TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Karl-Andreas Turvas 194145IACB

MONITORING A DYNAMIC HASHICORP NOMAD ORCHESTRATED CONTAINER ENVIRONMENT

Bachelor's Thesis

Supervisor: Ali Ghasempour
MSc

Co-supervisor: Lauri Anton
BSc

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Andreas Turvas 194145IACB

DÜNAAMILISE HASHICORP NOMADI ORKESTREERITUD KONTEINERKESKKONNA MONITOORIMINE

Bakalaureusetöö

Juhendaja: Ali Ghasempour
MSc
Kaasjuhendaja: Lauri Anton
BSc

Tallinn 2024

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Karl-Andreas Turvas

07.05.2024

Abstract

HashiCorp's Nomad serves as an orchestration platform facilitating the configuration and coordination of various workloads, including containerized tasks—a central focus of this thesis.

The primary objective of this research was to establish a monitoring solution for Nomad within the author's home laboratory environment. This endeavor aimed to deepen comprehension of dynamic containerized infrastructures while enhancing the monitoring capabilities of the author's laboratory setup.

Consequently, leveraging Prometheus and Grafana at its core, a monitoring system was implemented and deployed. This system provided timely notifications through defined and tuned alerts, along with bespoke and community-driven dashboards for enhanced visualization and deeper understanding of operational issues and performance metrics. Additionally, it offered a convenient means to query logs including for ephemeral and dynamic containers, alleviating the otherwise cumbersome process of accessing logs.

Version controlled code repositories utilized in the making of this thesis can be found publicly at GitHub for Packer [1], Terraform [2], and Ansible [3].

The thesis is written in English and is 40 pages long, including 6 sections, 13 figures and 1 table.

Annotatsioon
Dünaamilise HashiCorp Nomadi orkestreeritud konteinerkeskkonna monitoorimine

HashiCorp'i Nomad on orkestreerimisplatvorm, mis hõlbustab erinevate teenuste, sealhulgas - käesoleva töö keskmes oleva - konteineriseeritud teenuste konfigureerimist ja koordineerimist.

Käesoleva lõputöö esmane eesmärk oli luua Nomadile seirelahendus autori kodulabori keskkonnas. Selle ettevõtmise eesmärk oli süvendada teadmisi dünaamiliste konteineriseeritud infrastruktuuride kohta, parandades samal ajal autori laboratooriumi seirevõimalusi.

Töö tulemusena rakendati ja võeti kasutusele seiresüsteem, mille keskmes on Prometheus ja Grafana. See süsteem pakkus õigeaegseid teavitusi häälestatud hoiatuste kaudu ning visualiseerimisvõimalusi, et saada anda parem ülevaade operatiivsetest probleemidest ja jõudlusnäitajatest. Lisaks pakkus see mugavat lahendust logide lugemiseks, leevendades muidu efemerentsete ja dünaamilistele konteineritele omast tülikat logidele juurdepääsu protsessi.

Selle töö koostamisel kasutatud versioonihaldusega koodirepositoriumid on avalikult kättesaadavad GitHubis Packeri [1], Terraformi [2] ning Ansible jaoks [3].

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 40 leheküljel, 6 peatükki, 13 joonist, 1 tabelit.

List of Abbreviations and Terms

API	Application Programming Interface
CNCF	Cloud Native Computing Foundation
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
TLS	Transport Layer Security
SSL	Secure Sockets Layer
JSON	JavaScript Object Notation
MB	Megabyte
Metric	Time series that is defined by its unique name and optional key-value pairs called labels
PromQL	Prometheus Query Language
REST API	Restful Application Programming Interface
Scraping	Prometheus configuration task, what defines the endpoints, where data should be retrieved.
SSH	Secure Shell
TSDB	Time series database
UID	Unique Identifier
VM	Virtual Machine
YAML	YAML Ain't Markup Language, a recursive acronym for a human-readable data-serialization language
CPU	Central Processing Unit
RAM	Random Access Memory
IaC	Infrastructure as Code

Table of Contents

List of Figures

List of Tables

1. Introduction

The author independently manages a private computing environment (referred to as home laboratory and homelab henceforth) using cost-effective hardware and open-source software. By self-hosting applications, the homelab achieves autonomy and reduces dependence on external service providers with increased data privacy. Moreover, the homelab serves as an experiential playground facilitating learning opportunities, which will be the focus of this thesis. In essence, the author's IT homelab allows a blend of control, cost efficiency, learning, and deployment capabilities, empowering them to tackle the complexities of information technology with resilience and innovation.

Cloud Native Computing Foundation (CNCF) is a nonprofit organization that hosts and integrates open-source, vendor-neutral projects for container orchestration and microservices architectures in an effort to to accelerate the adoption of said technologies. In their annual surveys, CNCF has noted tremendous yearly increases in container and Kubernetes container orchestrator use in production to the point where containerization and orchestration can be considered part of the industry standard in managing services [4, 5, 6, 7], leading also to influence the development of the Linux kernel to support container features [8]. Just as CNCF focuses mostly on Kubernetes, so do all major cloud providers offer Kubernetes based services for orchestration [9, 10, 11]. Perhaps because of its complexity and wide scope Kubernetes has gained de-facto status for orchestration solutions [12]. However the complexity comes with a cost that many organizations are not ready to pay. For example the steep learning curve and rapid developments in Kubernetes add complexity to upgrade processes which have lead smaller teams to seek alternatives like the Nomad orchestrator, because as Endler puts it: *"If Kubernetes were a car, Nomad would be a scooter. Sometimes you prefer one and sometimes the other"* [13]. HashiCorp, the company behind Nomad, echoes Endler's description and emphasizes the simplicity in usage and maintainability compared to Kubernetes [14]. Nomad targets operational overhead more suited to small teams and on-premise environments by adhering to the Unix philosophy of maintaining a concise scope requiring administrators to add each component separately (providing a good learning experience as well as facilitating simpler deployments) instead of the all-batteries-included approach of Kubernetes [15, 16]. Author's one man homelab fits both the small team and on-premise bills making Nomad an attractive choice for learning about container orchestration. As such the author deployed a highly-available Nomad cluster as an experiment to evaluate the viability of orchestration in the context of a home laboratory and gain deeper understanding into orchestration as a whole.

The aim of this thesis was to further investigate orchestration within the realm of monitoring, addressing the unique challenges posed by the dynamic and ephemeral nature of orchestrated containers. This necessitated the deployment of a monitoring architecture distinct from traditional approaches used for plain containers or classical on-host services.

This thesis consists of 6 main sections including the introduction section you are reading now. The second section aims to give a short overview of the preceding context that lead to this thesis and explanation of the problems being solved. The third section will cover the implementation to solve the problems according to the requirements laid out in the section before it. The fourth section goes over the outcomes of the work. The fifth section will give some suggestions on further improvements to the implementation. Eight and final section is the summary.

2. Background

This chapter will provide an overview of technologies, and definitions, offering better insight into the orchestration platform and the proposed monitoring system.

2.1 Reproducibility

Packer [17], Terraform [18], and Ansible [19] are open source tools that automate infrastructure and were employed to configure all aspects discussed in this thesis, including the infrastructure for the Nomad orchestration. The work implicitly assumes that the all systems described were declared and managed through a reproducible infrastructure-as-code approach using the mentioned tools. Version controlled code repositories utilized in the making of this thesis can be found publicly at GitHub for Packer [1], Terraform [2], and Ansible [3].

2.2 Monitoring Requirements

IEEE defines **monitoring** as the supervising, recording, analyzing or verifying the operation of a system or component [20]. In dynamic environments such as container orchestration, where containers can be distributed across various machines within a cluster and can be assigned to different ports dynamically, monitoring becomes significantly more complicated *and* increasingly critical [21].

In containerization, each service is usually relatively simpler compared to monolithic systems. However, the interactions between these services and their underlying dependencies, along with the dynamic nature of the components within the distributed system, introduce additional operational challenges. Due to the constant changes and unpredictability in a dynamic environment like container orchestration, attempting to manage it manually becomes impractical, as a human operator would struggle to keep up with the rapid pace of changes, allocating resources, and ensuring the proper functioning of services across different nodes and ports thus requiring a monitoring system more sophisticated than manual digging [21].

IEEE defines **metrics** as a quantitative measure of the degree to which a system, component, or process possesses a given attribute [20]. In essence, metrics show what is happening in a system, component or process: the first part of the story of change mentioned in the last paragraph.

Developers insert **logging** statements into the source code which are then printed into log files, also known as execution logs and event logs [22]. Then, at a later time, while the system is running or during postmortem, operators can analyze the log files for why and when something was happening in a system, component or process. Logging is the second part of the story of change.

By collecting and correlating series of events to answer what, why and when an event occurred, it is possible to conclude an informative story of change that will help to understand systems as well as reveal root causes for changes in systems, this is the essence of monitoring as suggested by the authors of OpenTelemetry, a CNCF observability framework and toolkit [23]. For instance, elevated CPU usage in a service might stem from an underlying dependency change, from metrics we see the elevated CPU usage and from logs we will see what happened that could have caused this.

While gathering metrics and logs is essential for obtaining critical data, two additional components buttress a successful monitoring system, making the data actionable: alerts and visualization.

If abnormalities go unnoticed and are not promptly addressed, they have the potential to result in significant system failures and incidents. Therefore, the timely detection, intervention, and mitigation of anomalies are crucial for ensuring the reliability of systems. To achieve this, monitoring systems should usually issue alerts based on metrics for changes that require human intervention. In a system context, an **alert** signifies a notification indicating a specific abnormal condition or anomaly within the system [24]. An alert strategy dictates the criteria for alert generation, including when alerts should be triggered, the attributes and descriptions they should possess, and the intended recipients of the alerts.

Visualizations aid in comprehending systems through graphical representations are constructed and utilized to bolster in practically all data-informed decision-making processes over many fields enhancing clarity and insight for humans. In essence, visualizations serve to help in decision making as well as efficient communication and learning [25].

Based on the previous considerations, we arrive at the requirements for the monitoring system. The monitoring system must:

- be able to accommodate dynamic orchestrated systems
- possess capabilities for central metric and log collection.

- be equipped to generate alerts.
- have visualization capabilities.

2.3 HashiCorp Nomad Orchestration System

This section explains how the Nomad orchestration system works.

Nomad, developed by HashiCorp, is a scheduler and orchestrator for deploying and managing containers and non-containerized applications across on-premises and clouds at scale [26].

Consul, also developed by HashiCorp, is a software that provides service discovery, health checking, dynamic configuration, and service mesh for distributed applications [27]. Consul can be used as part of Nomad's architecture, as Nomad was intentionally crafted to prioritize cluster management and scheduling exclusively, omitting non-core features to ensure it can operate without dependencies as a single process. Consul can integrate with Nomad to provide it a storage back-end, networking functionality, automatic clustering, service registration, and service discovery [15].

2.3.1 Nomad Architecture

To explain the architecture of Nomad, some terms have to be explained.

A **Nomad agent** refers to a Nomad process functioning either in server or client mode. A **Nomad client** is tasked with executing assigned jobs while also enrolling itself with servers and monitoring task assignments. During agent operation, the client may alternatively be denoted as a node. A **Nomad server** oversees all job and client management, task monitoring, and controls task distribution across client nodes. The servers ensure high availability by replicating data among each other [28].

Task drivers are the run-time components for a Nomad client, a task driver can be Docker, Podman, Java, exec, QEMU, etc [15].

A **Consul agent** is also a Consul process running in server or client mode. A **Consul server** is an agent is responsible for maintaining cluster state, responding to queries, and participating in the consensus protocol. The **Consul clients** query the servers for information and provide local services with access to the Consul features. To achieve high availability, data replication is done between the servers [29].

Now to put Nomad and Consul together:

- a client node, as seen on Figure 1, runs:
  - a Nomad client agent
  - task driver(s)
  - a Consul client agent
- a server node, as seen on Figure 2, runs:
  - a Nomad server agent
  - a Consul server agent

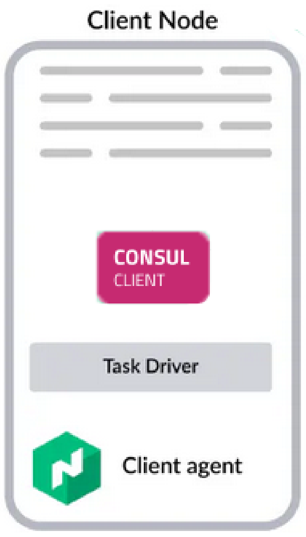


Figure 1. High-level overview of a Nomad client node [15]

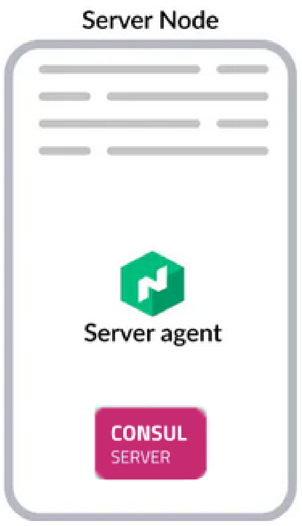


Figure 2. High-level overview of a Nomad server node [15]

Nomad and Consul organize infrastructure into regions and datacenters. A **datacenter** is conceptualized as a grouping of clients within a region. While clients don't necessarily need to be in the same datacenter as the servers they're associated with, they must be within the same region. Datacenters facilitate fault tolerance among jobs and infrastructure isolation. A region encompasses one or more datacenters. A collection of joined servers represents a single **region** [30].

A Nomad **cluster** typically consists of three to five server nodes and numerous client nodes. An example of the reference architecture by HashiCorp, depicted in Figure 3, illustrates a single datacenter containing a cluster of three servers and three clients [31], this is the same architecture utilized by the author to deploy Nomad.

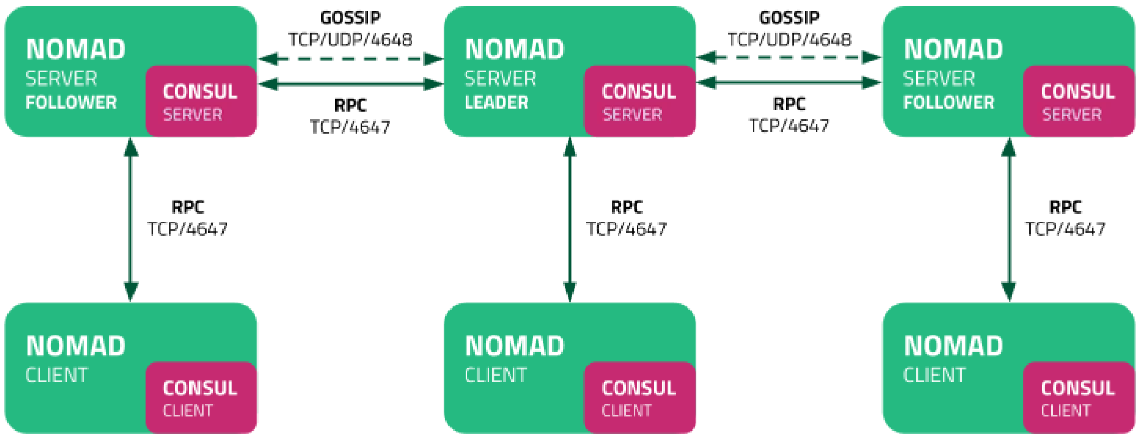


Figure 3. Reference architecture of a Nomad cluster in a single datacenter [31]

In Nomad, a **task** represents the smallest unit of work, executed by task drivers such as Docker or exec, enabling Nomad's flexibility across task types. Tasks specify their required driver, driver configuration, constraints, and resource needs. A **group** consists of tasks executed on the same Nomad client. A **job** serves as Nomad's core control unit, defining an application and its configurations, potentially comprising multiple tasks. A **job specification**, or jobspec, outlines the schema for Nomad jobs, detailing the job type, necessary tasks and resources, job metadata like eligible clients, and more. An **allocation** maps task groups within a job to client nodes. When a job is initiated, Nomad selects a capable client and allocates resources on the machine for the task(s) defined in the job's task group(s). An **application** is described within a jobspec with task groups, and upon submission to Nomad, a job is instantiated alongside allocations for each group specified in the jobspec [30].

A visual of a Nomad job containing two task groups, one with one task and a second one with two tasks, can be seen on Figure 4.

An orchestrator allocates a task group to any client node, typically dynamically, although static assignment is also possible. Each task group instance utilizes the node's own IP

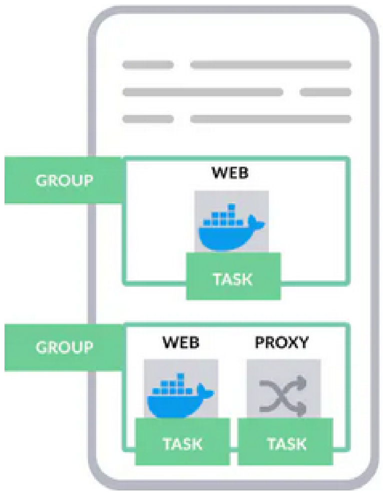


Figure 4. Nomad job containing two task groups [31]

network and receives a port through static or dynamic port assignment, as illustrated in Figure 5. There's no need for a virtual IP or an additional overlay network; the Nomad cluster network can integrate into an existing network without requiring a proxy [15]. Nomad jobs conform to predefined desired states. Client nodes determine the resulting state and manage any failures within the system. Should failures occur, Nomad initiates a new evaluation to assess the cluster's state and bring it in line with the desired state [32].

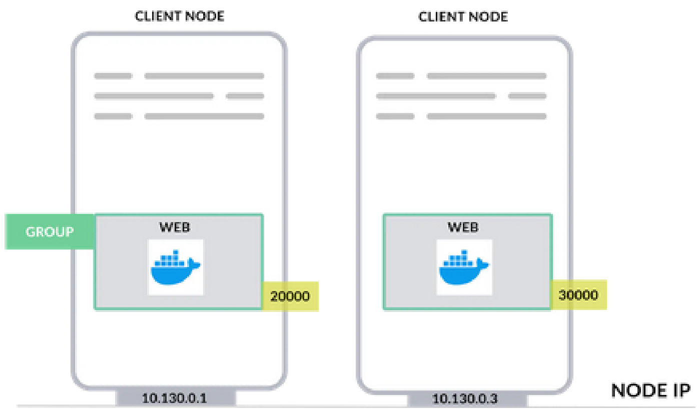


Figure 5. Nomad job port allocation example [31]

2.3.2 Monitoring Nomad

In Section 2.3.1, the primary challenge in monitoring a Nomad cluster and its services becomes evident: operators cannot assume a predictable, static allocation of nodes and ports for any submitted job. Consequently, the monitoring infrastructure must accommodate this dynamic nature.

In Section 2.2, metrics were highlighted as a crucial component of monitoring, with alerts and visualization also directly depending on metrics. Prometheus formatted metrics from the a provided API endpoint is the only native way to export metrics that both Nomad and Consul provide [33, 34]. Its native support ensures compatibility and potentially more reliable metric collection compared to third-party alternatives and ease of integration with existing metric collection tools. The metric collection tool chosen for the monitoring system should be compatible with the Prometheus format.

The second major part of monitoring mentioned in Section 2.2 was logging. There are two parts to accessing logs from a Nomad cluster:

- Nomad and Consul agent logs
- Task logs

Logs from agents are essential for understanding the operation of the system that runs container services. When it comes to agent logs, the process is straightforward: utilizing syslog for logging is a built-in solution for both Consul and Nomad [35, 36].

However, the method and location of a task's logs depends on the chosen task driver. In this case, the Podman task driver is utilized for container orchestration purposes [37]. Podman does not provide syslog support as a log driver; instead, journald serves as the default log driver for Podman [38].

Given the requirement to manage both agent and task logs, the logging infrastructure for the monitoring system should be capable of supporting both syslog and journald.

In summary, the selected metric collection tool for the monitoring system must be compatible with the Prometheus format. Additionally, to accommodate both agent and task logs, the logging infrastructure of the monitoring system should support both syslog and journald.

3. Implementation

This section will cover the implementation of the monitoring system for the Nomad orchestration cluster.

3.1 Prometheus

The Prometheus format, as discussed in Section 2.3.2, is supported by Nomad and originates from the Prometheus monitoring toolkit [39, 40]. Prometheus is a pull-based monitoring system designed for many types of different environments, also featuring dynamic service discovery options. It is open source and resource-efficient, so Prometheus stands out as a preferred option in the CNCF Technology Radar survey [41] and remains a popular choice among open-source projects, particularly for organizations seeking cost-effective monitoring solutions [42].

The **Prometheus server** consists of two components: **data transfer server** and a **Time Series Database (TSDB)** for data storage. Prometheus uses **scraping**, a procedure that pulls data from exporter endpoints and modifies it prior to TSDB storage. An **exporter** is a binary that operates alongside or from inside an application from which metrics are desired and exposes Prometheus metrics by either converting metrics available in a non-Prometheus format into a format supported by Prometheus or providing Prometheus formatted metrics directly [43].

A Prometheus web-interface is provided for querying and displaying metrics using **Prometheus Query Language (PromQL)** is provided by the Prometheus server, but it doesn't have visualization capabilities [43]. However, the raw metrics themselves often fall short of offering meaningful interpretations due to their lack of context, thus Prometheus' primary function is metric data storage; analysis and visualizations should be handled elsewhere.

All data is kept in Prometheus as **time series**, which are collections of data points arranged chronologically. A unique identifier, along with optional key-value pairs known as **labels**, is assigned to each measure in a time series. Labels serve to distinguish the specific attributes of the measured entity [44]. Labels play a crucial role in maintaining the performance efficiency of Prometheus while ensuring user-friendliness.

A visual overview of the Prometheus components can be seen on Figure 7.

3.1.1 Metrics Collection

Nomad and Consul both export Prometheus metrics as part of the services and do not require additional exporters [45, 46]. Nomad also provides metrics for its allocations (in this case Podman containers) and as such does not require a separate exporter for containers, although cAdvisor, short for container Advisor, is the preferred solution recommended by Prometheus for analyzing and exposing resource usage and performance data from active containers [47]. As cAdvisor provides a wider variety of metrics [48], it was deployed to all client nodes to further improve on container metrics.

Prometheus server needs to be deployed and instructed to scrape Nomad and Consul metrics using both static definitions and using supported Consul service discovery [49]. This method allowed scraping services that Consul reported as being alive.

While the exporters of Nomad and Consul mostly give metrics about the inner workings of their respective services, they do also provide metrics about their hosts as well [45, 46]. However the provided Prometheus **Node Exporter** [50] gives a more comprehensive list of metrics [51] so Node Exporters were installed on every node to reveal an extensive range of hardware- and kernel-related metrics. This enables the monitoring of hosts that run under the containerized services in more detail.

In summary a variety of exporters are deployed and configured to be scraped by the Prometheus server. The collected metrics are:

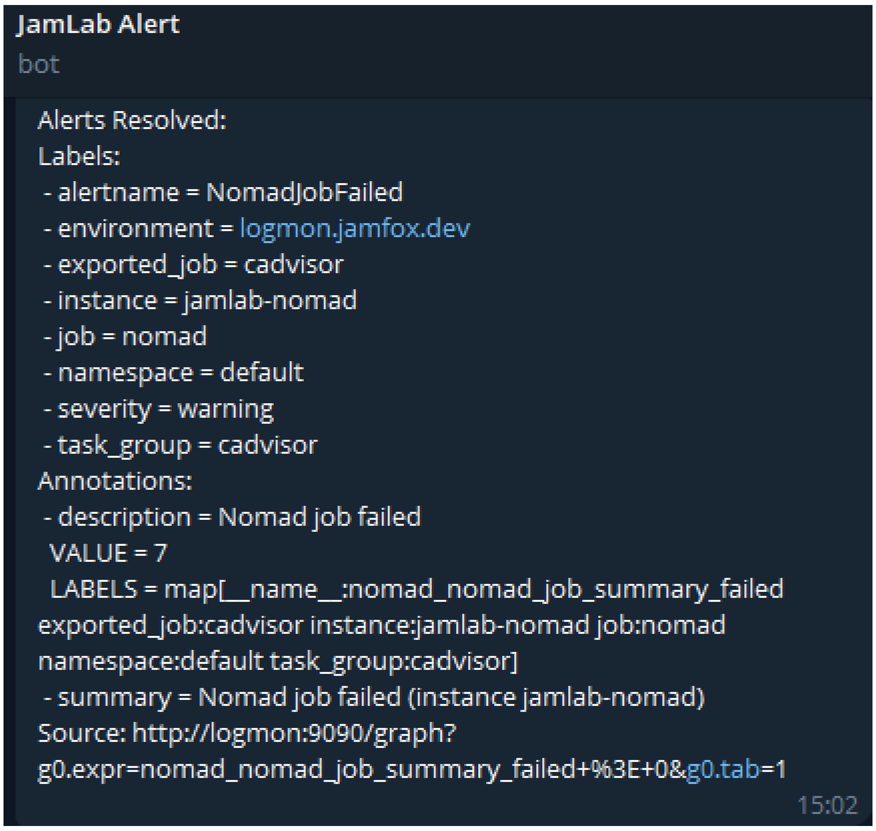
- Nomad metrics (including some container and host metrics) using its built in Prometheus metrics endpoint [45]
- Comprehensive container metrics using cAdvisor [48]
- Consul metrics (including some container and host metrics) using its built in Prometheus metrics endpoint [46]
- Comprehensive hardware- and kernel- and OS-related metrics using Node Exporter [51]

3.1.2 Alerting with Alertmanager

Prometheus Alertmanager enables the generation of alerts utilizing measurements and data ingested by Prometheus. Within Prometheus configuration the webhook functionality is provided for dispatching alarms and integration capabilities with various alerting services [52].

Prometheus alerting operates in two distinct phases. Alerting rules within Prometheus server trigger alerts directed to an Alertmanager instance. Subsequently, the Alertmanager assumes responsibility for overseeing these alerts, performing tasks such as silencing, inhibition, deduplication, aggregation, and propagation of notifications through channels like email, and chat platforms [53].

Using the metrics mentioned in Section 3.1.1, alert rules were set up to notify the author in Microsoft Teams and Telegram in case of various changes that might be of interest [54]. Figure 6 shows an example of a Telegram alert notification triggered by a Nomad job resolution event of a previously failed job allocation.


JamLab Alert
bot

```
Alerts Resolved:
Labels:
 - alertname = NomadJobFailed
 - environment = logmon.jamfox.dev
 - exported_job = cadvisor
 - instance = jamlab-nomad
 - job = nomad
 - namespace = default
 - severity = warning
 - task_group = cadvisor
Annotations:
 - description = Nomad job failed
  VALUE = 7
  LABELS = map[__name__:nomad_nomad_job_summary_failed
exported_job:cadvisor instance:jamlab-nomad job:nomad
namespace:default task_group:cadvisor]
 - summary = Nomad job failed (instance jamlab-nomad)
Source: http://logmon:9090/graph?
g0.expr=nomad_nomad_job_summary_failed+%3E+0&g0.tab=1
15:02
```

Figure 6. Example a Telegram notification showing a resolved fail of a Nomad job

One notable observation about Nomad allocation status design was revealed with alerts. The persistence of failed job alerts even after resolution poses a small challenge, prompting the need for systematic measures. These include scheduling regular garbage collection tasks or manually clearing failed statuses following issue resolution to ensure the accuracy of alert notifications.

3.1.3 Prometheus Summary

The layout of Prometheus components, as showcased in Figure 7, provides a visual representation of the workflow described in previous chapters of this section. Within this architecture, Prometheus exporters supply metrics for the Prometheus server to scrape. After that, the scraped data is used for alert generation, with the Alertmanager orchestrating the sending of notifications to the system operator.

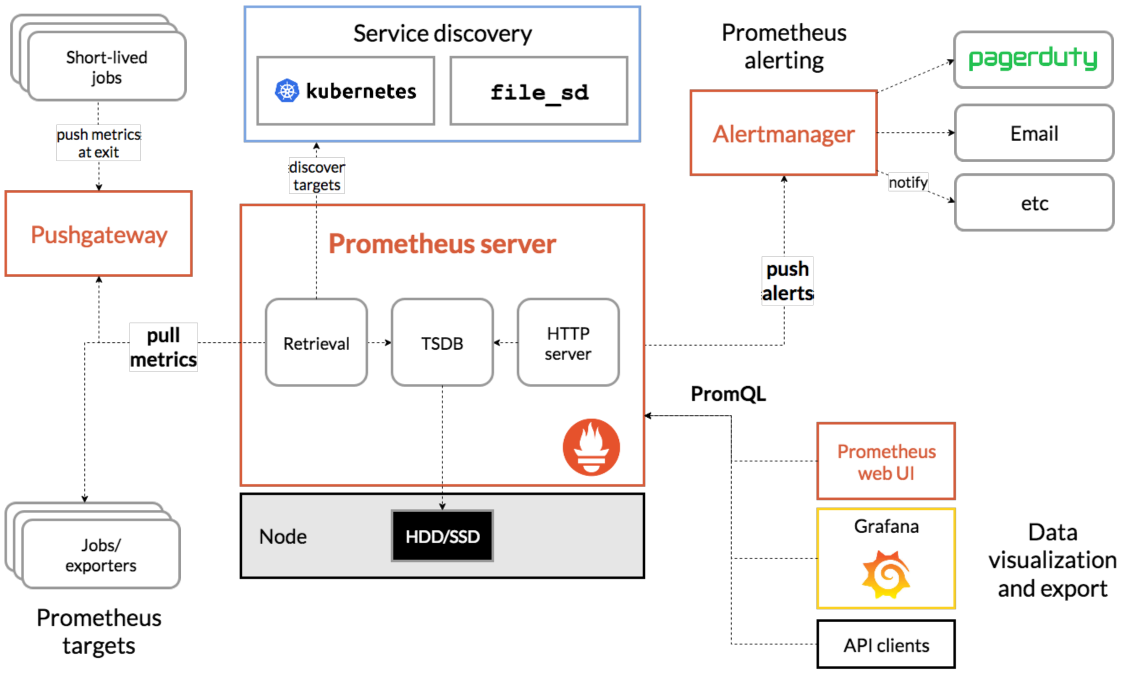


Figure 7. Example architecture of Prometheus components [40]

Considering the aforementioned chapters on Prometheus it is evident that Prometheus provides the functionalities to satisfy the requirements of metrics aggregation, storage, and alerting mentioned in Section 2.2. While Prometheus excels in these domains, it falls short in fulfilling the requirements for log aggregation and visualization. By complementing Prometheus with a suitable log aggregation tool and visualization platform, it is possible to establish a more comprehensive monitoring ecosystem that encompasses both metrics and log data, thereby ensuring holistic visibility and insights into system performance and health.

3.2 Grafana

Grafana is an open-source platform that facilitates the visualization and analysis of data from diverse sources in real-time, serving as a tool for gaining insights into metrics [55]. With its interface and extensive plugin ecosystem, Grafana offers flexibility for many a different visualizations. Grafana supports querying Prometheus and is the visualization solution recommended by Prometheus [56].

Grafana was deployed and Prometheus integration was incorporated as a readable datasource within Grafana, facilitating the availability of all metrics gathered by Prometheus for comprehensive analysis and visualization purposes.

3.2.1 Metric Visualization with Grafana Dashboards

Grafana includes a diverse range of panels, which simplifies the process of formulating appropriate queries and customizing visualizations to create an ideal dashboard for various requirements. Each panel possesses the capability to interact with data sourced from any configured Grafana data source [57].

In addition to facilitating the creation of bespoke dashboards, Grafana Labs maintains a repository of community-shared dashboards which can be downloaded and used [58].

Expounding further on Section 3.1.1, Node Exporters were used to monitor the health of hosts using hardware and operating system metrics. A small example of the visualization facilitated by Node Exporter metric data can be seen on Figure 8. To provide a concise summary of the 242 panels featured in the utilized Node Exporter dashboard [59], it encompasses overviews of aggregate CPU, RAM, swap, disk, and network traffic metrics and additionally the dashboard has more specialized panels tailored for in-depth examination across various categories.

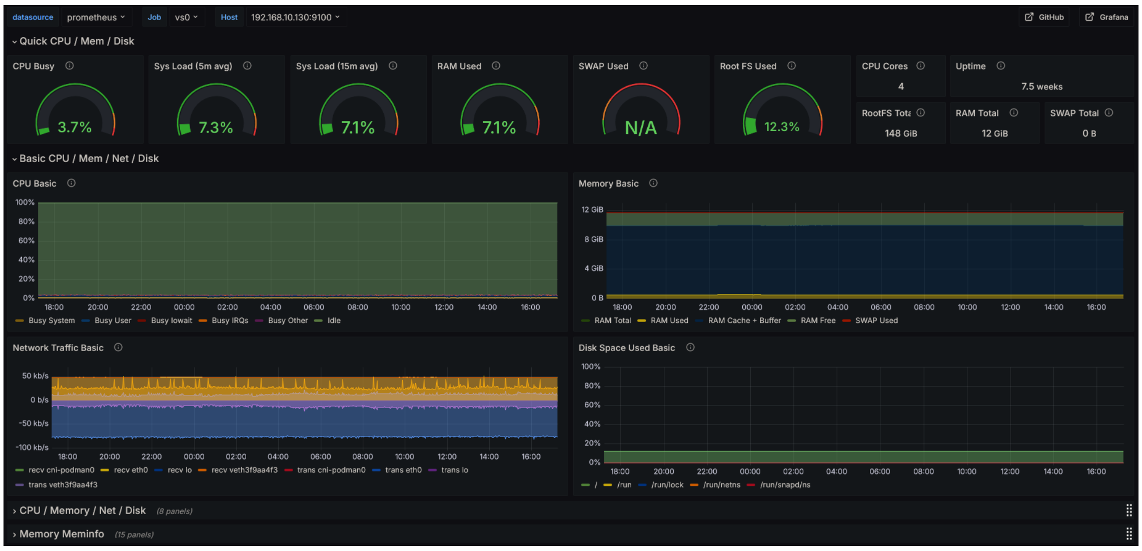
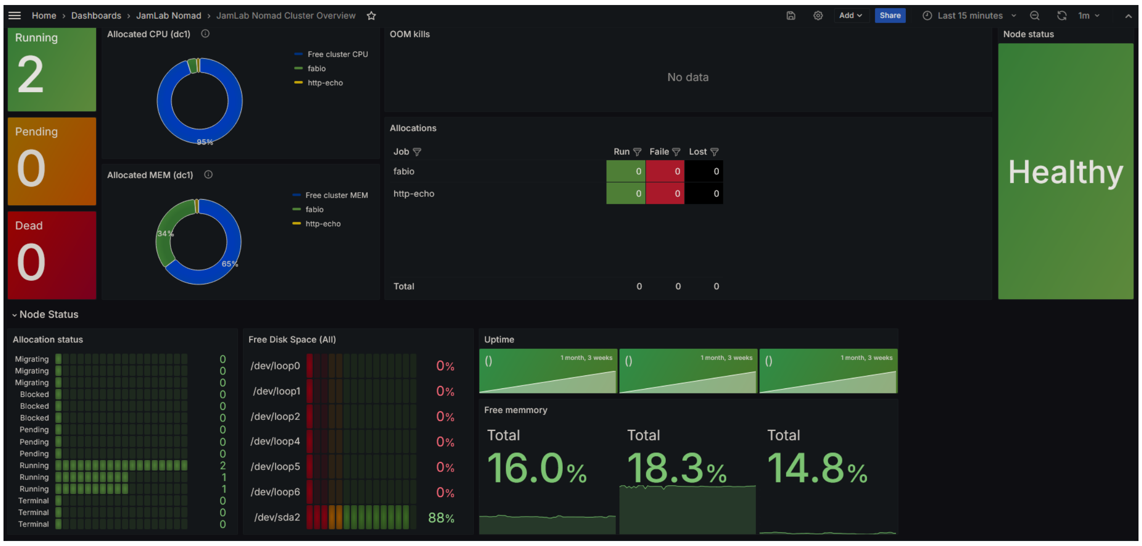


Figure 8. Snippet of one node's Node Exporter dashboard

Three Nomad dashboards sourced from the Grafana Labs Dashboard repository were adapted to provide a comprehensive representation of the Nomad cluster's status and allocation statuses. These dashboards included a general cluster overview dashboard [60], a more detailed dashboard focusing on allocations [61], and a dashboard specifically detailing the Nomad control plane [62]. Among these, the simplified dashboard offered a particularly insightful snapshot of the Nomad cluster's operational state, encompassing information on resource usage per job and their respective statuses at a glance, as seen on Figure 9.



Allocations

Job	Run	Faile	Lost
fabio	0	0	0
http-echo	0	0	0
Total	0	0	0

Allocation status	
Migrating	0
Migrating	0
Migrating	0
Blocked	0
Blocked	0
Blocked	0
Pending	0
Pending	0
Pending	0
Running	2
Running	1
Running	1
Terminal	0
Terminal	0
Terminal	0

Free Disk Space (All)	
/dev/loop0	0%
/dev/loop1	0%
/dev/loop2	0%
/dev/loop4	0%
/dev/loop5	0%
/dev/loop6	0%
/dev/sda2	88%

Figure 9. Snippet from the Nomad dashboard

The Consul dashboard, managed by HashiCorp [63], was employed for monitoring purposes. An illustration of this dashboard is provided in Figure 10. The dashboard was used have an overview of the health and performance of Consul servers and the services Consul manages. It facilitates the monitoring of key Consul server metrics such as service health, network activity, and Raft events providing valuable visibility for operational analysis and troubleshooting purposes.

The cAdvisor dashboard was utilized for container monitoring purposes [64]. A snippet of this dashboard can be seen in Figure 11. It provides a detailed overview of containers within the system, including those not managed by Nomad. It offered selectors for filtering by different hosts and interfaces observed in cAdvisor instances, enhancing monitoring capabilities in containerized environments.

The utilization of Grafana dashboards provide significant advantages in monitoring and understanding complex systems like Nomad orchestration. These dashboards serve as visual aids that provide a holistic view of system health and performance metrics, enabling quick identification of trends, anomalies, and potential issues. Unlike direct queries in

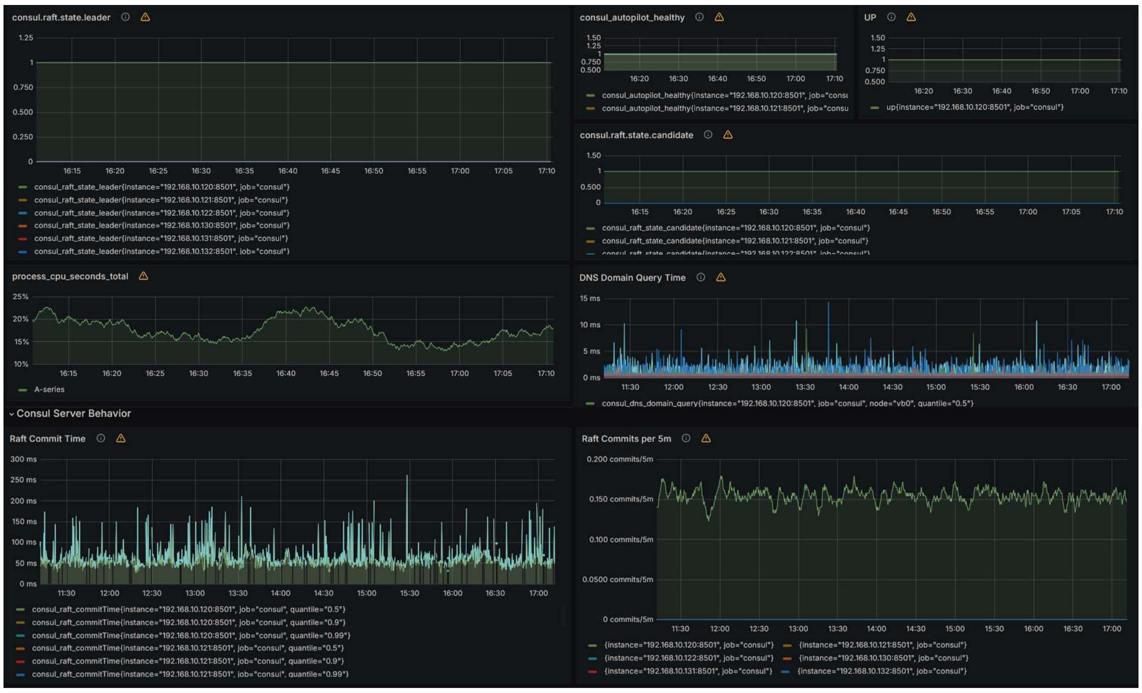
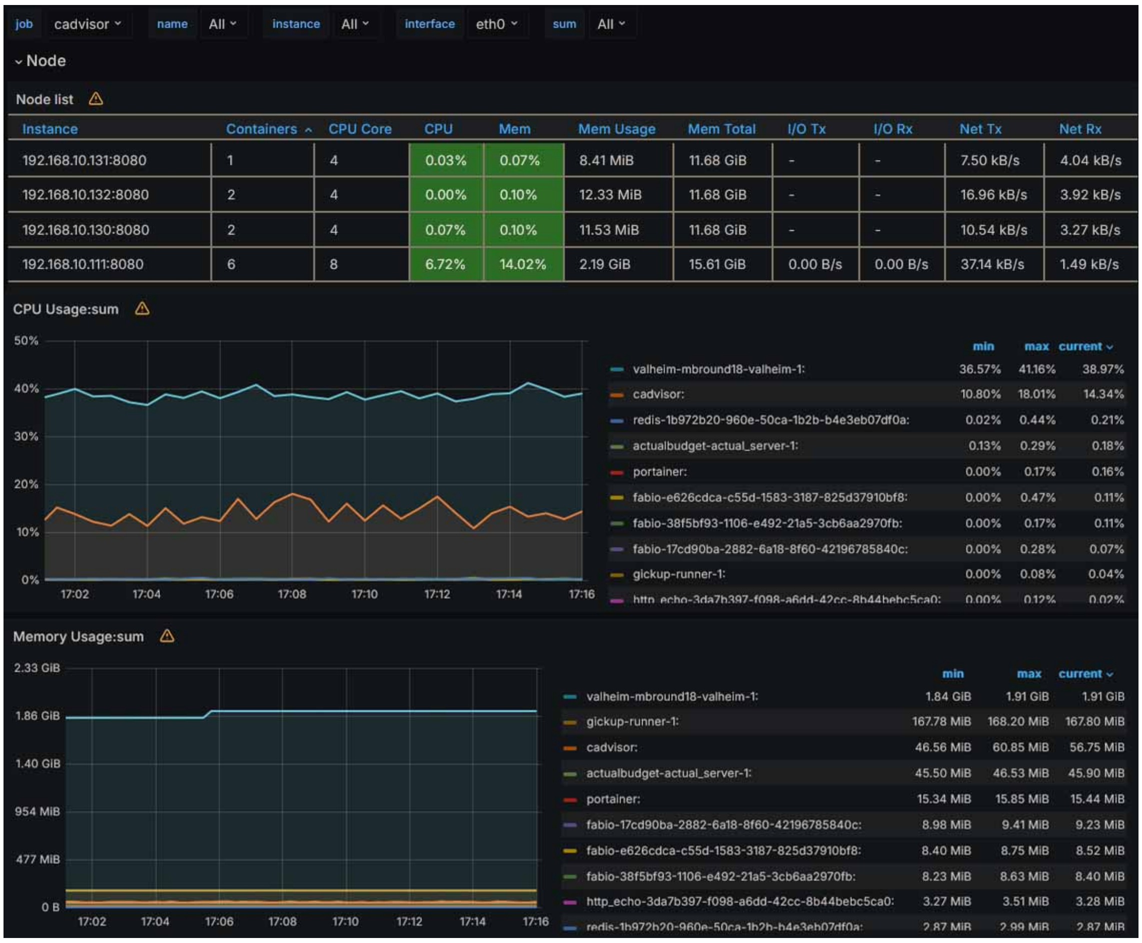


Figure 10. Snippet from the Consul dashboard

Prometheus which require interpreting raw data outputs, dashboards present information in a more intuitive manner, facilitating rapid comprehension and decision-making. With the aforementioned dashboards it was possible to better understand cluster status and allocation statuses, behaviour and resource usage of containers, including those not managed by Nomad, and insight into underlying node health. Overall, these dashboards enhance situational awareness by presenting data in a visually digestible format, offering a sort of zoom out effect for monitored components.

3.2.2 Log Analysis with Loki

Grafana Loki is an open-source log aggregation system, drawing inspiration from Prometheus, incorporates exceptionally compact indexing, and highly compressed log data [65]. Loki is designed to be as simple to use and inexpensive as possible; instead of indexing log contents, it labels individual log streams by leveraging labels similarly to Prometheus, mentioned in 3.1. The shared design with Prometheus labels at the center allows for correlation between metrics and logs. In practical terms, Prometheus metric labels are correlated with Loki's log labels, establishing a connection between the "what" (metrics) and "why" (logs) aspects of system behavior. This correlation enables operators to trace events in metrics back to the logs.



Instance	Containers	CPU Core	CPU	Mem	Mem Usage	Mem Total	I/O Tx	I/O Rx	Net Tx	Net Rx
192.168.10.131:8080	1	4	0.03%	0.07%	8.41 MiB	11.68 GiB	-	-	7.50 kB/s	4.04 kB/s
192.168.10.132:8080	2	4	0.00%	0.10%	12.33 MiB	11.68 GiB	-	-	16.96 kB/s	3.92 kB/s
192.168.10.130:8080	2	4	0.07%	0.10%	11.53 MiB	11.68 GiB	-	-	10.54 kB/s	3.27 kB/s
192.168.10.111:8080	6	8	6.72%	14.02%	2.19 GiB	15.61 GiB	0.00 B/s	0.00 B/s	37.14 kB/s	1.49 kB/s

	min	max	current
valheim-mbround18-valheim-1:	36.57%	41.16%	38.97%
cadvisor:	10.80%	18.01%	14.34%
redis-1b972b20-960e-50ca-1b2b-b4e3eb07df0a:	0.02%	0.44%	0.21%
actualbudget-actual_server-1:	0.13%	0.29%	0.18%
portainer:	0.00%	0.17%	0.16%
fabio-e626cdca-c55d-1583-3187-825d37910bf8:	0.00%	0.47%	0.11%
fabio-38f5bf93-1106-e492-21a5-3cb6aa2970fb:	0.00%	0.17%	0.11%
fabio-17cd90ba-2882-6a18-8f60-42196785840c:	0.00%	0.28%	0.07%
gickup-runner-1:	0.00%	0.08%	0.04%
http_echo-3da7b397-f098-a6dd-42cc-8b44bebc5ca0:	0.00%	0.12%	0.02%

	min	max	current
valheim-mbround18-valheim-1:	1.84 GiB	1.91 GiB	1.91 GiB
gickup-runner-1:	167.78 MiB	168.20 MiB	167.80 MiB
cadvisor:	46.56 MiB	60.85 MiB	56.75 MiB
actualbudget-actual_server-1:	45.50 MiB	46.53 MiB	45.90 MiB
portainer:	15.34 MiB	15.85 MiB	15.44 MiB
fabio-17cd90ba-2882-6a18-8f60-42196785840c:	8.98 MiB	9.41 MiB	9.23 MiB
fabio-e626cdca-c55d-1583-3187-825d37910bf8:	8.40 MiB	8.75 MiB	8.52 MiB
fabio-38f5bf93-1106-e492-21a5-3cb6aa2970fb:	8.23 MiB	8.63 MiB	8.40 MiB
http_echo-3da7b397-f098-a6dd-42cc-8b44bebc5ca0:	3.27 MiB	3.51 MiB	3.28 MiB
redis-1b972b20-960e-50ca-1b2b-b4e3eb07df0a:	2.87 MiB	2.99 MiB	2.87 MiB

Figure 11. Snippet of cAdvisor Grafana dashboard

Collecting Logs with Promtail

As mentioned in Section 2.3.2, there are two parts to accessing logs from a Nomad cluster:

- Nomad and Consul agent logs
- Task logs

Syslog integrates into the operational infrastructures of both Consul and Nomad [35, 36]. However, in the containerization domain, Podman, the designated task driver, does not natively support syslog. Instead, it employs journald as the default log driver for managing and analyzing container logs efficiently [38]. Leveraging the capabilities of the Loki agent, Promtail facilitates the collection of journal and syslog logs [66, 67]. Incorporating these logs into Promtail enables utilizing their distinct methods of organizing and labeling data, while also streamlining the logging process into a unified system. Additionally, this integration allows for the inclusion of specific labels mentioned in Section 3.2.2, enhancing the organization of log data.

3.2.3 Exploring Logs in Grafana

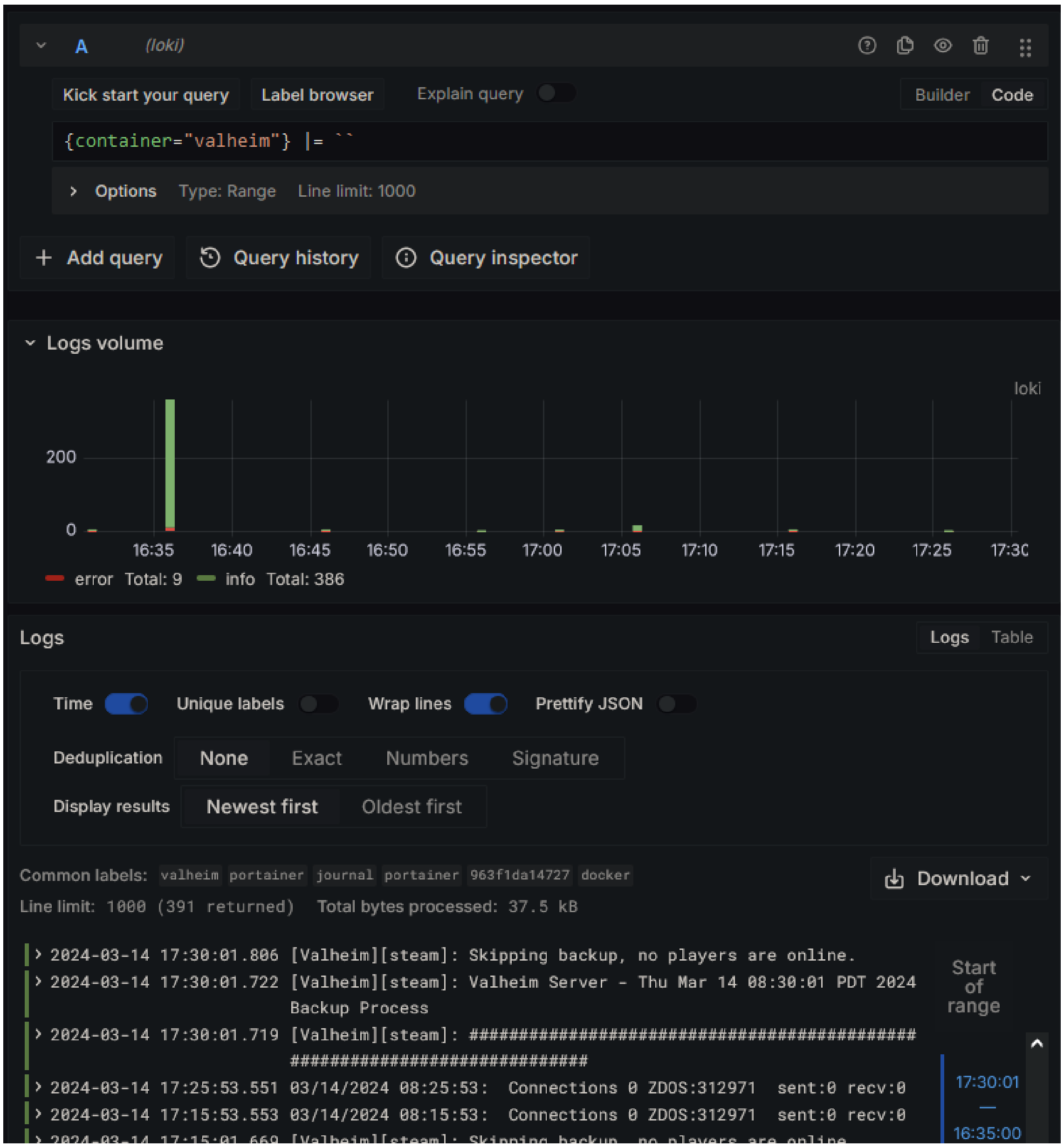
An example of a convenient log query in Grafana using a query to Loki is illustrated in Figure 12. It involves filtering by label, which has been relabeled as discussed in Section 3.2.2, to be called 'container=valheim'. This query facilitates the streamlined retrieval of logs specifically related to the 'container=valheim' label, combining all logs containers with this label into the query result. By specifying this label in the query, the author can quickly isolate relevant log entries, gaining insights into behavior of services without the need for manual filtering or sorting.

Container name relabeling significantly simplifies log analysis by allowing multiple instances to log to the same label, making it easier for the author to discern system activities. Previously, identifying failures amidst multiple container instances posed a significant challenge. For example, in a scenario with three proxy jobs where only one failed, the author had to tediously sift through each job, locate all allocation IDs, and inspect each allocation log individually to see which one was the failed one and then identify the root cause. This cumbersome process often resulted in time-consuming troubleshooting and delayed resolution. By aggregating previously separate log streams under unified labels, the visibility and clarity of system events are significantly enhanced, bettering the troubleshooting process and efficiency.

3.3 Final monitoring system

The monitoring system described in the preceding sections offers a comprehensive solution for observing and managing various aspects of the Nomad orchestration container environment. By leveraging a combination of tools such as Prometheus, Grafana, Loki, and other supporting agents, it provides real-time insights into the performance, health, and status of both containers and underlying infrastructure. This section outlines the key components and their functionalities within the monitoring ecosystem. Visual overview of the monitoring system can be seen on Figure 13.

Container metrics, including those pulled by cAdvisor and Nomad, are scraped by the Prometheus server. Additionally, native metrics endpoints from Consul and Nomad agents are also collected by Prometheus. Furthermore, the Node Exporter is utilized to publish OS, hardware, and kernel metrics, which are then scraped by Prometheus. These metrics, labelled by Prometheus, serve as vital indicators for assessing the overall health and performance of the infrastructure, Nomad system and containers managed by Nomad.



```
{container="valheim"} |= ``
```

Options Type: Range Line limit: 1000

🟥 error Total: 9 🟩 info Total: 386

Logs

Time: on · Unique labels: off · Wrap lines: on · Prettify JSON: off

Deduplication: [✓ None] Exact Numbers Signature

Display results: [✓ Newest first] Oldest first

Common labels: valheim portainer journal portainer 963f1da14727 docker

Line limit: 1000 (391 returned) Total bytes processed: 37.5 kB

```
> 2024-03-14 17:30:01.806 [Valheim][steam]: Skipping backup, no players are online.
> 2024-03-14 17:30:01.722 [Valheim][steam]: Valheim Server - Thu Mar 14 08:30:01 PDT 2024
                          Backup Process
> 2024-03-14 17:30:01.719 [Valheim][steam]: ##############################################
                          ##############################
> 2024-03-14 17:25:53.551 03/14/2024 08:25:53:  Connections 0 ZDOS:312971  sent:0 recv:0
> 2024-03-14 17:15:53.553 03/14/2024 08:15:53:  Connections 0 ZDOS:312971  sent:0 recv:0
> 2024-03-14 17:15:01.669 [Valheim][steam]: Skipping backup, no players are online
```

Figure 12. Example of container logs query from Loki

Prometheus generates alerts based on predefined thresholds and rules derived from the collected metrics. Alertmanager orchestrates these alerts, organizing them based on severity and routing them to appropriate channels for resolution. Notifications are dispatched to Microsoft Teams and Telegram, ensuring timely awareness and response to critical incidents.

Container logs from journald are gathered and relabeled by the Promtail agent before being dispatched to the Loki storage. Syslog is utilized to collect logs from Consul and Nomad agents. These logs are then processed by an rsyslog receiver, further refined by Promtail, and also stored in Loki. This centralized log storage mechanism simplifies log

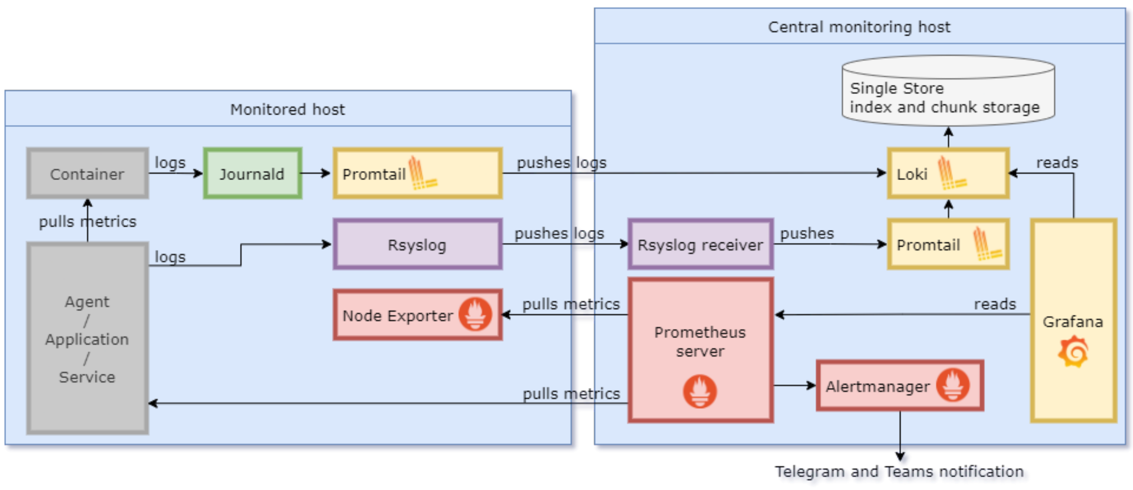


Figure 13. Overview of the implemented monitoring system

management and enhances visibility across the environment. By leveraging same labels as Prometheus, it is easy to correlate what is happening to why it is happening.

Grafana serves as the primary interface for visualization and analysis of the collected metrics and logs. Dashboards are configured to display key performance indicators and trends, providing actionable insights into the system's behavior. Leveraging Prometheus as the data source, Grafana offers a rich set of visualization options and customization capabilities. Furthermore, logs stored in Loki can be explored directly from Grafana, offering a seamless experience for troubleshooting and analysis.

4. Outcomes

This section describes the key outcomes of the implemented monitoring system.

4.1 Improved Observability

The implementation of the monitoring system for HashiCorp Nomad using Prometheus and Grafana has yielded significant outcomes. Most critically, timely notifications are now provided in the event of various critical incidents, such as job failures, host downtimes, and instances of proxy requests slowing down. Additionally, alerts are triggered for instances of CPU load either exceeding or falling below optimal levels, enabling proactive measures such as memory optimization for virtual machines.

Furthermore, the system offers comprehensive metrics and visualizations, giving the author a live overview of system activities. This insight into both historical and real-time changes enables more thorough analyses of system behaviour, facilitating informed decision-making and optimization strategies.

Moreover, the integration of aggregated logs from Loki into Grafana notably simplified the process of navigating through Nomad allocation logs. Previously, when dealing with multiple containers, the system operator had to manually search for allocation IDs, read their individual log streams, and identify problematic instances. However, with the unified labeling of all log streams, the task became streamlined. The operator is now able to effortlessly query a unified label to pinpoint error logs and discern their originating hosts, thus expediting troubleshooting procedures.

4.2 System Optimization

As mentioned in the previous Section 4.1 using alerts which notified when system resources were underutilized and leveraging metric visualization, it was possible to see in detail how much system resources the orchestration system was taking up.

On average, Nomad control plane nodes exhibited CPU usage ranging from 1.1% to 1.7% and consumed approximately 400-500MB of RAM while idle. Under light loads, CPU usage remained modest at 2.5%, with RAM usage hovering around 500MB. These observations indicated that the initially provisioned resources of 12GB of RAM and 4 CPU threads per control plane node were excessive, suggesting an opportunity for significant scaling down. Substantial resources could be freed up for either accommodating additional service nodes or bolstering existing ones, thereby optimizing resource utilization across the infrastructure.

Monitoring data from Nomad service nodes revealed that these nodes exhibited a CPU usage of around 5% and consumed approximately 600-700MB of RAM while idle with no services running on them. Under load usage would depend on the services running on them so scaling requirements for service nodes will vary a lot.

4.3 Flexible Monitoring System for Everything

The flexibility inherent in the monitoring stack deployed for Nomad has extended its utility beyond just orchestrating Nomad services. Leveraging this adaptable stack, it has become the go-to monitoring solution for all services within the homelab environment. As previously highlighted, cAdvisor plays a pivotal role in not only monitoring Nomad containers but also other container environments, amplifying its significance across the infrastructure.

Metrics derived from systemd services have proven to be particularly insightful. In the event of any service failures, the monitoring system promptly triggers notifications, ensuring swift responses to potential issues. This proactive approach has significantly contributed to maintaining the overall reliability and availability of services within the homelab environment.

Moreover, the monitoring system's versatility allows for the incorporation of additional monitoring targets seamlessly. Whether it's monitoring custom applications, network devices, or hardware components, the stack's flexibility ensures that it can adapt to diverse monitoring requirements across the homelab infrastructure. And the wide variety of shared dashboards in Grafana Labs repository provides out of the box visualization for a diverse variety of hardware and services.

4.4 Viability of Nomad as a Simple Orchestrator

The viability of Nomad as a simple orchestrator is assessed primarily through resource usage, with a distinct focus on monitoring in this thesis. Unlike traditional discussions on orchestration, which often delve into the initial setup complexities and considerations like accommodating dynamic IPs and ports for services, this section focuses on resource usage and performance.

As mentioned in Section 4.2 the adoption of Nomad does come with a resource cost. A node usage typically entails CPU utilization ranging from 1.1% to 1.7% and consumes approximately 500MB of RAM while idle. Moreover, maintaining the minimum requisite infrastructure, comprising three base nodes and two service nodes at the least, introduces a notable overhead. The cumulative resource overhead of all nodes must be substantially outweighed by the benefits, such as improved high-availability or self-healing capabilities, for the investment in Nomad to be justifiable.

It is possible to deploy server and client on same host, essentially adding the control plane resources to the service pool, in theory optimizing the system further. If nodes consistently maintain sufficient spare resources, jobs can execute without issues. However, Nomad's lack of self-management may lead to outages due to unaccounted growth in server agent memory, necessitating thorough testing and consideration of use case specifics, including agent and job updates, to ensure suitability of this approach [68]. While resource spikes of this nature were not observed with the comparatively light loads used in this thesis, it is something to consider in larger environments.

A benchmark comparison was conducted between the best-case scenario of plain rootful containerization and the worst-case scenario of Nomad orchestrated rootless containers. The optimal scenario of plain containerization employed rootful Docker, capitalizing on root privileges to optimize native system resources for prioritizing both resources and networking. Conversely, in the Nomad environment, a rootless Podman configuration was utilized, necessitating network traffic to traverse through slirp4netns, incurring a performance penalty [69]. To gauge performance, a basic HTTP server was set up in both environments, and measurements were recorded using Apache Bench [70].

HTTP and HTTPS requests were measured with 5,000 requests and 50 concurrent users.

Table 1. HTTP Speed Results

Scenario	Req/sec	TPR (ms)	TPR (ms, concurrent)	Transfer (Kb/sec)
HTTPS Plain	488.44	102.367	2.047	83.00
HTTPS Nomad	445.70	112.183	2.244	75.73
HTTP Plain	5258.37	9.509	0.190	893.51
HTTP Nomad	2124.95	23.530	0.471	361.08

TPR = Time Per Request

Req = Request

A notable disparity in speed was observed without encryption, particularly when utilizing plain HTTP. However, with SSL termination at the external load balancer, the difference becomes negligible as most of the time goes for encryption operations. This holds true even when considering that the Nomad system must additionally traverse internal load balancers, which, in turn, are tasked with verifying live nodes through Consul. As most environments will use HTTPS, then in terms of container networking, no substantial difference in speed penalty will be seen compared to traditional containers.

5. Further Improvements

This section will delve into additional refinements that could be made to the work described in previous sections.

5.1 Tracing

While the monitoring system implemented for HashiCorp Nomad has proven to be effective in providing insights into the dynamic orchestration environment, there are areas for further enhancement. One such area is tracing, which offers a deeper level of insight into application performance and behavior. However, it's essential to note that tracing is highly application-specific and often requires instrumentation on the application side. Tracing enables the detailed observation of requests as they propagate through various components of a distributed system, offering insights into latency, errors, and dependencies.

One should note that while tracing offers valuable insights, the focus of this thesis was primarily on monitoring the dynamic system as a whole, rather than monitoring individual services running on the system. Therefore, while tracing remains an avenue for potential improvement, its implementation would require a shift in focus towards monitoring specific applications and services within the Nomad environment.

5.2 High-Availability

While the current monitoring system provides valuable insights into the HashiCorp Nomad environment, of which one of the main selling points is high availability (HA), HA should be considered for the monitoring system as well. At present, the system components operate as single instances, which introduces a single point of failure. However, it's important to note that while HA configurations offer fault tolerance, they may not always be the most efficient solution: just as replication factor was a crucial consideration in comparing Nomad against plain containerization, mentioned in Section 4.4.

In some cases, such as the author's homelab environment, where resource constraints are a concern, implementing replication nodes for HA may not be practical. The additional resource overhead required for maintaining replicated instances of critical components like Prometheus, Loki, and rsyslog could exhaust available resources.

Instead, the focus may be on optimizing the existing infrastructure to maximize efficiency without compromising reliability. By fine-tuning resource allocation, optimizing configurations, and implementing robust monitoring and alerting mechanisms, the current monitoring system can continue to provide effective monitoring capabilities while remaining resource-efficient. Lack of fault tolerance can be offset by leveraging IaC deployment methods and backups of metrics and logs, providing a way to recover monitoring without HA.

So while the current system may not be HA, it is more efficient and tailored to the specific needs and constraints of the homelab environment. As the infrastructure evolves and resource availability changes, revisiting the possibility of implementing HA configurations for critical components remains a consideration. However, for now, the emphasis is on maintaining a balance between efficiency and reliability to ensure optimal performance within the existing infrastructure constraints.

6. Summary

This thesis aimed to explore orchestration within the author's home laboratory, focusing on monitoring and addressing the challenges posed by dynamic and ephemeral containers under the control of an orchestration system. To meet these challenges, a monitoring architecture distinct from traditional approaches for plain containers or classical on-host services was deployed.

The implementation and testing of the Prometheus stack with Grafana, Loki, and Alertmanager yielded significant outcomes. This architecture facilitated the provision of notifications, offered insights into both metrics visualization and log analysis. Especially noteworthy was the significant simplification of log analysis: Loki made navigating and analyzing logs from multiple instances of a dynamic service a straightforward task, enhancing troubleshooting and system management efficiency. In addition, the stack's flexibility extended its utility to encompass monitoring for all services within the homelab environment.

While HashiCorp Nomad emerges as an appetizing alternative to Kubernetes for orchestration due to its simplicity, it's essential to acknowledge that Nomad still requires significant effort to set up and manage effectively. Moreover, it may not be suitable for very small environments where the overhead of running the orchestrator's nodes exceeds the resource usage of the services running on the system.

Through this exploration, valuable knowledge was gained in orchestration and dynamic monitoring. The viability of HashiCorp Nomad as an orchestrator for the homelab was affirmed, underscoring its efficiency and suitability for managing dynamic workloads.

Version controlled code repositories utilized in the making of this thesis can be found publicly at GitHub for Packer [1], Terraform [2], and Ansible [3].

References

[1] Karl-Andreas Turvas. jamlab-packer. URL: https://github.com/JamFox/jamlab-packer.

[2] Karl-Andreas Turvas. jamlab-terraform. URL: https://github.com/JamFox/jamlab-terraform.

[3] Karl-Andreas Turvas. jamlab-ansible. URL: https://github.com/JamFox/jamlab-ansible.

[4] CNCF Survey 2019. 2021. URL: https://www.cncf.io/wp-content/uploads/2020/08/CNCF_Survey_Report.pdf. Used 2024.03.14.

[5] CNCF Survey 2020. 2021. URL: https://www.cncf.io/wp-content/uploads/2020/12/CNCF_Survey_Report_2020.pdf. Used 2024.03.14.

[6] CNCF 2021 Annual Survey. 2021. URL: https://www.cncf.io/wp-content/uploads/2022/02/CNCF-AR_FINAL-edits-15.2.21.pdf. Used 2024.03.14.

[7] CNCF 2022 Annual Survey. 2022. URL: https://www.cncf.io/reports/cncf-annual-survey-2022/. Used 2024.03.14.

[8] Progress for unprivileged containers. 2022. URL: https://lwn.net/Articles/909627/. Used 2024.03.14.

[9] Amazon elastic kubernetes service. URL: https://aws.amazon.com/eks/. Used 2024.03.14.

[10] Azure kubernetes service. URL: https://azure.microsoft.com/en-us/%20services/kubernetes-service/. Used 2024.03.14.

[11] Google kubernetes engine. URL: https://cloud.google.com/%20kubernetes-engine. Used 2024.03.14.

[12] Jordan Webb. The container orchestrator landscape. URL: https://lwn.net/Articles/905164/. Used 2024.03.14.

[13] Matthias Endler. Maybe You Don't Need Kubernetes. 2019. URL: https://endler.dev/2019/maybe-you-dont-need-kubernetes/. Used 2024.03.14.

[14] Chang Li Yishan Lin. Nomad, Kubernetes, and a Pragmatic Look at Choosing Orchestrators. 2020. URL: https://www.hashicorp.com/blog/nomad-kubernetes-a-pragmatic-look-at-choosing-orchestrators. Used 2024.03.14.

[15] Chang Li. A Kubernetes User's Guide to HashiCorp Nomad. 2021. URL: https://www.hashicorp.com/blog/a-kubernetes-user-s-guide-to-hashicorp-nomad. Used 2024.03.15.

[16] HashiCorp. Nomad vs Kubernetes. URL: https://developer.hashicorp.com/nomad/docs/nomad-vs-kubernetes. Used 2024.03.15.

[17] HashiCorp. Packer by HashiCorp. URL: https://www.packer.io/. Used 2024.03.15.

[18] HashiCorp. Terraform by HashiCorp. URL: https://www.terraform.io/. Used 2024.03.15.

[19] Inc. Red Hat. Ansible is Simple IT Automation. URL: https://www.ansible.com/. Used 2024.03.15.

[20] "IEEE Standard Glossary of Software Engineering Terminology". In: IEEE Std 610.12-1990 (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[21] Sina Niedermaier et al. "On Observability and Monitoring of Distributed Systems – An Industry Interview Study". In: Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 36–52. ISBN: 9783030337025. DOI: 10.1007/978-3-030-33702-5_3. URL: http://dx.doi.org/10.1007/978-3-030-33702-5_3.

[22] Marcello Cinque, Domenico Cotroneo, and Antonio Pecchia. "Event Logs for the Analysis of Software Failures: A Rule-Based Approach". In: IEEE Transactions on Software Engineer 39.6 (2013), pp. 806–821. DOI: 10.1109/TSE.2012.67.

[23] OpenTelemetry Authors. Observability Primer. URL: https://opentelemetry.io/docs/concepts/observability-primer/. Used 2024.04.05.

[24] Chen et al. "Towards intelligent incident management: why we need it and how we make it". In: 2020 Foundations of Software Engineering. ACM, 2020, pp. 1487–1497. URL: https://www.microsoft.com/en-us/research/publication/towards-intelligent-incident-management-why-we-need-it-and-how-we-make-it/.

[25] Alper Sarikaya et al. "What Do We Talk About When We Talk About Dashboards?" In: IEEE Transactions on Visualization and Computer Graphics 25.1 (2019), pp. 682–692. DOI: 10.1109/TVCG.2018.2864903.

[26] HashiCorp. Nomad by HashiCorp. URL: https://www.nomadproject.io/. Used 2024.03.15.

[27] HashiCorp. Consul by HashiCorp. URL: https://www.hashicorp.com/products/consul. Used 2024.03.15.

[28] HashiCorp. Nomad Overview. URL: https://developer.hashicorp.com/nomad/tutorials/get-started/gs-overview. Used 2024.03.15.

[29] HashiCorp. Consul Glossary. URL: https://developer.hashicorp.com/consul/docs/install/glossary. Used 2024.03.15.

[30] HashiCorp. Nomad Glossary. URL: https://developer.hashicorp.com/nomad/docs/glossary. Used 2024.03.15.

[31] HashiCorp. Nomad Reference Architecture. URL: https://developer.hashicorp.com/nomad/tutorials/enterprise/production-reference-architecture-vm-with-consul. Used 2024.03.15.

[32] HashiCorp. Nomad Scheduling. URL: https://developer.hashicorp.com/nomad/docs/concepts/scheduling/scheduling. Used 2024.03.15.

[33] HashiCorp. Monitoring Nomad. URL: https://developer.hashicorp.com/nomad/docs/operations/monitoring-nomad. Used 2024.03.15.

[34] HashiCorp. Monitor Consul Datacenter Health. URL: https://developer.hashicorp.com/consul/tutorials/day-2-operations/monitor-datacenter-health. Used 2024.03.15.

[35] HashiCorp. Nomad Configuration. URL: https://developer.hashicorp.com/nomad/plugins/drivers/podman. Used 2024.03.16.

[36] HashiCorp. Consul Agents Configuration File Reference. URL: https://developer.hashicorp.com/consul/docs/agent/config/config-files. Used 2024.03.16.

[37] HashiCorp. Nomad Podman Task Driver. URL: https://developer.hashicorp.com/nomad/plugins/drivers/podman. Used 2024.03.15.

[38] Podman. Podman Documentation: log-driver option. URL: https://docs.podman.io/en/latest/markdown/podman-create.1.html#log-driver-driver. Used 2024.03.16.

[39] The Linux Foundation. Prometheus - Monitoring system & time series database. URL: https://prometheus.io/. Used 2024.03.16.

[40] The Linux Foundation. Prometheus Overview. URL: https://prometheus.io/docs/introduction/overview/. Used 2024.03.16.

[41] CNCF End User Technology Radar - Observability. 2020. URL: https://radar.cncf.io/2020-09-observability. Used 2024.03.21.

[42] Nitin Sukhija et al. “Event management and monitoring framework for HPC environments using ServiceNow and Prometheus”. In: Proceedings of the 12th international conference on m
2020, pp. 149–156.

[43] The Linux Foundation. Prometheus Glossary. URL: https://prometheus.io/docs/introduction/glossary/. Used 2024.03.21.

[44] The Linux Foundation. Prometheus Metric and Label Naming. URL: https://prometheus.io/docs/practices/naming/. Used 2024.03.21.

[45] HashiCorp. Nomad Metrics Reference. URL: https://developer.hashicorp.com/nomad/docs/operations/metrics-reference. Used 2024.03.15.

[46] HashiCorp. Consul Telemetry. URL: https://developer.hashicorp.com/consul/docs/agent/telemetry. Used 2024.03.15.

[47] The Linux Foundation. Monitoring Docker container metrics using cAdvisor. URL: https://prometheus.io/docs/guides/cadvisor/. Used 2024.03.21.

[48] google. Monitoring cAdvisor with Prometheus. URL: https://github.com/google/cadvisor/blob/master/docs/storage/prometheus.md. Used 2024.03.21.

[49] The Linux Foundation. Prometheus configuration. URL: https://prometheus.io/docs/prometheus/latest/configuration/configuration/. Used 2024.03.17.

[50] The Linux Foundation. Monitoring Linux host metrics with the Node Exporter. URL: https://prometheus.io/docs/guides/node-exporter/. Used 2024.03.21.

[51] prometheus. Node exporter. URL: https://github.com/prometheus/node_exporter. Used 2024.03.21.

[52] The Linux Foundation. Prometheus Alertmanager. URL: https://prometheus.io/docs/alerting/latest/alertmanager/. Used 2024.03.17.

[53] The Linux Foundation. Prometheus Alerting Overview. URL: https://prometheus.io/docs/alerting/latest/overview/. Used 2024.03.17.

[54] The Linux Foundation. Prometheus Alertmanager Configuration. URL: https://prometheus.io/docs/alerting/latest/configuration/. Used 2024.03.17.

[55] Grafana Labs. Grafana OSS. URL: https://grafana.com/oss/grafana/. Used 2024.03.17.

[56] The Linux Foundation. Grafana support for Prometheus. URL: https://prometheus.io/docs/visualization/grafana/. Used 2024.03.17.

[57] Grafana Labs. Loki Storage. URL: https://grafana.com/docs/grafana/latest/dashboards/. Used 2024.03.17.

[58] Grafana Labs. Grafana Dashboards. URL: https://grafana.com/grafana/dashboards/. Used 2024.03.17.

[59] rfmoz. Node Exporter Full. URL: https://grafana.com/grafana/dashboards/1860-node-exporter-full/.

[60] ztd. Cluster overview. URL: https://grafana.com/grafana/dashboards/17549-cluster-overview/.

[61] mrkaran. Nomad Allocations. URL: https://grafana.com/grafana/dashboards/16925-allocations/.

[62] mrkaran. Nomad Server. URL: https://grafana.com/grafana/dashboards/16924-server/.

[63] HashiCorp Consul. Consul Server Monitoring. URL: https://grafana.com/grafana/dashboards/13396-consul-server-monitoring/.

[64] CodyGuo. Docker-cAdvisor. URL: https://grafana.com/grafana/dashboards/13946-docker-cadvisor/.

[65] Grafana Labs. Grafana Loki. URL: https://grafana.com/oss/loki/. Used 2024.03.17.

[66] Grafana Labs. Configuring Promtail for service discovery. URL: https://grafana.com/docs/loki/latest/send-data/promtail/scraping/. Used 2024.03.17.

[67] Grafana Labs. Grafana Loki Promtail agent configuration. URL: https://grafana.com/docs/loki/latest/send-data/promtail/configuration/. Used 2024.03.17.

[68] hashicorp. Issue 5053: What could go wrong if a nomad server and client ran on the same host? URL: https://github.com/hashicorp/nomad/issues/5053#issuecomment-451296159.

[69] Podman. Podman performance guide: Network performance for rootless Podman. URL: https://github.com/containers/podman/blob/main/docs/tutorials/performance.md#network-performance-for-rootless-podman. Used 2024.04.05.

[70] The Apache Software Foundation. Apache HTTP server benchmarking tool. URL: https://httpd.apache.org/docs/2.4/programs/ab.html. Used 2024.04.05.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Karl-Andreas Turvas

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Monitoring a dynamic HashiCorp Nomad orchestrated container environment", supervised by Ali Ghasempour and Lauri Anton
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

07.05.2024

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.