Prometheus Monitoring and Alerting

DD2482 Essay

Yiming Ju

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

I certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

1 Introduction

1.1 Monitoring and Alerting

With today's high speed of the application life cycle, code updates are very frequent. To adapt the fast upgrading, there is a complete development process shown in Figure 1.1, including planning, building, testing, deploying, operating, observing and continuous feedback. In the infinite cycle, each part is essential to the development environment. The left side of the infinity loop is considered as product side and the right side is operation side. In the operation side, observability is one of the important concepts, including logging events, tracking, and metrics that can turn large amounts of data into insights that anyone can manipulate and access to provide knowledge about how to resolve exceptions. Monitoring is a subset of observability, and only observable systems can be monitored. With frequent code changes now commonplace, development teams need DevOps monitoring, which provides a comprehensive and real-time view of the production environment.

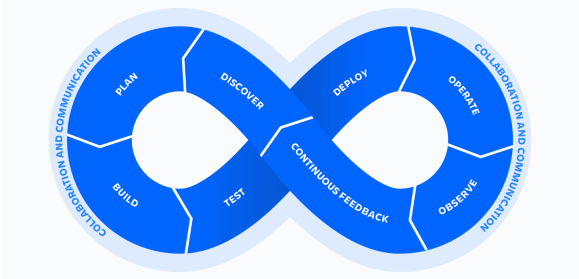

Figure 1.1: Development process

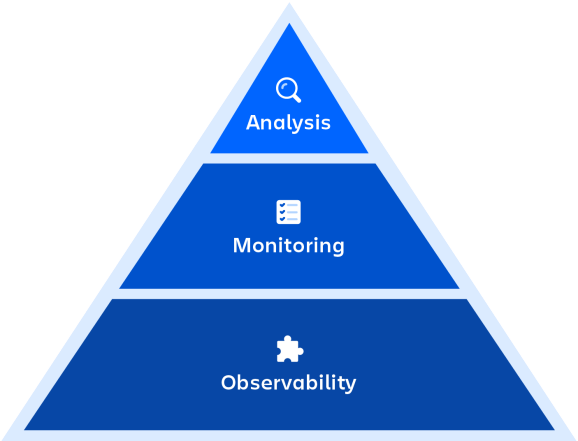

Figure 1.2: Monitoring and observation

In the software development life cycle (SDLC), to ensure more uptime and less downtime, an effective monitoring and alert solution needs to be implemented. The monitoring system enables DevOps engineers to view and understand the system status in real time, so that when a fault occurs, it can be detected and resolved in time. Monitoring and alerting provide visibility into the health of your system, help

monitor trends in performance and usage, and enable you to understand the impact of changes you make. Therefore, they are an important part of DevOps for automated problem detection and alerting, and collection of data for overall system health analysis. Monitoring and alerting and observability are closely related and can support each other, helping DevOps teams achieve more effective software development and deployment in the SDLC.

1.1.1 Monitoring

Monitoring is the process of collecting, aggregating, and analyzing the values obtained about the health and performance of your system, to improve awareness of your components' characteristics and behavior [7]. The primary responsibility of monitoring system is to accept and store incoming and historical data. While it is useful to represent values at the current point in time, it is almost always more helpful to view these numbers in relation to past values to provide context about changes and trends. This means that monitoring systems should be able to manage data over a period of time, which may involve sampling or aggregating old data. Secondly, It often needs to visualize the data. Humans are able to better understanding the data through individual values or tables, which is in a meaningful way. Monitoring systems often use configurable graphs and dashboards to ease complex variables or changes within a system. Finally it should help engineers find the correlation among various data.

1.1.2 Alerting

While monitoring systems are incredibly useful for active interpretation and investigation, one of the primary benefits of a complete monitoring system is letting administrators disengage from the system [7]. The alert itself should contain information about what is wrong and where to go to find additional information. The individual responding to the alert can then use the monitoring system and associated tooling like log files to investigate the cause of the problem and implementing a mitigation strategy.

1.2 Promethues

Prometheus is a popular open-source monitoring system and time series database written in Go. It was inspired by Google's Borgmon monitoring system and started

in 2012 by ex-Googlers working in Soundcloud as an open source project. Then it was publically launched in early 2015 [3]. It features a multi-dimensional data model, a flexible query language, and integrates aspects all the way from client-side instrumentation to alerting [4].

1.2.1 Metrics

Metrics are a fundamental concept in Prometheus, which are numeric measurements [1]. A simple case is that when the number of requests to the applications are high, a request count metric is needed to discover and analyze the problem. There are four types of metrics, counter, gauge, histogram, and summary. Counter metric is used to track things like the number of requests, completed or uncompleted tasks. Gauge metric is used for recording a numerical value like memory usage or queue size. Histogram is often used in the case of distribution of values in a set of samples, illustrating the quantiles, such as average and median value. The last one, summary, keeps track of the total sum and count of observations and then calculates percentiles over a fixed period of time.

1.2.2 Time series

Time series is another essential concept in Prometheus, which means that changes are recorded over time [1]. Prometheus stores time series data in a compressed and indexed format, which allows for efficient querying and aggregation of large amounts of data.

2 Architecture of Prometheus

The implementation architecture of Prometheus is not very complicated. In brief, it includes data collection, data processing, visual display, and then data analysis for alarm processing. But what makes it valuable is that it provides a whole set of viable solutions and an entire ecosystem.

In order for the services to be observed by Prometheus, the services must be able to provide data to Prometheus. Prometheus can collect data from services with an HTTP address you open. Prometheus also has many discovery features, so it can automatically detect your services and start monitoring them.

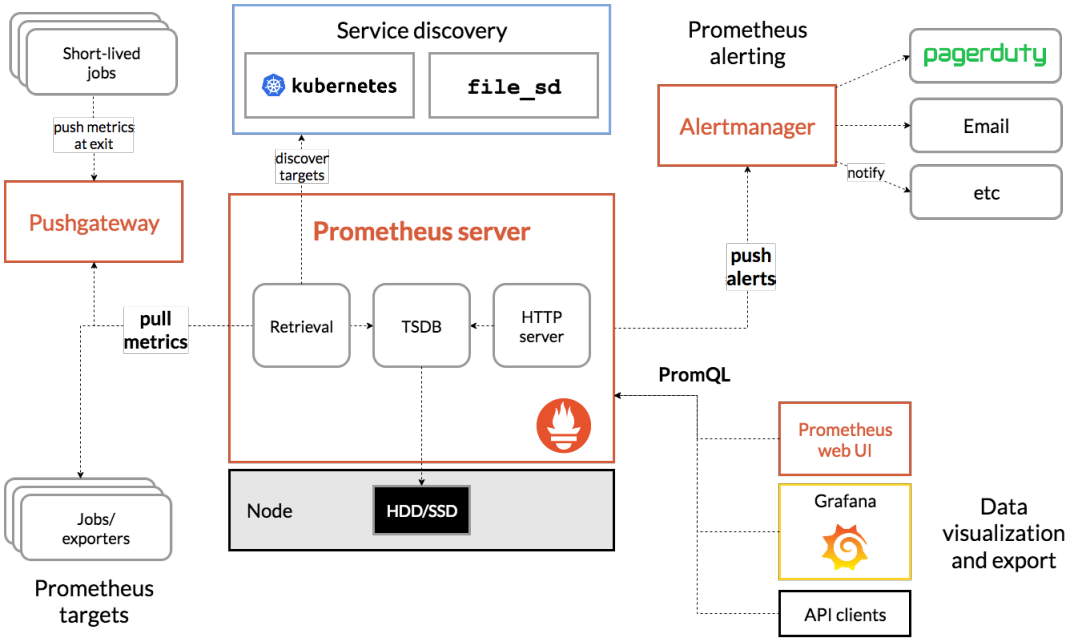

Figure 2.1: Architecture of Prometheus [1]

Main components in Prometheus Architecture [8] are shown in Figure 2.1

- Prometheus server: Prometheus Server is the core part of the Prometheus component, responsible for collecting multi-dimensional metrics in time series, querying and analyzing monitoring data.
- Push gateway: Prometheus Gateway is an intermediary source used for collecting metrics from endpoints that are unable to be scraped directly by the Prometheus server, like restricted network access and the need for authentication. The exporter is a third-party tool that can help fetch measures when they cannot be extracted directly.
- Alert manager: The alert manager is a component that configures alarm rules and handles alerts generated by the Prometheus server. If the rules are met, the alarm will be pushed to the Alert Manager for alarm processing. It can route alerts to different notification channels, such as email, Slack, or PagerDuty.
- Prometheus Targets: Prometheus targets represent how Prometheus extracts metrics from a different resource. In most of time, Prometheus obtains metrics through service discovery directly, such as Kubernetes. But for unexposed services, exporters are used to extract data and convert them into Prometheus format. For example,

(a) Hardware: Node/system

(b) HTTP: HAProxy, NGINX, Apache.

(c) APIs: Github, Docker Hub.

(d) Databases: MySQL, Elasticsearch.

(e) Messaging systems: RabbitMQ, Kafka.

- Data visualization and export: Users can use the Prometheus query language (PromQL) to select and aggregate existing time series data in real time. The Prometheus Expression Browser has the ability to display query results in either a graphical or tabular format. Additionally, the data can be integrated with external visualization tools through the HTTP API.

  Examples for PromQL:

  (a) CPU Usage

  ```
  sum by (cpu)(node_cpu_seconds_total{mode!="idle"})
  ```

  A counter metric is used here to count the number of seconds that the CPU has run from beginning to now. The sum function is to add the usage of all CPU modes. Additional, there are several modes such as iowait, idle, user, and system. Then if you want to view the rate of increase, the following command works

  ```
  (sum by (cpu)(rate(node_cpu_seconds_total{mode!="idle"
    }[5m]))*100
  ```

As illustrated above, Prometheus operates through a multi-step process. First, the Prometheus server regularly pulls data from targets that have been statically configured or dynamically discovered through services. When the newly pulled data exceeds the memory buffer size configured in Prometheus, the data is persisted to disk and can also be remotely persisted to the cloud. Prometheus then displays the data through PromQL, API, Console, and other visualization components such as Grafana and Promdash. Rules can be configured in Prometheus, which regularly queries data, and when the conditions are triggered, alerts are pushed to the configured Alertmanager. When Alertmanager receives an alert, it aggregates, deduplicates, reduces noise, and finally issues a warning based on its configuration.

3 Advantages and Features

Prometheus adopts a multi-dimensional data model that resembles a database, where time series data can be identified by metric name and key-value pairs. This data model allows for flexible querying and analysis through the use of PromQL. PromQL enables users to slice and dice data across different dimensions, and apply subqueries, functions, and operators to metrics data. Additionally, it offers the ability to filter and group data by labels, as well as use regular expressions for improved matching and filtering [2].

Another primary objective of Prometheus is operational simplicity [6]. Each server node is autonomous in Prometheus, which is different from some other monitoring and alerting tools which rely on a distributed storage system. On the one hand, the model using a single node is easier to operate. Prometheus, on the other hand, has a limit number of metrics that can be monitored by Prometheus. This feature makes it more suitable for short-lived jobs, which is also demonstrated in Figure 2.1.

Prometheus also offers integration with other tools. Its service discovery capabilities are tightly integrated with Kubernetes. With service discovery, Prometheus automatically finds all the services running in the Kubernetes cluster and extracts metrics from their Prometheus endpoints. It also integrates with various other tools, such as Grafana for visualization and Alertmanager for alerting. These integrations make it easy for developers to use Prometheus, allowing them to monitor and warn their applications more effectively.

4 Extension

4.1 A Scalable Prometheus Architecture

As mentioned in 2 and 3, Prometheus runs on a single server and it has limited storage. Thus it's necessary to find a solution to ensure availability without reducing the number of metrics. [5] introduces a way to scale Prometheus, which is called Federation. It means one Prometheus server can scrape time-series data from another Prometheus server. One type of Federation is cross-site. As shown in Figure 4.1, there are several sibling Prometheus servers. Each server can scrape metrics from a subset of the target services and then each sibling server will have a copy of the data pulled from the others.

In this way, the Prometheus server is capable of handling much larger amounts of data. Conclusively, a single Prometheus server scrapes data from each of the shards and aggregates it in one place.

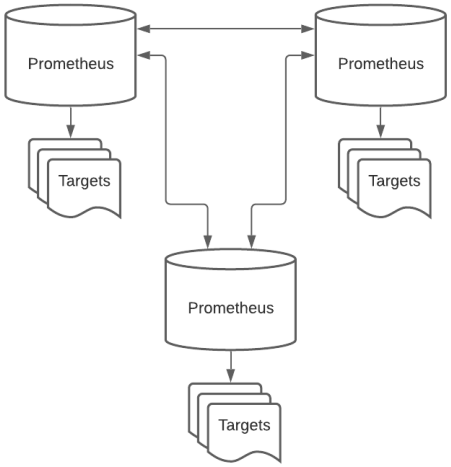

Figure 4.1: Cross-service Federation Illustration

4.2 Remote Storage

Although federation can solve the scalability of Prometheus, it's cumbersome to maintain as demand grows. Another solution is that Prometheus can write to and read from remote storage. It's also one of its advantages that there are many integrations. Figure 4.2 shows the workflow of remote storage. In this way, Prometheus almost has limitless capacity, which overcomes limitations imposed by Prometheus' reliance on local storage.

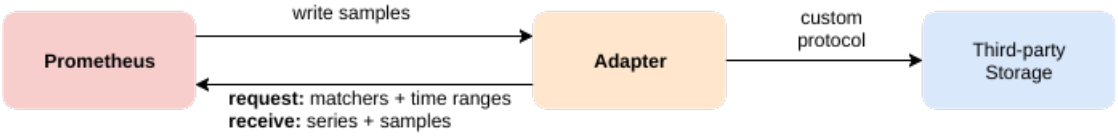

Figure 4.2: Prometheus Storage

5 Reflection

The exploration of Prometheus is a new experience for me. It enables me not only have a more complete understanding to the whole software development life cycle but also have a try to use a monitoring and alerting tool to implement a simple demo.

I only have a vague idea about why and how we monitor the applications before. However with further understanding, I realized that monitoring and alerting is quite essential in the development cycle, which helps developers detect and resolve system issues in time. Software development is only a small part of a software success, it is more about monitoring and maintaining the software to prevent it from bugs or making mistakes. That's the key to software going further.

Out of interest, I followed a beginner tutorial to build a monitoring system with docker, node-exporter, Prometheus and Grafana. Node-exporter provides data to Prometheus, which Prometheus provides to Grafana for data display and analysis. It was very surprising and exciting when I received email alerting once a minute after I closed node-exporter node for testing. Docker is a good choice for you to deploy Prometheus. It's easy to build and deploy Prometheus instances without having to install and configure complex dependencies and environments. Besides, It allows users to manage and maintain multiple instances of Prometheus with a secure and isolated environment.

6 Conclusion

Monitoring and alerting is indispensable in the software development life cycle. Monitoring provides insights into the behavior and health of the systems, while alerting enables teams to detect and address issues before they become critical. Prometheus is a good choice in modern software development, which has the ability to collect and store time-series data. This allows developers to monitor metrics over time and gain the performance trends. Besides, it provides a powerful querying language PromQL, making it easy to analyze and visualize performance data. Although there is a limitation for a single Prometheus server, the scalability problem can be addressed through federation architecture and remote storage. Prometheus will be a great helper in the software development life cycle.

References

[1] https://prometheus.io/docs/introduction/overview/. Accessed: May 6, 2023.

[2] Blogger, MetricFire. *Prometheus vs. ELK*. https://www.metricfire.com/blog/prometheus-vs-elk/. Accessed: May 6, 2023. 2022.

[3] *Prometheus overview*. https://www.slideshare.net/brianbrazil/prometheus-overview. Accessed: May 6, 2023. 2015.

[4] Rabenstein, Björn and Volz, Julius. "Prometheus: A Next-Generation Monitoring System (Talk)". In: Dublin: USENIX Association, May 2015.

[5] Reback, Gedalyah. *How to Build a Scalable Prometheus Architecture*. https://logz.io/blog/devops/prometheus-architecture-at-scale/. Accessed: May 7, 2023.

[6] Ryckbosch, Frederick. *Prometheus monitoring: Pros and cons*. https://devm.io/containers/prometheus-monitoring-pros-cons-136019. Accessed: May 6, 2023. 2017.

[7] SAI, KRISHNA. *DevOps Monitoring*. https://www.atlassian.com/devops/devops-tools/devops-monitoring. Accessed: May 7, 2023.

[8] Sinha, Prince. *Proetheus architecture Explained*. https://scoutapm.com/blog/prometheus-architecture. Accessed: May 6, 2023. 2021.