Site Reliability Engineering: Improving Performance Transparency in a Trading Platform

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Khushi Chawla

Fall, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Brianna Morrison, Department of Computer Science

Abstract

A global New York-based financial software company's trading system's data platform lacked the performance transparency necessary to inform decisions on resource allocation and outage response. The data platform is comprised of multiple components, called domains or services, each of which supports data storage functionality. To increase transparency, I logged and aggregated metrics from each domain in the system where our engineering users manage their domains. I published metrics to an internal telemetry management system, upon which I created a dashboard that queried the storage and displayed the quantity, type, time, and frequency of requests. I then incorporated the dashboard in a client-facing web portal to inform them of their domains' performance. I also integrated this process into the system's automation pipeline to automatically configure the dependencies when a new domain is created. The solution shed light on spikes in response time and potential causes. It also increased confidence in a newly created domain that the portfolio and risk team was rolling out to customers. Future work involves creating alarms to alert the domain user teams when a domain has breached performance goals and publishing end-to-end performance data (distributed trace) to the metrics management portal.

1. Introduction

Imagine getting a call at 3am from an Asian client, furious that they lost millions of dollars on a trade that took too long to execute. You are the software engineer on call and upon further investigation, you find out that because there were so many trades happening at the same time, the service handling the trades significantly slowed down. In the world of stock markets, such delays can make a huge difference. This is the problem that my project attempted to solve: add more metrics to track domain performance to inform teams in case of service failures and/or to automatically scale up resources to reduce delays and outages.

2. Related Works

In order to comply with industry accepted best practices, I researched the standard for site reliability engineering (SRE). Created by Google, SRE is an approach to software engineering to create highly reliable and scalable systems. According to Red Hat (2020), SRE helps system administrators and developers manage large systems through code. Following such a framework could allow me to achieve my goals of adding more transparency to the growing data platform.

Incorporating Site Reliability Engineering (SRE) guidelines, which state that services should perform above an established threshold (Hall, 2021), requires publishing metrics to evaluate how well the domains are achieving goals. These thresholds are called Service Level Objectives, or SLOs and are associated with performance goals specified by the client, called Service Level Agreements, or SLAs (Beyer et al., 2016).

According to Beyer, et al. (2016), a variety of classes of objectives that fall under the umbrella of SRE, the biggest three being speed, load, and accuracy. Ensuring that the associated Service Level Metrics, or SLMs, of latency, throughput and error rate of requests are performing better than our thresholds is vital in keeping a system running efficiently. To achieve this, I would need to create a pipeline to publish latency, throughput, and error rate metrics to a telemetry management system.

The other crucial element of SRE is making clients aware of how well metrics are achieving goals so they can take action to adjust resources and allocation accordingly (Hall, 2021). To follow these guidelines, I

would need to display the metrics in a client-facing platform.

3. Process Design

First, the existing system architecture will be discussed my functionality will be built upon it. Next, the I will establish the design requirements that the system will need to fulfill. Lastly, I will outline the design and implementation of the system I created to satisfy the requirements.

3.1 Review of System Architecture

The system that I worked with was created to maintain consistency across all elements of a trade using a data platform to standardize data storage and formatting for teams that utilize business logic. Rather than other teams individually handling data storage for their business logic and having to constantly communicate with each other to ensure that data is the same across different functionalities, they send their data to my team's data platform via the functionality's specified domain. The data platform then persists the data in such a way that when another team queries it for different functionality, they have the most recent updates from other domains. Additionally, because teams expand their functionality to create new domains, the domain creation pipeline needs to be automated. This ensures that the end-to-end process of creating a domain is standardized for all newly created domains and pre-existing domains.

3.2 Design Requirements

One of the issues with this system is that if a domain is slow or goes down, it affects the data retrieval and storage across the trading platform. Because of this, my team and the teams we support needed a way to see how long requests to the services are and how many requests are resulting in errors.

3.3 System Design and Implementation

Due to non-disclosure agreements, I cannot discuss some elements of the design and implementation of the system as it is built on internal software. However, the architecture designed and implemented was derived from work done by other teams and coding best practices. The metric tracking system was first implemented in two domains as a proof of concept, or POC, then incorporated into the platform's automation pipeline to get automatically configured when a new domain is made.

In order for our client teams to make resource allocation decisions to achieve their SLAs, they needed transparency regarding the speed, load and accuracy of our system. Given the scope of the project, I focused on publishing speed and load metrics from the methods associated with requests in the two POC domains. Using a Java annotation built into the framework that the domains were built upon, every time a request was made to a POC domain, the min, median, 75 percentile, 90 percentile, 95 percentile, 99 percentile and max response times and a request tally were published for a metrics collector to pick up. I used tags in the annotations to allow for filtering during the metrics collection and monitoring phases.

The runtime metrics collector was configured to automatically pick up messages with response time and request count from the domains when a request was made. Using the tags associated with a metric it could be aggregated and stored to a namespace associated with the domain from which it came. These metrics could now be retrieved by querying the metric store for visualizations and statistic generation.

Grafana is an open-source monitoring and visualization platform used to view graphs of the SLMs over time and for the different domains. I created a generic dashboard which allowed the user to select a domain and view associated response time

and throughput information by auto populating the graphs with templated requests to the metric store. Due to the nature of Grafana and the fact that metrics were retrieved using queries, the user could see live updates of averages and time-series plots when new requests were made. Additionally, I added filtering using the tags associated with metrics to allow the user to see information relating to specific use cases, times, resources, and end users.

For clients to be able to see these visualizations and statistics, I embedded the Grafana dashboard in our client web application, where clients can manage everything relating to their domains and make developer requests for changes and/or new domains. I also directly queried the metric store from the web-app to display high level metrics to give a snapshot of the overall health of the system. This allows clients to monitor the system in an easily readable and digestible manner, then navigate to the Grafana dashboard to see more detailed breakdowns.

After the proof-of-concept implementation was approved, this system needed to be integrated into the automation pipeline. To do so, metrics publication functionality was added to the code generation scripts that are run when a new domain is created to ensure that all new domains supported the SLO monitoring functionality. Additionally, new databases were created and updated to store configurations relating to domains and their metrics. The Grafana dashboard and web-app were dynamically updated to display metrics only for domains that were publishing them.

4. Outcomes

The system is still in the process of being implemented and used, so the full scale of its results is difficult to evaluate. However, some of the more immediate results have also shown themselves to be promising. We were able to see consistent spikes for one of the domains in response time in the middle of the night through the time-series graphs on the Grafana dashboard when developers were not actively monitoring systems. This indicated that it was likely being heavily used internationally, as their markets are open at different times, indicating the need to dynamically allocate more resources for that timeframe.

Additionally, we were able to confidently roll out a newer domain for public use with the performance information we collected about internal use through the metric monitoring system. This boosted our confidence and allowed us to make performance promises to our external clients.

Some anticipated outcomes include seeing a decrease in downtime and shorter outages. We also anticipate seeing more teams adopt our platform as we can now concretely demonstrate whether it can satisfy their performance goals.

5. Conclusion

The purpose of this project was to prevent outages in a heavily trafficked internal data managing system in a financial service platform, which could save thousands to millions of dollars in lost revenue. Using concepts from Site Reliability Engineering, I designed a flexible solution that tracks metrics relating to request throughput, response time and error rate which indicate how well the system is performing. This information was then displayed to internal clients to improve performance transparency between the system and its users. The underlying design of this solution can be extended to be used in other systems to increase performance tracking and improve reliability.

6. Future Work

Some future work for this system includes tracking more metrics and traces of

sub-processes to provide more detailed information about domain performance. This includes implementing distributed trace and expanding the SLOs used to load, memory usage, etc. Another improvement would include alarms for when SLOs are breached to immediately alert domain owners of performance failures. This allows for faster responses that could prevent outages and downtime.

7. Acknowledgments

I would like to acknowledge my mentor, manager and team for their continual support and help through this project.

References

Beyer, B., Jones, C., Murphy, N. R. and Petoff, J. (2016). Site reliability engineering. O'Reilly Media, Inc. Retrieved on June 16, 2022, from https://sre.google/sre-book/table-of-contents/

Hall, J. (2021). SRE principles: The 7 fundamental rules. Dotcom Monitor. Retrieved on June 17, 2022, from https://www.dotcom-monitor.com/blog/2021/11/16/sre-principles-the-7-fundamental-rules/

Red Hat. What is SRE? (2020). Retrieved on June 17, 2022 from https://www.redhat.com/en/topics/devops/what-is-sre