Grafana Labs

Embracing Disruption

Adding a Bit of Chaos to Help You Grow!

Paul Balogh
Developer Advocate, Grafana Labs
@javaducky

Overview

1. **Why are we here?**
2. A brief history of how we test
3. How fault injection can help us
4. Where do we go from here?

Application reliability is **hard**

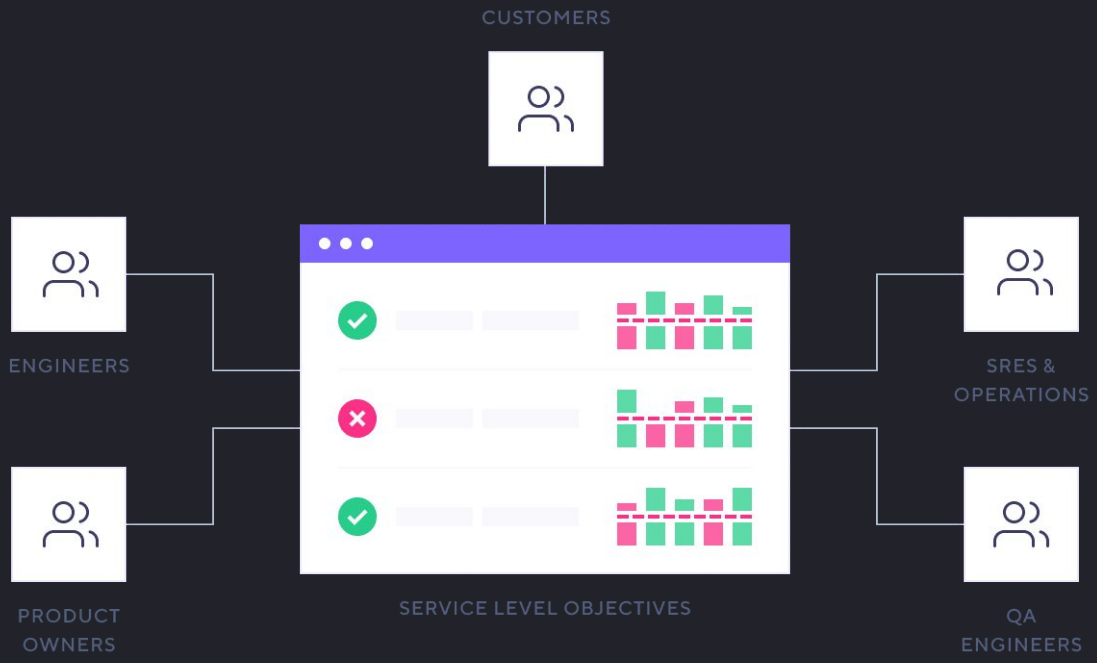
Complex architecture and infrastructure

Many potential points of failure

Inadequate tooling and practices

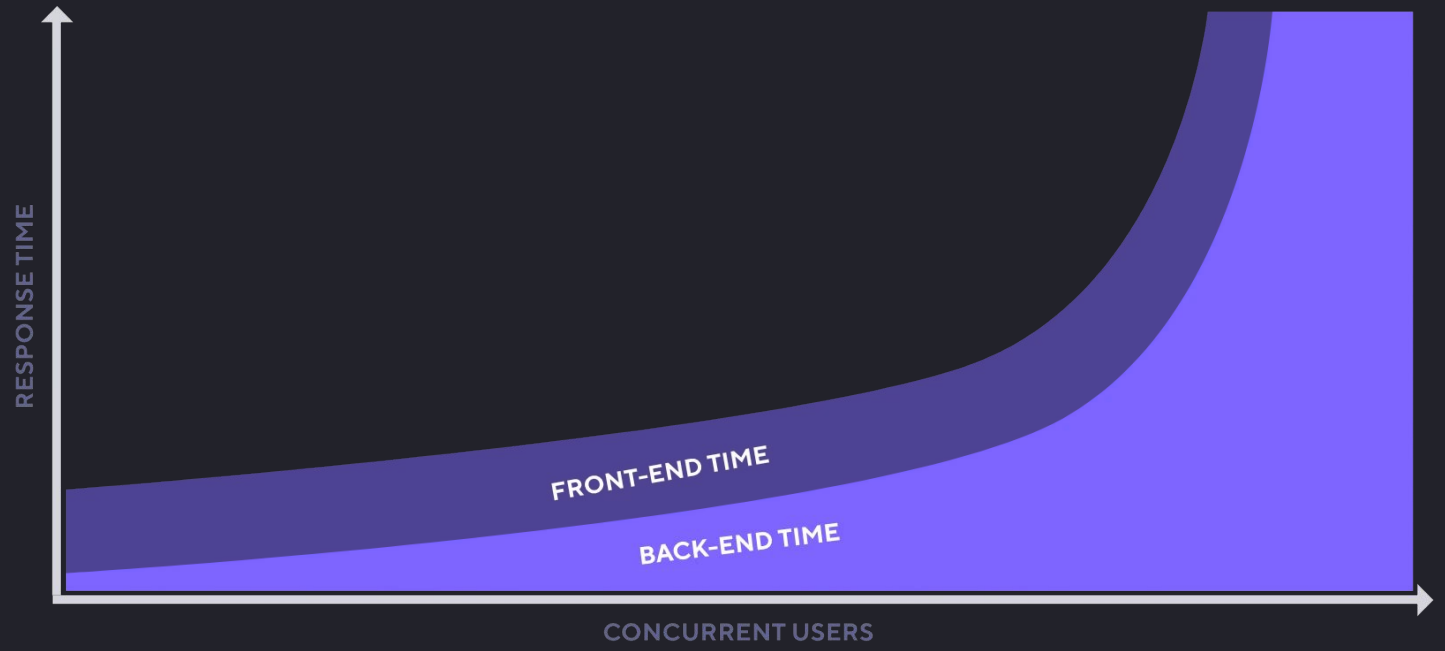
High demands on **availability**

SLOs



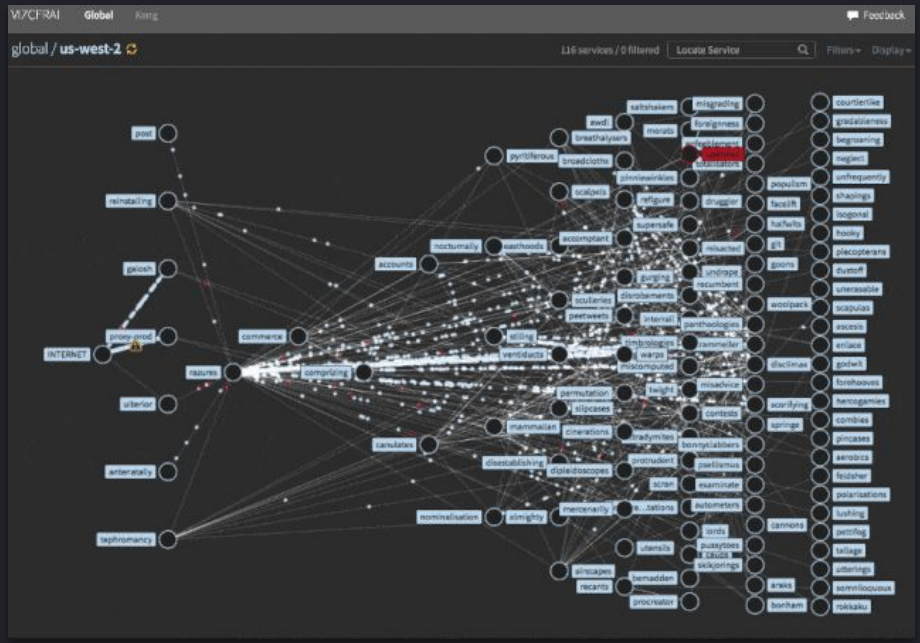
High demands on **usability**

UX



Ever increasing **complexity**

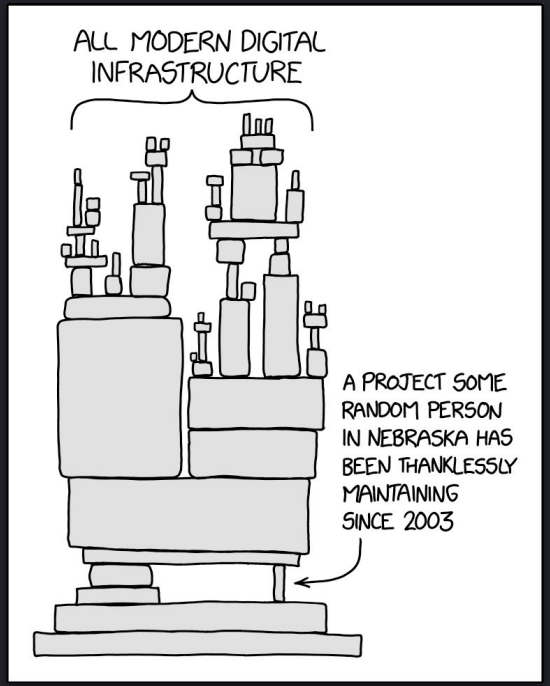
Distributed



(Example of Netflix services)

Potentially fateful **interdependency**

Fragility



https://xkcd.com/2347/

You are not alone

Overview

1. Why are we here?
2. **A brief history of how we test**
3. How fault injection can help us
4. Where do we go from here?

The way we test

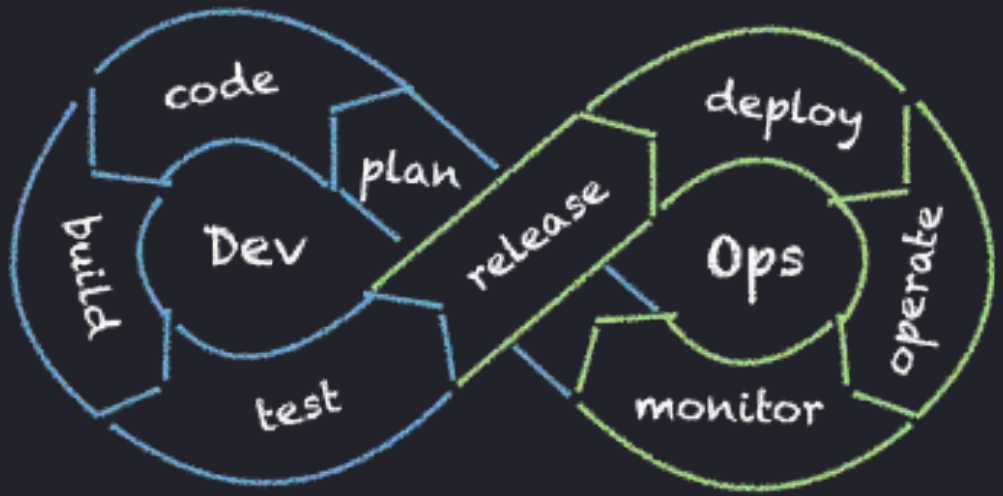
	Checklist / OLD WAY
Release frequency	Quarterly or biannually
Testing frequency	Before releases
How is initiated	Manually
Testing environment	Test and Production

- QA bottleneck
- Lower coverage
- Late in process

The way we test

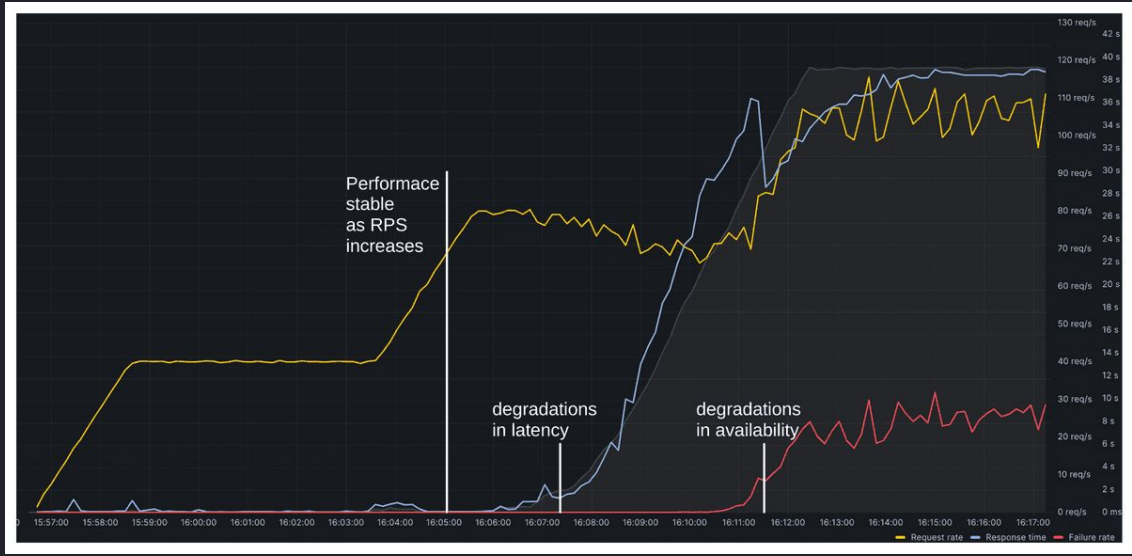
	Checklist / OLD WAY	DevOps / MODERN WAY
Release frequency	Quarterly or biannually	Weekly
Testing frequency	Before releases	AND nightly, feature branches, continuous, synthetic monitoring
How is initiated	Manually	Scheduled. Automatically as part of CI/CD
Testing environment	Test and Production	AND Staging, Long-lived and Short-lived ephemerals environment

The way we test



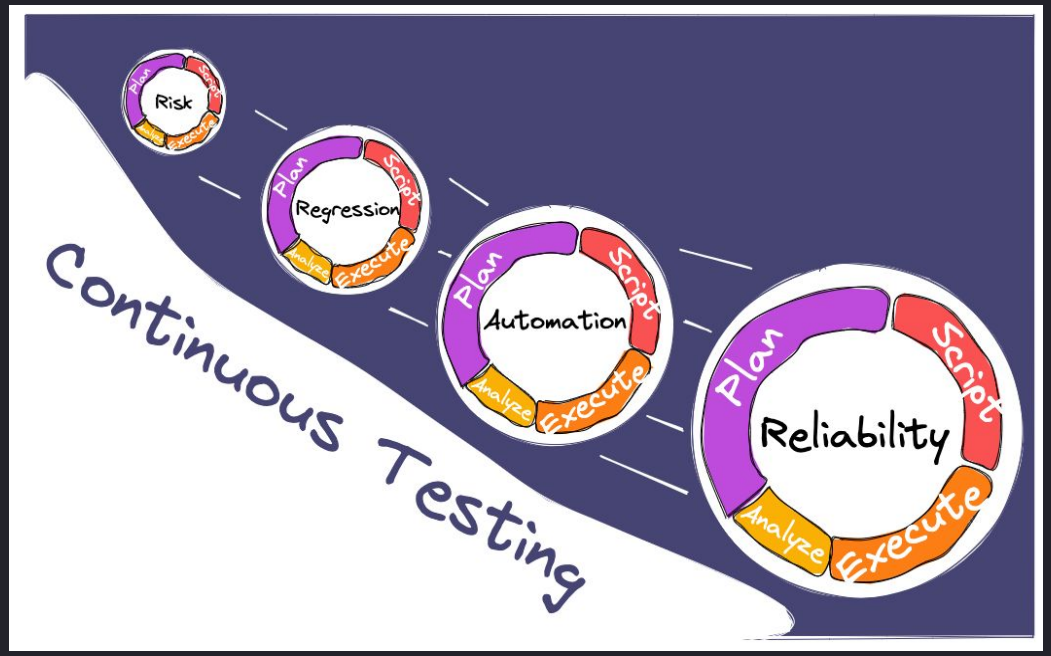
- Unit testing
- Integration testing
- Contract testing
- Functional testing
- E2E testing
- Load testing

The way we test



- Applications instrumented
- Observability platform available

The way we test



- Start simple
- Test frequently
- Continually expand
- Evolve over time

Yet we learn from failure

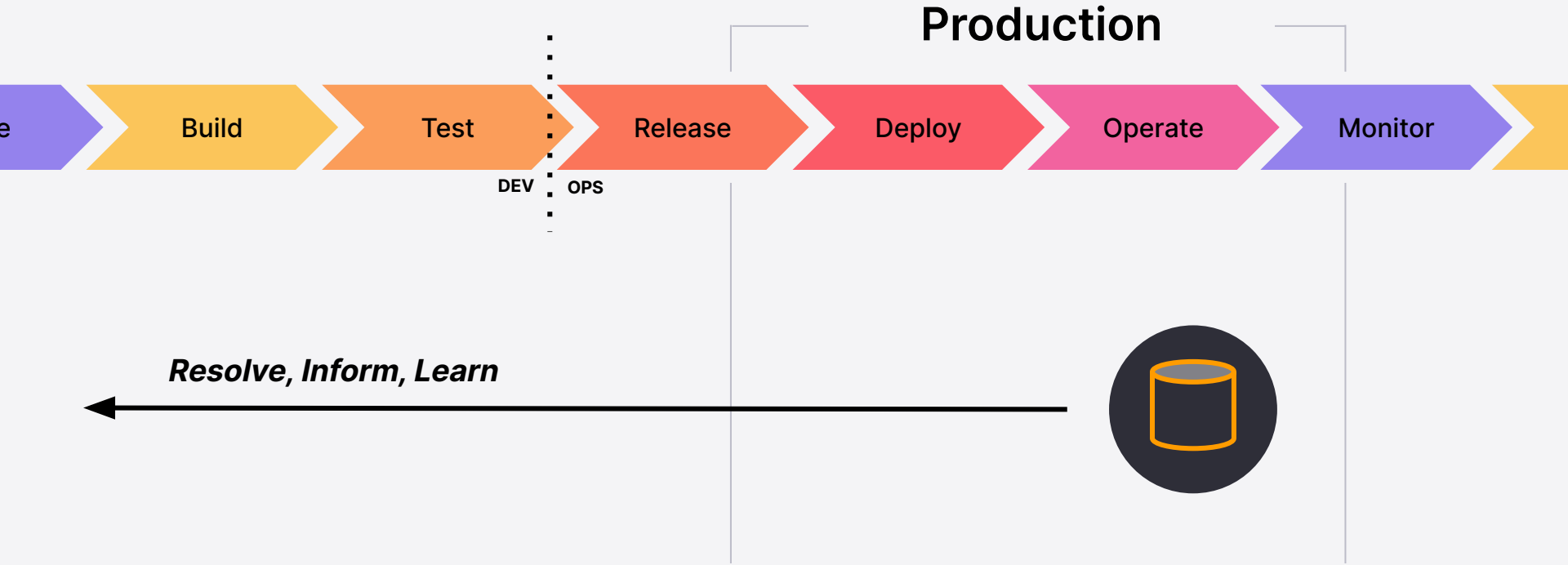
Overview

1. Why are we here?
2. A brief history of how we test
3. **How fault injection can help us**
4. Where do we go from here?

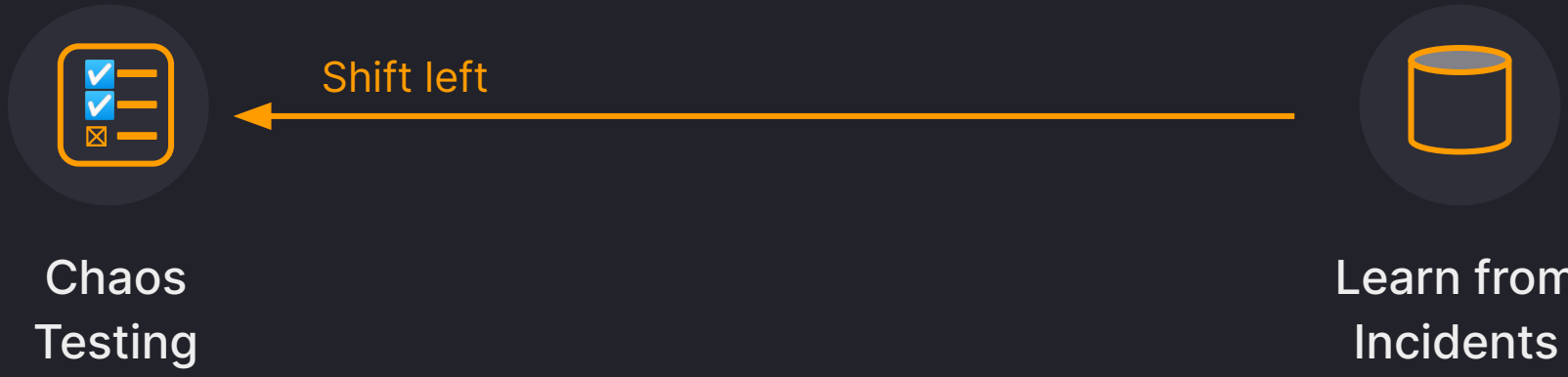
Fault Injection

A software testing technique which **introduces errors** to a system to **ensure** it can **withstand** and **recover** from those conditions.

Failure happens



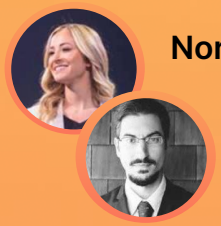
How organizations can build **more** confidence in their ability to withstand failures?



“”

From the distributed system perspective, almost all interesting availability experiments can be driven by affecting latency or response type.

- Chaos Engineering, O’Reilly



Nora Jones

Casey Rosenthal

Introducing **k6** and **xk6-disruptor**

k6, a *reliability testing* tool

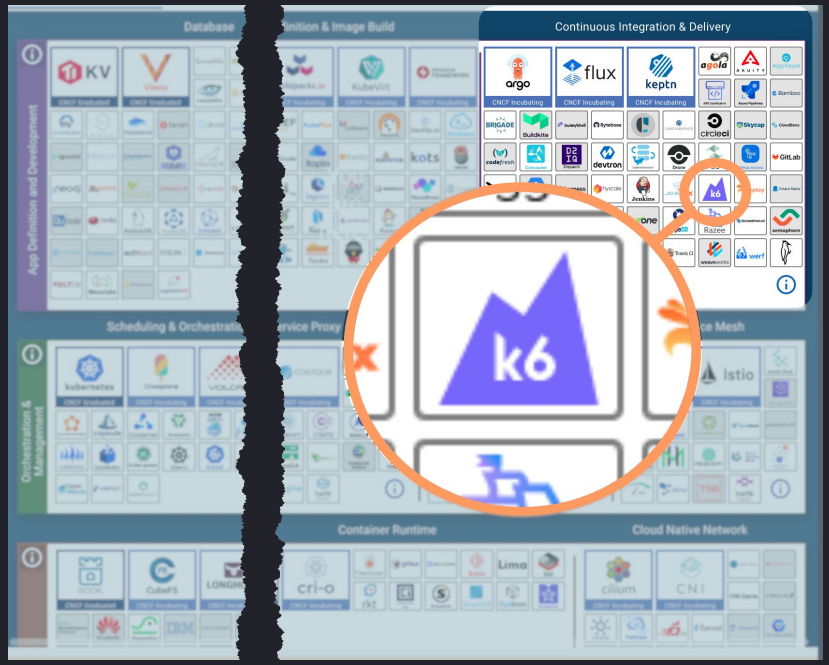
- Formerly known as *Load Impact*
- Open Source since 2016
- ~21.3k GitHub stars *(as of October 2023)*
- Promotes “shift-left” testing
- Acquired by Grafana Labs in 2021

github.com/grafana/k6

xk6-disruptor for *fault injection*

- Becomes a project in August 2022, evolved from previous experiments

github.com/grafana/xk6-disruptor



k6: a reliability testing tool

OpenSource

OSS is at the heart of **what we do** and helps leave the world a little better than we found it

Scriptable

CLI and API **designed for automating** your tests with pass/fail criteria using JavaScript syntax

Performant

A k6 engine **written in Go** making it one of the the best performing tools available

Extensible

Use Go(lang) code to **add support** for new outputs, protocols, and products from within your test scripts

How effective is testing known errors?

According to a study of failures in real world distributed systems:

92% of the catastrophic system failures are the result of incorrect handling of non-fatal errors

In 58% of the cases the resulting faults could have been detected through simple testing of error handling code

Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems, Yuan et al. USENIX OSDI 2014

How hard it to improve?

In 35% of the cases, the error handling code fall into one of three patterns:

Overreacted, aborting the system under non-fatal errors

Was empty or only contained a log printing statement

Contained expressions like “FIXME” or “TODO” in the comments.

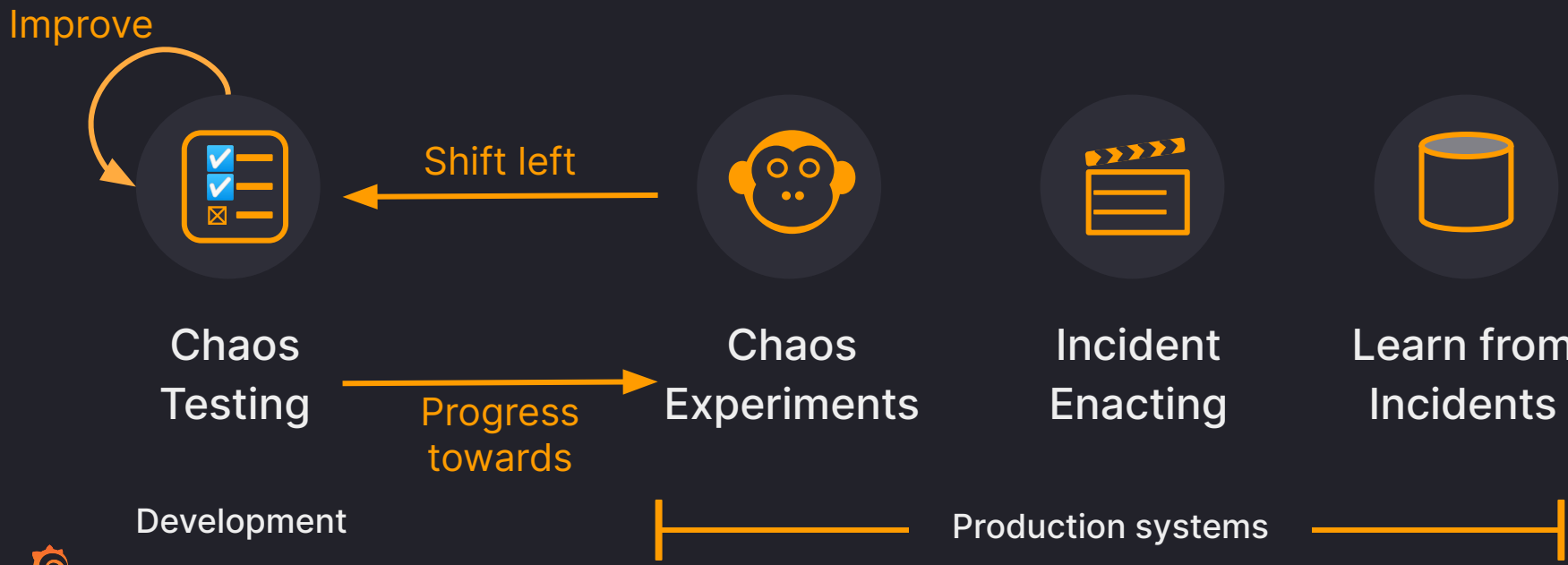
Chaos Testing

Incorporate the principles of chaos engineering early into the development process as an integral part of the testing practices

Shifting the emphasis from experimentation to verification

From uncovering unknowns faults to ensuring proper handling of known faults

Continuous Reliability Improvement



The four tenets of Chaos Testing

Incremental adoption

Application Centric

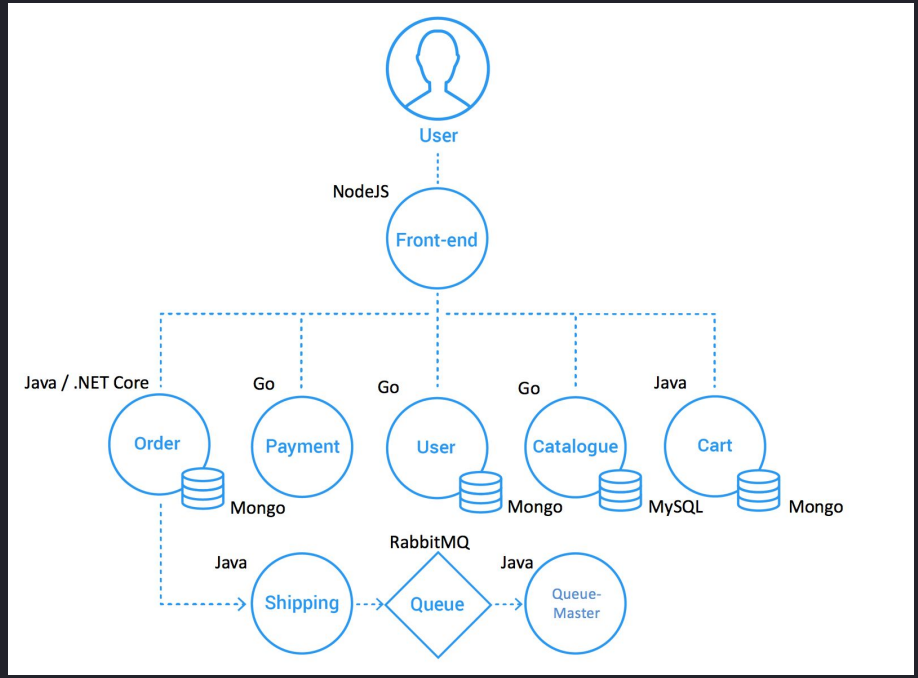
Chaos as Code

Controlled Chaos

Chaos Testing in action

Sock Shop application

- Microservices architecture
- Http-based communication between services
- Polyglot (Go, Java, JS, ...)
- K8s-ready



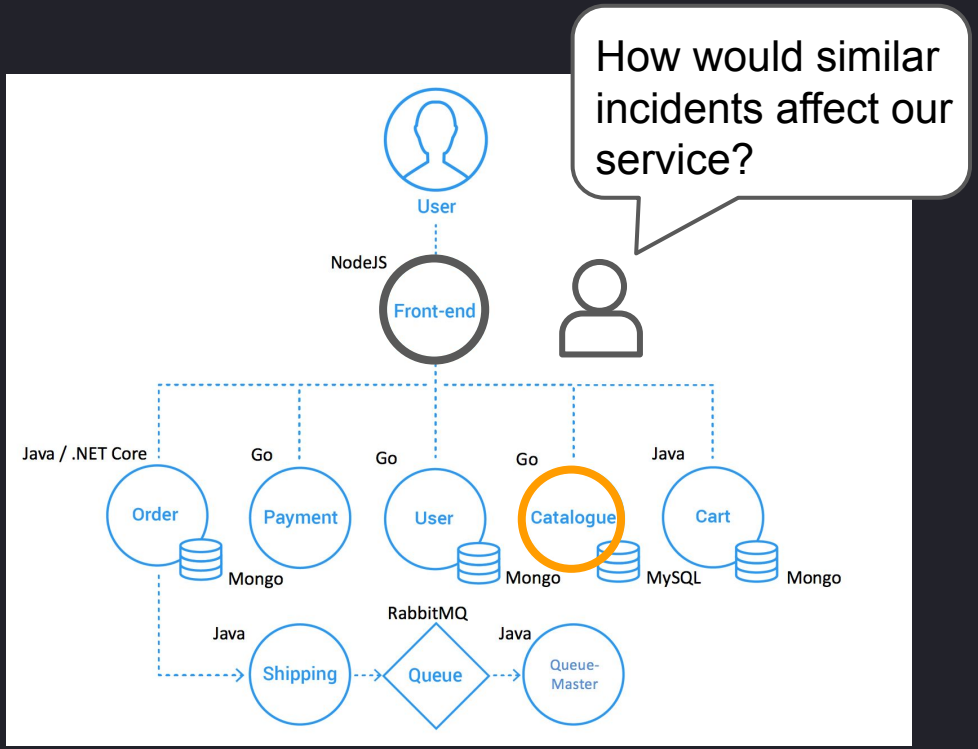
https://github.com/microservices-demo/microservices-demo

Fictitious Post Incident Review

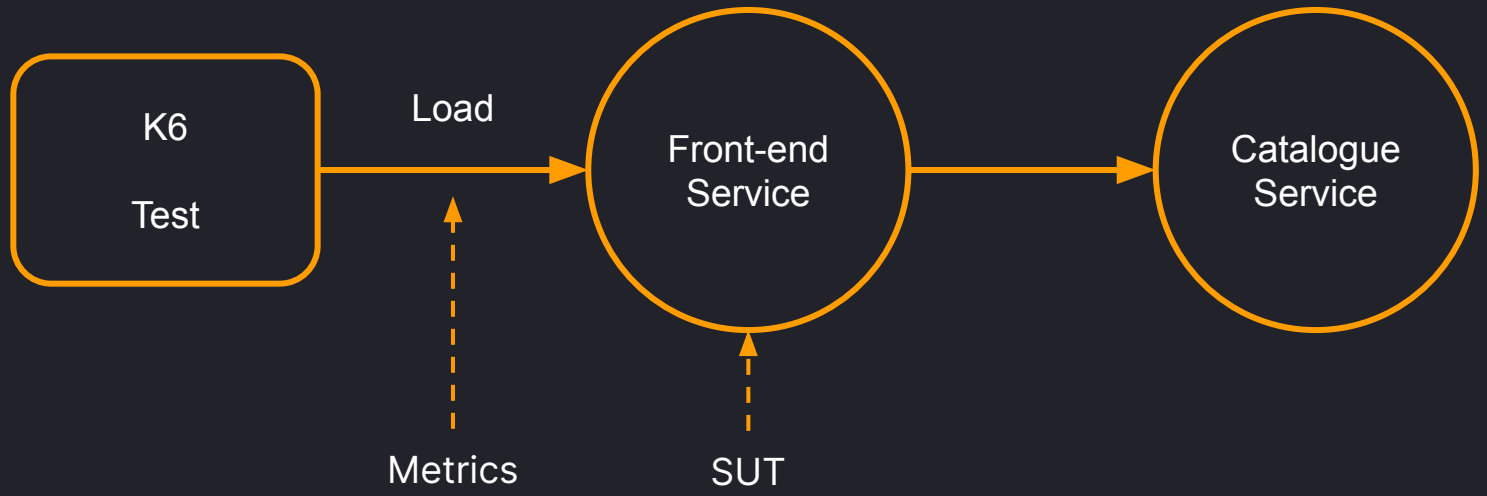
Long running queries in the **Catalogue service**'s DB caused **delays** in the requests the exhaustion of DB sessions that resulted in **errors** (HTTP 500)

Catalogue service team will investigate the incident to address the root cause

However, the front-end team wonders...



Let’s start with a load test for the Front-end service



Key concepts

Functions

Simulated user flow

Check

Response Validations

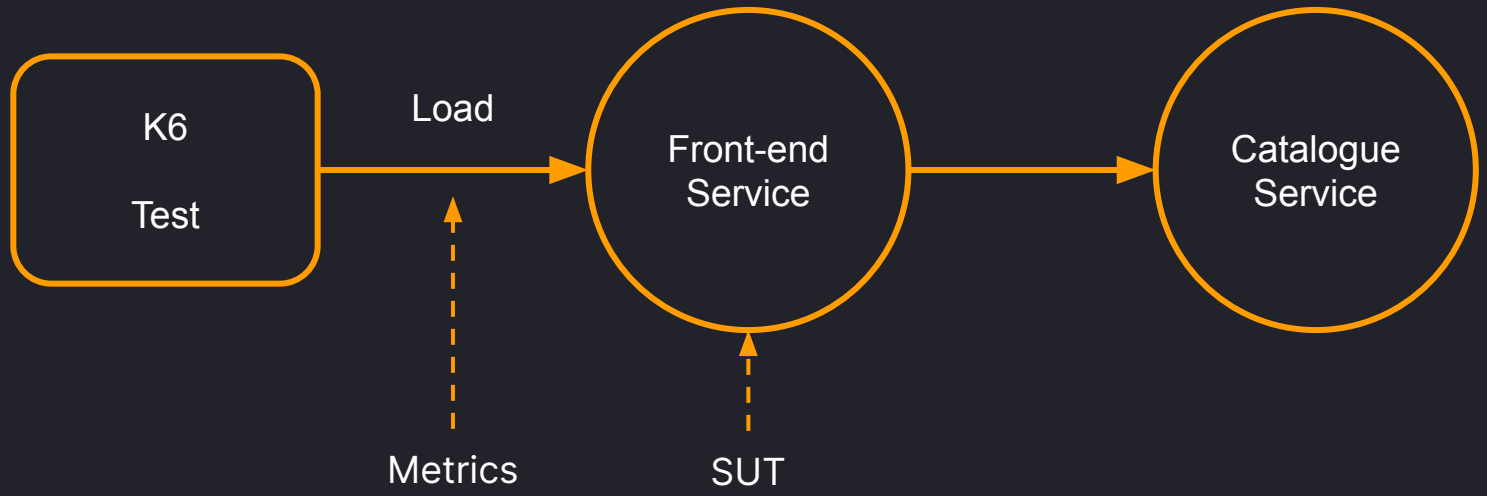
Scenario

Workload model

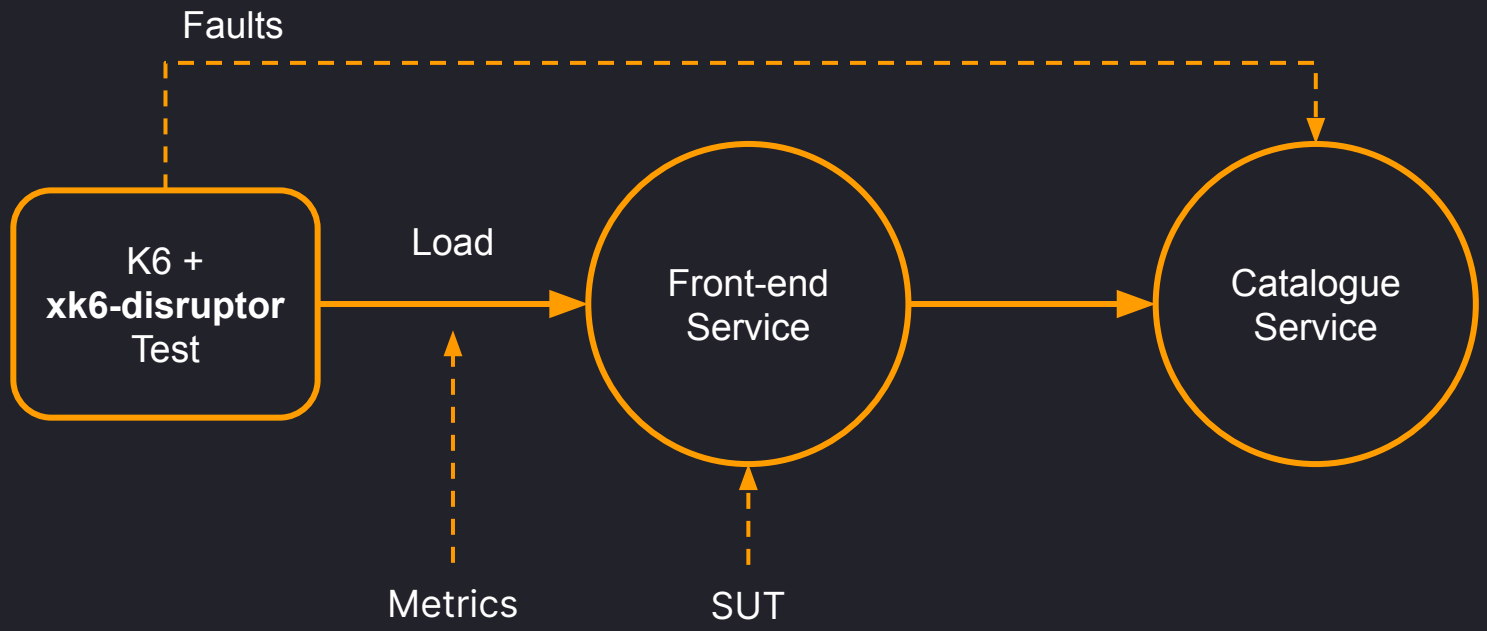
Thresholds

SLOs

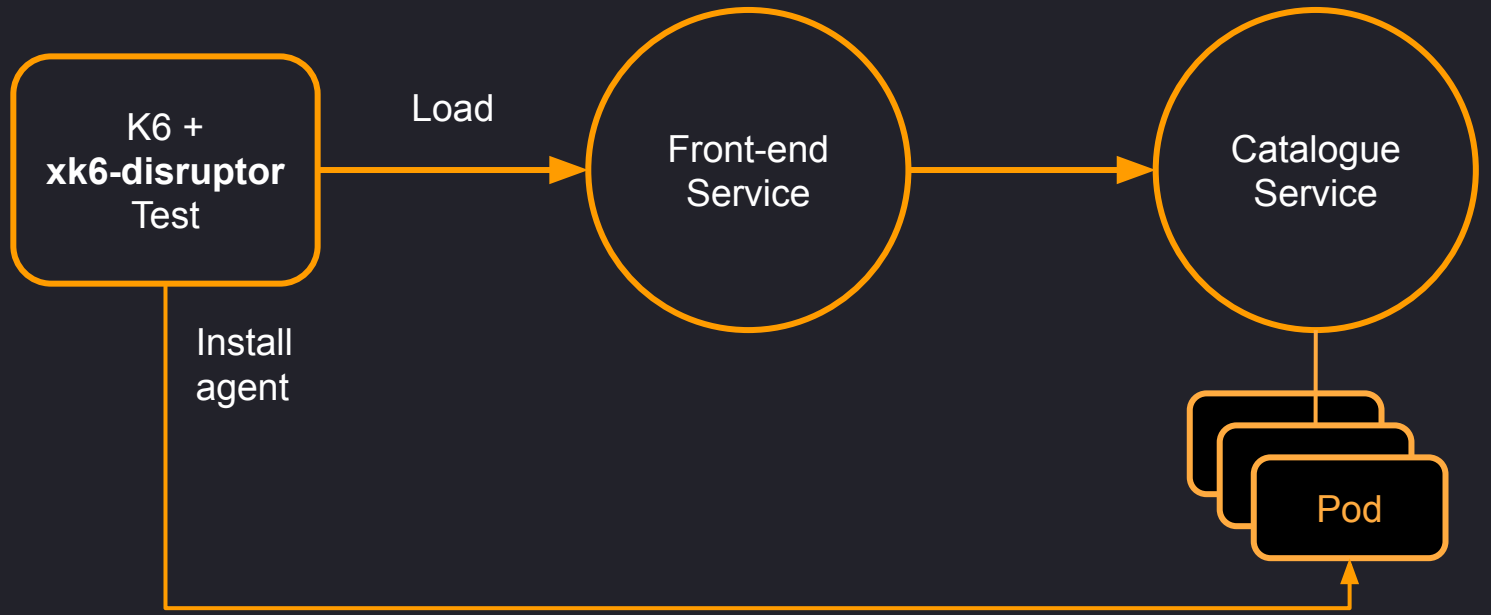
Now, let’s add some chaos to this test



Let’s add some chaos to this test



How the disruptor works



Demo!

How this test helps the front-end team?

Uncover improper error handling logic

Validate different solutions until obtaining an acceptable error rate

Fine-tune the solution and avoid issues such as retry storms

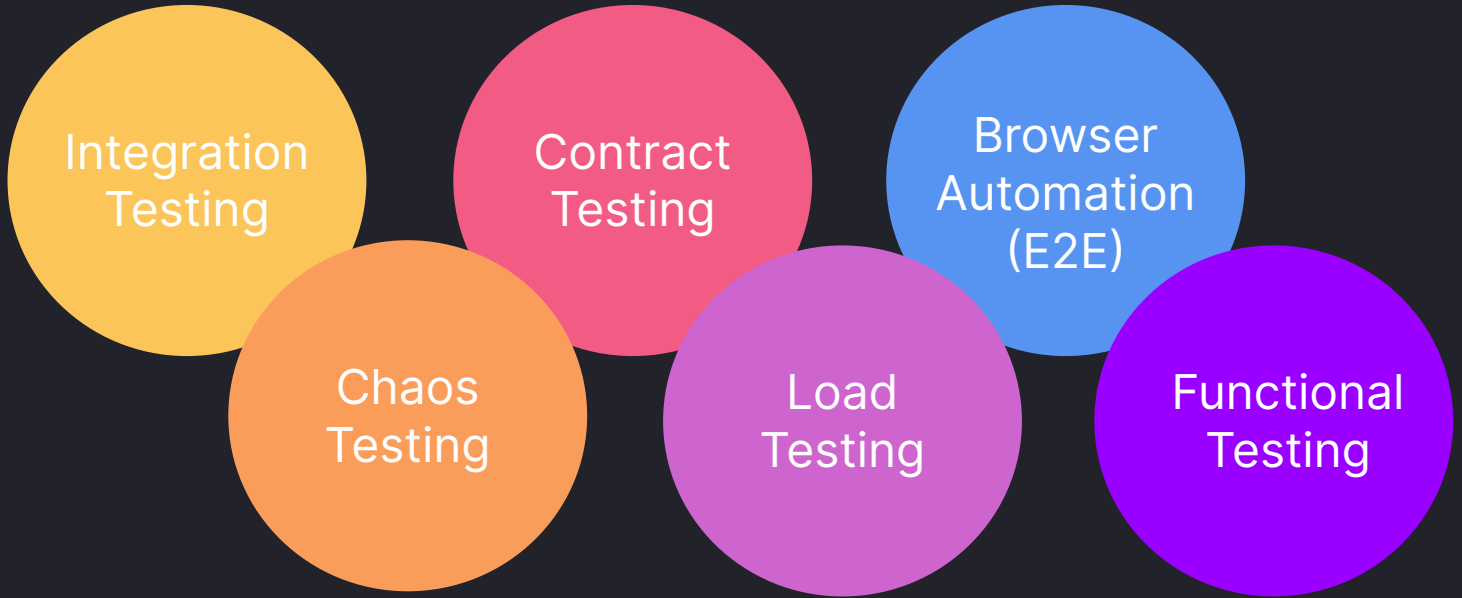
Chaos testing principles in action

- A load or functional test can be reused to test the system under turbulent conditions
- These conditions are defined in terms that are familiar to developers: latency and error rate
- The test has a controlled effect on the target service
- The test is repeatable and the results are predictable
- The fault injection is coordinated from the test code
- The fault injection does not add any operational complexity

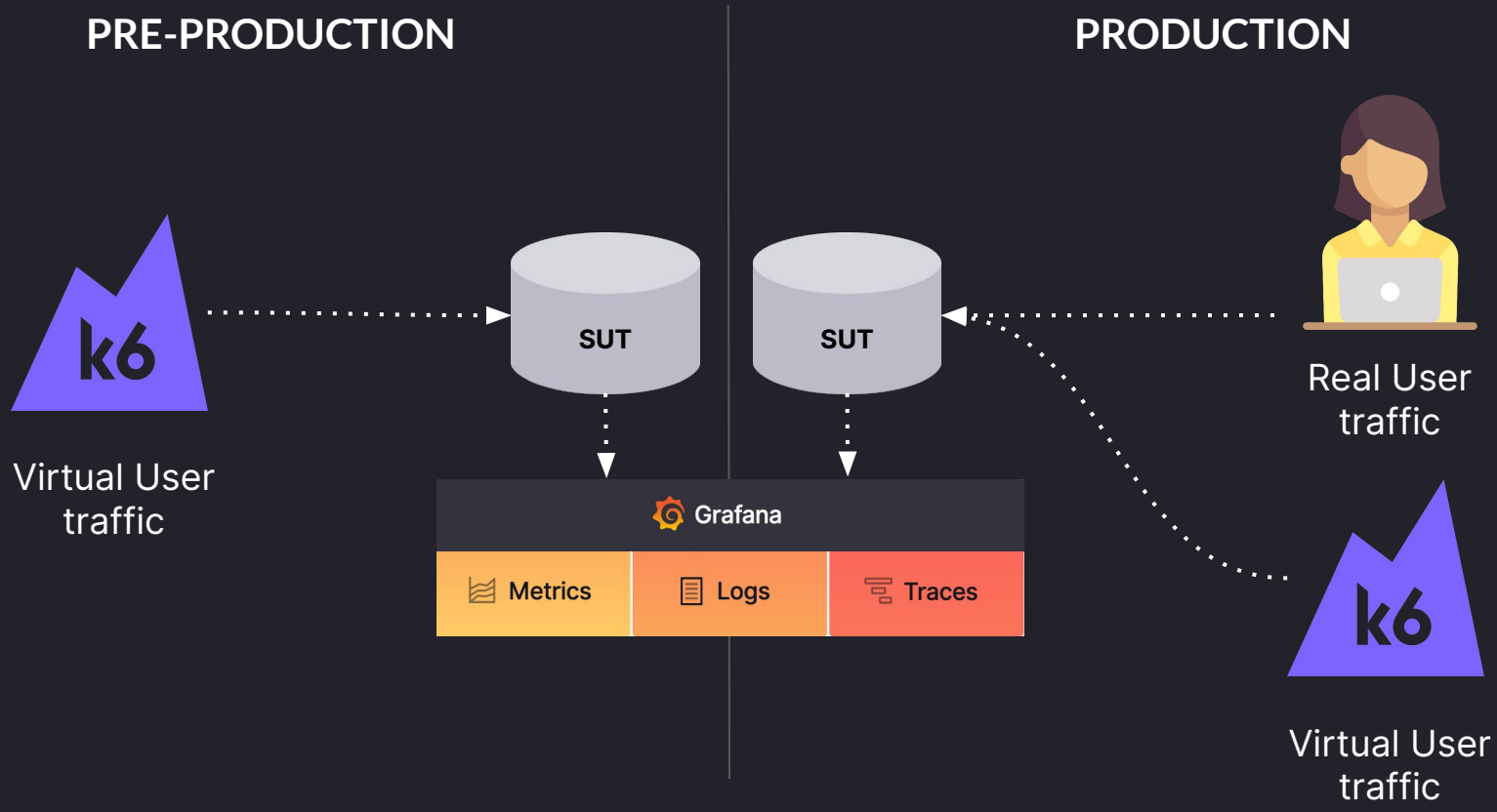
Overview

1. Why are we here?
2. A brief history of how we test
3. How fault injection can help us
4. **Where do we go from here?**

Reliability testing strategy



Proactively improve reliability



Final remarks

The ability to operate reliably should not be a privilege of the technology elite

Chaos Engineering can be democratized by promoting the adoption of Chaos Testing

To be effective, Chaos Testing must be compatible with the existing testing practices used by development teams

Our Goal

Make Chaos Engineering practices accessible to a broad spectrum of organizations by building a solid foundation from which they can progress towards more reliable applications.

Grafana Labs

Thanks for participating!

Connect with Paul as
@javaducky or linkedin/in/pabalogh

k6.io/slack

grafana/xk6-disruptor