

Trabajo Final de Máster Máster Universitario en Ingeniería de Telecomuni- cación

Metrics system

Autor: Daniel Neira Jaén

Tutor: Ramón González Carvajal

Co-tutor: Rafael Girona García

**Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla**

Sevilla, 2022



Trabajo Final de Máster
Máster Universitario en Ingeniería de Telecomunicación

Metrics system

Autor:

Daniel Neira Jaén

Tutor:

Ramón González Carvajal

Profesor Catedrático de Universidad

Co-tutor:

Rafael Girona García

Personal Investigador en Formación

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Final de Máster: Metrics system

Autor: Daniel Neira Jaén
Tutor: Ramón González Carvajal
Co-tutor: Rafael Girona García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Acknowledgements

First of all, I would like to thank Ramón and Rafael Girona (cienti) for everything in general, both in the project and outside, for helping and advising me and for calming me down in my moments of anxiety when I was looking at the dates.

To my family, my sister, my parents and my grandparents, for putting up with me in times of pressure and giving me support when I needed it most, really, thank you.

To my classmates, Pablo, Samuel and Gonzalo, without them nothing would have been the same.

To my flatmates Pablo and Alejandro for making me feel at home.

To my teachers and professors, without them it would not have been possible.

In short, to all of you who in one way or another have made it possible for me to be here today, thank you from the bottom of my heart.

Daniel Neira Jaén

Sevilla, 2022

Resumen

Lo que no se define no se puede medir. Lo que no se mide, no se puede mejorar. Lo que no se mejora, se degrada siempre.

WILLIAM THOMSON KELVIN (LORD KELVIN) (1824 – 1907)

La medición y el monitoreo de sistemas resultan prácticamente imprescindibles en un desarrollo maduro y es ahí donde nos centraremos en este trabajo, en desarrollar herramientas que por un lado monitoricen el propio stack y su servidor, así como sirva de base para almacenar y representar métricas proporcionadas por terceros agentes.

En este sentido y como todos los desarrollos, es necesario comenzar por el principio, realizando un análisis del estado del arte y de las herramientas disponibles en este momento. El proyecto está formado por 3 grandes partes diferenciadas:

- Como el objetivo final es desplegarlo en la nube, tendremos que examinar las opciones de deployments, desde aplicaciones tradicionales, hasta técnicas de virtualización pasando por máquinas virtuales hasta llegar a los actuales contenedores y sus orquestadores, analizando ventajas e inconvenientes de cada uno de ellos.
- Base de datos para almacenar métricas. En este caso la característica fundamental es la marca de tiempo, no basta solo con almacenar un valor, es casi más importante el momento temporal exacto en el que se produce para poder realizar los análisis posteriores, representaciones, etc. . . .
- El tercer elemento importante del proyecto es el visualizador, que nos permita analizar la información de una manera gráfica mucho más accesible. Así como realizar operaciones con los datos y representar también los resultados.

Una vez expuestas las herramientas actuales, analizadas y decididas las que vamos a usar, describiremos el desarrollo realizado, explicando paso a paso los problemas encontrados y las soluciones aplicadas, hasta llegar al apartado de resultados donde explicaremos el estado actual del proyecto y el punto alcanzado.

Además del sistema de métricas, en este trabajo, hemos usado e integrado 2 herramientas de generación de documentos: por un lado mkdocs, como generador de sitios estáticos, que junto con la función Pages de GitLab, nos ofrece un potencial muy interesante para representar la documentación. Por otro lado, pandoc, como conversor universal de distintos formatos, en nuestro caso, para pasar del markdown original al pdf de este documento, a través de la plantilla en latex. De esta manera, en un único repositorio, con los mismos ficheros fuente en markdown, podemos generar 2 salidas diferentes: la página web estática y el documento pdf.

Por último, expondremos las conclusiones obtenidas de este trabajo y analizaremos las líneas futuras que surgen de este proyecto.

Abstract

What is not defined cannot be measured. What is not measured, cannot be improved. What is not improved, is always degraded.

WILLIAM THOMSON KELVIN (LORD KELVIN) (1824 – 1907)

The measurement and monitoring of systems are practically essential in a mature development and that is where we will focus in this work, in developing tools that on the one hand, monitor the stack itself and its server, as well as serve as a basis for storing and representing metrics provided by third party agents.

In this sense, and like all developments, it is necessary to start from the beginning, performing an analysis of the state of the art and the tools available at this time. The project is made up of 3 main differentiated parts:

- As the final objective is to deploy it in the cloud, we will have to examine the options of deployments, from traditional applications, to virtualization techniques through virtual machines to the current containers and their orchestrators, analyzing advantages and disadvantages of each of them.
- Database to store metrics. In this case the fundamental characteristic is the time stamp, it is not enough just to store a value, it is almost more important the exact time in which it occurs to be able to perform subsequent analysis, representations, etc.
- The third important element of the project is the visualizer, which allows us to analyze the information in a much more accessible graphical way. As well as to perform operations with the data and also to represent the results.

Once the current tools have been exposed, analyzed and decided which ones we are going to use, we will describe the development carried out, explaining step by step the problems encountered and the solutions applied, until we reach the results section where we will explain the current status of the project and the point reached.

In addition to the metrics system, in this work, we have used and integrated 2 document generation tools: on the one hand mkdocs, as a static site generator, which together with the Pages function of GitLab, offers us a very interesting potential to represent the documentation. On the other hand, pandoc, as a universal converter of different formats, in our case, to move from the original markdown to the pdf of this document, through the latex template. In this way, in a single repository, with the same markdown source files, we can generate 2 different outputs: the static web page and the pdf document.

Finally, we will expose the conclusions obtained from this work and we will analyze the future lines that arise from this project.

Short Contents

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Short Contents</i>	VII
1 Introduction	1
1.1 Goals	1
1.2 State of the art	1
1.3 Scope	6
2 Environment	11
2.1 Git	11
2.2 GitLab	11
2.3 Visual Studio Code	12
2.4 Virtualization	12
2.5 OCI Container (Docker)	13
2.6 Kubernetes cluster	14
2.7 Prometheus	15
2.8 Grafana	16
3 Development	17
3.1 Prometheus (standalone)	17
3.2 Grafana (standalone)	18
3.3 Docker-compose	20
4 Deployment	23
4.1 Local vs remote server	23
4.2 Podman	25
4.3 Kubernetes	26
5 Metrics exporters	41
5.1 Lm-sensor exporter	41
5.2 Node-exporter	41
5.3 Pushgateway	43
6 Documentation	45
6.1 Static site generator: MkDocs	45
6.2 Document converter: Pandoc	46
7 Results	47
7.1 IaC. Deployment	47

7.2	Monitoring stack	50
8	Conclusions	53
9	Future line	55
Appendix A	DNS	57
	<i>List of Figures</i>	61
	<i>List of Tables</i>	63
	<i>List of Codes</i>	65
	<i>Bibliography</i>	67
	<i>Glossary</i>	69

Contents

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Short Contents</i>	VII
1 Introduction	1
1.1 Goals	1
1.2 State of the art	1
1.2.1 Operative system	1
1.2.2 Version control system	2
Advantages of using Git	2
Advantages of using SVN	2
1.2.3 Infrastructure: Container Orchestration	3
1.2.4 Kubernetes	4
1.2.5 Data base	5
1.2.6 Data visualization	6
1.3 Scope	6
1.3.1 Monitoring	7
1.3.2 Logging	7
1.3.3 Tracing	8
2 Environment	11
2.1 Git	11
2.2 GitLab	11
2.2.1 GitLab Pipelines	11
2.3 Visual Studio Code	12
2.4 Virtualization	12
2.5 OCI Container (Docker)	13
2.6 Kubernetes cluster	14
2.6.1 K3D	15
2.7 Prometheus	15
2.8 Grafana	16
3 Development	17
3.1 Prometheus (standalone)	17
3.2 Grafana (standalone)	18
3.3 Docker-compose	20
4 Deployment	23
4.1 Local vs remote server	23
4.1.1 Initial Deployment with Git	24
4.1.2 Deployment with Git and GitLab CI	25

4.2	Podman	25
4.2.1	Troubleshooting podman deployment	26
4.3	Kubernetes	26
4.3.1	Kind	27
4.3.2	K3d	27
4.3.3	Troubleshooting launching pod manifest	28
4.3.4	Current Usage	29
4.3.5	Add PV and PVC	30
	NFS example	30
	Using external NFS in the cluster	31
	Troubleshooting NFS	32
	Current usage	32
4.3.6	Added new pv and pvc for grafana	32
4.3.7	Kustomize	32
	Switch service and deployment in yaml file	34
4.3.8	K3D version update	34
4.3.9	Deployment kustomize	36
4.3.10	Ingress	38
5	Metrics exporters	41
5.1	Lm-sensor exporter	41
5.2	Node-exporter	41
5.3	Pushgateway	43
6	Documentation	45
6.1	Static site generator: MkDocs	45
6.2	Document converter: Pandoc	46
7	Results	47
7.1	IaC. Deployment	47
7.2	Monitoring stack	50
8	Conclusions	53
9	Future line	55
	Appendix A DNS	57
	<i>List of Figures</i>	61
	<i>List of Tables</i>	63
	<i>List of Codes</i>	65
	<i>Bibliography</i>	67
	<i>Glossary</i>	69

1 Introduction

If you want to go quickly, go alone. If you want to go far, go together.

AFRICAN PROVERB

Nowadays, where a new digital world gives more and more importance to information and data collection, new autonomous composite systems and new trends are emerging, such as Internet Of Things (IoT), which demand a higher quality of such data together with a greater capacity to manage the immense amount of data. In this work, we will consider the design and development of a metrics stack that allows us to manage, store and visualise this data.

1.1 Goals

Project goal is the design of a complete metrics system. To achieve that, we have defined the next specific objectives:

- Define system architecture: infrastructure, data base, data representation. . .
- Use Infrastructure as Code: managing and provisioning infrastructure through code instead of through manual processes.
- Follow GitOps principles: DevOps with Git as a source of truth.

1.2 State of the art

This will be followed by a state-of-the-art study of each specific objective, on which the final choice of each of the parts will be based.

1.2.1 Operative system

Although in some cases Windows can give satisfactory results, in general it is not a system oriented towards developers or open-source tools, and as the project will have some work to do on non-Windows servers, its feasibility is not being studied further. Mac, on the other hand, does natively support most or all of the points addressed in the project, so it would be a more than viable option. Finally, Linux is the system that is generally ready to support development, automation, compilation and cross-machine interaction natively. It has extensive software and hardware support, as well as a large and established community of users who provide easy access to information and troubleshooting.

The final choice will be Linux, and among the different distributions available, Ubuntu is chosen because it is one of the most established, stable and secure, as well as being oriented to servers, something that is sought for automation - Continuous Integration/Continuous Deployment (CI/CD).

1.2.2 Version control system

As the project became more complex, it became necessary to use version control. In today's market there are two major competitors in this area: SVN and Git. SVN and Git have a significant difference. SVN is a Centralised Version Control System (CVCS), whereas Git is a Distributed Version Control System (DVCS).

A centralised version control system operates on the idea that there is a single copy of the project where developers commit changes, and a single place where all versions of a project are stored. A distributed version control system, on the other hand, works on the principle that each developer clones the project repository to the hard drive of their device. A copy of the project is stored on each local computer, and changes must be pushed and pulled to and from the online repository in order to update the version that each developer has on their local computer.

Advantages of using Git

- **Distributed System:** Being a distributed system means that multiple, redundant repositories and branching are concepts of the tool. In a distributed system system like Git, each user has a complete copy of the project stored, making to everyone's history extremely fast. Because of this, it allows the use without an internet connection. It also means that each user has a complete copy of the repository, meaning that if for some reason one of them gets corrupted, only the changes that are not uploaded to the server will be lost. In SVN, only the central repository contains the complete history. So, users must communicate over the network to the central repository to get the history of the files. If the central repository is lost due to a system failure, it has to be restored from a backup and this can cause from a backup and this may cause some changes to be lost. Depending on the backup policies in place, weeks of work could be lost.
- **Access Control:** As a distributed system, you do not have to grant access to other people to use the access control functions. Instead, it is the repository owner who decides which changes to merge. In Git, users can have version control of their own project, while the main project is controlled by the repository owner. SVN on the other hand does, for example, the user requires access to perform commits.
- **Branching:** In Git using branches is very common and easy, whereas in SVN it can be considered a bit more cumbersome and not as common a process. In fact, the most common complaint about SVN is how tedious it is to work with branches. In SVN, branches are created as directories, which many developers don't like. Also in SVN 1.6, a concept called tree conflict, caused by changes in the directory structure, was introduced. It happens frequently and because of this, committing between branches becomes more complex.
- **Performance:** Since Git works with a local repository, there is no latency in most operations, except for push and fetch, where it does need to connect to the central repository.

Advantages of using SVN

- **Permanent history:** SVN will always fetch exactly the same information from the repository, as it was in the past. You can also track all changes to a file or folder, because the history in SVN is permanent. In Git on the other hand, you may lose the history of a file/directory. Git does not care about the precise trace or history of each file in the repositories. For example, renaming a file or the "git rebase" command makes it difficult to find the "true" history of files.
- **Single repository:** Since SVN only allows you to have one repository, you don't have to worry about where something is stored. In case we need a backup or want to search for something, we will have no doubt that everything we need is in the central repository. Because Git works with distributed repositories, it may be more difficult to know where some files are located.
- **Access control:** Because SVN has a central repository, read and write control can be specified there and will be enforced throughout the project.
- **Binary files:** Version control systems have the idea that most files that will be versioned are mergeable. That is, it should be possible to merge two simultaneous changes made to a file. This model is called Copy-Modify-Merge and both Git and SVN use it. The only problem is that this is generally not applicable to binary files, which is why SVN offers support for the Lock-Modify-Unlock model for these cases. On the other hand, Git does not support unique file locks, which complicates work in

companies with projects where there are many binary files that cannot be merged. If you want to use Git with binary files, you need to specify which of them are binary files.

After all of the above, we choose Git because we are not going to use too many binary files, which is its main handicap, and because once we have overcome its initial complexity, it is a more powerful and flexible tool.

Once we have chosen Git, we are going to use tools based on it such as GitLab and Visual Studio Code. This set of software will make things easier for us, on the one hand, the GitLab website to have access to the repo and on the other hand, Visual Studio Code, with its graphical interface as a code editor with the extra possibility to see the branches and upload new commits.

There are other ways of naming the versions of the project, but within Git there is GitLab Flow and SemVer, which will be the ones we will use.

1.2.3 Infrastructure: Container Orchestration

Main container orchestration are:

- **Amazon Elastic Container Service (ECS)** (aws.amazon.com/ecs). It's a container orchestration platform provided by AWS that manages Docker containers.
- **RedHat OpenShift** (redhat.com/en/technologies/cloud-computing/openshift). It is an open-source container application platform that operates as a PaaS (platform as a service). It can only be installed on Red Hat's proprietary: Red Hat Enterprise Linux Atomic Host (RHEL AH), Fedora, or CentOS.
- **Docker swarm** (docs.docker.com/engine/swarm). It is a container orchestration that consists of managers and workers. It's native clustering for Docker.
- **Nomad** (nomadproject.io). Nomad is a simple and flexible scheduler and workload orchestrator to deploy and manage containers and non-containerized applications across on-prem and clouds at scale. Native integrations with Terraform, Consul, and Vault.
- **Kubernetes** (kubernetes.io). Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem.

Parameter	ECS	RedHat OpenShift	Docker Swarm	Nomad	Fargate
Open Source	No	Yes	Yes	Yes	No
Vendor Lock-In	Yes, AWS	No	No	No	Yes
Cluster Setup	Easy	A little complex	Easy	Very easy	Available as launch type for ECS cluster
Built-In Workload Autoscaling	Supported	Supported	Not supported	Not supported	Supported
Manage Legacy Applications	No	No	No	Yes	No
Built-In Monitoring	Yes, Cloudwatch	Yes, Prometheus	No	No	ECS Metrics are available

Figure 1.1 Container Orchestration Comparison.

Source: <https://hackernoon.com/the-5-best-kubernetes-alternatives-ks3f344k>

We have chosen Kubernetes, in the next section, its advantages will be described. Just a data: In Cloud Native Computing Foundation (CNCF) Annual Survey 2021 (<https://cncf.io/reports/cncf-annual-survey-2021/>), 96 % of organizations are either using or evaluating kubernetes:

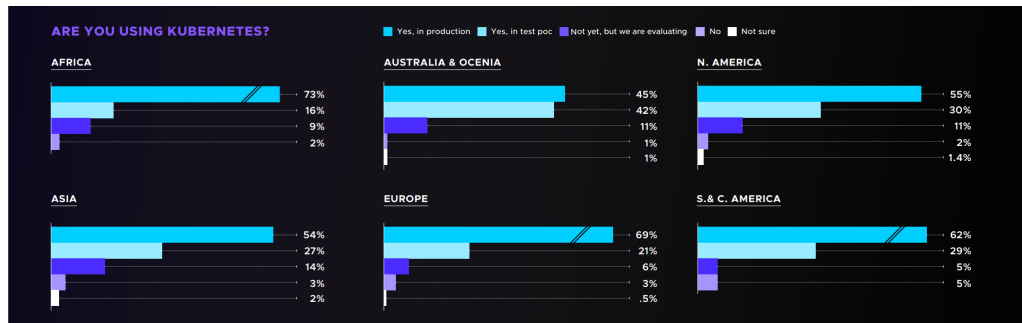


Figure 1.2 Kubernetes usage, CNCF Annual Survey.

Source: <https://cncf.io/reports/cncf-annual-survey-2021/>

1.2.4 Kubernetes

As an open source project, Kubernetes makes its source code publicly and freely available on GitHub. Anyone can download, compile and install Kubernetes on the infrastructure of their choice using this source code. But most people who want to install Kubernetes would never download and compile the source code, for several reasons:

- **Time and effort:** There is a lot of Kubernetes source code, and building it all from scratch would require a fair amount of time and effort. Plus, you'd have to rebuild it all every time you wanted to update your installation.
- **Multiple components:** Kubernetes is not a single application; it's a collection of different applications and tools. If you install it from source, you would have to install each of these parts separately on all the servers you are using to build your Kubernetes cluster.
- **Complex configuration:** Because Kubernetes doesn't have an installation wizard or auto-configuration script, you would also have to manually configure all of the Kubernetes components.

Most people turn to a Kubernetes distribution for their container orchestration needs. A Kubernetes distribution is a software package that provides a pre-built version of Kubernetes. Most Kubernetes distributions also offer installation tools to simplify the setup process. Some come with additional software integrations, too, to help handle tasks such as monitoring and security.

In this sense, we can think of a Kubernetes distribution as similar to a Linux distribution. When most people want to install Linux on a PC or server, they use a distribution that provides a precompiled Linux kernel integrated with other software packages. Almost nobody goes and downloads the Linux source code from scratch.

Main Kubernetes distribution are:

- **Kubernetes cloud management option**
3 major cloud providers: Amazon has Elastic Kubernetes Service, Azure has Azure Kubernetes Service, and Google has Google Kubernetes Engine.
- **Kubernetes enterprise option**
Redhat's OpenShift, Docker Swarm, and Cloud Foundry's KubeCF.
- **Kubernetes open source management option**
RKE, Kops and Kubeadm.
- **DIY option**
The hard way. It doesn't depend on anyone for anything.
- **Outlying "little" options**
K3s, Minikube, kind, ...

Deepening in this little options (Sept-2021):

Table 1.1 Kubernetes distribution comparison.

	Minikube	MicroK8s	kind	k3d (k3s)
Load images	Medium (internet)	-	Easy	Easy
Multi clusters	No (only 1)	No	Yes	Yes
Cluster config file	No	No	Yes	Yes
Running (linux)	1 VM	snap packages	Various containers	Various containers
Speed (linux)	Slow	Medium	Fast	Fastest
Resources (linux)	High	Medium	Low	Lowest
min CPU	2 CPUs	-	1 CPU	1 CPU
min disk space	20 GB	20 GB	1 GB	-
min RAM	2 GB	4 GB	512 MB	512 MB Min (1GB recom)

MicroK8s, microk8s.io/docs, needs: an Ubuntu 20.04 LTS, 18.04 LTS or 16.04 LTS environment to run the commands (or another operating system which supports snapd), at least 20G of disk space and 4G of memory are recommended.

One of the biggest benefits of using Microk8s is that it also supports various add-ons and extensions. They are shipped out of the box, the user just has to enable them.

Minikube, minikube.sigs.k8s.io/docs/start/, needs: 2 CPUs or more, 2GB of free memory, 20GB of free disk space.

Minikube and kind are both kubernetes community projects. Kind is newer and with almost the same goals than minikube, but lighter.

Between kind and k3d, the differences are in speed, resources and features. But, for most people, k3d is fine.

1.2.5 Data base

For a monitoring system, data base's main feature is timestamp. The data itself is important, but at a given point in time also. Data bases with timestamp features are:

- **Data Dog** (datadoghq.com). Data Dog is a tool to monitor infrastructure such as the servers, containers, processes, storage health as well as to load and look into metrics.
- **InfluxDB** (influxdata.com). It's a database purpose-built to handle the epic volumes and countless sources of time-stamped data produced by sensors, applications and infrastructure.
- **Graphite** (graphiteapp.org). Graphite does two things: Store numeric time-series data and render graphs of this data on demand. It's open source.
- **Netdata** (netdata.cloud/). Netdata collects all possible metrics from systems and applications, and shows these metrics in real-time
- **Elasticsearch** from ELK Stack (elastic.co). Elasticsearch is a tool to store, search, and analyze at scale.
- **Prometheus** (prometheus.io). Prometheus is an open-source systems monitoring and alerting toolkit.
- **Cortex** (cortexmetrics.io). Horizontally scalable, highly available, multi-tenant, long term storage for Prometheus.

Cortex is a CNCF incubation project used in several production systems including Weave Cloud and Grafana Cloud.

Cortex is primarily used as a remote write destination for Prometheus, with a Prometheus-compatible query API.

- **Thanos** (thanos.io). Open source, highly available Prometheus setup with long term storage capabilities.

Thanos leverages the Prometheus 2.0 storage format to cost-efficiently store historical metric data in any object storage while retaining fast query latencies. Additionally, it provides a global query view across all Prometheus installations and can merge data from Prometheus HA pairs on the fly.

1.2.6 Data visualization

- **Geckoboard** (geckoboard.com/). It's a cloud-based dashboard tool that displays key business metrics in real time.
- **Freeboard** (freeboard.io). It's an open source real-time dashboard builder for IOT and other web mashups.
- **Kibana** from ELK Stack (elastic.co/kibana). Kibana is a free and open user interface that lets you visualize your Elasticsearch data and navigate the Elastic Stack.
- **Grafana** (grafana.com/). Grafana is a multi-platform open source analytics and interactive visualization web application.

The main difference between Prometheus and ELK stack is the scope of use of these two systems. Prometheus is used for metric collection, various systems monitoring and setting up alerts based on these metrics. ELK is used to take all types of data, perform different types of analytics based on these data, search, and visualize it.

Prometheus uses TimeSeries DBMS as its primary database model. Actually ELK stack's primary database model is a search engine that supports storing different unstructured data types with an inverted index that allows very fast full-text searches.

1.3 Scope

The scope of the project is to develop a monitoring stack. To differentiate between monitoring, logging and tracing we use CNCF definitions.

The goal of the CNCF (Cloud Native Computing Foundation) landscape is to compile and organize all cloud native open source projects and proprietary products into categories, providing an overview of the current ecosystem. Organizations that have a cloud native project or product can submit a PR to request it to be added to the landscape.

According to the Cloud Native Survey (https://cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_-2020.pdf), 95% of respondents use monitoring, 95% use logging, and 74% use tracing.

It is much more common for those who are using these tools to run them on-premise within their infrastructure, rather than hosted via a remote service

For your monitoring, logging and tracing solutions, do you require the system to:

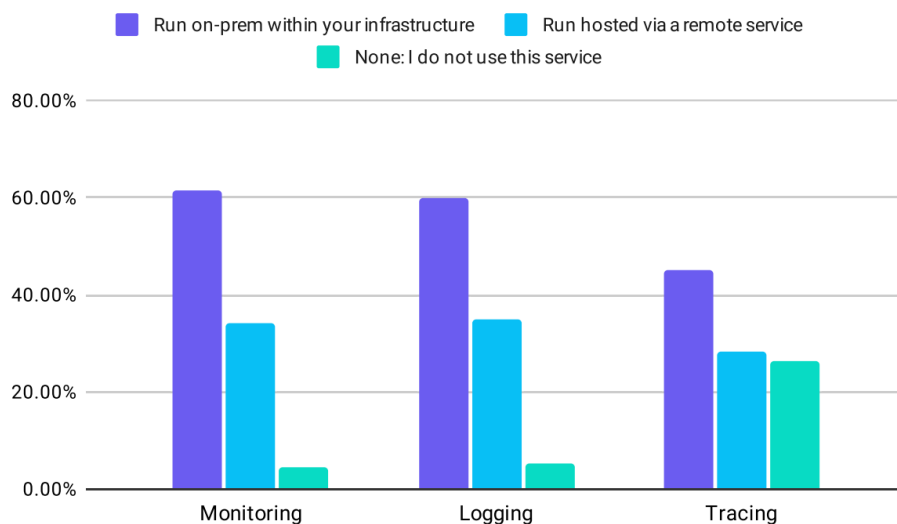


Figure 1.3 CNCF Survey.

Source: https://cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf

1.3.1 Monitoring

Monitoring refers to the instrumentation of an application to collect, aggregate and analyse logs and metrics to improve our understanding of its behaviour. While logs describe specific events, metrics are a measure of a system at a given point in time; they are two different things but both necessary to get a complete picture of the health of your system. Monitoring includes everything from looking at disk space, CPU usage and memory consumption on individual nodes, to performing detailed synthetic transactions to see if a system or application is responding correctly and in a timely manner. There are several different approaches to monitoring systems and applications.

When you run an application or platform, you want it to perform a specific task as designed and ensure that only authorised users access it. Monitoring lets you know if it is running correctly, securely, cost-effectively, if only authorised users are accessing it, as well as any other characteristics you can track.

Good monitoring allows operators to respond quickly, and even automatically, when an incident arises. It provides information on the current state of a system and keeps track of changes. Monitoring tracks everything from application state to user behaviour and is an essential part of keeping applications running effectively.

Monitoring in a cloud-native context is generally similar to monitoring traditional applications. Metrics, logs and events need to be tracked to understand the state of applications. The main difference is that some of the managed objects are ephemeral, meaning they may not be durable, so tying your monitoring to objects such as automatically generated resource names will not be a good long-term strategy. There are a number of CNCF projects in this space that largely revolve around Prometheus, the CNCF graduated project.

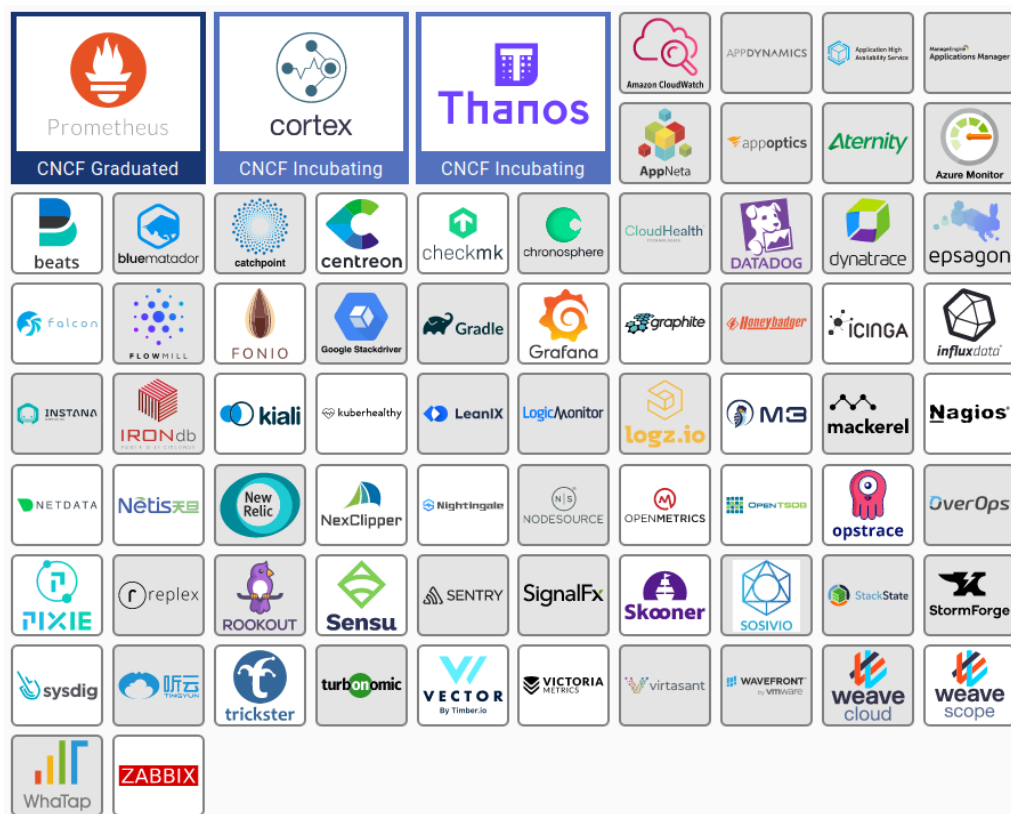


Figure 1.4 CNCF Monitoring.

Source: <https://landscape.cncf.io/guide#observability-and-analysis-monitoring>

1.3.2 Logging

Applications emit a constant stream of log messages describing what they are doing at any given time. These log messages capture various events that occur in the system, such as failed or successful actions, audit

information or health events. Logging tools collect, store and analyse these messages to track error reports and related data. Along with metrics and tracking, logging is one of the pillars of observability.

Collecting, storing and analysing logs is a crucial part of building a modern platform, and logging performs any or all of these tasks. Some tools handle all aspects, from collection to analysis, while others focus on a single task, such as collection. All logging tools aim to help organisations gain control of their logging messages.

By collecting, storing and analysing application log messages, you can understand what an application was communicating at any given time. But because logs only represent the messages that applications or platforms deliberately emit, they do not necessarily point to the root cause of a given problem. That said, collecting and retaining log messages over time is an extremely powerful capability and will help teams diagnose problems and meet regulatory and compliance requirements.

Collecting, storing and processing log messages is by no means a new problem, but cloud-native patterns and Kubernetes have significantly changed the way logs are managed. Some traditional approaches to logging that were appropriate for virtual and physical machines, such as writing logs to a file on a local disk, are not suitable for containerised applications, where file systems don't last longer than an application. In a cloud-native environment, log collection tools such as Fluentd run alongside application containers and collect messages directly from applications. The messages are then sent to a central log store for aggregation and analysis.



Figure 1.5 CNCF Logging.

Source: <https://landscape.cncf.io/guide#observability-and-analysis-logging>

1.3.3 Tracing

In a world of microservices, services are constantly communicating with each other across the network. Tracing, a specialised use of logging, allows tracing the path of a request as it moves through a distributed system.

Understanding how a microservices application behaves at any given time is an extremely difficult task. While many tools provide deep insight into the behaviour of services, it can be difficult to relate an action of an individual service to the broader understanding of how the entire application behaves.

Tracing solves this problem by adding a unique identifier to the messages sent by the application. That unique identifier allows you to track (or trace) individual transactions as they move through your system. You can use this information to see the health of your application as well as to debug problematic microservices or activities.

Tracing is a powerful debugging tool that allows you to troubleshoot and tune the behaviour of a distributed application. This power comes at a cost. Application code needs to be modified to emit trace data and any traces (a representation of the individual units of work performed in a distributed system) need to be propagated by infrastructure components (e.g., service meshes and their proxies) in your application's data path.



Figure 1.6 CNCF Tracing.

Source: <https://landscape.cncf.io/guide#observability-and-analysis-tracing>

2 Environment

2.1 Git

Git is a free, open source, distributed version control system designed by Linus Torvalds to manage small to very large projects quickly and efficiently.

Version control is a system that records changes made to a file or set of files over time so that specific versions can be retrieved later.

The main features of Git are:

- Open source software.
- Distributed system, it allows the user to work on a project on a local computer or on a server computer. Thus, if any server dies, any of the client repositories can be copied to the server to restore it.
- Maintain complete version control histories.
- Efficiency in managing large projects.
- Speed in managing branches.

To install git on a Debian-based distribution, such as Ubuntu, use apt: `sudo apt install git-all`. From then on, we can use the git commands, typically git add, git commit, git diff, git stash or git clone.

2.2 GitLab

GitLab is a web-based version control tool, an open source repository manager based on Git. It also offers wiki hosting and a bug tracking system, all published in open source.

It was developed by Ukrainian programmers Dmitriy Zaporozhets and Valery Sizov in the Ruby programming language, with some parts later rewritten in Go. At first it was a source code management solution for collaborating with your team on software development. It then evolved into an integrated solution covering the software development lifecycle and, later, the entire DevOps lifecycle. The current technology architecture includes Go, Ruby on Rails and Vue.js.

GitLab Inc. has a team of 1308 members and is used by organisations such as NASA, CERN, IBM and Sony. In the case of this project, all development will be carried out on official GitLab servers, although it also allows installation on private servers.

2.2.1 GitLab Pipelines

Pipelines are the highest level component of continuous integration, delivery and deployment.

Pipelines comprise:

Jobs, which define what to do. For example, jobs that compile or test code. Stages, which define when to execute the jobs. For example, stages that run tests after the stages that compile the code.

Jobs are executed by runners. Several jobs of the same stage are executed in parallel, if there are enough concurrent executors.

If all jobs in a stage succeed, the pipeline moves on to the next stage; if any job in a stage fails, the next stage is (usually) not executed and the pipeline terminates earlier.

In general, pipelines are executed automatically and do not require any intervention once they are created. However, there are also occasions when a pipeline can be manually interacted with.

2.3 Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop (available for Windows, macOS and Linux) or web browser. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity)

On our case, it has a pretty good integration with Git, GitLab, and also, Docker.

2.4 Virtualization

Evolution of deployments:

Traditional deployment era: In the beginning, organisations ran applications on physical servers. There was no way to define resource limits for applications on a physical server, and this caused resource allocation problems. For example, if multiple applications are running on a physical server, there may be cases where one application takes up most of the resources and, as a result, the other applications underperform. One solution to this would be to run each application on a different physical server. However, this was not scalable, as resources were underutilised and it was expensive for organisations to maintain many physical servers.

Virtualized deployment era: As a solution, virtualisation was introduced. It allows multiple virtual machines (VMs) to run on the CPU of a single physical server. Virtualisation allows isolation of applications between VMs and provides a level of security as information from one application cannot be freely accessed by another application.

Virtualisation allows better utilisation of the resources of a physical server and allows better scalability, as an application can be easily added or upgraded, reduces hardware costs and much more. With virtualisation, a set of physical resources can be presented as a cluster of disposable virtual machines.

Each VM is a complete machine running all components, including its own operating system, on the virtualised hardware.

Container deployment era: Containers are similar to VMs, but have relaxed isolation properties for sharing the operating system (OS) between applications. Therefore, containers are considered lightweight. Just like a VM, a container has its own file system, CPU quota, memory, process space, etc. Because they are decoupled from the underlying infrastructure, they are portable between clouds and OS distributions.

LXC: LXC (Linux Containers) is an OS-level virtualisation technology that allows multiple isolated Linux virtual environments (VE) to be created and run on a single control host. These isolation levels or containers can be used either to sandbox specific applications or to emulate an entirely new host. LXC offers the advantages of a Linux VE, primarily the ability to isolate your own private workloads from each other. It is a cheaper and faster solution to deploy than a VM, but doing so requires a bit more learning and experience.

Docker is a significant enhancement to the capabilities of LXC. Its obvious advantages are making Docker more and more popular. In fact, it is coming dangerously close to negating the advantage of VMs over VEs because of its ability to quickly and easily transfer and replicate any package created with Docker.

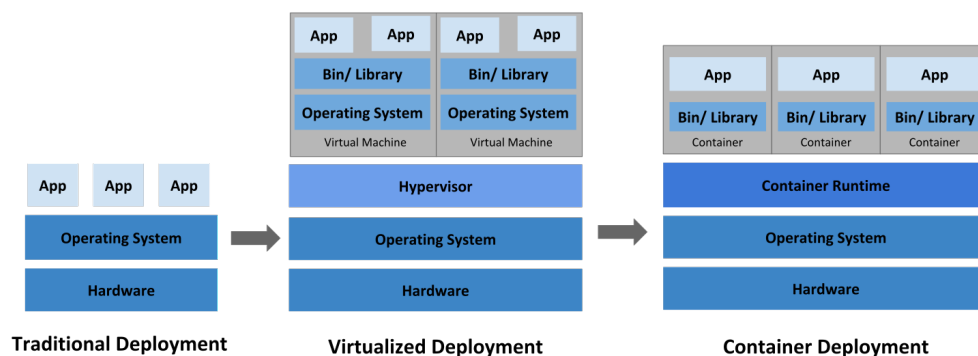


Figure 2.1 Deployments evolution.

Source: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Containers have become popular because they provide additional benefits, such as

- Agile application creation and deployment: greater ease and efficiency in creating container images compared to using VM images.
- Continuous development, integration and deployment: provides reliable and frequent creation and deployment of container images with fast and efficient rollbacks (due to immutability of images).
- Separation of concerns between development and operations: creates application container images at build/release time rather than at deployment time, thus decoupling applications from infrastructure.
- Observability: not only shows OS-level information and metrics, but also application state and other signals.
- Environmental consistency between development, testing and production: Runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, in major public clouds and elsewhere.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic and untethered microservices: applications are broken into smaller, independent pieces and can be dynamically deployed and managed, not a monolithic stack running on a large single-purpose machine.
- Resource isolation: predictable application performance.
- Resource utilisation: high efficiency and density.

2.5 OCI Container (Docker)

A container is a standard software unit that packages code and all its dependencies so that the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, self-contained, executable software package that includes everything needed to run an application: code, runtime, system tools, system libraries and configurations.

Container images become containers at runtime and, in the case of Docker containers, images become containers when run on the Docker Engine. Available for both Linux and Windows-based applications, containerised software will always run the same, regardless of infrastructure. Containers isolate the software from its environment and ensure that it runs consistently despite differences, for example, between development and staging.

Docker containers running on the Docker Engine:

- Standard: Docker created the industry standard for containers, so they could be portable anywhere.
- Lightweight: Containers share the machine's OS system kernel and therefore do not require one operating system per application, driving greater server efficiency and reducing server and licensing costs.
- Secure: Applications are more secure in containers and Docker provides the strongest default isolation capabilities in the industry.

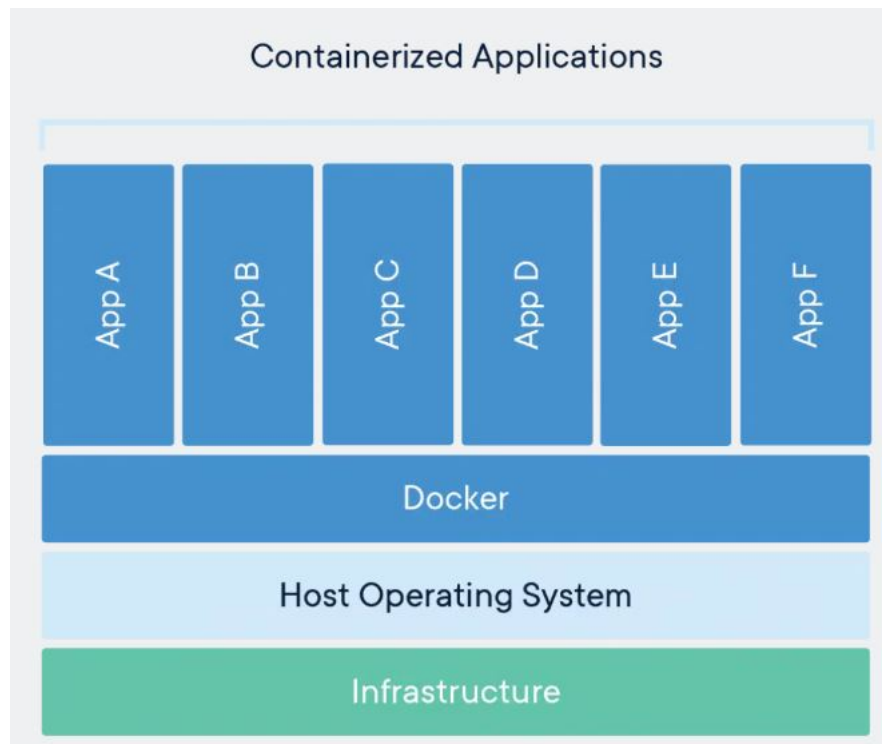


Figure 2.2 Containerized Application structure.

Source: <https://www.docker.com/resources/what-container/>

2.6 Kubernetes cluster

Kubernetes is a portable, extensible, open source platform for managing containerised workloads and services, facilitating both declarative configuration and automation. It has a large and rapidly growing ecosystem. Kubernetes services, support and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. The abbreviation K8s is the result of counting the eight letters between the “K” and the “s”. Google started the Kubernetes project in 2014. Kubernetes combines more than 15 years of Google’s experience in running production workloads at scale with best ideas and practices from the community.

Containers are a great way to cluster and run your applications. In a production environment, you need to manage the containers that run your applications and make sure there is no downtime. For example, if one container goes down, you need another container to start up. Wouldn’t it be easier if this behaviour was handled by a system?

That’s how Kubernetes comes to the rescue. Kubernetes gives you a framework for running distributed systems in a resilient way. It handles scaling and failover of your application, provides deployment patterns, and much more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides:

- Service discovery and load balancing Kubernetes can expose a container using the DNS name or using its own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute network traffic so that the deployment is stable.
- Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storage, public cloud providers, and so on.
- Automated deployments and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and Kubernetes can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for deployment, delete existing containers, and adopt all their resources to the new container.

- You provide Kubernetes with a cluster of nodes that it can use to run tasks on containers. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can arrange the containers on your nodes to make the best use of your resources.
- Kubernetes self-heals and restarts failed containers, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- Secrets and configuration management Kubernetes allows you to store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update application secrets and configuration without having to rebuild container images and without exposing secrets in the stack configuration.

2.6.1 K3D

k3d is a lightweight wrapper to run k3s (Rancher Lab's minimal Kubernetes distribution, k3s-io/k3s) in docker.

k3d creates containerized k3s clusters. This means, that you can spin up a multi-node k3s cluster on a single machine using docker.

k3d is a community-driven project, that is supported by Rancher (SUSE) and it's not an official Rancher (SUSE) product.

2.7 Prometheus

Prometheus is an open source system monitoring and alerting toolkit originally created on SoundCloud. Since its inception in 2012, many companies and organisations have adopted Prometheus, and the project has a very active community of developers and users. It is now an independent open source project and is maintained independently of any company. To emphasise this, and to clarify the governance structure of the project, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes.

Prometheus collects and stores its metrics as time-series data, i.e. metric information is stored with the timestamp at which it was recorded, along with optional key-value pairs called tags.

The main features of Prometheus are

- A multi-dimensional data model with time-series data identified by metric name and key/value pairs.
- PromQL, a flexible query language to take advantage of this dimensionality.
- No reliance on distributed storage; server nodes are self-contained
- Time series collection is done using an extraction model over HTTP
- Time series forwarding is done through an intermediate gateway
- Targets are discovered via service discovery or static configuration
- Multiple modes of graph and dashboard support

This diagram illustrates the Prometheus architecture and some of its ecosystem components:

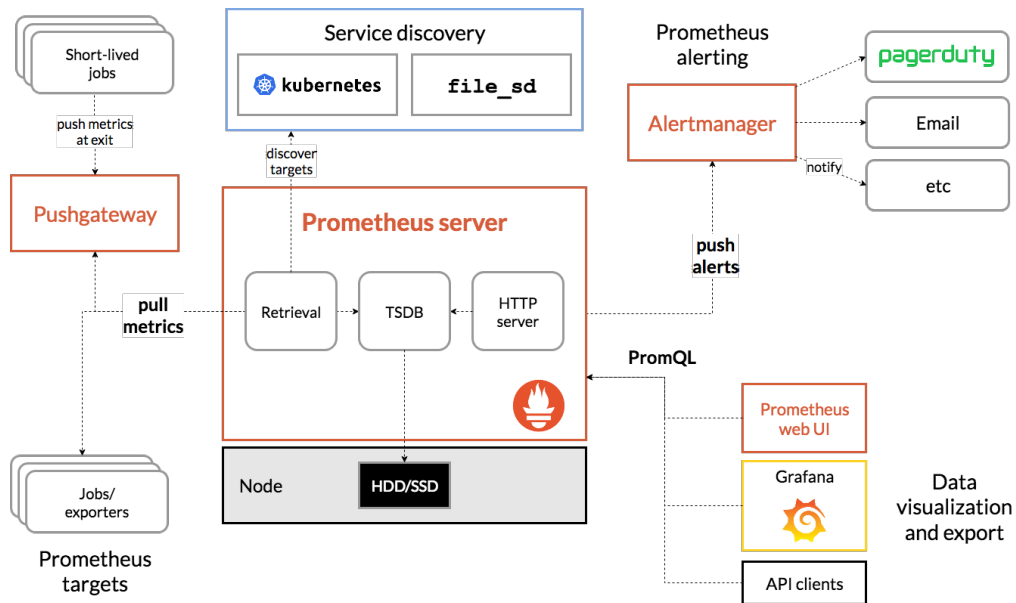


Figure 2.3 Prometheus Architecture.

Source: <https://prometheus.io/docs/>

Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.

2.8 Grafana

Grafana is a complete observability stack that allows you to monitor and analyse metrics, logs and traces. It allows you to query, visualise, alert and understand your data no matter where it is stored. Create, explore and share beautiful dashboards with your team and foster a data-driven culture.

Grafana's key features are:

- **Unify your data, not your database:** Grafana doesn't require you to ingest data into a backend warehouse or vendor database. Instead, Grafana takes a unique approach to providing a "single-pane-of-glass" by unifying your existing data, wherever it lives. With Grafana, you can take any of your existing data - whether it's from your Kubernetes cluster, raspberry pi, different cloud services, or even Google Sheets - and visualise it however you want, all from a dashboard.
- **Data that everyone can see:** Grafana was built on the principle that data should be accessible to everyone in your organisation, not just the single operations person. By democratising data, Grafana helps facilitate a culture where data can be easily used and accessed by the people who need it, helping to break down data silos and empower teams.
- **Dashboards anyone can use:** Grafana dashboards not only give insightful meaning to data collected from numerous sources, but you can also share the dashboards you create with other team members, allowing you to explore data together. With Grafana, anyone can create and share dynamic dashboards to foster collaboration and transparency.
- **Flexibility and versatility:** Translate and transform any of your data into flexible and versatile dashboards. Unlike other tools, Grafana allows you to build dashboards specifically for you and your team. With advanced query and transformation capabilities, you can customise your dashboards to create visualisations that are truly useful to you.

3 Development

On this section we are going to describe the development process of this project: from standalone containers to a complete stack in a kubernetes cluster deployed in a remote server. Starting with docker containers, later podman containers. After integration: docker-compose, and subsequently podman with kubernetes yaml language to eventually achieve a kubernetes cluster.

3.1 Prometheus (standalone)

At this initial point, we are going to build and test the stack with 2 endpoints, where Prometheus collects metrics:

- First, a test endpoint in a Node.js app with just an incremental counter, exporting metrics (port 8080).
- Also, we are using prometheus endpoint itself (port 9090).

First, create prometheus.yml config file, as follows:

Code 3.1 Prometheus config file.

```
1 # global config
2 global:
3   scrape_interval: 15s
4   # Set the scrape interval to every 15 seconds. Default is every 1 minute.
5   evaluation_interval: 15s
6   # Evaluate rules every 15 seconds. The default is every 1 minute.
7   # scrape_timeout is set to the global default (10s).
8
9 # Load rules once and periodically evaluate them according to the global
10 # 'evaluation_interval'.
11 rule_files:
12   # - "first_rules.yml"
13   # - "second_rules.yml"
14
15 # A scrape configuration containing exactly one endpoint to scrape:
16 # Here it's Prometheus itself.
17 scrape_configs:
18   # The job name is added as a label 'job=<job_name>' to any timeseries
19   # scraped from this config.
20   - job_name: 'prometheus'
21     # metrics_path defaults to '/metrics'
22     # scheme defaults to 'http'.
23     static_configs:
24       - targets: ['127.0.0.1:9090']
```

```

25
26 - job_name: 'bot'
27   #metrics_path: 'metrics'
28   # metrics_path defaults to '/metrics'
29   scrape_interval: 5s
30   static_configs:
31   - targets: ['HOST_IP:8080']

```

Need to replace `HOST_IP` with the machine IP. Using `localhost` won't work here because we'll be connecting to the host machine from the docker container.

Run Prometheus docker container with:

Code 3.2 Prometheus docker run execution command.

```

1 docker run --name prometheus -p 9090:9090 -v <PATH_TO_prometheus.yml_FILE>:/etc
  /prometheus/prometheus.yml prom/prometheus --config.file=/etc/prometheus/
  prometheus.yml

```

with the path where Prometheus configuration file has been stored.

Now Prometheus is available on: <http://localhost:9090>

We can see our demo incremental counter:

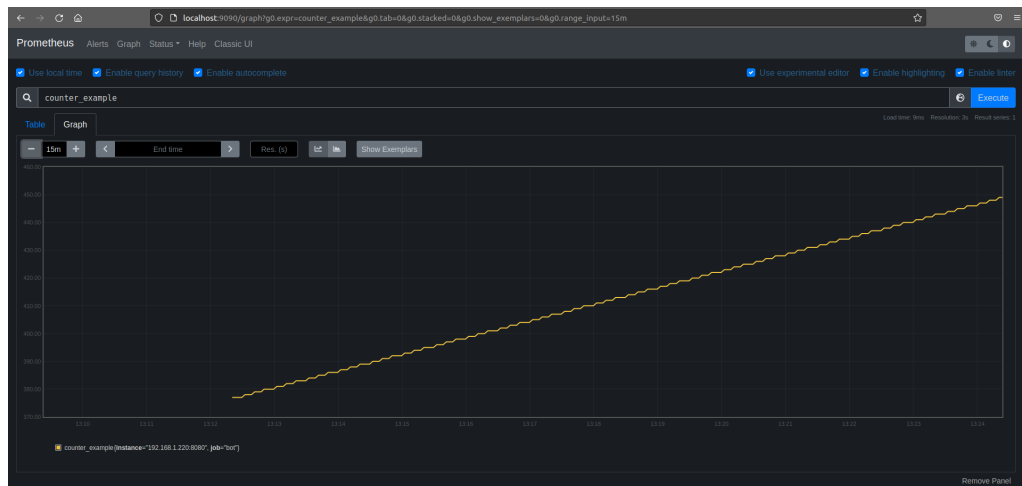


Figure 3.1 Incremental counter metric.

3.2 Grafana (standalone)

Now, we can run Grafana docker container with :

Code 3.3 Grafana docker run execution command.

```

1 docker run --name grafana -p 3000:3000 -v <PATH_TO_GRAFANA>/grafana/
  provisioning/datasources:/etc/grafana/provisioning/datasources grafana/
  grafana

```

Initially, we are going to add Prometheus as a datasource in grafana manually, and then with a config file, in order to achieve “Config as Code”

After docker run command, Grafana is available on: <http://localhost:3000>.

Log in Grafana. Click on Add Data Source and select Prometheus. Then, add prometheus url. At this point, we have set Prometheus-Grafana connection, and we can see Prometheus data in Grafana viewer. On the one side, we have counter example from our test endpoint, on the other one, we have Prometheus metrics itself, as we can see in the following picture.



Figure 3.2 Grafana demo dashboard.

Turn all off, and add “Config as Code” defining that connection in `datasource.yaml` file in above path:

Code 3.4 Grafana datasource.yaml file.

```

1 # config file version
2 apiVersion: 1
3
4 # list of datasources to insert/update depending
5 # what's available in the database
6 datasources:
7   # <string, required> name of the datasource. Required
8   - name: Prometheus
9     # <string, required> datasource type. Required
10    type: prometheus
11    # <string, required> access mode. proxy or direct. Required
12    access: proxy
13    # <int> org id. will default to orgId 1 if not specified
14    #orgId: 1
15    # <string> custom UID which can be used to reference this datasource in
16    # other
17    #parts of the configuration, if not specified will be generated
18    # automatically
19    #uid: my_unique_uid
20    # <string> url
21    #url: localhost:9090
22    url: http://localhost:9090
23    # <bool> allow users to edit datasources from the UI.
24    editable: false

```

Deploy Grafana again, and metrics are available directly, without adding manually Prometheus as data-source.

Auto refresh config has to be done in dashboard page, not inside each panel. This menu is only available there.

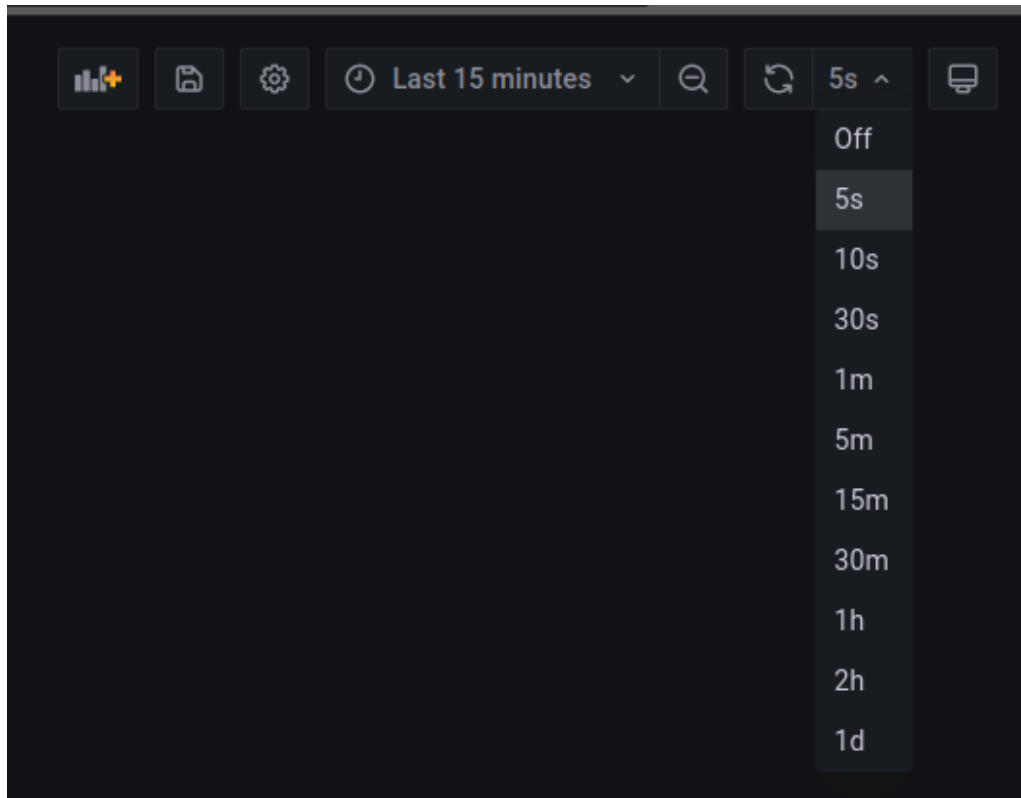


Figure 3.3 Grafana dashboard detail.

3.3 Docker-compose

In order to facilitate deployment, instead of 2 standalone containers, we create a docker-compose. Compose is a tool for defining and running multi-container Docker applications. With Compose, we use a YAML file to configure our application's services. Then, with a single command, we create and start all the services from configuration.

Define the following `docker-compose.yml` with 2 services: Prometheus and Grafana, and its respective volumes:

Code 3.5 `docker-compose.yml` file.

```
1 version: '3.8'
2
3 networks:
4   monitoring:
5     driver: bridge
6
7 volumes:
8   prometheus_data: {}
9   grafana_data: {}
10
11 services:
12   prometheus:
```

```
13 image: prom/prometheus:latest
14 container_name: prometheus
15 restart: unless-stopped
16 #ports:
17 # - '9090:9090'
18 volumes:
19   - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
20   - prometheus_data:/prometheus
21 command:
22   # Prometheus configuration file path.
23   - '--config.file=/etc/prometheus/prometheus.yml'
24   # Base path for metrics storage.
25   - '--storage.tsdb.path=/prometheus'
26   # Path to the console library directory.
27   #- '--web.console.libraries=/etc/prometheus/console_libraries'
28   # Path to the console template directory, available at /consoles.
29   #- '--web.console.templates=/etc/prometheus/consoles'
30   # Enable shutdown and reload via HTTP request.
31   #- '--web.enable-lifecycle'
32 networks:
33   - monitoring
34
35 grafana:
36   image: grafana/grafana
37   container_name: grafana
38   restart: unless-stopped
39   ports:
40     - '80:3000'
41   volumes:
42     - grafana_data:/var/lib/grafana
43     - ./grafana/provisioning:/etc/grafana/provisioning/
44   networks:
45     - monitoring
46   depends_on:
47     - prometheus
```

Launch it with: `docker-compose up`

Compose works fine on local, but no so fine in a deployment, production server with IaC. So, we continue studying alternatives.

4 Deployment

At this section, we are going to describe the steps followed to deploy our application, keeping in mind our objective of IaC. Infrastructure as code (IaC) allows infrastructure to be managed and prepared with code, rather than through manual processes.

With this type of infrastructure, we create configuration files that contain the specifications needed, making it easy to edit and distribute configurations. It also ensures that always the same environment is prepared. The IaC codifies and documents specifications to facilitate configuration management, and helps us avoid ad hoc and undocumented changes.

Version control is an important aspect of IaC that should be applied to configuration files, just like any other software source code file. The implementation of the IaC also allows it to be broken down into modular elements that will be combined in different ways through automation.

By automating infrastructure preparation with the IaC, developers will not have to manually prepare and manage servers, operating systems, storage or any other element each time they develop or deploy an application.

4.1 Local vs remote server

Once we have our stack running with `docker-compose up`, we can try to test it on the remote server side. There are different ways to deploy on a remote host:

1. Manual deployment by copying project files, install docker-compose and running it.

Copy the project source with `docker-compose.yml`, install docker-compose on the target machine and finally run it.

Code 4.1 Manual deployment: copying files.

```
1 scp -r hello-docker user@remotehost:/path/to/src
2 ssh user@remotehost
3 pip install docker-compose
4 cd /path/to/src/hello-docker
5 docker-compose up -d
```

The disadvantages in this case is that for any change in the application sources or compose file, we have to copy, connect to the remote host and re-run.

2. Using Git.

Using ssh key as deploy key. Adding `~/.ssh/id_dsa.pub` to GitLab (project's Settings > Repository page > Expand the Deploy keys section). Then, `git clone`, finally `docker-compose up`.

3. Using `DOCKER_HOST` environment variable to set up the target engine.

Use the `DOCKER_HOST` environment variable scenario to target docker hosts, also it can be achieved by passing the `-H`, `--host` argument to `docker-compose`.

Code 4.2 Change `DOCKER HOST` environment variable.

```
1 cd hello-docker
2 DOCKER_HOST="ssh://user@remotehost" docker-compose up -d
```

This is a better approach than the manual deployment. But it gets quite annoying as it requires to `set/export` the remote host endpoint on every application change or host change.

4. Using docker contexts.

Docker Contexts are an efficient way to automatically switch between different deployment targets:

Code 4.3 Docker context usage.

```
1 docker context create remote --docker "host=ssh://user@remotemachine"
2 remote
3 Successfully created context "remote"
4
5 docker context ls
6 NAME      DESCRIPTION          DOCKER ENDPOINT  KUBERNETES ENDPOINT
   ORCHESTRATOR
7 default * Current DOCKER_HOST unix:///var/run/docker.sock      swarm
8 remote                    ssh://user@remotemachine
9
10 docker-compose --context remote up -d
```

or using `docker context use remote`.

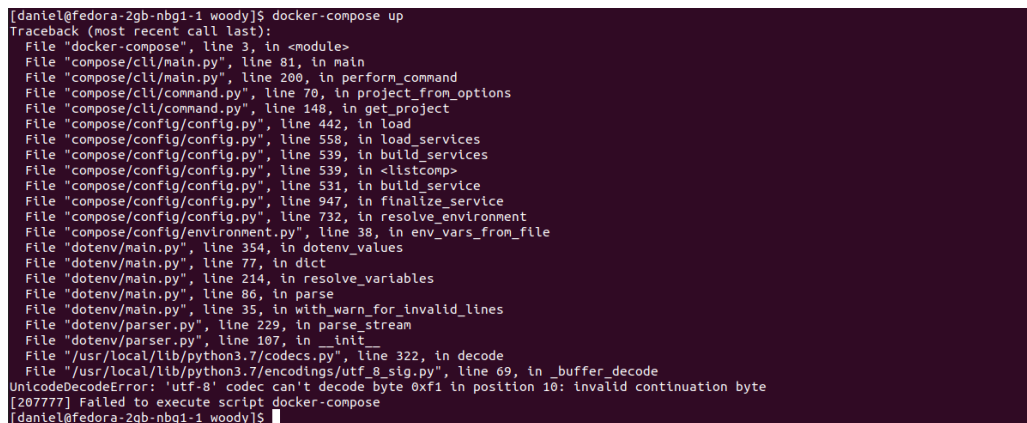
Docker Contexts are an efficient way to automatically switch between different deployment targets.

4.1.1 Initial Deployment with Git

Following these stages:

1. (remote) `ssh-keygen -t rsa -C stackGitLab`
2. (GitLab) Paste remote public key as deploy key
3. (remote) `git clone -b define-monitoring-stack ssh://git@url`
4. (remote) `docker-compose up`

And we get:



```
[dantel@fedora-2gb-nbg1-1 woody]$ docker-compose up
Traceback (most recent call last):
  File "docker-compose", line 3, in <module>
  File "compose/cli/main.py", line 81, in main
  File "compose/cli/main.py", line 200, in perform_command
  File "compose/cli/command.py", line 70, in project_from_options
  File "compose/cli/command.py", line 148, in get_project
  File "compose/config/config.py", line 442, in load
  File "compose/config/config.py", line 558, in load_services
  File "compose/config/config.py", line 539, in build_services
  File "compose/config/config.py", line 539, in <listcomp>
  File "compose/config/config.py", line 531, in build_service
  File "compose/config/config.py", line 947, in finalize_service
  File "compose/config/config.py", line 732, in resolve_environment
  File "compose/config/environment.py", line 38, in env_vars_from_file
  File "dotenv/main.py", line 354, in dotenv_values
  File "dotenv/main.py", line 77, in dict
  File "dotenv/main.py", line 214, in resolve_variables
  File "dotenv/main.py", line 86, in parse
  File "dotenv/main.py", line 35, in with_warn_for_invalid_lines
  File "dotenv/parser.py", line 229, in parse_stream
  File "dotenv/parser.py", line 107, in __init__
  File "/usr/local/lib/python3.7/codecs.py", line 322, in decode
  File "/usr/local/lib/python3.7/encodings/utf_8_sig.py", line 69, in _buffer_decode
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 10: invalid continuation byte
[207777] Failed to execute script docker-compose
[dantel@fedora-2gb-nbg1-1 woody]$
```

Figure 4.1 `docker-compose` command on remote.

This is due to the .secret encrypted folder and files. The solution is to create a gpg user in remote and add that user id to git-crypt to be able to unlock. But, we are going to deploy it by CI, making use of GitLab's features.

4.1.2 Deployment with Git and GitLab CI

1. **Install and configure a gitlab runner in remote** Following GitLab tutorial: <https://docs.gitlab.com/runner/install/index.html>
2. **Add git-crypt key as CI/CD variable**

It works successfully !

4.2 Podman

Next step is the use of podman instead of docker. Podman is a daemonless container engine for developing, managing, and running OCI Containers on your Linux System. Containers can either be run as root or in rootless mode. Advantages of Podman are present in its definition: daemonless and containers in rootless mode. With podman we can use a yaml language very similar to kubernetes one, which will be our next stage.

Create a pod, publishing port 8080/TCP from internal 3000/TCP

Code 4.4 Podman create pod command.

```
1 podman pod create \
2   --name stack-pod \
3   --publish 8080:3000/TCP
```

Create a first container inside the pod

Code 4.5 Podman create container command.

```
1 podman run --detach \
2   --pod stack-pod \
3   --name prometheus \
4   --env MY_VAR="my val" \
5   prom/prometheus:latest
```

Create a second container inside the pod

Code 4.6 Podman create second container command.

```
1 podman run --detach \
2   --pod stack-pod \
3   --name grafana \
4   --env MY_VAR="my val" \
5   grafana/grafana
```

Check by

Code 4.7 Check containers and pod command.

```
1 podman container ls; podman pod ls
```

Export it as a Pod manifest by using

Code 4.8 Export as a pod manifest command.

```
1 podman generate kube stack-pod > stack-pod.yaml
```

Use a Kubernetes-compatible Pod manifest to create and run a pod

Code 4.9 Use a Kubernetes-compatible Pod manifest to create and run a pod.

```
1 podman play kube stack-pod.yaml
```

4.2.1 Troubleshooting podman deployment

After changing every volume to HostPath type, we realize that Prometheus container was just exiting after creating, with Podman logs ID:

Code 4.10 Troubleshooting podman deployment.

```
1 ...
2 level=info ts=2021-09-24T09:10:45.791Z caller=main.go:446 vm_limits="(soft=
  unlimited, hard=unlimited)"
3 level=error ts=2021-09-24T09:10:45.791Z caller=query_logger.go:87 component=
  activeQueryTracker msg="Error opening query log file" file=/prometheus/
  queries.active err="open /prometheus/queries.active: permission denied"
4 panic: Unable to create mmap-ed active query log
5
6 goroutine 1 [running]:
7 github.com/prometheus/prometheus/promql.NewActiveQueryTracker({0x7fff7b505f60,
  0xb}, 0x14, {0x33ecfe0, 0xc000287b30})
8     /app/promql/query_logger.go:117 +0x3d7
9 main.main()
10     /app/cmd/prometheus/main.go:474 +0x6711
```

Permission denied when trying to access volume directory. Following github.com/prometheus/prometheus/issues/5976, we need to grant access with:

Code 4.11 Init container to grant access.

```
1   initContainers:
2   - name: prometheus-data-permission-setup
3     image: busybox
4     command: ["/bin/chown", "-R", "65534:65534", "/prometheus"]
5     volumeMounts:
6     - name: storage-volume
7       mountPath: /prometheus
```

Init Containers are in Pre-release 2 of v3.4.0 github.com/containers/podman/releases/tag/v3.4.0-rc2

4.3 Kubernetes

After these problems with podman, next stage is Kubernetes: after the research made in the state of art, we decide to use kind and k3d.

First, `kubectl`, Kubernetes command-line tool, allows you to run commands against Kubernetes clusters. You can use `kubectl` to deploy applications, inspect and manage cluster resources, and view logs. Install `kubectl`, following kubernetes.io/docs/tasks/tools/install-kubectl-linux/.

4.3.1 Kind

Install kind, following kind.sigs.k8s.io/docs/user/quick-start/

To create a cluster, just `kind create cluster --name my-cluster`.

In a real case, we would need a config file, then `kind create cluster --config multi-node.yaml`, a config file could be like that:

Code 4.12 Kind config file.

```

1 kind: Cluster
2 apiVersion: kind.x-k8s.io/v1alpha4
3 nodes:
4 - role: control-plane
5   kubeadmConfigPatches:
6   - |
7     kind: InitConfiguration
8     nodeRegistration:
9       kubeletExtraArgs:
10        node-labels: "ingress-ready=true"
11   extraPortMappings:
12   - containerPort: 80
13     hostPort: 80
14     protocol: TCP
15   - containerPort: 443
16     hostPort: 443
17     protocol: TCP
18 - role: control-plane
19 - role: control-plane
20 - role: worker
21 - role: worker
22 - role: worker

```

6 nodes: 3 control plane and 3 workers

4.3.2 K3d

Install k3d, following k3d.io/v4.4.8/#installation.

k3d is a lightweight wrapper to run k3s (Rancher Lab's minimal Kubernetes distribution) in docker.

To create a cluster, just `k3d cluster create my-cluster`.

With a config file `k3d cluster create --config k3d.yaml`, a config file could be like that:

Code 4.13 K3d config file.

```

1 kind: Simple
2 apiVersion: k3d.io/v1alpha2
3 name: my-cluster
4 image: rancher/k3s:v1.20.4-k3s1
5 servers: 3
6 agents: 3
7 ports:
8 - port: 80:80
9   nodeFilters:
10  - loadbalancer
11 # options:
12 #   k3s:
13 #     extraServerArgs:

```

```
14 # --no-deploy=traefik
```

6 nodes: 3 servers (control plane) and 3 agents (workers).
As we describe in the state of art section, we are going to use k3d.

4.3.3 Troubleshooting launching pod manifest

Running the command: `kubectl create -f stack-pod.yaml`

Code 4.14 Troubleshooting launching pod manifest.

```
1 Error from server (Invalid): error when creating "stack-pod.yaml": Pod "stack-
  pod" is invalid: [spec.volumes[0].name: Invalid value: "prometheus_data": a
  lowercase RFC 1123 label must consist of lower case alphanumeric
  characters or '-', and must start and end with an alphanumeric character (e
  .g. 'my-name', or '123-abc', regex used for validation is '[a-z0-9]([-a-z0
  -9]*[a-z0-9])?')
```

Solution is simple, just change name, replacing "_" to "-".
Then `kubectl create -f stack-pod.yaml` command ok.
Debugging: `kubectl describe pods`:

Code 4.15 kubectl describe pods command result.

```
1 ...
2 Events:
3   Type      Reason      Age           From          Message
4   ----      -
5   Normal    Scheduled   2m12s        default-scheduler Successfully
  assigned default/stack-pod to k3d-my-cluster-agent-1
6   Warning   FailedMount 117s (x6 over 2m12s) kubelet       MountVolume.SetUp
  failed for volume "prometheus-data" : hostPath type check failed: ./
  prometheus_data is not a directory
7   Warning   FailedMount 117s (x6 over 2m12s) kubelet       MountVolume.SetUp
  failed for volume "grafana-data" : hostPath type check failed: ./
  grafana_data is not a directory
8   Warning   FailedMount 117s (x6 over 2m12s) kubelet       MountVolume.SetUp
  failed for volume "prometheus-config" : hostPath type check failed: ./
  prometheus/prometheus.yml is not a directory
9   Warning   FailedMount 100s (x7 over 2m12s) kubelet       MountVolume.SetUp
  failed for volume "grafana-config" : hostPath type check failed: ./
  grafana/provisioning/ is not a directory
```

Changed volumes and run: `k3d cluster create my-cluster --volume /home/daniel/dev/c/stack:/tmp/k3dvol --servers 1 --agents 1`

Then, apply secret: `kubectl apply -f k8s/stack-secret.yaml`

Finally, apply deployment: `kubectl apply -f k8s/stack-deployment.yaml`

Now, status is:

Code 4.16 kubectl describe pods command result (2).

```
1 kubectl get pod
2 NAME                                READY   STATUS    RESTARTS   AGE
3 svclb-stack-service-7rxwv 1/1     Running   0           3m2s
4 svclb-stack-service-9gspl 1/1     Running   0           3m2s
```

```
5 stack-pod-7574b7d7c9-kfglh 0/3   CrashLoopBackOff 8           3m2s
```

Then, running `kubectl logs stack-pod-549fd5c79-62mx2 prometheus`:

Code 4.17 `kubectl log` command result.

```
1 level=error ts=2021-10-05T09:15:51.702Z caller=query_logger.go:87 component=
  activeQueryTracker msg="Error opening query log file" file=/prometheus/
  queries.active err="open /prometheus/queries.active: permission denied"
2 panic: Unable to create mmap-ed active query log
```

`InitContainers` are necessary to give access to volumes:

Code 4.18 `Init containers` to give access to volumes.

```
1 initContainers:
2   - name: grafana-vol-setup
3     image: alpine:3
4     # Give 'grafana' user (id 472) permissions a mounted volume
5     # https://github.com/grafana/grafana-docker/blob/master/Dockerfile
6     command:
7     - chown
8     - -R
9     - 472:472
10    - /var/lib/grafana
11    volumeMounts:
12    - name: grafana-data
13      mountPath: /var/lib/grafana
14  - name: prometheus-vol-setup
15    image: alpine:3
16    command:
17    - chown
18    - -R
19    - 65534:65534
20    - /prometheus
21    volumeMounts:
22    - name: prometheus-data
23      mountPath: /prometheus
```

4.3.4 Current Usage

Set-up environment:

Code 4.19 Set-up environment commands.

```
1 - k3d cluster create my-cluster --volume /home/daniel/dev/c/stack:/tmp/k3dvol
  --servers 1 --agents 1'
2 - kubectl apply -f k8s/stack-secret.yaml'
3 - kubectl apply -f k8s/stack-secret-regkey.yaml'
4 - kubectl apply -f k8s/stack-deployment.yaml'
```

Debugging:

Code 4.20 Debug commands.

```

1 - kubectl describe pods
2 - kubectl get pods
3 - kubectl get service
4 - kubectl logs POD CONTAINER

```

4.3.5 Add PV and PVC

Next step is to add a Persistent Volume. A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. Following kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/, we need:

- PVC file
- PV file
- Updated deployment/pod using this new volume.

From the above link:

In a production cluster, you would not use `hostPath`. Instead a cluster administrator would provision a network resource like a Google Compute Engine persistent disk, an NFS share, or an Amazon Elastic Block Store volume. Cluster administrators can also use `StorageClasses` to set up dynamic provisioning.

NFS example

Creating a NFS container with a `docker-compose`, following: blog.ruanbekker.com/blog/2020/09/20/setup-a-nfs-server-with-docker/ and sysadmins.co.za/setup-a-nfs-server-with-docker/

1. `docker-compose.yml`

Code 4.21 NFS `docker-compose.yml` file.

```

1 version: "2.1"
2 services:
3   # https://hub.docker.com/r/itsthetwork/nfs-server-alpine
4   nfs:
5     image: itsthetwork/nfs-server-alpine:12
6     container_name: nfs
7     restart: unless-stopped
8     privileged: true
9     environment:
10      - SHARED_DIRECTORY=/data
11     volumes:
12      - /home/daniel/dev/c/stack/nfs_test/data/nfs-storage:/data
13     ports:
14      - 2049:2049

```

2. Run: `docker-compose up -d`
3. Mount: `sudo mount -v -o vers=4,loud IP:/ /home/daniel/dev/c/stack/nfs_test/mnt`, where IP is our local ip
4. Testing, `df -h` and then, creating a file in `mnt` and checked that is also in `data/nfs-storage`.

Using external NFS in the cluster

Following kubernetes PV NFS description: github.com/kubernetes/examples/tree/master/staging/volumes/nfs
Create a new PV Claim:

Code 4.22 PV Claim yaml file.

```

1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4    name: nfs
5  spec:
6    accessModes:
7      - ReadWriteMany
8    storageClassName: ""
9    resources:
10   requests:
11     storage: 1Gi

```

Create new PV:

Code 4.23 PV yaml file.

```

1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: nfs
5    labels:
6      type: stack
7  spec:
8    capacity:
9      storage: 2Gi
10   accessModes:
11     - ReadWriteMany
12   nfs:
13     server: 192.168.1.220
14     path: "/"

```

Changed deployment file, just new claimName: claimName: nfs
After applying:

Code 4.24 Debugging volumes.

```

1  ...
2  Events:
3  ...
4  Warning FailedMount 2m54s          kubelet          Unable to attach or
      mount volumes: unmounted volumes=[prometheus-data-pv], unattached volumes=[
      grafana-data kube-api-access-w6q4k prometheus-data-pv prometheus-config
      grafana-config]: timed out waiting for the condition
5  Warning FailedMount 47s (x10 over 4m57s) kubelet          MountVolume.SetUp
      failed for volume "nfs" : mount failed: exit status 255
6  Mounting command: mount
7  Mounting arguments: -t nfs 192.168.1.220:/ /var/lib/kubelet/pods/60f1761f-c6ca
      -4209-af5d-e0920b977175/volumes/kubernetes.io~nfs/nfs

```

```

8 Output: mount: mounting 192.168.1.220:/ on /var/lib/kubelet/pods/60f1761f-c6ca
  -4209-af5d-e0920b977175/volumes/kubernetes.io~nfs/nfs failed: Not supported
9 Warning FailedMount 39s kubelet Unable to attach or mount volumes: unmounted
  volumes=[prometheus-data-pv], unattached volumes=[prometheus-config
  grafana-config grafana-data kube-api-access-w6q4k prometheus-data-pv]:
  timed out waiting for the condition

```

At this point, we add a makefile to make things easier, just run: `make all`.

Later, changed that makefile, to a bash script as we are treating with bash commands (k3d and kubectl...). Use: `sh env-up.sh up` to set environment up, or `sh env-up.sh down` to destroy the cluster. Use 'help' subcommand for support

Troubleshooting NFS

First, we change NFS ip, from 192.168.1.220 to 127.0.0.1, now the error is 'Connection refuse':

Code 4.25 Troubleshooting NFS.

```

1 Events:
2   Type      Reason      Age           From          Message
3   ----      -
4   Normal    Scheduled   14s          default-scheduler Successfully assigned
  default/stack-pod-5bcff58c66-ck8hg to k3d-my-cluster-server-0
5   Warning   FailedMount 7s (x5 over 15s) kubelet      MountVolume.Setup failed
  for volume "nfs" : mount failed: exit status 255
6 Mounting command: mount
7 Mounting arguments: -t nfs 127.0.0.1:/ /var/lib/kubelet/pods/41c9cda7-2882-44f1
  -868d-827e6ddb5433/volumes/kubernetes.io~nfs/nfs
8 Output: mount: mounting 127.0.0.1:/ on /var/lib/kubelet/pods/41c9cda7-2882-44f1
  -868d-827e6ddb5433/volumes/kubernetes.io~nfs/nfs failed: Connection refused

```

Change NFS server IP in `pv-volume-nfs.yaml` file to 172.17.0.1 (docker0), and added again `initContainers` to give permissions to access volumes.

Current usage

1. In `nfs_test` directory, run `docker-compose up` to start nfs server in a container
2. Use `env-up.sh` file to start and stop the stack. Run `./env-up.sh up`
3. Grafana is available at `service_IP:30000`, in my case: 172.25.0.2:30000

4.3.6 Added new pv and pvc for grafana

It's not possible to use the same persistent volume for 2 different claims, so, new pv and pvc for grafana are needed.

Code 4.26 1 Volume and 2 usages problem.

```

1 kubectl logs stack-pod-6b96665766-wc75p grafana
2 GF_PATHS_DATA='/var/lib/grafana' is not writable.
3 You may have issues with file permissions, more information here: http://docs.
  grafana.org/installation/docker/#migrate-to-v51-or-later
4 mkdir: can't create directory '/var/lib/grafana/plugins': Permission denied

```

Need to change path in each PV, in order to not write in the same directory and avoiding permission problems.

4.3.7 Kustomize

Kustomize lets you customize raw, template-free YAML files for multiple purposes, leaving the original YAML untouched and usable as is. Kustomize targets kubernetes; it understands and can patch kubernetes

style API objects. It's like make, in that what it does is declared in a file, and it's like sed, in that it emits edited text.

Create a config map to avoid mounting local directory as volumen in the cluster. To maintain also yaml config file, we use kustomize to create updated config file.

Now: `kubectl apply -k k8s`

Code 4.27 Initial kustomize apply command result.

```
1 error: loading KV pairs: file sources: [../prometheus/prometheus.yml]: security
   ; file '/home/daniel/dev/c/stack/prometheus/prometheus.yml' is not in or
   below '/home/daniel/dev/c/stack/k8s'
```

Prometheus directry needs to be in k8s folder.

It's importanto to add resource field to kustomization file in order to link with deployment file. [kubectl.docs.kubernetes.io/references/kustomize/configmapgenerator/#propagating-the-name-suffix](https://kubernetes.io/docs/reference/kustomize/configmapgenerator/#propagating-the-name-suffix)

So, kustomize file is now:

Code 4.28 Kustomize yaml file.

```
1 apiVersion: kustomize.config.k8s.io/v1beta1
2 kind: Kustomization
3 configMapGenerator:
4 - name: prometheus-config
5   files:
6 - prometheus/prometheus.yml
7 - name: grafana-config
8   files:
9 - grafana/provisioning/datasources/datasource.yml
10 resources:
11 - stack-deployment.yaml
```

If we remove `kubectl apply -k k8s` from sript and then running it:

Code 4.29 Debugging kustomization.

```
1 kubectl get pod
2 NAME                                READY STATUS    RESTARTS AGE
3 stack-pod-7988b55b58-4pfbq 0/3   Init:0/2    0         29s
4 svclb-stack-service-st5zt 1/1   Running     0         29s
5 svclb-stack-service-mqgv5 1/1   Running     0         29s
6
7 kubectl describe pod stack-pod-7988b55b58-4pfbq
8
9 ...
10 Events:
11 Type      Reason      Age           From           Message
12 ----      -
13 Normal    Scheduled   42s          default-scheduler Successfully assigned
   default/stack-pod-7988b55b58-4pfbq to k3d-my-cluster2-server-0
14 Warning   FailedMount 11s (x7 over 42s) kubelet        MountVolume.SetUp
   failed for volume "prometheus-config" : configmap "prometheus-config" not
   found
```

```

15 Warning FailedMount 11s (x7 over 42s) kubelet      MountVolume.SetUp
    failed for volume "grafana-config" : configmap "grafana-config" not found
16
17 kubectl apply -k k8s
18 configmap/grafana-config-thc62ch6fk created
19 configmap/prometheus-config-fgchfkb88h created
20 service/stack-service unchanged
21 deployment.apps/stack-pod configured
22
23 kubectl get pod
24 NAME                                READY   STATUS              RESTARTS   AGE
25 stack-pod-7988b55b58-4pfbq 0/3     Init:0/2           0           102s
26 svclb-stack-service-st5zt 1/1     Running            0           102s
27 svclb-stack-service-mqgv5 1/1     Running            0           102s
28 stack-pod-57779fd84d-42rz6 0/3     PodInitializing    0           8s
29
30 kubectl get pod
31 NAME                                READY   STATUS    RESTARTS   AGE
32 svclb-stack-service-st5zt 1/1     Running   0           2m24s
33 svclb-stack-service-mqgv5 1/1     Running   0           2m24s
34 stack-pod-57779fd84d-42rz6 3/3     Running   0           50s

```

Switch service and deployment in yaml file

Reading <https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/>:

The resources will be created in the order they appear in the file. Therefore, it's better to specify the service first, since that will ensure the scheduler can spread the pods associated with the service as they are created by the controller(s), such as Deployment.

4.3.8 K3D version update

At this point, we update manually k3d config file version (from v1alpha2 to v1alpha3)

Updated k3d-config.yaml:

Code 4.30 K3d config yaml file.

```

1 # https://github.com/rancher/k3d/blob/main/pkg/config/v1alpha3/schema.json
2 apiVersion: k3d.io/v1alpha3
3 kind: Simple
4 name: my-cluster
5 servers: 1
6 agents: 1
7 #kubeAPI:
8 # host: "dredg.local"
9 # hostIP: 0.0.0.0
10 # hostPort: "6443"
11 image: rancher/k3s:v1.22.2-k3s1
12 network: my-cluster-network
13 #Failed Cluster Preparation: Failed Network Preparation: cannot specify subnet
    for exiting network
14 #subnet: "172.28.0.0/16"
15 #volumes:
16 # - volume: /tmp:/tmp/somepath
17 #   nodeFilters:
18 #     - all
19 ports:

```

```

20 - port: 8080:80
21   nodeFilters:
22     - loadbalancer
23 - port: 0.0.0.0:8443:443
24   nodeFilters:
25     - loadbalancer
26 options:
27   k3d:
28     wait: true
29     timeout: 6m0s
30     disableLoadbalancer: false
31     disableImageVolume: false
32     disableRollback: false
33     #loadbalancer:
34     # configOverrides:
35     #   - settings.workerConnections=2048
36   k3s:
37     extraArgs:
38       - arg: --tls-san=127.0.0.1
39       nodeFilters:
40         - server:*
41     #nodeLabels:
42     # # same as '--k3s-node-label 'foo=bar@agent:1'' -> this results in a
43     #   Kubernetes node label
44     # - label: foo=bar
45     #   nodeFilters:
46     #     - agent:1
47   kubeconfig:
48     updateDefaultKubeconfig: true
49     switchCurrentContext: true
50   runtime:
51     gpuRequest: ""
52     serversMemory: ""
53     agentsMemory: ""
54     #labels:
55     # - label: foo=bar
56     #   nodeFilters:
57     #     - server:0
58     #     - loadbalancer

```

If we just try to set it up:

Code 4.31 Debugging new k3d version.

```

1 ./env-up.sh up
2 Going up
3 FATA[0000] Schema Validation failed for config file k3d-config.yaml: - options.
4   k3s: Additional property extraArgs is not allowed
5 - apiVersion: apiVersion must be one of the following: "k3d.io/v1alpha2"

```

Our current version is:

Code 4.32 K3d and K3s versions.

```

1 k3d --version

```

```

2 k3d version v4.4.8
3 k3s version v1.21.3-k3s1 (default)

```

After updating k3d to v5, now it works. Following GitOps principles, we had minimize this kind of problems, as we have Git as a source of truth, and rollback is available.

4.3.9 Deployment kustomize

Testing kustomize, to avoid double pod creation.

If we let that part of the script:

Code 4.33 Set-up script.

```

1 ...
2 ## Apply deployment
3 kubectl apply -f k8s/stack-deployment.yaml
4
5 # Use kustomize to generate update config map
6 kubectl apply -k k8s

```

Grafana panel is:

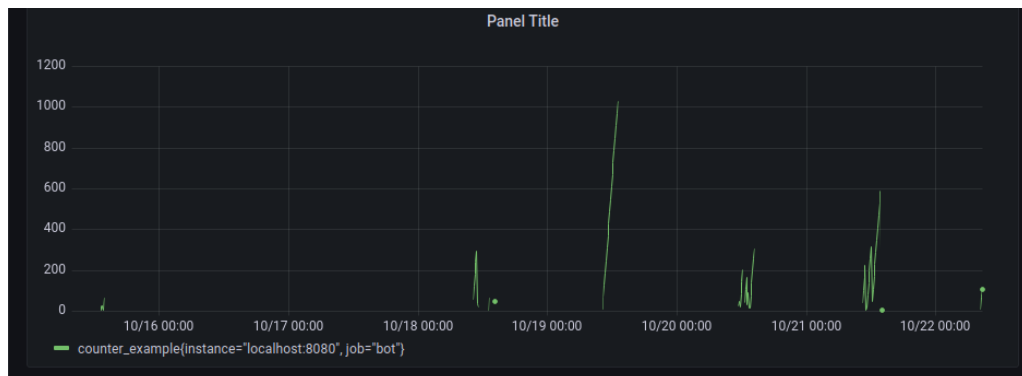


Figure 4.2 Grafana panel (1).

But, if we remove deployment apply:

Code 4.34 Set-up script (2).

```

1 ...
2 ## Apply deployment
3 #kubectl apply -f k8s/stack-deployment.yaml
4
5 # Use kustomize to generate update config map
6 kubectl apply -k k8s

```

Grafana panel now is:

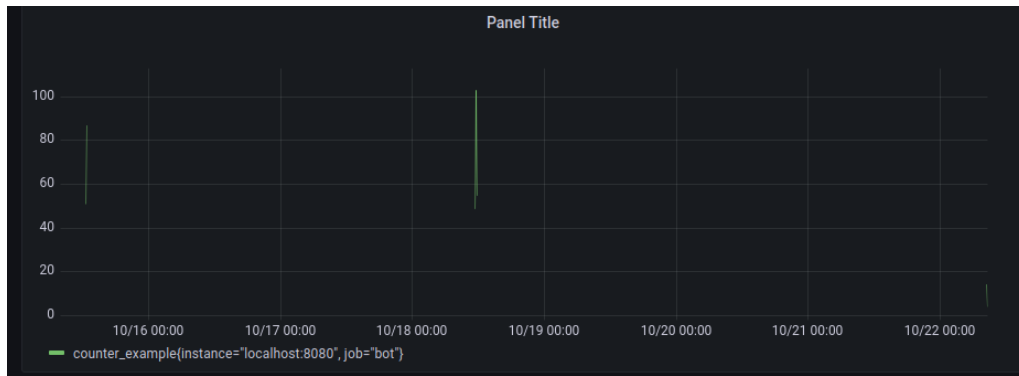


Figure 4.3 Grafana panel (2).

Also, without deployment, another execution:

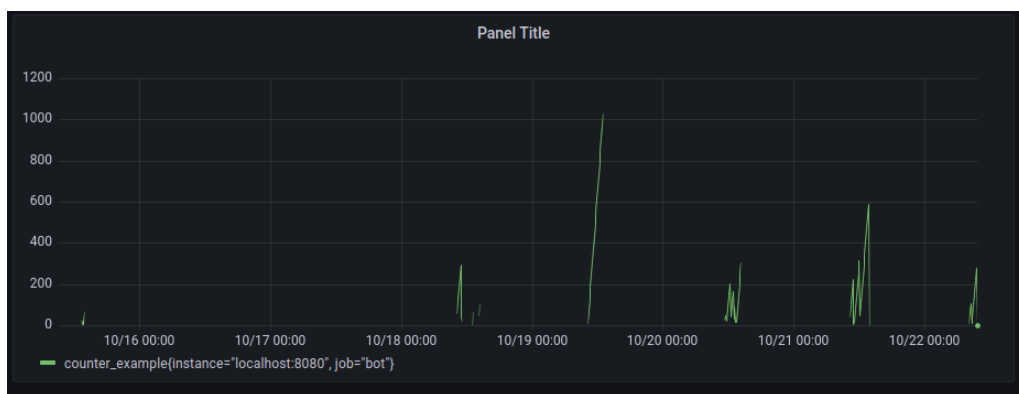


Figure 4.4 Grafana panel (3).

To fix deployment with kustomize, move resources from script to kustomization file.
Old script:

Code 4.35 Old set-up script.

```

1  ...
2  # Apply secrets
3  kubectl apply -f k8s/stack-secret.yaml
4  kubectl apply -f k8s/stack-secret-regkey.yaml
5
6  # Apply config map
7  #kubectl apply -f k8s/prom-configmap.yaml
8  #kubectl apply -f k8s/grafana-configmap.yaml
9
10 # Apply volumes nfs
11 kubectl apply -f k8s/pv-volume-nfs-prom.yaml
12 kubectl apply -f k8s/pv-volume-nfs-grafana.yaml
13 kubectl apply -f k8s/pv-claim-nfs-prom.yaml
14 kubectl apply -f k8s/pv-claim-nfs-grafana.yaml
15
16 ## Apply volumes
17 #kubectl apply -f k8s/pv-volume.yaml
18 #kubectl apply -f k8s/pv-claim.yaml
19

```

```

20  ## Apply deployment
21  kubectl apply -f k8s/stack-deployment.yaml
22
23  # Use kustomize to generate update config map
24  kubectl apply -k k8s
25  ...

```

Old kustomization file:

Code 4.36 Old kustomization file.

```

1  ...
2  Resources:
3  - stack-deployment.yaml

```

New script:

Code 4.37 New set-up script.

```

1  ...
2  # Use kustomize to apply resources
3  kubectl apply -k k8s

```

New kustomization file:

Code 4.38 New kustomization file.

```

1  ...
2  resources:
3  # Secrets
4  - stack-secret.yaml
5  - stack-secret-registry.yaml
6  # Config map
7  #- prom-configmap.yaml
8  #- grafana-configmap.yaml
9  # volumes nfs
10 - pv-volume-nfs-prom.yaml
11 - pv-volume-nfs-grafana.yaml
12 - pv-claim-nfs-prom.yaml
13 - pv-claim-nfs-grafana.yaml
14 ## volumes
15 #- pv-volume.yaml
16 #- pv-claim.yaml
17 ## Deployment
18 - stack-deployment.yaml

```

4.3.10 Ingress

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Here is a simple example where an Ingress sends all its traffic to one Service:

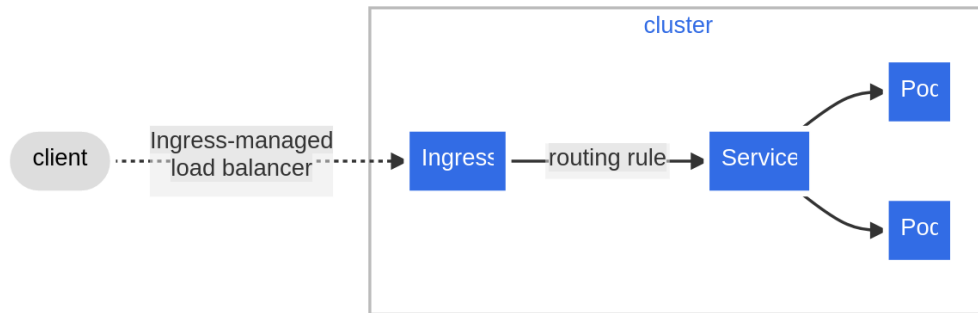


Figure 4.5 Ingress architecture.

Source: <https://kubernetes.io/docs/concepts/services-networking/ingress/>
 Ingress yaml now is:

Code 4.39 Ingress yaml file.

```

1 #apiVersion: networking.k8s.io/v1beta1 # for k3s < v1.19
2 apiVersion: networking.k8s.io/v1
3 kind: Ingress
4 metadata:
5   name: stack-ingress
6   annotations:
7     ingress.kubernetes.io/ssl-redirect: "false"
8 spec:
9   rules:
10  - http:
11    paths:
12    - path: /
13      pathType: Prefix
14      backend:
15        service:
16          name: stack-service
17          port:
18            number: 80

```


5 Metrics exporters

Once the stack is developed and deployed, we need exporters to monitor systems and make metrics available to Prometheus.

5.1 Lm-sensor exporter

lm_sensors (Linux-monitoring sensors) is a free open-source software-tool for Linux that provides tools and drivers for monitoring temperatures, voltage, humidity, and fans. Instead of installing it locally, following our GitOps principles, we are going to deploy it in a container, using this image with the prometheus exporter included: github.com/epfl-sti/cluster.coreos.prometheus-sensors

Steps:

- Run container with `docker run --rm -d --publish 172.17.0.1:9255:9255 epflsti/cluster.coreos.prometheus-sensors`
- Add this new exporter to prometheus config file
- Fix ip connection problems server-cluster-docker

lm-sensor:



Figure 5.1 lm-sensor grafana.

5.2 Node-exporter

Node Exporter is a Prometheus exporter for server level and OS level metrics with configurable metric collectors. It helps us in measuring various server resources such as RAM, disk space, and CPU uti-

lization, Prometheus node exporter. Like lm-sensor exporter, we will deploy it in a container image : github.com/prometheus/node_exporter#docker.

Data available: CPU and disk I/O statistics, hardware monitoring, memory and network statistics, boot time, forks and interrupts, thermal zone & cooling device statistics, To see all data collectors : github.com/prometheus/node_exporter#enabled-by-default

Steps:

- Update prometheus config file
 - Run container image: Change --net flag in docker run command to --publish ip:port:port
- Run:

```
docker run -d --publish 172.17.0.1:9100:9100 \
--pid="host" \
-v "/:/host:ro,rslave" \
quay.io/prometheus/node-exporter:latest \
--path.rootfs=/host
```

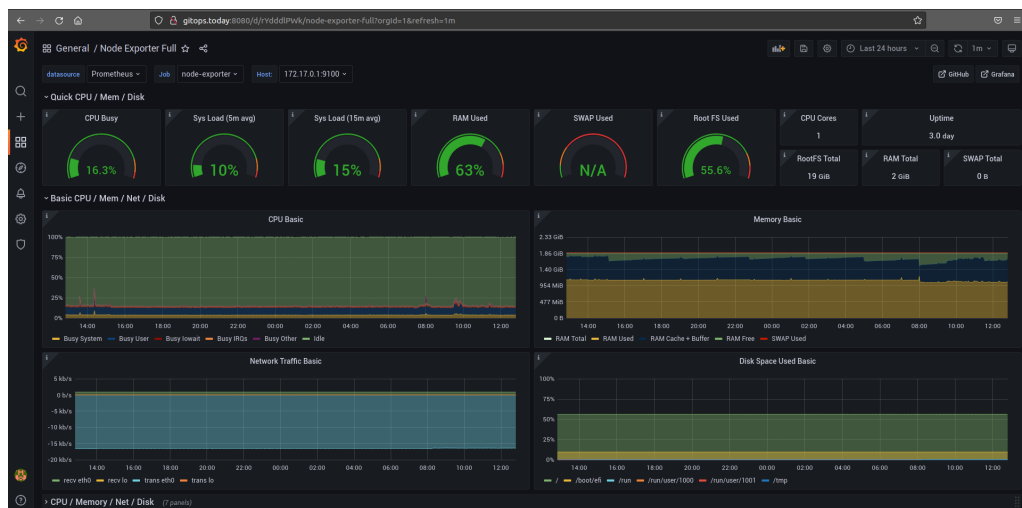


Figure 5.2 Node exporter 1.

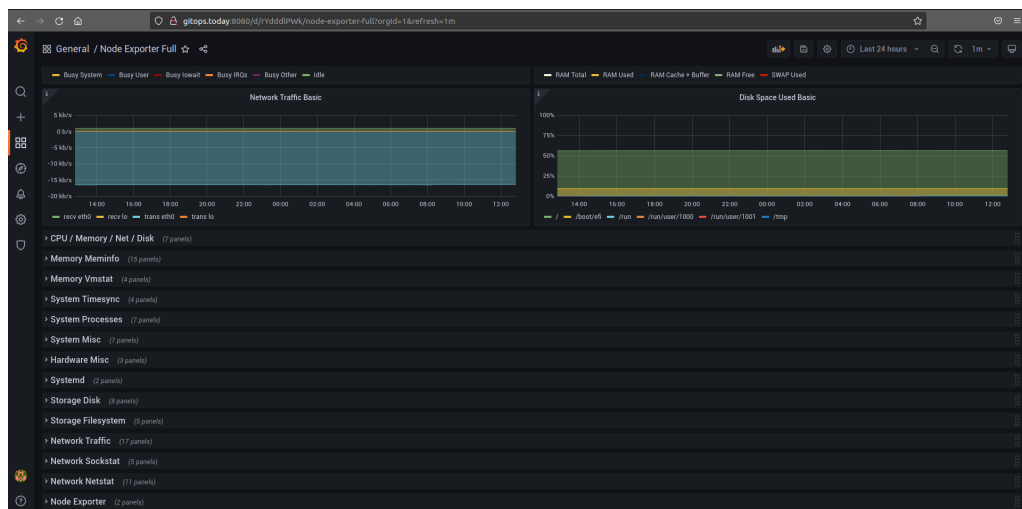


Figure 5.3 Node exporter 2.

5.3 Pushgateway

The Prometheus Pushgateway exists to allow ephemeral and batch jobs to expose their metrics to Prometheus. Since these kinds of jobs may not exist long enough to be scraped, they can instead push their metrics to a Pushgateway. The Pushgateway then exposes these metrics to Prometheus. So it can be considered an exporter.

6 Documentation

In this project we have use and integrate 2 systems to generate documentation::

- Static site generator, mkdocs
- Document converter, pandoc

GitLab Pipeline used has 2 stages:

- Gitlab pages stage: with 3 jobs to generate static site: resolve variables, deploy pages and stop environment.
- Nicedoc stage: with 1 job: nicedoc create PDF.

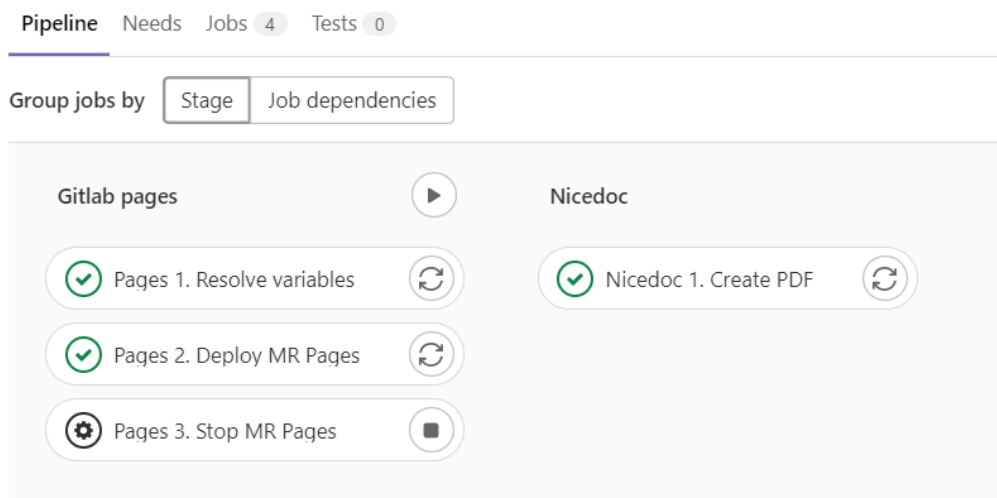


Figure 6.1 GitLab Pipeline used.

6.1 Static site generator: MkDocs

MkDocs is a fast, simple static site generator that's geared towards building project documentation. Documentation source files are written in Markdown, and configured with a single YAML configuration file. Mkdocs main features are:

- **Great themes available:** There's a stack of themes available for MkDocs, built in themes: mkdocs, readthedocs, third-party ones or build your own, the option chosen.
- **Easy to customize:** Possibility to customize not just the theme but installing some plugins and/or extension. Many configuration options are available.

- **Preview your site as you work:** The built-in dev-server allows to preview the documentation as it's been written it. It will even auto-reload and refresh your browser whenever you save your changes.
- **Host anywhere:** MkDocs builds completely static HTML sites that you can host on GitHub pages, Amazon S3, or GitLab pages in our case.

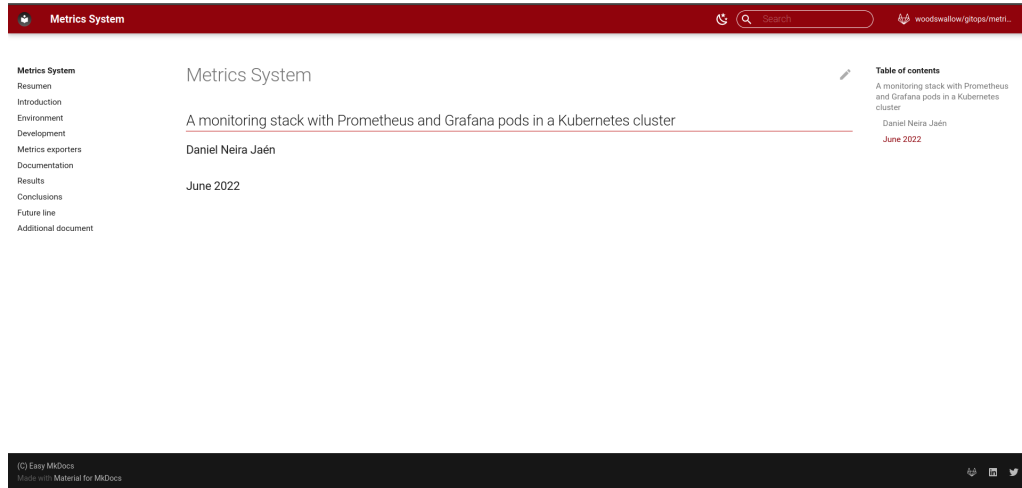


Figure 6.2 Home Pages MkDocs.

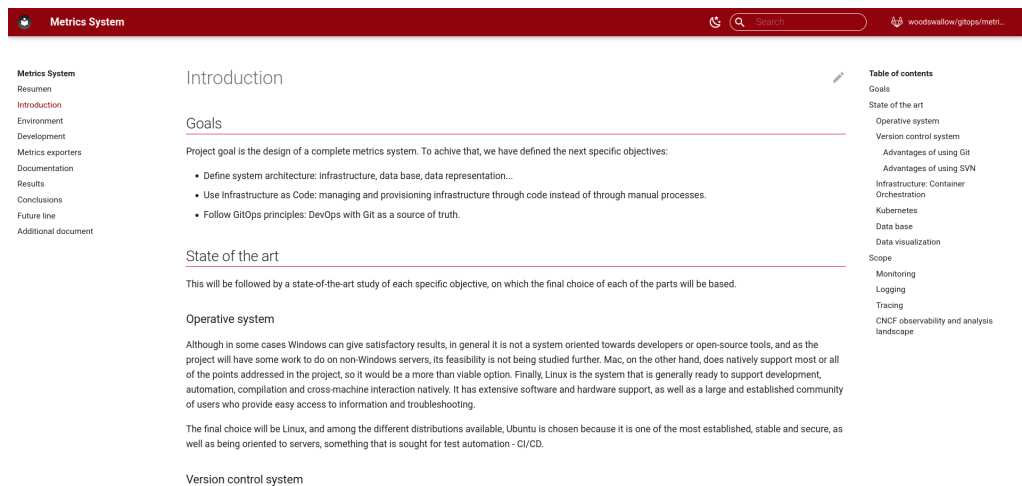


Figure 6.3 Introduction Pages MkDocs.

6.2 Document converter: Pandoc

Pandoc is a free and open-source document converter. It has a modular design: it consists of a set of readers, which parse text in a given format and produce a native representation of the document (an abstract syntax tree or AST), and a set of writers, which convert this native representation into a target format. Thus, adding an input or output format requires only adding a reader or writer. Custom pandoc filters are also supported to modify the intermediate AST.

This document has been generated with pandoc: source written in markdown, latex template and PDF output.

7 Results

7.1 IaC. Deployment

The files structure is:

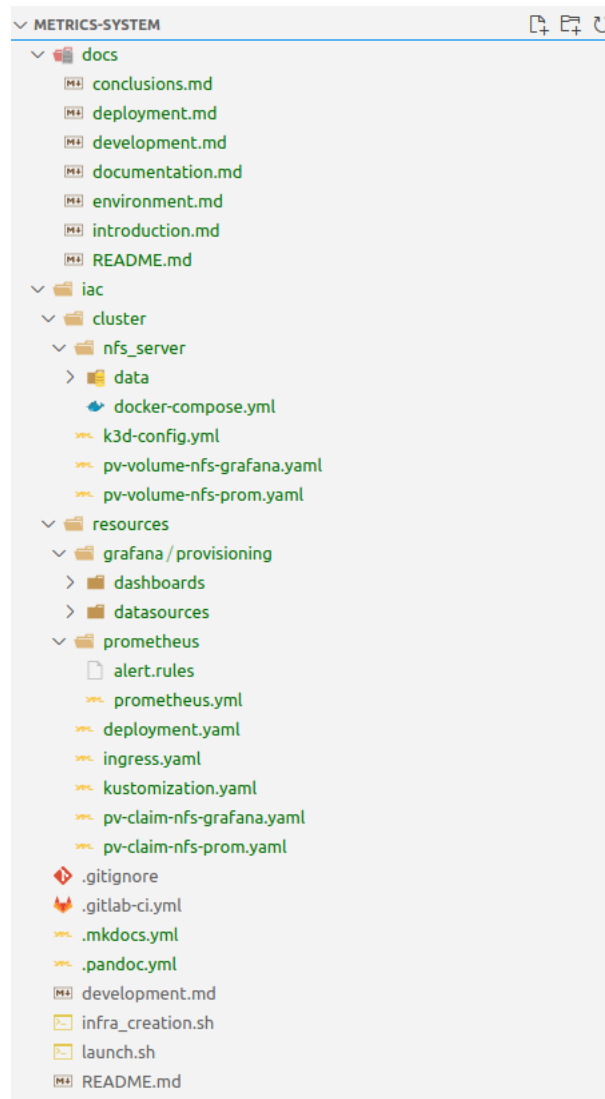


Figure 7.1 Files structure.

And we use the following scripts:

Infrastructure creation script. To start NFS server, create K8s cluster and apply persistent volumes. Also, with down option, we can stop NFS server and destroy K8s cluster.

Code 7.1 Infrastructure creation script.

```

1  #!/usr/bin/env bash
2
3  case "${1}" in
4    "up")
5      echo "Creating infra..."
6      echo "Starting NFS server ...";
7      docker-compose -f k8s_infra/nfs_server/docker-compose.yml up -d;
8      echo "Creating Kubernetes cluster using k3d ...";
9      k3d cluster create --config k8s_infra/k3d-config.yml;
10     kubectl apply -f k8s_infra/pv-volume-nfs-prom.yaml;
11     kubectl apply -f k8s_infra/pv-volume-nfs-grafana.yaml;
12     ;;
13   "down")
14     echo "Destroying Kubernetes cluster ...";
15     k3d cluster delete my-cluster;
16     echo "Stopping NFS server ...";
17     docker-compose -f k8s_infra/nfs_server/docker-compose.yml down;
18     ;;
19
20   "help")
21     echo "Usage: './infra_creation.sh options [arguments]' ";
22     echo "";
23     echo "Possible options are:";
24     echo "  up           Set environment up, creating cluster and nfs-server";
25     echo "  down        Delete cluster and stop nfs-server";
26     exit 0;
27     ;;
28
29   *)
30     echo "ERROR: Invalid option. Use 'help' subcommand for support";
31     exit 1;
32     ;;
33   esac

```

After infrastructure creation, we have the launch script, to deploy our app:

Code 7.2 Stack launch script.

```

1  #!/usr/bin/env bash
2
3  case "${1}" in
4    "up")
5      echo "Deploying app...";
6      kubectl apply -k k8s
7      ;;
8
9    "help")
10     echo "Usage: './launch.sh options [arguments]' ";
11     echo "";

```



```
12 echo "Possible options are:";
13 echo " up          Applying deployment";
14 exit 0;
15 ;;
16
17 *)
18 echo "ERROR: Invalid option. Use 'help' subcommand for support";
19 exit 1;
20 ;;
21 esac
```

We have this automate process through Gitlab CI/CD, with 2 jobs: a deploy job, to create and deploy stack, and a second job, stop, to delete cluster. This second action needs to manually push the button.

The screenshot displays a GitLab CI/CD pipeline interface. At the top, a green status bar indicates the pipeline is "passed" with ID #534414551, triggered 1 month ago by Daniel N. Below this, the pipeline is titled "Merge branch '1-create-poc-of-k8s-infra' into 'main'", with a description "Resolve 'Create PoC of K8s infra'", "Closes #1", and a link to "See merge request !1". A summary bar shows "2 jobs for main in 49 seconds (queued for 3 seconds)" and a commit hash "13e12d99". Below this, it states "No related merge requests found." The main section is titled "Pipeline" and shows "Needs", "Jobs 2", and "Tests 0". Under the "Deploy" section, two jobs are listed: "deploy_test" with a green checkmark and a play button, and "stop_deploy_test" with a gear icon and a stop button.

Figure 7.2 Pipeline deploy.

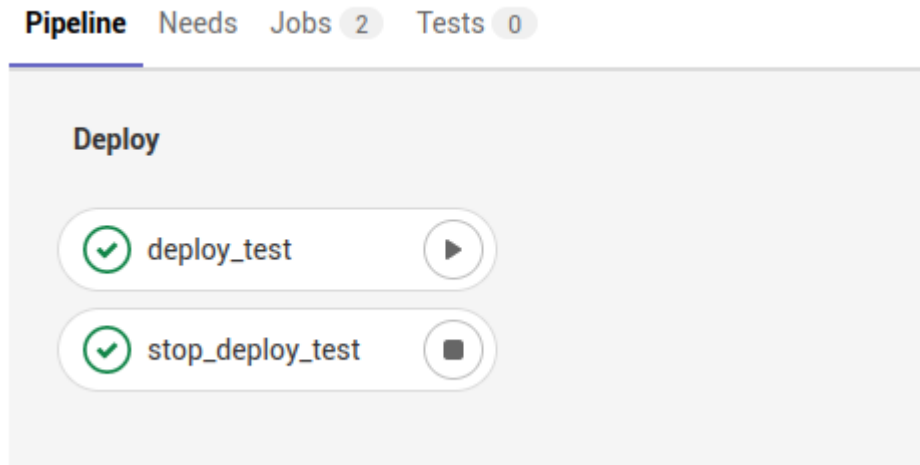


Figure 7.3 Pipeline stop.

7.2 Monitoring stack

Our project is available at <http://gitops.today:8080/>. After logging, we can see grafana available dashboards.

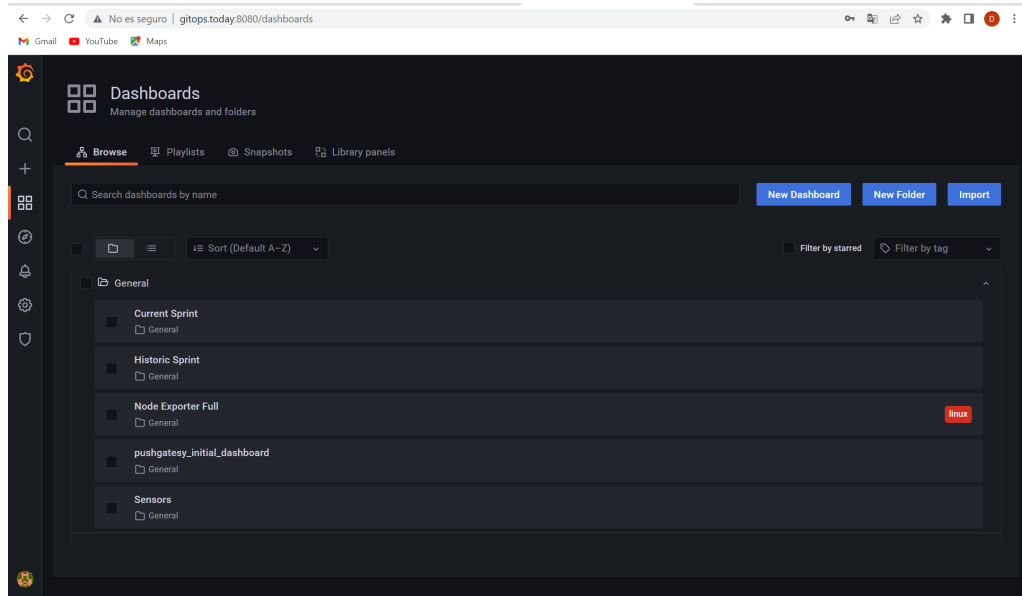


Figure 7.4 Grafana dashboards.

In node exporter dashboard, we can see server metrics, linke basic cpu, ram memory, disk:

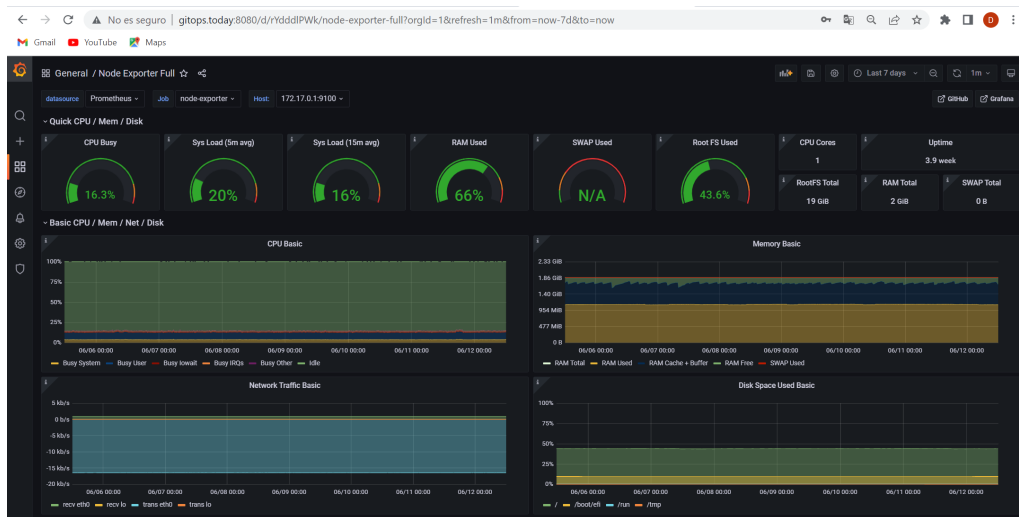


Figure 7.5 Node exporter dashboard.

Storage statistics:

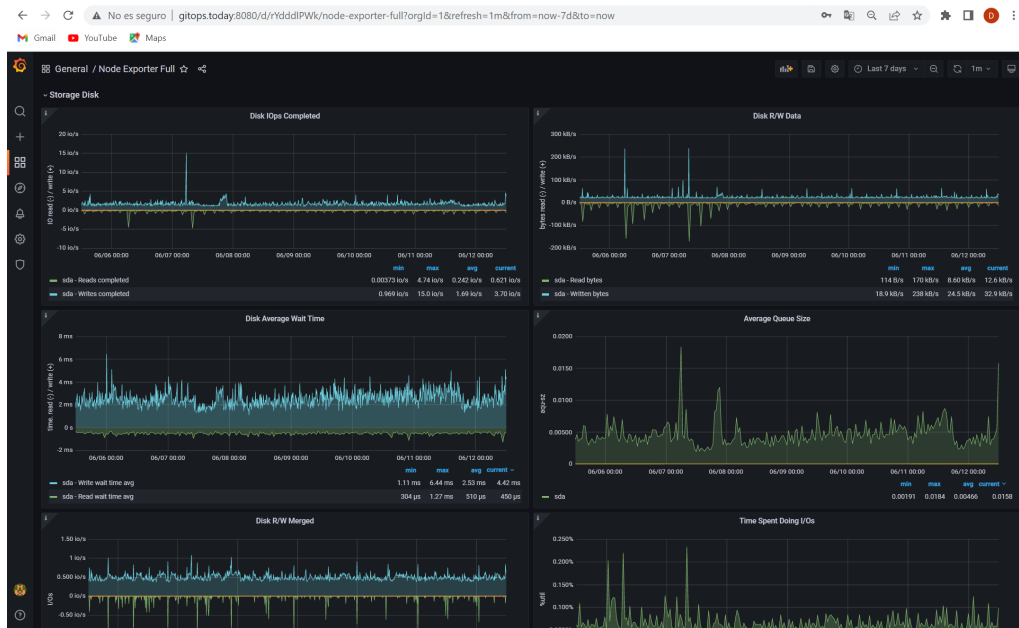


Figure 7.6 Storage statistics.

or Network traffic, among others:

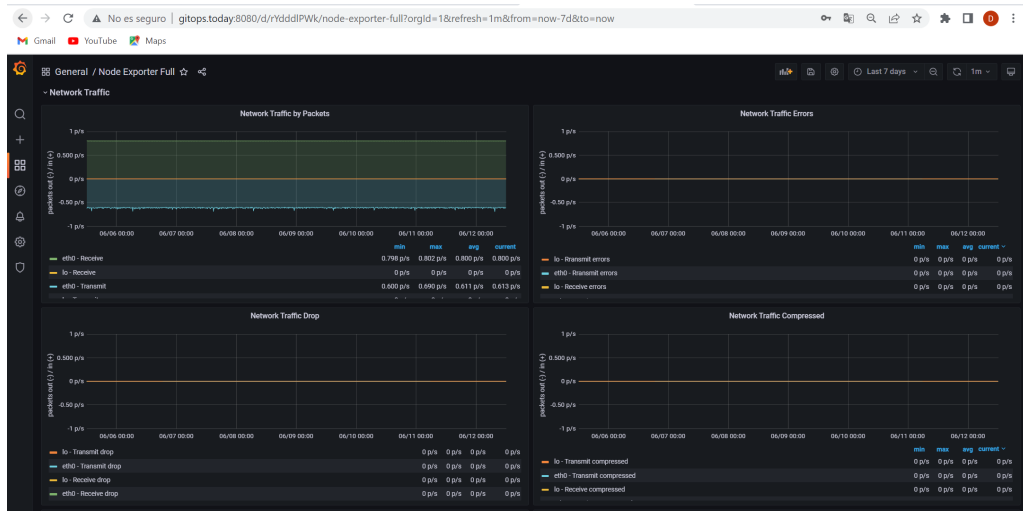


Figure 7.7 Network traffic.

8 Conclusions

In this project, we have developed a full metric stack with Prometheus and Grafana pods in a Kubernetes cluster. At the beginning we made an initial research, analysing different solutions and choosing ones that better fit in our needs. Then, we have described the environment used, mainly software, followed by the development process: studying and applying the different options and finding solutions to problems. Then, we have launch 2 different metrics exporters: lm-sensor and node-exporter, and verify analytics in our Grafana. Later, we have expose also 2 solutions used in the project to generate documentation: mkdocs as a static site generator with GitLab Pages and pandoc as a document convertor in order to generate this pdf file.

9 Future line

At the end of the project several options appear as future lines of development:

- **Config as Code.** Following GitOps principles, we have Prometheus and Grafana datasource configuration in files in a git repository. We should add also dashboards, each dashboard can be described in a json file and a global yaml config file specifying paths.
- **VPN:** Current stack is in the same machine. In the future, we should study the creation of a VPN (virtual private network) in order to spread components in different computers and separate function. Just like the line below, we need a VPN to separate storage in a secure way.
- **Storage:** Current storage in the stack is a NFS server in the same machine. So, if the machine gets corrupted, all data will be lost. We should use a storage server (NFS, S3, ...) in a different machine with backup mechanisms in order to being able to recover the information.

Appendix A

DNS

Development and test of DNS feature in a kubernetes cluster. Progress in this section has been stopped until an available VPN.

Doc:

- kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/
- kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/

After creating corefile.yaml, `kubectl apply -f k8s/corefile.yaml`:

```
Warning: resource configmaps/coredns is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
configmap/coredns configured
```

This is not about applying a resource, it's to modify current config inside cluster, so:

- `kubectl get -n kube-system configmaps coredns -o yaml > core_dns.yaml` to get config file.
- Modify it locally. (in this demo, add `consul.local`)
- `kubectl replace -n kube-system -f core_dns.yaml` to load changes.
- Probably restart is needed: `kubectl rollout restart -n kube-system deployment/coredns`

stackoverflow.com/questions/56675972/kubernetes-how-to-edit-coredns-corefile-configmap

At this point, we can test it with our `test_dns` pod. After applying it, `kubectl exec -i -t dnsutils -- nslookup kubernetes.default`

```
Server:      10.43.0.10
Address:     10.43.0.10#53

Name:   kubernetes.default.svc.cluster.local
Address: 10.43.0.1
```

Then, test it with `consul.local` which is our example, `kubectl exec -i -t dnsutils -- nslookup consul.local`

```
Server:      10.43.0.10
Address:     10.43.0.10#53
```

```
** server can't find consul.local: SERVFAIL
command terminated with exit code 1
```

We can see the log, `kubectl logs --namespace=kube-system -l k8s-app=kube-dns`

```
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:52483->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 consul.local. A: read udp
10.42.1.8:49286->10.150.0.1:53: i/o timeout
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:33201->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:34998->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:44998->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:37320->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:40139->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 3338554411985736187.5844331930135755221. HINFO: read
udp 10.42.1.8:33872->8.8.8.8:53: i/o timeout
[ERROR] plugin/errors: 2 consul.local. A: read udp
10.42.1.8:35436->10.150.0.1:53: i/o timeout
[ERROR] plugin/errors: 2 consul.local. A: read udp
10.42.1.8:56792->10.150.0.1:53: i/o timeout
```

It could be a namespace problem, so: `kubectl exec -i -t dnsutils -- nslookup consul.local.kube-system`

```
Server:      10.43.0.10
Address:     10.43.0.10#53

** server can't find consul.local.kube-system: NXDOMAIN
command terminated with exit code 1
```

NXDOMAIN means not existing domain www.dnsknowledge.com/whatis/nxdomain-non-existent-domain-2/

We can see also log:

```
[INFO] 10.42.0.5:42404 - 37940 "A IN gitlab.com.svc.cluster.local. udp 46 false
512" NXDOMAIN qr,aa,rd 139 0.000092792s
[INFO] 10.42.0.5:41040 - 17910 "AAAA IN gitlab.com.cluster.local. udp 42 false
512" NXDOMAIN qr,aa,rd 135 0.000042583s
[INFO] 10.42.0.5:41040 - 17770 "A IN gitlab.com.cluster.local. udp 42 false 512
" NXDOMAIN qr,aa,rd 135 0.0000373s
[INFO] 10.42.0.5:49108 - 43323 "A IN gitlab.com. udp 28 false 512" NOERROR qr,
aa,rd,ra 54 0.000051826s
[INFO] 10.42.0.5:49108 - 43493 "AAAA IN gitlab.com. udp 28 false 512" NOERROR
qr,aa,rd,ra 66 0.000074751s
[INFO] 10.42.0.5:33998 - 37917 "A IN gitlab.com. udp 28 false 512" NOERROR qr,
rd,ra 54 0.015352859s
```

```
[INFO] 10.42.1.6:60771 - 18112 "A IN consul.local.kube-system.default.svc.
cluster.local. udp 68 false 512" NXDOMAIN qr,aa,rd 161 0.000104845s
[INFO] 10.42.1.6:54217 - 31084 "A IN consul.local.kube-system.svc.cluster.local
. udp 60 false 512" NXDOMAIN qr,aa,rd 153 0.000072721s
[INFO] 10.42.1.6:50659 - 29174 "A IN consul.local.kube-system.cluster.local.
udp 56 false 512" NXDOMAIN qr,aa,rd 149 0.000051199s
[INFO] 10.42.1.6:57946 - 33875 "A IN consul.local.kube-system. udp 42 false 512
" NXDOMAIN qr,rd,ra 42 0.008439717s
```

Another attempt, following stackoverflow.com/questions/65283827/how-to-change-host-name-resolve-like-host-file-in-coredns

Adding to corefile:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      hosts custom.hosts myapi.local {
        192.168.49.2 myapi.local
        fallthrough
      }
      forward . 172.16.0.1
      cache 30
      loop
      reload
      loadbalance
    }
    consul.local:53 {
      errors
      cache 30
      forward . 10.150.0.1
    }
  }
```

But still not working:

```
kubectl exec -i -t dnsutils -- nslookup miapi.local
Server:          10.43.0.10
Address:         10.43.0.10#53

** server can't find miapi.local: SERVFAIL

command terminated with exit code 1
```

Test nfs pod manifest:

```
apiVersion: v1
kind: Pod
metadata:
  name: dnsutils
  namespace: default
spec:
  containers:
  - name: dnsutils
    image: gcr.io/kubernetes-e2e-test-images/dnsutils:1.3
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
```

List of Figures

1.1	Container Orchestration Comparison	3
1.2	Kubernetes usage, CNCF Annual Survey	4
1.3	CNCF Survey	6
1.4	CNCF Monitoring	7
1.5	CNCF Logging	8
1.6	CNCF Tracing	9
2.1	Deployments evolution	12
2.2	Containerized Application structure	14
2.3	Prometheus Architecture	16
3.1	Incremental counter metric	18
3.2	Grafana demo dashboard	19
3.3	Grafana dashboard detail	20
4.1	docker-compose command on remote	24
4.2	Grafana panel (1)	36
4.3	Grafana panel (2)	37
4.4	Grafana panel (3)	37
4.5	Ingress architecture	39
5.1	Im-sensor grafana	41
5.2	Node exporter 1	42
5.3	Node exporter 2	42
6.1	GitLab Pipeline used	45
6.2	Home Pages MkDocs	46
6.3	Introduction Pages MkDocs	46
7.1	Files structure	47
7.2	Pipeline deploy	49
7.3	Pipeline stop	50
7.4	Grafana dashboards	50
7.5	Node exporter dashboard	51
7.6	Storage statistics	51
7.7	Network traffic	52

List of Tables

1.1 Kubernetes distribution comparison

5

List of Codes

3.1	Prometheus config file	17
3.2	Prometheus docker run execution command	18
3.3	Grafana docker run execution command	18
3.4	Grafana datasource.yaml file	19
3.5	docker-compose.yml file	20
4.1	Manual deployment: copying files	23
4.2	Change DOCKER HOST environment variable	24
4.3	Docker context usage	24
4.4	Podman create pod command	25
4.5	Podman create container command	25
4.6	Podman create second container command	25
4.7	Check containers and pod command	25
4.8	Export as a pod manifest command	26
4.9	Use a Kubernetes-compatible Pod manifest to create and run a pod	26
4.10	Troubleshooting podman deployment	26
4.11	Init container to grant access	26
4.12	Kind config file	27
4.13	K3d config file	27
4.14	Troubleshooting launching pod manifest	28
4.15	kubectl describe pods command result	28
4.16	kubectl describe pods command result (2)	28
4.17	kubectl log command result	29
4.18	Init containers to give access to volumes	29
4.19	Set-up environment commands	29
4.20	Debug commands	30
4.21	NFS docker-compose.yml file	30
4.22	PV Claim yaml file	31
4.23	PV yaml file	31
4.24	Debugging volumes	31
4.25	Troubleshooting NFS	32
4.26	1 Volume and 2 usages problem	32
4.27	Initial kustomize apply command result	33
4.28	Kustomize yaml file	33
4.29	Debugging kustomization	33
4.30	K3d config yaml file	34
4.31	Debugging new k3d version	35
4.32	K3d and K3s versions	35
4.33	Set-up script	36
4.34	Set-up script (2)	36
4.35	Old set-up script	37

4.36	Old kustomization file	38
4.37	New set-up script	38
4.38	New kustomization file	38
4.39	Ingress yaml file	39
7.1	Infrastructure creation script	48
7.2	Stack launch script	48

Bibliography

- [1] Abi Tyas Tunggal, *LXC vs Docker: Why Docker is Better in 2022*.
- [2] Anca Iordache, *How to deploy on remote Docker hosts with docker-compose*.
- [3] Christopher Tozzi, *How to Choose the Right Kubernetes Distribution*.
- [4] CNCF Authors, *CNCF Landscape Guide*.
- [5] Docker Inc, *Use containers to Build, Share and Run your applications*.
- [6] Git Authors, *Git*.
- [7] GitLab Company, *The first single application for the entire DevOps lifecycle - GitLab | GitLab*.
- [8] Grafana Labs, *Why Grafana?*
- [9] k3d Authors, *k3d*.
- [10] MetricFire Corporation, *Prometheus vs. ELK*.
- [11] MkDocs Authors, *MkDocs*.
- [12] Pandoc Authors, *Pandoc User Guide*.
- [13] Podman Authors, *podman*.
- [14] Prometheus Authors, *What is Prometheus?*
- [15] Red Hat, Inc, *What is Infrastructure as Code (IaC)?*
- [16] Sabrina, *Git vs SVN: ¿Cuál es mejor? Ventajas y Desventajas - Guiadev*, apr 2019.
- [17] The HackerNoon Newsletter, *The 5 Best Kubernetes Alternatives*.
- [18] The Kubernetes Authors, *minikube start*.
- [19] Visual Studio Code Authors, *Visual Studio Code - Getting started*.
- [20] Wes Coffay, *Which Kubernetes distribution is right for you?*

Glossary

CI/CD Continuous Integration/Continuous Deployment. 1, 49

CNCF Cloud Native Computing Foundation. 3, 5–7

IoT Internet Of Things. 1