SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Extension of Prometheus for Scalable and Resilient Black Box Monitoring

Jih-Wei Liang

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Extension of Prometheus for Scalable and Resilient Black Box Monitoring

Erweiterung von Prometheus für skalierbares und belastbares Black-Box-Monitoring

Author:	Jih-Wei Liang
Supervisor:	Michael Gerndt; Prof. Dr.-Ing.
Advisor:	Samuel Torre Vinuesa
Submission Date:	15.03.2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.03.2024 Jih-Wei Liang

Acknowledgments

Abstract

Contents

1 Introduction

Black box monitoring is a system analysis technique used in software testing and network monitoring. It assesses system functionality based on its output in response to certain inputs, without considering its internal mechanisms, thereby treating the system as a "black box" [BW96] [Hie06].

Uniping designed to enable black box monitoring for applications of Open Back End (OBE) and Service Integration (SI) in Amadeus Cloud Service (ACS) was a deprecated project in Amadeus. Recently, demands for black box monitoring in ACS have increased, leading to the reboot of this project. Therefore, the goal for Amadeus is to enhance Uniping in ACS for distributed and reliable Black Box Monitoring and contribute Uniping to the Open Source Community as an innovative solution in the realm of system monitoring.

Uniping is composed of three main components: the Operator, the Scheduler, and the Pinger. The Operator is responsible for handling the Custom Resource Definition (CRD) that defines the service and related information for monitoring. The Scheduler is tasked with scheduling monitoring requests based on the CRD and collecting data from the Pingers. Lastly, the Pinger is a custom worker designed for various monitoring purposes. It sends requests to targets and then returns the necessary data back to the Scheduler.

However, Uniping has several limitations, including a single point of failure, lack of scalability, and limited customizability. Specifically, the Scheduler, which dispatches monitoring requests and collects response data, represents a single point of failure and could potentially become a traffic bottleneck. Furthermore, Uniping can only be deployed to OpenShift Clusters individually, with no common control plane for all Unipings deployed across different clusters. These design and architectural issues pose significant challenges for the rebooted project, especially for other teams within Amadeus who rely on Uniping for reliable black box monitoring of a large number of distributed and cross-cluster services.

Nowadays, there are renowned tools to implement system monitoring such as Prometheus, Datadog, Nagios, and so on. These tools aim to collect a variety of metrics from targets for monitoring computational resources. Generally, they provide an agent to scrape desired data and then wait for the server's request or actively push these data to the responsible server. Most of them feature great professionals in the realm of monitoring, offering enterprise-level services in reaction to kinds of requirements [NVP21].

Uniping aims to provide real-time monitoring, logging, and analysis capabilities. These features enable system administrators to quickly detect and diagnose issues, thereby improving system reliability and performance [JD21]. Furthermore, the proposed improvements to Uniping will offer scalability, availability, and failure tolerance through a redesigning and refactoring of the system architecture. Consequently, the objective of the project is to redesign and implement an open-source black box monitoring system that meets current requirements in the context of distributed architecture.

2 Background

2.1 Black Box Monitoring

Black box monitoring is a well-established approach to system analysis, most commonly applied in software testing and network monitoring. It evaluates a system's functionality based on its responses to given inputs while abstracting away the system's internal workings [JD21]. This technique metaphorically views the system as a "black box" in Figure 2.1, signifying that the internal mechanisms are not visible or accounted for in the evaluation process [Mye04].

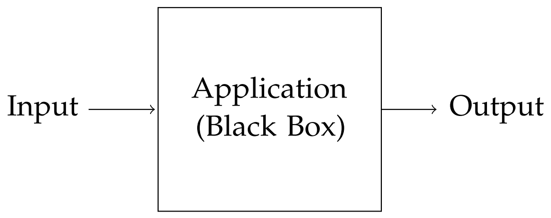

Figure 2.1: An example of a black-box monitoring.

One of the key advantages of black box monitoring is its simplicity and ease of implementation. Since it doesn't require in-depth knowledge of the system internals, it can be quickly deployed across various platforms and systems. This quality fosters wide applicability, accommodating diverse programming languages and system architectures. Furthermore, black box monitoring has a unique focus on the user experience. By simulating user actions and recording the system's responses, it offers a valuable measure of system functionality from a user's perspective. Thus, it is instrumental in ensuring that the system effectively meets the end-users' requirements [Nid12].

However, while the black box approach has its merits, it is not without challenges. One significant limitation is the potential for limited coverage. Since black box monitoring concentrates on the inputs and outputs, it might overlook internal system issues. If a problem doesn't immediately or directly affect the system output, it may remain undetected, only to potentially cause serious issues down the line [SL21]. Moreover, black box monitoring could contribute to inefficiency in troubleshooting. A system failure detected by black box testing can be difficult to diagnose and fix due to the lack of visibility into the system internals. Pinpointing the specific area of the system causing the issue may turn into a challenging endeavor.

Despite existing challenges, the prospects for black box monitoring are increasingly optimistic, particularly in our rapidly evolving digital landscape. As customer satisfaction and user experience become paramount in the digital economy, the ability of black box monitoring to accurately reflect system availability and performance from the user's perspective proves invaluable. Moreover, with the advancement of other evolving techniques like eBPF, the scope of black box monitoring continues to expand [NVP21] [BS20]. For instance, eBPF's ability to execute custom code in kernel space can provide deeper insights into system operations, thereby enhancing internal observability [Jon14].

As the IT landscape becomes more complex with microservices and cloud-based applications, understanding the internals of every service can be overwhelming. In such a scenario, black box monitoring's ability to provide a holistic view of system behavior can be highly beneficial. Moreover, in light of growing privacy regulations and data protection measures, black box monitoring's non-invasive approach aligns with the current trends. Focusing on system outputs rather than internals could potentially minimize privacy or data protection concerns.

In conclusion, black box monitoring remains a vital tool for system testing and monitoring, despite its inherent challenges. Its future appears increasingly integrated with AI technologies and aligned with user-centric design principles. Nevertheless, black box monitoring should not be considered a stand-alone solution but rather a component of a comprehensive monitoring strategy that incorporates various techniques to effectively manage system performance.

2.2 Prometheus Agent and Blackbox Exporter

The renowned Prometheus ecosystem provides two important components: the Prometheus Agent and the Blackbox Exporter. These technologies are extensively employed in various enterprises for monitoring applications in testing and production environments. Their widespread adoption is a testament to the reliability and robustness of the Prometheus platform in handling diverse monitoring needs.

The Prometheus Agent is a specialized mode of a standard Prometheus instance aimed at efficient data scraping and remote write as in Figure 2.2. Unlike a full-fledged Prometheus server with functionalities like data storage and querying, the Prometheus Agent is streamlined for specific tasks, making it an ideal choice in distributed systems where resource optimization is crucial. This mode is invaluable when a full Prometheus server setup is unnecessary, facilitating a more streamlined and resource-efficient deployment. It is particularly effective in large-scale environments where managing the overhead and complexity of multiple complete Prometheus instances would be impractical [Prob].

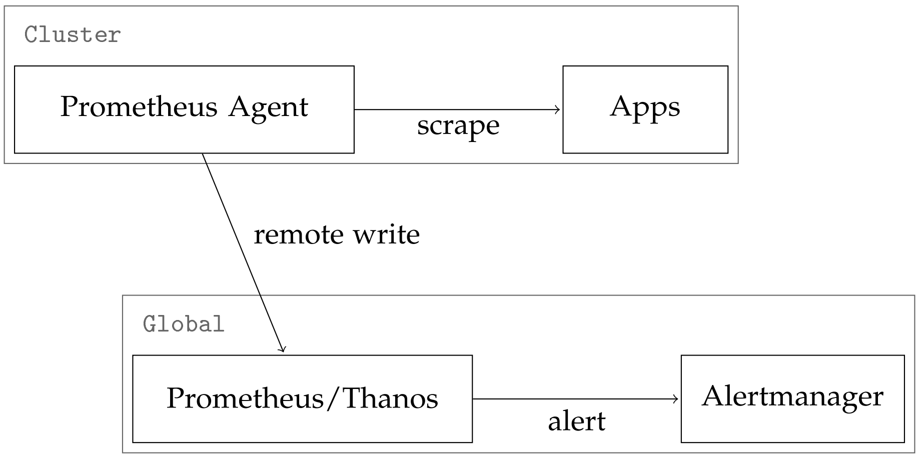

Figure 2.2: Prometheus Agent with scrape and remote write.

The Blackbox Exporter, an essential element of the Prometheus toolkit, is designed to probe external endpoints across multiple protocols, including HTTP/HTTPS, TCP, and ICMP. This capability is fundamental to black box monitoring, enabling the assessment of system health and performance from an external viewpoint without necessitating internal access to the monitored system [Pro23]. The Blackbox Exporter is instrumental in situations where internal monitoring is either not feasible or insufficient, such as in third-party services or in environments where internal metrics are not available or reliable.

Integrating the Prometheus Agent with the Blackbox Exporter fosters a distributed scheduler-executor architecture. This approach allows for a more granular and distributed monitoring framework, which is highly adaptable and can be effectively implemented across various platforms as service (PaaS) clusters with latencies [Prod]. Such an architecture bolsters the scalability and resilience of the monitoring system, rendering it suitable for large-scale and intricate environments. This distributed nature enhances the system's ability to scale and ensures a more resilient monitoring setup capable of withstanding node failures and network partitions [Prob].

Achieving optimal availability and scalability with the Prometheus Agent and Blackbox Exporter needs additional configuration and management. For the Prometheus Agent, employing relabeling is a crucial horizontal scaling or sharding strategy [Proa]. This method distributes the workload across multiple Prometheus Agent instances with a hidden label with hashed value, augmenting the system's capacity to handle substantial data volumes efficiently. Regarding the Blackbox Exporter, scalability is further enhanced by integrating a load balancer with properly configured scaling parameters, ensuring an even distribution of the probing load and maintaining system performance, even under high demand [Prob].

Overall, the active and extensive community surrounding Prometheus plays a significant role in the continuous improvement and reliability of the Prometheus Agent and Blackbox Exporter. This support ensures consistent updates and maintenance, effectively addressing the evolving needs and challenges in the monitoring domain. However, utilizing the Prometheus Agent and Blackbox Exporter to achieve scalability and manageability requires additional configurations and operations. Consequently, automating the scaling and management operations for both the Prometheus Agent and Blackbox Exporter emerges as a critical issue.

2.3 Operator Pattern and Prometheus Operator

The Operator Pattern, defined by the Cloud Native Computing Foundation (CNCF), represents a paradigm shift in Kubernetes, enabling users to automate the maintenance and configuration of applications [Kub]. The Prometheus Operator embodies this pattern, serving as an implementation that addresses the practical needs of deploying and managing Prometheus components. Through reconciliation, the Prometheus Operator aligns the deployment's actual state with the user's desired state, streamlining its lifecycle in Kubernetes [Ope20].

The Operator Pattern extends Kubernetes' native capabilities by introducing Custom Resources (CRs) and controllers. The operator is the essential software extension that utilize the Kubernetes control plane and API to create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user [DW]. They encapsulate operational knowledge, automating the management tasks that would typically require manual operations [CNC].

The Prometheus Operator is tailored to simplify the deployment and management of Prometheus within Kubernetes environments. It enables Kubernetes-native deployment and automated management of Prometheus and its associated monitoring components. Key features of the operator include Kubernetes CRs for deploying and managing Prometheus, Alertmanager, and so on, streamlined deployment configurations for setting up Prometheus essentials like versions and retention policies, and automatic generation of monitoring target configurations. This approach facilitates easy installation and version upgrades, simplifies configuration management, and ensures seamless integration with existing Kubernetes resources [Ope20].

In terms of current advancements in black box monitoring, the Prometheus Operator supports two CRs: Probe and PrometheusAgent. The Probe resource defines monitoring for a set of static targets or ingresses, such as specifying the target for the Blackbox Exporter and the module to be used. On the other hand, PrometheusAgent is responsible for defining a Prometheus agent deployment. This includes configurations like the number of replicas, ScrapeConfigs, and advanced features like sharding [Opeb]. Notably, it incorporates the ProbeSelector feature, which links to the previously defined Probe resource, enhancing its functionality and integration [Opea].

Despite these advancements, a significant gap persists in the management and configuration of the Blackbox Exporter within the Prometheus Operator framework. Specifically, there is no support for using a custom resource definition to manage the Blackbox Exporter. Consequently, users are required to deploy their own instances and modules of the Blackbox Exporter [Pro23]. This limitation underscores the need for more integrated solutions that can simplify the deployment and scaling of the Blackbox Exporter, ensuring it meets the evolving requirements of black box monitoring in complex environments.

In conclusion, while the Prometheus Operator has made significant strides in enhancing the ease and efficiency of deploying Prometheus in Kubernetes environments, there is still room for improvement, particularly in the realm of black box monitoring. Addressing the current limitations in the management of the Blackbox Exporter will be crucial in realizing the full potential of Prometheus as a comprehensive monitoring solution.

3 Overview

3.1 System Goals

3.2 System Overview

3.3 System Workflow

3.4 Integrated GitOps

4 Design

4.1 Section

4.1.1 Subsection

5 Implementation

5.1 Section

5.1.1 Subsection

6 Evaluation

6.1 Section

6.1.1 Subsection

7 Related Work

7.1 Using Prometheus for Black Box Monitoring

The Prometheus Blackbox Exporter is a valuable extension of the Prometheus monitoring system, specifically designed for monitoring external services and endpoints. This tool permits users to probe targets via HTTP, TCP, and ICMP protocols, yielding metrics related to availability, latency, SSL certificate expiration, and DNS resolution, among others. Its configuration options allow for the customization of probe settings, including timeout periods, headers, and probe methods. The metrics gathered are presented in a Prometheus-compatible format, enabling seamless integration with existing Prometheus monitoring setups [Proc] [Pro23].

The Prometheus community is renowned for its high-quality software, a variety of plugins, and robust support, which can be leveraged when creating a Blackbox Monitoring system. According to a presentation by Aaron Wieczorek titled "Spotlight on Cloud: Using Prometheus for Black Box Monitoring," utilizing a Prometheus Blackbox Exporter based architecture for blackbox monitoring is a practical choice for systems already leveraging Prometheus for monitoring and alerting [ORe]. The following figure depicts the setup:

However, there are a few challenges. Prometheus' default data retention period is limited to fourteen days because its primary function is real-time system status monitoring and alert delivery. It is not built to retain large amounts of data for extended periods. Furthermore, Prometheus is not designed for a distributed setup, increasing complexity when configuring and deploying each Blackbox Exporter separately, especially during inevitable configuration changes (see Figure 2) [Proc]. Another critical drawback is that Prometheus acts as a single point of failure, jeopardizing the availability of the monitoring system and risking data loss during a system failure.

In conclusion, implementing black box monitoring with the Prometheus Blackbox Exporter comes with benefits and drawbacks [ORe]. While it is praised for its simplicity, customization, and extensive community support, it has inherent limitations from its design, constraining its ability to fully exploit the power of a distributed architecture. Data retention for analytics is increasingly vital for ensuring the Service Level Agreement (SLA) of private services. Without centralized control over all Blackbox Exporters, configuration management can become unwieldy, and the single point of failure inherent in the design presents a risk of data loss. Therefore, careful consideration of these trade-offs is essential in the design and decision-making process of the project.

7.2 Datadog in the Context of Black Box Monitoring Systems

In the realm of modern IT infrastructure management, monitoring and analytics platforms play an integral role in ensuring system reliability and performance. Among the existing solutions, Datadog has emerged as a prominent player, known for its robust functionalities and capabilities [Datd]. In the context of this project, which aims to design a dedicated and distributed black box monitoring system, understanding Datadog's approach and technology is pivotal.

Datadog offers a unified view of an organization's IT infrastructure by collecting and analyzing data from multiple sources. This approach enables real-time monitoring and troubleshooting, as well as optimization of applications and infrastructure. A standout feature of Datadog is its Synthetic Monitoring, which is particularly applicable for black box monitoring [Datd] [Data]. This feature simulates requests and actions, thereby offering insights into the performance of APIs across various network layers, from backend to frontend [Data] [Datb].

There are two distinct elements of Datadog's Synthetic Monitoring that are of particular interest: API Tests and Browser Tests [Data] [Datb]. API Tests actively probe target services, gathering their statuses and metrics. These tests offer a snapshot of the availability of the monitored services, a crucial factor in any black box monitoring system. In Datadog's implementation, API Tests form the bulk of the monitoring process, thereby highlighting their importance. Browser Tests, on the other hand, execute defined scenarios on target services using a chosen browser. This essentially reproduces the actions of a user, providing experiential feedback about the services. This user-centric approach complements the more technical data gathered by the API Tests, contributing to a more holistic view of the system's health.

When it comes to monitoring availability, it is crucial to decide the position of source endpoints that proceed with monitoring [Datc]. Datadog provides the feature to customize private locations to execute Synthetic Monitoring. By installing Docker containers as private endpoints in desired locations, Synthetic Monitoring, including both API Tests and Browser Tests, can employ these private endpoints to compose its line of departure.

While Datadog's synthetic monitoring approach presents a well-rounded solution, it operates primarily within a centralized model in terms of the agents' side. To elaborate, monitoring targets from a private location requires an installed agent in the same cluster, and the agent could be a single point of failure.

This brings us to the primary focus of our thesis: to explore the feasibility and benefits of a distributed black box monitoring system. As businesses scale and become increasingly distributed, there may be unique advantages to employing a monitoring system that mirrors this distributed structure. By building on the strengths of Datadog's approach, while addressing its potential limitations in distributed environments, we aim to advance the field of black box monitoring.

8 Summary and Conclusion

8.1 Section

8.1.1 Subsection

9 Future Work

9.1 Section

9.1.1 Subsection

Abbreviations

ACS	Amadeus Cloud Service
CNCF	Cloud Native Computing Foundation
CRs	Custom Resources
CRD	Custom Resource Definition
OBE	Open Back End
SI	Service Integration
SLA	Service Level Agreement

List of Figures

List of Tables

Bibliography

[BS20] R. Brondolin and M. D. Santambrogio. "A Black-box Monitoring Approach to Measure Microservices Runtime Performance." In: *ACM Transactions on Architecture and Code Optimization* 17.4 (Nov. 10, 2020), 34:1–34:26. ISSN: 1544-3566. DOI: 10.1145/3418899.

[BW96] B. Beizer and J. Wiley. "Black Box Testing: Techniques for Functional Testing of Software and Systems." In: *IEEE Software* 13.5 (Sept. 1996), pp. 98–. ISSN: 1937-4194. DOI: 10.1109/MS.1996.536464.

[CNC] CNCF. *CNCF Operator White Paper - Final Version*. GitHub. URL: https://github.com/cncf/tag-app-delivery/blob/163962c4b1cd70d085107fc579e3e04c2e14d59c/operator-wg/whitepaper/Operator-WhitePaper_v1-0.md (visited on 12/28/2023).

[Data] Datadog. *API Tests*. Datadog Infrastructure and Application Monitoring. URL: https://docs.datadoghq.com/synthetics/api_tests/ (visited on 05/21/2023).

[Datb] Datadog. *Browser Tests*. Datadog Infrastructure and Application Monitoring. URL: https://docs.datadoghq.com/synthetics/browser_tests/ (visited on 05/21/2023).

[Datc] Datadog. *Run Synthetic Tests from Private Locations*. Datadog Infrastructure and Application Monitoring. URL: https://docs.datadoghq.com/synthetics/private_locations/ (visited on 05/21/2023).

[Datd] Datadog. *Synthetic Monitoring*. Datadog Infrastructure and Application Monitoring. URL: https://docs.datadoghq.com/synthetics/ (visited on 05/21/2023).

[DW] J. Dobies and J. Wood. *Kubernetes Operators*. ISBN: 978-1-4920-4803-9.

[Hie06] R. M. Hierons. "Software Testing Foundations: A Study Guide for the Certified Tester Exam. By Andreas Spillner, Tilo Linz and Hans Schaefer. Published by Dpunkt.Verlag, Heidelberg, Germany, 2006. ISBN: 3-89864-363-8, Pp 266: Book Reviews." In: *Software Testing, Verification & Reliability* 16.4 (Dec. 1, 2006), pp. 289–290. ISSN: 0960-0833.

[JD21] P. Jorgensen and B. DeVries. *Software Testing: A Craftsman's Approach*. May 31, 2021. 7-8. ISBN: 978-1-00-316844-7. DOI: `10.1201/9781003168447`.

[Jon14] C. Jonathan. *BPF: The Universal in-Kernel Virtual Machine [LWN.Net]*. May 21, 2014. URL: `https://lwn.net/Articles/599755/` (visited on 06/01/2023).

[Kub] Kubernetes. *Operator Pattern*. Kubernetes. URL: `https://kubernetes.io/docs/concepts/extend-kubernetes/operator/` (visited on 12/28/2023).

[Mye04] G. J. Myers. *The Art of Software Testing*. Newark, UNITED STATES: Wiley, 2004. 9-11. ISBN: 978-1-280-34616-3.

[Nid12] S. Nidhra. "Black Box and White Box Testing Techniques - A Literature Review." In: *International Journal of Embedded Systems and Applications* 2 (June 30, 2012), pp. 29–50. DOI: `10.5121/ijesa.2012.2204`.

[NVP21] F. Neves, R. Vilaça, and J. Pereira. "Detailed Black-Box Monitoring of Distributed Systems." In: *ACM SIGAPP Applied Computing Review* 21 (Mar. 1, 2021), pp. 24–36. DOI: `10.1145/3477133.3477135`.

[Opea] P. Operator. *API Reference*. Prometheus Operator. URL: `https://prometheus-operator.dev/docs/operator/api/` (visited on 12/28/2023).

[Opeb] P. Operator. *Prometheus Agent Support*. GitHub. URL: `https://github.com/prometheus-operator/prometheus-operator/blob/main/Documentation/designs/prometheus-agent.md` (visited on 12/29/2023).

[Ope20] P. Operator. *Introduction*. Prometheus Operator. Oct. 6, 2020. URL: `https://prometheus-operator.dev/docs/prologue/introduction/` (visited on 12/28/2023).

[ORe] O'Reilly. *Spotlight on Cloud: Using Prometheus for Black Box Monitoring with Aaron Wieczorek*. O'Reilly Online Learning. URL: `https://learning.oreilly.com/videos/spotlight-on-cloud/0636920360216/0636920360216-video328883/` (visited on 05/21/2023).

[Proa] Prometheus. *How Relabeling in Prometheus Works*. Grafana Labs. URL: `https://grafana.com/blog/2022/03/21/how-relabeling-in-prometheus-works/` (visited on 12/27/2023).

[Prob] Prometheus. *Introducing Prometheus Agent Mode, an Efficient and Cloud-Native Way for Metric Forwarding | Prometheus*. URL: `https://prometheus.io/blog/2021/11/16/agent/` (visited on 12/27/2023).

[Proc] Prometheus. *Overview | Prometheus*. URL: `https://prometheus.io/docs/introduction/overview/` (visited on 05/26/2023).

[Prod] Prometheus. *Understanding and Using the Multi-Target Exporter Pattern | Prometheus*. URL: `https://prometheus.io/docs/guides/multi-target-exporter/` (visited on 12/27/2023).

[Pro23] Prometheus. *Blackbox Exporter*. Prometheus, May 26, 2023.

[SL21] A. Spillner and T. Linz. *Software Testing Foundations, 5th Edition*. dpunkt, 2021. ISBN: 978-1-09-812962-0.