The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 154-158
DOI 10.37547/tajet/Volume07Issue05-14

Check for updates

Ensuring the Availability of Critical Cloud Services Through SRE Practices

Alexandr Hacicheant
Software Engineer and Head of Reliability Engineering at Mayflower.

Author LinkedIn Profile: - Alexandr Hacicheant
company's site: - Mayflower

OPEN ACCESS

SUBMITTED 24 March 2025
ACCEPTED 20 April 2025
PUBLISHED 19 May 2025
VOLUME Vol.07 Issue 05 2025

CITATION
Alexandr Hacicheant. (2025). Ensuring the Availability of Critical Cloud Services Through SRE Practices. The American Journal of Engineering and Technology, 7(05), 154–158. https://doi.org/10.37547/tajet/Volume07Issue05-14

Abstract: The availability of cloud services is a critical factor in the success of digital products. Downtime in essential systems—whether for fintech platforms or major online retailers—can lead to substantial financial losses and reputational damage. Meanwhile, modern cloud infrastructures continue to grow in complexity. Distributed architectures, automated scaling, and frequent software releases all increase the risk of system failures.

In this dynamic environment, companies are actively seeking strategies to minimize incidents and mitigate their impact. One of the most effective approaches is Site Reliability Engineering (SRE)—a discipline pioneered at Google that combines engineering best practices with operational processes to enhance the reliability and resilience of cloud services.

This article examines how Site Reliability Engineering (SRE) principles address the challenges of maintaining cloud services availability. Alexandr Hacicheant, Head of Reliability Engineering at Mayflower, provides an analysis of key issues in this field, the core methodologies of SRE, and real-world applications that contribute to minimizing downtime.

Keywords: *Site Reliability Engineering,* SRE, *Cloud services, Infrastructure, SLI*, SLO

Introduction:

Challenges in Cloud Infrastructure Availability

Modern cloud services function within distributed environments characterized by fluctuating workloads

and infrastructure spanning multiple data centers worldwide. This complexity presents challenges in scalability, traffic balancing, and maintaining continuous service availability.

A primary challenge in managing distributed systems is their inherent complexity. Cloud services operate across thousands of nodes, each vulnerable to hardware failures, network congestion, and request processing delays. As demand increases, resources must be rapidly scaled; however, dynamic scaling itself can introduce instability.

Another significant risk factor is human error. When resource scaling is performed manually in response to rising demand, delays in provisioning additional servers may lead to service downtime. Misconfigurations—such as incorrect hostnames, IP addresses, or port settings—can also contribute to system failures. Furthermore, erroneous changes may be applied to the wrong environment, or critical resources, such as virtual machines or databases, may be unintentionally deleted. In the absence of well-defined testing procedures and rollback mechanisms, such errors can result in substantial business disruptions.

Service availability is also constrained by unpredictable failures that cannot be entirely eliminated. These may include sudden outages of major data centers, network disruptions between regions, or failures in third-party APIs on which a service relies. Even when systems are designed with redundancy, unforeseen circumstances can still result in service disruptions.

In this context, a key challenge for cloud infrastructure is predicting failures and mitigating their impact. Effectively addressing this issue requires a comprehensive strategy that integrates automation, continuous monitoring, and rapid recovery mechanisms.

How SRE Addresses These Challenges

Site Reliability Engineering (SRE) is an engineering-driven discipline focused on ensuring system reliability and high availability. At its core, SRE establishes measurable standards for evaluating service performance and systematically managing incidents. Rather than aiming for absolute failure prevention, this approach prioritizes minimizing the impact of disruptions and utilizing incidents as opportunities for continuous learning and process refinement.

Failures as an Expected Norm: Accepting the Inevitability of Outages

Given the complexity of modern distributed systems, eliminating failures entirely is infeasible. SRE acknowledges failures as an inherent aspect of system operations and shifts the focus from prevention to mitigation. The objective is to minimize the effects of failures and restore service functionality as efficiently as possible.

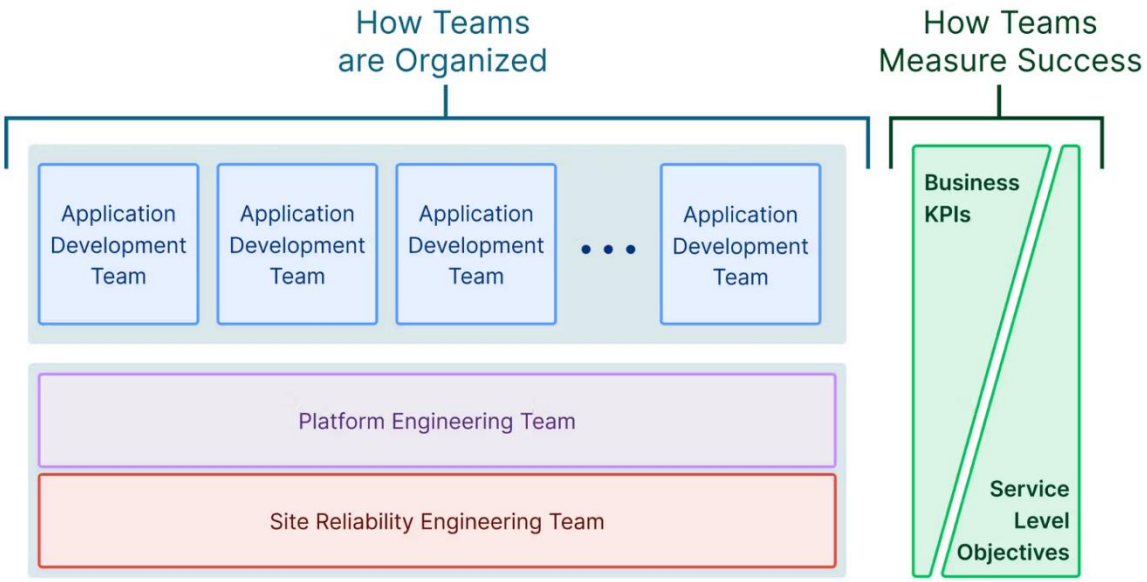
Rather than adopting a purely reactive approach, SRE teams rely on comprehensive monitoring and proactive alerting mechanisms to anticipate potential system degradation. For instance, if API response latency surpasses a predefined threshold, an automated scaling process can be triggered to allocate additional resources before users experience any perceptible decline in service quality. Moreover, instead of allowing an outage to fully disrupt operations, systems can be designed to degrade gracefully, maintaining partial functionality under adverse conditions. For example, if a database becomes temporarily unavailable, the system can retrieve cached query results, ensuring that users continue to receive responses while engineers resolve the underlying issue.

A Unified Culture of Responsibility

A key limitation of traditional development models is the separation of responsibilities: developers primarily focus on building new features, while operations engineers are tasked with maintaining system stability. This division often leads to inefficiencies and delays in deploying updates. SRE addresses this challenge by integrating these roles, fostering a collaborative culture that enhances consistency and accelerates the deployment of new functionality without compromising reliability.

Under this model, developers assume accountability for the operational performance of their code. Rather than limiting their responsibilities to writing and shipping software, they also monitor its real-world behavior in production environments. This approach improves the quality of releases, reduces unexpected failures, and encourages a proactive stance toward system reliability. Conversely, SRE engineers actively contribute to the development process by designing

robust and scalable systems from the outset, ensuring that reliability is embedded at every stage of the software lifecycle.



Error Budget: A Risk Management Tool

While the complete elimination of failures is unattainable, their frequency can be managed within acceptable limits. To achieve this, SRE introduces the concept of an error budget—a predefined threshold that specifies the permissible amount of downtime within a given period.

For example, if a service has an availability target of 99.95%, its error budget allows for 0.05% downtime per month. This mechanism provides flexibility in release management: when a system demonstrates high reliability, updates can be deployed more frequently. Conversely, if the error budget is depleted due to a series of incidents, deployments are temporarily paused to prioritize system stabilization. Furthermore, error budgets help teams assess acceptable risk levels and reinforce a reliability-first mindset across development and operations.

Rapid Detection and Isolation of Issues

Minimizing downtime requires a structured approach to failure recovery, with Mean Time to Recovery (MTTR) serving as a critical performance metric. To reduce MTTR, automated rollback strategies are employed, enabling the system to revert to a previously stable version without requiring manual intervention in the event of a failed deployment. Additionally, feature flags allow teams to selectively disable problematic features rather than rolling back an entire release, thereby preserving system stability while addressing localized issues.

Another widely adopted strategy for risk mitigation is canary releases, wherein new features are initially deployed to a limited subset of users. If the system remains stable, the deployment is incrementally expanded to a broader audience. This controlled rollout minimizes the potential impact of defects while facilitating continuous innovation.

Measuring Reliability with Metrics

- **Service Level Objectives (SLO)** define target reliability levels that organizations commit to maintaining. These objectives help establish clear risk thresholds based on key performance indicators such as uptime, response times, and request success rates.

  For instance, an SLO may specify that a system must maintain 99.9% availability per month, meaning total downtime should not exceed 43 minutes. If real-time monitoring data indicates a deviation from this target, engineers receive alerts, often

through tools such as Prometheus Alertmanager, facilitating a rapid response.

- **Service Level Indicators (SLI)** provide an empirical assessment of system health by tracking metrics such as response latency, API success rates, and error frequencies. These indicators allow organizations to detect anomalies and address potential reliability concerns before they escalate into major incidents. Analytical tools such as Grafana and Kubernetes' built-in monitoring capabilities are commonly used to collect and analyze SLI data.

Practical Example: Configuring SLI and SLO Metrics

Implementing SRE best practices requires a structured approach to measuring and improving service reliability. This process involves defining key performance indicators, monitoring system behavior, and continuously refining infrastructure to minimize downtime.

Consider a large international marketplace that processes millions of orders each month across multiple regions. Critical service scenarios include product catalog browsing, search functionality, cart operations, and payment processing.

For the payment system, an SLI may be defined as the percentage of successful transactions, transaction processing time, and service uptime. Similarly, for search functionality, SLIs could include request success rates, response times, and the accuracy of retrieved product metadata.

To establish performance benchmarks, historical data must be collected using tools such as Sentry Distributed Tracing or Jaeger for request routing analysis, Web Vitals for frontend performance insights, and ELK stack-based log management systems for in-depth incident analysis.

Based on this data, baseline performance indicators can be defined. For example:

- Search requests demonstrate an availability of 99.92%, with a response time of 350 ms in 95% of cases.
- Cart operations exhibit 99.95% availability, with a response time of 240 ms in 95% of cases.
- Payments are successfully processed in 99.93% of cases, with a transaction time of 3.1 seconds in 95% of cases.

From these baselines, SLOs can be established in alignment with business objectives. For instance, an SLO for search functionality might require 99.95% availability (with an error budget allowing 22 minutes of downtime per month), response times below 400 ms for 95% of requests. For the payment system, the SLO could specify a transaction success rate of 99.95%, a processing time of no more than 3.5 seconds for 95% of transactions, and system availability of at least 99.98% (error budget: 8.6 minutes per month).

Properly configured SLI and SLO metrics enable marketplaces to scale while maintaining service stability, ensuring an optimal user experience even as traffic volumes grow.

Limitations of SRE

Despite its advantages, implementing SRE presents several challenges. One of the primary concerns is cost—establishing robust monitoring systems, automating operational processes, and developing incident response frameworks require substantial upfront investment and dedicated team efforts.

From a technical standpoint, selecting appropriate SLIs is critical. Poorly defined metrics can result in alert fatigue, leading to either excessive false positives or insufficient incident detection. For example, setting an overly aggressive response time threshold may generate an overwhelming number of alerts, causing teams to overlook genuinely critical issues.

Cultural resistance also poses a significant challenge. Successful SRE adoption requires fundamental shifts in organizational processes, including close collaboration between development and operations teams. However, in companies with rigidly established workflows, this shift may be met with resistance. Additionally, some organizations may have punitive cultures that discourage open discussions about failures, making it difficult to implement post-mortem analyses effectively.

To address this issue, implementing a blameless culture can be beneficial. This approach focuses on systemic problems rather than individuals, viewing failures as a consequence of system imperfections rather than mistakes made by specific employees.

With this mindset, the approach to errors becomes proactive: they are seen as opportunities to improve the system and foster team growth. Early detection of failures is encouraged, leading to open discussions and knowledge sharing among teams.

For example, Google follows a postmortem culture: after each incident, a document is created to analyze the root causes without pointing fingers. The principle of "Learning, not blaming" is applied, emphasizing lessons learned from each incident and preventing future occurrences through actionable steps. Additionally, the Error Budget serves as an objective tool for balancing development speed and system stability.

Amazon, on the other hand, follows the "Two-Pizza Team" approach, which involves forming small, autonomous teams with full ownership of their services. These teams participate in every stage of the product lifecycle, from gathering business requirements to delivering the final solution into production.

CONCLUSION

This article has examined the key challenges associated with ensuring high availability in cloud environments and demonstrated how the adoption of SRE methodologies can mitigate downtime and enhance infrastructure resilience. By integrating error budgets, proactive monitoring, automated recovery mechanisms, and retrospective incident analyses, organizations can systematically improve reliability while maintaining operational efficiency.

Despite requiring significant investment in training, process restructuring, and automation, the long-term benefits—such as reduced failure rates, faster incident resolution, and enhanced user satisfaction—position SRE as an industry standard in modern cloud computing. Future research directions include refining failure prediction models, developing self-healing systems, and integrating SRE methodologies with emerging architectural paradigms such as serverless computing and edge infrastructure.

REFERENCES

Impact of Site Reliability Engineering on Manufacturing Operations: Improving Efficiency and Reducing Downtime, IJSRP, 2020

https://www.arxiv.org/abs/2008.06717

Site Reliability Engineering (SRE), Google https://sre.google/

SITE RELIABILITY ENGINEERING A MODERN APPROACH TO ENSURING CLOUD SERVICE UPTIME AND RELIABILITY, IJCET, 2024

https://www.researchgate.net/publication/378032569_SITE_RELIABILITY_ENGINEERING_A_MODERN_APPROACH_TO_ENSURING_CLOUD_SERVICE_UPTIME_AND_RELIABILITY

Evaluating the Impact of Site Reliability Engineering on Cloud Services Availability, WJAETS, 2020

https://www.researchgate.net/publication/386087642_Evaluating_the_Impact_of_Site_Reliability_Engineering_on_Cloud_Services_Availability

Using Cloud-Native and SRE Principles to Achieve Speed and Resiliency, IBM https://www.ibm.com/think/insights/using-cloud-native-and-sre-principles-to-achieve-speed-and-resiliency