LTIMindtree

Whitepaper

The Need for Chaos Engineering in Modern Architectures

Part I

LTIMindtree

Table of Contents

01 Executive Summary

As the landscape of software development evolves, the imperative of maintaining high system reliability and resilience has become paramount. This white paper explores the growing need for Chaos Engineering in contemporary architectures, highlighting the challenges posed by unplanned downtimes, the rapid pace of software releases, the intricate nature of distributed systems, and the limitations of current testing methodologies.

To address these challenges, Chaos Engineering emerges as a pivotal approach. By intentionally introducing controlled disruptions into production environments, organizations can uncover vulnerabilities, identify weak points, and proactively enhance system resilience. The ability to visualize and understand failures within modern architectures is a core component of Chaos Engineering. By simulating various failure scenarios and observing their impact, teams can gain insights into the complex interactions within their systems, facilitating more informed decision-making and targeted improvements.

In conclusion, this white paper underscores the urgency of integrating Chaos Engineering into modern software development practices. Through Chaos Engineering, companies can fortify their systems against failures, enhance their understanding of system behaviors, and ultimately deliver more robust and reliable services to their users.

02 Introduction

A recent Dzone report on the state of DevOps shows that nearly 70% of the surveyed organizations make multiple monthly production releases. Speeding up releases will further accelerate with the advancement in architecture patterns, tooling, cloud adoption, and proliferation of ready-to-plug COTS solutions.

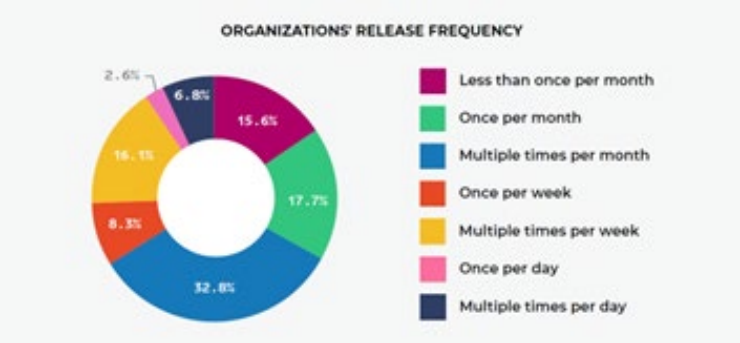

Figure 1: ***Organizations' Release Frequency***, DZone, Feb. 16, 2023: https://dzone.com/trendreports/devops-4)

However, this has increased the complexity of the software and dependencies beyond traditional boundaries, making it even more difficult for operations to meet the Service Level Objectives (SLOs). The demands will continue to rise with increasing digital transformation, ever-demanding customers, and increased competition. Hence, the traditional ways of validating the software's readiness and reliability may not be adequate, and a different approach is necessary. One of the options to mitigate the impact is to leverage Chaos Engineering along with existing validation methods. We will delve into the intricacies of Chaos Engineering in **two parts.** Part I will cover the following aspects:

1. Emphasizing the importance of uptimes
2. Evolution of software architectures and its impact on resilience
3. Limitation of traditional validation methods
4. Understanding of failures and failure domains in a modern architecture

By combining the subjective insights in the whitepaper with the objective analysis provided in the next, we aim to offer a holistic view of Chaos engineering. Let's begin!

LTIMindtree

03 Understanding the Unplanned Downtimes and Their Impacts

In 2016, Delta Airlines had a power failure causing three days of critical systems' outage. As a result, nearly 2000 flights were grounded. The losses were to the tune of USD 150 million. In 2018, on a major Prime Day Sale, Amazon's multiple failures created a cascading impact. This resulted in a revenue loss of around USD 99 million. In 2021 Facebook (Now Meta) had a network configuration error resulting in 5.5 hours of outage impacting nearly 3.5 billion users. This resulted in a loss in advertising sales of USD 60 million.

These incidents simply indicate the level of damage unplanned downtimes can bring to businesses. While the above metrics emphasize the economic impacts, the cost of downtime is not merely the loss of revenues. One needs to factor in both direct and indirect costs of such outages like the ones below.

DIRECT COSTS	INDIRECT COSTS
Lost revenues (Refer above)	Productivity losses (Troubleshooting, Detection, Fixing, Validation, etc.), Employee churns
Business disruption	Legal & regulatory impacts
Customer churn (Immediate & Eventual)	Brand impacts
Fines	Data losses

Table 1: *Direct and Indirect costs of outages*

Various research firms (Gartner to Ponemon Institute) put the average cost of downtime to $9000 per minute. These numbers can spiral to millions of dollars/hr for some industries or systems.

Cloud migration may improve the uptime of systems in general, but that is not a panacea. Hyperscalers, too, go down from time to time. Organizations have many years of experience and maturity in building and running systems. They do have runbooks and run periodic drills. Shouldn't they be able to contain problems? Why should it be so difficult even for tech organizations, even the likes of Facebook/Amazon, unable to prevent unplanned downtimes in the modern era?

While there could be many factors, some prominent ones constraining the teams to address Mean Time to Resolve (MTTR), Mean time to detect (MTTD), and Service Level Objectives (SLOs) are mentioned below.

The rate of software release has gone many folds

Traditionally releases were done at monthly or quarterly intervals. This helped the operations team easily maintain the runbooks and plan for failure. Many organizations for years did have break failure simulations as part of their Standard Operating Procedures. But typically, they were executed before going live for the first time, which may not be adequate for modern infrastructures. With the speed of releases, running manual simulations and preparing for every unforeseen situation is nearly impractical.

04 The Complexity of Distributed Architectures

As nicely put by Benoit Hediard in his article, software architecture has evolved over the decades. Microservices Serverless architectures, and other recent variations have enabled the independent evolution of services with improved ownership. Such separation greatly helped teams to ship software faster and reliably.

But on the other hand, an application that used to be deployed with a single bundle into VMs or on a physical machine is broken into tiny units(containers) and deployed into some orchestration platforms(K8s), demanding multiple components and processes to work together to achieve the same functionality.

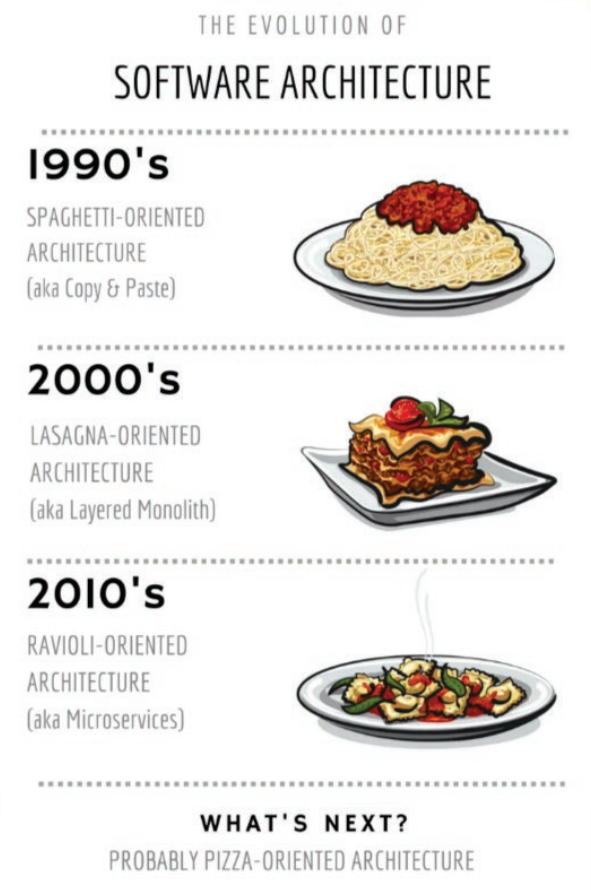

Figure 2: ***The evolution of software architecture***, Benoit Hediard, Benorama, May 29, 2015: https://benorama.com/the-evolution-of-software-architecture-bd6ea674c477

Figure 3: *Service Dependency Graph – Amazon,* Werner Vogels, X, June 11, 2016: https://twitter.com/Werner/status/741673514567143424

Figure 4: *The butterfly effect,* Wikipedia: https://en.wikipedia.org/wiki/Butterfly_effect

The quality-of-service offerings from Cloud Providers and modern architectures can significantly improve reliability and resiliency. However, with such architectures, a new set of challenges emerges. It is articulated well with the fallacies of distributed computing by Gosling and L Peter Deutsch.

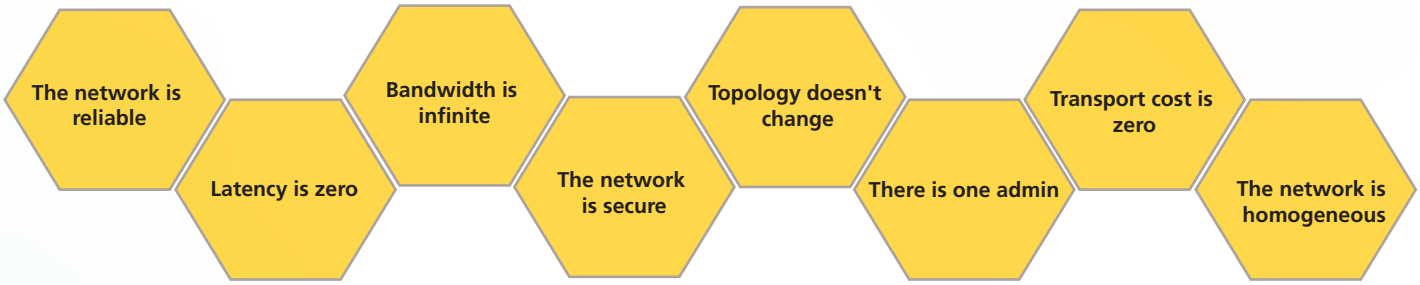

Figure 5: *Fallacies of distributed computing*

Figure 2, is an original service dependency graph from Amazon that reveals the magnitude of dependencies between services. These dependencies may involve several network calls with varying payloads under varying load conditions serviced by different process instances. The complexity of the landscape is truly overwhelming. This makes it more difficult to detect problems (MTTD) and resolve them on time (MTTR).

This leads us to Choas theory's butterfly effect, which shows that small changes in initial conditions can lead to significantly different outcomes in complex systems.It is not the initial issues that have such an impact; rather, the whole set of interdependencies makes the small changes more vulnerable.

Though DevOps Tools are handy and enable shaping the observability of such platforms, they alone will not be sufficient to prevent production issues.

05 Limitations of Current Testing Methods

Traditionally we apply various performance, penetration, and other testing methods to safeguard the quality of our software. Most executions happen in lower environments, and we assume the configurations, topologies, and production behaviors are identical. While these approaches might have worked in the past, currently, they are inadequate due to ever-increasing dependencies between components in a distributed architecture.

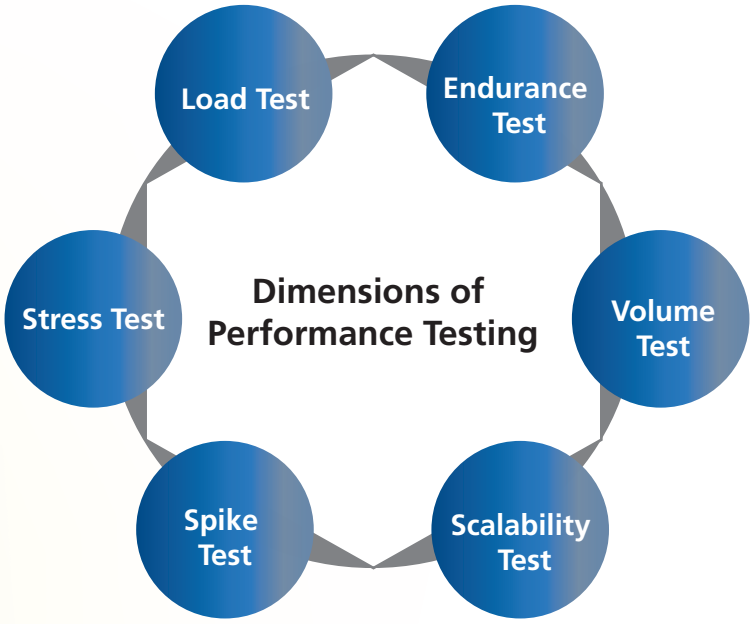

Figure 6: *Dimensions of performance testing*

Imagine scenarios like this and the challenges it brings to system conformance:

1. Can the application handle the same load if a part of the infrastructure is down?
2. If dependent services are down, will the application work under normal conditions?
3. How will the application behave if servers run out of memory or disk space?
4. Will the application exhibit similar behavior in successive executions?
5. What if latency is introduced on some of the services? Will it create a cascading impact?

Coupling scaling with resilience is essential to fully understand the concept of scaling.

06 Visualizing Failures in a Modern Architecture

While designing and preparing systems for resiliency, it is important to understand the impact of architectural choices, the nature of failures one can anticipate, and how to arrive at patterns and processes to address them.

Such patterns and processes are constrained by the architectural and design trade-offs we need to make due to reasons not limited to operational and infrastructure costs, place of deployment (on-prem/cloud), time to market, availability of tools, inter/intra-org dependencies, organization's technology standards, and roadmap. Let's run through two architectural patterns and their impact on resiliency.

The diagram below is a simplified view of a typical monolithic deployment with redundancy for the application layer. The application, as such, is deployed as a single bundle comprising multiple modules. Traditionally, we have built our monolithic applications with sticky sessions, thus having an affinity to specific nodes.

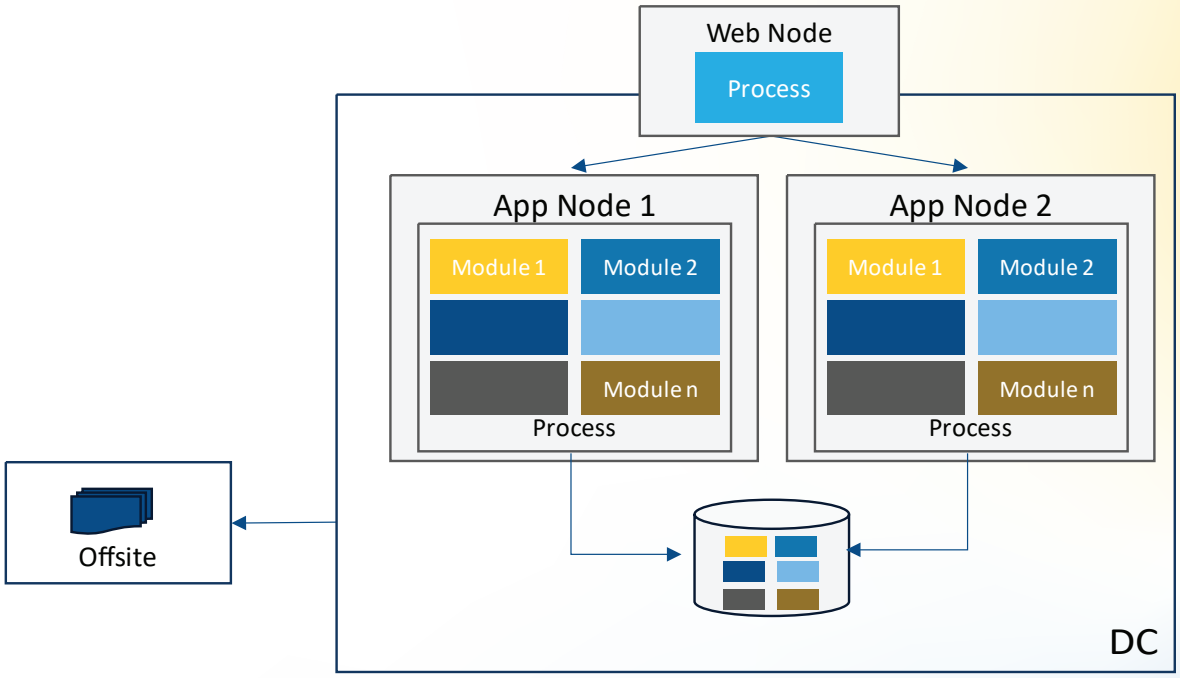

Figure 7: *Monolothic Deployment*

A single, vertically scaled DB server with logical isolations (Synonyms etc.) supports multiple modules. Periodic snapshots are obtained and backed up offsite in line with recovery point objectives (RPO). The entire application runs on a single DC with offsite backups. Now let's imagine what can go wrong in this deployment. A few possibilities include:

Application Node failures

1. This will result in a loss of user sessions leading to poor customer experience.
2. Increased load on the remaining nodes, which may not be able to provide the required service. Should you need to support this, additional capacity must be installed. How much and how soon?
3. The application may still withstand the failure and serve requests.

DB Node Failure

1. Complete loss of service Single point of failure.
2. There is a loss of data since the last snapshot if the node is not recoverable.
3. If recovery procedures are not regularly tested, it is likely to take longer to reinstate the DB with offsite backups. The data size, bandwidth limitations, and reliability between offsite and on-prem may lead to recovery time objectives (RTO) violations.

Network Failure

1. Partial or complete loss of service, depending on the area of impact
2. Potential Single Point of Failure

DC failure

1. Complete loss of service. Single point of failure

Other Failures may include:

1. Disk Failures
2. Long Waits
3. Cascading Failures
4. Retry Overload Failures
5. Unexpected Traffic Spike related failures
6. Deployment-induced failures
7. Env-centric configuration errors (Both infra and app-specific level)

The list may go on. However, not all failures lead to a complete loss of service. Some are localized, and some have widespread impact. This is what blast radius means. The larger the failure spread (DB/Network/DC failures) larger the blast radius is. One way to improve the resiliency of applications is to contain the failure domains and blast radius, leading to better fault isolation.

Modern Architectures (Microservices etc.) and cloud-native deployments are very handy for isolating faults and may benefit from smaller fault domains and blast radii. The diagram below depicts an abstract view of a microservice deployment with a sidecar infrastructure.

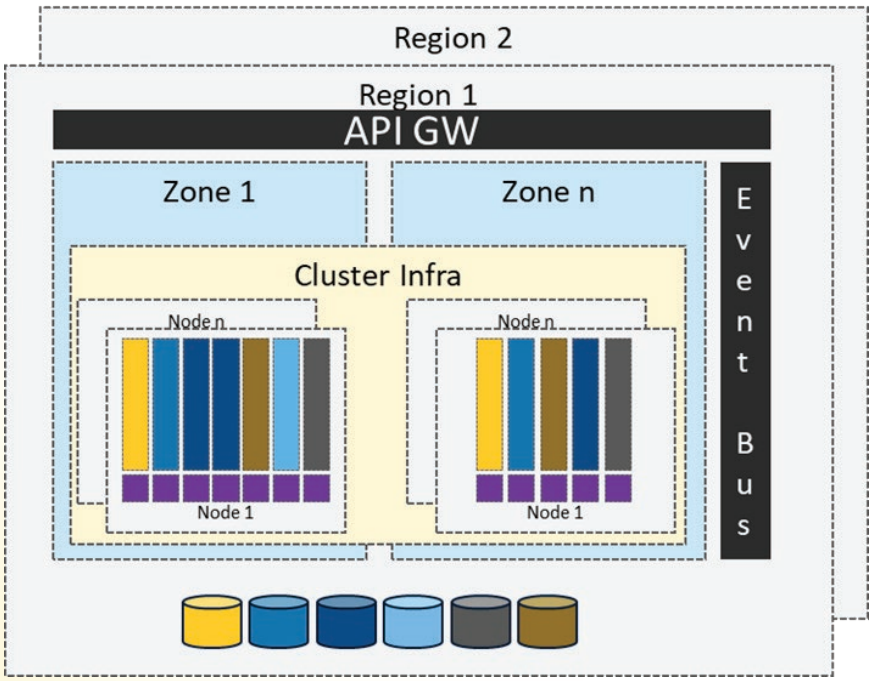

Figure 8: *Microservice deployment with a sidecar infrastructure*

We needed to build systems that embrace failure as a natural occurrence even if we did not know what the failure might be."

-Werner Vogels

Let us look at the nature of failures and potential patterns we can apply for such distributed/microservice deployments.

FAILURE	POSSIBLE PATTERNS/OPTIONS
Application, DB, and Messaging Platform Failures	Loose coupling, redundancy & replication, db per service, Scaled out DBs/NoSQL, message persistence, multi-az & microservices
Request Delays, Errors, Service Unavailability & Positive Feedback overload	Circuit breakers, exponential backoff, code level handling
Traffic Overload & Cascading Failures	Handshaking, Autonomous Scaling, Failure Isolation, Bulkhead & Circuit Breaker
Infra side Component Failures(Storages, Nodes, Zonal and Regional)	Redundancy, Identical/Swappale Designs, Multi Zonal/Regional Replication and Backups MultiAZ & Global and right managed Services Static Stability Stateless, Containerization InfraAsCode
Release/Deployment based failures(Rollback Minimization, Deployment Downtimes Reduction)	Blue-Green Deployments, Canary Releases, A/B Testing, Infra as Code & Rollback Automation
Scalability constraints	Self-Healing Infrastructure, Serverless/Containerization patterns Stateless services/No node or session affinity Event Driven

Table 2: *Failures and potential patterns for distributed/microservice deployments*

To summarize, distributed architectures with cloud-native deployments provide a greater opportunity to improve systems reliability. However, bringing adequate controls and testing them for resilience still takes a lot of effort.

07 Conclusion

Modern architectures have brought numerous benefits to systems through overall quality and reliability improvements. As a side effect, they have introduced ever-expanding dependencies. Traditional testing methods are deterministic, and it is hard to factor every testing scenario and execute the same during every deployment. Hence, an empirical approach is needed to complement the current resiliency testing methods where we can induce controlled disruptions and validate systems' behavior.

This is where Chaos Engineering comes into play. **Part two of this Whitepaper- Adopting Chaos Engineering**, will cover the the CE principles and tools, and how an organization can adopt Chaos Engineering.

We take an analytical approach by delving into Chaos Engineering deeper.

"An evolving system increases its complexity unless work is done to reduce it"

– Meir Lehman

08 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.

LTIMindtree

09 References

- *Chaos Engineering 101*, Sharpended.io, Mathias Lafeldt, February 10, 2016: https://sharpend.io/chaos-engineering-101/
- *Principes of Chaos Engineering, Principles of Chaos,* March 2019: http://principlesofchaos.org/
- *Chaos Engineering For Cloud Native - A Definitive Guide,* Xenostack, August 19 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,* Tamy Butow, Gremlin, May 5, 2021: https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and-practice/#user-content-a-brief-history-of-chaos-engineering
- *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,* Capital One, 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
- https://medium.com/@nabtechblog/observability-in-the-realm-of-chaos-engineering-99089226ca51
- *Digital Transformation Statistics and Digital Skills [2022-2023],* Digital Adoption, February 10, 2023: https://www.digital-adoption.com/digital-transformation-statistics/
- *Are you an Elite DevOps performer? Find out with the Four Keys Project,* Google Cloud, September 23, 2020: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance
- *The 10 Biggest Cloud Outages Of 2022 (So Far),* Wade Tyler Millward, Crn, July 01, 2022: https://www.crn.com/news/cloud/the-10-biggest-cloud-outages-of-2022-so-far-/5

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/**