International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Benefits of Site Reliability Engineering (SRE) in Modern Technology Environments

Sibaram Prasad Panda[1], Subramanya Bharathvamsi Koneti[2], Mohanraju Muppala[3]

[1]Email: *spsiba07[at]gmail.com*

[2]Email: *subramanyabkoneti[at]gmail.com*

[3]Email: *mohanraju. m[at]outlook.com*

Abstract: ***Site Reliability Engineering (SRE) has its origin in efforts to help the IT operations organization prioritize and develop software solutions that would reduce the toil and inefficiencies inherent in traditional operations work. By automating operations using software, engineers can improve dynamic and constantly evolving systems, while engineers in traditional IT organizations tend to manage systems that are static and relatively unchanging. The term SRE was coined by a leader who started by hiring a small number of software engineers to write software to help manage its growing fleet of production systems. Since that time, thousands of SREs specializing in many different technical areas of expertise have been hired, and SRE has evolved into a substantial organization.***

Keywords: Site Reliability Engineering (SRE) Terraform, AWS CloudFormation, MTTR. Service Level Objectives (SLOs)

1. Introduction

As the name implies, an SRE is much more than an engineer, a sysadmin don, a gray beard coder. SREs have abilities and responsibilities that reach into several disciplines including coding and technical skills, DevOps, systems engineering, Linux and Solaris expertise, and performance monitoring. What makes SRE different from other engineering and operations disciplines is the emphasis on deploying and maintaining production systems by leveraging the same software and abstraction techniques used in their development. Using coding efforts to achieve efficiencies in production and be able to process more work as a product, to not burden an overworked operations organization, is the central tenet behind the SRE concept. In true DevOps tradition, SRE is a culture, a mindset, that seeks to unify the goals of development and operations and solve the inherent conflicts that too often separate these two essential functions.

2. Historical Context of SRE

To determine SRE's place in the world of engineering, you must understand the larger picture of modern technology environments and the evolution of strategies for operating them. First, however, a brief discussion of the general hacker culture that birthed the idea of SRE: it is one of open collaboration, exploration of the new territory of modern technology, encouragement (and, on occasion, gentle ridicule) of peers who invent new techniques and apply them for the first time to real - world problems, and sharing findings for the benefit of others. While this ethos is often compromised in the wider, more commercial technology world, the pioneering engineers who embraced it during its formative years continue to exert a powerful influence on the technology industry.

3. Core Principles of SRE

Service Reliability Engineering (SRE) communicates the core principles of SRE, along with the mindset, fluency, and skills necessary to understand and exploit these powerful concepts. The key idea is to manage the reliability of the services owned by a team. In the implementation of SRE, reliability is an explicit component of the service definition. The services that an SRE team is responsible for include cost, latency, availability, capacity, communication reliability, and correctness. Other definitions of reliability may vary in writing but embrace the same underlying concepts.

3.1 Service Level Objectives (SLOs)

A Service Level Objective (SLO) is a specific and measurable performance goal for a service. It determines an actual target value, and a threshold on, a metric. Usually, SLOs are expressed as "The service will be less than X% error over Y time period". This gives an indication of how stable a service is. A naive user would think it is good for a service to have 99.99999999% availability year long, but that can very well be the service doing almost nothing at all for that year. A simple fake example of an SLO would be "For search engine

Besides deciding how to calculate an SLO, teams must also figure out what SLO to create. Since optimization is expensive, choosing the right one is very important. By doing so, Dev and SRE teams would be ensuring they choose metrics that give value to the service and its users. In this way, users are not discouraged from using the service while internal deadlines on launch and feature implementations are being met. The SLO metric would help reduce and invalidate the amount of technical debt.

3.2 Error Budgets

Error budgets allow engineering teams to find the sweet spot between releases and reliability. To get an error budget, you

start with SLOs, which provide an upper limit on allowable service downtime. As a rough analogy, if your SLO is 99% availability over a given budget period, your error budget is that during that period the service is allowed to have been down for (100 - 99) % of the period length. Larger budgets are good for frequent, low - impact changes, while smaller budgets are more appropriate for critical code that does not change frequently. A two - week change window has an error budget of 150 minutes—easily exceeded by a single DoS attack or a minor service configuration error.

4. SRE vs. Traditional Operations

Site Reliability Engineering (SRE) was originally created as a new way of managing services. SRE can be seen as "a new approach to operations, " which uses some ideas from traditional operations teams, software development, and DevOps in new ways. SRE is not a descriptive term for SaaS or "web scale" businesses; nearly all of those companies have traditional operations teams as well.

4.1 Key Differences

The world of computers has been around for a long time, but the world of computers at scale is relatively new. Internet - facing services with a user base of billions are an entirely different story than the data center operations that came before them. Such services require different tools, different operations; not just faster optimization of what came before. Reliability, availability, and scalability cannot come after the fact. They must be planned from the very beginning, with specific consideration paid to how failures will occur and how systems will respond.

4.2 Key Differences

Site Reliability Engineering was born out of necessity at a company with a tremendous number of services running at substantial scale. The company had already implemented a site operations team within its Software Engineering department (which mostly worked on tooling problems), but when faced with the growth rate of its services, the need for applying a more engineering - oriented discipline to address this challenge became apparent. Lacking a name for this team, the initial engineers came up with a way to express it in a humorous way: “SRE – It’s like Ops, but with really large Data Sets.” A few years later, several companies started paying attention to this new discipline and began applying its principles to help them solve their scalability needs.

4.3 Advantages of SRE Approach

When engineers first discover SRE, they often have an Aha! moment. SRE allows the insights gained from decades of experiencing, developing, deploying, supporting, and scaling web and distributed systems at both startups and large companies to be synthesized into actionable software engineering and product management practices that change how a company operates. Different companies will emphasize different parts of SRE, and inevitably you will have to make tradeoffs that suit your environment, but the SRE approach describes how one company decided to interpret and implement SRE. Before defining SRE, it is worth describing our perspective on software engineering. Any engineer who has worked on large systems for a while realizes that it’s usually at least an order of magnitude more work to build a large system from scratch than to keep an existing system operating. But there are no easy shortcuts to scalable, distributed, and complex systems black box solution that can be reused. Additionally, there is a real and complicated problem. Scalable systems are difficult, if not impossible, to build. New frontiers emerge and old paradigms shift. What we do requires carefully coordinating teams building and managing components but on a much smaller scale. Therefore, we have had to painstakingly document patterns, practices, and tradeoffs as talking to other engineers.

5. Impact of SRE on System Reliability

The most common metric used to measure system reliability is uptime or availability. A reliable system is accessible, functional, and performing well about its Service Level Objectives over an expected duration of time. The principles and practices of SRE help organizations improve system reliability. Several high - profile websites that suffered traumatic outages partnered with experts who had launched the original SRE teams that pioneered necessary roles, books, and practices. Using the detailed postmortem processes championed by SREs, such organizations learned to diagnose the reasons for incidents and how to take measures to prevent them in the future, resulting in better MTTD and MTTR for the overall service. Several of these individuals were CTOs and technology leads at organizations we might consider SAAS pioneers.

5.1 Improved Uptime

Most websites have grown to be so essential today that even a few minutes of downtime can cause immeasurable damage. Major companies have reported thousands of dollars lost due to an unexpected outage. Instead of awaiting crises and reacting to them as they occur, SRE focuses on reducing the amount of time and energy developers spend fighting fires. SRE Teams employ various techniques such as redundancy, automated failover, graceful degradation, and monitoring. These techniques help prevent situations

from arising in the first place, reducing service interruption and prolonged downtime.

5.2 Faster Incident Response

In any technological environment, incidents are inevitable. Things happen, and sometimes they lead to outages, slowdowns, or reduced functionality. User experience suffers, and the organization loses money at least in lost productivity. Efforts are put in place to monitor service performance. Alerting is set up for on - call engineers to receive notifications when something goes wrong. The on - call engineer experiences varying levels of success in identifying what's going on, finding the right people to help, communicating with users, and implementing a mitigation or fix. With SRE, this time is reduced significantly. Environments are instrumented with the right tools to help correlate alerts, allow for debugging, and scale. Service behavior is defined within service APIs and make debugging easier. In recent years, the information technology environment is changing. New capabilities are added, such as container orchestration, microservices, and serverless to allow for scaling and rapid rollouts. But they also bring higher complexity and more chance for incidents. SREs find themselves relying more on algorithms to help reduce the incident time. Artificial intelligence and machine learning or heuristics are used for anomaly detection. Developers are implementing canaries, red/black and dark launches to allow for automatic or manual cutovers to new capabilities. SREs help define the governing aspects of the tools, such as service - level indicators and service - level objectives so that effective monitoring can be implemented, which lead to faster incident detection and resolution.

6. SRE in Cloud Environments

Cloud computing has lowered the barrier for implementing modern service architectures, especially for microservices. Organizations can now move from on - premises data centers to the cloud, where they can solve many issues automatically through cloud constructs, orchestration, and automation. This shift also can offer the possibility of avoiding full internal staffing for many functions. Yet reliability needs to be considered as the volume of services increases. Site reliability engineering is the scientific approach to solving these issues in cloud environments. Cloud service offerings greatly facilitate taking advantage of SRE principles and practices, but organizations must choose them wisely.

Scalability Benefits

SRE at scale in cloud environments means being able to build automations once and have work done to keep things running done for the organization and the users around the clock. Well - built, well - tested automation and tools can handle the load of many human operators, with the added advantage of having very few individuals needed at times of heavy load. Tools can give sites assistance storing their infrastructure as code while using CI/CD to enable change management. Additionally, cloud vendors provide many on - demand or near - demand services that can automate operational functions. DNS, load balancers, databases, and other services can scale without organizational interaction.

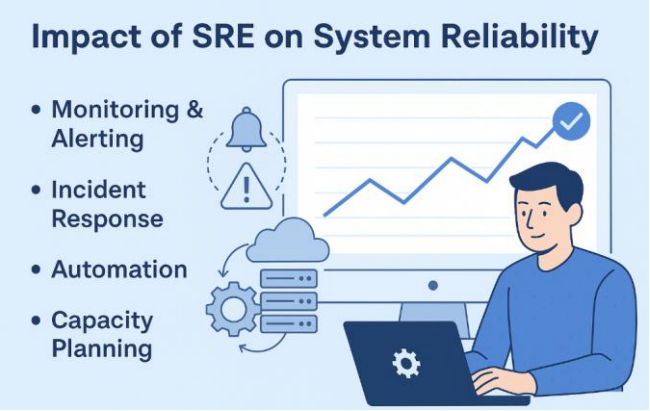
Cost Efficiency

Cost - efficiency is another focus for many sites. If a service can run in a cloud vendor's hosting environment more cheaply than on - premises, organizations should consider using it. Using a vendor means dealing with the vendor's outage since their organizations have no visibility during an event and little recourse if numbers deviate from contractual SLAs. Creating cloud environments to run internal systems such as websites that promote a business or collaborative services to assist employees makes economic sense for many organizations. These are generally non - revenue generating services whose downtime can lead to missed sales opportunities or lost workforce time.

6.1 Scalability Benefits

Scalability is defined as the ability to grow, handling a progressively larger load. An example of scalability would be a restaurant where, as customer demand increases, the owner can hire more cooks and servers to accommodate growth. With online services, one big catalyst of growth is a spike in demand, where a site experiences a sudden influx of new users. As their needs increase, the challenges are twofold. First is keeping the service running smoothly during the spike. Secondly, it is how to capitalize on that moment, ensuring that what such services do symbolizes great value when those users are most likely to require and use it onboard effectively.

Scalability is important because it makes SRE fundamentally different from traditional IT. In addition to making sure a service is available to meet current demand, SREs also take the long view and coordinate across teams to optimize systems and procedures so that the service can handle higher and higher loads. Unfortunately, the transition to managing more loads introduces problems: as more people use a service, its infrastructure becomes more complex. What was composed of a handful of servers and systems operated by a few people turns into thousands of systems, possibly located in multiple data centers and rented from different cloud providers and operated by an army of people working in different time zones. Because these systems are more complicated, a greater chance exists that some components will fail or be misconfigured, temporarily making the service unavailable or degrading its performance. Meeting the operational requirements of a non - exploding service has become the technical challenge for teams building and scaling such systems.



6.2 Cost Efficiency

SRE designs eliminate wastage, leading to better use of budgets. Because SRE employs tools such as SLI, SLO, and risk budgets, engineering teams can identify the metrics that matter to the business and its customers. SRE promotes a method of collaborative goal setting for service health based on customer satisfaction. Knowing the SLO and error budget helps the teams prioritize their work better and get rid of the queues in request - response systems. Analyzing errors can help engineering teams explicate flaws in tech stack choices as well as system performance.

7. Cultural Shifts with SRE Implementation

SRE was born from a cultural belief that operations could be done differently than it had been in the past. Operations had historically been a dry, boring job, a career for people who did not want to write code. In our opinion, operations should be different — it should be a rewarding job, one that talented people want to do. Most importantly, it should be a part of the software engineering career path. We want the people who build our systems to also be responsible for keeping them up. We think this idea is the key to building reliably and operating at a high level of efficiency.

7.1 Collaboration Between Dev and Ops

Site Reliability Engineering originated, in most cases, as a near - hands - on effort between the developers of a product and the team responsible for Operations. DevOps, as a hybrid culture, had transformed some engineering teams into mini streets on the road to production. The introduction of SRE as an exemplary of best practice, support for the goal of production quality development, and the sharing of operational coverage in the team allowed developers to buy into their mentioned mini - side support teams.

7.2 Emphasis on Continuous Learning

It is of utmost importance that SREs devote time to continue to build their skill set. Educating their workforce is likely the best investment they can make and have established a formal program called “20% time. ” Twenty percent of the time allows engineers to spend one day a week, or 20% of their workweek, on projects outside of their normal responsibilities. These can either be personal projects or company - related projects that will add value to the ecosystem but are solely employee - driven. This program has become highly regarded throughout the technical world. Often, other companies want to replicate the power that comes from 20% time, and the developers are known specifically for pushing the envelope with their projects from 20% time. Whenever a new, innovative idea is introduced, it is likely that the origin of that idea was conceived during 20% - time efforts.

8. Challenges of Implementing SRE

While developing a strong SRE practice or an SRE team can provide significant benefits to an organization, there are typical challenges along the way. In this chapter, we will discuss some of the organizational challenges that you may encounter as you scale your SRE efforts.

8.1 Resistance to Change

Organizational change is never easy. Corporate and research alike are filled with stories of initiatives that have stalled or failed. Some of the challenges have to do with inertia: organizations tend to find it easier to just continue doing what they are doing. People want to feel confident in their understanding of how the world works and that their efforts are aligned with the broader goals of the organization. In practical terms, this means that change must be driven from above, who must not only set the vision and the tone for the change but also show a willingness to make waves and to drive the effort forward. Foundational work — making organization - wide changes to how systems and services are monitored and managed, building tools to consolidate and improve developer access to operations data — can be difficult to fund and staff, especially if they’re only seen as improvements to how the company performs maintenance on existing systems. While such proactive maintenance is an important part of SRE, organizations must also make moves to shift production risk from the operations group to the development group and then back again.

That means that the wider organization needs to be prepared to make changes to its perception of its own culture. Developers need to expect that issues will arise in their services even after they’ve been deployed, that fixing those issues is a part of their jobs, and that talking about failures will lead to better reliability. They need to believe that SRE is there to work with them and help them improve their systems over time, to better enable a product and feature set that meets the company’s goals. And they need to believe that they have the necessary authority to make changes, especially if those changes are to the products and services of their colleagues in other teams. Otherwise, work you’re doing will not help.

8.2 Skill Gaps in Teams

The Team Topologies model helps teams understand their interactions, the cognitive load they carry, and the importance of shaping a team's culture to help implement the organization’s strategy faster. The model defines four team types, enabling fast flow from idea to production: Enabling teams, who help a Stream - aligned team to clear obstacles; Complicated Subsystem teams, who provide capabilities that are too complicated to be managed by Stream - aligned teams; Stream - aligned teams, aligned to a flow of change for a single, valuable work product; and Platform teams, grouping together the capabilities to accelerate the delivery by Stream - aligned teams of digital services. The company involved in this case study had implemented a DevOps and platform architecture alongside the Platform team model, and yet Neurodivergent and Early Career hires were still finding it hard to break into customer - facing and technical solution architect roles.

Why is that? It turns out that the customer - facing Development team was expecting a level of expertise on the technical solutions provided that was unrealistic at that level

and not in alignment with the idea of any platform. This was generating a higher than expected turnover of Early Career and Neurodivergent hires from Technical Solution Architect customer - facing roles. These hires were considered to be in learning hubs; they were picking up the processes and systems, the expectations of the role, and starting to gain the technical knowledge necessary for the post, but the company was not applying the appropriate timeframes or an appropriate onboarding process for these roles. There was no mapped knowledge creation plan to support role incumbents through the transitionary period. Given the culture of the organization and the reluctance to embrace more psychometrically aligned automated recruitment tools, it became clear that if the reluctance to create a more equitable onboarding/knowledge creation plan continued, the company was always going to struggle.

9. Case Studies of Successful SRE Adoption

This section outlines some real world successful adoption of SRE and DevOps. While no tool, training, certification, or concept push can guarantee successful DevOps adoption, previous experience from other teams and companies can provide inspiration, pointers to potential pitfalls, models for success, and navigation aid for the DevOps journey.

9.1. Tech Giants' Approaches Google was the pioneering company to brand its Site Operations and Software Engineering collaboration roles as Site Reliability Engineering. Watching the brilliant pages at SRE: the Book is an inspiration to any operationsing or development team – DevOps or not! Netflix is famous for its resilience engineering. The company adapted the Chaos Monkey tool from its earliest service disruptions using the Netflix platform in its home to foster resilience through chaos experiments at scale. The company compiled its engineering approach to diversification, experimentation, observability, availability, and scaling latency. Facebook has many teams involved with production operations, reliability, and security – but it calls none of them site reliability engineering. Instead they adapt SRE practices to their many - hats rotating on - call support teams that reply to alerts and check the reliability dashboards. The company publishes many technical posts through their engineering blog, including its artificial intelligence - driven platform for managing infrastructure.

9.2. Startups Implementing SRE On the developer side, Github argues for embracing SRE as a multiple stakeholder approach to production. Pinterest hired SRE teams rooted in software engineering and coding that collaborate with developers in all areas of reliability – from SLIs to testing in production, and together they onboard on - call responsibilities. Salesforce discusses embracing Continuous Delivery and DevOps principles, practices, and tools while still maturing as a company with site reliability engineering inside a hybrid - cloud model. Workable, a simpler SRE implementation rooting developers on product engineering and operations has adapted a lighter touch with success with all dev team hands on deck during outages.

9.1 Tech Giants' Approaches

Before many of us even used the Internet, companies were already leveraging thousands of servers in datacenters to deliver services to customers all over the world. From workflow search to Internet search, ads delivery, and many other services besides, companies had been scaling their infrastructures for over a decade before we started thinking about building our own cloud infrastructures. Over those years, they had to solve many difficult problems—like creating a system that could withstand a specifically planned Global Wide Area Network outage while continuing to serve requests—that we now can ingest as best practices codified in books and standards. One company became one of the first to produce unified distributed fault tolerance and load balancing solutions to solve such challenges and evolved all of its services towards SLO - driven architectures. In doing so, it created a formalized role through which to operate its services.

9.2 Startups Implementing SRE

Just like bigger enterprises, a number of startups explore the opportunities in shaping their "post - MVP" web services and related/extended mission critical services upon the solid grounds of SRE. Unlike bigger companies that have SRE as a mature established team or even a number of matured specialized teams, smaller startups embrace the philosophy and outline presented by the SRE discipline in less than a few years of their journey. Nevertheless, the impact is extremely significant and may appear in the following forms:

10. Tools and Technologies Supporting SRE

All the standard processes or practices of Site Reliability Engineering outlined above require the implementation of some operational tool or technology to support it; otherwise, it'd be just a set of abstract ideas with little concrete, actionable guidance. In this section, we outline the existing or available tools and technologies supporting some of the important key practices of SRE.

10.1 Monitoring Tools

The first principle of any Site Reliability Engineering team is to instrument the software or product you are operating for visibility into key parameters and monitoring data. Thus, the first duty of the SRE team is to establish extensive monitoring for a product with systems that aid the product owners and developers in understanding what the product is doing, easy to bring up new instances of existing services, and make it easy to scale it up. Use a common set of tools and packages for infrastructure management and monitoring to minimize knowledge transfer and make the organization less brittle.

10.2 Monitoring Tools

Standard automated and manual monitoring tools are critical to the success of SREs. To monitor task completion times and system features that are critical to user happiness, synoptic views of those properties via dashboards and flat -

file logs are the obvious starting point for SRE monitoring technology.

Tools are not true monitoring tools, in that they do not allow the user to specify arbitrary metrics of interest for arbitrary test intervals, nor do they automatically trigger asynchronous alerts. Monitors based on these tools are often augmented by a combination of cronjobs scheduled on remote machines, simple time - interval polling of availability and transactional service properties, agent - programmed counters or submissions tied to service events, log - file scanning, and traffic and performance accounting.

A more elegant model of monitoring permits an arbitrary software event counter and interval time accumulation in an easily accessible bucket or accumulator. Time intervals and event counter resets would be triggered by system events or be administratively configured. Both centralized and distributed monitoring architectures have their benefits and costs. In a centralized architecture, buckets for all monitored events are hosted on a single server; alerting and monitoring policy commands and metric queries have no network latency. In a distributed architecture, program - generated bucket updates are forwarded by machines as service events occur to a few collector servers, where queries and alerts are processed.

While there is no single tool that is appropriate for monitoring all metrics of interest to an SRE, there are two tools that fill large niches: a distributed monitoring tool designed for monitoring clusters. It's designed to do one thing extremely well: to show the real - time state of a cluster's qualities. It maintains six wide - area ganglia, a general - time - scale mosaic of heavy extractions from many clusters, and a static international client.

10.2 Incident Management Software

SRE is concerned with the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of services. While there is a significant amount of effort needed to prevent incidents, by thoroughly addressing each of these areas, no matter how much effort is made, failures will eventually occur. In reality, in these complex modern systems, service outages are inevitable. As a consequence, one of the critical responsibilities of an SRE is an incident response. Incident management tools and technologies exist to minimize the impact of outages when they do occur.

11. Metrics for Measuring SRE Success

While metrics monitoring should help give SREs and those dependent on service health a baseline understanding of service health, there are also a smaller number of metrics that are geared toward assessing the performance and success of the SRE function itself. These metrics can be useful in answering the question: “How well is our SRE team doing?” They allow senior management and the SRE team to assess whether the SRE team’s efforts are aligned with the needs of the organization and if the team has the resources and support necessary to be successful. These metrics also help to determine whether to phase in SRE gradually or to go for break and jump in with both feet. These metrics are also helpful for individual engineers trying to answer the question: “Can an SRE team succeed here?”

11.1. Key Performance Indicators (KPIs) Different SRE teams may have special indicators that reflect the unique circumstances of their environment. However, the most important of these Key Performance Indicator (KPI) categories are:

Balance between effort spent on new features and maintenance: An initial goal is to decrease the amount of effort spent on maintenance. This is typically depicted as a percentage of engineering effort going into maintenance, as opposed to producing new features. State (or age) of the known problems list: Most mature organizations will have a “known problems” list. This is an active list of defects or requested enhancements that users have pointed out that need to be rectified. The length and age of this list is an indicator of the responsiveness of the SRE team. “Heroics” required: In the early evolution of a service, “heroics, ” or periods of frantic activity to resolve a service outage, may be necessary. This is indicative of an infrastructure that is in an unstable state. Having to drop everything and focus on fixing a service for prolonged periods of time may be indicative of a service that is screaming, “Please help, I need your attention!” SRE teams ideally want the number of heroic incidents to go down over time, indicating that they are doing a good job.

11.1 Key Performance Indicators (KPIs)

Someone once remarked, "What gets measured, gets done. " The success of the SRE approach relies on assuming responsibility for the availability of a service and its performance as a business capacity. A lack of attention to and improvement of these characteristics, particularly as the organic growth of a service continues, would put the business at risk. Therefore, measuring these business capabilities is essential. Moreover, it's critical to make sure service owners possess insight into their service's performance and to support them in meeting the expected availability and performance levels. Just as SLOs are fuel for the SRE engine, Key Performance Indicators (KPIs) are SLOs' older and, some would say, less powerful relatives.

11.2 Feedback Loops

While KPIs, SLIs, and SLOs provide a far clearer mechanism for aligning specific efforts to both company - wide objectives as well as day - to - day engineering efforts, the real - time push - pull feedback mechanism also provides a powerful way of observing and reacting to the state of complex systems in operation. In a constantly changing environment, there is an on - going need to ask some basic questions: How closely are we operating within our defined SLIs/SLOs? When was the last time we operated close to that boundary? If we have changed our code, architecture, cloud provider, or host configuration, what effect has it had on our operating characteristics? If we didn't make those changes with the intention of changing those things we'd be best advised to fix them, presumably we made specific changes to make some of these operating characteristics change.

12. Future Trends in SRE

Although Site Reliability Engineering (SRE) is well established in most technological companies, its products, practices, and experiences will continue to evolve. With SRE maturing, many of our support practices merge directly into the general community of development and operations engineers. While new implementations of SRE as a practice still emerge, we're also seeing some organizations invent their own variations and focus, thereby recasting the SRE role back into a specialized operations engineer or site operations specialist.12.1. AI and Machine Learning Integration

One of the main focuses of every Site Reliability Engineer's work is to maintain and develop a stable production environment while keeping the support costs to a minimum. Unfortunately, this is not an easy business when every year (to some extent, every month) your application is getting bigger with new features, and the growing complexity of every SRE - support job doesn't match with the diminishing size of the team. To further cap this geometric growth of the complexity of support tasks for SRE, we try to automate as much as we can, applying the same techniques that were used to solve the original task to the resulting SRE - generated systems. This last step opens a feedback loop, which drives us towards increasing automation of the monitoring - tuning - load - balancing process. However, creating the automated systems is a somewhat laborious process especially for specific jobs like tuning caches or load balancers. In order to further reduce the costs of operations we vision that eventually it will be possible to translate our intention of reducing latency into an acquisition cost of the automated system that can then learn how to perform these specific sub - tasks.

12.2 Evolution of DevOps Practices

Over the past 10 years, there have been many suppositions and presumptions about the definitions of SRE and DevOps, often leading to confusion and delaying agreement on what sets the methodologies apart from each other, and how organizations of varying infrastructures and business needs could benefit from either alone or from both in combination. As the skills and resources shortages for basic platform operations have begun to impact all businesses adopting cloud and microservice - based architectures for their infrastructure, the exploration of the SRE model in organizations previously leveraging pure DevOps principles has gained renewed attention. An inherent misunderstanding is that the methodology of DevOps demands the removal of all traditional platform operations, i. e., lack of SILO awareness and shared accountability. The commonality of all business operations is heavily reliant on platforms, and the mediating service lives of all microservices become a dependency cycle that will only grow in complexity. The evolution of mature and successful DevOps practices has led organizations to invest in microservices modeling, implementing, CI/CD tooling, and operational shared tooling across multi - discipline teams. As initial tooling has enabled rapid product releases, and organizations are now learning from applying these principles for products also applied to internal tooling, platform technology is not a stagnant resource.

13. Conclusion

SRE promises to deliver inputs to design, but it is also an opportunity to build a discipline that understands inputs from design as well.

References

[1] Tabbassum, A., Malik, V., Singh, J., & Surendranath, N. (2024, October). Integrating Site Reliability Engineering Principles with DevSecOps for Enhanced Security Posture. In *2024 International Conference on Intelligent Systems and Advanced Applications (ICISAA)* (pp.1 - 6). IEEE.

[2] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site reliability engineering: how Google runs production systems*. " O'Reilly Media, Inc. ".

[3] Oviedo, E. I. (2021, May). Software Reliability in a DevOps Continuous Integration Environment. In *2021 Annual Reliability and Maintainability Symposium (RAMS)* (pp.1 - 4). IEEE.

[4] Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site reliability engineering: how Google runs production systems*. " O'Reilly Media, Inc. ".

[5] Erich, F. M., Amrit, C., & Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of software: Evolution and Process*, *29* (6), e1885.

[6] Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., & Lang, M. (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. *Service Oriented Computing and Applications*, *11*, 233 - 247.

[7] Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., & Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. *Information and software technology*, *114*, 217 - 230.

[8] Malawski, M., Gajek, A., Zima, A., Balis, B., & Figiela, K. (2020). Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda

and google cloud functions. *Future Generation Computer Systems*, *110*, 502 - 514.

[9] Jamil P (2022) A Survey on Infrastructure as Code (IaC) Security. IEEE Transactions on Dependable and Secure Computing 1.

[10] Farley N (2010) Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison - Wesley Professional.

[11] Beyer P (2016) Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media.

[12] Botvich A (2020) Machine Learning for Resource Provisioning in Cloud Environments. IEEE International Conference on Cloud Engineering (ICEE) 1 - 10.

[13] Chen M (2019) AI for Anomaly Detection in IT Infrastructure. IEEE International Conference on Big Data (Big Data) 5303 - 5307.

[14] Mao Y (2021) Reinforcement Learning for Cloud Resource Allocation. Proceedings of the 2021 ACM Symposium on Cloud Computing 185 - 196.

[15] Patel P (2022) Containerization and Cloud Security: A Survey. IEEE Transactions on Engineering Management 1.

[16] Singh, C., Gaba, N. S., Kaur, M., & Kaur, B. (2019, January). Comparison of different CI/CD tools integrated with cloud platform. In 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence) (pp.7 - 12). IEEE.

[17] Zampetti, F., Geremia, S., Bavota, G., & Di Penta, M. (2021, September). CI/CD pipelines evolution and restructuring: A qualitative and quantitative study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp.471 - 482). IEEE.

[18] Rangnau, T., Buijtenen, R. V., Fransen, F., & Turkmen, F. (2020, October). Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)* (pp.145 - 154). IEEE.

[19] Spence, J. T., Helmreich, R., & Stapp, J. (1973). A short version of the Attitudes toward Women Scale (AWS). *Bulletin of the Psychonomic society*, *2* (4), 219 - 220.

[20] Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., & Lang, M. (2016, May). Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid)* (pp.179 - 182). IEEE.

[21] Holmes, J., Sacchi, L., & Bellazzi, R. (2004). Artificial intelligence in medicine. *Ann R Coll Surg Engl*, *86*, 334 - 8.

[22] Hunt, E. B. (2014). *Artificial intelligence*. Academic Press.

[23] Jiang, Y., Li, X., Luo, H., Yin, S., & Kaynak, O. (2022). Quo vadis artificial intelligence? *Discover Artificial Intelligence*, *2* (1), 4.

[24] Winston, P. H. (1984). *Artificial intelligence*. Addison - Wesley Longman Publishing Co., Inc. .

[25] Holzinger, A., Langs, G., Denk, H., Zatloukal, K., & Müller, H. (2019). Causability and explainability of artificial intelligence in medicine. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *9* (4), e1312.

[26] Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial Intelligence Through.

[27] Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach*. pearson.

[28] Nilsson, N. J. (2009). *The quest for artificial intelligence*. Cambridge University Press.

[29] Nilsson, N. J. (2014). *Principles of artificial intelligence*. Morgan Kaufmann.

[30] Szolovits, P. (2019). Artificial intelligence and medicine. In *Artificial intelligence in medicine* (pp.1 - 19). Routledge.

[31] Minsky, M. (2007). Steps toward artificial intelligence. *Proceedings of the IRE*, *49* (1), 8 - 30.

[32] Berente, N., Gu, B., Recker, J., & Santhanam, R. (2021). Managing artificial intelligence. MIS quarterly, 45 (3).

[33] Minsky, M. (2007). Steps toward artificial intelligence. *Proceedings of the IRE*, *49* (1), 8 - 30.

[34] Negnevitsky, M. (2005). *Artificial intelligence: a guide to intelligent systems*. Pearson education.

[35] Brynjolfsson, E., & Mcafee, A. N. D. R. E. W. (2017). The business of artificial intelligence. *Harvard business review*, *7* (1), 1 - 2.

[36] McCarthy, J., & Hayes, P. J. (1981). Some philosophical problems from the standpoint of artificial intelligence. In *Readings in artificial intelligence* (pp.431 - 450). Morgan Kaufmann.