

University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

Monitoring Methods Analysis for Cloud Native Technology

Nvidia Corporation
Santa Clara, California, United States

Prepared by
Youjing Li
20602178
y864li@uwaterloo.ca
4A Electrical Engineering

11 May 2019

740 Karlsfeld Road
Waterloo, Ontario, Canada
N2T 2W4

December 5, 2019

Vincent Gaudet, Chair
Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Sir,

This report, entitled “Monitoring Methods Analysis for Cloud Native Technology”, was prepared as my 4A Work Report for Nvidia Corporation. This report is intended for the WKRPT 301 course and was written during my work term at Nvidia Corporation, where I worked for the Kubernetes Development team.

Nvidia is a leader in visual computing technologies and the inventor of the GPU, a high-performance processor used to accelerate the creation of visual effects. As the cloud industry booms, methods to build virtual GPUs on cloud are being investigated. As a result, the Kubernetes Development team is formed to conduct research on emerging technologies.

The Kubernetes Development team develops and maintains the infrastructure as a service(IaaS) solution on cloud. This term, the focus of my work has been on developing cluster monitoring solutions for Kubernetes. This report analyzes several types of monitoring strategies used to collect performance metrics and to derive cluster insights.

I would like to thank my supervisor Smitesh Pawar for guidance and support throughout my collaboration my internship. I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

A handwritten signature in black ink, reading "Youjing Li" in a cursive script.

Youjing Li
ID 206021

Contributions

The Kubernetes Development Team I worked for was relatively small for the company. My immediate team consisted of three full time employees and myself. The larger group this team belonged to, Cloud Platform Team, was composed of hundreds of software engineers working on development of various cloud components. I was the only person working on my specific project, other than some review process.

The team's main goal was to develop version 2.0 of infrastructure as a service(IaaS) to be used on existing datacenter products to allow market competitions with other companies offering identical products. IaaS is the fundamental building blocks for cloud services; it utilizes virtual machines(VMs) to deploy enterprise workload and offers services like cost tracking, performance monitoring, network traffic balancing, etc. The first version of IaaS was built for internal applications. While the system supported various applications across the company, it is solely built for internal use and is not entirely open source. To open the datacenter product to the general public, version 2.0 of IaaS needs to be built to tailor to the market needs. It was decided that open source is a good investment for the future and therefore my team was formed to investigate into the usability of Kubernetes--the highest-velocity open source project in history.

My tasks were to explore existing monitoring technologies and to develop a unique solution that tracks VM health and ensures maximum visibility into cluster performance. To achieve my goals, I first had to understand the fundamentals of Kubernetes architecture and learn about operations of individual components that make up Kubernetes. Kubernetes is a container management platform that helps to scale applications, automate deployments, and to establish container-centric infrastructure. Since the project is open source, there is a big online community and abundant resources which I can learn from. Kubernetes offers its own tutorials which inform people of the basic functionalities that the project offers—creating a cluster using Minikube, building a deployment using kubectl, exposing applications using a service, etc. After establishing the basic understanding of Kubernetes, I compared the different metrics and visualization solutions available and decided to develop the monitoring solution on top of Prometheus and Grafana. Throughout the term, I communicated design options and made decisions based on feedback from upper management. The solution I built includes Prometheus and Grafana for metrics collection and data visualization. Future work involves strategies for alerting and multi-cluster enabling.

The relationship between this report and my job is that the report investigates into the monitoring strategy chosen and explains both the benefits and drawbacks in details. My team is currently building on top of Kubernetes so having a monitoring tool that generates more insight into the

cluster activities can help with the development process. I was the only person developing the monitoring solution, so all the testing and deployments were carried out by me with weekly team meetings to discuss about project needs and design decisions. The project was carried out with engineering analysis and insight in which I gained an in-depth understanding of cloud-native environment. The technical skills that I possessed were made stronger by hands-on software engineering practices of deploying configurations, scaling applications, and considering engineering trade-offs. My analytical skills were also enriched through building analytical dashboards.

In the broader scheme of things, the work that I performed helped Nvidia Corporation in deciding on an optimal monitoring solution that could be used to track not only VM performance, but also system and Kubernetes cluster health. The development process of Kubernetes was improved with the addition of a monitoring tool. Reoccurring patterns were detected and tracked to evaluate engineering trade-offs; system metrics were regularly accessed to ensure maximum stability. In addition, the monitoring solution served as the foundation to all future extensions. Many layers could build on top of the existing layers. For example, networking tools like ISP, DNS, DHCP as well as data center devices like servers, network devices, and security devices could also be added to the monitoring page to create an all-in-one monitoring solution for all cluster related components.

Summary

The main purpose of the report is to analyze and compare the various monitoring technologies that could be used to track Kubernetes performance. The goal of the project is to propose a reliable setup which could be used as the foundation to not only track cluster components but also be applied to a multi-cluster environment in the future. The monitoring solution introduced in this report could be broken down into two parts—metrics collection and data visualization. Prometheus and Grafana are selected to fulfill the tasks respectively; alternatives are also compared throughout the report.

The major points covered in this report are as follows. The first section(1.0 Introduction) discusses the problem and the objective of this report. The next section further expands on the project goals and an overview of the whole report. Section 3.0 covers basic background information on technical material necessary for understanding emerging technologies in cloud. The fourth and fifth sections outline the differences among popular products used for metrics collecting and data visualizing respectively. The conclusions and recommendations come next, followed by a glossary to define technical acronyms used in this report.

The major conclusions in this report are that the chosen container orchestration system, Kubernetes, has many benefits over its competitors and that Prometheus is to be used for collecting metrics and Grafana for graphing data.

The major recommendations in this report are that alerting strategies should be introduced along with the monitoring solution, with some minor suggestions for future reference.

Table of Contents

Contributions	iii
Summary	v
List of Figures	vii
List of Tables	viii
1 Introduction.....	1
2 Description of Monitoring Solution.....	1
2.1 Scope.....	1
2.2 Outline	1
3 Container Orchestration	2
3.1 Kubernetes	3
3.2 Prometheus.....	4
3.3 Grafana.....	4
4 Metrics Collection.....	5
4.1 Push vs. Pull Model	5
4.2 Query Tool.....	7
4.3 Monitoring Targets	9
5 Data Visualization.....	10
5.1 Dashboard Import	10
5.2 Manual Customization	12
6 Conclusions.....	14
7 Recommendations.....	15
Glossary	16
References.....	17

List of Figures

Figure 1. Gaining popularity of Kubernetes	4
Figure 2. Traditional monitoring system.....	6
Figure 3. Prometheus monitoring system	6
Figure 4. Prometheus scraping example	7
Figure 5. Prometheus metrics labels	7
Figure 6. PromQL sample usages	8
Figure 7. Prometheus graph interface	8
Figure 8. Static scraping configuration	9
Figure 9. Dynamic scraping configuration	9
Figure 10. Prometheus monitoring targets.....	10
Figure 11. High level overview of cluster capacity and utilisation.....	11
Figure 12. JSON file content downloaded from Grafana Labs.....	12
Figure 13. Panel editing options	13

List of Tables

Table 1. Cloud container service comparison.....	3
Table 2. Kubernetes versus other container schedulers	3
Table 3. Metrics collecting system comparison.....	5

1 Introduction

The Kubernetes Development Team at Nvidia Corporation designs and develops the future generation of IaaS layer to be used for Nvidia's datacenter products. As the team grows and evolves, the work complexity builds and the demand for visibility into cluster performance increases. Therefore, it was decided to allocate resources to investigate into the various monitoring methods to keep track of system metrics and Kubernetes components with the hope to potentially extend the solution to network and datacenter layers in the future.

This report is concerned with the investigated methods for metrics collection and data visualization in order to establish a reliable monitoring solution for Kubernetes.

2 Description of Monitoring Solution

This report will be concerned with the Kubernetes monitoring solution that is currently being developed at Nvidia Cooperation. The solution involves integration of software tools that enables metrics collecting and data visualizing. Different open source technologies are investigated and compared to ensure maximum visibility into cluster health and performance.

2.1 Scope

The purpose of this report is to determine the best cloud monitoring solution to accelerate development process for the cloud platform at Nvidia. This will ensure a better quality of cluster performance. Results from comparisons of open-source projects will be analyzed to make a recommendation as to the best solution to implement in terms of container orchestration, metrics collection, and data visualization. Container orchestration systems that were compared are Kubernetes, Swarm, Compose, and Consul. Upon deciding on the container orchestration system, Kubernetes, comparisons were made between competing products for metrics—Sensu, TICK, and Prometheus. Lastly, a data visualization platform was chosen to provide real-time updates of cluster health and performance.

2.2 Outline

The balance of this report is divided into three sections. The first section, on various container orchestration platforms, compares the different open-source technologies used by industry giants and explains the decision behind Kubernetes adoption. The second and third sections explain the different options available to aid metrics collecting and data visualizing processes respectively.




Following these are conclusions that summarize the decisions made to build the cloud monitoring solution and a recommendation as to the best practices to improve reliability and scalability of the proposed solution.

3 Container Orchestration

Container technology is shaping the future of how software developers build, ship, and maintain applications. The shift to containers provides enterprises with many benefits such as environment flexibility, resource efficiency, operational simplicity, and immense scalability. While containers gain popularity, the problem of container management comes to the public's attention. When operating on a large scale, the integration of a container orchestration tool—deploying, scaling, and managing containerized applications—becomes essential.

While many public cloud providers offer managed container services, Nvidia is investing in employees to research and develop an on-premise cloud solution to leverage existing projects and to reduce cost on the long run. Architectures from popular public cloud providers are taken into consideration when architecting the solution to take advantage of the developer community and to leverage open source projects. Table 1 summarizes the open source technologies used by Google, Microsoft, and Amazon—three well-established cloud providers. From the table, the most popular orchestration used is Kubernetes and monitoring solutions like Prometheus and Grafana are also common across the three companies.

Table 1. Cloud container service comparison [1].

	 Google Container Engine (GKE)	 Microsoft Azure Container Service	 Amazon EC2 Container Service (ECS)
Orchestration (container scheduling, cluster management, and provisioning)	Kubernetes	Marathon and DC/OS, Docker Swarm, or Kubernetes	Based on a proprietary clustering technology
Pricing	Cluster Management: Free up to 5 nodes, more than 5 nodes Google charges \$0.15 per cluster per hour Nodes instances: You will be billed for these instances according to Compute Engine's pricing, until the nodes are deleted.	Cluster management: ACS is a free service. Nodes instances: You pay for the VM instances, associated storage and networking resources consumed.	Cluster management: Amazon EC2 Container Service is a free. Nodes instances: You pay for AWS resources (e.g. EC2 instances or EBS volumes) you create to store and run your application.
Security	Leverage Kubernetes security features Control access in the cluster with your Google accounts and role permissions	Each orchestrator available in Azure Container Service has its own security considerations	Parameter Store and task IAM roles or custom integration with Vault or KeyWhiz Encryption of secrets with custom keys with KMS
DevOps tool integrations	Google Container Builder, CircleCI, Codefresh, Codeship, Jenkins, Semaphore, Shippable, Solano CI, Spinnaker, TeamCity, Wercker, Cloud Shell, Google Container Registry	Azure Container Registry, Azure CLI, Visual Studio Team Services, Jenkins, Solano CI, Spinnaker, TeamCity	AWS CodePipeline, AWS CodeBuild, and AWS CloudFormation, Amazon EC2 Container Registry, AWS CLI, CircleCI, Codefresh, Codeship, Jenkins, Semaphore, Shippable, Solano CI, Spinnaker, TeamCity, Wercker
Monitoring	Google Cloud Monitoring, InfluxDB and Grafana, Stackdriver, Hawkular, Wavefront, OpenTSDB, Kafka, Riemann, ELK, Prometheus	ELK, OMS, Datadog, Sysdig, Dynatrace, CoScale, Prometheus	CloudWatch, Datadog, statsd, ELK, Prometheus, Graphite, Grafana
Service Discovery	Only requests from outside the cluster are passing through a load balancer. A virtual IP provides access to internal services without the need for a load balancer.	Azure uses load balancers for discovery. Azure Load Balancer provides public entry points. The Marathon Load Balancer (marathon-lb) routes inbound requests to container instances that service these requests.	ECS is using load balancers for service discovery. External as well as internal services are accessible through load balancers. The Application Load Balancer (ALB) offers path- and host-based routing as well as internal or external connections.

3.1 Kubernetes

Container orchestration system manages container lifecycles inside a cluster. Although there are many offerings on the market, Kubernetes continues to gain popularity. At KubeCon 2016, Open edX shared its comparison among all the popular container management tools and concluded that Kubernetes provides the most services as shown in Table 1[2].

Table 2. Kubernetes versus other container schedulers[2].

	 Kubernetes	 Swarm	 Compose	 Consul
Scheduling	✓	✓		
Service discovery	✓	✓	✓	✓
Container scaling	✓		✓	
Health checking	✓			✓
Secret management	✓			
Rolling updates	✓			

In recent years, Kubernetes has been deployed to a wide range of enterprises. Figure 1 shows the increasing popularity of Kubernetes based on several metrics—mentions in news articles and scholarly publications, appearances in web searches, as well as Github stars and commits.

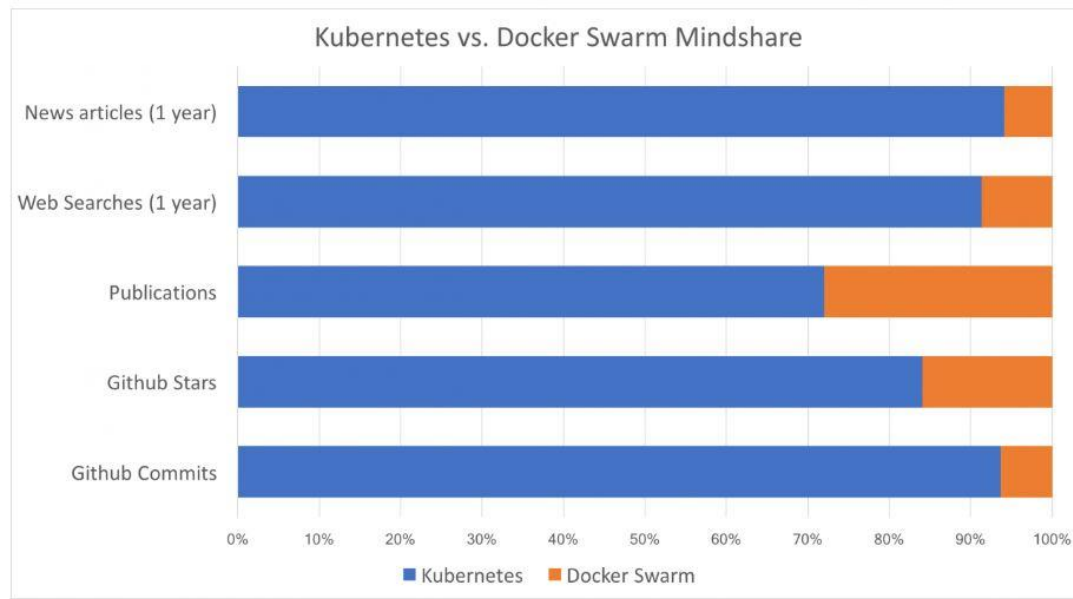


Figure 1. Gaining popularity of Kubernetes [3].

3.2 Prometheus

Prometheus is an open-source application that collects metrics from endpoints and stores the data locally on nodes. The application comes with a built-in time-series database and its own scripting language—PromQL[4]. The project is selected as the metrics collecting piece to integrate on top of Kubernetes; the choice is discussed further in Section 4 Metrics Collection.

3.3 Grafana

Grafana is a visualization tool frequently used with Prometheus to host metrics as web applications. The tool utilizes JSON for object definitions, which makes the dashboards portable and easy to use with version control[5]. We will examine the importance of Grafana in Section 5 Data Visualization.

4 Metrics Collection

As the cloud product develops, the establishment of a monitoring system is essential to improve the development process. With reliable metrics, developers could regularly assess system longevity and cluster performance throughout the development cycle. Table 2 compares some of the most popular monitoring projects in the market. From the side to side comparisons, Prometheus is chosen as to be investigated first from the cost perspective.

Table 3. Metrics collecting system comparison [6].

	Prometheus	TICK	Sensu
Pros	Completely Free Very efficient server Dedicated metrics DB Backed by Google & K8s x2 collectors (Telegraf)	A fast moving project Baked by a company A full time series DB Clustering (Enterprise) x2 collectors (Prom)	Highly scalable Existing market share (7%) Supports several DBs
Cons	No DB clustering	Clustering with license A young project	A large setup DB is External No K8s per pod metrics

4.1 Push vs. Pull Model

Traditional metrics collecting methods rely on monitoring agents to collect metrics and to push them to a centralized server(see Figure 2). Prometheus, on the other hand, have agents called exporters to expose the collected metrics to defined endpoints so that the different servers can retrieve the data with a simple pull(see Figure 3)[7].

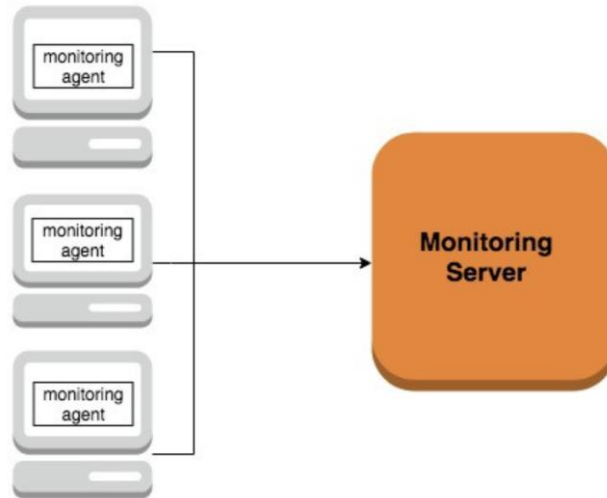


Figure 2. Traditional monitoring system [7].

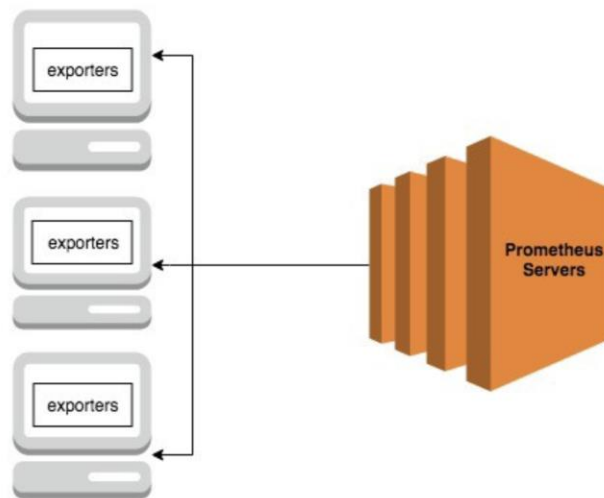


Figure 3. Prometheus monitoring system [7].

By allowing multiple servers to monitor the same set of exporters, high availability is achieved. In a Prometheus model, less work is required on the clients side—the server is now in charge of scraping metrics[7]. Figure 4 shows an example where a HTTP end point is being scraped—the server reads the “/metrics” endpoint to retrieve the updated state of metrics, and passes the information to its built-in time-series database.

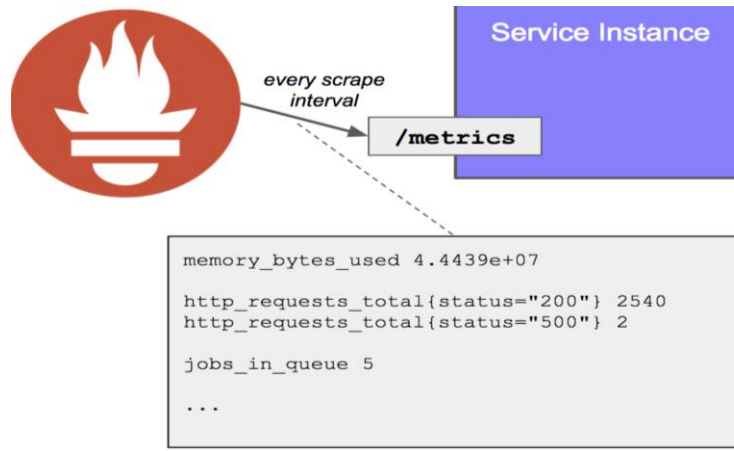


Figure 4. Prometheus scraping example[8].

From example in Figure 4, the metric “http_requests_total” is being collected and labelled based on the status code, service instance, and jobs that they relate to.

4.2 Query Tool

Once the metrics are collected and stored into the time-series database, developers are then able to view and filter the data through PromQL—querying language used by Prometheus[9]. As shown in Figure 5, the example metrics “http_requests_total” are being pulled from the database upon calling the query and are being differentiated by their unique labels—code, handler, instance, job, and method.

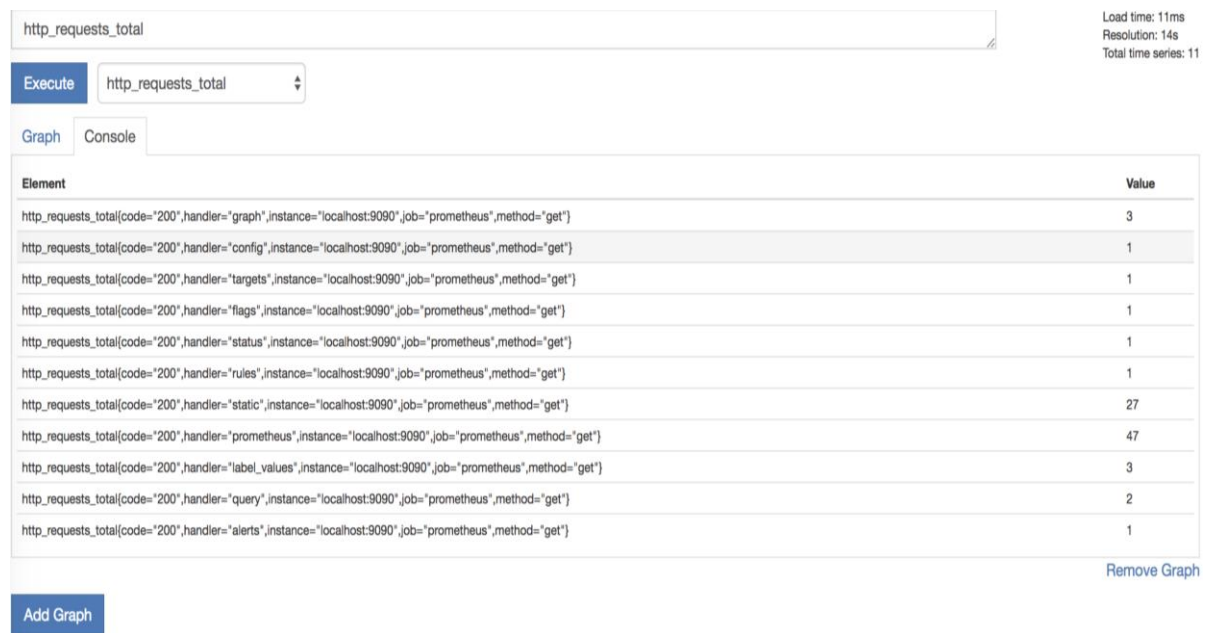


Figure 5. Prometheus metrics labels [9].

Given the data format in Figure 5, we are able to query the metrics and compute more insights by using filtering and grouping features offered by PromQL. Figure 6 shows some samples usages such as hardcoding the labels, setting the time range, matching labels to regex pattern, and using built-in functions to formulate useful statistics.

```
# Return all data with the given job and handler labels
http_requests_total{job="apiserver", handler="/api/comments"}
# Applying a time range vector to the selected data
http_requests_total{job="apiserver", handler="/api/comments"}[5m]
# Using regex pattern to select only job names that end with server
http_requests_total{job=~".*server"}
# Evaluating overall performance(sum of per-second rate) by job
sum(rate(http_requests_total[5m])) by (job)
```

Figure 6. PromQL sample usages [10].

Taking the Figure 6 query example, “sum(rate(http_requests_total[5m])) by (job),” we gain an insight of the total per second rate for HTTP requests over the time interval of 5 minutes for every job. By utilizing the graphing feature on Prometheus UI, the time series data are presented in the form shown in Figure 7.

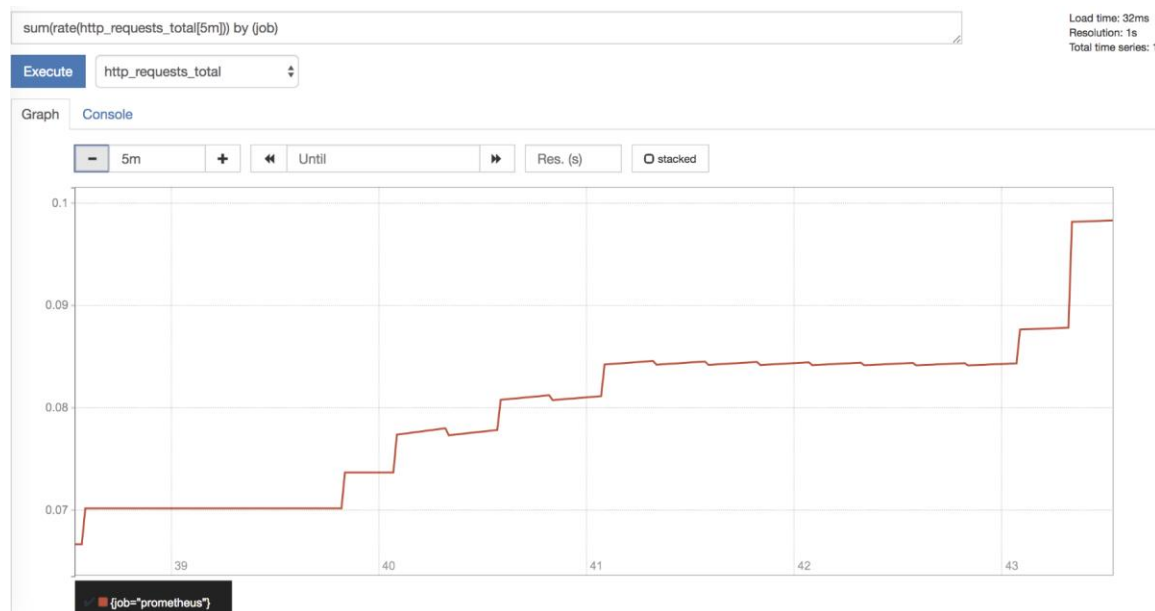


Figure 7. Prometheus graph interface [9].

4.3 Monitoring Targets

In order to have a reliable setup to assess Kubernetes cluster health, the Prometheus server can be configured to monitor the targets of interest. There are two methods to define the target points—dynamically through Kubernetes’ built-in service discovery or statically via pointers to IP addresses. Figure 5 shows an example where we could statically point at the IP address of an etcd cluster. Figure 6 demonstrates how we can take advantage of the service discovery feature and make customizations—filtering metrics using regex, updating pre-defined jobs, attaching labels to the scraped metrics.

```
- job_name: 'etcd'
target_groups:
- targets:
- 172.17.4.51:2379
```

Figure 8. Static scraping configuration [11].

```
- job_name: 'kubernetes_components'
kubernetes_sd_configs:
- api_servers:
- 'https://kubernetes'
in_cluster: true
tls_config:
ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
relabel_configs:
# Only scrape apiserver and kubelets.
- source_labels: [__meta_kubernetes_role]
action: keep
regex: (?apiserver|node)
# Redefine the Prometheus job based on the monitored Kubernetes component.
- source_labels: [__meta_kubernetes_role]
target_label: job
replacement: kubernetes_$1
# Attach all node labels to the metrics scraped from the components
- action: labelmap
regex: __meta_kubernetes_node_label_(.+)
```

Figure 9. Dynamic scraping configuration [11].

As shown in Figure 6, the Prometheus pod is configured to identify itself through the TLS configurations by passing in the CA certification and bearer token. By enabling the service discovery feature, metrics could be dynamically collected with little to no changes required on the configuration file. By applying the configuration defined in Figure 8, we define the monitoring

job to scrape from the etcd cluster; by applying the configurations demonstrated in Figure 9, we spin up two additional monitoring targets—the Kubernetes API server and its nodes. The successful application of configuration can be tracked via the “Targets” page of Prometheus UI. As shown in Figure 10, successful scrapes are being tracked with the state “Up”.

Prometheus Alerts Graph Status ▾ Help				
Targets				
etcd				
Endpoint	State	Labels	Last Scrape	Error
http://172.17.4.51:2379/metrics	UP	none	20.29264027s ago	
kubernetes_apiserver				
Endpoint	State	Labels	Last Scrape	Error
https://kubernetes:443/metrics	UP	none	17.032030345s ago	
kubernetes_node				
Endpoint	State	Labels	Last Scrape	Error
http://172.17.4.101:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.101"	18.419285752s ago	
http://172.17.4.201:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.201"	4.854667304s ago	
http://172.17.4.202:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.202"	6.940801592s ago	
http://172.17.4.203:10255/metrics	UP	node_kubernetes_io_hostname="172.17.4.203"	6.390530271s ago	

Figure 10. Prometheus monitoring targets [11].

5 Data Visualization

Grafana provides a power platform to visualize time series metrics. As a popular dashboard building tool, Grafana has the capability to connect to various data sources such as Elasticsearch, CloudWatch, and Prometheus. Once Grafana is connected, developers will have access to the thousands of pre-built dashboards shared on the Grafana community.

5.1 Dashboard Import

The community built dashboards are accessible and free to everyone from Grafana Labs[12]. The example provided in Figure 11 is a pre-configured dashboard that displays capacity and calculates utilisation.

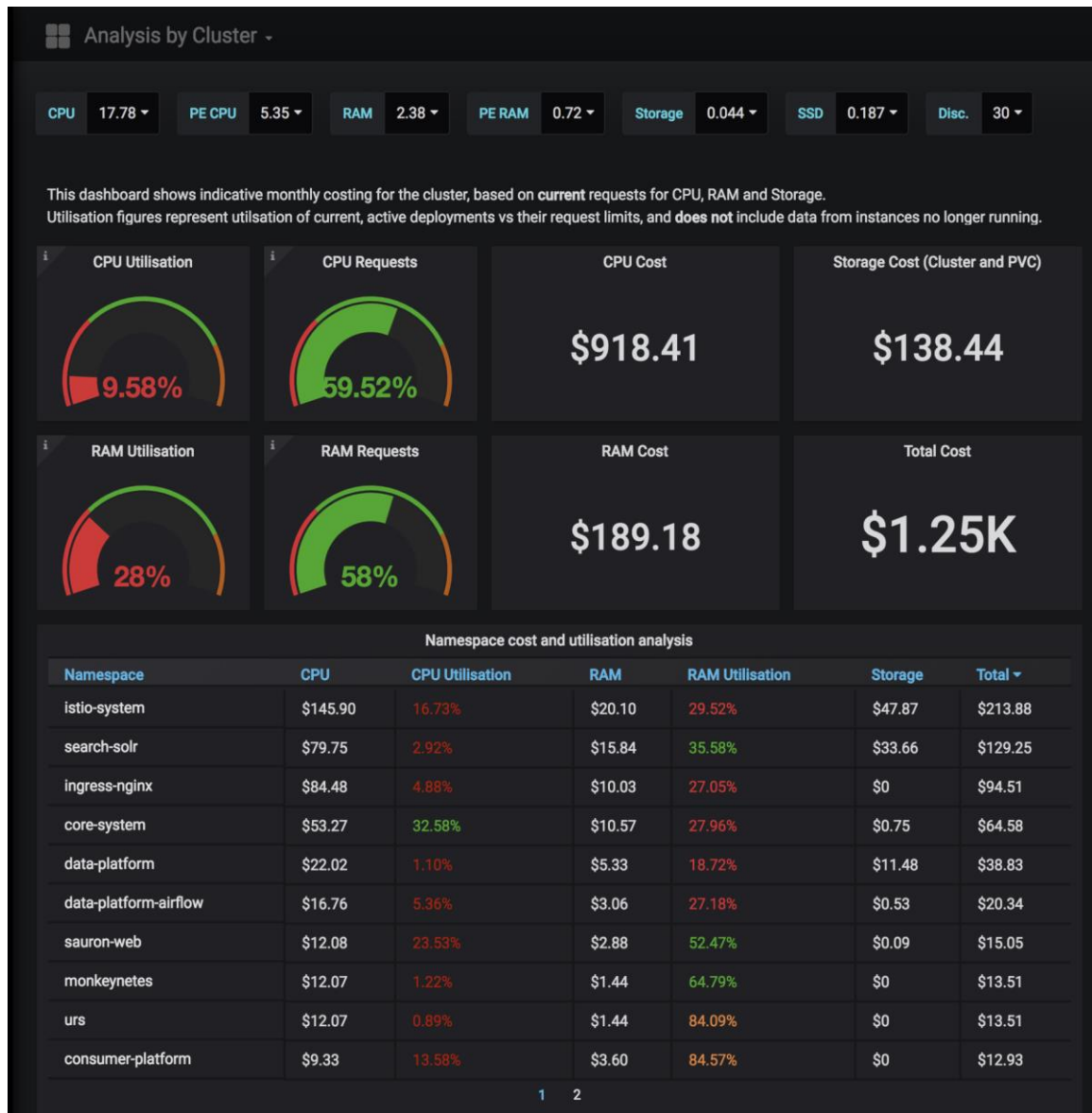


Figure 11. High level overview of cluster capacity and utilisation [12].

The utilisation in Figure 11 is calculated based on average CPU and RAM usage versus pod requests. These calculations and analysis could be leveraged by simply importing the dashboards via JSON format. For example, the automatically generated JSON version for the CPU utilisation panel in Figure 11 is shown in Figure 12. The expression used for calculation is written in PromQL discussed in section 4.2 of this report.

```

    "targets": [
      {
        "expr": "(\n  sum(\n    count(irate(container_cpu_usage_seconds_
total{id=\"/\"}[1m])) by (instance)\n    * on (instance) \n    sum(irate(c
ontainer_cpu_usage_seconds_total{id=\"/\"}[1m])) by (instance)\n  ) \n  /
\n  (sum (kube_node_status_allocatable_cpu_cores))\n) * 100",
        "format": "time_series",
        "interval": "",
        "intervalFactor": 1,
        "refId": "A",
        "step": 10
      }
    ],
    "thresholds": "30, 80",
    "timeFrom": "",
    "title": "CPU Utilisation",

```

Figure 12. JSON file content downloaded from Grafana Labs [12].

5.2 Manual Customization

Aside from importing JSON files, the Grafana user interface also provides an easy user experience to manually edit and customize dashboards. By clicking on a panel's title followed by the Edit button, users can enter the edit mode. As labelled in Figure 13, options are available to change panel, legend, series, and data source settings.

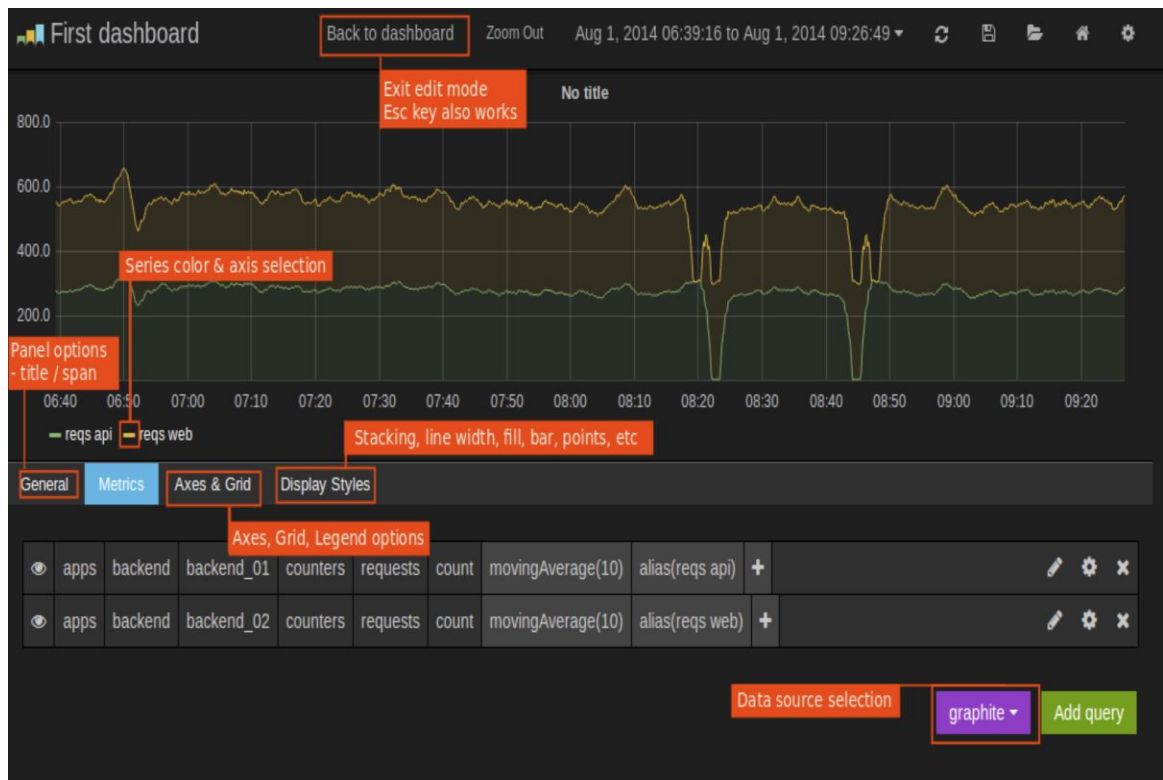


Figure 13. Panel editing options [13].

The panels can also be duplicated or moved around from the user interface. Any changes made through the user interface will be automatically translated and saved in JSON format for easy sharing.

6 Conclusions

From the analysis in the report body, it is concluded that the combination of Prometheus(Section 3.2) and Grafana(Section 3.3) offers the best solution to meet the monitoring needs for Kubernetes adoption at Nvidia. Prometheus covers the metrics collecting task(Section 4) while Grafana takes over the role of data visualization(Section 5).

In order to have a reliable metrics collecting setup, Prometheus is selected to collect cluster metrics for its low cost, efficient server, and powerful query. In Section 4, Prometheus is compared to other competitors and stands out due to the fact the project is completely free to adopt. Through further investigation, Prometheus has a built-in database, query language, and a web UI to help users to organize the metrics being collected(Section 4.2). The unique pull model implemented by the Prometheus server also takes off a great amount of workload from the clients' side to push metrics to a centralized server(Section 4.1). Lastly, all targets that we wish to monitor could be statically and dynamically discovered by defining configuration requirements(Section 4.3).

To ensure maximum visibility into the metrics collected, Grafana is integrated on top of Prometheus to allow a seamless user experience. Not only does the Grafana user interface provide capabilities to manually edit the panels(Section 5.2), the Grafana community also provides a platform for sharing and collaboration(Section 5.1).

Through the aforementioned sections, the solution established was analyzed to fulfill all monitoring requirements of the Kubernetes cluster, and to also satisfy the need for an optimized cluster environment—more speed and stability.

7 Recommendations

Based on the analysis and conclusions in this report, it is recommended that the cloud monitoring solution be implemented to establish the foundation for metrics collection and data visualization processes—Prometheus offers a robust data polling server while Grafana provides an easy-to-use user interface. Although this method satisfies all the criteria defined, future work is required to improve scalability and availability of Prometheus servers.

It is recommended to introduce alerting components like AlertManager on top of the monitoring solution. Defining alerting rules in the Prometheus server will prevent system downtime. With the help from AlertManager, we can send alerts from Prometheus through a variety of platforms like Slack, Email, and PagerDuty so that system downtime is minimized[14].

Furthermore, Thanos is an open-source project that is helpful when managing multiple clusters. Thanos not only provides a centralized view across multiple clusters, but also establishes a highly available metrics system by providing unlimited storage capacity. By implementing Thanos, we can store petabytes of historical data without sacrificing responsive query times[15].

Glossary

IaaS: Infrastructure as a service(IaaS) is a computing infrastructure that provides virtualized computing resources over the internet.

VM: A virtual machine(VM) is a software program or operating system that provides functions of a physical computer.

KubeCon: KubeCon is a conference dedicated to Kubernetes' open-source platform.

Regex: A regular expression(Regex) is a special text string for defining search patterns.

UI: A user interface(UI) is the series of screens, pages, and visual elements displayed to users.

Etc: A distributed key value store created by the CoreOS team.

TLS: Transport Layer Security(TLS) is a communication protocol that provides encryption.

CA certificate: A digital certificate issued by a certificate authority(CA).

Bearer Token: A security token created by the Authentication server.

CPU: A central processing unit(CPU) is the main processor in a computer.

RAM: Random-access memory(RAM) is the computer memory used for short-term data store.

JSON: JavaScript Object Notation(JSON) is a popular format used for exchanging data.

References

- [1] Pablo Iorio, *Container based Architectures III: Public cloud providers options—AWS, Azure, and Google*, 26-Jul-2017. [Online]. Available: <https://medium.com/@pablo.iorio/container-based-architectures-iii-iii-public-cloud-providers-options-aws-azure-and-google-708667198b5e>. [Accessed: 10-May-2019].
- [2] Marco Meinardi, *Kubernetes vs other container schedulers*, 10-Mar-2016. [Online]. Available: <https://twitter.com/meinardi/status/707935835400884224>. [Accessed: 10-May-2019].
- [3] Soumyajit Dutta, *Kubernetes vs Docker Swarm*, 01-Feb-2019. [Online]. Available: <https://medium.com/faun/kubernetes-vs-docker-swarm-whos-the-bigger-and-better-53bbe76b9d11>. [Accessed: 10-May-2019].
- [4] Prometheus Authors, *From metrics to insight: Power your metrics and alerting with a leading open-source monitoring solution*, 2014-2019. [Online]. Available: <https://prometheus.io>. [Accessed: 10-May-2019].
- [5] Grafana Labs, *The open observability platform: Grafana is the open source analytics & monitoring solution for every database*, 2019. [Online]. Available: <https://grafana.com>. [Accessed: 27-October-2019].
- [6] Amit Bezael, *The docker age: Monitoring showdown Prom vs. TICK vs. Sensu*, 28-Jan-2017. [Online]. Available: <https://medium.com/@amit.bezael/the-docker-age-monitoring-showdown-bda595b4b599>. [Accessed: 27-October-2019].
- [7] Arush Salil, *Cloud-Native Monitoring with Prometheus and Grafana*, 16-Nov-2018. [Online]. Available: <https://blog.kubernauts.io/cloud-native-monitoring-with-prometheus-and-grafana-9c8003ab9c7>. [Accessed: 27-October-2019].
- [8] Pierre Vincent, Prometheus Blog Series (Part 3): Exposing and collecting metrics, 26-Dec-2017. [Online]. Available: <https://blog.pvincent.io/2017/12/prometheus-blog-series-part-3-exposing-and-collecting-metrics/>. [Accessed: 27-October-2019].
- [9] James Turnbull, Prometheus, 10-Oct-2017. [Online]. Available: <https://www.kartar.net/2017/10/prometheus/>. [Accessed: 31-October-2019].
- [10] Prometheus Authors 2014-2019, Query Examples, 31-Oct-2019. [Online]. Available: <https://prometheus.io/docs/prometheus/latest/querying/examples/>. [Accessed: 31-October-2019].
- [11] Fabian Reinartz, Prometheus Kubernetes—Up and Running with CoreOS, 27-June-2016. [Online]. Available: <https://coreos.com/blog/prometheus-and-kubernetes-up-and-running.html>. [Accessed: 31-October-2019].
- [12] Karl Stoney, *Analysis by Cluster*, 07-July-2018. [Online]. Available: <https://grafana.com/grafana/dashboards/6873> [Accessed: 04-December-2019].
- [13] Grafana Documentation, *User Guides: Getting Started*, 01-February-2019. [Online]. Available: <https://grafana.com/docs/features/intro/>. [Accessed: 04-December-2019].
- [14] Mitesh, *Prometheus with AlertManager*, 19-November-2018. [Online]. Available: <https://itnext.io/prometheus-with-alertmanager-f2a1f7efabd6>. [Accessed: 10-November-2019].
- [15] Fabian Reinartz, *Introducing Thanos: Prometheus at scale*, 18-May 2019. [Online]. Available: <https://improbable.io/blog/thanos-prometheus-at-scale>. [Accessed: 10-November-2019].