SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing and Building an Active Monitoring Platform with Prometheus Ecosystem

Jih-Wei Liang

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing and Building an Active Monitoring Platform with Prometheus Ecosystem

Entwerfen und Aufbauen einer aktiven Überwachungsplattform mit dem Prometheus-Ökosystem

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

System monitoring is essential in the modern Information Technology (IT) industry. It offers observability to both users and developers, enabling rapid detection and notification of issues. For Amadeus's extensive network of interlinked applications, observability is vital to swiftly identifying, diagnosing, and resolving problems, ensuring our customers receive the highest quality service.

The Amadeus Observability Platform accommodates a variety of monitoring data, including metrics, logs, structured logs, and events. It is built on the Prometheus ecosystem and integrates with Splunk. Additionally, Amadeus actively contributes to the Prometheus ecosystem. A prime example is Perses, a new CNCF project developed and maintained by Amadeus that aims at observability data visualization.

Recently, demands for probing on kinds of applications in Amadeus Cloud Service (ACS) by customized requests, such as those of Hypertext Transfer Protocol (HTTP) or Simple Network Management Protocol (SNMP), have increased, which stemmed from the increasing focus on observability. This kind of monitoring actively sends simulated requests and observes the response without internal knowledge, which could benefit observability with low cost and fast reaction. As part of the overall strategy for rock-solid operations, Amadeus needs this kind of monitoring on scale.

As mentioned above, the tendency brings about the demand for introducing active monitoring or Black Box Monitoring to Amadeus. By definition, active monitoring features the simulation of requests and observation of responses, covering the scope of Black Box monitoring. However, the two monitoring methods have slight differences in concepts. Active monitoring emphasizes proactively watching the reactions to simulated behaviors [Spl23]. On the other side, Black Box Monitoring focuses on ignoring the internal mechanism, only caring about the input and output of the application [Bey+16]. In conclusion, although the two monitoring methods have different names and concepts, they still feature similar behaviors and fit the latest monitoring requirements in Amadeus.

Following alignment meetings with Platform Product Management and Tech Senior Leaders, a consensus was reached regarding the requirements for the final solution in active monitoring. The solution must be highly scalable, reliable, and cloud-ready to meet the needs of a modern IT environment. It should support multiple protocols such as HTTP, Simple Object Access Protocol (SOAP), Representational State Transfer (REST),

Electronic Data Interchange for Administration, Commerce and Transport (EDIFACT), etc. Additionally, the system must be modular to facilitate the easy addition of new protocols.

Additionally, other essential features concerning the Operating Model should be considered. For example, the active monitoring platform must be self-service and fully support "as-code" configurations such as GitOps. Also, it should support deploying probers and selecting their origin and target with an easy interface. Lastly, seamless integration with the Amadeus Observability Platform, including the Event Management stack, is crucial to ensure a cohesive and efficient monitoring ecosystem.

Nowadays, there are renowned tools to implement system monitoring, such as Prometheus, Datadog, Nagios, and so on. These tools aim to collect various 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 professions in monitoring, offering enterprise-level services in reaction to kinds of requirements [NVP21].

This project considers three solutions to align with the Amadeus Observability Platform. The first is a customized application with scheduled probings, probing targets, and a centralized user interface. The second option involves deploying a Prometheus Agent and Blackbox Exporter, accompanied by an additional control plane. The third solution explores using the Prometheus Operator, Open Policy Agent, and GitOps.

The first solution comprises three main components: the Operator, the Scheduler, and the Prober. 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 Probers. Lastly, the Prober is a custom worker designed for various monitoring purposes. It sends target requests and then returns the data required to the Scheduler. The significant advantage of this method is full customization; however, the disadvantages are the heavy maintenance load and the inconsistency with the open-source framework.

The second solution is similar to the first one. The difference is that the scheduler is replaced with the Prometheus Agent, and general probers are replaced with the Blackbox Exporter, except for some customized probers of specialized protocols in Amadeus. With these changes mentioned above, a well-designed open-source framework is employed, decreasing the maintenance load and hidden danger of homemade solutions.

The third solution introduces the open-source implementation for managing different components in the Prometheus ecosystem with the Operator Pattern, the Prometheus Operator. Like the operator in the first solution, the Operator Pattern allows users to

automate configurations and management in container orchestrators like Kubernetes and OpenShift [Kub]. Therefore, the Prometheus Operator helps users deploy and manage the components in the ecosystem, which are needed for active monitoring, achieving sharding and auto scalability without additional effort [Ope20]. On the other hand, users must understand and utilize official CRDs to conduct settings, which brings some learning curve to the promoting.

In selecting the final solution, the third option involving the Prometheus Operator stands out as the preferred choice for several reasons. Firstly, the Prometheus Operator can be seen as an enhanced and official version of the first solution, offering improved compatibility. Additionally, while the second solution requires extra effort to attain scalability and availability, the Prometheus Operator naturally includes automated configuration management and sharding capabilities. Lastly, the Prometheus Operator holds an advantage over the other two solutions regarding the potential for increased support from other Prometheus-related projects.

This thesis is organized as follows: Section 2 provides background on key concepts related to Monitoring, Prometheus, and Operator. Section 3 outlines the design of the active monitoring Platform at Amadeus, including data workflow and system integration with GitOps. Section 4 discusses design criteria and proposed solutions, such as cross-cluster probing and support for customized probers, highlighting their value and design decisions. Section 5 details the implementation process, focusing on integrating the Prometheus Operator into the active monitoring Platform and designing the interface for customized probers. Section 6 presents and evaluates experiments, examining the efficiency and benefits of the proposed solution. Section 7 reviews related work in the field, comparing this thesis to other projects and highlighting distinctions. Section 8 summarizes the implementation and results, underscoring the thesis's significant contributions. Finally, Section 9 suggests potential avenues for future research, exploring promising areas for further investigation.

2 Background

2.1 Active Monitoring and Black Box Monitoring

Monitoring IT systems brings insight into systems in terms of observability, becoming increasingly critical to improving systems. Based on simulation data generated synthetically, active monitoring proactively monitors the performance of networks, applications, and infrastructure [Spl23]. Another well-known monitoring method is black box monitoring. Like active monitoring, black box monitoring evaluates a system's functionality based on its responses to given inputs while abstracting away the system's internal workings [JD21].

Active monitoring, also called synthetic monitoring, mainly identifies potential application issues, including health checks, Application Programming Interface (API) endpoint tests, etc. In addition, active monitoring also includes evaluating the performance of hardware resources and benchmarking the network performance [Spl23]. For example, the Quality of Service (QoS) requirements of the network like latency and bandwidth, the Central Processing Unit (CPU) utilization, and the memory utilization.

Black box monitoring is a well-established approach to system analysis, commonly applied in software testing and network monitoring. In contrast to white box monitoring, which baesd on metrics exposed by the internals of the system, it focuses on external behavior rather than internal metrics [Bey+16]. 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].

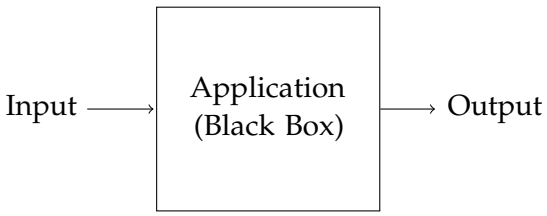

Figure 2.1: An example of a black-box monitoring.

Active monitoring and black box monitoring share some similar features but embody distinct concepts. Both involve simulating requests to target APIs or other endpoints,

such as those of Transmission Control Protocol (TCP), HTTP, or other customized protocols. Specifically, These symptom-oriented techniques mimic user behavior, thereby reflecting real issues [Bey+16]. By definition, active monitoring can encompass black box monitoring but focuses on different aspects. Active monitoring emphasizes proactive check-ups, actively testing and querying systems, while black box monitoring is more of an observational approach, focusing on the outcomes and behaviors of systems without delving into their internal mechanics. In summary, despite of slight difference, both methods are valuable in system monitoring and could contribute to the comprehensive view of system health and performance.

In system monitoring, the "Four Golden Signals" are essential: Latency, Traffic, Errors, and Saturation. Latency signifies response times, differentiating between successful and failed requests. Traffic gauges demand, such as HTTP requests per second. Errors identify the rate of failures, whether explicit, implicit, or policy-driven. Saturation looks at resource utilization and potential bottlenecks [Bey+16]. Bridging these metrics, active monitoring proactively tests systems for performance and reliability, while black box monitoring observes system behavior without internal visibility. The combination enhanced the effectiveness of the "Four Golden Signals," ensuring comprehensive system monitoring and optimal system performance.

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, active monitoring and 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, the non-invasive approach aligns with the current trends. Simulating test inputs and focusing on system outputs rather than internals could minimize privacy or data protection concerns.

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

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 simplified 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 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].

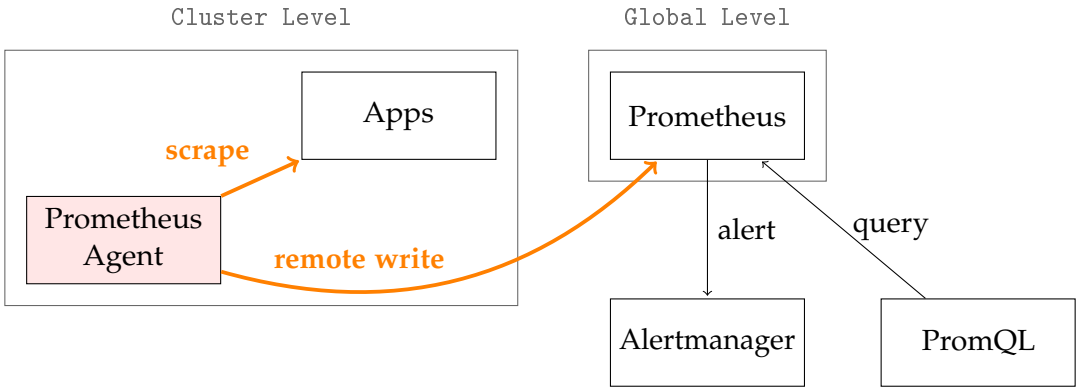

Figure 2.2: Prometheus Agent with scrape and remote write.

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 the Custom Resource (CR) and controllers. The operator, that is, the controller, is the essential software extension that utilizes the Kubernetes control plane and API to create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user [DW]. As shown in Figure 2.3, the controller continuously reconciles the current state with the desired state, encapsulated in custom resources, using a loop to transition watched objects to their target state. In conclusion, the Operator Pattern encapsulates operational knowledge, automating the management tasks typically requiring manual operations [CNC].

The Prometheus Operator is tailored to simplify the deployment and management of Prometheus within Kubernetes environments. It enables Kubernetes-native deployment

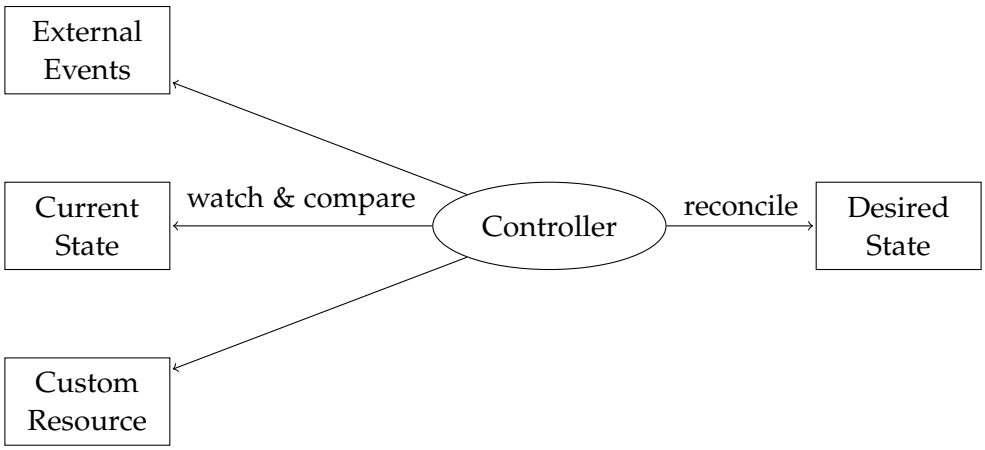

Figure 2.3: Illustration of the Operator Pattern.

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 en-

hancing 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

This project aims to develop a scalable, reliable, and cloud-ready monitoring platform within the Prometheus ecosystem, designed for seamless integration into the Amadeus Observability Platform. To maintain compatibility and streamline integration with the existing platform, it will also support various probers based on custom protocols, including the Transport Cluster-Interconnect-Logic (TCIL) protocol used at Amadeus.

To ensure scalability, the platform should examine the bottleneck and consider the automated horizontal scaling among all components. Specifically, the design must keep all components stateless as much as possible. As for the inevitable stateful components, the maintenance of states should be handled properly for scaling. Meanwhile, distinct strategies like load balancing and sharding should be considered appropriately in reaction to different circumstances.

Reliability decides the trustworthiness of the Active Monitoring Platform, and the critical point to assure reliability is fault tolerance. To elaborate, as a prober deployed responsible for probing targets is down, another prober could take over and continue operating.

For cloud readiness, the design must be compatible with cloud infrastructure. Amadeus operates thousands of applications across numerous OpenShift clusters, necessitating that the Active Monitoring Platform be tailored for cloud platforms. Additionally, artifact versioning is crucial in cloud readiness, offering a centralized and managed system package for cloud deployment and the prerequisite for GitOps.

Overall, the proposed design addresses the demands for active monitoring among thousands of applications deployed and distributed in dozens of clusters. With a user-friendly and efficient solution, the existing Amadeus Observability Platform could seamlessly integrate the active monitoring results and achieve high availability and efficiency.

The current system for active monitoring features lightweight scheduling and probing with an operator managing configurations as CRDs. Substituting the scheduler with the Prometheus Agent is critical, improving efficiency with the remote write feature and decreasing peak requests by spreading requests over a scrape interval. Next, employing the Prometheus Operator breeds further enhancements, including

deployment management, automatic scalability, configuration sharding, etc. In brief, implementing the above two aspects successfully meets Amadeus's active monitoring goals.

3.2 System Overview

The suggested design utilizes container orchestrators such as Kubernetes or OpenShift. At Amadeus, the predominant version is the OpenShift 4.11, typically set up as individual and independent clusters. The subsequent section details the system design shown in Figure 3.1.

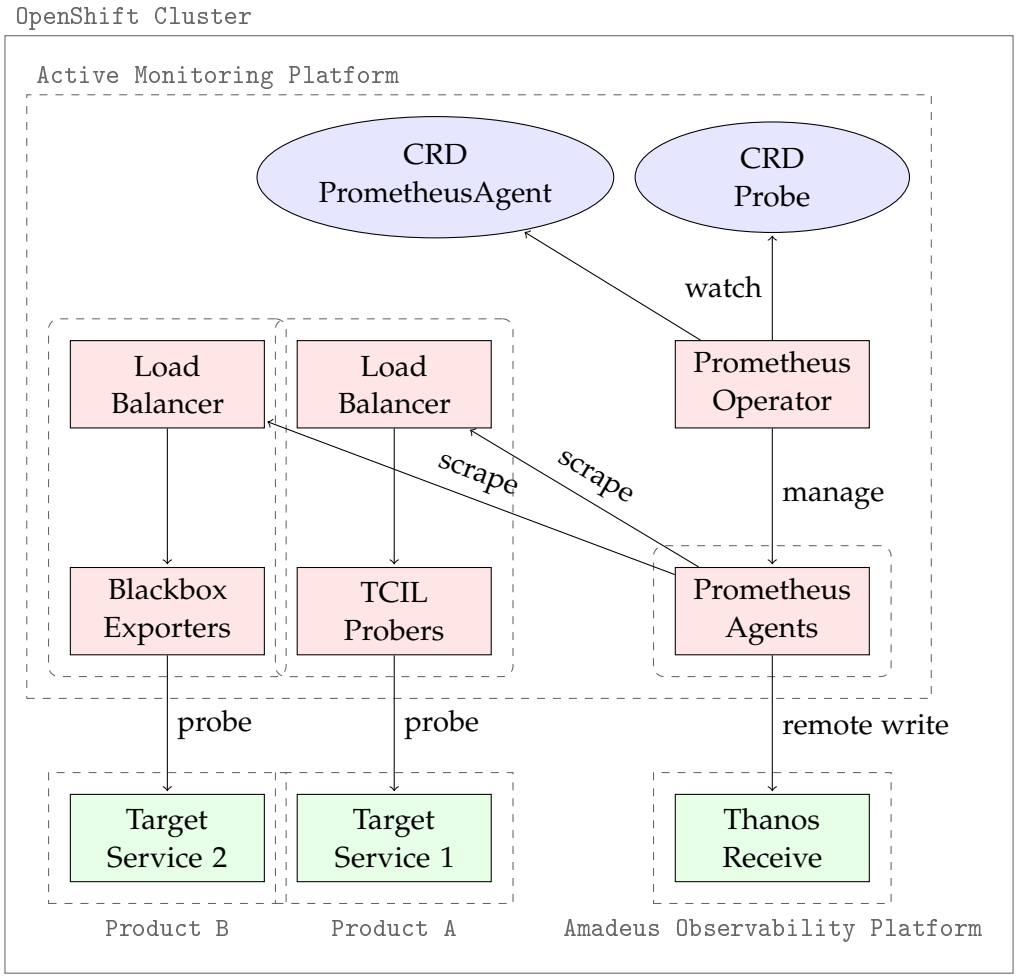

Figure 3.1: System Design with Prometheus Operator, Prometheus Agent and Probers.

Inside the cluster, a slightly modified Prometheus Operator would be installed, responsible for deploying and managing Promenteus Agents and probe configurations. Next, Blackbox Exporters and customized probers like the TCIL Pinger, which is still in development, would be deployed with load balancers, leveraging horizontal scalability for better load distribution.

At first, the deployed Prometheus Operator will monitor the CRDs created for deploying Prometheus Agents and probe configurations. It will maintain its state, offering scaling or sharding as required. Subsequently, the Prometheus Agent will handle scheduling probes and remotely transmit metrics to the Prometheus remote endpoint or Thanos Receive.

Secondly, deploying probers with the load balancer enables the Prometheus Agent to target the same host, leveraging load distribution and enhancing reliability without extra effort. Additionally, with Kubernetes or OpenShift, the inherent scalability and the ingress or load balancer provide valuable and reliable support, significantly boosting this design.

Finally, users with active monitoring requirements across various clusters and namespaces can leverage the official CRD: Probe to customize their specific requests. In summary, this system architecture offers a more efficient, seamlessly integrated, and highly available solution for active monitoring for the Amadeus Observability Platform.

3.3 System Workflow

Three workflows have to be discussed to illustrate this design's system workflow. They are responsible for maintaining Prometheus Agent, configuring probe settings, and scheduling probes.

Firstly, for the active monitoring platform's initialization, the admin must deploy Prometheus Agent via the official CRD: PrometheusAgent. As shown in Figure 3.2, the admin creates or updates the CR and receives the successful response. Then, the Prometheus Operator that keeps watching the CR retrieves the configuration and generates a new template to create or update the deployment of the Prometheus Agent.

Secondly, the workflow to create or modify probe configurations is similar to the above one, as they both leverage the Prometheus Operator for management and execution. In Figure 3.3, the user utilizes the CR: Probe, specifying the target Uniform Resource Locator (URL), scrape interval, and timeout interval, etc. As mentioned above, the Prometheus Operator would take over it, reloading the configuration of Prometheus Agent and bringing about the desired scheduling of probes.

Lastly, the third workflow is fully automatic, carrying out probes over different kinds of targets and operated by the Prometheus Agent as in Figure 3.4. With proper

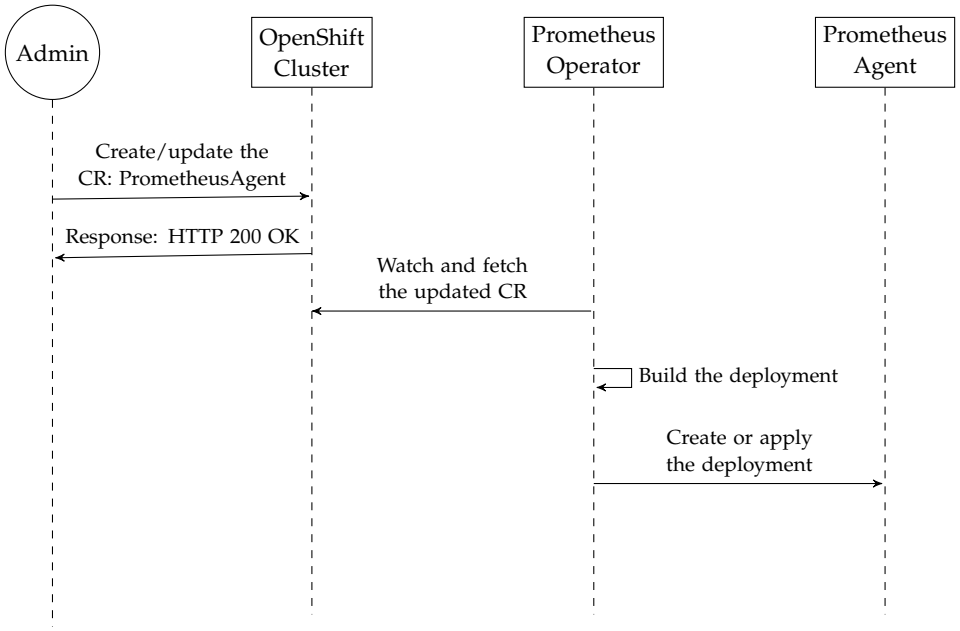

Figure 3.2: Deploying or updating Prometheus Agent by admin.

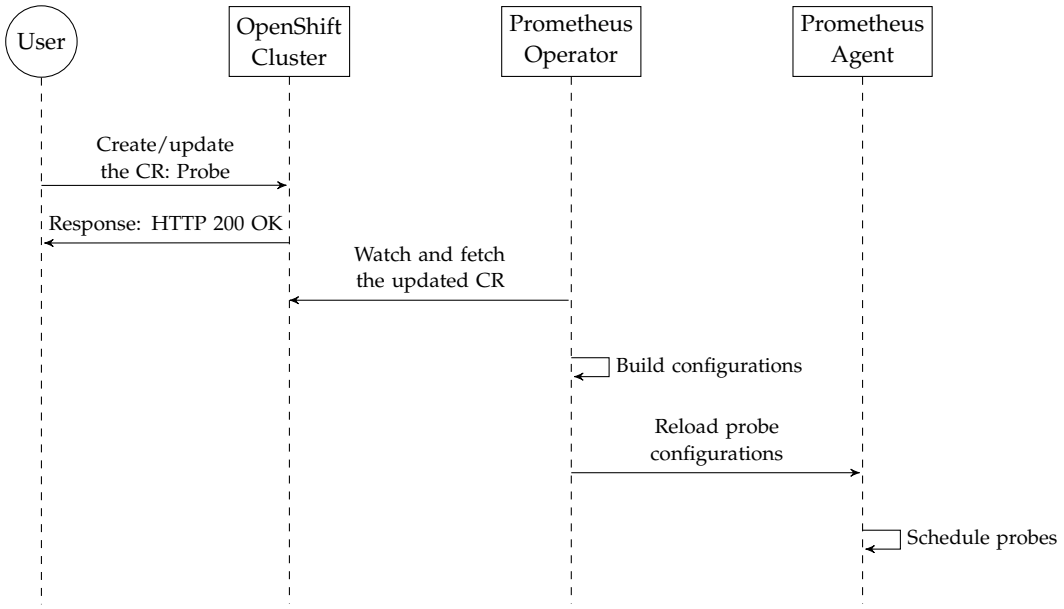

Figure 3.3: Creating or updating probe configurations by user.

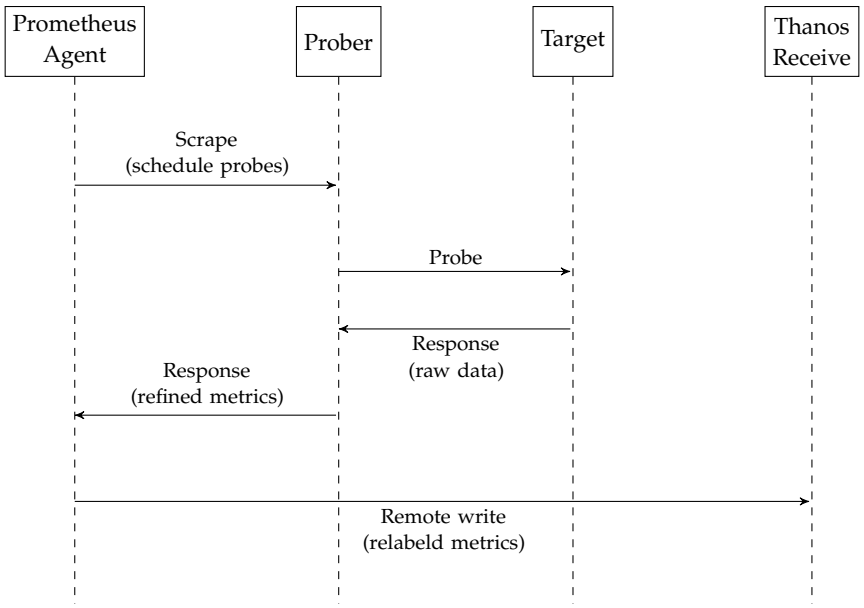

Figure 3.4: Scheduling probes by Prometheus Agent.

configurations of Probes, which the Prometheus Operator should load with the CR: Probe as mentioned in the previous paragraph, The Prometheus Agent would keep scraping the Probe with specified labels, parameters, modules, etc., and the Prober would probe targets accordingly, and then respond to the Prometheus Agent with collected raw information. Finally, after some operation on the metrics like relabeling or filtering, the Prometheus Agent sends these data to the Thanos Receive via remote write.

3.4 Integration with GitOps

So far, the design successfully addresses all requirements regarding active monitoring. However, Amadeus has plenty of clusters and namespaces, and numerous developers or users would deploy their own CR in their clusters and namespaces. So, managing deployed CRs becomes another crucial issue.

GitOps is a new approach to automate the management of the cloud infrastructure. Specifically, GitOps employs the Git repository, based on its modifications, triggering automatic reconfiguration and synchronization of the infrastructure as shown in Figure 3.5. Therefore, GitOps could also achieve Infrastructure as Code (IaC) by managing configurations in a Git repository.

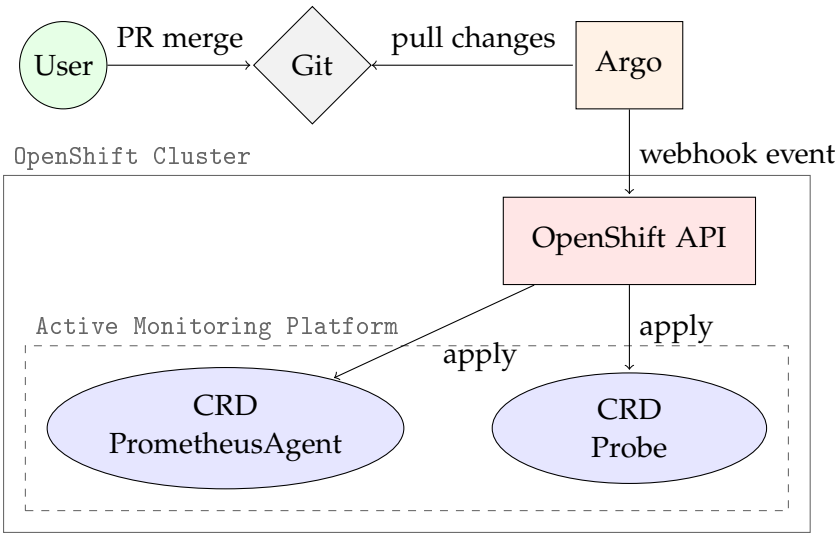

Figure 3.5: Utilizing GitOps to manage CRDs.

3.5 User Interface with Grafana

The implementations described above enable users to achieve scalable, reliable, and cloud-ready active monitoring. However, non-visualized results can be challenging for intuitive human understanding. Therefore, a vital final step involves using Grafana, an open-source analytics and visualization project, to create a dedicated dashboard for active monitoring. Since the Amadeus Observability Platform has already deployed the complete Grafana web application, no further deployment is necessary, apart from configuring the dashboard.

4 Design

4.1 Prometheus Operator

Prometheus Operator is an open-source tool designed to simplify the deployment, configuration, and management of Prometheus components in the Kubernetes ecosystem. Prometheus has grown in popularity due to its effectiveness in cloud monitoring, making it a cornerstone for developers focusing on reliability and performance. The Prometheus Operator, developed under the support of the CNCF, offers cloud deployment and management, aligning seamlessly with the principles and practices of cloud-native computing.

The primary goal of the Prometheus Operator is to make running Prometheus on the cloud platform as straightforward and efficient as possible. It leverages the container orchestrator's APIs to offer a scalable and highly available solution that fits the dynamic nature of cloud computing. The operator automates the complex processes of deploying, configuring, and managing Prometheus instances, making it easier for teams that don't have deep expertise in monitoring systems. By introducing CRDs representing distinct Prometheus components, such as Alertmanager, PrometheusAgent, and Prometheus itself, the integration lets users define their monitoring requirements with YAML configuration files declaratively.

Based on the orchestrator's API, one of the critical features of the Prometheus Operator is to dynamically discover and manage configurations, such as monitored targets. This dynamic management is crucial in environments where services and workloads constantly change. The operator automates updating Prometheus configurations in response to changes in the cluster, such as adding or removing pods, services, and endpoints. This ensures that monitoring is consistently aligned with the current state of the cluster, providing instant and accurate insights into applications and infrastructure.

In addition, to enhance the user experience, the Prometheus Operator also supports features like high availability, sharding, etc. To elaborate, for Prometheus and Alertmanager, it is ensured that monitoring and alerting systems remain operational even if individual components fail. For Prometheus Agent, the configuration file is divided into several configurations for multiple agent instances using the hash function, achieving Prometheus Agent sharding for distributing loads. Moreover, the operator also facilitates easy backup and restoration of Prometheus data, integrating with Kubernetes'

RBAC model to provide fine-grained access control over monitoring resources.

In conclusion, the Prometheus Operator enhances cloud monitoring by automating the deployment, configuration, and management of Prometheus. Its dynamic configuration adaptation, high availability, and sharding features increase efficiency and reliability. Therefore, utilizing the Prometheus Operator for robust cloud monitoring solutions in the Amadeus Observability Platform is valuable.

4.2 Prometheus Agent

The Prometheus Agent represents a simplified, focused approach to monitoring distributed systems, particularly in large-scale and cloud-native environments. As a lightweight variant of the Prometheus instance, the Prometheus Agent is designed to be a highly efficient data collector optimized for forwarding metrics to the Prometheus server or a compatible remote receiver.

One of the core advantages of the Prometheus Agent lies in its simplicity. Unlike an entire Prometheus server, which includes data storage, querying, and alerting functionalities, the agent is only responsible for scraping metrics and sending them out. This reduction leads to a smaller memory and CPU footprint, critical in resource-constrained environments, making it ideal for edge computing, microservices architectures, and multi-cluster environments.

In addition, the Prometheus Agent maintains excellent compatibility with the Prometheus ecosystem. This is a crucial consideration for organizations invested in Prometheus-based monitoring. It can scrape metrics in the same Prometheus exposition format, ensuring users integrate the agent seamlessly into the existing monitoring infrastructure. Furthermore, the agent's ability to integrate with service discovery mechanisms in orchestration platforms ensures its dynamically adapting to changes in the monitored environment.

Finally, the Prometheus Agent enhances the scalability and reliability of monitoring systems. The Prometheus Agent can achieve more efficient horizontal scaling depending on scrape configurations as shown in Figure 4.1 and focusing on scraping and forwarding metrics. This is particularly beneficial in large-scale environments, where a central Prometheus server can aggregate and process data from multiple agents, reducing duplication of storage and computation. Also, this architecture enhances the resilience of the monitoring system, as the failure of a single agent has a limited impact, ensuring reliable monitoring of the infrastructure.

Overall, the Prometheus Agent is an inevitable addition to the design of the active monitoring platform, offering a scalable and reliable solution for the existing monitoring system.

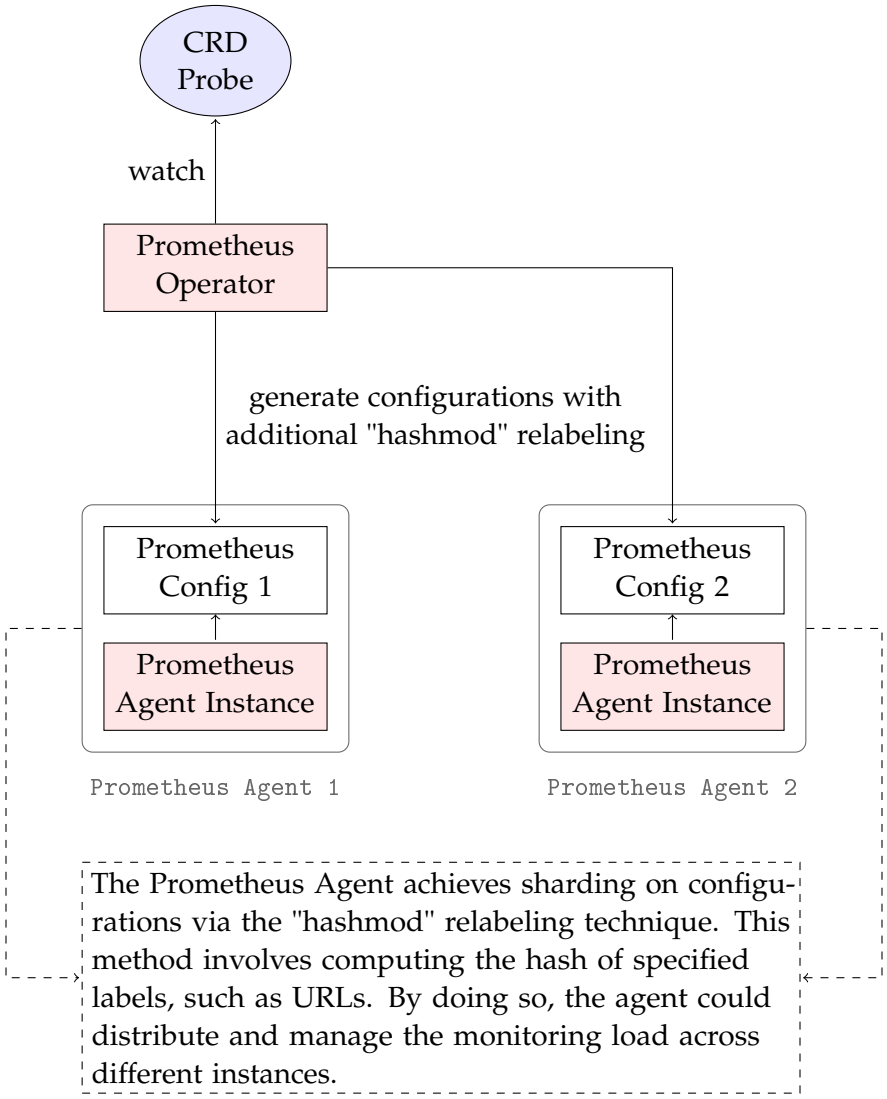

Figure 4.1: Prometheus Agent Sharding Mechanism.

4.3 Blackbox Exporter and Load Balancing

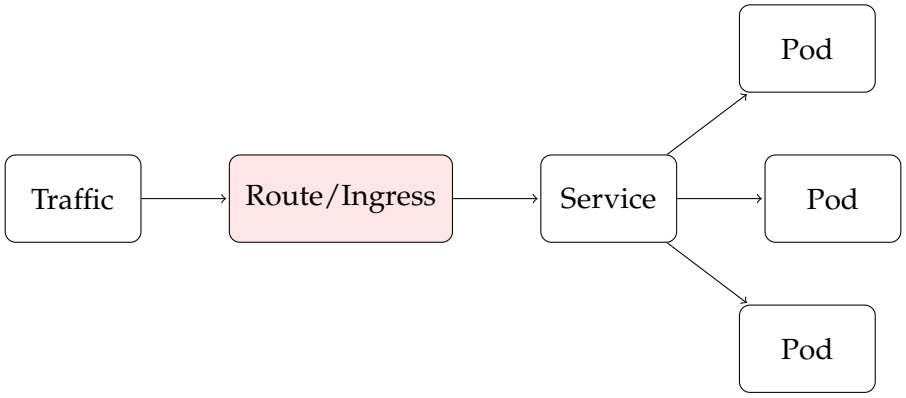

Figure 4.2: Route is the resource responsible for Load Balancing in Openshift.

Blackbox Exporter, a probing tool for the Prometheus monitoring system, is essential for assessing the performance and availability of services in network environments. It plays an important role in monitoring and diagnosing systems designed for probing endpoints over various protocols like HTTP, HTTPS, DNS, TCP, and ICMP.

In complex infrastructures with numerous services, integrating Blackbox Exporter with load-balancing solutions like OpenShift Routes or Kubernetes Ingress, as shown in Figure 4.2, is crucial. The integration facilitates scalable and efficient monitoring, ensuring the monitoring system remains reliable as the number of services grows.

Moreover, combined with automatic horizontal scaling, resource utilization efficiency could be significantly enhanced. Load balancers optimize traffic flow to the Blackbox Exporter instances; the orchestrator scales the number of the Blackbox Exporter instances, ensuring that the monitoring process does not overuse or run out of resources.

Another benefit is the high availability and fault tolerance. If an exporter instance fails, the load balancer can redirect traffic to other operational instances, ensuring uninterrupted monitoring. On the other hand, in cloud-native environments where services often change, this setup supports dynamic service discovery natively, reducing the need for manual configuration and simplifying overall management.

In conclusion, integrating Blackbox Exporter with load-balancing solutions offers enhanced scalable and reliable performance monitoring with simplified management. Load balancers ensure even distribution of monitoring loads, leading to more reliable performance metrics and easier identification of actual issues. This integration is indispensable for efficiently maintaining the health and performance of network services, making it a strategy for enhancing large-scale deployment and management.

5 Implementation

5.1 Deployment of the Prometheus Operator

This section aims to detail the preparations and procedures for the setup of the Prometheus Operator.

5.1.1 Modifying CRDs of the Prometheus Operator

5.1.2 Rebuilding the Prometheus Operator

5.1.3 Deploying the Prometheus Operator

5.2 Deployment of the Prometheus Agent

5.2.1 Configuration in CRD

5.2.2 Deploying the CRD: PrometheusAgent

5.3 Deployment of the Blackbox Exporter

5.3.1 Load Balancing from the Openshift Route

5.3.2 Deploying via Helm Chart

5.4 Creating Active Monitoring Configurations

5.4.1 Configuration in CRD

5.4.2 Deploying the CRD: Probe

5.5 GitOps for managing CRDs

5.5.1 Creating Git Repository for CRDs

5.5.2 Registering the Repository in Argo CD

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 inher-

ent 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

API Application Programming Interface

CNCF Cloud Native Computing Foundation

CPU Central Processing Unit

CR Custom Resource

CRD Custom Resource Definition

EDIFACT Electronic Data Interchange for Administration, Commerce and Transport

HTTP Hypertext Transfer Protocol

IaC Infrastructure as Code

IT Information Technology

QoS Quality of Service

SLA Service Level Agreement

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol

REST Representational State Transfer

TCIL Transport Cluster-Interconnect-Logic

TCP Transmission Control Protocol

URL Uniform Resource Locator

List of Figures

List of Tables

Bibliography

[Bey+16] B. Beyer, C. Jones, N. R. Murphy, and J. Petoff. *Site Reliability Engineering*. O'Reilly Media, Inc., Apr. 2016. ISBN: 978-1-4919-2911-7.

[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.

[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.

[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.

[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.

[Spl23] Splunk. *Active vs. Passive Monitoring: What's The Difference?* Splunk-Blogs. Nov. 20, 2023. URL: https://www.splunk.com/en_us/blog/learn/active-vs-passive-monitoring.html (visited on 01/08/2024).