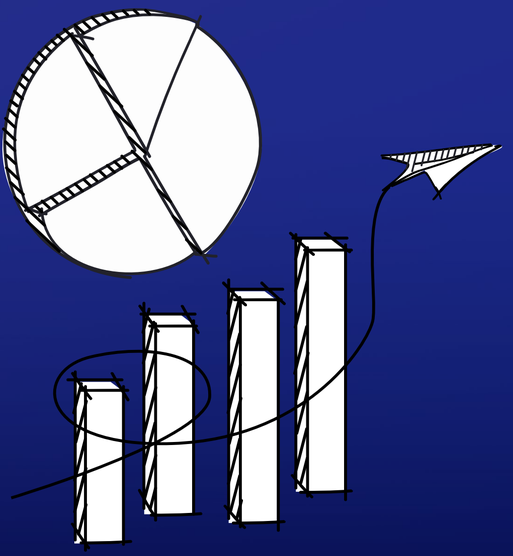
Ebook

The Chaos Engineering Maturity Model

Four Levels to Improving Software Reliability Through Chaos Engineering



harness

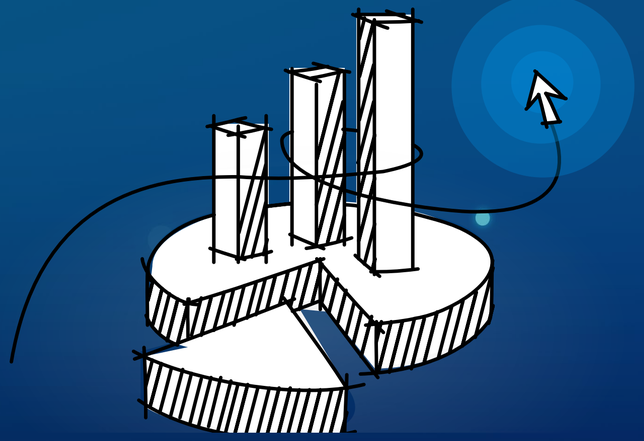
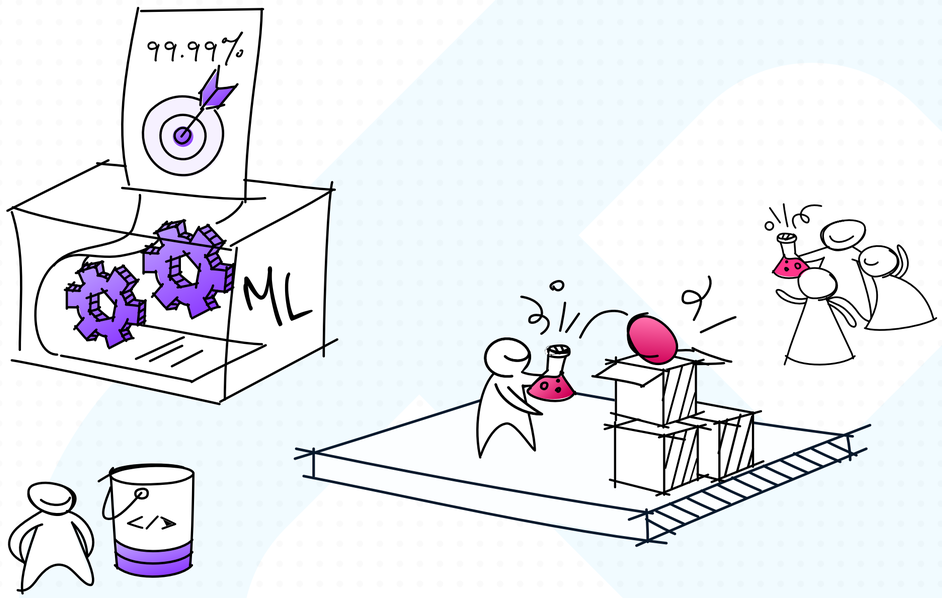


Table of Contents



Foreword

The inspiration for the Chaos Engineering Maturity Model came from the past ten years I've spent working with thousands of developers who are interested in improving the reliability of their software, understanding their complex systems, and minimizing the impact of unplanned downtime. The problem that I kept hearing was that developers didn't know where to get started, and there was no urgency from their leadership to implement.

Hearing this reactive nature of responding to incidents and an ever growing backlog of issues that were never prioritized led me to develop a simple maturity model. The intent was to enable an engineering team to get started, understand the journey it takes to evolve, and most importantly, communicate to leadership and the business.

My hope is that reading this will enable you to start your journey and move through the stages of maturity, so you can learn, improve, and build a culture of reliability.

Uma Mukkara

Head of Chaos Engineering at Harness

Uma Mukkara is the head of Chaos Engineering at Harness. Previously, Uma was a co-founder of ChaosNative and MayaData, both of which he helped lead to successful acquisitions. He also co-created the popular CNCF open source projects openEBS and LitmusChaos, which he continues to actively maintain. Uma speaks regularly about chaos engineering and cloud native DevOps topics at various industry events, as well as meetups and conferences related to site reliability engineering. He is passionate about building solutions around resilience through chaos testing in the cloud native space. Uma holds a master's degree in telecommunications and software engineering from the Illinois Institute of Technology, Chicago.

Introduction

Businesses are increasingly adopting cloud native deployments as a means to increase developer velocity. The CNCF 2021 annual survey stated, “Kubernetes has crossed the adoption chasm to become a mainstream global technology.” According to CNCF’s respondents, 96% of organizations are either using or evaluating Kubernetes. This rapid adoption of Kubernetes has created significant complexity and revealed the inadequacy of traditional systems testing.

Chaos engineering has emerged as a new testing discipline and a means to transform the reliability of cloud native services. According to Gartner, “40% of organizations will implement chaos engineering practices as part of DevOps initiatives by 2023, reducing unplanned downtime by 20%.”

While there is a high level of interest – and a growing need – for chaos engineering as a systems-level testing method in a Kubernetes world, many solutions for this need remain nascent. Many organizations that are considered early adopters of chaos engineering are manually running experiments on a few applications in a pre-production environment. Very few organizations are automating this practice throughout the entire software delivery lifecycle (SDLC) due to complexity and the lack of industry maturity in the practice.

In this ebook, we will introduce the Chaos Engineering Maturity Model and how Harness Chaos Engineering (CE) can help organizations understand their journey to improve software reliability.

According to Gartner, “**40% of organizations** will implement chaos engineering practices as part of DevOps initiatives by 2023, **reducing unplanned downtime by 20%**.”

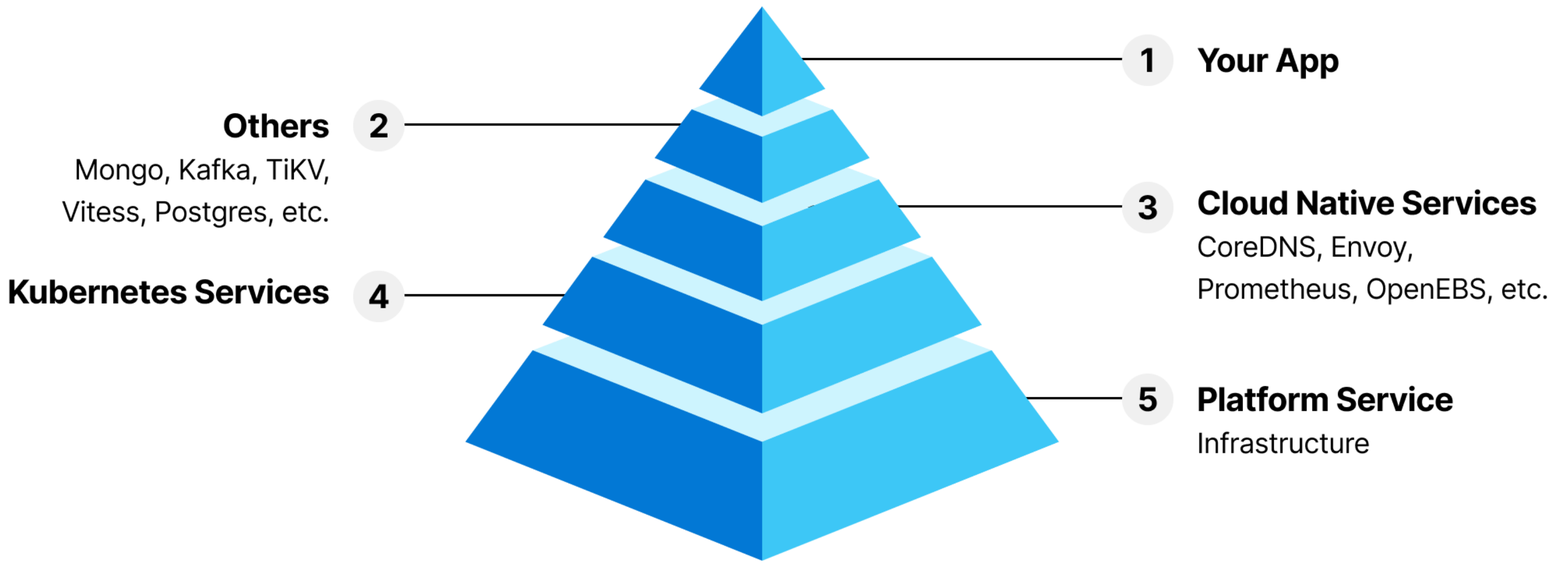
Current State of Chaos Engineering

In 2010, when Netflix introduced Chaos Monkey, the first chaos engineering tool, the teams focused on ensuring applications could withstand the impact of a single server failure. This exercise helped developers ensure that proper monitoring and alerting were in place, and it also helped them build systems that were redundant and did not impact the business when a dependent service failed.

Since then, open source tools and a few commercial tools have introduced chaos engineering capabilities that enabled organizations to adopt the practice, but few have been able to introduce this practice into the entire software delivery lifecycle (SDLC) like Netflix was able to achieve. Most of the tooling today focuses on periodic events called 'GameDays,' which only give the developer a point-in-time reliability snapshot and understanding of their system.

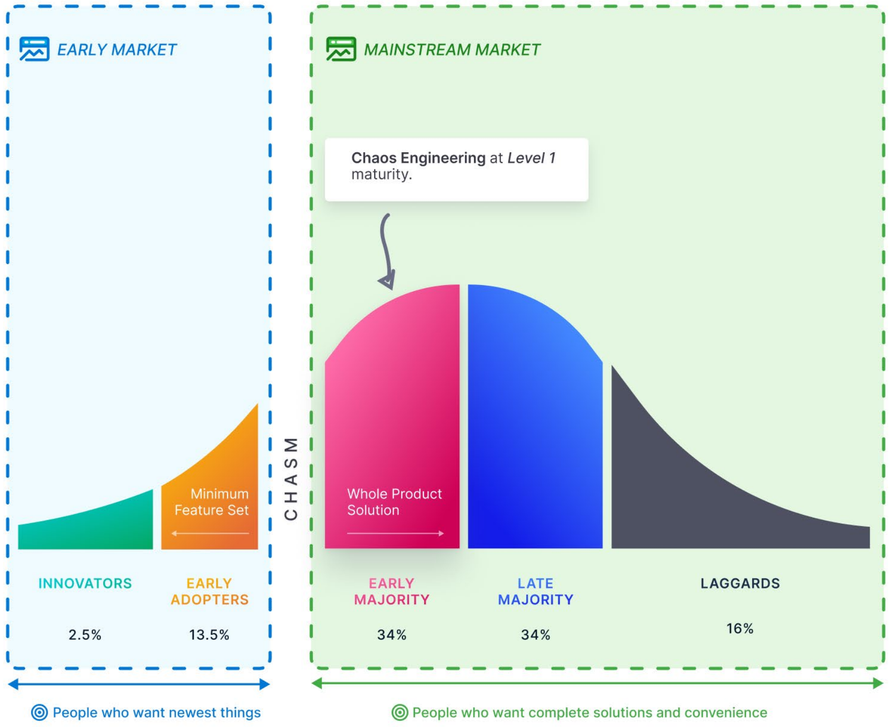
Most of the tooling today focuses on periodic events called **'GameDays,'** which only give the developer a point-in-time reliability snapshot and understanding of their system.

The overall system view of a developer has shifted from understanding the entire monolithic application and technology stack to understanding their application running on a container. Developers often have minimal understanding of how their application behaves in response to the faults happening on the infrastructure that support it, leading to incidents that cause system downtime. The impact of this shift, along with microservice proliferation, is causing DevOps organizations to adopt chaos engineering to ensure reliability throughout software development.



The cloud native developer focuses on what runs inside the application container. The pyramid illustrates the dependency stack widening for each dependent service that the application runs on.

“Crossing the Chasm” by Geoffrey Moore describes five phases of software market adoption: innovations, early adopters, early majority, late majority, and the laggards. Applying this framework to trends in chaos engineering adoption, chaos engineering tools have primarily been associated with the early market of innovators and early adopters. With the onset of more teams adopting this practice throughout the SDLC, these tools have crossed the chasm and have entered into the mainstream market. Chaos engineering has become a team sport that benefits the entire business.



Geoffrey Moore’s “Crossing the Chasm” shows the five stages of software market adoption.

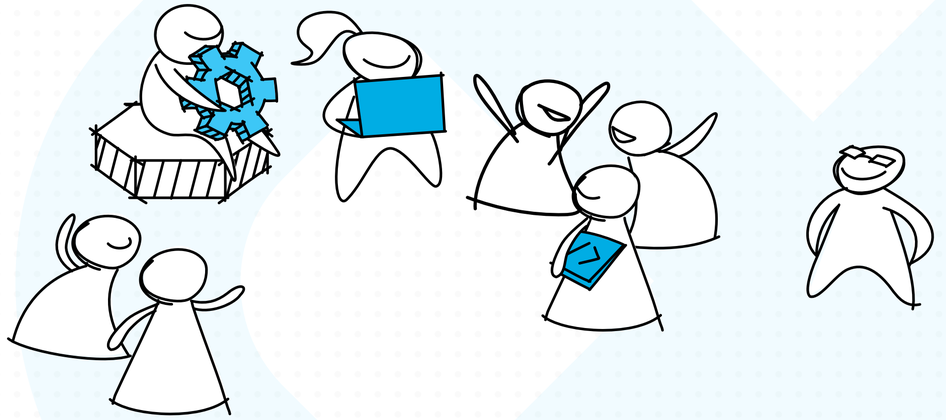
Key Roles Involved in Chaos Engineering

Typically, use of chaos engineering tools as part of software delivery and operations has been associated with site reliability engineers (SREs) since they are the operations part of DevOps. However, with increasing adoption of modern continuous integration (CI) and continuous delivery (CD) pipelines, we see new emerging patterns involving developers, quality assurance (QA) teams, and SREs in the earliest stages of the SDLC.

In theory, DevOps as a practice is intended to be an overarching concept applying to teams that own coding, QA testing, load testing, bug fixing, and monitoring. In reality, this isn't how DevOps has been implemented across the thousands of companies delivering software today. Developers, QA engineers, and SREs can now make reliability improvements – such as finding and fixing an issue that otherwise would have gotten released to a customer – by verifying the resilience of the components and individual services together in a pre-production environment.

Let's breakdown how various roles in the engineering organization benefit from chaos engineering in different ways.

Bear in mind that as an organization grows in chaos engineering, it's important for all of these roles to collaborate and work together to mature the practice.



Site Reliability Engineers

Site Reliability Engineers (SREs) are primarily responsible for ensuring the reliability of services in production, so naturally, chaos engineering directly applies to them. SREs can introduce chaos engineering to the organization and start building the necessary chaos culture. SREs also contribute to the maintenance of the service map, which is the backbone of a successful chaos engineering practice. The service map is the list of services in production, along with service owners, associated criticality levels and dependencies. SREs also ensure visibility of the chaos experiment by implementing the required metrics to monitor or observe the system. SREs can influence other stakeholders to adopt chaos engineering to promote the continuous resilience verification of the code and services, which avoids the overall cost of finding and fixing a weakness in the production system.

Quality Assurance (QA)

Quality Assurance (QA) Team members are finding added responsibilities around reliability, especially in cloud native engineering. The cloud native environment has become so dynamic that there is a good chance many critical experiments related to reliability are unverified as the code moves to production. The CD pipelines play an important role in introducing automated chaos tests, whether they're developed by the QA teams or the SREs as part of the larger initiative around chaos engineering in the organization. QA teams can also influence developers by providing simple, meaningful chaos tests that can be included in the CI pipelines to help automate chaos engineering in cloud native environments.

Cloud-native Developers

Cloud-native Developers are actively participating in DevOps initiatives to improve quality by adding increasingly more tests in the CI pipelines. Developers like to be involved with the process of testing the code either before or after it's merged. The "chaos first" approach is leading to the advent of chaos-driven development, just like test-driven development has become a common practice. With this approach, developers write the conditions in the code to address the failures of external components in the deployment stack, API failures, resource constraints, and network delays. These tests not only help to immediately improve the code quality, but they also help QA teams get an idea of where to start the chaos experiment development process. In an efficient DevOps culture, chaos tests are a shared responsibility between Developers, QA, and SREs.

DevOps Management

DevOps Management plays a critical role as chaos engineering has become an acceptable approach to improving the resilience of services due to its improved role in pre-production, QA, and development—not to mention the already proven production environment. One of the reasons that this change has happened is because the management stakeholders in DevOps, either engineering managers, DevOps managers, QA managers, or SRE managers, have understood the benefits of chaos testing in DevOps. Chaos engineering results are always expected in the medium term, not the short term. Someone in DevOps management must be a champion of this perspective to drive approval of budgets and processes. DevOps managers also understand the bigger picture of chaos engineering's benefits, which allows them to drive the resource allocation, including setup of reliability metrics in observability systems.

Roadmap to Chaos Engineering Maturity

When setting out to improve software reliability with chaos engineering, there are various resources needed to create a successful outcome. While not all of these resources are required immediately, they are required to achieve a mature chaos engineering practice in each environment.

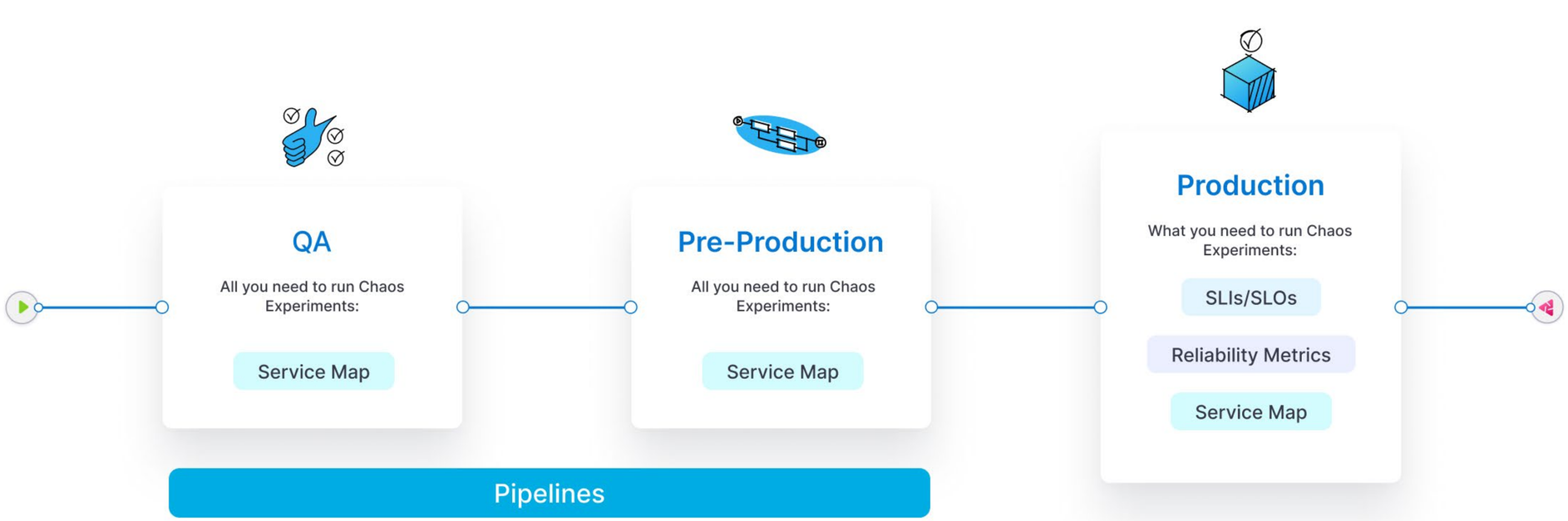


Diagram of resources needed to start a chaos engineering practice

To start chaos engineering in a QA and pre-production environment, you need to have a service architecture diagram and an understanding of how your service is deployed. The diagram depicting the details of various service components is called a service map.

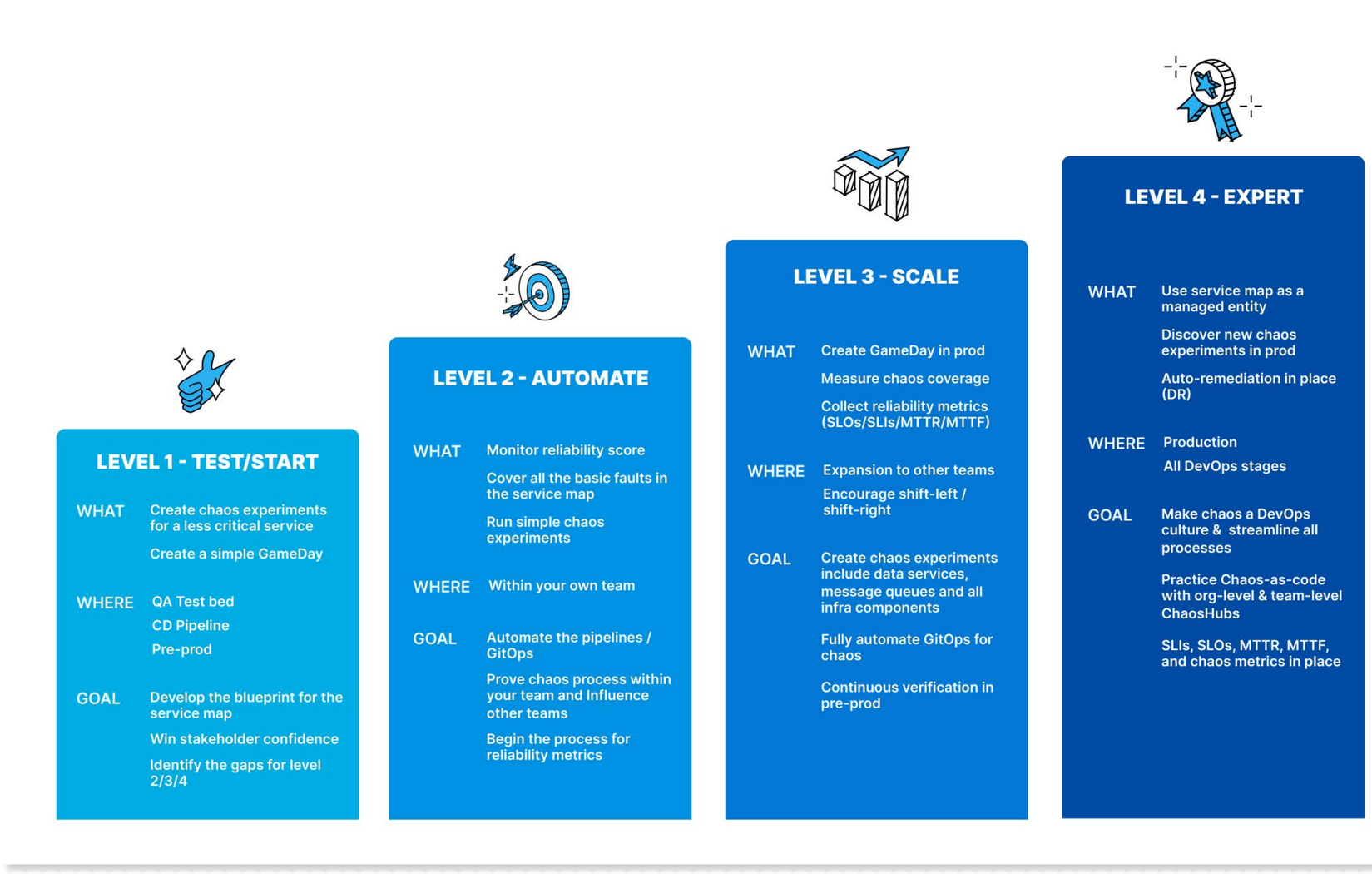
Best practices for a service map include a listing of the criticality of its components, incident history, the code or binary components, and an understanding of the underlying infrastructure components with associated dependencies. In its simplest form, the service map needs to outline the tech stack including databases, cache, message brokers, and dependencies. This enables the team to understand the architecture of the system and what chaos experiments should be tested.

Before running chaos experiments in production, it's important to have the necessary reliability metrics in place to monitor the impact of the experiment. Often, these metrics are validated and refined in the pre-production environment, which enables safer controls of chaos experiments in production environments. Standard reliability metrics, such as mean time to failure (MTTF) and mean time to recovery (MTTR), enable the organization to understand the impact and duration of the failure and recovery from the failure. Concerning the service being tested, service reliability metrics, such as service level indicators (SLIs) and service level objectives (SLOs) enable the user to closely monitor the health metrics of the service so that testing can be safely aborted before impacting the service level agreements (SLAs) with the customer.

A sample service map based on Google's Online Boutique sample application.

The Chaos Engineering Maturity Model

Chaos engineering maturity can be categorized into four levels for any organization. Attaining a certain chaos maturity level requires achieving specific goals by different roles within the organization.

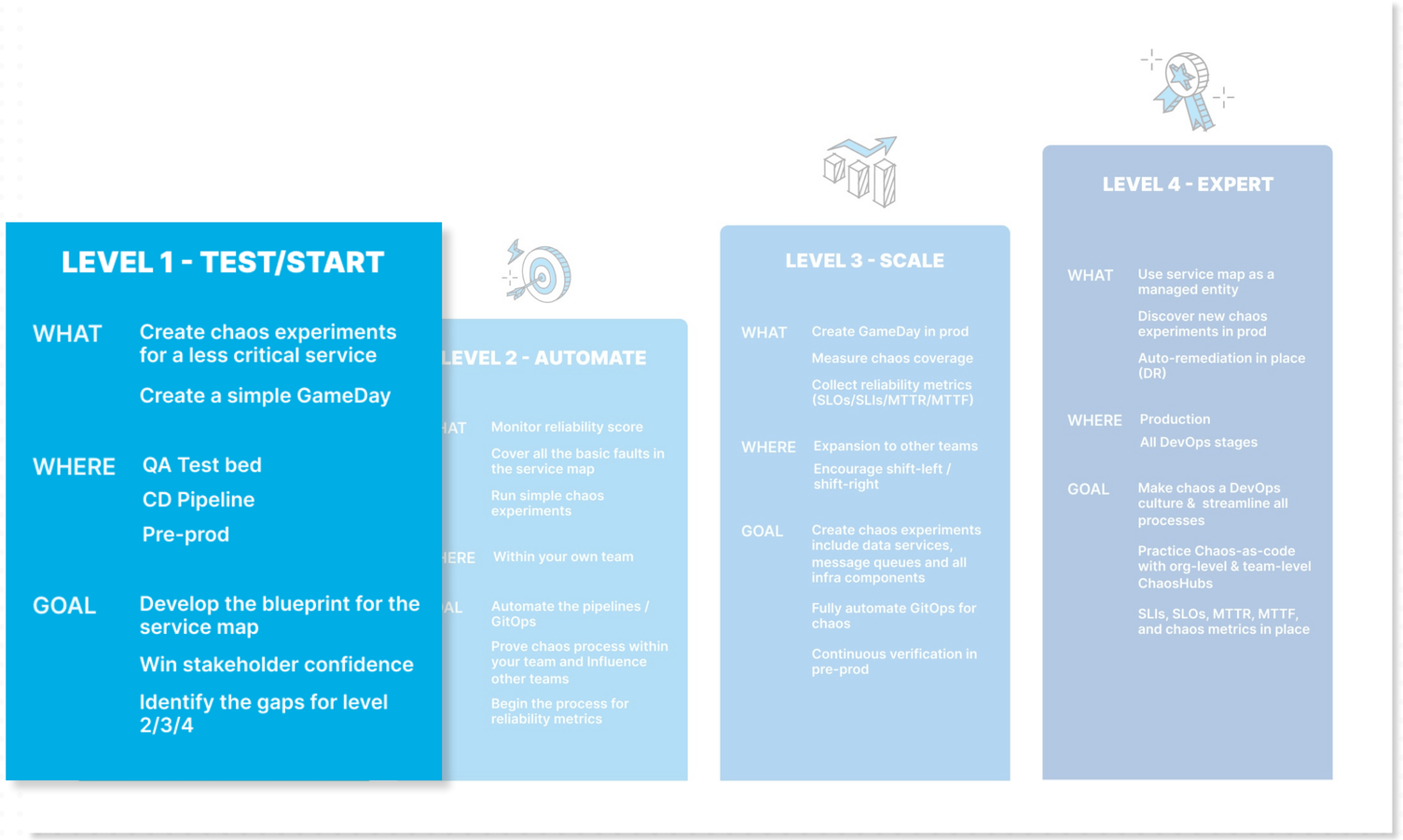


Level 1: Test/Start

Organizations can start their chaos engineering practices by identifying one service that they can run a few experiments on in a QA pipeline.

The team can develop a service map, gain confidence from stakeholders, and identify issues that would have made it into production had it not been for testing.

This level of effort proves that chaos engineering can be adopted successfully by one team in two to three months and also allows the team to start planning for more test automation across the organization.

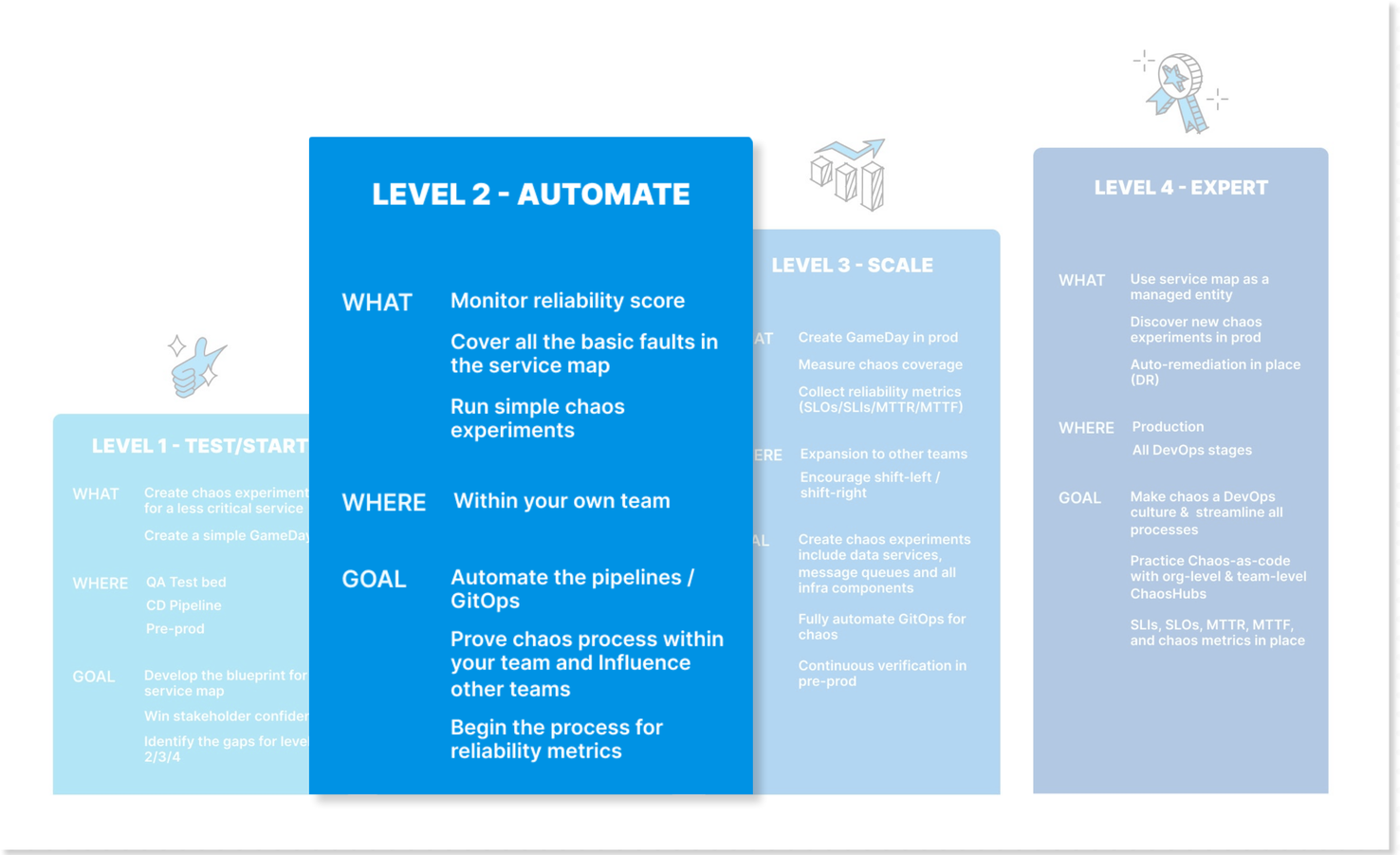


Level 2: Automate

Test automation is key to growing the adoption of chaos engineering within an organization. The goal for Level 2 is to automate simple chaos experiments for the team in the CD pipeline and start reporting metrics on service health and resiliency scores to generate reports for stakeholders to measure the initial reliability of a service.

Within three to six months after Level 1, a team should achieve these results and be able to demonstrate repeatable results and reliability improvements through:

- Basic system fault testing in the service map
- Proven automation in the pre-production pipeline
- Introducing chaos GitOps to automate infrastructure changes



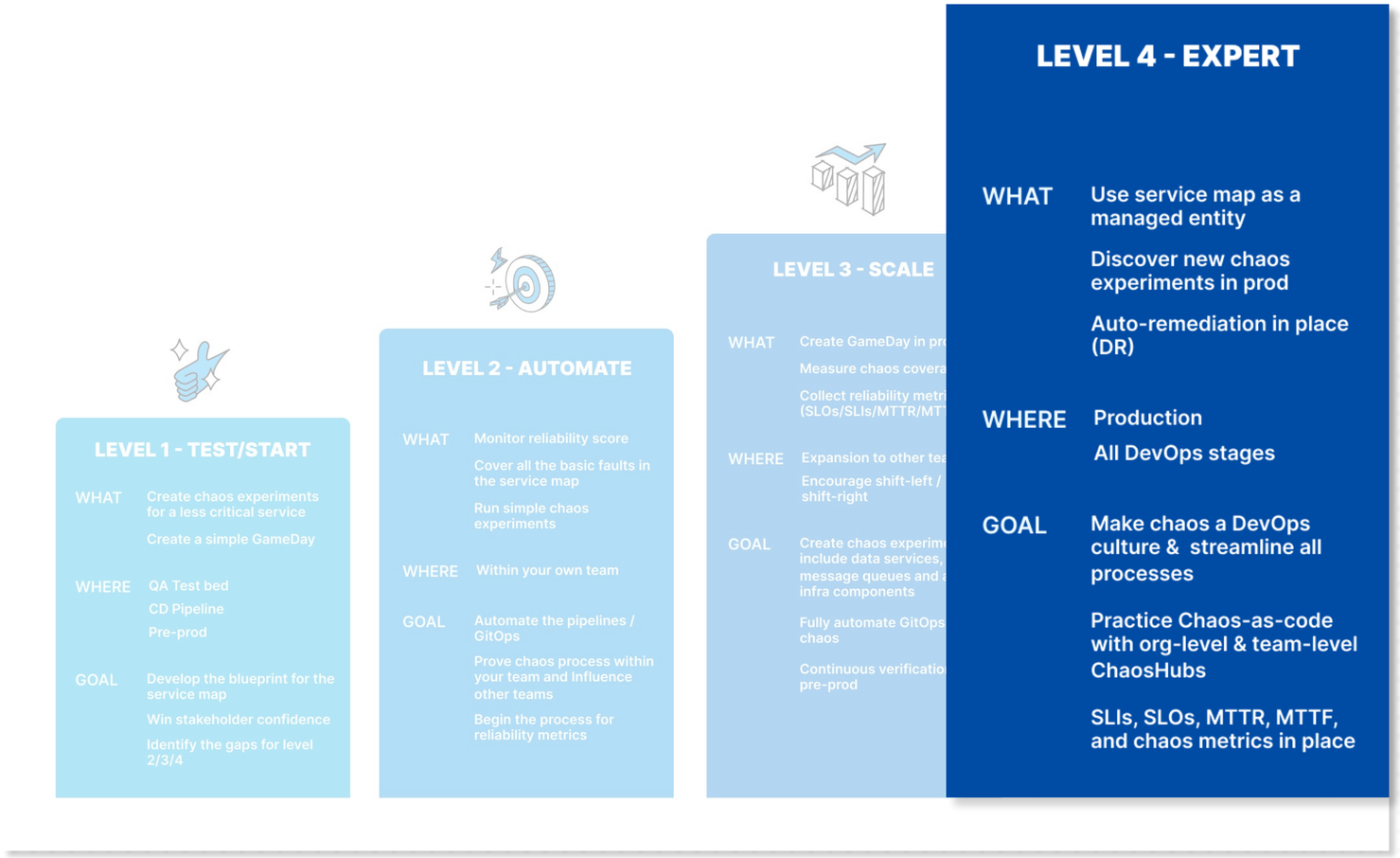
Level 3: Scale

Here, the testing and automation that was set up successfully for one team can be scaled out across all teams and services. The QA and pre-production environments can be scaled to ensure shift left developer testing of the pipeline to catch any issues before releasing to production. Chaos coverage can be measured, service maps are updated and reviewed as changes are made in environments, and processes are streamlined with GitOps to control blast radius and auto-remediation of experiments that result in system failure.

Scaling chaos engineering can take three to six months, but leveraging GitOps in the pipeline can drastically reduce the time and effort.

Level 4: Expert

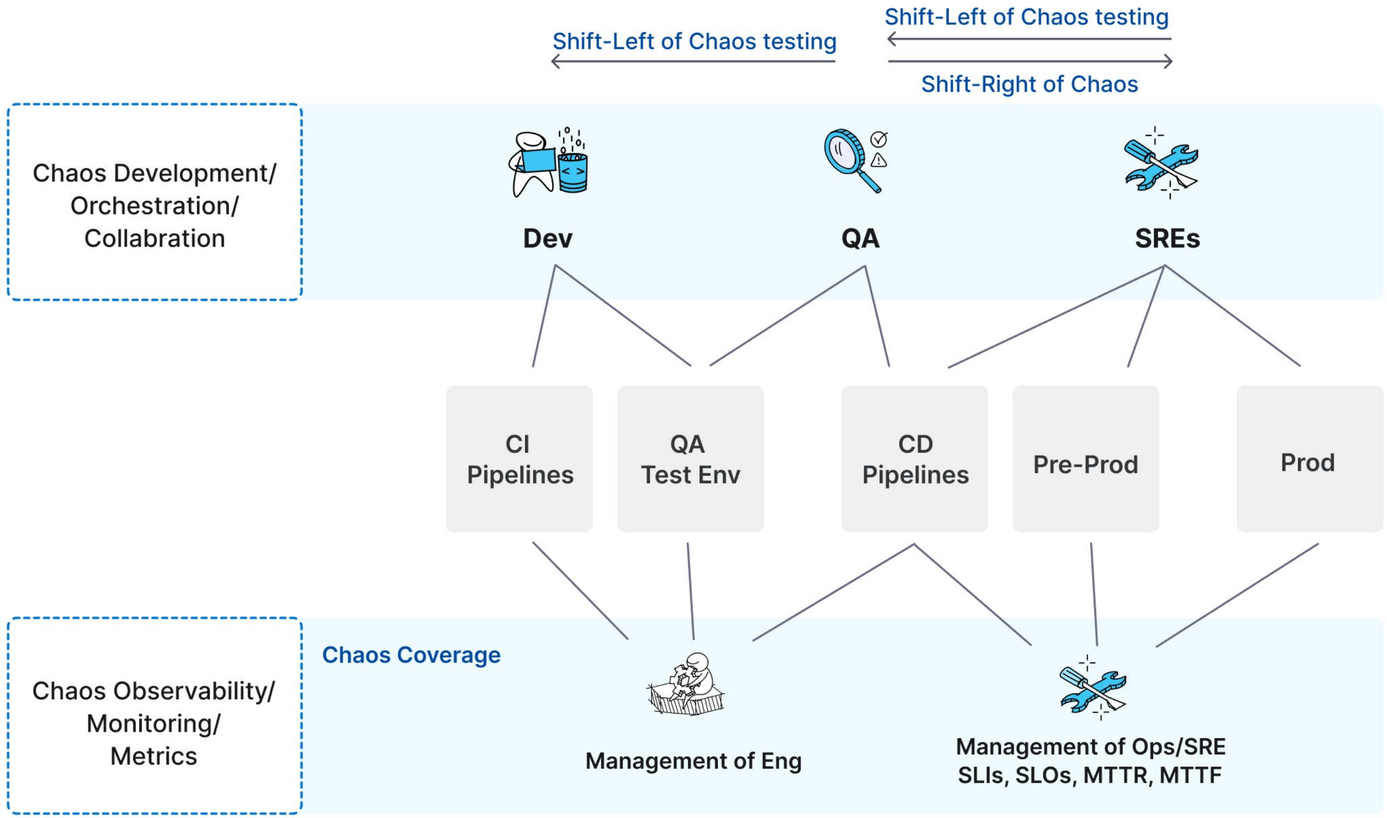
At Level 4, service map coverage is measured in production for known chaos experiments and new chaos experiments are developed by the SRE team, which is managing production incident recovery. The new chaos experiments are then introduced into the lower environments to validate that the issues are resolved so the incident cannot happen again. The team has adopted metrics associated with service level objectives and implemented pipeline guardrails to limit production releases depending on the health of the service.



The Enterprise Chaos Engineering Journey

There is a common assumption within DevOps that chaos engineering follows a shift-left approach, meaning that chaos engineering is practiced in pre-production or production and then moves to QA or development environments. While this assumption was true in the earlier days of chaos engineering, when setting up chaos was considered a complex task with a monolithic architecture, it is not true anymore. Chaos engineering is a more common practice in QA because teams are taking the additional responsibility of validating the functional reliability of cloud native applications during the build process because of the ease of cloud native chaos engineering.

As shown below, chaos testing can start throughout the SDLC and propagate to other personas through shift-left and shift-right testing. This bi-directional capability allows verification and validation of failure modes that enable teams to measure the reliability of their service throughout all the changes occurring in the environment.



For example, an SRE will resolve a production incident that caused unplanned downtime. After the incident, the SRE will work with the development team to fix the underlying issue in the service. To validate the issue has been fixed, the team will proactively run a chaos experiment to recreate the impact that occurred previously during the incident but validate that the system mitigated or prevented impact to the end user. To prevent a recurrence of this issue as the system changes through development, the SRE can recommend this chaos experiment be incorporated into the developer pipeline, which shifts the failure validation to the beginning of the SDLC where the developer can understand the impact of changing the service code before it goes out to a customer.

Another path in the enterprise journey is the growth of observability and reliability metrics. Initially, the metrics for QA engineers and developers are mostly percentages of code coverage, while SREs will have standard reliability metrics such as SLIs, SLOs, MTTF, and MTTR. As teams work together, they can build these into reliability metrics to have a full understanding of their chaos engineering maturity throughout the SDLC.

Another path in the enterprise journey is the **growth of observability** and **reliability metrics**.

Use Harness to Mature your Chaos Engineering Practice

Harness CE provides a reliability platform enabling developers, QA engineers, and site reliability engineers to collaborate and reduce risk to the business by proactively addressing system failures that cause unplanned downtime. Unlike some solutions that don't offer cloud native capabilities and require manual effort to automate experiments, Harness CE empowers enterprise customers to move fast while maturing the reliability of their systems and team.

Whether your chaos engineering practice is maturing slowly with your current solution or you have yet to start the journey, Harness CE can help you achieve a fully mature chaos initiative in a year or less. Harness, as your chaos engineering partner, provides the support, onboarding, and expertise, so you can quickly scale to the next level of chaos engineering maturity.

Ready to get started?

Talk to one of our experts to learn more about how Harness CE can help mature your chaos engineering practice across your entire organization.

Get Started

The Modern Software Delivery Platform™

Follow us on

/harnessio

/harnessinc

Contact us on

www.harness.io