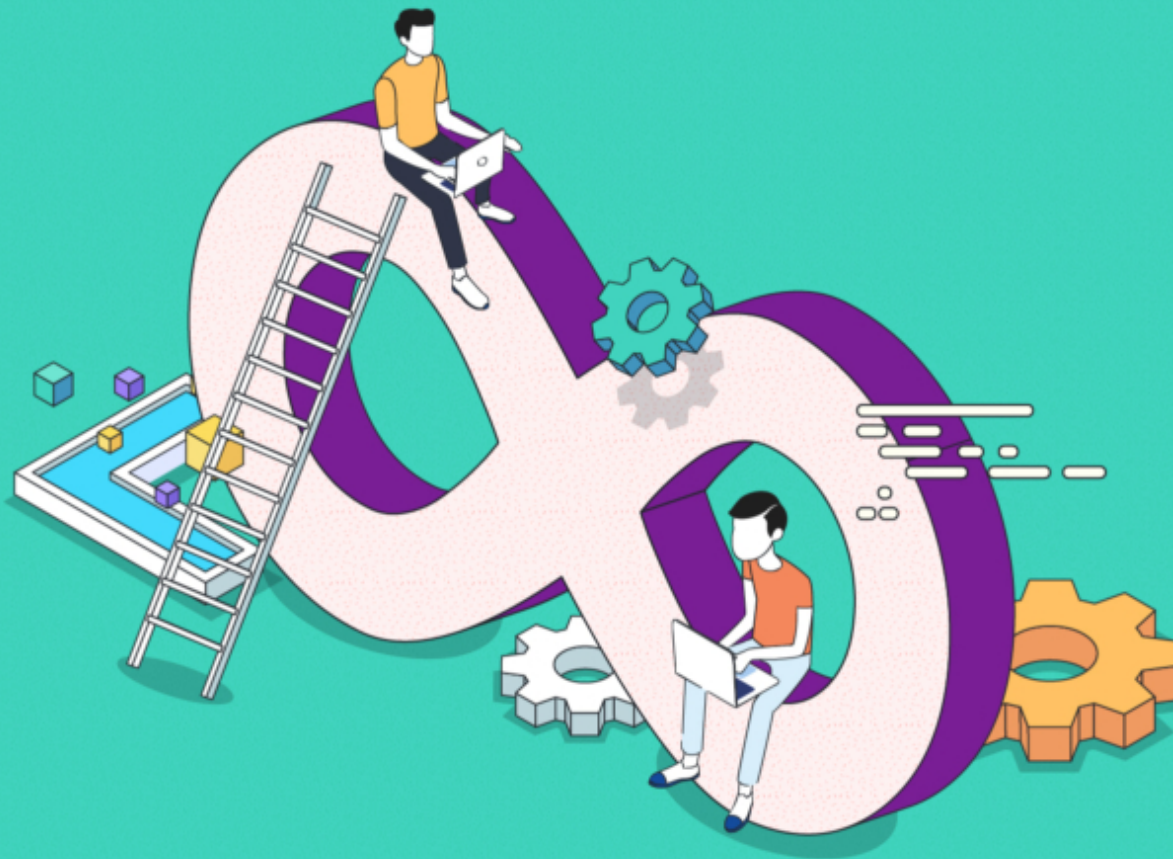


Image credit: [tsh.io](https://tsh.io)



PLATFORM  
PERSPECTIVE

# Duration

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



# Mean time to resolve

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



PLATFORM  
PERSPECTIVE

# Success rate

- With low success rates, look at MTTR and shorten recovery time first
- **Set baseline success rate, then aim for continuous improvement**, looking for flaky tests or gaps in test coverage
- Be mindful of patterns and 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**

Almost done....

...but first a little more interesting data

# Some Key Results We Found

- Largest productivity declines were concentrated around public holidays

# Some Key Results We Found

- Largest productivity declines were concentrated around public holidays
- Major, nationally significant events resulted in localized productivity drops

# Some Key Results We Found

- Largest productivity declines were concentrated around public holidays
- Major, nationally significant events resulted in localized productivity drops
- Politics, tech & cultural events, and major shopping days had no real impact



# Team size

- $\leq 100$  contributors
  - Throughput, Success rate, Duration improve
    - Duration:
      - $< 10$  contributors:  $< 2\text{min}$  on average
      - 51 - 100 contributors:  $\sim 6\text{min}$  on average
      - 100+ contributors:  $\sim 5\text{min}$  on average
- $> 100$  contributors
  - Duration and MTTR fall
  - Throughput remains steady

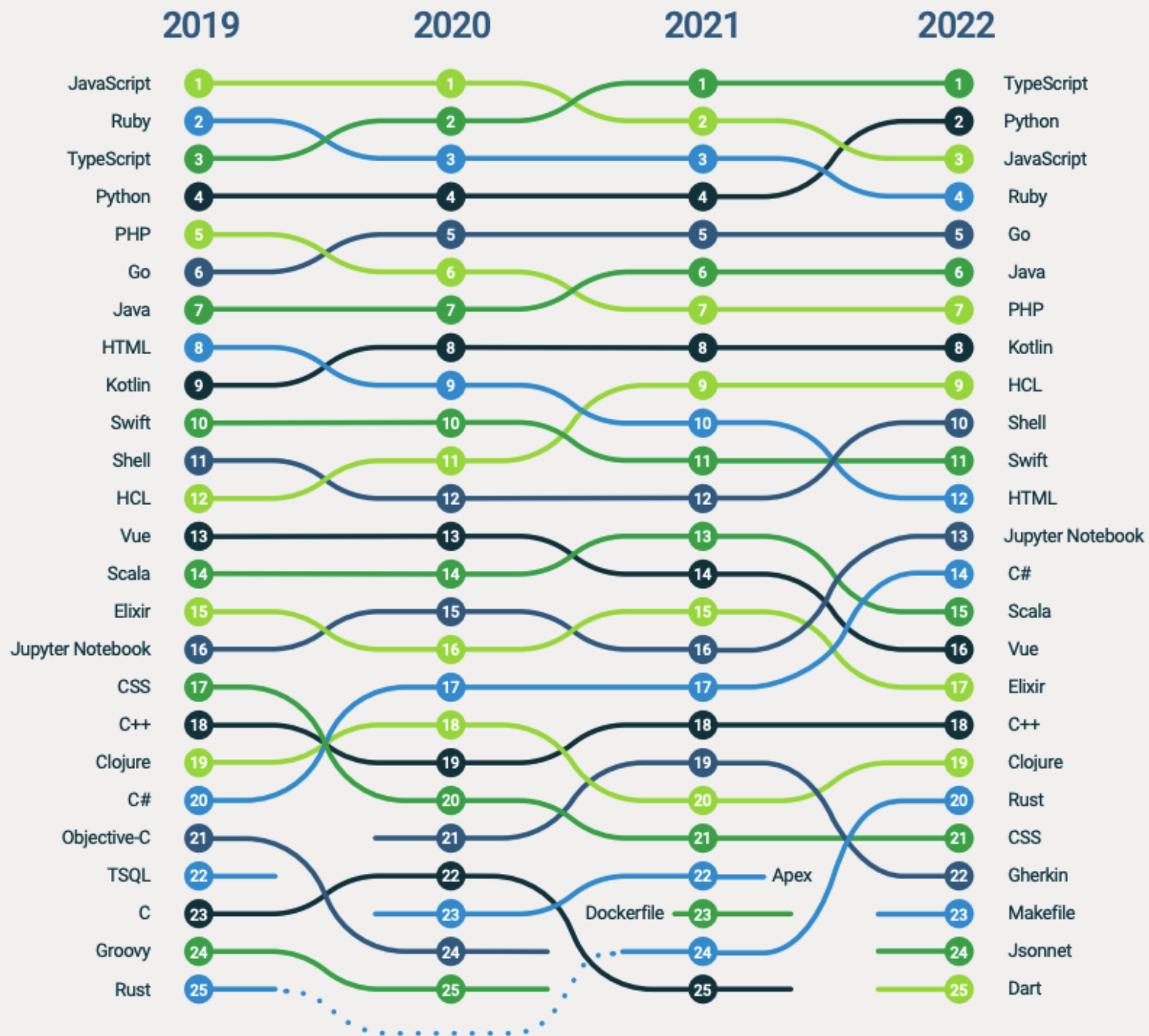




# Company size

- IT sector
  - Duration: 3.4min
  - Throughput: 1.56 workflows
  - MTTR: 1hr, 8min
- Automotive, Retail, Insurance sectors
  - MTTR: 4hrs +

*"Surely <insert programming language>  
helps me achieve the "Holy Grail"!?"*



	Duration
1	Makefile
2	LookML
3	Shell
4	HCL
5	Mustache
6	Nix
7	SaltStack
8	Open Policy Agent
9	Smarty
10	Dockerfile
11	Jsonnet
12	Batchfile
13	Liquid
14	VCL
15	EJS
16	Jinja
17	PLSQL
18	PowerShell
19	SCSS
20	Haml
21	R
22	CSS
23	Python
24	C#
25	Vue

	Duration	MTTR
1	Makefile	Gherkin
2	LookML	JavaScript
3	Shell	PHP
4	HCL	HCL
5	Mustache	Go
6	Nix	Ruby
7	SaltStack	TypeScript
8	Open Policy Agent	Perl
9	Smarty	Python
10	Dockerfile	HTML
11	Jsonnet	Java
12	Batchfile	Clojure
13	Liquid	CSS
14	VCL	Elixir
15	EJS	Vue
16	Jinja	Shell
17	PLSQL	Kotlin
18	PowerShell	C#
19	SCSS	Rust
20	Haml	Dart
21	R	Jupyter Notebook
22	CSS	Jinja
23	Python	PL/pgSQL
24	C#	C
25	Vue	C++

	Duration	MTTR	Success Rate
1	Makefile	Gherkin	Mustache
2	LookML	JavaScript	Perl
3	Shell	PHP	Smarty
4	HCL	HCL	Go
5	Mustache	Go	PL/pgSQL
6	Nix	Ruby	HCL
7	SaltStack	TypeScript	Vue
8	Open Policy Agent	Perl	Scala
9	Smarty	Python	Makefile
10	Dockerfile	HTML	Elixir
11	Jsonnet	Java	Shell
12	Batchfile	Clojure	HTML
13	Liquid	CSS	Jupyter Notebook
14	VCL	Elixir	Rust
15	EJS	Vue	RobotFramework
16	Jinja	Shell	C#
17	PLSQL	Kotlin	Python
18	PowerShell	C#	Clojure
19	SCSS	Rust	TypeScript
20	Haml	Dart	Ruby
21	R	Jupyter Notebook	Jinja
22	CSS	Jinja	C
23	Python	PL/pgSQL	PHP
24	C#	C	Kotlin
25	Vue	C++	Dockerfile

	Duration	MTTR	Success Rate	Throughput
1	Makefile	Gherkin	Mustache	Hack
2	LookML	JavaScript	Perl	Jsonnet
3	Shell	PHP	Smarty	Dart
4	HCL	HCL	Go	Swift
5	Mustache	Go	PL/pgSQL	Elixir
6	Nix	Ruby	HCL	Ruby
7	SaltStack	TypeScript	Vue	Mustache
8	Open Policy Agent	Perl	Scala	Jupyter Notebook
9	Smarty	Python	Makefile	TypeScript
10	Dockerfile	HTML	Elixir	Python
11	Jsonnet	Java	Shell	Elm
12	Batchfile	Clojure	HTML	Liquid
13	Liquid	CSS	Jupyter Notebook	Haskell
14	VCL	Elixir	Rust	Starlark
15	EJS	Vue	RobotFramework	PL/pgSQL
16	Jinja	Shell	C#	Jinja
17	PLSQL	Kotlin	Python	Lua
18	PowerShell	C#	Clojure	HTML
19	SCSS	Rust	TypeScript	Clojure
20	Haml	Dart	Ruby	Apex
21	R	Jupyter Notebook	Jinja	XSLT
22	CSS	Jinja	C	Perl
23	Python	PL/pgSQL	PHP	C++
24	C#	C	Kotlin	PureScript
25	Vue	C++	Dockerfile	Gherkin

## 2020 Report



<https://circle.ci/ssd2020>

## Full 2022 Report



<https://circle.ci/ssd2022>



# 2023 State of Software Delivery Report



[circle.ci/sosdr2023](https://circle.ci/sosdr2023)

# Thank You.

For feedback and swag: [circle.ci/jeremy](https://circle.ci/jeremy)



[timeline.jerdog.me](https://timeline.jerdog.me)



[IAmJerdog](https://twitter.com/IAmJerdog)



[jerdog](https://dev.to/jerdog)



[/in/jeremy.me](https://in.jeremy.me)



[@jerdog@hachyderm.io](mailto:@jerdog@hachyderm.io)