# Prometheus Bundle monitoring on EUXDAT infrastructure

Contents

[Prometheus Bundle monitoring on EUXDAT infrastructure 1](#_Toc20381722)

[First part: Prometheus on Kubernetes 2](#_Toc20381723)

[1. Create the config map: 2](#_Toc20381724)

[2. Create the deployment, service and ingress: 2](#_Toc20381725)

[3. Add the applications as targets to monitor: 2](#_Toc20381726)

[4. Monitor Kubernetes itself 5](#_Toc20381727)

[Second part: Additional components: Alertmanager, Push gateway, Grafana, external storage. 7](#_Toc20381728)

[1. Alertmanager 7](#_Toc20381729)

[2. Grafana 7](#_Toc20381730)

[Third part: Torque Exporter 10](#_Toc20381731)

[1. How to add a metric to torque\_exporter and the EUXDAT Monitoring bundle 10](#_Toc20381732)

## Introduction

The Prometheus bundle for EUXDTA is comprised of 3 components: Prometheus, Grafana and torque\_exporter.

Below we describe EUXDAT-specific installation, configuration, etc. issues that should help the developer to troubleshoot when things go wrong and add new functionality when it is required. There are hardcoded paths, usernames, and such, for illustrative purposes, in this document, so these should obviously be substituted accordingly.

## First part: Prometheus on Kubernetes

Based on: <https://sysdig.com/blog/kubernetes-monitoring-prometheus/>

### Create the config map:

kubectl create configmap prometheus-example-cm --from-file /data/git/euxdat-monitor/prometheus/prometheus.yml

This is a yaml file that describes the configuration file needed for prometheus to run against specific targets, e.g. if we want to monitor prometheus itself, it looks like this:

scrape\_configs:

- job\_name: 'prometheus'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:9090']

Because prometheus will run on its own pod, we put ‘localhost:9090’ because it will query itself for its metrics.

### Create the deployment, service and ingress:

kubectl create -f /data/git/euxdat-monitor/prometheus/prometheus\_sysdig.yaml

You should now be able to view Prometheus running at <https://prometheus.test.euxdat.eu/graph>

### Add the applications as targets to monitor:

#### Traefik (… a Kubernetes service with Prometheus)

This comes with Prometheus support already, so we start with this example:

kubectl create -f /data/git/euxdat-monitor/prometheus/traefik-prom.yaml

Then get the IP of the pod:

[root@euxdat-test-0001 ~]# kubectl get pods traefik-ingress-controller-5d7ccbbcbc-7wqtq -o wide

NAME READY STATUS RESTARTS AGE IP NODE

traefik-ingress-controller-5d7ccbbcbc-7wqtq 1/1 Running 0 9m 10.244.2.188 euxdat-test-0003.novalocal

Edit the prometheus.yml file, from:

- job\_name: 'traefik'

static\_configs:

- targets: ['10.99.197.153:8080']

To:

- job\_name: 'traefik'

static\_configs:

- targets: ['10.244.2.188:8080']

In the guide it says can just add ‘traefik:8080’, or using the long FQDN format: traefik.default.svc.cluster.local, but neither seems to work for me right now. When the time comes to use actual applications we want to monitor, you should find out how to do it (ask Ugo for example).

Now need to patch the configmap like this:

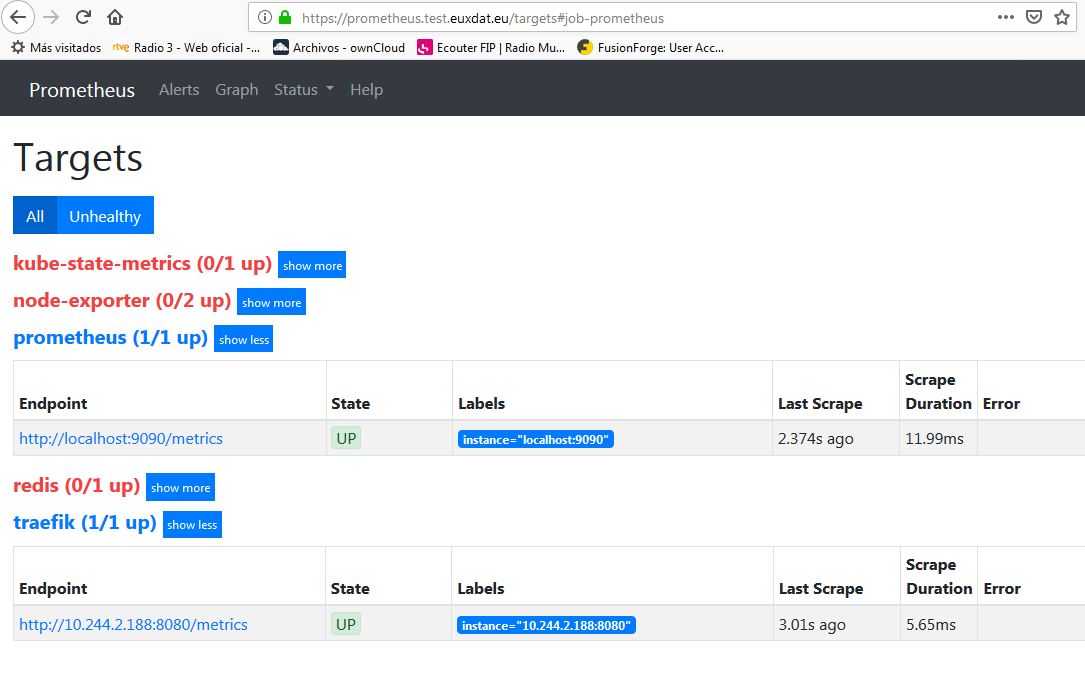
kubectl create configmap prometheus-example-cm --from-file=/data/git/euxdat-monitor/prometheus/prometheus.yml -o yaml --dry-run | kubectl apply -f -

And should patch the deployment too, but the example in the guide doesn’t work, and I don’t understand what he’s trying to do with that ‘date + %s’ thing anyway… So, delete and redeploy:

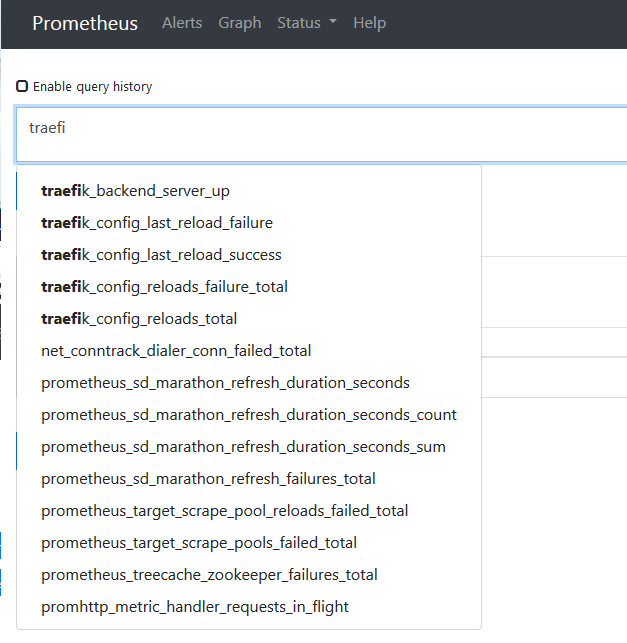
kubectl delete deployments. prometheus-deployment

kubectl create -f /data/git/euxdat-monitor/prometheus/prometheus\_just\_deploy.yaml

Should now see Traefik as a working/up/running/healthy target in Prometheus:



Should now also be able to use Traefik specific metrics:



#### Redis (… Kubernetes services with Prometheus exporters)

This is an example of an application which doesn’t have built in Prometheus support, so needs a Prometheus exporter, an additional process able to retrieve the state/logs/other metric formats of the main service and expose this info as Prometheus metrics (a Prometheus adapter).

First create the redis and redis-exporter deployment with this command:

kubectl create -f /data/git/euxdat-monitor/prometheus/redis\_prometheus\_exporter.yaml

This yaml creates a deployment with two containers in its pod, one for redis and one for redis-exporter with their respective images.

Then do the configmap + redeploy prometheus cycle:

Edit /data/git/euxdat-monitor/prometheus/prometheus.yml

Patch the configmap:

kubectl create configmap prometheus-example-cm --from-file=/data/git/euxdat-monitor/prometheus/prometheus.yml -o yaml --dry-run | kubectl apply -f -

Delete and deploy:

kubectl delete deployments. prometheus-deployment

kubectl create -f /data/git/euxdat-monitor/prometheus/prometheus\_just\_deploy.yaml

And check the targets and metrics in Prometheus to make sure it’s worked.

### Monitor Kubernetes itself

#### node-exporter

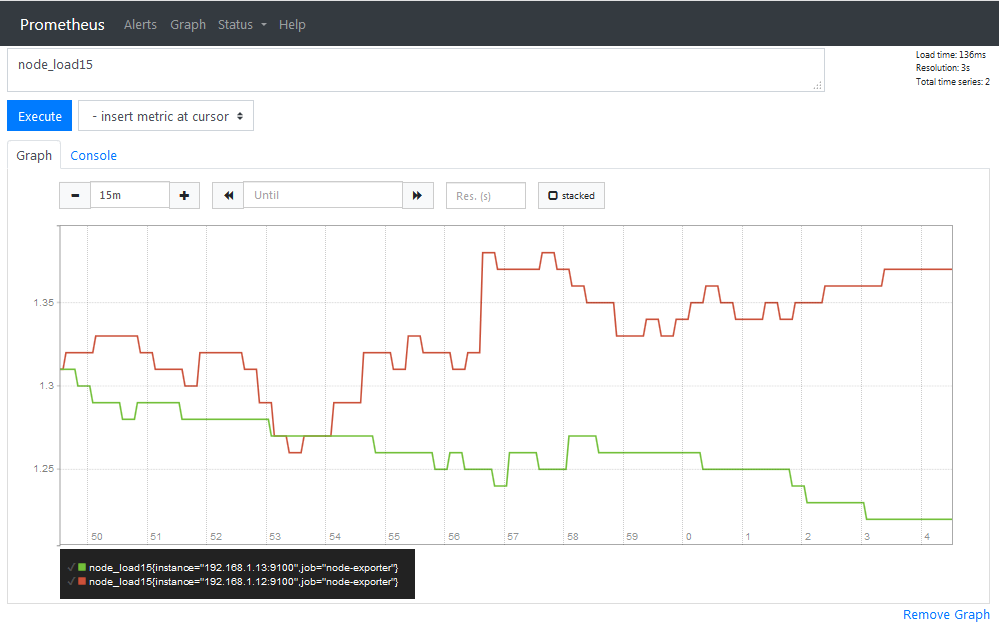
Node-exporters can monitor the cluster nodes themselves.

To get them working:

kubectl create -f /data/git/euxdat-monitor/prometheus/node-exporter-daemonset.yml

Then check the running pods for the IPs. In my case the IPs are the same I had put in the prometheus.yml configmap file already, so I didn’t need to change anything and the targets in Prometheus were working.

Can see a nice graph in Prometheus now with a node metric:



#### kube-state-metrics

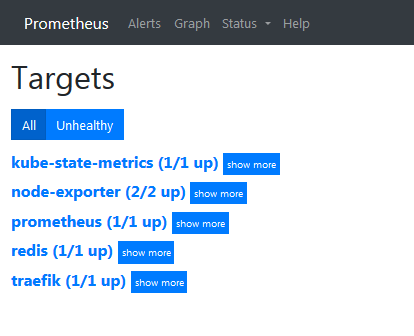
This component monitors Kubernetes elements s.a. Pods, Services, … etc.

Create it with:

kubectl create -f /data/git/euxdat-monitor/prometheus/kubernetes/

If any errors, do the individual yamls and diagnose the problem, because there are various namespaces… etc.

And now Prometheus has all the targets we want:



## Second part: Additional components: Alertmanager, Push gateway, Grafana, external storage.

Based on: <https://sysdig.com/blog/kubernetes-monitoring-with-prometheus-alertmanager-grafana-pushgateway-part-2/>

### Alertmanager

First, let’s install the Alertmanager, though I haven’t been able to see it working, nor do I understand what I’m supposed to see… Anyway, that link (part-2) explains a few things before the Alertmanager installation that should be interesting, such as the configuration of Alertmanager… etc. I also think it may be pointing at the 2nd prometheus pods…

kubectl create cm prometheus-example-cm2 --from-file /data/git/euxdat-monitor/alertmanager/prometheus.yml

or put the lines in …/alertmanager/prometheus.yml in …/prometheus/prometheus.yml and patch it and restart the deployment.

This configmap needed for alertmanager:

kubectl create cm prometheus-rules-general --from-file /data/git/euxdat-monitor/prometheus/generalrules.yaml

Then access <https://webhook.site/> and replace the random URL you will get in the alertmanager.yml file, url: ‘your url here’ parameter, and then:

kubectl create cm alertmanager-config --from-file=/data/git/euxdat-monitor/alertmanager/alertmanager.yml

Followed by:

kubectl create -f /data/git/euxdat-monitor/alertmanager/prometheus-example.yaml

Or, put the differences (few, just rules-general related) in …/prometheus/prometheus\_sysdig.yaml and delete and create the deployment, service and ingress in prometheus\_sysdig.yaml and recreate that one.

kubectl create -f /data/git/euxdat-monitor/alertmanager/alertmanager-deployment.yaml

Should then be able to look at that webhooks url to see sth… but I get that ridiculous “Too Many Attempts.” Message.

But the alertmanager pods are running, so that is good enough for me at the moment.

### Grafana

Now we want to install Grafana to visualize the metrics that Prometheus is gathering.

First, helm install it (as root):

helm install --name mygrafana stable/grafana --set sidecar.datasources.enabled=true --set sidecar.dashboards.enabled=true --set sidecar.datasources.label=grafana\_datasource --set sidecar.dashboards.label=grafana\_dashboard

Then the grafana yamls. First, this ingress allows us to access grafana at <https://grafana.test.euxdat.eu> :

kubectl create -f /data/git/euxdat-monitor/grafana/grafana-ingress.yaml

Get the mygrafana service url with this command:

kubectl get svc mygrafana

And configure grafana for our demo purposes:

kubectl create -f /data/git/euxdat-monitor/grafana/dashboard-capacity.yaml

kubectl create -f /data/git/euxdat-monitor/grafana/datasource.yaml

Now go to the before mentioned url: <https://grafana.test.euxdat.eu> and log in with user = admin and the password determined with this command:

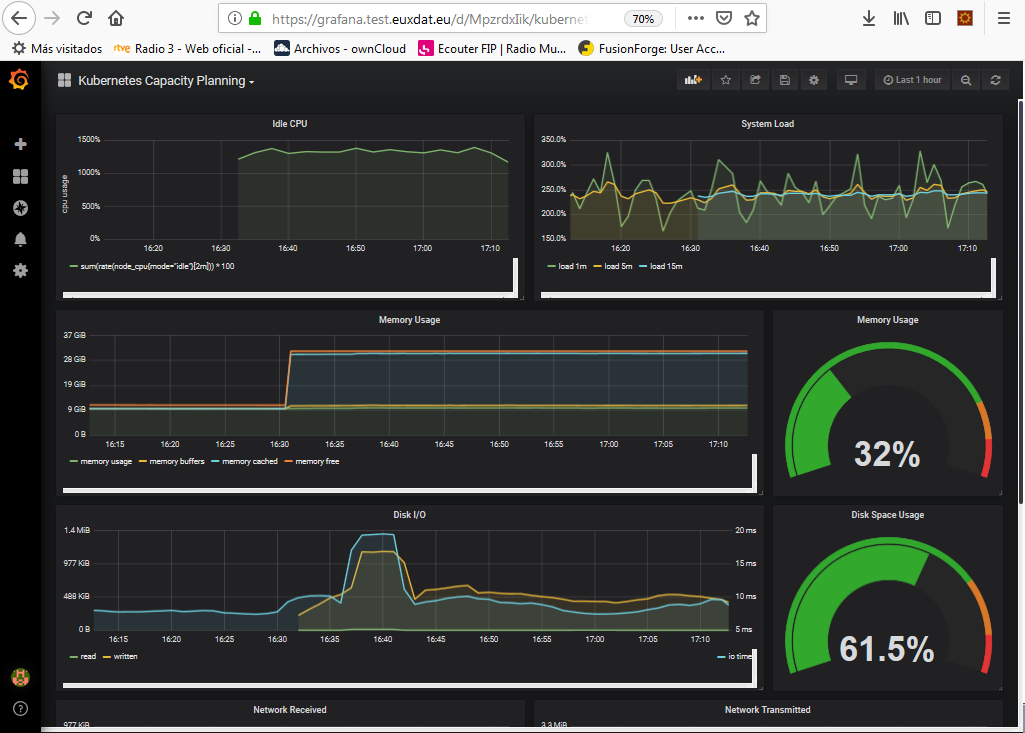
kubectl get secret --namespace default euxdat-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

pGk33XtTz4NLLLCDBKdacyfVoSrtbqh4HQYTBmKU

Once in, go to Configuration 🡪 Data Sources 🡪 Add Data Source

And choose the Prometheus one. Change the name to Prometheus-server to correspond to the yamls, and change the URL to the prometheus one on euxdat: <https://prometheus.test.euxdat.eu>

Now go to the Dashboards and click on “Kubernetes Capacity Planning”. If you can’t find it, it should be somewhere, linked to the Data source, or somewhere around there.



And there you have it! A Prometheus installation being visualized by Grafana on the EUXDAT platform.

## Third part: Torque Exporter

To monitor the HPC cluster, we have developed a Prometheus plugin called torque\_exporter (https://github.com/ari-apc-lab/torque\_exporter). It is a module written in Go (as is Prometheus), which connects to the HPC cluster via ssh and uses Torque commands to retrieve information about the status of running jobs.

To date, two metrics have been implemented: te\_showq\_r and te\_qstat\_u.

* te\_showq\_r stands for torque\_exporter showq -r.
* te\_qstat\_u stands for torque\_exporter “qstat -u <user\_id>”

A third metric should be implemented:

* te\_qstat\_f which will return detailed job information via the qstat -f <job\_id> command

Further metrics can and will be added after discussions with relevant parties as to what is necessary, and especially with regards to SLA enforcement and Accounting and Billing.

### How to add a metric to torque\_exporter and the EUXDAT Monitoring bundle

te\_qstat\_u is a Prometheus metric that returns information about a user’s running jobs. It relays the response to the command qstat -u <user\_id> from the HPC cluster back to Prometheus and Grafana.

This how-to-guide will show how te\_qstat\_u was added, so a similar approach should be followed to add te\_qstat\_f and any other metrics required.

#### Torque\_exporter

The first file to look at is **qstat.go**. and the function collectQstat(). This function executes the command qstat -u xeuspimi and parses the response, which is 0 or more running jobs for the user xeuspimi. The lifecycle of jobs I have seen running on the cluster go through 3 phases: Q 🡪 R 🡪 C, i.e. queued 🡪 running 🡪 completed. After the job has completed, it only remains in this ‘C’ state for a short while (~ 5 min) and then disappears off the list, so qstat -u xeuspimi usually returns 0 lines.

TODO: the user name needs to be passed in programmatically instead of the currently hardcoded (and soon to be expired) “xeuspimi”

TODO: improve and tidy up the code

Unlike qstat -u, the response to qstat -f is not a list of jobs, but information about a specific job, so it needs to be parsed differently. Debugging torque\_exporter using Delve (see README file) is very helpful, at least for me (Spiros), when developing this code.

A call to the function collectQstat() needs to be made in **collector.go**, in the function Collect().

userJobs is declared as a TorqueCollector structure (collector.go). It is described as a prometheus metric named “te\_qstat\_u”, with its corresponding fields in NewerTorqueCollector() (also collector.go) and it is sent to Prometheus in the Describe() function. It is then associated to the actual parsed metric in qstat.go in the collectQstat() function and a returned line is sent to Prometheus via the Prometheus.Metric channel.

Finally, commit and push torque\_exporter to git. (This of course is done because I do the development on my Windows PC, and then get the code on the EUXDAT infrastructure via git).

#### Prometheus

Once the above has been implemented and tested locally (I do the work on my Windows PC), torque\_exporter needs to be built and deployed on the EUXDAT Infrastructure.

1. Update the torque exporter code:

$ cd /home/linux/ATOSES\_spiros/GITHUB-torque\_exporter/torque\_exporter

$ git pull

1. Redeploy torque\_exporter with the script redeploy\_torque\_exporter.sh

$ cd scripts

$ ./redeploy\_torque\_exporter.sh

deployment.extensions "torque-exporter" deleted

deployment.extensions/torque-exporter created

1. Modify Prometheus.yml with the new IP address

First, get torque\_exporter pod’s new IP:

# kubectl get pod torque-exporter-85688459dd-kjzfh -o wide

NAME READY STATUS RESTARTS AGE IP NODE

torque-exporter-85688459dd-kjzfh 1/1 Running 0 2m 10.244.1.46 euxdat-test-0002.novalocal

Put new IP in /home/linux/ATOSES\_spiros/euxdat-monitor/prometheus/prometheus.yml, by replacing

- job\_name: 'torque-exporter'

static\_configs:

- targets: ['10.244.1.16:9100']

with

- job\_name: 'torque-exporter'

static\_configs:

- targets: ['10.244.1.46:9100']

1. Redeploy Prometheus with the script redeploy\_prometheus.sh

$ ./redeploy\_prometheus.sh

deployment.extensions "prometheus-deployment" deleted

configmap/prometheus-example-cm configured

deployment.extensions/prometheus-deployment created

1. Check the metric is published and working in Prometheus console.

Give it a few minutes and then check the console, Figure 1.

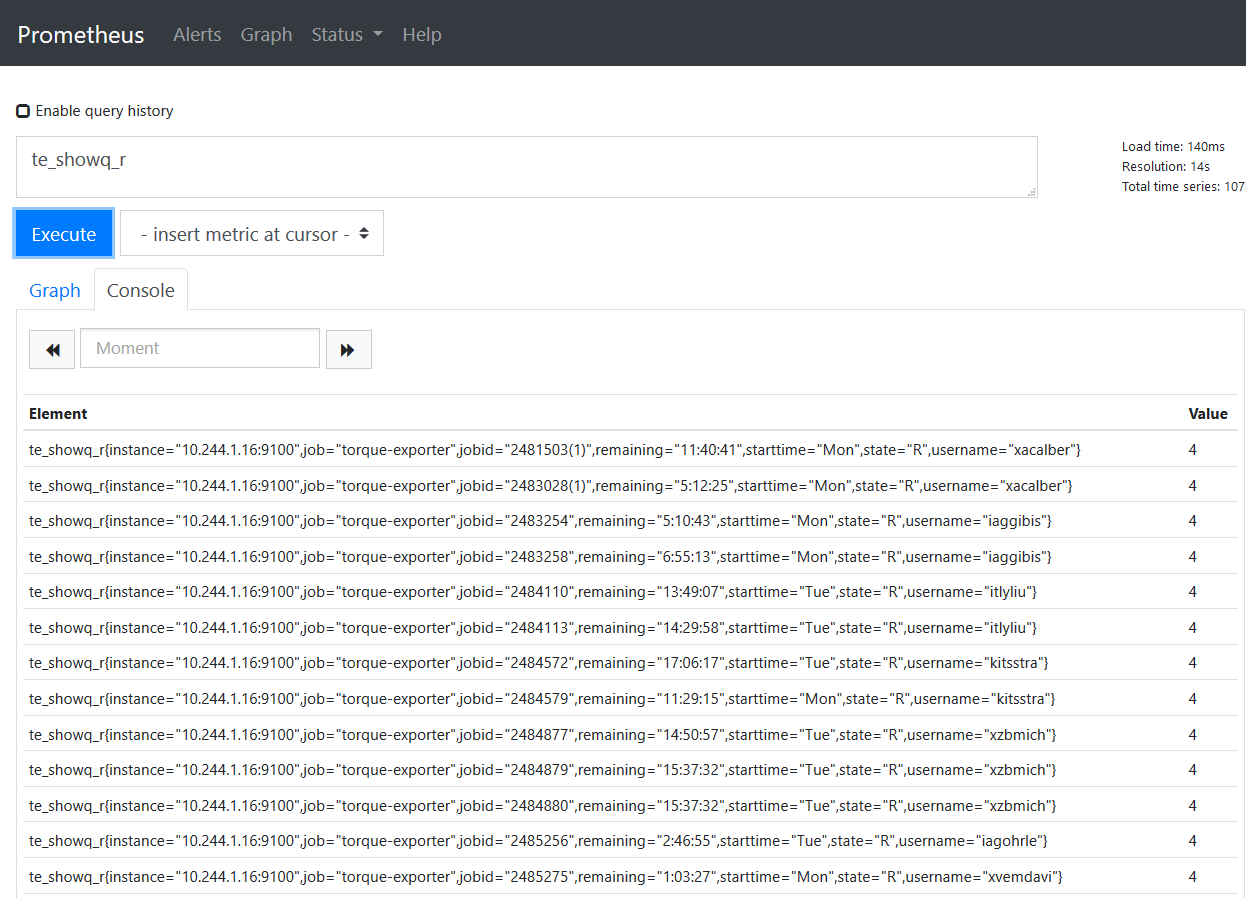


Figure 1: Prometheus te\_showq\_r, a torque\_exporter metric

#### Grafana

1. Add a new panel in Grafana (Add Panel 🡪 Add Query 🡪te\_showq\_r).

A torque\_exporter specific Grafana dashboard is being developed for the project. Figure 2 depicts the views of the two metrics te\_showq\_r and te\_qstat\_u. The dashboard is named *HPC Torque Dashboard*. It is the same information as can be seen in the Prometheus console, but in a more user friendly and user configurable view. For example, in Grafana, the user can sort the various jobs by *jobid*, *username*, etc., whereas from within Prometheus a modified query would need to be issued.

TODO: find out how to get the metrics to update automatically in Grafana. At the moment have to refresh manually.

TODO: display a nicer message when no data available, e.g. when no user jobs running, can say exactly that: “No jobs running for user ‘xeuspimi’”, or something like that.

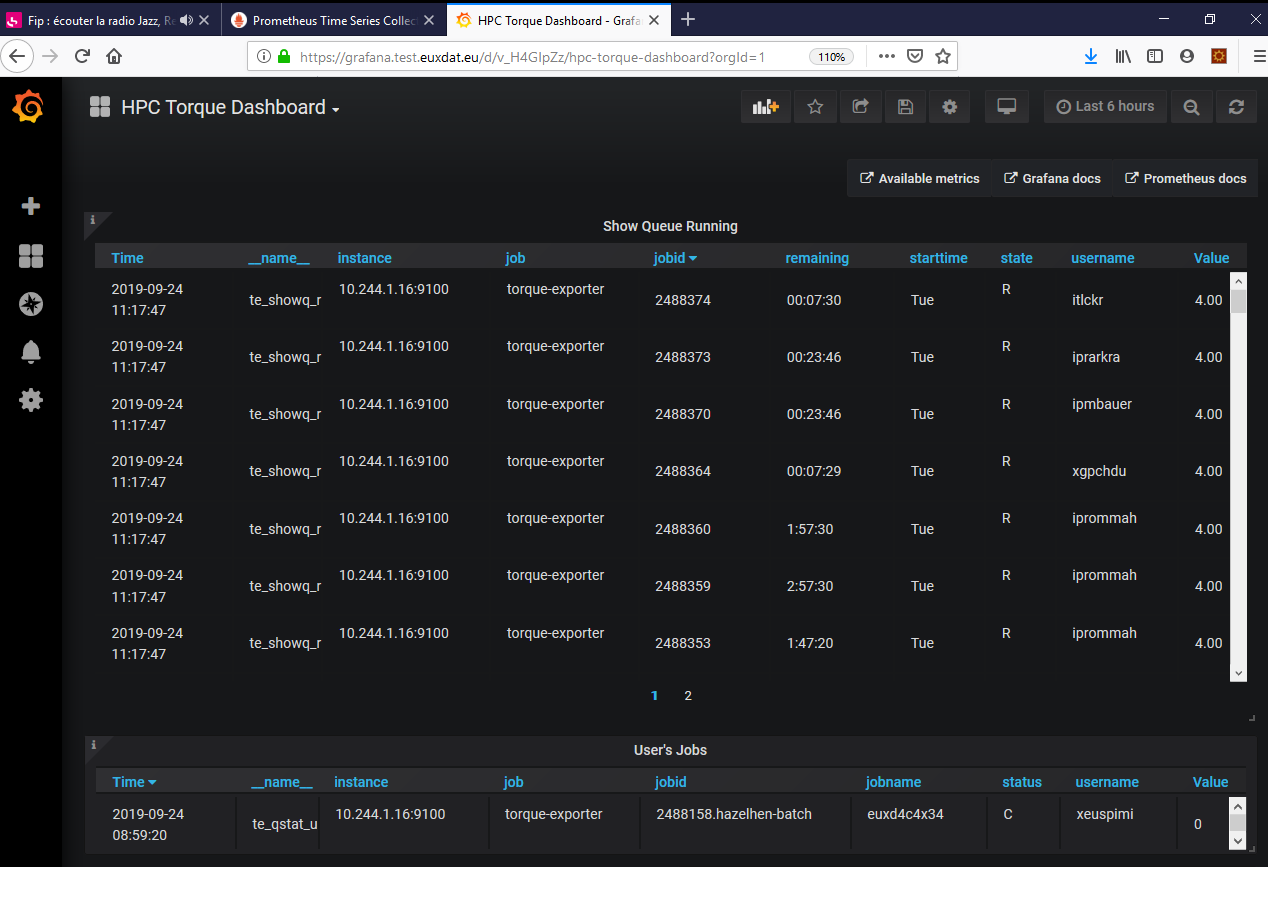


Figure 2: HPC Torque Dashboard