Experiment: Prometheus Operator on k3d

In this experiment, we will deploy Prometheus and configure it in Grafana.

Create a cluster, for the experiment

```
k3d cluster create prometheus-demo --api-port 6550 --agents 2
```

It is a good practice to run your Prometheus containers in a separate namespace, so let's create one:

```
kubectl create ns monitor
```

Install Helm

For MacOS

$ brew install helm

For Windows

$ choco install kubernetes-helm

For MacOS and Windows

First we'll update the helm repo for the Prometheus operator from the Google Kubernetes Chart repository

```
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com
```

"stable" has been added to your repositories

```
c:\helm\helm repo update
```

Hang tight while we grab the latest from your chart repositories...

...Successfully got an update from the "ingress-nginx" chart repository

...Successfully got an update from the "kubernetes-dashboard" chart repository

...Successfully got an update from the "jetstack" chart repository

...Successfully got an update from the "rancher-latest" chart repository

...Successfully got an update from the "banzaicloud-stable" chart repository

...Successfully got an update from the "stable" chart repository

Update Complete. ⎈Happy Helming!⎈

Then, proceed with the installation of the Prometheus operator:

```
helm install prometheus-operator stable/prometheus-operator --namespace monitor
```

- “prometheus-operator” is the name of the release. You can change this if you want.
- “stable/prometheus-operator” is the name of the chart.
- “monitor” is the name of the namespace where we are going to deploy the operator.

You can verify your installation using:

```
kubectl get pods -n monitor
```

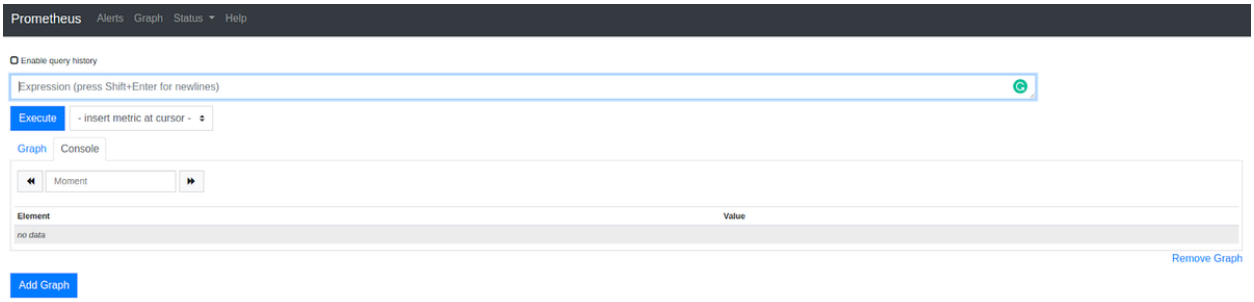
You should be able to see the Prometheus Alertmanager, Grafana, kube-state-metrics pod, Prometheus node exporters and Prometheus pods.

```
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-operator-alertmanager-0          2/2     Running   0          49s
prometheus-operator-grafana-5bd6cbc556-w9lds             2/2     Running   0          59s
prometheus-operator-kube-state-metrics-746dc6ccc-gk2p8   1/1     Running   0          59s
prometheus-operator-operator-7d69d686f6-wpjtd            2/2     Running   0          59s
prometheus-operator-prometheus-node-exporter-4nwbf       1/1     Running   0          59s
prometheus-operator-prometheus-node-exporter-jrw69       1/1     Running   0          59s
prometheus-operator-prometheus-node-exporter-rnqfc       1/1     Running   0          60s
prometheus-prometheus-operator-prometheus-0              3/3     Running   1          39s
```

Now that our pods are running, we have the option to use the Prometheus dashboard right from our local machine. This is done by using the following command:

```
kubectl port-forward -n monitor prometheus-prometheus-operator-prometheus-0 9090
```

Now visit http://127.0.0.1:9090 to access the Prometheus dashboard.



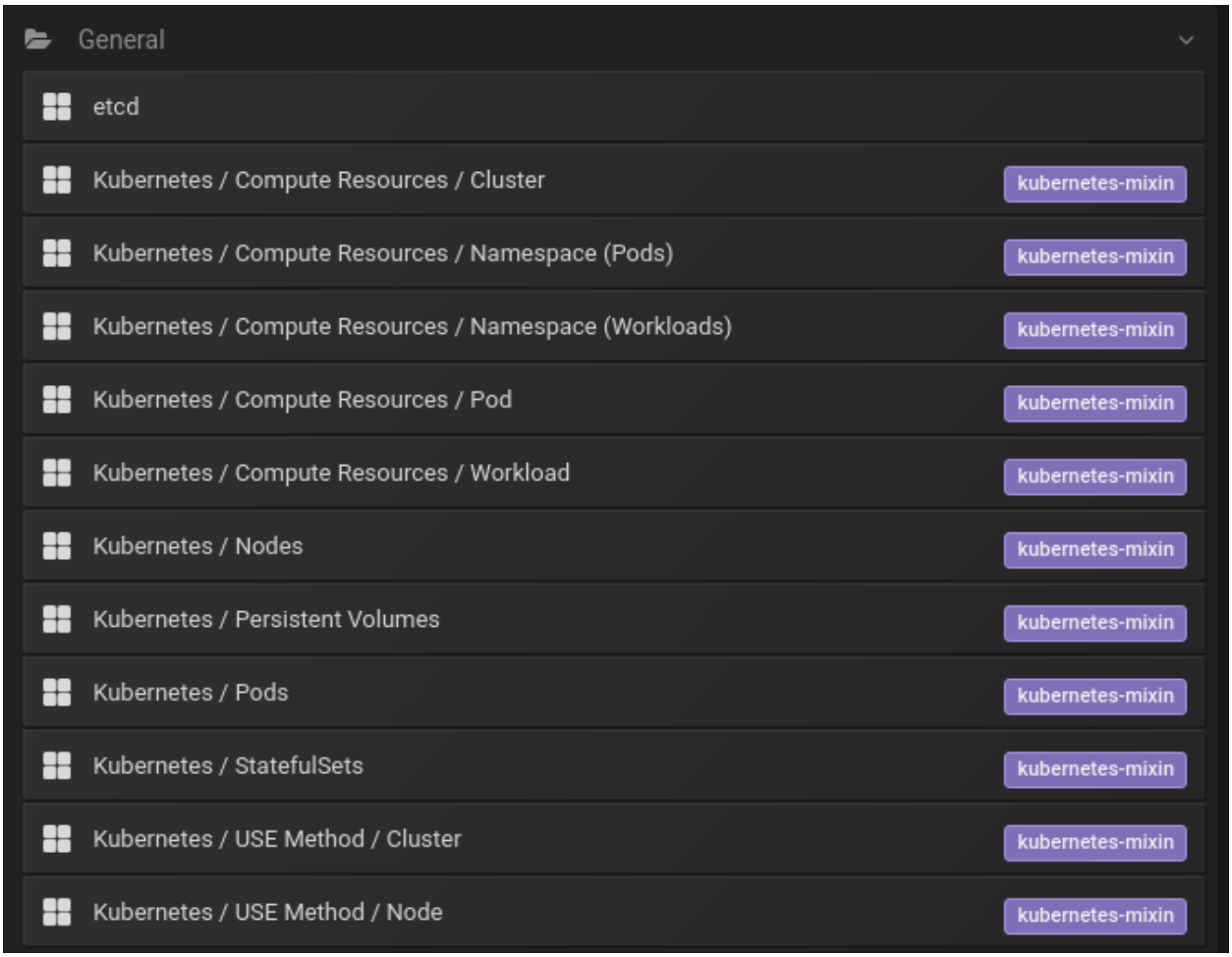
Same as Prometheus, we can use this command to make the Grafana dashboard accessible:

```
kubectl port-forward $(kubectl get pods --selector=app=grafana -n monitor --output=jsonpath="{.items..metadata.name}") -n monitor 3000
```

or if that fails

```
kubectl kubectl get pods -o wide | grep Grafana
kubectl port-forward prometheus-operator-grafana-88648f94b-5wfp9 -n monitor 3000
```

After visiting http://127.0.0.1:3000 you will be able to discover that there are some default preconfigured dashboards:



Note that you should use "admin" as the login and "prom-operator" as the password. Both can be found in a Kubernetes Secret object:

```
kubectl get secret --namespace monitor grafana-credentials -o yaml
```

You should get a YAML description of the encoded login and password:

```
apiVersion: v1 data: password: cHJvbS1vcGVyYXRvcgo= user: YWRtaW4=
```

You need to decode the username and the password using:

```
echo "YWRtaW4=" | base64 --decode echo "cHJvbS1vcGVyYXRvcgo=" | base64 --decode
```

Using the "base64 --decode", you will be able to see the clear credentials. Should be prom-operator

Preconfigured Grafana dashboards for Kubernetes

By default, Prometheus operator ships with a preconfigured Grafana - some dashboards are available by default, like the one below:

Some of these default dashboards, are "Kubernetes / Nodes", "Kubernetes / Pods", "Kubernetes / Compute Resources / Cluster", "Kubernetes / Networking / Namespace (Pods)" and "Kubernetes / Networking / Namespace (Workload)", etc.

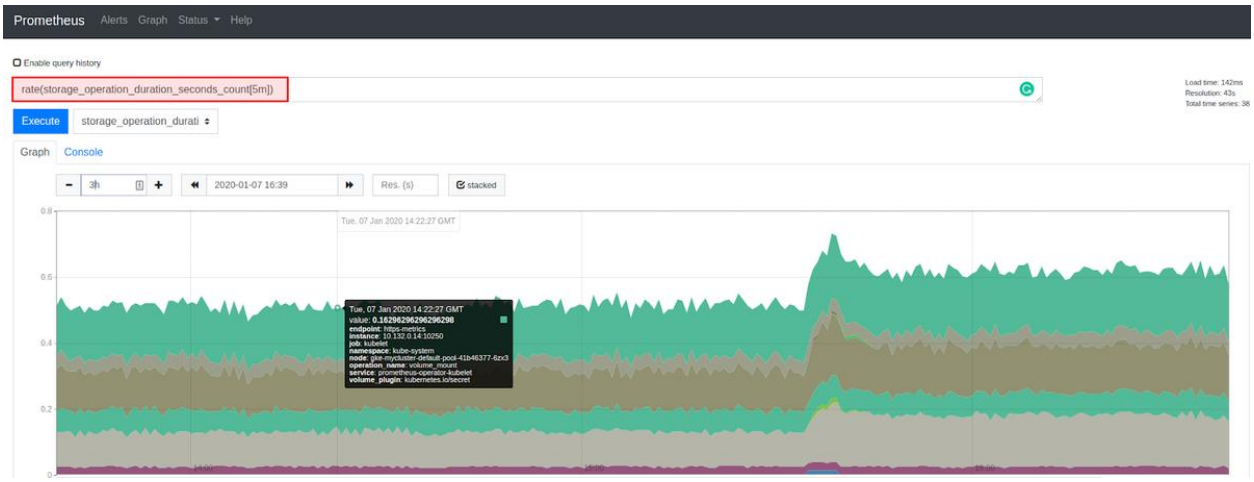
The metrics that are available as a result of the Helm install

To understand how Prometheus works, let's access the Prometheus dashboard. Use this port forwarding command:

```
kubectl port-forward -n monitor prometheus-prometheus-operator-prometheus-0 9090
```

Then visit "http://127.0.0.1:9090/metrics".
You will be able to see a long list of metrics.

Prometheus uses PromQL (Prometheus Query Language), a functional query language, that lets the user select and aggregate time series data in real time. PromQL can be complicated especially if you start to comprehensively learn it, but you can start with these examples from the official documentation and you will be able to understand a good part of it. For more information about PromQL, check out our great article that illustrates 10 key examples of PromQL.



When we deployed Prometheus using Helm, we used this chart, and it actually deployed not just Prometheus but also:

- prometheus-operator
- prometheus
- alertmanager
- node-exporter
- kube-state-metrics

- grafana
- service monitors to scrape internal kubernetes components
- kube-apiserver
- kube-scheduler
- kube-controller-manager
- etcd
- kube-dns/coredns
- kube-proxy

So in addition to Prometheus, we included tools like the service monitors that scrape internal system metrics and other tools like "kube-state-metrics".

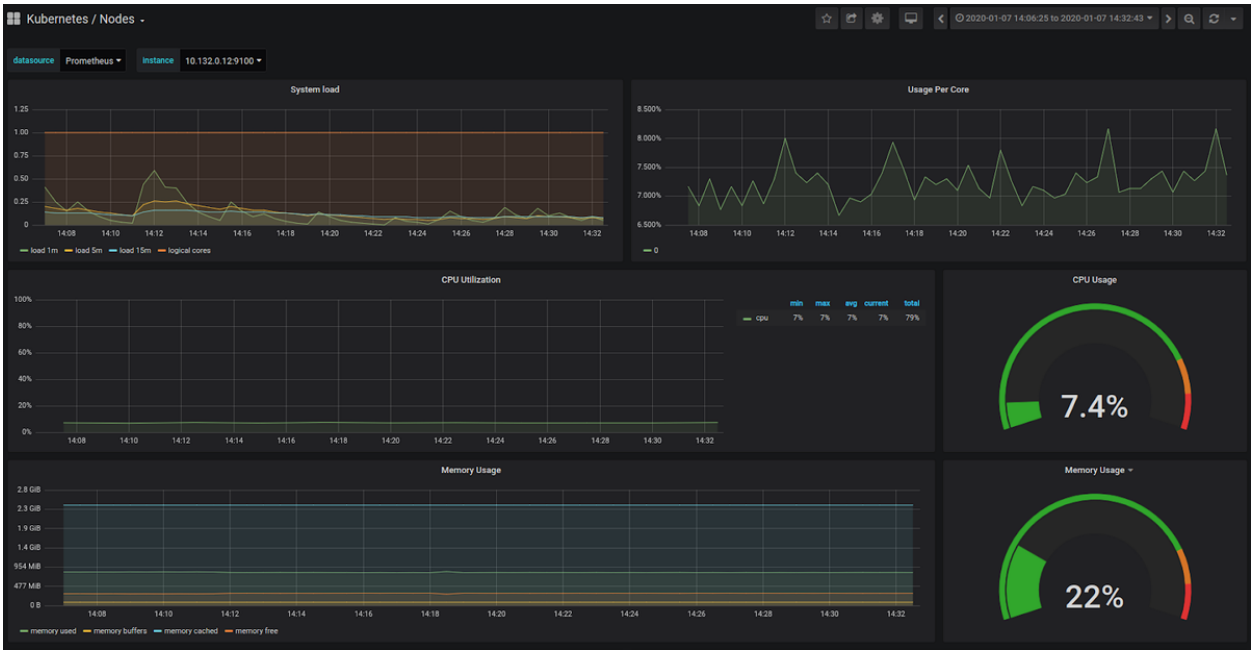
kube-state-metrics will also export information that Prometheus server can read. We can see a list of these metrics by visiting "http://127.0.0.1:8080/metrics" after running:

```
kubectl port-forward -n monitor prometheus-operator-kube-state-metrics-xxxx-xxx 8080
```

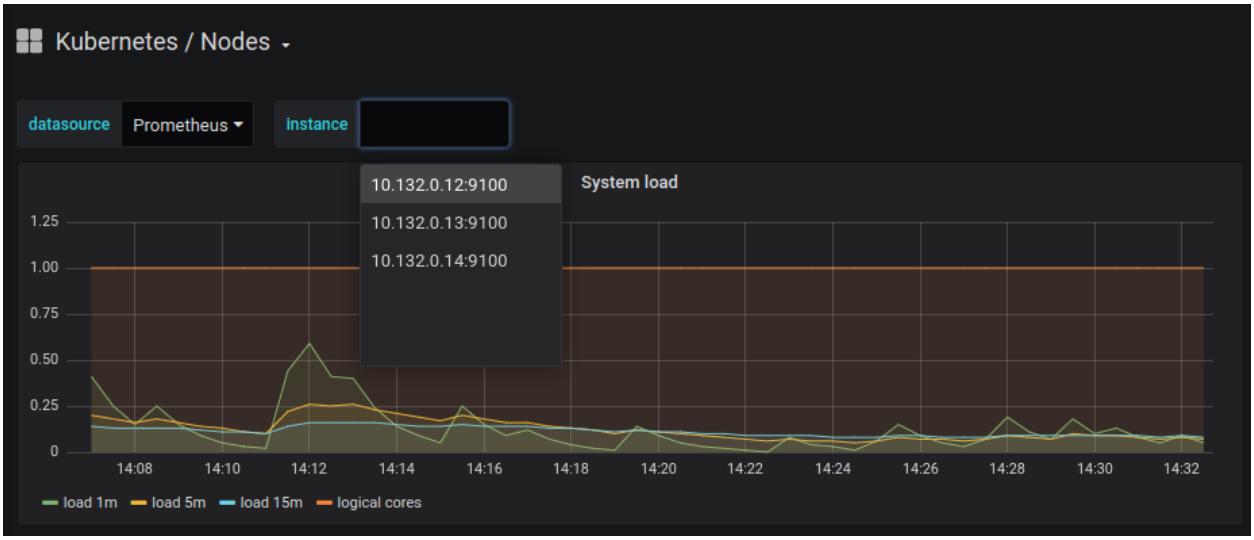
Important metrics to watch in production

Now that we have Prometheus set up as a Datasource for Grafana - what metrics should we watch, and how do we watch them?

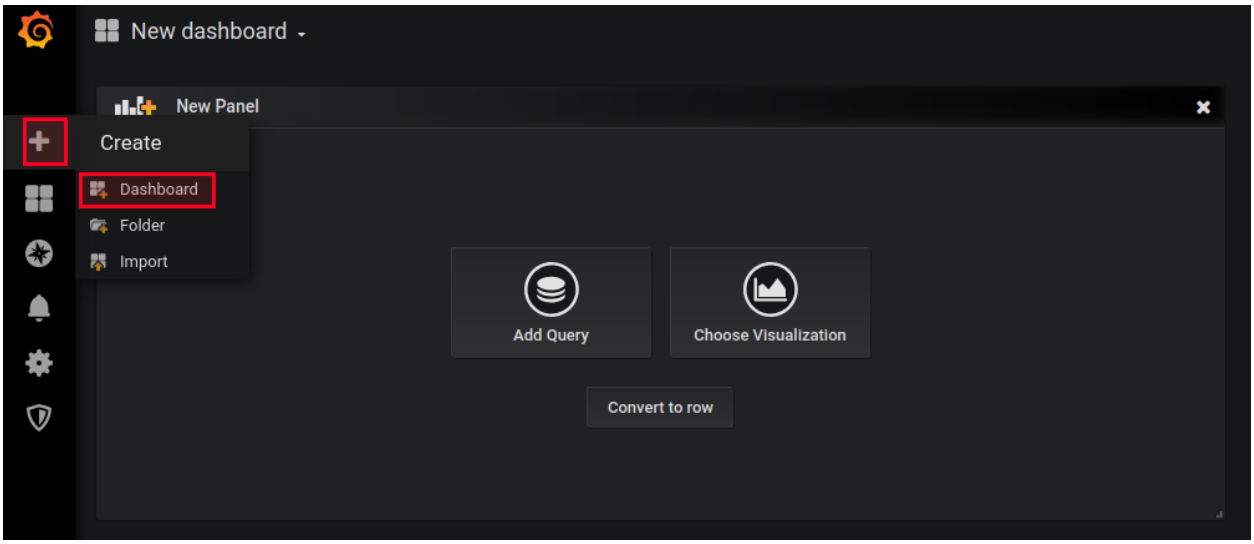
There is a large selection of default dashboards available in Grafana. Default dashboard names are self-explanatory, so if you want to see metrics about your cluster nodes, you should use "Kubernetes / Nodes". The dashboard below is a default dashboard:



Unless you are using k3d or one of its alternatives, a Kubernetes cluster usually runs more than one node. You must make sure that you are monitoring all of your nodes by selecting them one at a time:

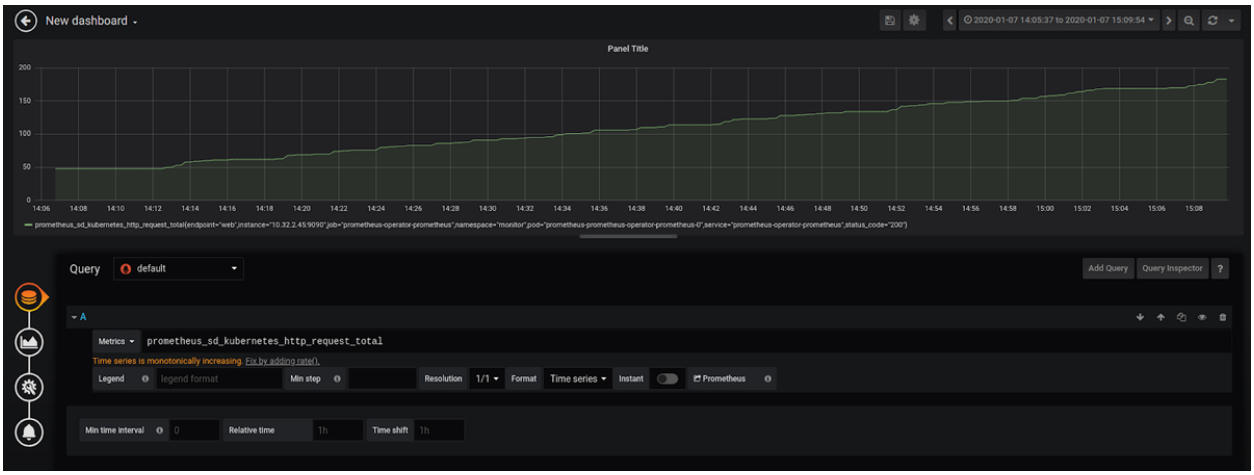


It is possible to add your own dashboards using a similar ConfigMap manifest, or directly using the Grafana dashboard interface. You can see below the “create dashboard” UI:



When creating a new custom dashboard, you will be asked whether you want to add a query or use visualization. If you choose visualization, you will be asked to choose a type of graph and a datasource. We can choose Prometheus as our datasource. We can also choose to add a query using Prometheus as a datasource.

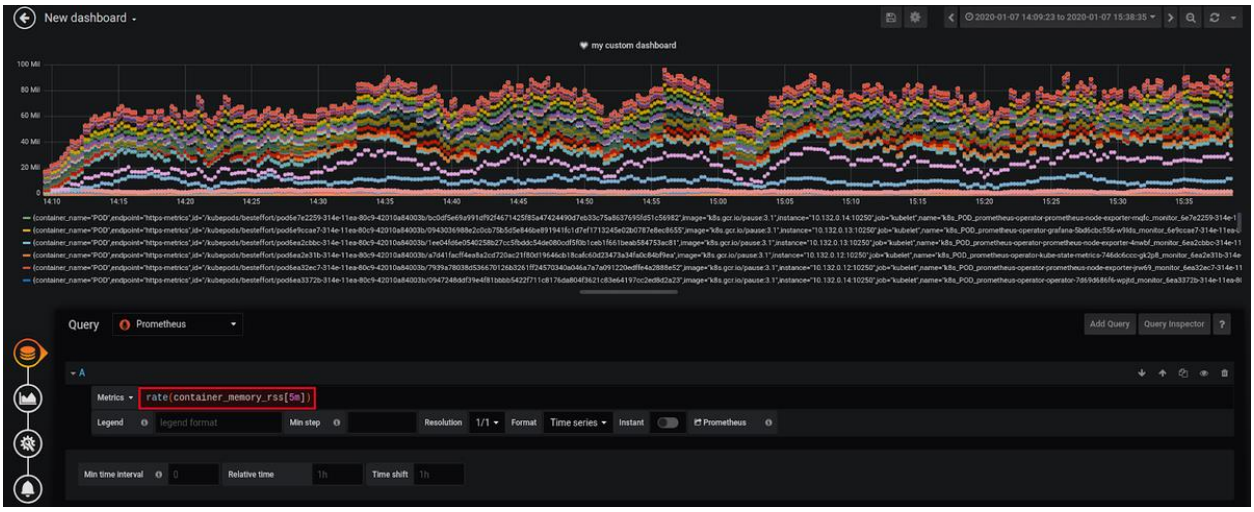
As an example, we can set up a graph to watch the prometheus_sd_kubernetes_http_request_total metric, which gives us information about the number of HTTP requests to the Kubernetes API by endpoint.



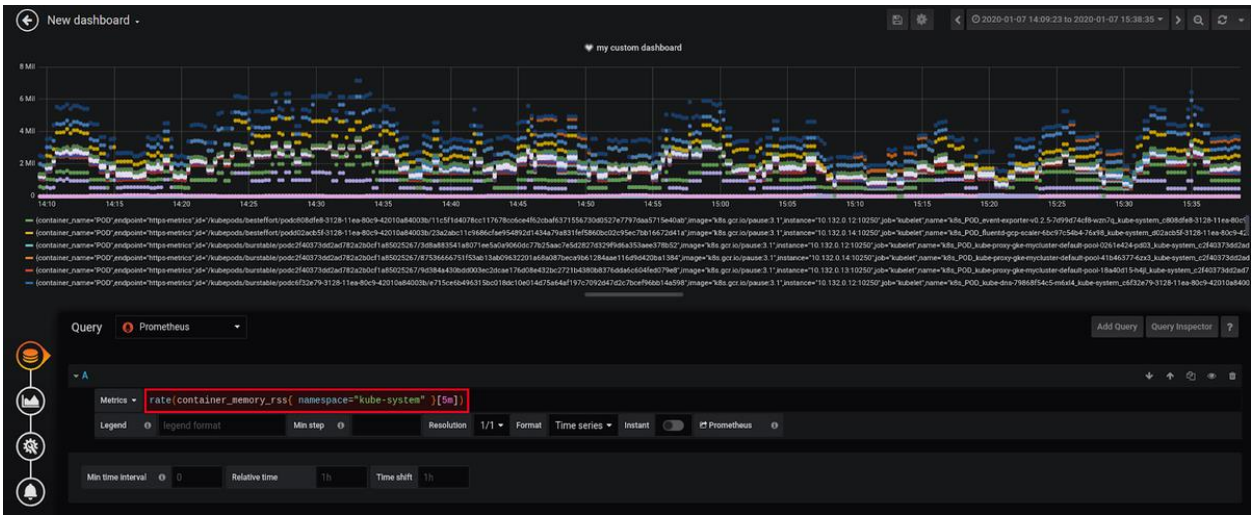
Since this metric is cumulative, Grafana asked us if we would rather use the function rate(), which gives us better information about our metric.

After setting up the name of the visualization, you can set alerts. Let's say that the average of the metric we chose should not exceed "0.06":

Let's choose a more significant metric, like the Resident Set Size (RSS) of Kubernetes system containers (RSS is the portion of memory occupied by a process that is held in main memory (RAM)). See the dashboard below:



Sometimes, some views are overpopulated, and in our example, since we want to monitor the system containers, we can refine our dashboard by filtering by namespace:



Let's try a third important metric to watch, like the total number of restarts of our container. You can access this information using kube_pod_container_status_restarts_total or kube_pod_container_status_restarts_total{namespace="<namespace>"} for a specific namespace.

There are many other important metrics to watch in production. Some of them are common and related to Nodes, Pods, Kube API, CPU, memory, storage and resources usage in general. These are some examples:

- kube_node_status_capacity_cpu_cores : describes nodes CPU capacity
- kube_node_status_capacity_memory_bytes : describes nodes memory capacity
- kubelet_running_container_count : returns the number of currently running containers
- cloudprovider_*_api_request_errors : returns the cloud provider API request errors (GKE, AKS, EKS, etc.)
- cloudprovider_*_api_request_duration_seconds: returns the request duration of the cloud provider API call in seconds

In addition to the default metrics, our Prometheus instance is scraping data from kube-apiserver, kube-scheduler, kube-controller-manager, etcd, kube-dns/coredns, and kube-proxy, as well as all the metrics exported by "kube-state-metrics". The amount of data we can get is huge, and that's not necessarily a good thing.

The list is long but the important metrics can vary from one context to another, you may consider that Etcd metrics are not important for your production environment while it can be for someone else.

However, some abstractions and observability good practices like Google Golden Signals or the USE method, can help us to choose what metrics we should watch.

$ **k3d cluster delete prometheus-demo**