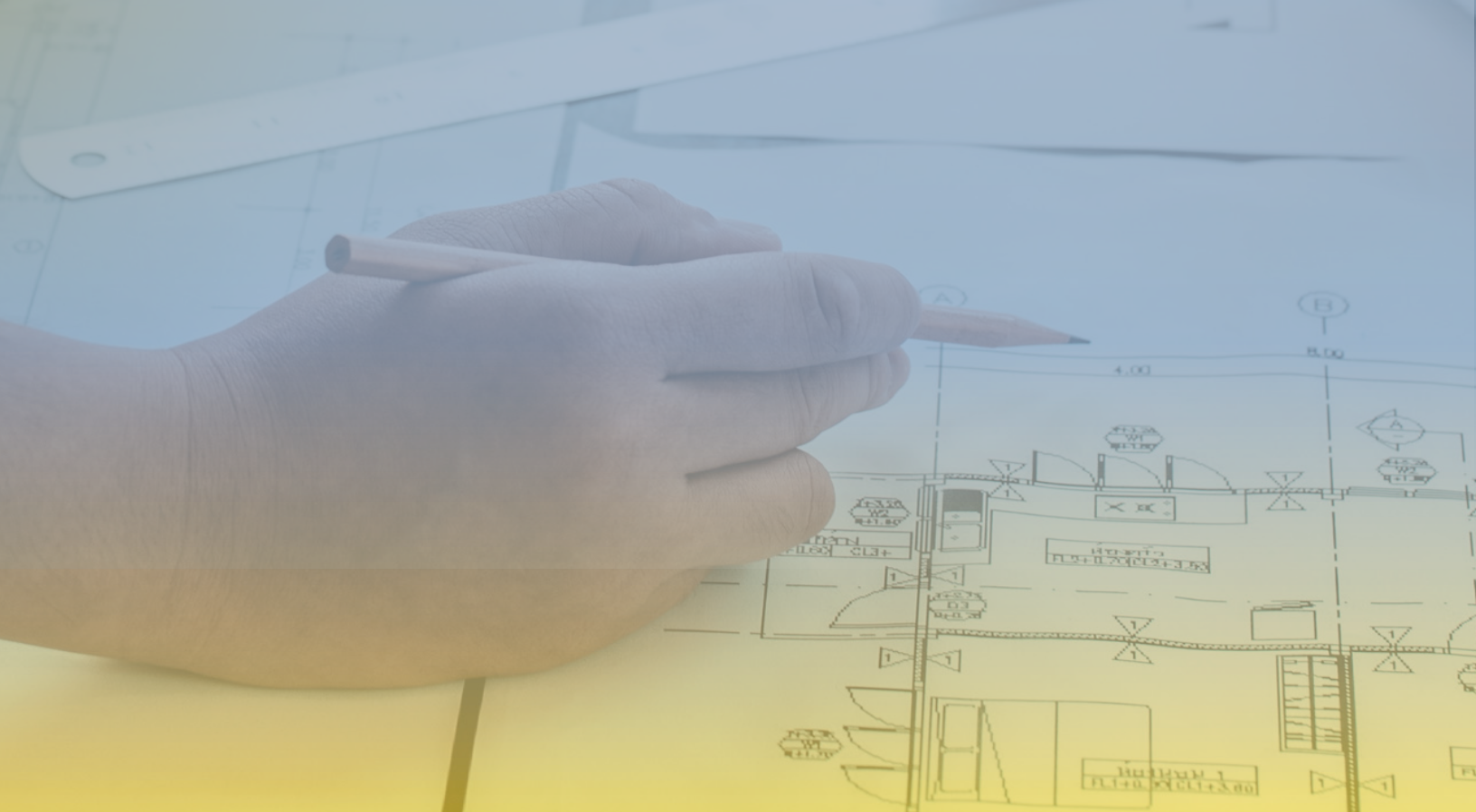
LTIMindtree

Whitepaper

Adopting Chaos Engineering

Part II



LTIMindtree

Table of Contents

1 Excecutive Summary 3

2 Introduction 4

3 Intoduction to Chaos Engineering..... 7

4 How Does Chaos Engineering Work?..... 9

5 Introducing Chaos Engineering Into Organizations..... 13

6 Embracing Chaos Engineering Into an Application 15

7 Running an Experiment for an Application..... 17

8 Examples of a Chaos Test..... 20

9 Chaos Engineering Tools- A Comparison..... 21

10 Benefits of Chaos Engineering..... 22

11 Conclusion..... 23

12 Authors 24

13 References 25

01 Executive Summary

In the realm of modern architectures, the pursuit of reliability reaches new heights, and Chaos Engineering emerges as a crucial ally. This whitepaper provides an insightful glimpse into the world of well-architected frameworks while introducing the transformative concept of Chaos Engineering (CE).

Chaos Engineering is a proactive approach that empowers teams to uncover vulnerabilities by deliberately introducing controlled chaos into their systems. It functions through meticulous experimentation, where failures, latency, or disturbances are injected to observe their impact on the architecture, paving the way for system fortification.

To successfully embrace CE, organizations must foster a culture that embraces failure as an opportunity for growth. The whitepaper provides strategic guidance for integrating CE, emphasizing the importance of collaboration, trust, and aligning chaos experiments with business goals.

Furthermore, it offers a practical roadmap for implementing CE within an application, covering critical aspects such as steady-state criteria, hypothesis formulation, and effective experiment execution.

02 Introduction

Building robust and resilient systems has become increasingly challenging in today's rapidly evolving technological landscape. Traditional approaches to ensure system reliability often fall short in the face of complex distributed architectures and hyperscale demands. As organizations strive to deliver uninterrupted services and exceptional user experiences, an approach is necessary to embrace the chaos rather than avoid it.

In part 1 of this whitepaper titled **"The Need for Chaos Engineering in Modern Architectures,"** we explored the inherent difficulties in building resilient systems and the limitations of traditional approaches. We identified the pressing need for a paradigm shift that embraces chaos as an opportunity for growth rather than a hindrance to stability.

In this whitepaper, we delve deeper into the transformative power of chaos engineering, offering a comprehensive understanding of this innovative discipline. With a sneak peek into the world of reliability through well-architected frameworks for hyperscalers, we unlock the potential of chaos engineering to revolutionize how we design, test, and operate modern systems. With chaos engineering as our guiding principle, we embark on a journey to unveil the secrets of resilience and enable organizations to thrive in an increasingly complex and dynamic technological landscape.

A sneak peek into reliability through well-architected frameworks of hyperscalers

Reliability is one of the pillars of a well-architected framework of all cloud providers. It provides checklists, architectural patterns, and best practices to build reliable systems. While building reliable systems, one needs to focus on the Non-Functional Requirements (NFRs) of the application. These requirements will determine the architectural and deployment choices one must consider in the cloud.

Some of the key items to consider before making the architectural choices are:

- Business criticality of the application
- Service Tier (Tier 1/2/3--> Organizations may have their definitions) to determine the
  - Uptime and availability of the application
  - Mean Time to Recover (MTTR), Mean Time Between Failures (MTBF)
  - Availability SLAs and so on
- User distribution (Local/Regional/Global)

Combining these requirements with the cloud providers' service SLAs, we can arrive at the right architectural/deployment pattern for the application. Refer to cloud providers' best practices while making such design decisions.

The shared responsibility model provides visibility regarding where application teams focus, while cloud providers offer reliability in the foundational areas. For example, AWS is responsible for providing a reliable network, storage, computing, and other services in line with the published SLAs.

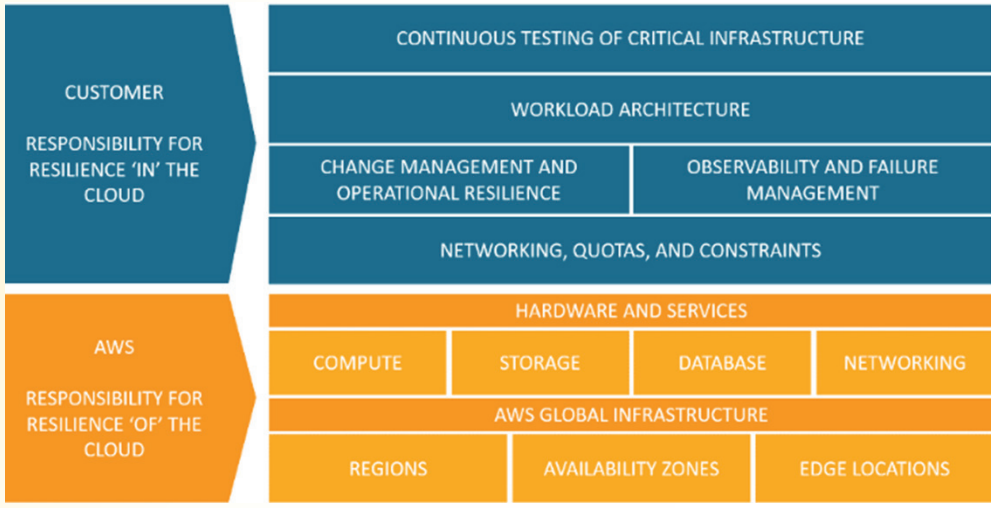


Figure 1: ***Shared Responsibility Model,*** AWS : https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/shared-responsibility-model-for-resiliency.html

Given the nature of the cloud business, there are several services cloud providers offer in every area. These choices will directly impact the application reliability; hence, as a customer, cloud providers make us responsible for areas as below:

- **Networking, Quotas, and Constraints**
  - Ensuring we choose the right network topologies, services with the right level of quotas, etc.

- **Change Management and Operational Resilience**
  - Having the right level of monitoring, metrics, and alerting in place for all the components of the system.
  - A better understanding of the continuous capacity needs, reliable scaling up/down practices.

- **Observability and Failure Management**
  - Ensuring the DR practices and other recovery processes meet business SLAs.
  - Visibility of systems health with the right automation in place to heal.

- **Workload architecture**
  - Right architectural choices to have a smaller blast radius, and improved fault isolation.
  - Applicable implementation patterns like the ones referred in the previous post. (Visualizing failures in modern architectures)

- **Continuous Testing of Critical Infrastructure**
  - Improved rigor on traditional testing methods with
  - **New practices for testing resiliency (Chaos Engineering).**

Each of the above items demands a separate article on its own. In our future blogs, we will try to bring details for some of them. However, **this whitepaper will cover the last responsibility item** with particular emphasis on chaos engineering.

"Good intentions don't work, mechanisms do."

- Jeff Bezos

03 Introduction to Chaos Engineering

Chaos engineering is a field that involves experimenting on a system to instill confidence in its ability to withstand turbulent conditions in production. The idea here is to identify weak points in the system before they lead to outage conditions.

By regularly doing these experiments in the target system, we continue to identify and resolve resiliency bottlenecks, thus improving the overall system reliability. It is not about randomly breaking things in production or shutting down nodes. Instead, it involves systematically injecting failures with the right plans should something go wrong. But who in the world wants to disrupt a running system? Until recently, teams hardly had the confidence to do such experiments. Over the years, organizations have started introducing chaos engineering into their Software Development Life Cycle.

Chaos engineering has already crossed the chasm of the technology adoption lifecycle. According to Gartner, nearly 40% of organizations will be using chaos engineering in some shape or other by this year (2023). The report says this will result in a 20% reduction in unplanned downtimes. Such reduction means a lot for critical applications

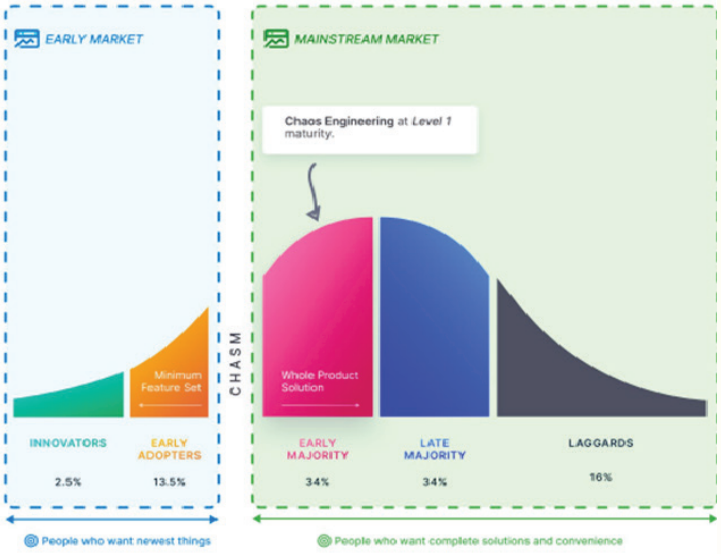


Figure 2: *Chaos Engineering on the Technology Adoption Lifecycle,* https://www.harness.io/downloads/the-chaos-engineering-maturity-model

It is essential to understand that chaos engineering (CE) does not replace the traditional validation methods to confirm system performance, endurance, and other critical qualities. We still need to have the same rigor in terms of evaluating applications. Unlike other testing methods, CE complements through resiliency tests, preferably in a production environment. There is often a myth that CE is always executed in a production environment, and teams often get nervous about moving the needle forward due to such a mindset.

Although the pattern is highly convincing and suitable for real-world experiments, the decision usually hinges on factors like risk tolerance, the criticality of the application, etc. Imagine financially sensitive applications or applications where outages may endanger lives. It gets more challenging to conduct production experiments for these applications. In such situations, lower environments become an obvious option though this may not entirely remove the resiliency uncertainties.

04 How Does Chaos Engineering Work?

CE emphasizes the importance of systematically planning and executing the resiliency tests in a controlled manner. Fix if the experiments result in failures, and repeat the process to improve the system's resiliency progressively. The below diagram depicts a standard CE process followed today. Let's go into the detail of each step.

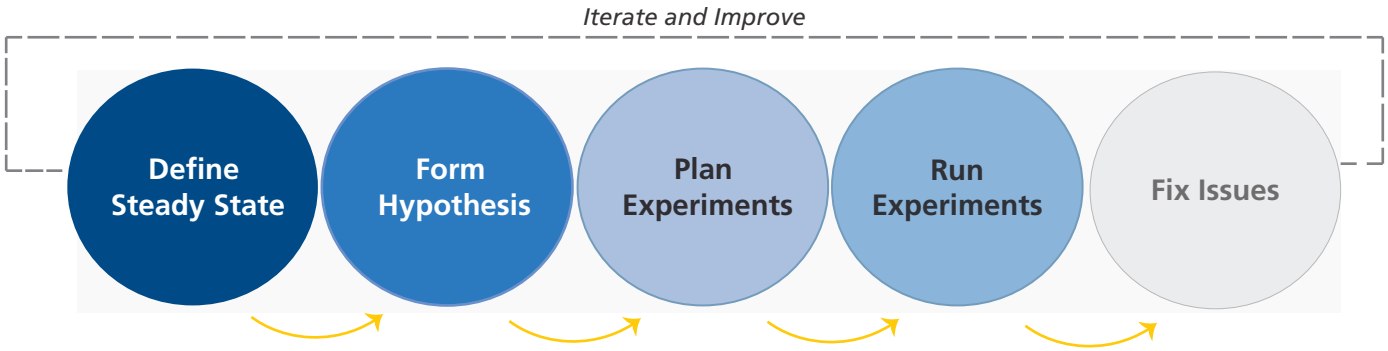


Figure 3: *Test Execution Cycle*

Define Steady State

A steady state defines our belief in the status of the application before, during, and after the execution of a chaos experiment. If the application does not return to the expected state, it is considered violated. A good definition of steady-state behavior is to relate the expected experiences of end users with the disruptions introduced.

Hence consider metrics like SLOs, performance SLAs, and service availability for defining steady state. Leverage the NFRs captured during the solution design phase. Some key parameters you may leverage include technical metrics like latency, error rate, traffic volume, and resource exhaustion. Business metrics include the number of logins during peak hours and the number of failed logins. Relate these impacts to the user's expected experience.

Form Hypothesis

A hypothesis is the expectation we have from running the experiments. A few examples include:

- The application health check endpoint will not be impacted throughout the experiment.
- If 30% of the nodes in the EKS node group are terminated, the application still responds under 500ms for the 99th percentile of traffic.
- The application will function normally during database failover but should throw a friendly message for certain transactions failing.
- When the nodes are down, alerts will be triggered in 2 mins.

Plan Experiments

Identify experiments with minimal impact and blast radius. A recommended way to define experiments is to follow the "Known/Unknowns" matrix.

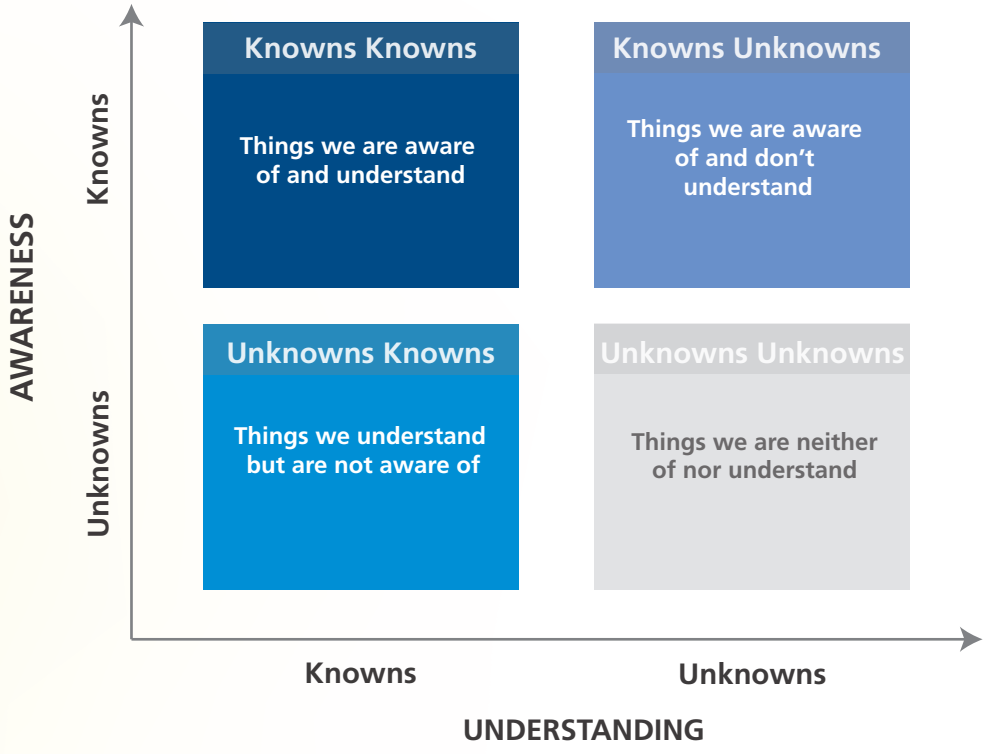


Figure 4: *Known-Unknown Matrix*

Let's relate this to a microservices solution running on a Kubernetes cluster. Sequence your experiments in the order mentioned below.

Known-Knowns: When one of the pods for any service terminates, another similar pod will be started as part of the replica set.

Known-Unknowns: When a pod terminates, we don't know how long the load balancer takes to direct all the requests to the healthy pod and the behavior of the requests forwarded to the unhealthy pod.

Unknown-Knowns: If we terminate two pods(replica) simultaneously, we don't know the meantime for two new pods to start.

Unknown-Unknowns: We don't know what will happen if 70% of the nodes in a node group fail in our primary region.

Run Experiments

Experiments are run by dividing the workload into control and experimental groups. Control groups are isolated from chaos failures and are immune to the impact. The experimental group is where the disruption happens. While running experiments, don't jump directly into production, as we may not be fully aware of the behavior of the experiment. Hence in a non-prod environment,

- Execute functional tests to validate business metrics.
- Execute load, endurance, and stress tests to expose vulnerabilities during an experiment and fix issues.
- Run experiments.
- Record the outcome, mitigate, and re-run.

The next page has a sample experiment approach from AWS.

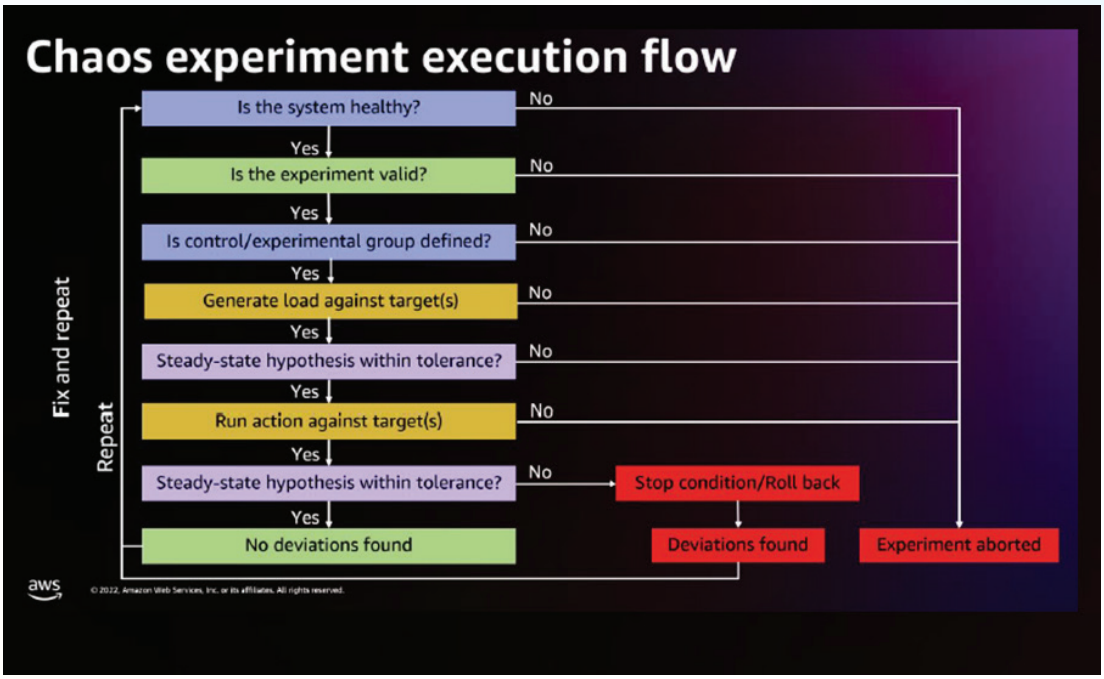


Figure 5: ***Building confidence through chaos engineering on AWS,*** AWS, November 29, 2022: https://d1.awsstatic.com/events/Summits/reinvent2022/ARC307_Building-confidence-through-chaos-engineering-on-AWS.pdf

Fix/Learn & Iterate

During the execution of these experiments, stop the experiment if the steady state is disrupted. Focus on understanding what failed the steady-state hypothesis. Bring the necessary changes to the platform before further running any experiments.

05 Introducing Chaos Engineering Into Organizations

With uncontrolled executions and a lack of oversight, CE will result in true chaos in the organizations. As a result, it may fizzle out soon. Hence one needs to induct CE progressively and methodically. One of the ways to establish CE in an organization is via a **Centre of Excellence** focused on chaos engineering. We will explain in detail how this may work and how platform teams can engage and start their CE journey for their applications. The CoE could be an extension of the SRE team but should be multi-disciplinary with representations from Architecture, QA, Platform/DevOps, and SRE.

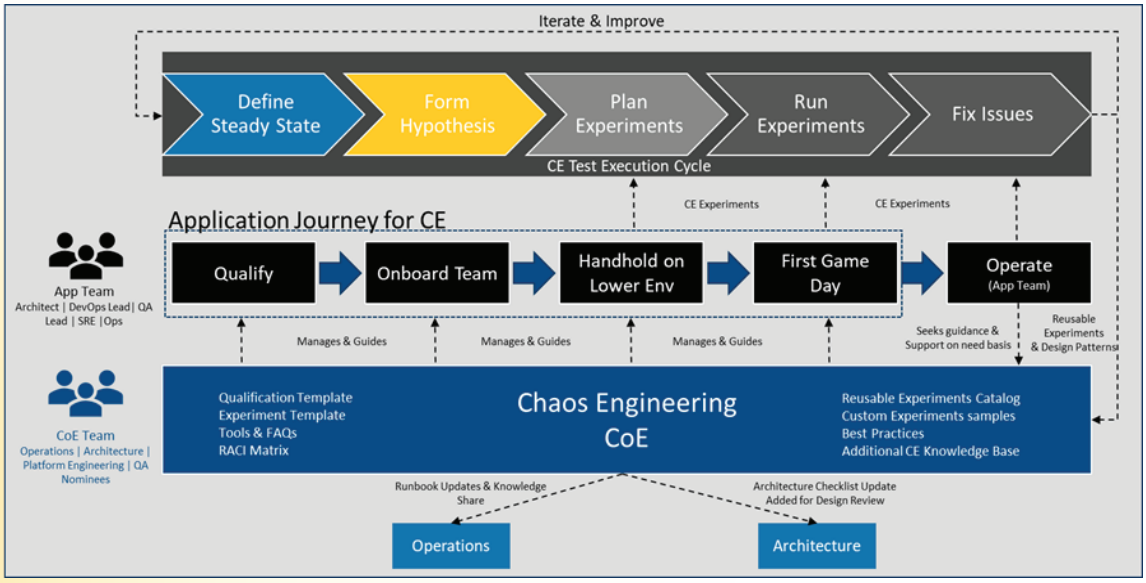


Figure 6: *Chaos Engineering Establishment*

The first team kickstarting the CE program may form and lead the CoE until multiple applications are willing to start their CE journey. However, keeping distinct roles from the beginning is better to ensure smoother adoption. The job of the CoE includes but is not limited to the below ones.

- Be an enabler for adopting CE in a systematic way into application teams. Empower and guide the application team to kickstart their CE Journey. (Hub & Spoke Model)
- Conduct market research and arrive at appropriate CE tools per organizational needs.
- Help application teams through readiness assessments and, upon successful onboarding, guide them with initial experiments until First Game Day. Provide clarity on key roles required for the CE journey for the application.
- Provide standardized experiment templates to define and capture outcomes of test executions.
- Help teams to understand what a CE test case truly is.
- Provide simple real-world experiments to get conversant with the approach.
- Provide best practices and common reusable patterns based on past executions within and outside the enterprise.
- Obtain learnings from different application executions and evaluate for the fitment to make it a reusable pattern. This can be fed to Architecture and operations teams for appropriate use in their respective areas.

LTIMindtree

06 Embracing Chaos Engineering Into an Application

Like any other testing procedure, CE does consume labor and money and is not a substitute for existing testing practices. Some of the criteria to qualify applications for the CE journey may include:

Business/Mission Criticality of the application

Will the outages lead to major revenue/brand impact? Look for applications that are classified for higher service tiers. Applications with lower availability SLAs, relatively simple to moderate in complexity, may be deprioritized for later; if not, we can eliminate them.

State of the application

Is the application in the early stages of development and hasn't completed the MVP? Are the environments stable enough? Integrating early into product/application development may cause prioritization and execution challenges. Hence wait for the application for its first stable state as a general practice.

Observability of the application

Running disruptive tests is just one part of the CE, but the objective is understanding what happens with the induced disruption. Are we able to detect signals, and do they mean any? Having the right level of metrics, tracing, logging, and alerting, along with observability of the application, is a must. This is also one of the reasons why the stage of the application matters, as teams often deprioritize observability over feature development in the early stages. The application team should be able to make better decisions given the knowledge of their application for better instrumentation and guardrails needed.

Deployment Homogeneity

Start with applications deployed in a single cloud provider while starting the CE journey. As CE matures in the organization with adequate tooling to handle Hybrid/Multi-Cloud topologies, we can onboard such applications. There may be a steeper learning curve to consider otherwise.

Service Dependency Map

With so many moving parts in modern architecture, having a good view of the component and service dependencies is necessary. Running an experiment without knowing these dependencies will lead to confusion and chaos. Ensure teams adequately understand the dependencies and have an up-to-date service dependency map.

DevOps Maturity

To get the true benefit of CE, avoid executing chaos experiments manually in any environment, as it lacks speed, consistency, frequency of execution, and volume of executions. If there are existing pipelines, we can gradually embed the chaos test through shift-left behavior. Teams that integrate IaC into their delivery make it flexible to execute chaos tests using different execution patterns.

Revisit the well-architected framework's reliability pillar, ensuring the best practices are in place and have been embraced along with the application change.

07 Running an Experiment for an Application

Based on the above validations, if there is a need to bring the required changes, teams can prioritize and implement the necessary changes to make it CE-ready. Once this is done, we can kickstart the onboarding procedure with the help of CoE.

Focus on making the team comfortable with CE processes, scenarios and able to visualize what is to be truly tested. Start with 1 or 2 simple real-world experiments with a smaller blast radius. Make sure you have enough guardrails to manage should things go wrong. Use standard experiment templates and preferably have this within the CoE repository for wider reuse.

A sample experiment template may look like the one below.

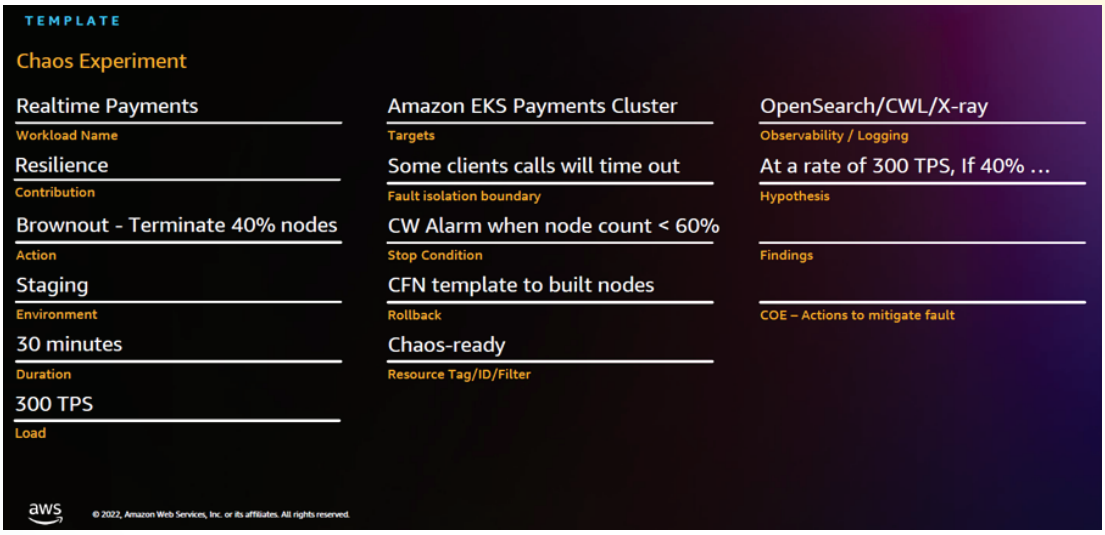


Figure 7: ***Building confidence through chaos engineering on AWS,*** Laurent Domb, AWS Static, December 2, 2022: https://d1.awsstatic.com/events/Summits/reinvent2022/ARC307_Building-confidence-through-chaos-engineering-on-AWS.pdf

Start conducting the experiments in the lower environments a couple of times, and once you are comfortable, move them upstream. At this stage, the team is comfortable and conversant with the execution, and it is time for experimentation in the production environment.

Seek guidance from the CoE and validate the preparation procedures for the Game Day. Ensure required parties are all in place, including Incident management, operations, application, and release teams. Ensure the runbooks are up to date and the IAM policies align with the lower environments. Perform the execution and record your learnings. There are a few different ways a test can be executed in production. The sample execution flow is highlighted below.

1. Direct execution on product deployment

It a straightforward from an execution standpoint but the riskiest. On the other hand, it does not demand any additional infrastructure.

2. Leverage canary deployments or other identical methods

These deployments create an immutable infrastructure where we may apply controlled routing policies, ensuring we have better control should there be an impact. This identical infrastructure, however, introduces additional costs and demands automated infra spin-off to be in place.
A sample pattern in AWS is highlighted below. We can do similar ones for other cloud providers as well.

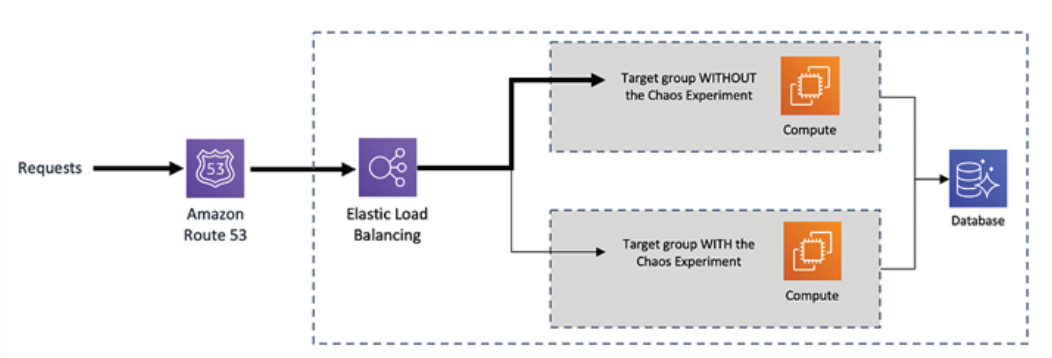


Figure 8: ***Chaos Engineering — How to safely inject failure in your application?,*** Adrian Hornsby, Medium, May 8, 2020: https://medium.com/the-cloud-architect/chaos-engineering-q-a-how-to-safely-inject-failure-ced26e11b3db

The result could confirm the steady-state hypothesis, which is good. It is also welcome if there is a failure and the need to abort the experiment because we have found a resiliency gap in the application.

Fixing the gaps should be prioritized, ensuring it is not falling into the backlog abyss. Ensure product owners know the risks, push the fixes early, and repeat the experiments, ensuring the steady state hypothesis works this time. If the experiments are done manually, ensure you integrate them into mainstream DevOps pipelines through automation. Manual experiments will increase the TOIL.

Every successive execution should result in the improvement of system resiliency. Should you think that even with these executions, if there is no reduction of incidents with time and there is no improvement in MTTD and MTTR metrics, revisit the Chaos tests. Understand whether we are addressing the problem in the right area. Consult with other teams and engage the CoE. At the end of the day, this should not add to the overhead; instead, it should help build the resiliency of the applications.

08 Some Examples of a Chaos Test

- **Simulate Node/Component Failures:** To validate whether the application can withstand the loss of single or multiple nodes.
- **Resources Exhaustion**: Application behavior for peaking CPU or memory usage.
- **Latency simulation**: What happens if we gradually increase latency for critical flows? How sensitive is the application for the latency drops? Is it able to continue to function?
- Drain container instances on the cluster.
- Abrupt termination of instances
- Termination of nodes in a K8s cluster. Can the application shift loads to other available nodes, and create additional nodes without disrupting the application?
- Terminate a group of nodes that pertain to the node group.
- Loss of cache service like Redis. What happens to the application performance? Does the DB get overloaded? Is it able to sustain?
- Key HTTP error simulations (500/404 and others) partial and full
- Delay injection for third-party integrations. Is this creating a cascading impact? Are our retries work fine? Are we able to circuit break as expected?
- **DB state change simulation:** What happens if a DB is in shutting down/failing over /rebooting state? Does the application demonstrate expected behavior?
- **Loss of single zone**: Are our services able to withstand the loss of a zone as expected? Is the cluster still able to function normally?
- **Loss of a Region:** For a multi-region design, is the application fully functional even with a loss of complete region? Is it able to recover and operate within the SLOs?
- **Network failures**: In hybrid or Multi cloud networks, does the application behave as expected if the connectivity fails?
- Network Packet loss simulations.

There are many ways to simulate these experiments, from using cloud consoles to CLIs of cloud providers. The APIs are handy, and often the tools available in the markets use them and provide a better level of abstraction.

LTIMindtree

09 Chaos Engineering Tools - A Comparison

Unlike the days of Chaos Monkey, we have many tools in the market. Indeed, cloud providers themselves have started introducing the tools for CE. Avoid going into the route of custom scripts, as it may not scale for the long run. These tools vary in applicability (Vendor-Lock ins), ability to script custom experiments, etc. The choice would depend on the needs of the organization.

	CE Tools				
Capabilities	Litmus	Chaos Mesh	AWS FIS	Azure Chaos Studio	Gremlin
License	Open Source	Open Source	Commercial	Commercial	Commercial
Works with	Kubernetes	Kubernetes	Cloud (AWS-only)	Cloud (Azure-only)	Kubernetes, containers, Linux, and Windows hosts
Application Faults	No	No	Yes	Yes	Yes
Host Faults	Yes	Yes	Yes	Yes	Yes
Container / Pod faults	Yes	Yes	No	Yes	Yes
GUI	Yes	Yes	Yes	Yes	Yes
CLI	Yes	Yes	Yes	No	Yes
Rest API	No	No	No	Yes	Yes
Reliability scoring	Yes	No	No	No	Yes
Metrics reporting	Yes	Yes	Yes	Yes	Yes
Custom faults	No	No	Yes	Yes	No
Health Checks	Yes	No	No	No	Yes

Table1: *Chaos engineering tools and their capabilities*

Tools that are not covered but still worthy to look at include **Harness, Steadybit, ChaosBlade, and Chaos Toolkit.**

10 Benefits of Chaos Engineering

As mentioned in **Part I of this document,** the average cost of downtime can vary from few thousand to millions of dollars an hour. Frequent downtimes will heavily impact direct and indirect costs. While we cannot eradicate the downtimes, the adoption of Chaos Engineering can lead to:

- **Reduction in frequency of outages:** More reliable runbooks and well-prepared teams as they are exposed to different controlled disruptions regularly. This will lead to improved MTTD and MTTR. Teams can become more productive, shifting their energy to improve other critical areas.

- **Improved service uptimes leading to improved customer trust:** Many eCommerce companies make most of their sales during the peak periods (Thanksgiving/Cyber Monday/Boxing Day/Others). A resilient system during these periods will significantly improve overall revenues and the NPS. Companies have turned this into their competitive advantage and provide a differentiator to their customers.

- Reduction in Legal and Regulatory impacts in sectors like **Healthcare & Financial Services.**

- **Improvement in Deployment Frequency:** With higher levels of operational confidence, teams start embracing more deployments, creating the opportunity to ship new features faster, thus aiding businesses well.

11 Conclusion

We have highlighted the challenges to building resilient systems and how CE can help improve the same. Further, we have provided a framework to introduce CE into organizations systematically. The exploration of Chaos Engineering **divided in two parts**, within the context of modern architecture has revealed its immense potential to revolutionize system reliability. By recognizing the limitations of traditional approaches and embracing controlled chaos, organizations can proactively identify weaknesses, enhance fault tolerance, and build more resilient systems.

We encourage organizations to embrace Chaos Engineering as a transformative discipline, leveraging its power to unlock the full potential of their systems and achieve unparalleled levels of resilience in the face of uncertainty and complexity.

12 Authors

Ragupathi Palani (Ragu)

Associate Vice President - Head of Architecture Europe/UK

Ragu heads the Architecture Practice for Europe and UK, providing architecture & advisory services to CIOs, CTOs & Heads of Architecture of LTIMindtree Customers. A technology leader with over 22 years of experience envisioning, architecting, and implementing medium to complex enterprise applications. Besides, he has diverse experience ranging from Portfolio Rationalization, Platform Assessment, Reference Architecture, Performance Engineering, and AI/ML to anchoring large pre-sales engagements from Architecture. Outside work, he enjoys playing badminton and spending time with family and friends.

Joydeep Gupta

Principal Architecture

Joydeep is a keen technology enthusiast and evangelist with over 15 years of experience defining end-to-end architectures for clients in various domains. He has been part of pre-sales solutions and leading digital transformations across different customer portfolios. He has been an advocate of MACH architectures and has successfully transformed multiple projects from on-prem monolith-based systems to cloud-based, open-source, microservice, and API-first designs. Apart from work, he is a music buff, likes playing guitar, and loves to travel.

13 References

- *Chaos Engineering 101,* Mathias Lafeldt, Sharpended.io, February 10, 2016: https://sharpend.io/chaos-engineering-101/
- *PRINCIPLES OF CHAOS ENGINEERING,* Principlesofchaos, 2019 March: http://principlesofchaos.org/
- *Chaos Engineering For Cloud Native - A Definitive Guide,* Navdeep Singh Gill, Xenonstack, 19 August 2022: https://www.xenonstack.com/blog/chaos-engineering-for-cloud-native
- *The Dual Approach in Scaling: Chaos Engineering and Performance Engineering,* Kyle McMeekin. Gremlin, MARCH 15, 2022: https://www.gremlin.com/blog/the-dual-approach-in-scaling-chaos-engineering-and-performance-engineering/
- *Chaos Engineering: the history, principles, and practice,* Tammy Butow, Gremline, May 5, 2021: https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and-practice/
- *2022 Gartner Hype Cycle for Agile & DevOps Report Identifies Four Solutions Chef Continues to Lead,* Michelle Sebek, Progress Chef, December 15, 2022: https://www.chef.io/blog/2022-gartner-hype-cycle-for-agile-devops-report-identifies-four-solutions-chef-continues-to-lead
- *Continuous Chaos—Introducing Chaos Engineering into DevOps Practices,* Sathiya Shunmugasundaram, CapitalOne, August 10, 2018: https://www.capitalone.com/tech/software-engineering/continuous-chaos-introducing-chaos-engineering-into-devops-practices/
- *Shared Responsibility Model for Resiliency*, AWS: https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/shared-responsibility-model-for-resiliency.html
- *Observability in the realm of Chaos Engineering,* Subhra Mondal, Prateek Sachan, Medium, Jun 10, 2020: *https://medium.com/@nabtechblog/observability-in-the-realm-of-chaos-engineering-99089226ca51*
- *Digital Transformation Statistics and Digital Skills [2022-2023],* Josh Sultan, Digital Adoption, August 9, 2023: *https://www.digital-adoption.com/digital-transformation-statistics/*
- *Are you an Elite DevOps performer? Find out with the Four Keys Project,* Dina Graves Portman, Google Cloud, September 23, 2020: *https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance*
- *Chaos Engineering — How to safely inject failure in your application?* Adrian Hornsby, Medium, May 8, 2020: *https://medium.com/the-cloud-architect/chaos-engineering-q-a-how-to-safely-inject-failure-ced26e11b3db*
- *AWS re:Invent 2022 - Building confidence through chaos engineering on AWS (ARC307),* AWS Events, YouTube, January 2023: *https://www.youtube.com/watch?v=tm5GEePP1PY*
- *Chaos Engineering Fundamentals: The Steady State,* Joshua Root, GitHub, 2019: *https://github.com/chaosiq/chaosiq/blob/master/chaos-engineering-steady-state.md*

LTIMindtree is a global technology consulting and digital solutions company that enables enterprises across industries to reimagine business models, accelerate innovation, and maximize growth by harnessing digital technologies. As a digital transformation partner to more than 700 clients, LTIMindtree brings extensive domain and technology expertise to help drive superior competitive differentiation, customer experiences, and business outcomes in a converging world. Powered by 82,000+ talented and entrepreneurial professionals across more than 30 countries, LTIMindtree — a Larsen & Toubro Group company — combines the industry-acclaimed strengths of erstwhile Larsen and Toubro Infotech and Mindtree in solving the most complex business challenges and delivering transformation at scale. For more information, please visit **https://www.ltimindtree.com/**