Fontys

6/5/2022

Monitoring

4Energy - Infiniot

Date	:	**05-06-2022**
Version	:	**v0.1**
State	:	finalized
Author	:	**S6-RB02-Group3**

VERSION HISTORY

Version	Date	Author(s)	Changes	State
v0.1	05-06-2022	Samir	Combined everything in one document	finalized

CONTENTS

CH 1. INTRODUCTION

This document is made to provide a clear understanding of the monitoring serviced added in our group project. The research is done by following the DOT-framework provided by Fontys. In Chapter two the Library method is used to figure out the current state of monitoring. In chapter 3 & 4 you can find how Prometheus & Grafana is setup using the Workshop-method prototyping.

CH 2. SYSTEM MONITORING

This chapter will explain what system monitoring is, the benefits and what tools/services are applicable/suited for the current project.

2.1 WHAT IS SYSTEM MONITORING?

An IT management's continuous monitoring of an infrastructure – or an IT system – is known as system monitoring. CPU, server memory, routers, switches, bandwidth, and applications, as well as the performance and availability of critical network components, are all monitored.

A monitoring service will collect metrics of data and output that in a dashboard (usually) in order for developers to be able to quickly scan issues. Applying monitoring in an Enterprise Software should be a must, monitoring will help out in organizing costs and efficiency when deploying such large software in the Cloud.

2.2 WHAT ARE THE BENEFITS OF MONITORING MY SYSTEM?

There are many benefits in what monitoring can do for your system. However, the main reasons for monitoring that I think currently is important are:

- Indicates performance issues.
- Shows Kubernetes cluster information
- Alerts when something goes wrong (
- Visualizing current status of pods and nodes.

CH 3. PROMETHEUS

In the previous chapter you can read what monitoring is and the amount of tools and services there are to apply in ones project. This chapter will explain why we choose this tool instead of others, and how to apply it in the group project.

3.1 WHY PROMETHEUS

When doing the library research, we came across a lot of advantage that Prometheus have over other monitoring services. The following list is a short example of such benefits:

1. Prometheus is TSDB.
2. Prometheus is Pull based tool.
3. Centralised control.
4. In built Alerting facility.
5. Easy for monitoring teams.
6. Data visualisation.
7. Service discovery (sd)
8. Scalability.

More can be read in detail on this website https://www.crybit.com/advantages-of-prometheus//

For us on the other hand, the main reasons where:

- It is open source, so free to use.
- Exists for quite a while, so enough documentation to learn from.
- Easy configuration with Grafana for visualization.

3.2 SETTING UP PROMETHEUS IN THE SERVICES?

As Prometheus can process a lot of variant data, we can setup logs in each microservice to provide monitoring data to the Prometheus Service. For each Controller-class/Api-endpoint we can apply counters or many more that Prometheus provides, in which Prometheus can get the metrics from.

See https://github.com/prometheus/client_java for explanation in how to implement this into the services. This is a great way to monitor what is happening in the services. However, as we are more interested into setting up Prometheus in the cloud to monitor our services, pods and the cluster in general, this chapter will be skipped for now.

3.3 SETTING UP PROMETHEUS IN LOCAL KUBERNETES CLUSTER.

In order to be able to setup Prometheus in the cluster the following is required:

- Kubernetes cluster running (local or in cloud).
- The yaml files provided in git.

There are multiple ways in setting up Prometheus in the cluster, the easiest however, will be using the yaml files. In each yaml file, the configuration for Prometheus is setup accordingly. The yaml files can for now be found in the folder k8s in Authentication-Service. Make sure you are in the **k8s** folder before executing the following steps.

3.3.1 QUICK SETUP

This sub-chapter is made to quickly setup Prometheus with Grafana. This way we do not have to go through each step every time after we learned what is happening in each step in the next chapters.

Execute the following commands to setup Prometheus and Grafana in your cluster under the namespace 'monitoring'.

```
$ kubectl create namespace monitoring
```

```
$ kubectl apply -f ./prometheus
```

If this is the first time you set up Prometheus/Grafana in your cluster, you do not have the community edition Dashboards implemented yet within Grafana (in volumes). To apply the community Dashboards, follow the steps from Chapter **4.1, step 4.**

3.3.2 GUIDED SETUP

Step 1
The first step is to create a new namespace in the cluster. This will help us to separate the monitoring completely from the other services, and only this namespace will have a clusterRole (see step 2) added for accessibility.

```
$ kubectl create namespace monitoring
```

Step 2
Prometheus reads all available metrics from Nodes, Pods, Deployments, and other Kubernetes components via Kubernetes APIs. As a result, it is needed to establish an RBAC policy that gives required API groups read access and connect it to the monitoring namespace. In the '*clusterrole.yaml*' file the get, list, and watch rights to nodes, services endpoints, pods, and ingresses are given in the monitoring namespace. If other objects are needed, you can add them in the yaml-file under resources.

Create the role by executing the following command

```
$ kubectl apply -f ./k8s/prometheus/clusterRole.yaml
```

Step 3
The settings (*'prometheus.yaml'*) and alerts (*'prometheus.rules'*) for Prometheus can be found in the config-yaml file. The file contains all of the configurations for dynamically discovering pods and services in a Kubernetes cluster. In our current Prometheus scrape configuration, we have the following scrape jobs.

1. `kubernetes-apiservers`: It gets all the metrics from the API servers.
2. `kubernetes-nodes`: It collects all the kubernetes node metrics.
3. `kubernetes-pods`: All the pod metrics get discovered if the pod metadata is annotated with `prometheus.io/scrape` and `prometheus.io/port` annotations.
4. `kubernetes-cadvisor`: Collects all cAdvisor metrics.
5. `kubernetes-service-endpoints`: All the Service endpoints are scrapped if the service metadata is annotated with prometheus.io/scrape and prometheus.io/port annotations. It can be used for black-box monitoring.

The rules contain all the alert rules for sending alerts to the Alertmanager. For this project, currently only one rule is applied to alert on high memory usage.

Create the config and rules by executing the following command

```
$ kubectl apply -f ./k8s/prometheus/config-map.yaml
```

Step 4
We want to be able to recollect our data when Prometheus crashes or restarts. This will have the history of the data, and losing this might be impactful. However, for us this will not be that important, but for best practice we will add PV-claim for Prometheus by executing the following command:

```
$ kubectl apply -f ./k8s/prometheus/prometheus-pvc.yaml
```

Step 5
As all the files needed for Kubernetes are added in the cluster. The only thing now left is to configure the deployment and service files. The service will be used to access Prometheus. Execute the following commands to add the deployment and service of Prometheus.

```
$ kubectl apply -f ./k8s/prometheus/prometheus-deployment.yaml
```

```
$ kubectl apply -f ./k8s/prometheus/prometheus-pvc.yaml
```

Accessing Prometheus
In order to access Prometheus we need to port-forward the service. Follow **Step 4** of **Ch 4** for Prometheus instead of Grafana.

CH 4. GRAFANA

Grafana is an open-source lightweight dashboard tool that can be used to easily monitor your application. In this chapter we will integrate the Grafana Dashboard with Prometheus (previous Chapter). The benefits for working with Grafana instead of the Prometheus Dashboard are:

- No need the learn new language PromQl.
- Many community dashboards available for Grafana, which can be modified to own liking.

4.1 INTEGRATE GRAFANA WITH PROMETHEUS IN CLUSTER.

Step 1
In order for Grafana to visualize data, it needs to get its data from somewhere. The Configmap of Grafana makes sure it can access Prometheus to get data for visualization. See the yaml files for more information in where to add the configuration. Execute the following command to apply the configuration.

```
$ kubectl apply -f ./k8s/prometheus/grafana-datasource-config.yaml
```

Step 2
Just like with Prometheus, we want to be able to recollect our data when the Grafana pod crashes or restarts. This way we do not have to setup the dashboard every time from scratch. Add the PV-claim by executing the following command:

```
$ kubectl apply -f ./k8s/prometheus/grafana-pvc.yaml
```

Step 3
Just like Prometheus, we deploy Grafana by using the latest image of DockerHub and in order to be able to access Grafana, we add a Service as well. Execute the following commands to add the Deployment and Service for Grafana

```
$ kubectl apply -f ./k8s/prometheus/grafana-deployment.yaml
```

```
$ kubectl apply -f ./k8s/prometheus/grafana-service.yaml
```

Step 4 (only local)
After everything runs correctly in the cluster, the easiest way to access Grafana is by port forwarding. This can be achieved by first getting your Grafana pod name with the following command

```
$ kubectl get pods -n monitoring
```

The result should be something like the image below.

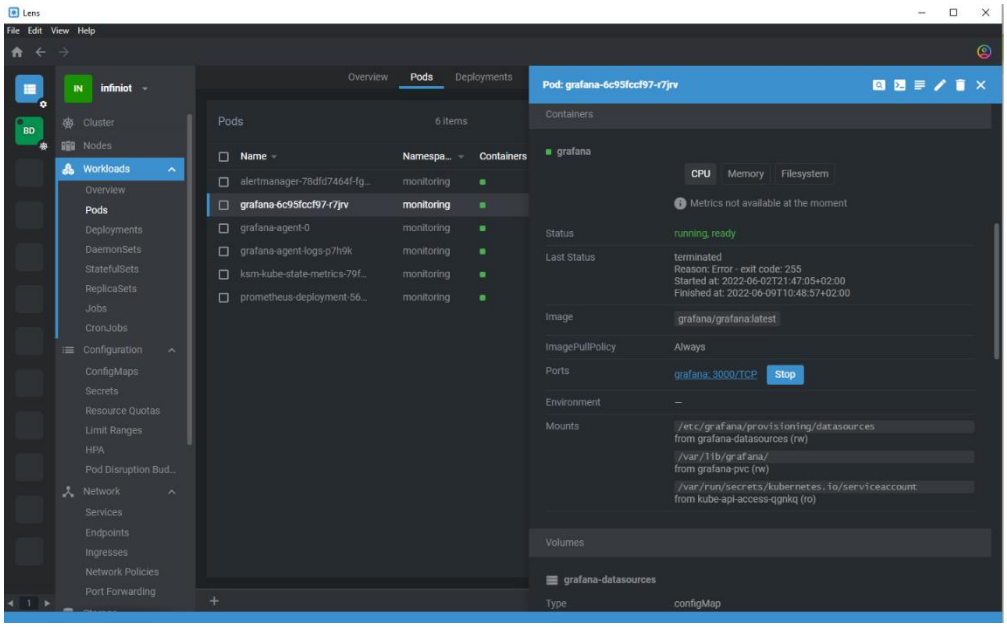
```
PS C:\Windows\system32> kubectl get pods -n monitoring
NAME                                  READY   STATUS    RESTARTS   AGE
alertmanager-78dfd7464f-fg8x6         1/1     Running   1          6d12h
grafana-6c95fccf97-r7jrv              1/1     Running   1          6d13h
grafana-agent-0                       1/1     Running   1          6d12h
grafana-agent-logs-p7h9k              1/1     Running   1          6d12h
ksm-kube-state-metrics-79f9d45c88-8gxp5   1/1     Running   2          6d12h
prometheus-deployment-566479cc8f-8x44c    1/1     Running   1          6d13h
```

Execute the following command to port-forward, in my case I would replace '<grafana-pod-name>' with 'grafana-6c95fccf97-r7jrv'. Make sure you do not exit the shell you are using to port-forward, else it will stop.

```
$ kubectl port-forward -n monitoring <grafana-pod-name> 3000
```

If you are making use of Lens (Kubernetes IDE), then you can simply port forward by:

1. Connect to your cluster in Lens.
2. Go to your pods (make sure your namespace is set to All or Monitoring).
3. Click on your pod (grafana-6c95fccf97-r7jrv') and scroll down a bit on the right side.
4. Click on the link to automatically port forward or, click on start to set port yourself (see image here below).



Access Grafana by visiting your browser on http://localhost:3000 and login with the credentials username: admin, password: admin. Change your password to your liking (local).

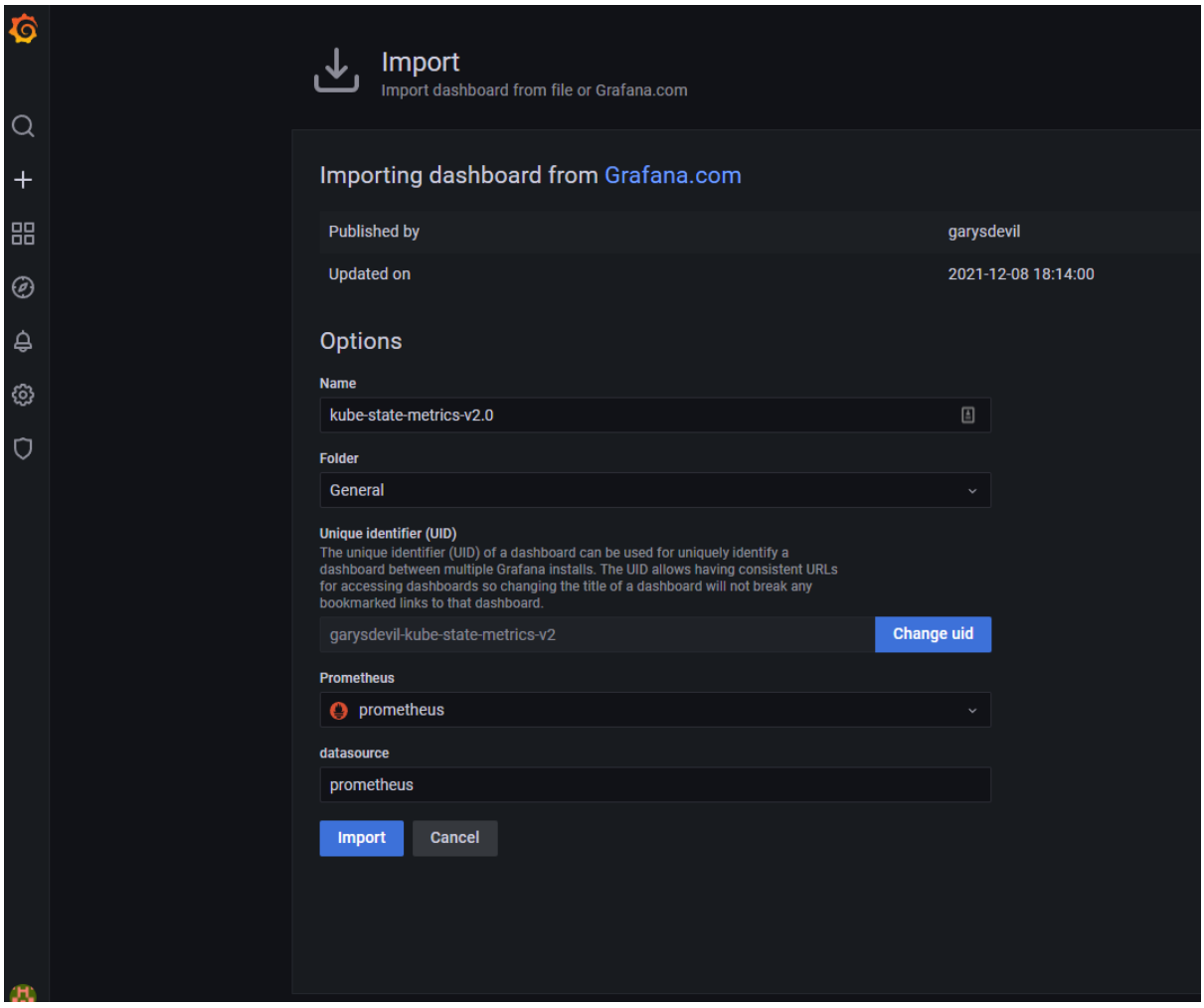
4.2 VISUALIZE DATA IN GRAFANA

There are many ways to visualize data in Grafana. Grafana works with queries and the first thing you can see is to add new visualization. Here you can import the data of Prometheus and play around with the data. However, for us this will not be feasible to invest so much time in learning the complete visualization queries of Grafana. Instead, to log the data of the Kubernetes cluster, we can make use of community Dashboards, that are already premade by other users of Grafana.

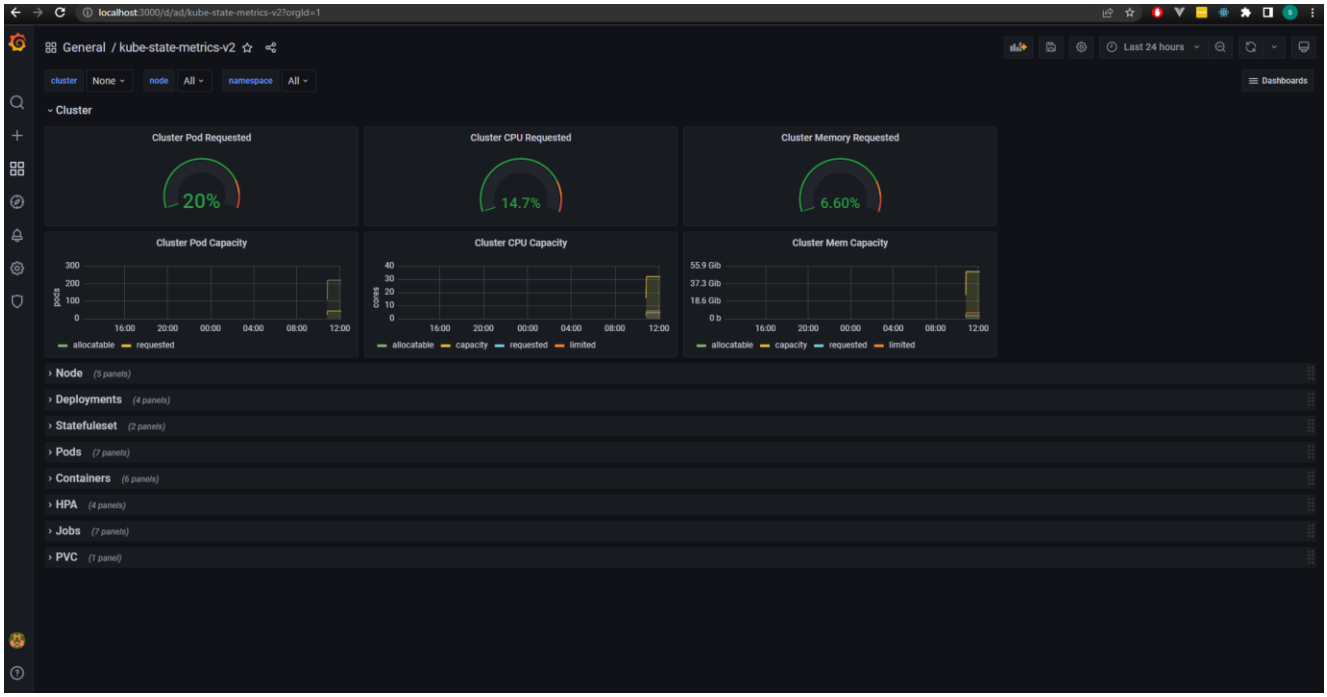
4.2.1 COMMUNITY DASHBOARDS

Visit the following site https://grafana.com/grafana/dashboards/?search=kubernetes to see what kind of dashboards already exist for Kubernetes. You can check for yourself the templates you might want to use.

However, for this guide, we added the kube-state-metrics in our Prometheus, which we can also add in our Grafana for visualization. Click on the + sign left in the dashboard and click on import. Add this code 13332 and click on load (search in the link for kube-state-metrics and get the new code if this one is outdated). Before clicking on import, make sure your datasource is the same as the image down below.



After importing the template, you should see the same as the image here below. This will output everything that is going on currently in your cluster, as you would see in Lens or other Kubernetes IDE.



CH 5. SETTING UP PROMETHEUS & GRAFANA IN THE CLOUD.

In the Cloud services the steps are basically the same. Each yaml file needs to be created in the cluster. However, there are two steps that are different since we are using Google Cloud Engine (GKE) as our Cloud-provider.

GKE admin
Since we are running the cluster in GKE, we need to run the following commands as we need privileges to create cluster roles for this Prometheus setup.

```
$ ACCOUNT=$(gcloud info --format='value(config.account)') kubectl create clusterrolebinding owner-cluster-admin-binding \ --clusterrole cluster-admin \ --user $ACCOUNT
```

Load-balancer.
The second part is that our services will not be accessible over localhost anymore. However, the services are of Type Load-balancer, which GKE will recognize and assign an IP-address to that Service, in which it will be accessible through that IP.

A guided explanation will be added in the future once Prometheus & Grafana run in the GKE-environment.

CONCLUSION

Having a monitoring system is a must for an Enterprise Software project, regardless of the choice for the system. This will benefit the project in multiple ways, but the most important ones for us are, cost and performance efficiency.

Adding these monitoring systems is not difficult, especially with Grafana having community dashboards which everyone can use. However, if you want to monitor properly and with cause, some more research needs to be done in how to visualize every data of Prometheus or other data sources. Another solution will be to purchase the monthly subscription of Grafana, which already has most of the stuff needed to visualize everything.

SOURCES

https://en.wikipedia.org/wiki/System_monitor

https://www.crybit.com/advantages-of-prometheus/

https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/

https://devopscube.com/setup-grafana-kubernetes/