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

I am profoundly grateful to my advisor, Samuel Torre Vinuesa, whose steadfast guidance and insightful criticism have been instrumental in my journey through this research. His deep knowledge and clear articulation of the subject matter have significantly enhanced my understanding and appreciation of the field. Samuel's support in defining the research topic, structuring the thesis, and navigating the challenges encountered during my research journey has been invaluable.

Additionally, I extend my heartfelt thanks to my supervisor, Prof. Dr.-Ing. Michael Gerndt, for offering me the chance to conduct my research within the chair of Computer Architecture and Parallel Systems at the Department of Informatics, Technical University of Munich. This opportunity has been pivotal in my academic and professional growth.

My colleagues at Amadeus Data Processing GmbH deserve special mention for their unwavering support throughout my research. Jerome Mazeries and Gais Ameer's assistance with technical issues and their constructive feedback have been crucial in the completion of my work. I am deeply appreciative of the supportive and collaborative environment they provided.

Last but certainly not least, I owe a tremendous debt of gratitude to my family. Their moral support and belief in my abilities have been my stronghold. It is their encouragement that fortified my resolve and enabled me to bring this thesis to fruition.

Abstract

With the increasing complexity and scale of the IT infrastructure, Amadeus's traditional monitoring solutions often fail to provide the necessary observability and operational efficiency. To address the gap, a scalable, reliable, and cloud-ready active monitoring platform becomes a need. The research investigates and compares two potential active monitoring solutions: a self-developed system employing operators, schedulers, and probers and a comprehensive solution using the Prometheus Operator, Prometheus Agent, and Blackbox Exporter for advanced cloud integration. The study concludes that with its automated configuration and management capabilities, the Prometheus Operator aligns most with Amadeus's scalability, reliability, and cloud readiness requirements. It ensures compatibility with custom protocols and optimizes resource usage, notably improving operational efficiency and reducing memory overhead compared to original monitoring methods. The architecture integrates seamlessly with Kubernetes and OpenShift, facilitating efficient deployment and management of monitoring components. A detailed evaluation of active monitoring solutions within a complex IT ecosystem was provided to contribute to the broad monitoring community, highlighting the Prometheus Operator's role in enhancing observability and operational efficacy. Future research will focus on automating Blackbox Exporter configurations, developing interfaces for custom probers, integrating Extended Berkeley Packet Filter (eBPF) for precise monitoring, and exploring advanced machine learning and tools tailored for microservices, aiming to enhance the efficiency, flexibility, and observability of monitoring solutions in complex IT environments.

Contents

Acknowledgments **iv**

Abstract **v**

1 Introduction **1**

2 Background **4**
2.1 Active Monitoring and Black-Box Monitoring 4
2.2 Prometheus Agent and Blackbox Exporter 5
2.3 Operator Pattern and Prometheus Operator 7
2.4 GitOps and Pull-based Continuous Deployment 9

3 Overview **11**
3.1 System Goals 11
3.2 System Overview 12
3.3 System Workflow 13
3.4 System Deployment 15

4 Design **17**
4.1 Prometheus Operator 17
4.2 Prometheus Agent 18
4.3 Blackbox Exporter and Load Balancing 20
4.4 Argo CD and GitOps 21

5 Implementation **23**
5.1 Deployment Tools 23
5.1.1 Helm Chart 23
5.1.2 Jenkins 24
5.1.3 Argo CD 24
5.2 Deployment of the Prometheus Operator 25
5.2.1 Context 25
5.2.2 Rebuild the Prometheus Operator and CRDs 26
5.2.3 Deploy the Rebuilt Prometheus Operator 26

5.3 Deployment of the Blackbox Exporter 27
5.3.1 Build the Blackbox Exporter Image with Necessary Modules . . 27
5.3.2 Deploy the Blackbox Exporter 29
5.3.3 Load Balancing via the Service 29
5.3.4 Scalability via the Horizontal Pod Autoscaler 30
5.4 Deployment of the Prometheus Agent 31
5.4.1 Configure the Prometheus Agent Configuration based on the PrometheusAgent CRD 32
5.4.2 Deploy the Prometheus Agent Configuration based on the PrometheusAgent CRD 32
5.5 Create the Monitoring Configuration 33
5.5.1 Configure the Monitoring Configuration Based on the Probe CRD 33
5.5.2 Deploy the Monitoring Configuration Based on the Probe CRD . 34
5.6 GitOps for managing CRDs 35
5.6.1 Create Git Repository for CRDs 35
5.6.2 Register the Repository in Argo CD 36
5.6.3 Deploy CRDs with Argo CD 36

6 Evaluation **37**
6.1 Methodology 37
6.1.1 CPU Utilization 37
6.1.2 Memory Utilization 38
6.1.3 Network I/O 38
6.2 System Analysis 38
6.2.1 Scheduler Analysis 39
6.2.2 Prober Analysis 40
6.3 Overhead Analysis 41
6.4 Scraping Analysis 42
6.5 Metrics Analysis 43

7 Related Work **45**

8 Summary and Conclusion **51**

9 Future Work **54**

Abbreviations **56**

List of Figures **58**

Bibliography **59**

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, events, and traces for Site Reliability Engineering (SRE) [7]. It is built on the Prometheus [38] and integrates with Splunk [45]. Additionally, Amadeus actively contributes to the Prometheus ecosystem. A prime example is Perses [35], a new Cloud Native Computing Foundation (CNCF) [12] 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), Hypertext Transfer Protocol Secure (HTTPS), Domain Name System (DNS), Internet Control Message Protocol (ICMP), Simple Network Management Protocol (SNMP), etc, 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 [33] or black-box monitoring [7] 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 [49]. On the other side, black-box monitoring focuses on ignoring the internal mechanism, only caring about the input and output of the application [7]. 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 [6]. 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 [38], Datadog [11], Nagios [27], 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 [29].

This project considers two 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 third solution involves leveraging the Prometheus Operator, the Prometheus Agent, and the Blackbox Exporter to construct an integrated, compatible active monitoring platform.

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 mirrors the first with substitutions: the operator for the Prometheus Operator [40], the scheduler for the Prometheus Agent [39], and general probers for the Blackbox Exporter [8], except for certain custom probers designed for specialized protocols by Amadeus. Similar to the first solution's operator, the Operator Pattern [32] facilitates the automation of configuration and management within container orchestration platforms like Kubernetes [37] and OpenShift [42]. Therefore, the Prometheus Operator helps users deploy and manage the components in the ecosys-

tem, which are needed for active monitoring, achieving sharding and auto scalability without additional effort [36]. 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 second 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 first solution requires extra effort to attain scalability and availability, the Prometheus Operator naturally includes automated configuration management, evenly scraping strategy, and sharding capabilities. Lastly, the Prometheus Operator holds an advantage over another solution 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 system design, data workflow, and system integrations. Section 4 discusses design criteria and proposed solutions, such as the employment of Prometheus Agent and the combination of load balancers, highlighting their value and design decisions. Section 5 details the implementation process, focusing on integrating the Prometheus Operator into the active monitoring Platform and employing the load balancer for 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 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 [49]. 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 [23].

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 [49]. For example, the Quality of Service (QoS) requirements of the network like latency and bandwidth [7], 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 [25]. 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 [7]. 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 [26].

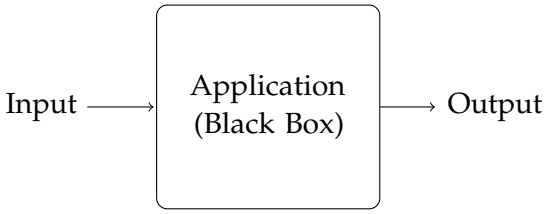

Figure 2.1: An illustration of the Black-Box Monitoring.

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 [7]. 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 [7]. 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. 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 [38] ecosystem allows users to harness it with its family of exporters. Thus, the Prometheus Agent [39] and the Blackbox Exporter [8] could operate for the objective of this project and even be compatible with other exporters. The

technology is extensively employed in various enterprises for monitoring applications in testing and production environments. The widespread adoption is a testament to the reliability and robustness of the Prometheus ecosystem 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 [36].

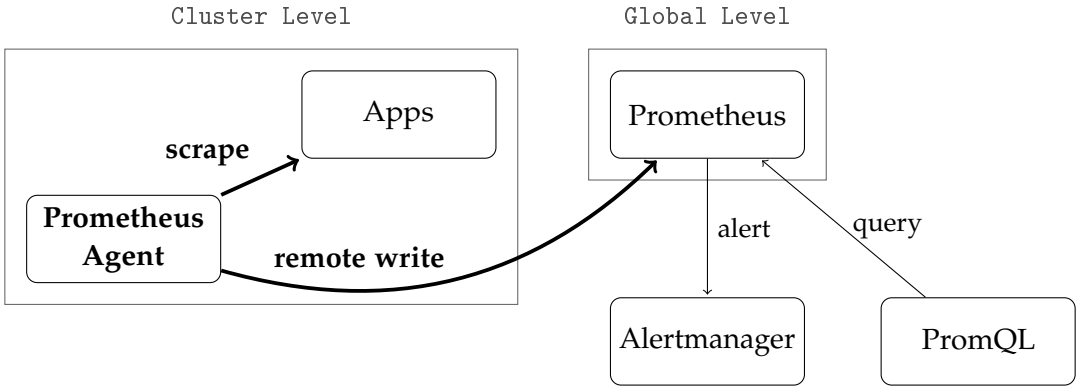

Figure 2.2: Prometheus Agent with Scrape and Remote Write.

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. 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 Platform as a Service (PaaS) clusters with latencies [47]. 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 [36].

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 [46]. 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 [36].

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 [32], defined by the CNCF, represents a paradigm shift in Kubernetes [37], enabling users to automate the maintenance and configuration of applications. The Prometheus Operator [40] 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.

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 [15]. 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 [13].

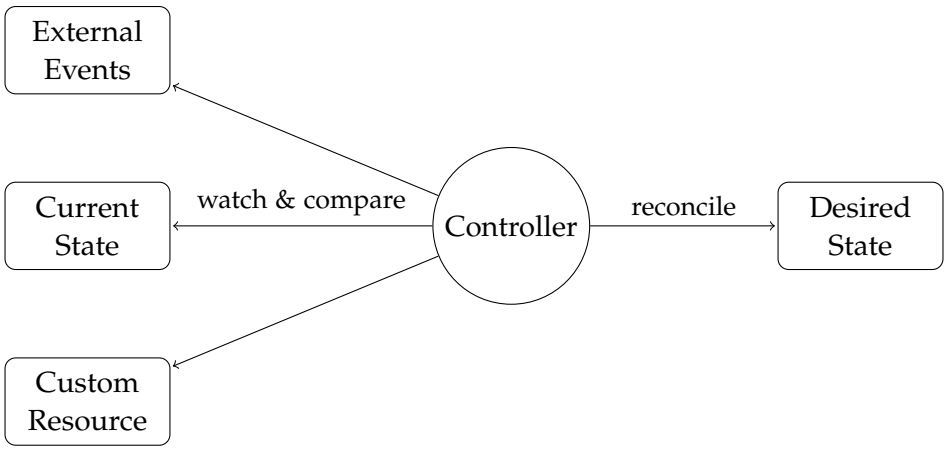

Figure 2.3: Illustration of the Operator Pattern.

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.

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 [8] and the module to be used. On the other hand, PrometheusAgent is responsible for defining a Prometheus Agent [39] deployment. This includes configurations like the number of replicas, ScrapeConfigs, and advanced features like sharding. Notably, it incorporates the ProbeSelector feature, which links to the previously defined Probe resource, enhancing its functionality and integration.

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. 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.

2.4 GitOps and Pull-based Continuous Deployment

GitOps [48], a concept that has rapidly gained traction in the realm of DevOps, introduces a novel approach to Continuous Delivery (CD) by leveraging Git repositories as the definitive source of truth for managing cloud-native applications and infrastructure. The origins of GitOps can be traced to the growing need for more robust and secure deployment practices in cloud-native ecosystems. This methodology builds upon Infrastructure as Code (IaC) principles and facilitates a declarative way to safely automate deployment and operational tasks [41].

Central to the GitOps methodology is its reliance on a pull-based strategy for CD, diverging from the traditional push-based deployment mechanisms [6]. Tools like Argo CD [3] exemplify the GitOps paradigm by monitoring Git repositories for changes and automatically applying those changes to the target environment. As shown in Figure 2.4, this pull-based model enhances security by minimizing direct access to the deployment environment and ensures that the deployed state matches the version-controlled configuration [41].

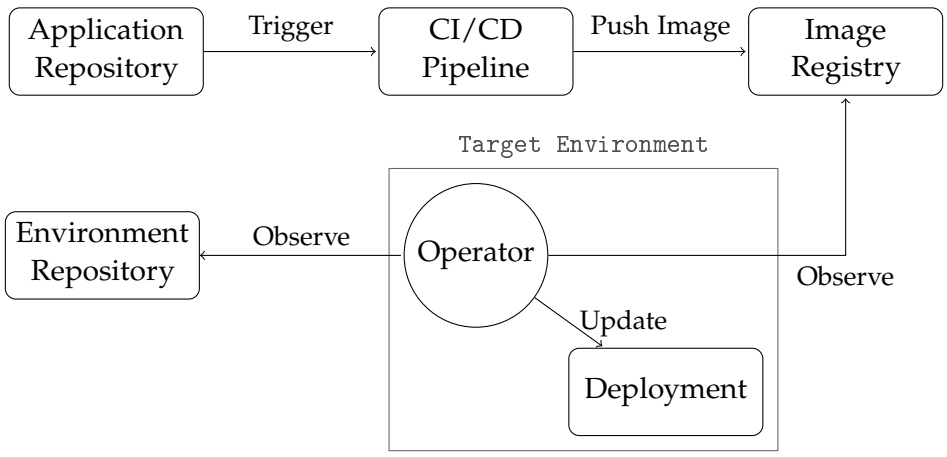

Figure 2.4: Pull-based GitOps Workflow.

Argo CD, among other GitOps tools, has become instrumental in operationalizing the GitOps model. It serves as an application delivery tool that enables the automated deployment of applications to various environments, following the configurations stored in Git. Argo CD's design allows for real-time synchronization between the desired application state defined in Git and the actual state in the cloud environment [43], providing a clear, auditable trail of all changes.

The evolution of GitOps underscores a commitment to enhancing collaboration between development and operations teams, streamlining deployment processes for greater efficiency and fewer errors. Organizations adopting GitOps unify code and operational changes under a single Git workflow, promoting transparency and shared accountability. This method not only quickens deployments but also minimizes the risk of misconfigurations. The pull-based strategy improves security by internalizing credentials and automatically aligning the infrastructure with the desired state [6], ensuring consistent deployments and heightened system reliability.

However, the shift towards GitOps and tools like Argo CD also necessitates refactoring a team's approach to configuration management and deployment strategies. As teams adopt GitOps, they must ensure that their Git repositories are well-maintained and that their deployment pipelines are securely configured to prevent unauthorized changes. Lastly, the learning curve associated with mastering these tools and practices represents a challenge organizations must navigate as they transition to a GitOps-centric model.

In conclusion, GitOps, with its pull-based approach to CD offers a more secure, transparent, and efficient method for managing cloud-native applications and infrastructure. Tools like Argo CD have been pivotal in bringing the GitOps vision to life, providing the mechanisms needed to bridge the gap between Git and operational environments. As the industry continues to embrace cloud-native technologies, the principles of GitOps will play a crucial role in future deployment and operational practices.

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 [42] 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 [48].

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 [39] 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 [40] 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 design proposal utilizes container orchestrators, including Kubernetes [37] and OpenShift. Within Amadeus, the OpenShift Container Platform is predominantly employed, often set up as separate and autonomous clusters. The system design described in the upcoming paragraphs is illustrated in Figure 3.1.

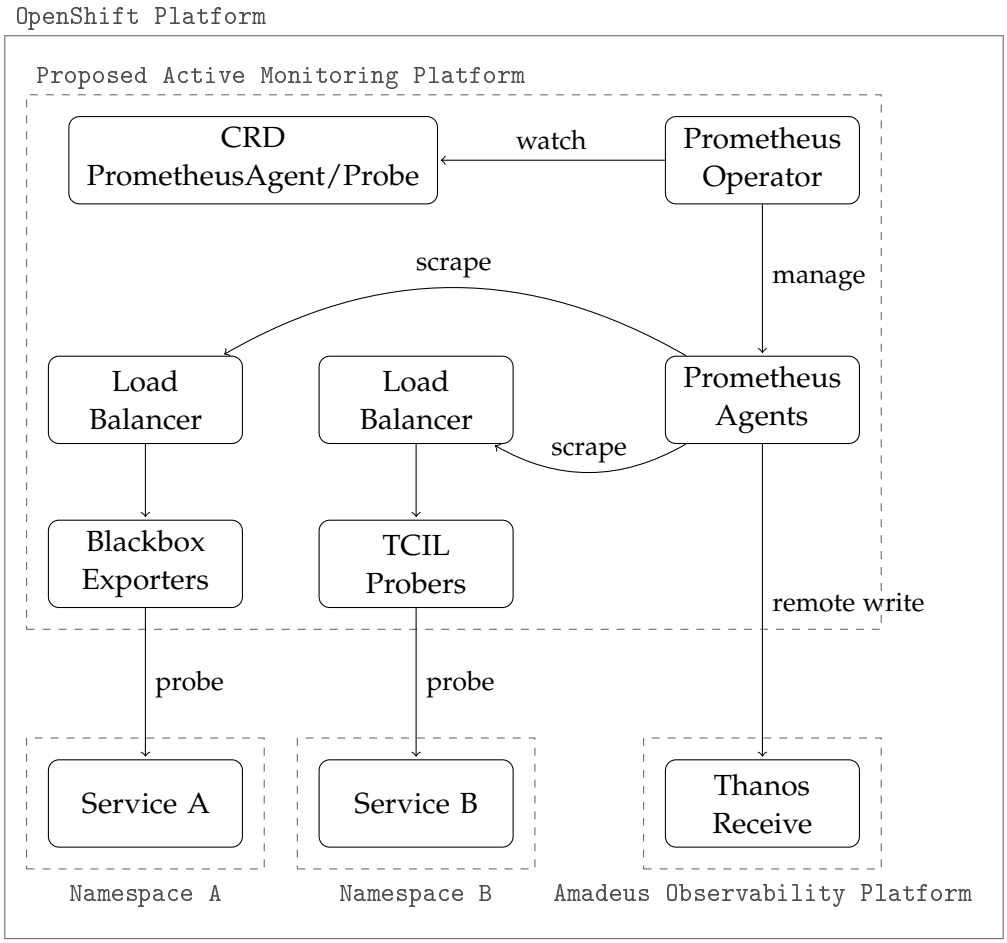

Figure 3.1: System Design with Prometheus Operator, Prometheus Agent and Probers.

Inside the cluster, a slightly modified Prometheus Operator would be installed, and the Operator would be responsible for deploying and managing Promenteus Agents and probe configurations. Next, Blackbox Exporters and customized probers like the TCIL Prober would be deployed with load balancers, leveraging automated 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 meets the needs of the Amadeus Observability Platform, offering an efficient, seamlessly integrated, and highly available solution for active monitoring.

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

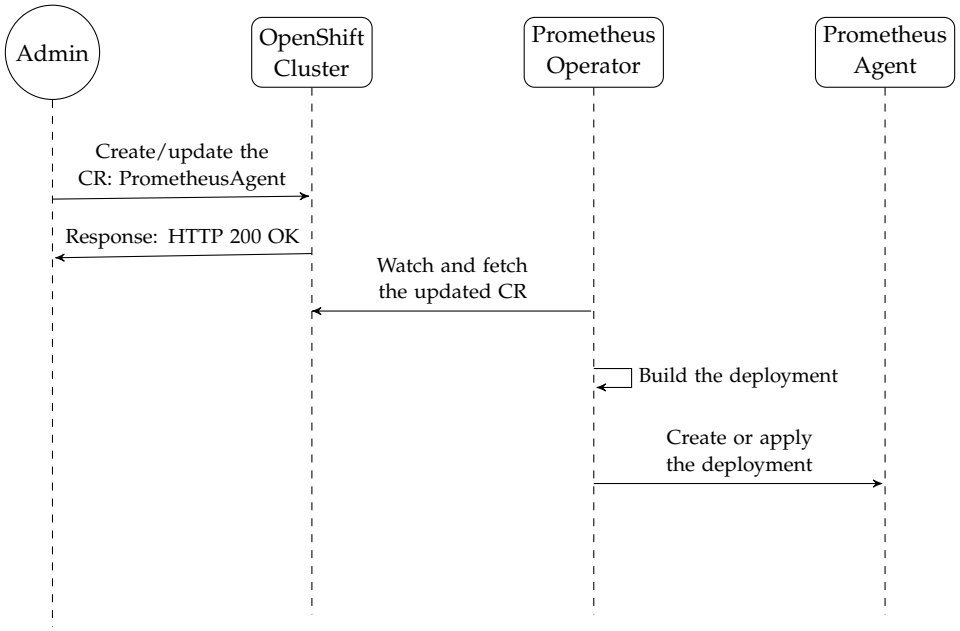

Figure 3.2: Deploying or updating Prometheus Agent by admin.

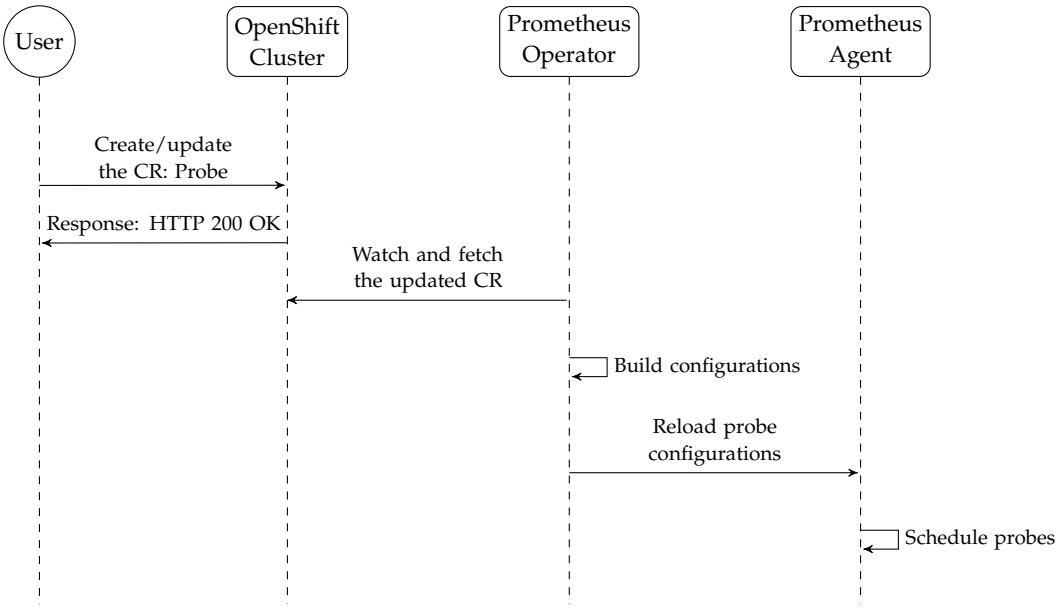

Figure 3.3: Creating or updating probe configurations by user.

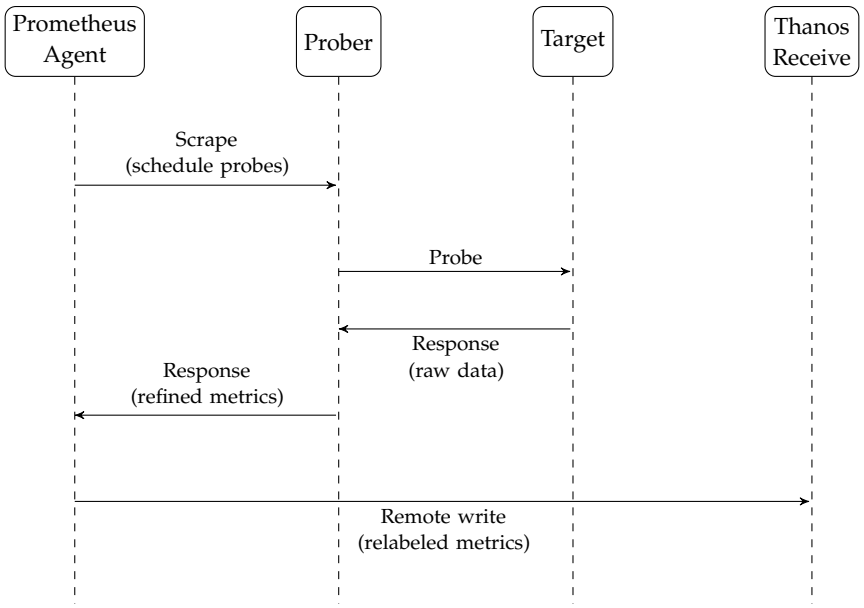

Figure 3.4: Scheduling probes by Prometheus Agent.

kinds of targets and operated by the Prometheus Agent as in Figure 3.4. With proper 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 System Deployment

The design effectively meets all requirements for active monitoring so far. However, Amadeus's large number of clusters and namespaces, along with the varied monitoring strategies employed by a wide range of developers and users on the Active Monitoring Platform, presents a significant challenge in managing the platform's deployment and CRs. To address the needs of developers and simplify project management, Amadeus utilizes a variety of Continuous Integration (CI)/CD tools, including Bitbucket [5], GitHub [10], Jenkins [22], Argo CD [3], as well as self-developed tools like the Application Centric Automation (ACA) and the Deployment Manager (DM),

which are leveraged for their flexibility and integration capabilities.

For CI, Jenkins is extensively used across numerous projects at Amadeus to automate the build process, significantly reducing bottlenecks [17]. Jenkins enables users to customize the integration pipeline, automating software development's build, test, and integration phases. It also offers comprehensive support for CD, including artifact archiving and deployment management across various platforms. However, the complexity of Jenkins pipeline scripts can present a steep learning curve for newcomers, and crafting scripts for sophisticated pipelines might be challenging.

Argo CD realizes the GitOps [48] principle, an innovative methodology aimed at automating cloud infrastructure management, leveraging the capabilities of Git repositories as the single source of truth for system states and configurations. This approach ties the Git version control system directly to the deployment and management processes, enabling automatic reconfiguration and synchronization of the infrastructure based on changes committed to the Git repository [6]. By doing so, GitOps extends the IaC principles by ensuring that all system configurations are maintained and version-controlled within a Git repository.

The project strategically utilizes Jenkins for CI and Argo CD for CD, harnessing their respective strengths. Jenkins facilitates the automation of the build, test, and integration phases, efficiently managing developmental bottlenecks, while Argo CD, embodying the GitOps principle, streamlines cloud infrastructure management via Git integration. This combination ensures an efficient and auditable deployment process, not only addressing the automation of deployment, also benefiting the deployment and management of monitoring configuration via CRs.

4 Design

4.1 Prometheus Operator

Prometheus Operator [40] is an open-source tool designed to simplify the deployment, configuration, and management of Prometheus components in the Kubernetes ecosystem. Prometheus [38] 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 [39] 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.

In conclusion, the Prometheus Agent improves the scalability and reliability of monitoring in large-scale environments, where a central Prometheus server can aggregate and process data from multiple agents. As illustrated in Figure 4.1, it achieves this by enabling more efficient horizontal scaling based on scrape configurations, focusing on scraping and forwarding metrics, thus reducing the duplication of storage and computation. Moreover, this highly available architecture boosts the monitoring system's resilience and reliability, as a single agent's failure could have an impact. Overall, the Prometheus Agent is an essential addition to the architecture of the active monitoring platform, offering a scalable and reliable solution for the existing monitoring system.

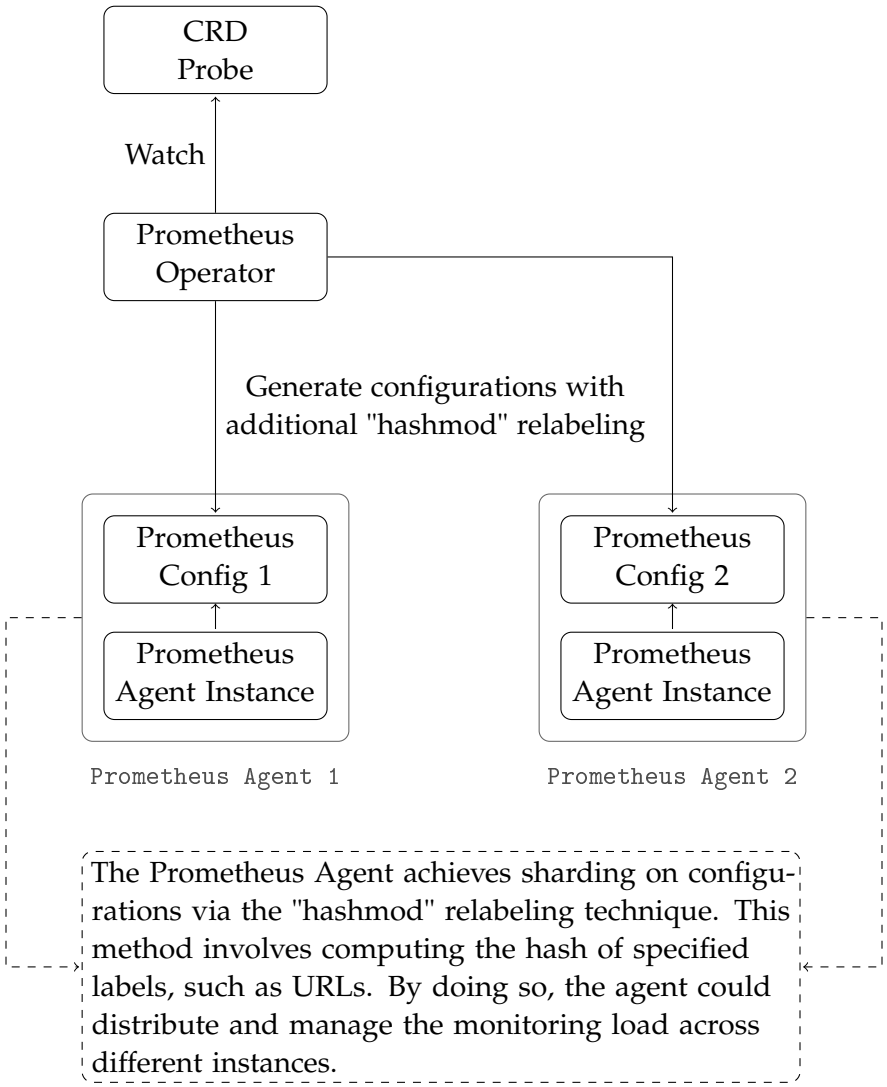

Figure 4.1: Prometheus Agent Sharding Mechanism.

4.3 Blackbox Exporter and Load Balancing

Blackbox Exporter [8], 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 hosting multiple services, it is crucial to integrate the Blackbox Exporter with Open Systems Interconnection (OSI) Layer 7 load-balancing solutions such as built-in Load Balancers, OpenShift Routes, Kubernetes Ingress, or OSI Layer 3 and 4 solutions like the Service, as depicted in Figure 4.2. This integration enables scalable monitoring, ensuring the reliability of the monitoring system as the service count increases.

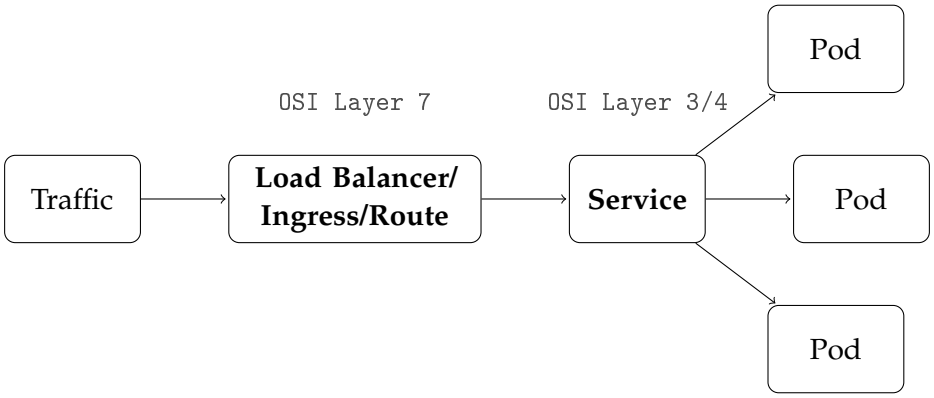

Figure 4.2: Route and Service are responsible for Load Balancing in Openshift.

Furthermore, automatic horizontal scaling significantly enhances resource utilization. Load balancers direct traffic to Blackbox Exporter instances efficiently, adjusted by an orchestrator to prevent resource overuse or depletion. This ensures high availability and fault tolerance, as traffic is rerouted to operational instances in case of failure, ensuring continuous monitoring. Additionally, in dynamic cloud-native environments, this setup facilitates native dynamic service discovery, minimizing manual configuration and simplifying management.

In conclusion, integrating the Blackbox Exporter with load-balancing solutions enhances scalability and reliability in performance monitoring. Load balancing evenly distributes monitoring loads, leading to more accurate performance metrics and easier issue identification. This integration is essential for effectively maintaining network services and is crucial in large-scale deployment and management.

4.4 Argo CD and GitOps

Argo CD [3], the declarative GitOps [48] continuous delivery tool for Kubernetes, automates the deployment of applications, ensuring they match the desired state configurations stored in Git repositories. GitOps emphasizes infrastructure automation and a pull-based deployment strategy via Git, improving deployment transparency, security, and efficiency.

Integrating Argo CD and GitOps for managing CRDs for PrometheusAgent and Probe on the OpenShift Platform can simplify the deployment and update processes as shown in the Figure 4.3. Users commit configuration changes to the Git repository, serving as the single truth source. By monitoring this repository, Argo CD automatically detects changes and communicates with the OpenShift API to update the CRDs accordingly, maintaining a continuous synchronization between the desired state in Git and the actual state in the cluster.

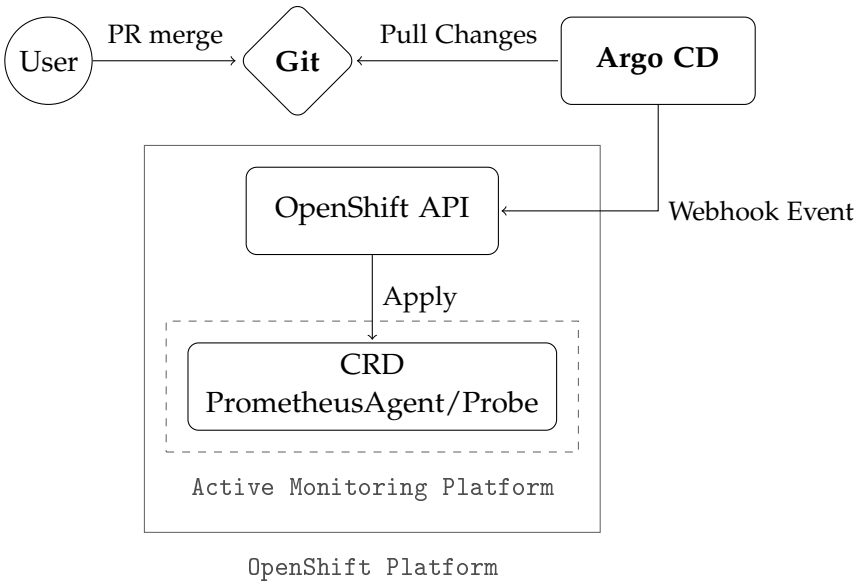

Figure 4.3: Utilizing GitOps to manage CRDs.

This methodology presents several benefits. Firstly, it enhances security by reducing direct access to the OpenShift environment, as changes are pulled from the repository rather than pushed from external sources. Secondly, it improves deployments' reliability and stability by ensuring they are always aligned with the version-controlled configurations. Additionally, it facilitates an auditable deployment process, enabling easy tracking and rollback of changes.

However, employing GitOps to manage configurations for Prometheus Agent and Probe has challenges. The learning curve for understanding and setting up GitOps workflows can be steep initially. Moreover, relying on a single source of truth requires strict repository management and version control practices to prevent configuration drift and ensure the repository accurately reflects the deployed environment's state.

Applying Argo CD and GitOps principles to manage CRDs for PrometheusAgent and Probe within the OpenShift Platform offers a powerful approach to automating deployments. By leveraging Git as the basis of the deployment strategy, users can achieve a more secure, reliable, and transparent infrastructure management process. Despite some challenges, the benefits of adopting a GitOps approach, particularly in complex cloud-native environments, are worthwhile for its value in facilitating efficient and secure deployment workflows.

5 Implementation

5.1 Deployment Tools

With the right tools, building a system on Kubernetes [37] or OpenShift [42] platforms becomes more convenient and efficient. The following sections will explore three dominating tools for deployment, outlining their advantages and disadvantages in various scenarios.

5.1.1 Helm Chart

Helm Chart [20] is a tool that simplifies applications' definition, installation, and upgrade processes in the orchestrator. It encapsulates complex deployments into manageable units by allowing users to package related resources—such as Deployments, Services, Routes, and ConfigMaps—into a single Helm Chart. This approach reduces the need to handle numerous, diverse YAML configurations and facilitates structured and versioned deployment projects. Users can craft a Helm library based on their applications' microservices, mitigating the redundancy and clutter of similar configuration files.

One of the critical advantages of Helm is its ability to simplify cloud application management. Helm provides a cohesive mechanism for deploying and managing applications by bundling related resources. It supports version control and enables easy rollbacks to previous application states, thus ensuring deployment stability. Moreover, Helm's templating engine offers dynamic resource file generation, promoting code reuse and configuration flexibility. Lastly, a community-driven charts repository further enhances its utility by providing access to a wealth of pre-defined charts for typical applications and services.

However, Helm still has some drawbacks. New users may experience a learning curve to effectively create and manage charts, mainly when dealing with Helm's templating syntax and functions. Managing chart dependencies and customizations can become frightening as deployments become complex, necessitating a thorough understanding of Helm alongside Kubernetes or OpenShift. Additionally, security is a concern; improper chart management or unverified third-party charts could pose risks to cluster security.

After all, Helm's advantages in application deployment and management often outperform its drawbacks. The tool's overhead is justifiable for complex or scalable applications, though direct resource files could be more straightforward for simpler applications. By offering an efficient pathway to manage deployments on the orchestrator, Helm aligns well with Amadeus' needs for cloud-native development, balancing flexibility with control.

5.1.2 Jenkins

Jenkins [22] is an open-source automation server that enables developers to build, test, and deploy their applications efficiently. It is widely used for CI and CD to automate various stages of the development process. Jenkins supports many plugins, allowing it to integrate with numerous development, testing, and deployment tools [4].

Jenkins' flexibility and extensibility are its most celebrated attributes. With an active plugin ecosystem and community, Jenkins provides a range of customization options, plugins, and integrations to meet almost any CI/CD requirement [44], making it one of the most renowned CI/CD tools among developers. Additionally, its platform independence guarantees compatibility with various operating systems and environments, facilitating its adoption.

Nonetheless, the sophistication of Jenkins breeds some disadvantages. The flexibility also introduces a level of complexity, particularly evident in the configuration and management of pipelines, plugins, and integrations. For larger projects, Jenkins can be resource-intensive, demanding significant hardware or cloud resources to maintain optimal performance. Additionally, the APIs and user interface of Jenkins often appear outdated and less intuitive, which can steepen the learning curve for new users.

Despite the above disadvantages, Jenkins could meet specific project needs, contributing to the automation of development processes [2]. Besides, the support backed by an extensive community and a wealth of plugins makes Jenkins a critical tool in the CI/CD realm. Though the complexity and resource demands may pose challenges, the benefits of simplified development cycles, enhanced productivity and deployment efficiency always outweigh these concerns.

5.1.3 Argo CD

Designed for the Kubernetes ecosystem, leveraging a declarative, GitOps-focused approach, Argo CD [3] empowers developers to automate and manage application lifecycles efficiently. The core concept of Argo CD is the GitOps [48] principle, indicating the desired state of the application deployment is maintained in a Git repository. This innovative approach ensures that any changes from the repository are automatically

reflected in the cluster, aligning the deployment state with the defined source of truth in Git.

The advantages of Argo CD are diverse, beginning with its adherence to the GitOps methodology. This alignment simplifies the management of infrastructure and application changes and improves the auditability of such modifications. By declaratively defining configurations in YAML files, Argo CD facilitates straightforward version control and rollback capabilities with its pull-based strategy, making maintaining and updating applications easier [6]. Also, its integrated user interface offers a comprehensive view of the application state across clusters, simplifying developers' monitoring and management tasks.

However, there are some downsides to Argo CD. Firstly, for those new to the GitOps paradigm or declarative management, there can be a steep learning curve to master Argo CD's functionalities. Furthermore, as it is specifically tailored for orchestrators, Argo CD's utility is somewhat limited outside of containerized environments, which might restrict its applicability in diverse infrastructural contexts. Last but not least, Argo CD focuses solely on continuous deployment, so developers will still need additional tools for continuous integration [41].

In conclusion, Argo CD is a robust continuous deployment solution for the containerization environment. Its design philosophy, centered around the GitOps approach, enhances deployment reliability, configuration accuracy, and operational transparency. While the initial learning phase, lack of continuous integration, and orchestrator-centric nature may pose some obstacles, This tool improves deployment accuracy, enhances oversight, and streamlines the entire application management process.

5.2 Deployment of the Prometheus Operator

This section aims to detail the preparations and procedures for the setup of the Prometheus Operator [40]. As mentioned in previous sections, the Prometheus Operator takes responsibility for deploying and managing Prometheus Agents [39] and probe configurations, enhancing users' experience in active monitoring. The following subsections outline and explain the modifications and deployment of the Prometheus Operator.

5.2.1 Context

Red Hat, Inc. has been actively promoting the integration of the Prometheus Operator into OpenShift. However, the most recent distribution hasn't incorporated the Prometheus Agent. Consequently, to avoid affecting the native deployment of the Prometheus Operator in OpenShift, a modified version of the operator, along with

the associated CRDs, is required at this stage. This adaptation ensures compatibility with current OpenShift configurations and employs the Prometheus Agent's advanced monitoring features.

5.2.2 Rebuild the Prometheus Operator and CRDs

To deploy a distinct Prometheus Operator for specific use cases and to prevent conflicts in environments where the native Prometheus Operator is already in use, it's essential to modify the group name of the targeted CRDs. This process consists of implementing necessary adjustments and constructing a custom version. The steps involved are as follows:

1. **Edit the Group Name in the CRD Manifests:** Modify the "apiVersion" field in each CRD manifest by changing the group name part. This unique group name helps to distinguish the custom Prometheus Operator from the default installation.
2. **Update the Operator Code:** Besides the CRD manifests, updating references to the group name in the Prometheus Operator's codebase is necessary. This ensures that the operator correctly interacts with the modified CRDs.
3. **Build the Prometheus Operator Image with the Modified Group Name:** Once the changes are complete, build a new Docker image of the Prometheus Operator.
4. **Push the New Image to a Private Container Registry:** After building the new image, push it to your private container registry that the cloud platform can access.

These steps ensure that the modified Prometheus Operator functions independently of the native Openshift Prometheus Operator, thus allowing for customized configurations and deployments.

5.2.3 Deploy the Rebuilt Prometheus Operator

Deploying a modified Prometheus Operator requires careful steps to ensure that it integrates seamlessly and functions correctly. This process not only involves applying the modified CRDs but also deploying the Operator itself. The steps are as follows:

1. **Check the Targeted OpenShift Cluster:** Examine the targeted cluster to ensure it is ready for deployment. This includes checking for sufficient resources, access rights, etc.

2. **Apply the Modified CRDs:** Deploy modified CRDs to the cluster.
3. **Deploy the Prometheus Operator:** Use an OpenShift template or a Helm chart to deploy the Prometheus Operator. Ensure that the location of the rebuilt image is specified in the private container registry in the deployment configuration.
4. **Verify the Deployment:** Check the status of the Prometheus Operator to confirm the successful deployment by console or Command-line Interface (CLI) tools. It is essential to ensure that the Operator runs without errors and can manage Prometheus instances as expected.

These steps help identify potential issues with the modified Prometheus Operator and ensure its compatibility and functionality within the OpenShift ecosystem. By following these detailed steps, the deployment of the modified Prometheus Operator can be conducted smoothly.

5.3 Deployment of the Blackbox Exporter

As an extension for the Prometheus [38] ecosystem, the Blackbox Exporter [8] has to be deployed additionally as the terminal to raise probes. This section will introduce the setup and steps for the Blackbox Exporter.

5.3.1 Build the Blackbox Exporter Image with Necessary Modules

In Amadeus, two fundamental checks - the existence check and the certificate check - require the creation of specific modules for the Blackbox Exporter:

- *Existence Check Module:* Confirms network connectivity through a HTTP GET request to an endpoint, ensuring a successful response. Configurable parameters include the URL, anticipated HTTP status codes, and timeout settings.
- *Certificate Check Module:* Validate the existence and Transport Layer Security (TLS) certificates of HTTPS endpoints to ensure the presence, validity, and signing. Configuration includes the URL, expiry thresholds, and Certificate authority (CA) verification details.

These modules are used to build a customized Blackbox Exporter image, which is subsequently deployed in the environment. The construction process involves the following steps:

1. **Create the Configuration of Two Modules:** Create a YAML configuration file for the Blackbox Exporter that includes these two modules. For example:

```
modules:
  existence_check:
    prober: http
    timeout: 30s
    http:
      preferred_ip_protocol: "ip4"
  certificate_check:
    prober: http
    timeout: 30s
    http:
      fail_if_not_ssl: true
      tls_config:
        ca_file: "/certs/ca.crt"
      preferred_ip_protocol: "ip4"
```

2. **Build the Blackbox Exporter Image:** Once the configuration file is ready, incorporate it into the Blackbox Exporter Docker image for a new build. This involves creating a Dockerfile that uses the Blackbox Exporter base image, adds the configuration file, and sets necessary environment variables. Here is the Dockerfile example:

```
FROM prom/blackbox-exporter:v0.24.0

COPY blackbox-config.yml /etc/blackbox_exporter/config.yml
ADD certs /certs

EXPOSE 9115
ENTRYPOINT [ "/bin/blackbox_exporter" ]
CMD [ "--config.file=/etc/blackbox_exporter/config.yml" ]
```

3. **Push the Image to the Private Container Registry:** After building the image, push it to a private container registry for deployment. Ensure that the cluster is authorized to pull images from this private registry.

Following these steps, a customized Blackbox Exporter with specific modules for existence and certificate checks is created, aiming at the needs of Amadeus's infrastruc-

ture monitoring. This ensures targeted and efficient monitoring, specifically designed for the network and certificate validation requirements.

5.3.2 Deploy the Blackbox Exporter

Deploying the customized Blackbox Exporter is straightforward, with the image already prepared with the necessary configurations. To ensure a successful deployment with the monitoring platform, follow these steps:

1. **Deploy the Blackbox Exporter:** Depending on the preference and the existing setup, the deployment can be done using either an OpenShift template or a Helm chart.

2. **Verify the Deployment:** After the deployment, ensure it functions correctly by checking its status via console or CLI. Necessary checks include the pod's running status, resource utilization, log outputs for errors, and connectivity to targets. This step is essential to verify the operational readiness for monitoring.

By following these steps, the deployment of the Blackbox Exporter can be ensured to provide reliable monitoring capabilities for the cluster.

5.3.3 Load Balancing via the Service

In the context of container orchestration, leveraging an OpenShift Service for load distribution at OSI Layer 4 is preferable and more efficient when there is no requirement for Layer 7 capabilities, such as path-based routing, thereby minimizing load-related overhead. The Service functions as a layer above the pods, creating a consolidated access point that distributes traffic evenly among different Blackbox Exporter pod instances. This method leverages kube-proxy's iptables mode for efficient traffic distribution. The following steps guide setting up load balancing with the Service:

1. **Create the Service Resource for the Blackbox Exporter:** Define a Service resource for the Blackbox Exporter. This Service will route traffic to the selected pods. Here is an example configuration:

```
apiVersion: v1
kind: Service
metadata:
  name: blackbox-exporter-service
spec:
  selector:
  app: blackbox-exporter
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9115
  type: ClusterIP
```

2. **Check the Connectivity of the Service Resource:** Once the Service is created, it gets assigned a ClusterIP and a DNS record within the cluster. Check the connectivity to this DNS record to ensure that the Service is accessible. You can do this by running a DNS lookup or a simple curl command from a pod within the same cluster:

```
$ nslookup blackbox-exporter-service
$ curl http://blackbox-exporter-service/
```

The created Service of the Blackbox Exporter serves as a central gateway for Prometheus Agent requests. The iptables proxy mode of the kube-proxy naturally directs traffic to available pods in a random and balanced way. This configuration ensures efficient and direct routing for monitoring traffic.

5.3.4 Scalability via the Horizontal Pod Autoscaler

The Horizontal Pod Autoscaler (HPA) boosts elasticity and cost efficiency by enabling dynamic adjustment of pod replicas in response to varying workloads, thus enhancing application scalability. It scales the number of pod replicas based on the usage of CPU, memory, or any other customized metrics [14]. This section explains how to implement an HPA for the Blackbox Exporter, allowing it to scale dynamically according to workload changes:

1. **Ensure the Metrics Server:** The HPA requires metric data to make scaling decisions. Ensure that the Metrics Server responsible for collecting resource usage data is deployed in the cluster. Here is an example:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: blackbox-exporter-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: blackbox-exporter
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 75
```

2. **Create the HPA Resource:** Define an HPA resource in YAML format. Specify the deployment to scale, the metrics to be used for decisions, and the desired target values for those metrics.

3. **Apply the HPA Configuration:** Apply the HPA configuration to your Kubernetes cluster.

4. **Adjust HPA Parameters:** Based on the observed performance, you might need to adjust the HPA parameters, such as the target CPU utilization or the maximum number of replicas. Update the HPA configuration file and reapply it as necessary.

Implementing an HPA for the Blackbox Exporter ensures that the deployment can adapt to changing demands automatically, maintaining optimal performance and resource utilization.

5.4 Deployment of the Prometheus Agent

To utilize the Prometheus Agent [39], in addition to deploying the instance, it's critical to set up the scraping metrics from targets depending on the configuration file. With the management by the Prometheus Operator [40], the efforts of deploying, loading, and reloading the instance could be released, greatly easing the maintenance complexity. In

this section, utilizing the Prometheus Operator to deploy and maintain the Prometheus Agent will be discussed in detail.

5.4.1 Configure the Prometheus Agent Configuration based on the PrometheusAgent CRD

The Prometheus Operator monitors the PrometheusAgent CRD, which is derived from the Prometheus configuration. The primary settings in this project are the "probeSelector" and "remoteWrite" fields, which are responsible for probe configurations and specifying the remote write endpoint, respectively.

The probeSelector functions as the Label Selector in this CRD, enabling users to define the labels of probes that will be selected by the Prometheus Agent. This feature allows the Prometheus Agent to distinguish the Probe used by the active monitoring platform from other monitoring systems, even though they might utilize the same official CRD.

The remoteWrite feature enables users to configure the remote endpoint to which the Prometheus Agent uploads metrics. By establishing the distributed Thanos Receive as the remote write endpoint in each local cluster, this architecture spreads some computational tasks to the edge, enhancing efficiency and availability.

5.4.2 Deploy the Prometheus Agent Configuration based on the PrometheusAgent CRD

Deploying a custom object from the PrometheusAgent CRD, which is monitored by the Prometheus Operator involves several steps to ensure successful integration and configuration. This process can be broken down into the following stages:

1. **Define a Custom Object from the PrometheusAgent CRD:** Create a custom object in YAML format with the necessary specifications of the PrometheusAgent CRD, including "probeSelector" and "remoteWrite" configurations. Here is an example:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusAgent
metadata:
  name: example-prometheus-agent
  namespace: monitoring
spec:
  probeSelector:
    matchLabels:
    apps: active-monitoring
  remoteWrite:
    - url: http://thanos-receive.monitoring:19291/api/v1/receive
```

2. **Apply a Custom Object:** Deploy the Prometheus Agent by applying the custom object in the cluster.

3. **Verify the Deployment:** Once the custom object is deployed, verify its status and ensure that the Prometheus Operator deploys the Prometheus Agent.

Following these steps, the Prometheus Agent is successfully deployed and configured in the cluster, leveraging the Prometheus Operator for efficient management.

5.5 Create the Monitoring Configuration

As of now, the preliminary settings have been completed, and users can utilize the Probe CRD to create an active monitoring configuration. This section will provide detailed instructions on configuring it, followed by the necessary steps.

5.5.1 Configure the Monitoring Configuration Based on the Probe CRD

With the Probe CRD, users can conveniently define monitoring for a set of targets, facilitating cloud-native configuration management within the cloud platform. The module, prober, and targets are the three most important fields for the configuration.

The "module" field in Prometheus configures how targets are probed, often using settings from the Blackbox exporter. The "prober" field specifies the prober details, with the essential "prober.url" parameter. "targets" defines a set of static or dynamic targets for the prober to monitor. Also, the "metadata.labels" needs to be set in coherent with the "probeSelector.matchLabels" in PrometheusAgent.

5.5.2 Deploy the Monitoring Configuration Based on the Probe CRD

Deploying the custom object from the Probe CRD is the last crucial step in active monitoring using Prometheus. It involves creating and applying the monitor configuration, ensuring that the specified targets are actively monitored based on the defined parameters. Here are the steps for the deployment:

1. **Define a Custom Object from the Probe CRD:** Create a custom object in YAML format to establish the desired monitoring configurations. Ensure that this object includes all essential fields, such as module, prober, and targets. Additionally, confirm that the metadata.labels within the object match the probeSelector.matchLabels specified in the previous section. For example, please refer to the provided code below.

```
apiVersion: monitoring.ruup.amadeus.net/v1
kind: Probe
metadata:
  labels:
    apps: active-monitoring
  name: probe-example
spec:
  interval: 15s
  jobName: probe-example
  module: existence_check
  prober:
    path: /probe
    scheme: http
    url: blackbox-exporter-service:9115
  targets:
    staticConfig:
      static:
        - 'http://www.example.amadeus.net/'
```

2. **Apply a Custom Object:** Once the custom object is ready, apply it to the targeted cluster.

3. **Verify the Deployment:** After applying the object, verify that it has been successfully deployed.

4. **Check the Monitoring Results:** Once the object is deployed, the Prometheus Agent will begin monitoring the specified targets according to the intervals and

parameters set in the configuration. The monitoring data and metrics can be viewed on the dashboards of the Prometheus Agent or the Blackbox Exporter.

After these steps, the custom object based on the Probe CRD is deployed, enabling active monitoring of specified targets using Prometheus Agent. This setup facilitates efficient and dynamic monitoring in cloud-native platforms.

5.6 GitOps for managing CRDs

GitOps [48] can automate and manage infrastructure configurations using Git as a single source. In this section, we realize GitOps principles to manage CRDs using, the Argo CD [3], a declarative continuous delivery tool for GitOps.

5.6.1 Create Git Repository for CRDs

1. **Initialize a Git Repository:** Create a new Git repository, which will host all the CRD manifests and related configurations. Users could use GitHub, GitLab, or any other Git hosting service for this purpose.

2. **Add CRD Manifests:** Upload CRD manifests into the repository with a well-set structure and directory. Ensure each CRD manifest is written in YAML format and correctly structured as the following example:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: amadeus-active-monitoring-crds
spec:
  destination:
    name:
    namespace: amadeus-active-monitoring
    server: 'https://kubernetes.default.svc'
  source:
    path: active-monitoring-crds
    repoURL: 'https://bitbucket.amadeus.net/aop/am.git'
    targetRevision: HEAD
  project: default
```

3. **Version Control:** Adopt the branching strategy to maintain different versions or environments. Then, commit and push changes to the remote repository.

5.6.2 Register the Repository in Argo CD

1. **Add the Repository with Argo CD UI:** In the Argo CD dashboard, navigate to the settings and add your Git repository. Provide the necessary credentials if your repository is private.

2. **Configure Repository Settings:** Configure the synchronization settings, such as automated sync policies, sync frequency, and other repository-specific settings as required by your deployment strategy.

5.6.3 Deploy CRDs with Argo CD

1. **Create an Application in Argo CD:** In Argo CD, "Application" represents a set of resources to be deployed. Create a new Application and link it to the directory in your Git repository where the CRD manifests are stored.

2. **Define Sync Policy:** Choose an appropriate synchronization policy for your CRDs. You can opt for manual sync, which requires manual intervention to apply changes, or automatic sync, where changes in the Git repository are automatically applied to the cluster.

3. **Version Management:** For any updates or changes in CRDs, update the manifests in the Git repository. Argo CD will synchronize these changes based on the configured sync policy, ensuring that your cluster state matches the desired state defined in Git.

By integrating GitOps practices for CRD management, organizations can achieve automated, reliable, and version-controlled deployment processes, aligning infrastructure and application states with complex configurations.

6 Evaluation

This evaluation study seeks to determine if the design leads to a tolerable overhead compared to the original in-house monitoring application that Amadeus developed. Both applications operate on a private OpenShift [42] Cluster managed by the Amadeus SRE team. The primary metrics collected include Openshift Metrics for CPU, memory, and network, gathered using cAdvisor [19] or the Network Metrics Daemon [31]. Subsequent sections will outline the evaluation methodology and present the findings from the system analysis.

6.1 Methodology

The evaluation focuses on analyzing system behaviors concerning CPU utilization, memory utilization, and network traffic. These three entities are critical for developers to gain insights into system overhead and performance intuitively. Understanding how a system allocates and utilizes its resources, along with how it communicates internally and externally, provides a comprehensive view of its efficiency and scalability.

6.1.1 CPU Utilization

For CPU utilization analysis within a pod, the metric "container_cpu_usage_seconds_total" will be utilized. This metric measures the cumulative CPU time consumed by a container in seconds. The methodology involves:

- Collecting CPU usage data over time to understand the baseline and peak CPU utilization patterns.
- Analyzing the CPU consumption in correlation with different applications to identify any potential inefficiencies.
- Comparing CPU utilization across different applications to assess resource allocation effectiveness.

This analysis will help in understanding the CPU demands of the applications running in OpenShift Pods and how effectively CPU resources are utilized.

6.1.2 Memory Utilization

Memory utilization will be assessed using the "container_memory_working_set_bytes" metric, which provides the amount of memory actively used by a container, excluding unused pages. The evaluation methodology includes:

- Monitoring the working set memory to identify memory usage under different operational conditions.
- Investigating memory patterns to ensure that applications operate appropriately and efficiently use memory resources.
- Evaluating memory usage trends over time to evaluate resource requirements.

This evaluation aims to highlight memory utilization efficiency and potential areas for optimization within the OpenShift environment.

6.1.3 Network I/O

Network I/O will be analyzed through "container_network_receive_bytes_total" and "container_network_transmit_bytes_total" metrics, representing the total bytes received and transmitted by the container network interface, respectively. The methodology involves:

- Monitoring inbound and outbound network traffic to identify communication patterns.
- Assessing network traffic volume in relation to application activity.
- Analyzing network traffic trends to assess data flow efficiency.

By examining network I/O, we aim to understand the network performance and efficiency of applications running in OpenShift Pods, highlighting areas for network optimization.

6.2 System Analysis

Prometheus' and Amadeus's solutions feature a similar architecture: the scheduler triggers the prober to carry out monitoring tasks while the prober probes targets. The forthcoming analysis will span 12 hours, focusing on the average hourly values of each metric. Given that the average target count in production environments is nearing 100, the following analysis will address both the scheduler and the prober within the scenario of probing 100 targets.

6.2.1 Scheduler Analysis

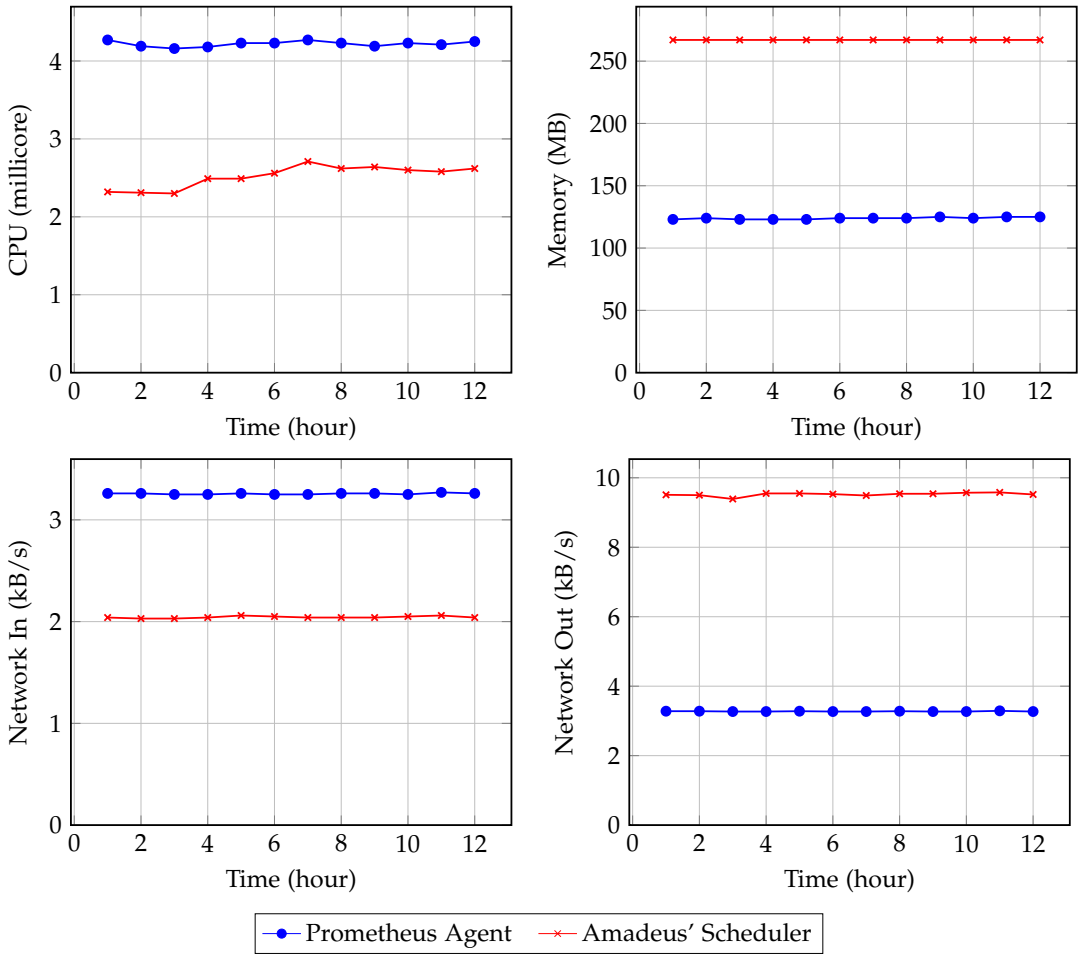

Figure 6.1: Scheduler analysis.

The Prometheus Agent and Amadeus' Scheduler perform similar roles but differ in their implementations, with the Prometheus Agent providing additional features. As depicted in the Figure 6.1, the CPU utilization between the two schedulers shows slight differences due to the minor units in the measurement. Surprisingly, memory utilization by the Prometheus Agent was significantly reduced by approximately 140 MB. Regarding network I/O, given that the measurements are in kB, the differences are relatively inconsequential. Adopting the Prometheus Agent results in an acceptable overhead while substantially decreasing memory consumption.

6.2.2 Prober Analysis

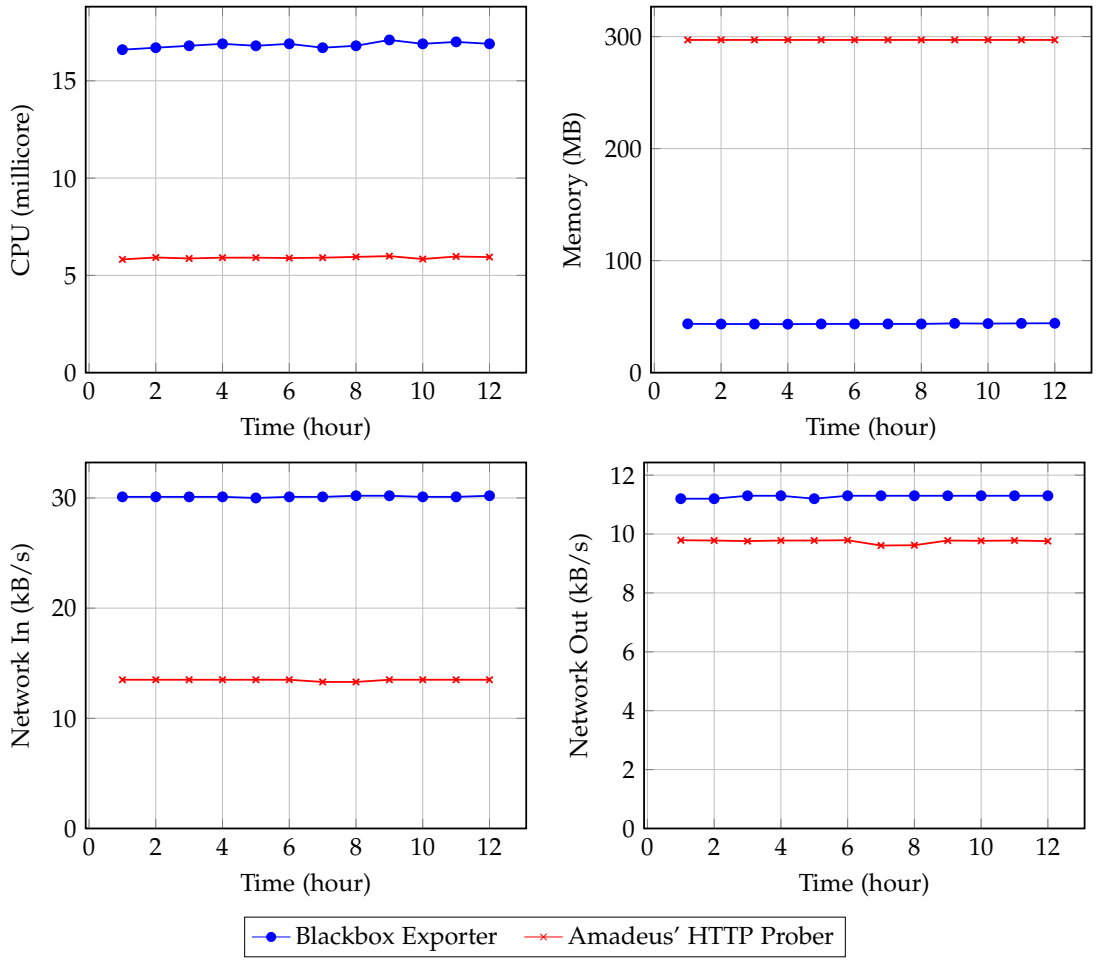

Figure 6.2: Prober analysis.

The Blackbox Exporter, capable of conducting customized probes across different protocols, contrasts with Amadeus' HTTP Prober's limitation to only HTTP probes. For a fair comparison, only the HTTP module is used in the Blackbox Exporter. Figure 6.2 shows the Blackbox Exporter's CPU usage is slightly higher than Amadeus' by about 10 millicores, a minor increase justified by its extended features and considered acceptable. Notably, it also offers a substantial memory usage improvement of approximately 250 MB. Although its network input is double that of its counterpart, investigations reveal this is due to the Prometheus Agent's more detailed HTTP GET requests, which are not expected to cause issues, even when scaling up to 10 or 100 times the number of targets

due to the lightweight nature of these requests. In summary, the minor increases in CPU and network input by the Blackbox Exporter are more than compensated for by considerable memory savings, making it a fully acceptable solution.

6.3 Overhead Analysis

This overhead analysis evaluates the escalating load rate as the target number increases from 100 to 1000. Utilizing these findings could enhance the use of HPA by identifying appropriate threshold metrics. All metrics were recorded as hourly averages for each target number, expected to increase proportionally with the target number in theory:

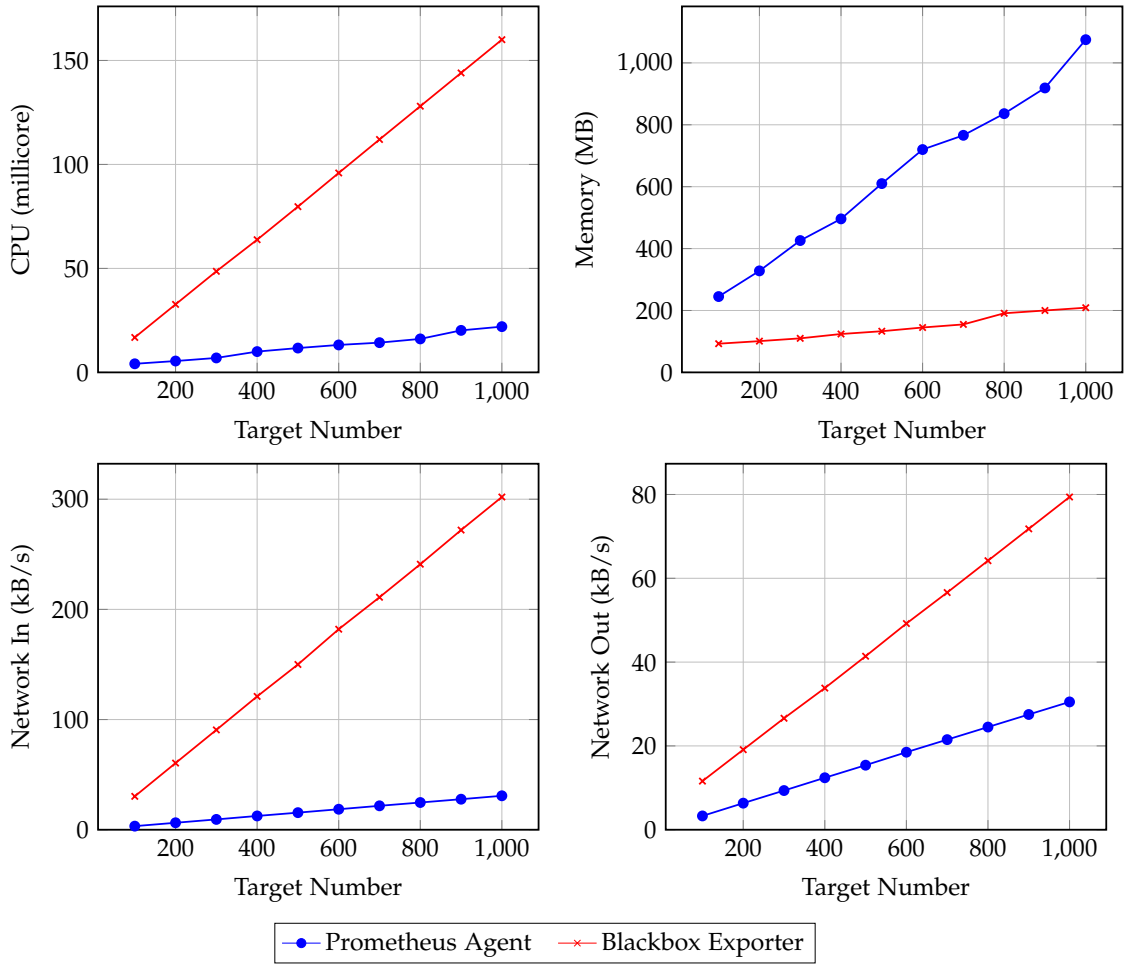

Figure 6.3: Overhead analysis.

As shown in the above Figure 6.3, all metrics demonstrate a consistent linear growth trend with each addition of 100 targets. Initially, the Prometheus Agent exhibits significant memory usage and a higher rate of increase, attributed to its role in scheduling probes, which involves managing numerous scraping jobs and their associated configurations. Therefore, utilizing the memory metrics from the Prometheus Agent for HPA is a reasonable choice. Conversely, the behavior of the Blackbox Exporter differs from the Prometheus Agent. It displays a modest upward trend in memory usage alongside more pronounced CPU and network utilization increases. Notably, the growth rates for CPU and network usage are comparable, indicating that the overhead is primarily relevant to network-related tasks. Thus, CPU metrics are suitable as threshold indicators for the Blackbox Exporter.

6.4 Scraping Analysis

This analysis assesses the efficiency of the Prometheus Agent's scraping approach by tracking the "Network In" metric of the Prober, which can be regarded as an indicator of the traffic generated by the Scheduler's probing requests. The traffic is quantified as the average per minute, with the following figure metrics over 30 minutes:

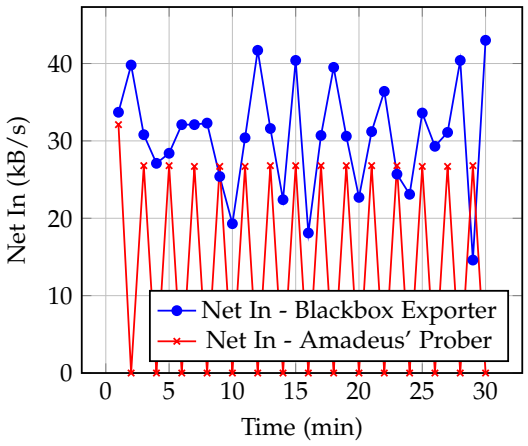

Figure 6.4: Scrape Analysis.

As demonstrated in the Figure 6.4, it is evident that there is no zero traffic in the Blackbox Exporter within 30 minutes, in contrast to Amadeus' Prober, which exhibits regular saw-tooth traffic patterns touching the bottom, indicating periodical drops in traffic to zero. This observation suggests that Prometheus Agent scatters traffic across the time interval, resulting in more optimized and spread network traffic.

6.5 Metrics Analysis

In addition to the overall system performance analysis, the metrics returned by probes from the Blackbox Exporter offer crucial insights for users investigating the root cause of incidents or other unusual occurrences. Typically, users rely on the "probe_success" metric to assess the health of targeted services. For more detailed analysis, they can examine metrics such as "probe_http_status_code," "probe_http_version," and "probe_http_content_length" to check the HTTP status, version, and response length, among other details.

Furthermore, the "probe_http_duration_seconds" metric is valuable for diagnosing issues or identifying unexpected failures. This compound metric breaks down the duration of an HTTP connection into five phases: resolve, connect, tls, processing, and transfer, each highlighting a different stage of the HTTP request. Consider the example figure below for an actual incident:

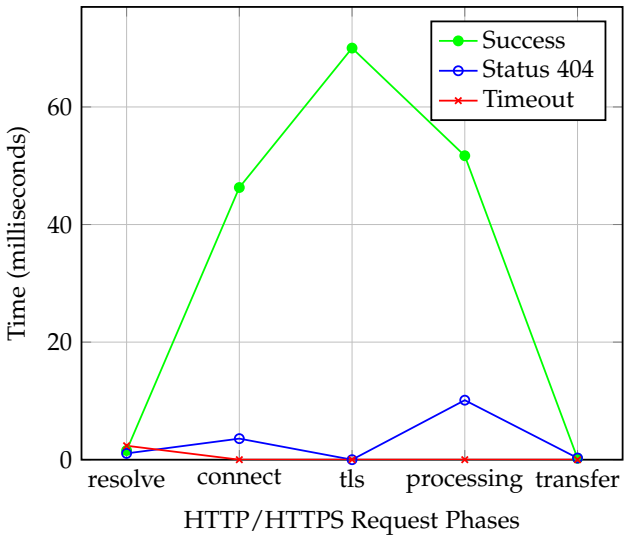

Figure 6.5: Metrics Analysis.

Figure 6.5 illustrates three probes, each capturing distinct metrics. In the successful case, all phases display non-zero values, indicating a seamless progression through each phase of the HTTPS request, although the transfer phase shows a small value. The blue line represents a successful HTTP request that, however, results in a status code 404, with the absence of a value in the TLS phase subtly confirming the request's purely HTTP nature without redirection to HTTPS. On the other hand, the Timeout probe reveals activity solely in the "resolve" phase, suggesting it stalled post-domain name resolution before connection establishment, hinting at a bottleneck impeding

further progress.

In summary, the Blackbox Exporter offers more than probe results; it provides users with various metrics for conditional judgment and troubleshooting. These metrics enable users to verify the TLS version, check the Internet Protocol (IP) protocol, and even validate the CA certificate. Additionally, the detailed measurement of HTTP request durations enhances observability over targeted services, yielding more profound insights into the monitoring process.

7 Related Work

In the rapidly evolving landscape of IT infrastructure, the appearance of cloud computing has fundamentally reshaped the demand for effective monitoring and analytics platforms. These platforms are essential for maintaining system reliability and performance in an increasingly complex environment. The shift towards cloud services involves scalable and dynamic monitoring solutions to address challenges cloud architectures pose, such as the diverse and transient nature of resources.

The development of cloud computing has heightened the complexity and necessity for efficient monitoring systems. In the framework developed by IBM in 2012, Cloud Monitor is highlighted as a pivotal solution for monitoring applications within the cloud environment. The framework depicted in Figure 7.1 consisting of the Monitoring Server, the Monitors, and the Agents was proposed, addressing the multifaceted challenges cloud applications present, such as multiple failure points and difficulty tracking failures due to the expansive nature of cloud infrastructures [1]. This architecture marks an essential evolution in cloud monitoring strategies, impacting future research and implementations.

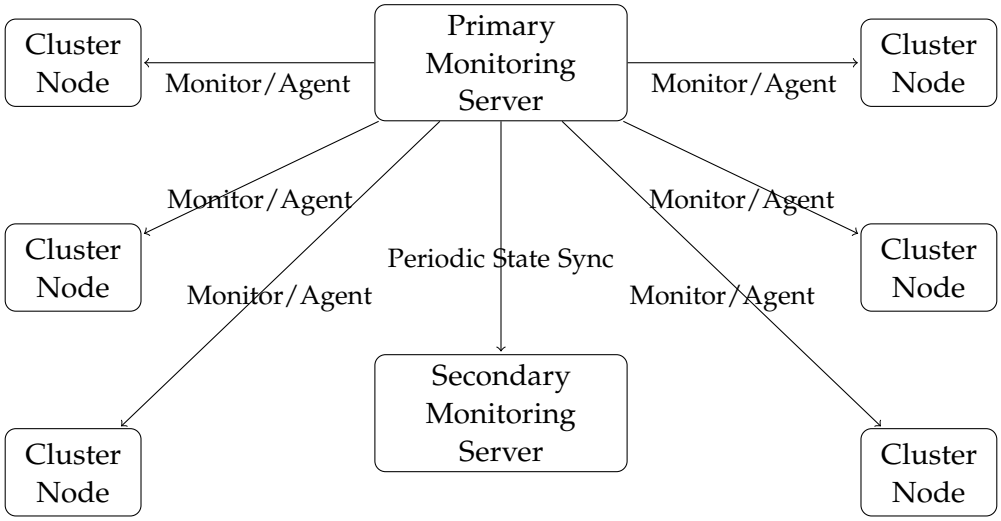

Figure 7.1: IBM's Cloud Monitoring.

However, further complexities arise when monitoring large-scale distributed systems, where scalability and integration become critical. Research about a scalable and integrated cloud monitoring framework that utilizes distributed storage to enhance efficiency and stability in cloud platforms was proposed [50]. This approach not only improves query efficiency but also ensures the reliability of cloud services, demonstrating the evolving strategies to address the dynamic and large-scale nature of cloud resources. Nowadays, utilizing scalable time series databases like InfluxDB [21] has become the prevailing solution for cloud monitoring.

Similarly, a paper details another distributed architecture for effectively monitoring private clouds by combining various tasks into a single service, including Collectors, Metasensors, Distributors, and a Visualizer. [34]. With the aid of Metasensors, Collectors gather data from machines and classify workloads, allowing for proactive resource management. This architecture offers a comprehensive solution leveraging machine learning to provide a detailed overview of the private cloud infrastructure. With trained neural network-based modules, the architecture's emphasis on avoiding data centralization and enhancing load balancing and fault tolerance reflects the nuanced requirements for monitoring private cloud environments effectively.

The development of message collectors significantly strengthens the foundation of cloud monitoring. cAdvisor [19], a container monitoring tool developed by Google, offers a comprehensive overview of resource usage and performance for running containers with native support. Designed specifically to monitor container metrics, it provides valuable insights into CPU, memory, filesystem, and network usage, thereby aiding in resource management and optimization.

Regarding network monitoring, the Network Metrics Daemon [31], an initiative by OpenShift, marks a significant advancement by collecting essential network performance metrics directly from pod and node-level operations. This tool plays a crucial role in enhancing visibility into network behavior, which is vital for identifying and addressing network-related issues within Kubernetes [37] environments.

Compared to internal logs or metrics exported from an application, apparently, the metrics collected by tools like cAdvisor and the Network Metrics Daemon are cost-effective. As a result, to address challenges like latency or performance limitations in client-side monitoring, an approach that analyzes user behaviors with CPU and network metrics to pinpoint issues at the network edge has been proposed [18]. This method leverages clients' response times to infer server and network performance, thereby identifying both internal CPU and external network bottlenecks through black-box monitoring. Proofing the effectiveness of black-box and active monitoring techniques, this research offers a complement to traditional white-box monitoring strategies, enhancing the ability to detect system bottlenecks.

To reduce overhead and achieve cost efficiency, the following researches present in-

novative approaches to tracking microservices and distributed systems' performance in the context of monitoring distributed systems without impacting system performance. Brondolin et al. [9] and Neves et al. [29] both advocate for utilizing eBPF [16] technology for black-box monitoring. By running the sandboxed programs as metrics collectors in the system kernel, eBPF technology provides detailed insights into microservices' architectural metrics, application performance, and resource usage with minimal overhead. These studies underscore the significance of fine-grained observability data in managing complex modern distributed systems.

Also, an active monitoring system based on black-box monitoring principles was developed in Amadeus, utilizing the operator pattern [32] for a straightforward cloud monitoring solution. As shown in Figure 7.2, this system is structured around three main components: the operator, the scheduler, and the prober, enabling a practical and lightweight approach to cloud-native monitoring. The monitoring setup begins with users defining their configurations through CRs, which the operator uses to set up and maintain the scheduler's monitoring tasks. The scheduler, acting as the orchestrator, initiates monitoring according to the CRs and works with probers responsible for probing services and collecting data. Despite its efficient design, the system faces scalability issues and a potential single point of failure in the scheduler, limiting its effectiveness in large-scale, distributed environments.

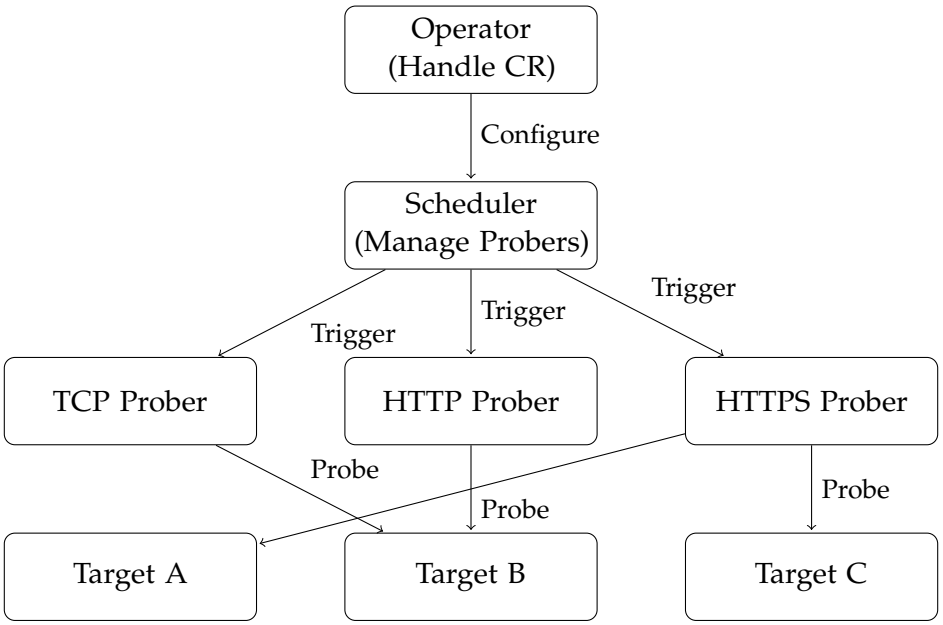

Figure 7.2: Amadeus' Active Monitoring.

Meanwhile, comprehensive monitoring platforms like Prometheus [38], Datadog [11], and Nagios [27] all offer robust solutions for overseeing and alerting on system metrics, reflecting the diverse ecosystem of tools and platforms catering to the nuanced needs of IT infrastructure monitoring.

Initially developed by SoundCloud, Prometheus is a widely adopted open-source toolkit for system monitoring and alerting. Its success stems from a potent data model, a flexible query language (PromQL), and the capability to manage high-dimensional data. Utilizing a pull model for metrics collection, Prometheus simplifies system monitoring by scraping data from endpoints without needing additional agents. This system supports extensive querying and various graphing tools, offering deep insights into system performance. However, its scalability hinges on the deployment architecture, posing challenges in managing extensive setups efficiently.

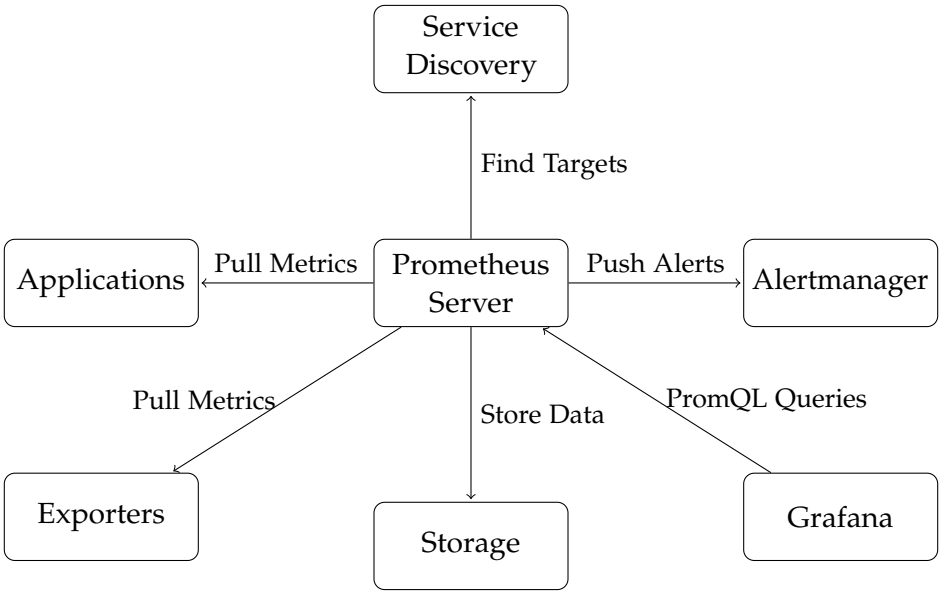

Figure 7.3: Prometheus's Monitoring Architecture.

Datadog is an integrated platform for monitoring, providing a comprehensive view of an organization's IT infrastructure by collecting and analyzing data from various sources. It features Synthetic Monitoring, which actively simulates requests and actions to gauge the performance of APIs across different network layers. This monitoring is divided into API Tests, which assess the availability and metrics of services, and Browser Tests, which simulate user interactions with services to offer experiential insights. Additionally, Datadog allows for the customization of private locations for monitoring by utilizing Docker containers, enhancing the flexibility of where and how

monitoring tasks are executed.

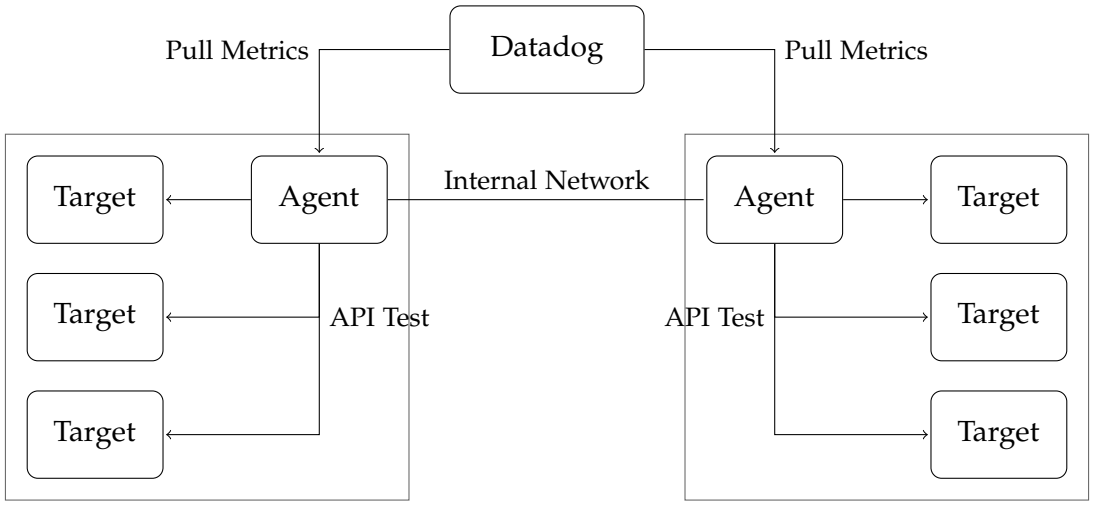

Figure 7.4: Datadog's Synthetic Monitoring.

On the other hand, Nagios is a widely renowned player in the monitoring landscape, known for its flexibility and comprehensive plugin ecosystem. This extensibility is supported by a strong community that continuously contributes to developing new plugins and features. For example, the famous Nagios Remote Plugin Executor (NRPE) [30] allows users to give remote access to plugins [28] designed to perform software and hardware checks as shown in the following Figure 7.5. Nagios operates on a proactive monitoring model, alerting administrators to potential issues before they become critical [24], thereby aiding in maintaining system uptime and performance. Despite its powerful features, Nagios's configuration and setup can be complex and may require a steep learning curve for new users. Additionally, its user interface and visualizations may not be as modern or intuitive as newer monitoring tools. Still, its reliability and effectiveness keep it operating in many enterprise environments.

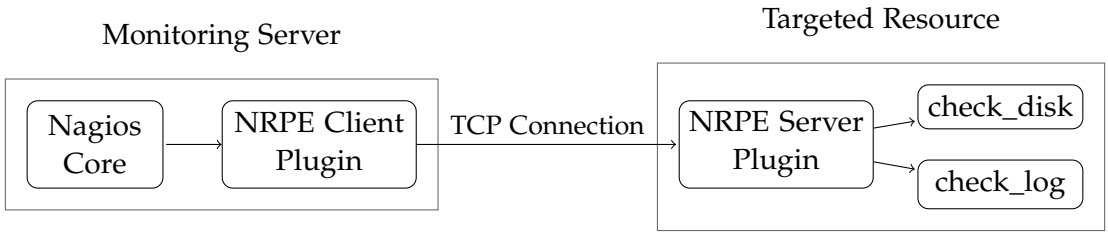

Figure 7.5: Nagios's Monitoring with NRPE plugin.

In summary, the related work underscores the critical evolution and diversity of monitoring strategies and solutions in addressing the complexities of modern IT infrastructure, especially in cloud and distributed system environments. The literature highlights the challenges faced and presents innovative approaches and technologies that contribute to the ongoing development of efficient, scalable, and integrated monitoring frameworks.

8 Summary and Conclusion

A sound system monitoring solution is indispensable in the IT industry. Amadeus explicitly emphasizes the unique requirements and solutions applicable to complex networks of interconnected applications, so adopting a proper monitoring approach becomes gradually crucial. The increasing demands have proven that observability goes beyond just being practical, which ensures modern IT industries operate smoothly and deliver top-quality services.

The creation and deployment of the Amadeus Observability Platform, which utilizes the Prometheus [38] ecosystem and incorporates Splunk [45], highlights the necessity for a durable, scalable, and adaptable monitoring system. Besides, researching active monitoring [33] methodologies reflects Amadeus's dedication to early detection, issue resolution, and service quality. Thus, two potential solutions were investigated to harness active monitoring in the Amadeus Observability Platform: the self-developed system employed operator, scheduler, and probers, and the integration of the Prometheus Operator, the Prometheus Agent, and the Blackbox Exporter.

Within this project, The selection of the second option, which involves the Prometheus Operator [40], Prometheus Agent [39], and Blackbox Exporter [8], was based on its alignment with the platform's pre-existing framework and its superior scalability, reliability, and manageability. With its capacity for automating configuration and management tasks and the Operator pattern [32] compatibility with container orchestrators such as Kubernetes [37] and OpenShift, the Prometheus Operator emerges as a particularly persuasive choice for implementation.

The goals focus on developing a scalable, reliable, and cloud-ready active monitoring platform within the Prometheus ecosystem, tailored explicitly for integration with the Amadeus Observability Platform. It emphasizes compatibility with various custom protocols, such as the TCIL protocol used at Amadeus, and seeks to ensure scalability through strategies like load balancing and sharding. The platform prioritizes reliability through fault tolerance, enabling seamless takeover by another prober in case of failure. Cloud readiness is achieved by ensuring compatibility with Amadeus's cloud infrastructure, particularly OpenShift clusters.

The design incorporates Kubernetes and OpenShift for container orchestration, focusing on deploying a modified Prometheus Operator to manage Prometheus Agents and probe configurations efficiently. This approach facilitates lightweight scheduling

and probing, leveraging Prometheus's remote write and scraping strategy feature to enhance efficiency and reduce peak request loads[36]. Additionally, implementing Blackbox Exporters and custom probers, like the TCIL Pinger, alongside load balancers aims to optimize load distribution and reliability.

The architecture supports active monitoring across various clusters and namespaces through customizable probes managed and deployed via Prometheus Operator and Prometheus Agent. This system architecture integrates seamlessly with the existing Amadeus Observability Platform and offers improved efficiency, high availability, and scalability.

In addition, Integrating Argo CD [3] and GitOps [48] offers a delicate approach to managing deployments on the OpenShift Platform, particularly for CRDs like PrometheusAgent and Probe. Argo CD automates deployments to ensure they align with configurations stored in Git. This process minimizes direct system access and ensures deployment consistency by aligning the actual state with the desired state in Git, making deployments more secure and auditable. While the GitOps approach streamlines deployment workflows and improves reliability, it comes with challenges, such as a steep learning curve and meticulous repository management to prevent configuration drift.

Overall, the proposed monitoring platform represents an improvement in active monitoring capabilities for the Amadeus Observability Platform. It is meticulously crafted to tackle the complex challenges of overseeing thousands of applications distributed across numerous clusters by leveraging modern container orchestration technologies alongside the Prometheus ecosystem. The emphasis on scalability, reliability, and cloud readiness ensures that the platform can adapt to the dynamic nature of cloud infrastructure while providing a robust and efficient monitoring solution. More than just a technical upgrade, this project highlights the critical role of active monitoring in preserving the reliability of expansive cloud-based applications. Through its cloud-native design and strategic implementation approaches, the monitoring platform is set to support the observability and operational efficacy of the Amadeus Observability Platform.

The concluding analysis contrasts the newly designed observability solution with Amadeus's original in-house monitoring application, focusing on their operational overhead within a proprietary OpenShift Cluster. A systematic comparison of crucial performance metrics—such as CPU, memory, and network usage—utilizing OpenShift built-in tools like cAdvisor [19] and the Network Metrics Daemon [31] forms the basis of this evaluation.

The observations reveal a minor difference in resource usage between the Prometheus Agent and Amadeus's Scheduler, even though they serve similar purposes. A detailed comparison of CPU usage highlights the superior operational efficiency of the Prometheus Agent. Significantly, the agent shows a notable improvement in memory

usage, saving around 140 MB more than the conventional scheduler, with minimal differences in network I/O.

The analysis also contrasts the Blackbox Exporter with Amadeus's HTTP Prober. While the Blackbox Exporter shows a slight increase in CPU consumption, this is balanced by its improved functionality and considerable memory saving, saving approximately 250 MB. The slight rise in network input is considered negligible and is not expected to negatively impact the system's scalability or overall performance.

Regarding the overhead associated with increasing targets, both the Prometheus Agent and the Blackbox Exporter exhibit linear growth trends, indicative of good scalability. The Prometheus Agent incurs more significant memory overhead and a sharper increase attributed to its tasks, such as scraping and managing configurations. Conversely, the Blackbox Exporter is more computationally demanding, with the bulk of its overhead from CPU usage. Therefore, the optimal metrics for configuring HPA should be memory for the Prometheus Agent and CPU for the Blackbox Exporter.

Additionally, an evaluation of the scraping strategies reveals the Prometheus Agent's superior approach in distributing network loads evenly over time, contrasting with the peak traffic patterns associated with Amadeus's Scheduler. This strategic load distribution suggests a more efficient and stable network traffic scenario, further validating the Prometheus Agent's implementation.

Beyond performance measurement, the Blackbox Exporter provides an array of detailed metrics crucial for troubleshooting and root cause analysis of service incidents. Key metrics like "probe_success," "probe_http_status_code," etc. offer insights into service health and HTTP response details. Remarkably, the "probe_http_duration_seconds" metric breaks down the HTTP connection into phases (resolve, connect, tls, processing, transfer), aiding in the identification of specific lifecycle issues. By comparing successful and failed probes, users can pinpoint where a problem occurs, such as a failure to progress beyond domain name resolution. These comprehensive metrics enhance service monitoring, troubleshooting capabilities, and overall observability.

Therefore, the shift towards the Prometheus Agent and Blackbox Exporter presents an acceptable overhead and significantly boosts memory efficiency while maintaining CPU and network performance. In addition to the scalability brought about by the Prometheus Operator, these resource utilization and load management enhancements substantially improve the overhead of the original in-house solution.

In summary, the thesis contributes to the existing knowledge on active monitoring by offering an in-depth look at the monitoring requirements of a vast and complicated IT ecosystem and proposing a scalable solution that leverages contemporary technology and methodologies. The comprehensive evaluation of system design, integration, and deployment process provides valuable perspectives on establishing and managing an active monitoring system.

9 Future Work

No matter how IT industries evolve, advancing system monitoring techniques cannot be overlooked. The thesis has laid the groundwork for understanding and enhancing the observability of complex IT infrastructures with active monitoring, emphasizing utilizing the Prometheus Operator and Blackbox Exporter. Nonetheless, a vast expanse of uncharted territory remains ripe for exploration. The following paragraphs introduce promising directions for future research, each offering the potential to significantly refine and advance our current monitoring capabilities. These directions include the automation of Blackbox Exporter configurations, developing interfaces for custom probers, and integrating eBPF programs for low-cost, high-precision monitoring.

A promising area for further investigation involves enhancing the Prometheus Operator's functionality, particularly in managing the Blackbox Exporter's configuration through CRDs. Automating this aspect would mark a significant leap toward more manageable and efficient monitoring systems, reducing the manual overhead for maintainers and aligning the Blackbox Exporter more closely with the automated, operator-managed setup of the Prometheus Agent. This evolution toward automated configurations promises to elevate the efficiency of monitoring operations and ensure a higher degree of flexibility and ease of maintenance.

Exploring an interface for creating and managing custom probers within the Blackbox Exporter framework represents another direction for research. The limitation to officially supported probers restricts the tool's applicability across diverse monitoring scenarios. By developing a customizable interface, users could employ monitoring solutions to their specific needs, significantly enhancing the tool's versatility and the relevance of the data it collects. This advancement would enable more delicate monitoring approaches, allowing for a broader application spectrum and more targeted data collection strategies.

Furthermore, the potential integration of eBPF programs into the Blackbox Exporter offers an exciting prospect for the future of monitoring. eBPF, known as Extended Berkeley Packet Filter, is a technology that allows efficient performance monitoring within the kernel space without requiring traditional overheads. Leveraging eBPF's ability to perform low-level, low-cost analyses could transform the observability landscape of active monitoring, providing insights into system performance with minimal overhead. This approach could lead to more effective and efficient monitoring strategies,

enabling a deeper understanding of system behaviors and facilitating more fine-grained performance improvement.

Beyond these specific areas, the continuous evolution of cloud computing and container technology presents ongoing opportunities for the refinement of monitoring strategies and tools. The integration of advanced machine learning techniques for predictive monitoring, the development of finer-grained tools for microservices architectures, and the investigation of automated system recovery mechanisms based on monitoring data are critical areas where future research could drive significant improvements. By pushing the boundaries in these directions, the field can anticipate and adapt to the changing landscape of IT infrastructures, ensuring that monitoring solutions remain robust, effective, and aligned with the needs of modern technologies.

In sum, the future work mentioned here is not just an extension of this thesis but a roadmap for advancing system monitoring into new realms of efficiency, flexibility, and observability. By pursuing these research directions, the field can look forward to monitoring solutions that are not only more capable but also more adapted to the dynamic complexities of modern IT environments.

Abbreviations

eBPF Extended Berkeley Packet Filter

ACA Application Centric Automation

ACS Amadeus Cloud Service

API Application Programming Interface

CA Certificate authority

CI Continuous Integration

CD Continuous Delivery

CLI Command-line Interface

CNCF Cloud Native Computing Foundation

CPU Central Processing Unit

CR Custom Resource

CRD Custom Resource Definition

DM Deployment Manager

DNS Domain Name System

EDIFACT Electronic Data Interchange for Administration, Commerce and Transport

HPA Horizontal Pod Autoscaler

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IaC Infrastructure as Code

ICMP Internet Control Message Protocol

IP Internet Protocol

IT Information Technology

NRPE Nagios Remote Plugin Executor

OSI Open Systems Interconnection

PaaS Platform as a Service

QoS Quality of Service

SNMP Simple Network Management Protocol

SOAP Simple Object Access Protocol

SRE Site Reliability Engineering

REST Representational State Transfer

TCIL Transport Cluster-Interconnect-Logic

TCP Transmission Control Protocol

TLS Transport Layer Security

URL Uniform Resource Locator

List of Figures

2.1 Black-Box Monitoring Illustration 4
2.2 Prometheus Agent with Scrape and Remote Write 6
2.3 Illustration of the Operator Pattern 8
2.4 Pull-based GitOps Workflow 9

3.1 System Design - Prometheus Operator, Prometheus Agent and Probers 12
3.2 System Workflow - PrometheusAgent Configurations 14
3.3 System Workflow - Probe Configurations 14
3.4 System Workflow - Probe Scheduling 15

4.1 Prometheus Agent Sharding 19
4.2 OpenShift Load Balancing 20
4.3 Utilizing GitOps to manage CRDs 21

6.1 Scheduler Analysis 39
6.2 Prober Analysis 40
6.3 Overhead Analysis 41
6.4 Scrape Analysis 42
6.5 Metrics Analysis 43

7.1 IBM's Cloud Monitoring 45
7.2 Amadeus' Active Monitoring 47
7.3 Prometheus's Monitoring Architecture 48
7.4 Datadog's Synthetic Monitoring 49
7.5 Nagios's Monitoring with NRPE plugin 49

Bibliography

[1] M. Anand. "Cloud Monitor: Monitoring Applications in Cloud." In: *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. 2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM). Bangalore, KA, India: IEEE, Oct. 2012, pp. 1–4. ISBN: 978-1-4673-4422-7 978-1-4673-4421-0 978-1-4673-4420-3. DOI: 10.1109/CCEM.2012.6354603.

[2] S. Arachchi and I. Perera. "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management." In: *2018 Moratuwa Engineering Research Conference (MERCon)*. 2018 Moratuwa Engineering Research Conference (MERCon). May 2018, pp. 156–161. DOI: 10.1109/MERCon.2018.8421965.

[3] *Argo CD - Declarative GitOps CD for Kubernetes*. URL: https://argo-cd.readthedocs.io/ (visited on 03/08/2024).

[4] V. Armenise. "Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery." In: *2015 IEEE/ACM 3rd International Workshop on Release Engineering*. 2015 IEEE/ACM 3rd International Workshop on Release Engineering (RELENG). Florence, Italy: IEEE, May 2015, pp. 24–27. ISBN: 978-1-4673-7070-7. DOI: 10.1109/RELENG.2015.19.

[5] Atlassian. *Bitbucket | Git Solution for Teams Using Jira*. Bitbucket. URL: https://bitbucket.org/product (visited on 03/11/2024).

[6] F. Beetz and S. Harrer. "GitOps: The Evolution of DevOps?" In: *IEEE Software* 39.4 (July 2022), pp. 70–75. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2021.3119106.

[7] 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.

[8] *Blackbox Exporter*. Prometheus.

[9] 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.

[10] *Build Software Better, Together*. GitHub. URL: https://github.com (visited on 03/11/2024).

[11] *Cloud Monitoring as a Service | Datadog*. URL: https://www.datadoghq.com/ (visited on 03/10/2024).

[12] *Cloud Native Computing Foundation*. CNCF. URL: https://www.cncf.io/ (visited on 03/10/2024).

[13] *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).

[14] *Custom Metrics Autoscaler Operator Overview - Automatically Scaling Pods with the Custom Metrics Autoscaler Operator | Nodes | OpenShift Container Platform 4.14*. URL: https://docs.openshift.com/container-platform/4.14/nodes/cma/nodes-cma-autoscaling-custom.html (visited on 03/10/2024).

[15] J. Dobies and J. Wood. *Kubernetes Operators*. Feb. 2020. ISBN: 978-1-4920-4803-9.

[16] *eBPF - Introduction, Tutorials & Community Resources*. URL: https://ebpf.io (visited on 03/10/2024).

[17] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. Ernst, and M.-A. Storey. "Uncovering the Benefits and Challenges of Continuous Integration Practices." In: *IEEE Transactions on Software Engineering* 48.7 (July 1, 2022), pp. 2570–2583. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2021.3064953. arXiv: 2103.04251 [cs].

[18] R. Filipe, R. P. Paiva, and F. Araujo. "Client-Side Black-Box Monitoring for Web Sites." In: *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA). Oct. 2017, pp. 1–5. DOI: 10.1109/NCA.2017.8171343.

[19] *Google/Cadvisor*. Google, Mar. 10, 2024.

[20] *Helm*. URL: https://helm.sh/ (visited on 03/07/2024).

[21] *Influxdata/Influxdb*. InfluxData, Mar. 10, 2024.

[22] *Jenkins*. URL: https://www.jenkins.io/ (visited on 03/07/2024).

[23] 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.

[24] S. M. Magda, A. B. Rus, and V. Dobrota. "Nagios-Based Network Management for Android, Windows and Fedora Core Terminals Using Net-SNMP Agents." In: *2013 11th RoEduNet International Conference*. 2013 11th RoEduNet International Conference. Jan. 2013, pp. 1–6. DOI: 10.1109/RoEduNet.2013.6511742.

[25] L. Mariani, M. Pezzè, and D. Zuddas. "Chapter Four - Recent Advances in Automatic Black-Box Testing." In: *Advances in Computers*. Ed. by A. Memon. Vol. 99. Elsevier, Jan. 1, 2015, pp. 157–193. DOI: 10.1016/bs.adcom.2015.04.002.

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

[27] *Nagios Open Source | Nagios Open Source*. URL: https://www.nagios.org/ (visited on 03/10/2024).

[28] *Nagios Plugins | Nagios Open Source*. URL: https://www.nagios.org/downloads/nagios-plugins/ (visited on 03/11/2024).

[29] 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.

[30] *NRPE - Nagios Remote Plugin Executor - Nagios Exchange*. URL: https://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details (visited on 03/11/2024).

[31] *Openshift/Network-Metrics-Daemon*. OpenShift, Aug. 19, 2023.

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

[33] K. Park, S. Sung, H. Kim, and J.-i. Jung. "Technology Trends and Challenges in SDN and Service Assurance for End-to-End Network Slicing." In: *Computer Networks* 234 (Oct. 1, 2023), p. 109908. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2023.109908.

[34] J. Perez-Espinoza, V. J. Sosa-Sosa, J.L.Gonzalez, and E. Tello-Leal. "A Distributed Architecture for Monitoring Private Clouds." In: *2015 26th International Workshop on Database and Expert Systems Applications (DEXA)*. 2015 26th International Workshop on Database and Expert Systems Applications (DEXA). Valencia, Spain: IEEE, Sept. 2015, pp. 186–190. ISBN: 978-1-4673-7581-8 978-1-4673-7582-5. DOI: 10.1109/DEXA.2015.51.

[35] *Perses/Perses*. Perses, Mar. 7, 2024.

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

[37] *Production-Grade Container Orchestration*. Kubernetes. URL: https://kubernetes.io/ (visited on 03/10/2024).

[38] *Prometheus - Monitoring System & Time Series Database*. Prometheus. URL: https://prometheus.io/ (visited on 03/10/2024).

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

[40] *Prometheus Operator*. Prometheus Operator. URL: https://prometheus-operator.dev/ (visited on 03/10/2024).

[41] Ramadoni, E. Utami, and H. A. Fatta. "Analysis on the Use of Declarative and Pull-based Deployment Models on GitOps Using Argo CD." In: *2021 4th International Conference on Information and Communications Technology (ICOIACT)*. 2021 4th International Conference on Information and Communications Technology (ICOIACT). Yogyakarta, Indonesia: IEEE, Aug. 30, 2021, pp. 186–191. ISBN: 978-1-66543-394-5. DOI: 10.1109/ICOIACT53268.2021.9563984.

[42] *Red Hat OpenShift Enterprise Kubernetes Container Platform*. URL: https://www.redhat.com/en/technologies/cloud-computing/openshift (visited on 03/10/2024).

[43] J. Roller. *The Argo Project: Making GitOps Accessible to Everyone*. IEEE Computer Society. May 20, 2022. URL: https://www.computer.org/publications/tech-news/trends/the-argo-project/ (visited on 03/11/2024).

[44] M. Shahin, M. Ali Babar, and L. Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices." In: *IEEE Access* 5 (2017), pp. 3909–3943. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2685629.

[45] *Splunk | The Key to Enterprise Resilience*. Splunk. URL: https://www.splunk.com (visited on 03/10/2024).

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

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

[48] *Weaveworks/Weave-Gitops: Weave GitOps Provides Insights into Your Application Deployments, and Makes Continuous Delivery with GitOps Easier to Adopt and Scale across Your Teams.* URL: https://github.com/weaveworks/weave-gitops (visited on 03/10/2024).

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

[50] H. Yongdnog, W. Jing, Z. Zhuofeng, and H. Yanbo. "A Scalable and Integrated Cloud Monitoring Framework Based on Distributed Storage." In: *2013 10th Web Information System and Application Conference*. 2013 10th Web Information System and Application Conference (WISA). Yangzhou, China: IEEE, Nov. 2013, pp. 318–323. ISBN: 978-1-4799-3219-1 978-1-4799-3218-4. DOI: 10.1109/WISA.2013.66.