

Getting Started with Prometheus

Monitoring Kubernetes infrastructure and applications for reliability

Class Labs

Version 1.3 by Brent Lester on behalf of Tech Skills Transformations

04/22/2022

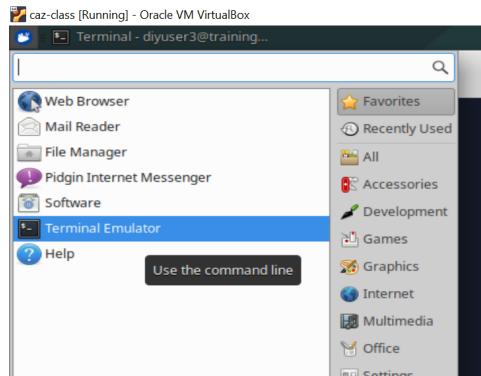
Important Prereq: These labs assume you have already followed the instructions in the separate setup document and have VirtualBox up and running on your system and have downloaded the *prom-start.ova* file and loaded it into VirtualBox. If you have not done that, please refer to the setup document for the workshop and complete the steps in it before continuing!

Startup - to do before first lab

1. Enable networking by selecting the up/down arrow icon at top right and selecting the option to "Enable Networking". See screenshot below.



2. Open a terminal session by using the one on your desktop or clicking on the little mouse icon in the upper left corner and selecting **Terminal Emulator** from the drop-down menu.



3. First, let's make sure we have the latest files for the class. For this course, we will be using a main directory *prom-start* with subdirectories under it for the various labs. In the terminal window, cd into the main directory and update the files.

```
$ cd prom-start
$ git stash
$ git pull
```

3. Next, start up the paused Kubernetes (minikube) instance on this system using a script in the *extras* subdirectory. This will take several minutes to run to start up minikube and also it will be waiting for a directory to be in existence to fix a permissions issue with it. So you can just let it run while we do the first lecture portion.

```
$ extra/start-mini2.sh
```

4. (OPTIONAL IF NEEDED FOR LAB 1.) If you can't get the Prometheus home page at localhost:31000 to come up, it is probably because of a permissions issue with the mount that it is trying to use. This is a bug with the version of Prometheus we are using, but there is a fix. In the "extra" directory is a file named "fixtmp.sh". You can run it to change the permissions on the mount. After that you may need to wait for the "Prometheus-server" pod to restart in the namespace monitoring (or delete it and restart).

```
$ k get pods -n monitoring (look for Prometheus server pod as 1/2 ready or
crashloopbackoff, etc.)
```

```
$ extra/fixtmp.sh
```

Example:

```
diyuser3@training1:~/prom-start/extras$ k get pods -n monitoring | grep
server
```

NAME	READY	STATUS
RESTARTS	AGE	
pc-prometheus-server-85f8d4c78f-zdmnn	1/2	CrashLoopBackOff
6	3d19h	

```
diyuser3@training1:~/prom-start$ extra/fixtmp.sh
```

Lab 1 - Monitoring with Prometheus

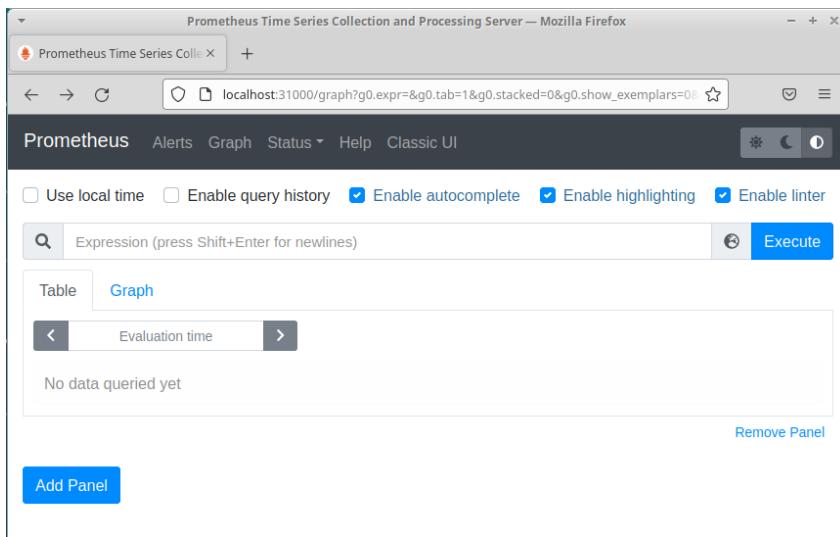
Purpose: In this lab, we'll run an instance of the node exporter and use Prometheus to surface basic data and metrics.

- For this lab, we have already setup an instance of the Prometheus Community edition main server running in a namespace in our cluster named monitoring. Take a quick look at the different pieces that we have running there since we installed the prometheus-community/prometheus helm chart.

```
$ k get all -n monitoring
```

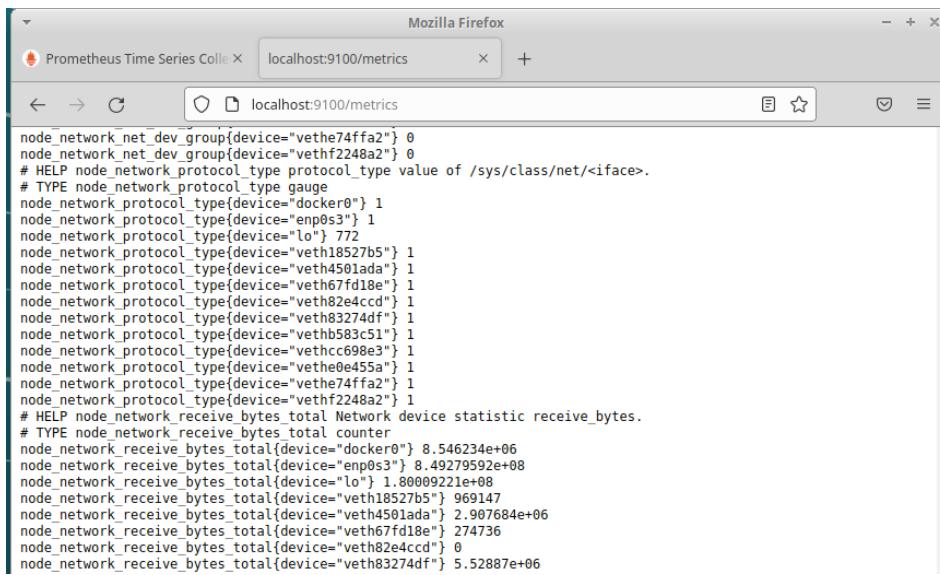
- Notice that we have both Prometheus itself and the node exporter piece running there (among others). Let's take a look at the Prometheus dashboard in the web browser. Open up a web browser and go to the url below to see it.

<http://localhost:31000/>



- We'll get to use the dashboard more in the labs. For now, lets open up the node exporter's metrics page and look at the different information on it. (Note that we only have one node on this cluster.) Go to the link below. Once on that page, scan through some of the metrics that are exposed by this exporter. Then see if you can find the "total number of network bytes received on device "lo". (Hint: look for this metric "node_network_receive_bytes_total{device="lo"}").

<http://localhost:9100/metrics>

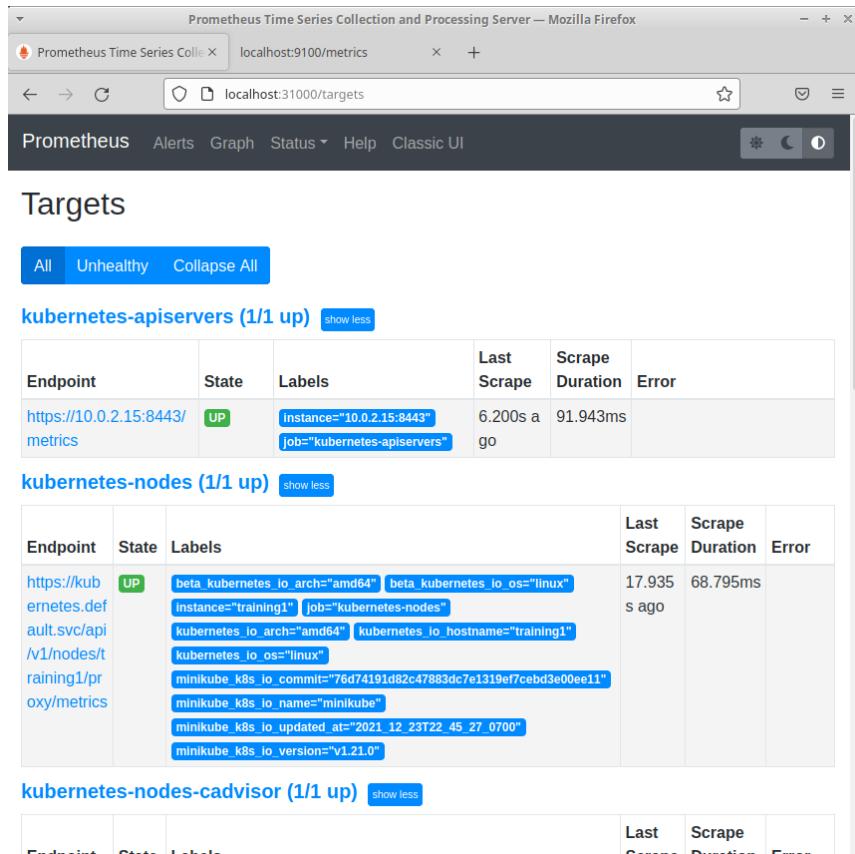


```

node_network_net_dev_group{device="vthe74ffa2"} 0
node_network_net_dev_group{device="vethf2248a2"} 0
# HELP node_network_protocol_type protocol_type value of /sys/class/net/<iface>.
# TYPE node_network_protocol_type gauge
node_network_protocol_type{device="docker0"} 1
node_network_protocol_type{device="enp0s3"} 1
node_network_protocol_type{device="lo"} 772
node_network_protocol_type{device="veth18527b5"} 1
node_network_protocol_type{device="veth4501ada"} 1
node_network_protocol_type{device="veth67fd18e"} 1
node_network_protocol_type{device="veth82e4ccd"} 1
node_network_protocol_type{device="veth83274df"} 1
node_network_protocol_type{device="vethb583c51"} 1
node_network_protocol_type{device="vethcc698e3"} 1
node_network_protocol_type{device="vthe0e455a"} 1
node_network_protocol_type{device="vthe74ffa2"} 1
node_network_protocol_type{device="vethf2248a2"} 1
# HELP node_network_receive_bytes_total Network device statistic receive_bytes.
# TYPE node_network_receive_bytes_total counter
node_network_receive_bytes_total{device="docker0"} 8.546234e+06
node_network_receive_bytes_total{device="enp0s3"} 8.49279592e+08
node_network_receive_bytes_total{device="lo"} 1.80009221e+08
node_network_receive_bytes_total{device="veth18527b5"} 969147
node_network_receive_bytes_total{device="veth4501ada"} 2.907684e+06
node_network_receive_bytes_total{device="veth67fd18e"} 274736
node_network_receive_bytes_total{device="veth82e4ccd"} 0
node_network_receive_bytes_total{device="veth83274df"} 5.52887e+06

```

- Now, let's see which targets Prometheus is automatically scraping from the cluster. Back in the top menu (dark bar) on the main Prometheus page tab, select Status and then Targets (or go to <http://localhost:31000/targets>). Then see if you can find how long ago the last scraping happened, and how long it took for the *kubernetes-nodes-cadvisor* target.



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://10.0.2.15:8443/	UP	instance="10.0.2.15:8443" job="kubernetes-apiservers"	6.200s ago	91.943ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://kubernetes.default.svc/api/v1/nodes/training1/proxy/metrics	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="training1" job="kubernetes-nodes" kubernetes_io_arch="amd64" kubernetes_io_hostname="training1" kubernetes_io_os="linux" minikube_k8s_io_commit="76d74191d82c47883dc7e1319ef7cebd3e00ee11" minikube_k8s_io_name="minikube" minikube_k8s_io_updated_at="2021-12-23T22:45:27-0700" minikube_k8s_io_version="v1.21.0"	17.935s ago	68.795ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error

5. Let's setup an application in our cluster that has a built-in Prometheus metrics exporter - traefik - an ingress. The Helm chart is already loaded for you. So, we just need to create a namespace for it and run a script to deploy it. After a few moments, you should be able to see things running in the traefik namespace.

```
$ k create ns traefik
$ ~/prom-start/extra/helm-install-traefik.sh
$ k get all -n traefik
```

6. You should now be able to see the metrics area that Traefik exposes for Prometheus as a pod endpoint. Take a look in the **Status > Targets** area of Prometheus and see if you can find it (**localhost:31000/targets**) and use a **Ctrl-F** to try to find the text "traefik". Note that this is the pod endpoint and not a standalone target. (If you don't find it, see if the "kubernetes-pods (1/1 up)" has a "show more" button next to it. If so, click on that to expand the list.)

The screenshot shows the Prometheus web interface with the URL `localhost:31000/targets`. The top navigation bar includes links for Prometheus, Alerts, Graph, Status (with a dropdown arrow), Help, and Classic UI. Below the navigation, there are two sections: "kubernetes-pods (1/1 up)" and "kubernetes-service-endpoints (3/3 up)". A red circle highlights the "Status" dropdown menu. Below these sections is a table with columns: Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The table currently has no data rows.

The screenshot shows the Prometheus web interface with the URL `localhost:31000/classic/targets`. The top navigation bar includes links for Prometheus, Alerts, Graph, Status (with a dropdown arrow), Help, and New UI. Below the navigation, there is a table with columns: Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. One row is visible, showing an endpoint at `http://172.17.0.10:9100/metrics` with state `UP`. The "Labels" column contains several labels: `app_kubernetes_io_instance="traefik"`, `app_kubernetes_io_managed_by="Helm"`, `app_kubernetes_io_name="traefik"`, and `helm_sh_chart="traefik-10.9.1"`. At the bottom of the screen, there is a search bar containing the text "traefik".

You can then click on the link in the Endpoint column to see the metrics that Traefik is generating.

```

process_start_time_seconds 1.64080812031e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 8.15730688e+08
# HELP process_virtual_memory_max_bytes Maximum amount of virtual memory available in bytes.
# TYPE process_virtual_memory_max_bytes gauge
process_virtual_memory_max_bytes 1.8446744073709552e+19
# HELP traefik_config_last_reload_failure Last config reload failure
# TYPE traefik_config_last_reload_failure gauge
traefik_config_last_reload_failure 0
# HELP traefik_config_last_reload_success Last config reload success
# TYPE traefik_config_last_reload_success gauge
traefik_config_last_reload_success 1.640808121e+09
# HELP traefik_config_reloads_failure_total Config failure reloads
# TYPE traefik_config_reloads_failure_total counter

```

- While we can find it as a pod endpoint, we don't yet have the traefik metrics established as a standalone "job" being monitored in Prometheus. You can see this because there is no section specifically for "traefik (1/1 up)" in the Targets page. Also, Traefik is not listed if you check the Prometheus service-discovery page at <http://localhost:31000/service-discovery>

Prometheus Time Series Collection and Processing Server — Mozilla Firefox

Prometheus Time Series Collection and Processing Server — localhost:9100/metrics

Prometheus

- kubernetes-nodes-cadvisor (1 / 1 active targets)
- kubernetes-pods (1 / 21 active targets)
- kubernetes-service-endpoints (3 / 13 active targets)
- prometheus (1 / 1 active targets)
- prometheus-pushgateway (1 / 13 active targets)
- kubernetes-pods-slow (0 / 24 active targets)
- kubernetes-service-endpoints-slow (0 / 15 active targets)
- kubernetes-services (0 / 13 active targets)

kubernetes-apiservers [show more](#)

kubernetes-nodes [show more](#)

traefik

- So we need to tell Prometheus about traefik as a job. There are two ways. One way is just to apply two annotations to the service for the target application. However, this will not work with more advanced versions of Prometheus. So, we'll do this instead by updating a configmap that the Prometheus server uses to get job information out of. First let's take a look at what has to be changed to add this job. We have a "before" and "after" version in the extra directory. We'll use a tool called "meld" to see the differences.

```
$ cd ~/prom-start/extra (if not already there)
```

```
$ meld ps-cm-start.yaml ps-cm-with-traefik.yaml
```

```

apiVersion: v1
data:
  alerting_rules.yml: |
    {}
  alerts: |
    {}
  prometheus.yml: |
    global:
      evaluation_interval: 1m
      scrape_interval: 1m
      scrape_timeout: 10s
    rule_files:
      - /etc/config/recording_rules.yml
      - /etc/config/alerting_rules.yml
      - /etc/config/rules
      - /etc/config/alerts
    scrape_configs:
      - job_name: prometheus
        static_configs:
          - targets:
              - localhost:9090
            bearer_token_file: /var/run/secrets/kubernetes-api/servers
            job_name: kubernetes-apiservers
            kubernetes_sd_configs:

```

```

apiVersion: v1
data:
  alerting_rules.yml: |
    {}
  alerts: |
    {}
  prometheus.yml: |
    global:
      evaluation_interval: 1m
      scrape_interval: 1m
      scrape_timeout: 10s
    rule_files:
      - /etc/config/recording_rules.yml
      - /etc/config/alerting_rules.yml
      - /etc/config/rules
      - /etc/config/alerts
    scrape_configs:
      - job_name: traefik
        static_configs:
          - targets:
              - traefik.traefik.svc.cluster.local:9100
            job_name: traefik
            instance: "traefik.traefik.svc.cluster.local:9100"
            labels: "job=traefik"

```

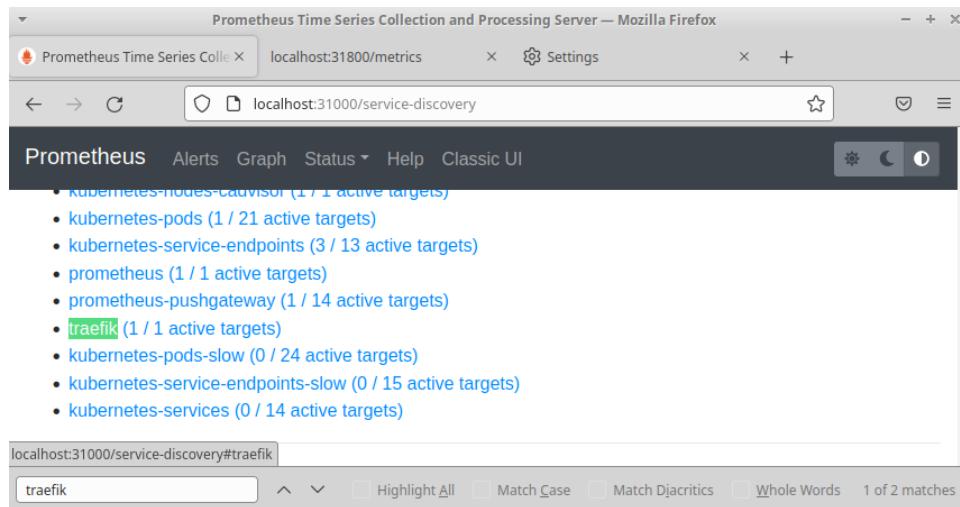
Ln 18, Col 1 INS

- It's easy to see the difference here. When you're done viewing, just go ahead and close meld. Now we'll apply the new configmap definition with our additional job. (Ignore the warning.)

```
$ k apply -n monitoring -f ps-cm-with-traefik.yaml
```

- Now if you refresh and look at the Status->Targets page in Prometheus at <http://localhost:31000/targets> and the Service Discovery page at <http://localhost:31000/service-discovery> and do a Ctrl-F to search for traefik, you should find that the new item shows up as a standalone item on both pages. (It may take a moment for the traefik target to reach (1/1 up) in the targets page, so you may have to refresh after a moment.)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://traefik.traefik.svc.cluster.local:9100/metrics	UP	instance="traefik.traefik.svc.cluster.local:9100" job="traefik"	36.310s ago	1.664ms	



END OF LAB

Lab 2- Deploying a separate exporter for an application

Purpose: In this lab, we'll see how to deploy a separate exporter for a mysql application running in our cluster.

1. We have a simple webapp application with a mysql backend that we're going to run in our cluster. The application is named "roar" and we have a manifest with everything we need to deploy it into our cluster. Go ahead and deploy it now.

```
$ k create ns roar
```

```
$ k apply -f roar-complete.yaml
```

2. After a few moments, you can view the running application in the roar namespace and also in the browser at <http://localhost:31790/roar> .

```
$ k get all -n roar
```

R.O.A.R (Registry of Animal Responders) Agents						
Show 10 entries		Search:				
ID	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	Perry	platypus	2013-01-20	2015-04-09	H. Doofenshmirtz	...inator
4	Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
5	Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun

Showing 1 to 5 of 5 entries Previous 1 Next

- Since we are using mysql on the backend, we'd like to be able to get metrics on that. Let's see if there's already a mysql exporter available. The best place to look if you're using the Prometheus community edition is in the helm charts area of that. Open a browser tab to <https://github.com/prometheus-community/helm-charts/tree/main/charts>. Notice that there's a prometheus-mysql-exporter chart. You can click on that and look at the README.md for it.

The screenshot shows a GitHub repository page for 'prometheus-community/helm-charts'. The repository has 51 pull requests, 2.2k forks, and 2k stars. The commit history for the 'main' branch shows the following recent commits:

- jimengze [kube-prometheus-stack] update kubernetes-system-kube-proxy.yaml (#1629) ... - f33b075 18 hours ago
- ..
- alertmanager [alertmanager] add podLabels value (#1429) 3 months ago
- kube-prometheus-stack [kube-prometheus-stack] update kubernetes-system-kube-proxy.yaml (#1629) 18 hours ago
- kube-state-metrics [kube-state-metrics] Version bump to 2.3.0 (#1600) 12 days ago
- prometheus-adAPTER [prometheus-adAPTER] Recommended Kubernetes labels , custom labels an... 2 months ago
- prometheus-blackbox-exporter [prometheus-blackbox-exporter] revert change from #1450 and fix servi... 2 months ago
- prometheus-cloudwatch-exporter Add missing whitespace to cloudwatch-exporter chart (#1533) last month
- prometheus-consul-exporter [prometheus-consul-exporter] Add Pod Annotations Value (#1416) 2 months ago
- prometheus-couchdb-exporter feat(prometheus-couchdb-exporter): add configurable apiVersion (#384) 14 months ago
- prometheus-druid-exporter [prometheus-druid-exporter] Add nodeSelector, tolerations, and affini... 3 months ago
- prometheus-elasticsearch-exporter [prometheus-elasticsearch-exporter] Add support for indices mappings (#... 9 days ago
- prometheus-json-exporter [prometheus-json-exporter] New Exporter (#1460) 2 months ago
- prometheus-kafka-exporter [prometheus-kafka-exporter] - added SASL support and upgrade to lates... 3 months ago
- prometheus-mongodb-exporter Prep initial charts indexing (#14) 17 months ago
- prometheus-mysql-exporter [prometheus-mysql-exporter] Add namespace support to service monitor ... 6 days ago

Prometheus MySql Exporter

A Prometheus exporter for MySQL metrics.

This chart bootstraps a Prometheus MySQL Exporter deployment on a Kubernetes cluster using the Helm package manager.

Get Repo Info

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

See [helm repo](#) for command documentation.

Install Chart

```
# Helm 3
$ helm install [RELEASE_NAME] prometheus-community/prometheus-mysql-exporter

# Helm 2
$ helm install --name [RELEASE_NAME] prometheus-community/prometheus-mysql-exporter
```

- As part of the configuration for this, we need to setup a new user with certain privileges in the database that's running for our backend. For simplicity, I've provided a simple script that you can run for this. You can take a look at the script to see what it does and then run it to add the user and privileges. (Note that it requires the namespace as an argument to be passed to it.) This script and other files are in a different directory under prom-start named "mysql-ex".

```
$ cd ~/prom-start/mysql-ex
$ cat update-db.sh
$ ./update-db.sh roar  (note we supply namespace where db is running)
```

- Now we are ready to deploy the mysql helm chart to get our mysql exporter up and running. To do this we need to supply a values.yaml file that defines the image we want to use, a set of metrics "collectors" and the pod to use (via labels). We also have a data file for a secret that is required with information on the service, user, password, and port that we want to access. Take a look at those files.

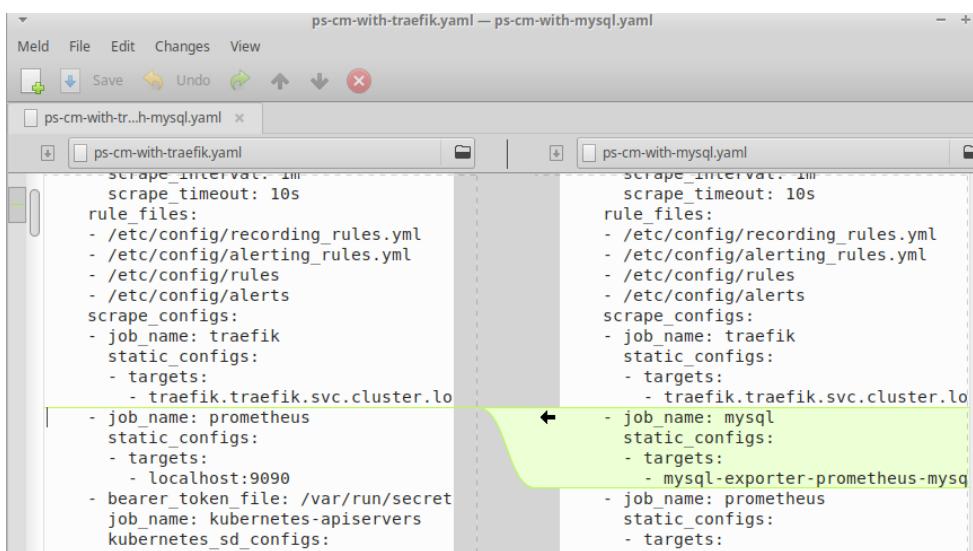
```
$ cat values.yaml
$ cat secret.yaml
```

- Now we can go ahead and deploy the helm chart for the exporter with our custom values. For convenience, there is a script that runs the helm install. After a few moments you should be able to see things spinning up in the monitoring namespace.

```
$ ./helm-install-mysql-ex.sh
$ k get all -n monitoring | grep mysql
```

- Finally, to connect up the pieces, we need to define a job for Prometheus. We can do this the same way we did for Traefik in Lab 1. To see the changes, you can look at a diff between the configmap definition we used for Traefik and one we already have setup with the definition for the mysql exporter. You can exit meld when done.

```
$ meld ./extra/ps-cm-with-traefik.yaml ps-cm-with-mysql.yaml
```



- Now you can apply the updated configmap definition.

```
$ k apply -n monitoring -f ps-cm-with-mysql.yaml
```

- You should now be able to see the mysql item in the targets page (localhost:31000/targets) and also in the service-discovery page (localhost:31000/service-discovery#mysql). (Again, it may take a few minutes for the mysql target to appear and reach (1/1 up).)

The screenshot shows two browser tabs. The top tab is 'localhost:31800/metrics' titled 'Prometheus Time Series Collection and Processing Server — Mozilla Firefox'. It displays a table of scraped targets:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mysql-exporter-prometheus.mysql-exporter.monitoring.svc.cluster.local:9104/mysql	UP	namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"	39.483s ago	70.044ms	

The bottom tab is 'localhost:31000/targets' titled 'Prometheus — Mozilla Firefox'. It shows a search results table for 'mysql':

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
mysql					

Below the tables are search filters: 'Highlight All', 'Match Case', 'Match Diacritics', and 'Whole Words'. A message at the bottom says '1 of 6 matches'.

Service Discovery

- kubernetes-apiservers (1 / 20 active targets)
- kubernetes-nodes (1 / 1 active targets)
- kubernetes-nodes-cadvisor (1 / 1 active targets)
- kubernetes-pods (1 / 24 active targets)
- kubernetes-service-endpoints (3 / 18 active targets)
- mysql (1 / 1 active targets)
- prometheus (1 / 1 active targets)

10. (Optional) If you want to see the metrics that are exposed by this job, there is a small script named **pf.sh** (in mysql-ex) that you can run to setup port-forwarding for the mysql-exporter. Then you can look in the browser at <http://localhost:9104/metrics>.

```
$ ./pf.sh
```

```
localhost:9104/metrics
go_memstats_stack_sys_bytes 425984
# HELP go_memstats_sys_bytes Number of bytes obtained from system.
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 7.4269704e+07
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 5
# HELP mysql_exporter_collector_duration_seconds Collector time duration.
# TYPE mysql_exporter_collector_duration_seconds gauge
mysql_exporter_collector_duration_seconds{collector="collect.binlog_size"} 0.004260215
mysql_exporter_collector_duration_seconds{collector="collect.global_status"} 0.003972186
mysql_exporter_collector_duration_seconds{collector="collect.global_variables"} 0.012792833
mysql_exporter_collector_duration_seconds{collector="collect.info_schema.clientstats"} 0.005440051
mysql_exporter_collector_duration_seconds{collector="collect.info_schema.innodb_cmp"} 0.015794649
mysql_exporter_collector_duration_seconds{collector="collect.info_schema.innodb_cmpmem"} 0.016336161
```

END OF LAB

Lab 3 - Writing queries with PromQL

Purpose: In this lab, we'll see how to construct queries with the PromQL language.

1. We're now going to turn our attention to creating queries in the Prometheus interface using Prometheus' built-in query language, PromQL. First, to get ready for this, in the browser that is running the Prometheus interface, switch back to the main Prometheus window by clicking on "Prometheus" in the dark line at the top, or going to localhost:31000. Once there, click to enable the five checkboxes under the main menu.

The screenshot shows the Prometheus Graph interface. At the top, there are navigation icons (back, forward, search) and a URL bar showing "localhost:31000/graph?g0.expr=&g0.tab=1&g0.stacked=0&g0.show_e". Below the URL is a dark header bar with the word "Prometheus" and links for "Alerts", "Graph", "Status", "Help", and "Classic UI". Underneath the header are five checked checkboxes: "Use local time", "Enable query history", "Enable autocomplete", "Enable highlighting", and "Enable linter". Below these checkboxes is a search bar with placeholder text "Expression (press Shift+Enter for newlines)". Under the search bar are two tabs: "Table" and "Graph", with "Graph" being the active tab. Below the tabs is a "Evaluation time" selector with arrows. A message "No data queried yet" is displayed. At the bottom left is a blue "Add Panel" button.

2. There are a couple of different ways to find available metrics to choose from in Prometheus. One way is to click on the query explorer icon next to the blue "Execute" button on the far right. Click on that and you can scroll through the list that pops up. You don't need to pick any right now and you can close it (via the "x" in the upper right) when done.

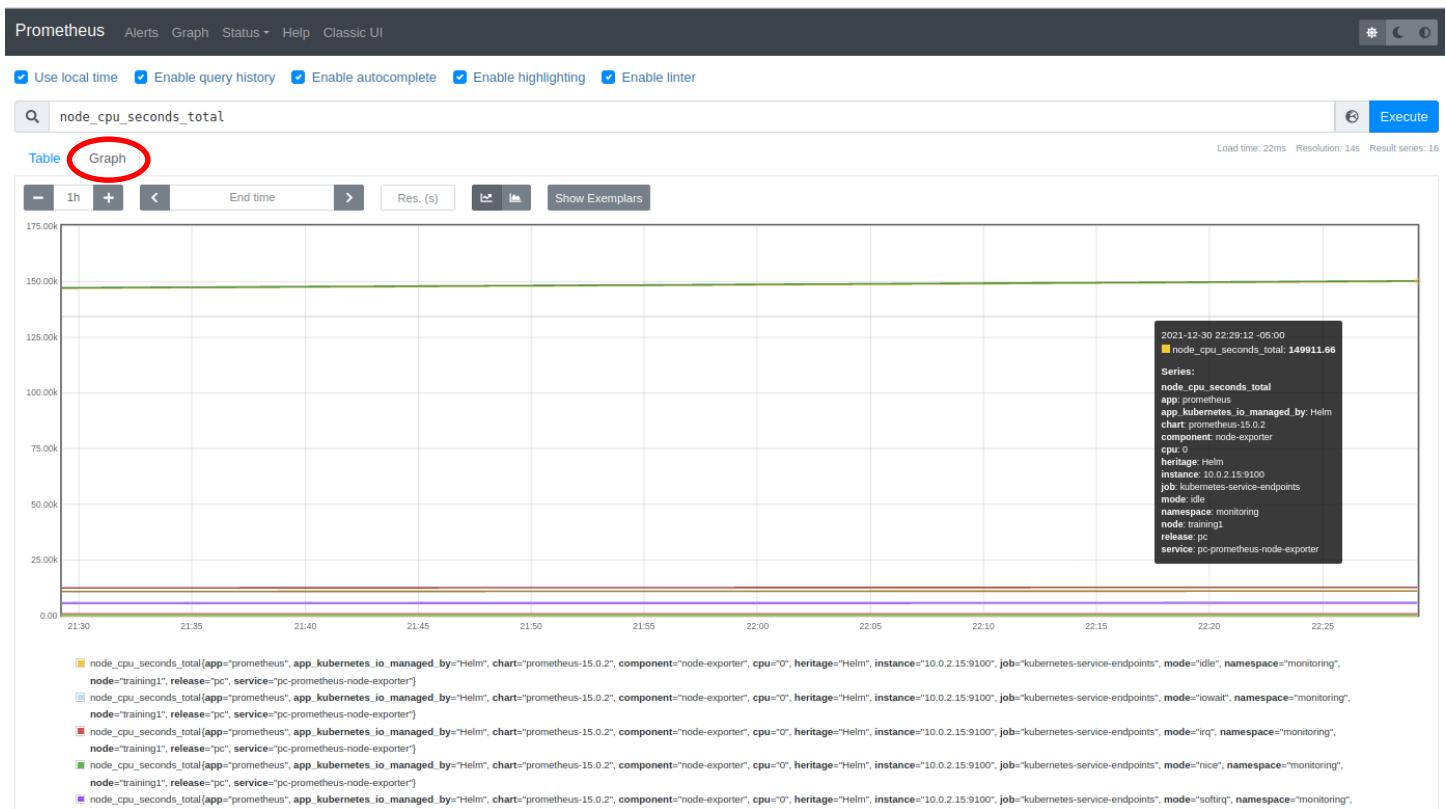
The screenshot shows the Metrics Explorer dialog box. It has a title bar "Metrics Explorer" and a close button "x". Inside the dialog, there is a list of metric names: "aggregator_openapi_v2_regeneration_count", "aggregator_openapi_v2_regeneration_duration", "aggregator_unavailable_apiservice", "apiextensions_openapi_v2_regeneration_count", "apiserver_admission_controller_admission_duration_seconds_bucket", "apiserver_admission_controller_admission_duration_seconds_count", "apiserver_admission_controller_admission_duration_seconds_sum", and "apiserver_admission_step_admission_duration_seconds_bucket". To the right of the list is a blue "Execute" button with a small circular icon above it. At the bottom right of the dialog is a "Remove Panel" link.

3. Another way to narrow in quickly on a metric you're interested in is to start typing in the "Expression" area and pick from the list that pops-up based on what you've typed. Try typing in the names of some of the applications that we are monitoring and see the metrics available. For example, you can type in "con" to see the ones for containers, "mysql" to see the ones for mysql, and so on. You don't need to select any right now, so once you are done, you can clear out the Expression box.

4. Now, let's actually enter a metric and execute it and see what we get. Let's try a simple "time series" one. In the Expression box, type in "node_cpu_seconds_total". As the name may suggest to you, this is a metric provided by the node exporter and tracks the total cpu seconds for the node. In our case, we only have one node. After you type this in, click on the blue "Execute" button at the far right to see the results.

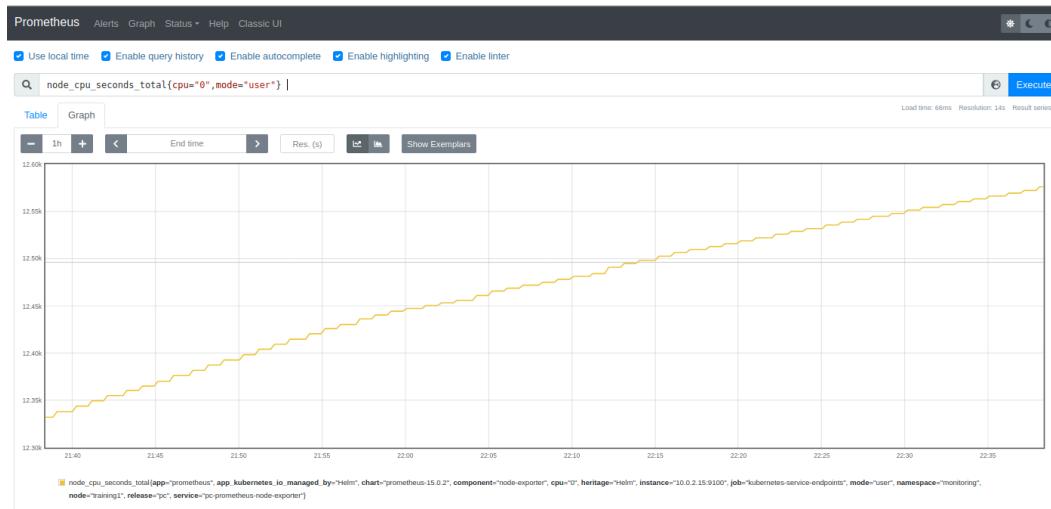
Series	Labels	Value
node_cpu_seconds_total{app="prometheus", chart="prometheus-15.0.2", component="node-exporter", cpu="0", heritage="Helm", instance="10.0.2.15:9100", job="kubernetes-service-endpoints", mode="idle", namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"}		149642.6
node_cpu_seconds_total{app="prometheus", chart="prometheus-15.0.2", component="node-exporter", cpu="0", heritage="Helm", instance="10.0.2.15:9100", job="kubernetes-service-endpoints", mode="lowlat", namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"}		137.21
node_cpu_seconds_total{app="prometheus", chart="prometheus-15.0.2", component="node-exporter", cpu="0", heritage="Helm", instance="10.0.2.15:9100", job="kubernetes-service-endpoints", mode="irq", namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"}		0
node_cpu_seconds_total{app="prometheus", chart="prometheus-15.0.2", component="node-exporter", cpu="0", heritage="Helm", instance="10.0.2.15:9100", job="kubernetes-service-endpoints", mode="nice", namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"}		80.2
node_cpu_seconds_total{app="prometheus", chart="prometheus-15.0.2", component="node-exporter", cpu="0", heritage="Helm", instance="10.0.2.15:9100", job="kubernetes-service-endpoints", mode="softirq", namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"}		503.56
node_cpu_seconds_total{app="prometheus", chart="prometheus-15.0.2", component="node-exporter", cpu="0", heritage="Helm", instance="10.0.2.15:9100", job="kubernetes-service-endpoints", mode="steal", namespace="monitoring", node="training1", release="pc", service="pc-prometheus-node-exporter"}		0

5. Notice that we have a lot of rows of output from this single query. If you look closely, you can see that each row is different in some aspect, such as the cpu number or the mode. Rows of data like this are not that easy to digest. Instead, it is easier to visualize with a graph. So, click on the "Graph" link above the rows of data to see a visual representation. You can then move your cursor around and get details on any particular point on the graph. Notice that there is a color-coded key below the graph as well.



- What if we want to see only one particular set of data? If you look closely at the lines below the graph, you'll see that each is qualified/filtered by a set of "labels" within {} and }. We can use the same syntax in the Expression box with any labels we choose to pick which items we see. Change your query to the one below and then click on Execute again to see a filtered graph. (Notice that Prometheus will offer pop-up lists to help you fill in the syntax if you want to use them.) After you click Execute, you will see a single data series that increases over time.

```
node_cpu_seconds_total{cpu="0",mode="user"}
```



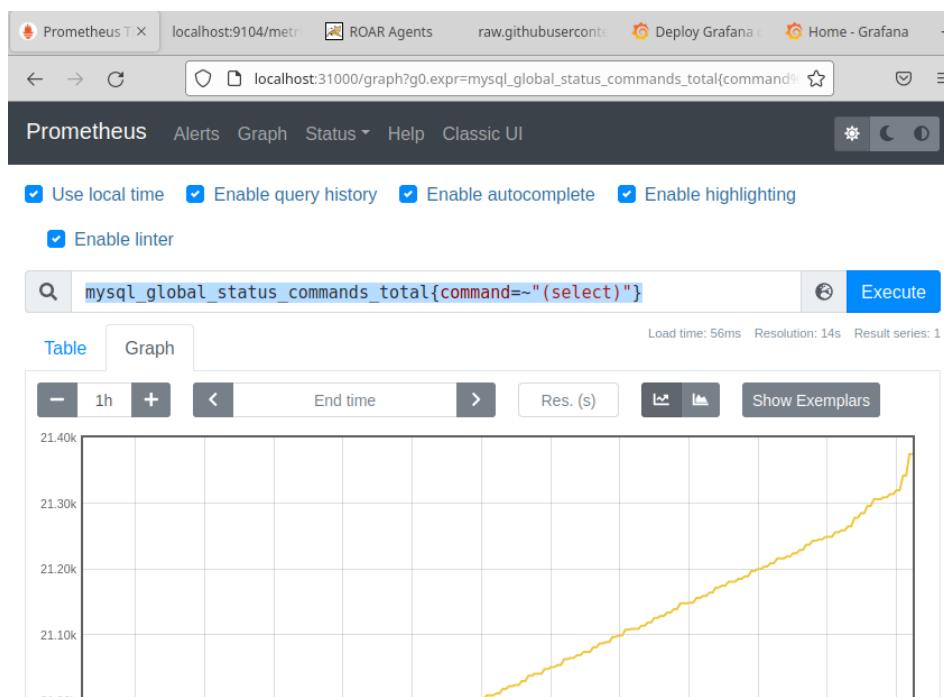
7. So far we have used counters in our queries - a value that increases (or can be reset to 0) as indicated by the "total" in the name. However, there are other kinds of time series such as "gauges" where values can go up or down. Let's see an example of one of those. Change your query to the expression below and then click the blue Execute button again.

node_memory_Active_bytes



8. Let's look at queries for another application. Suppose we want to monitor how much applications are referencing our database and doing "select" queries. We could use a mysql query to see the increase over time. Enter the query below in the query area and then click on **Execute**. A screenshot below shows what this should look like.

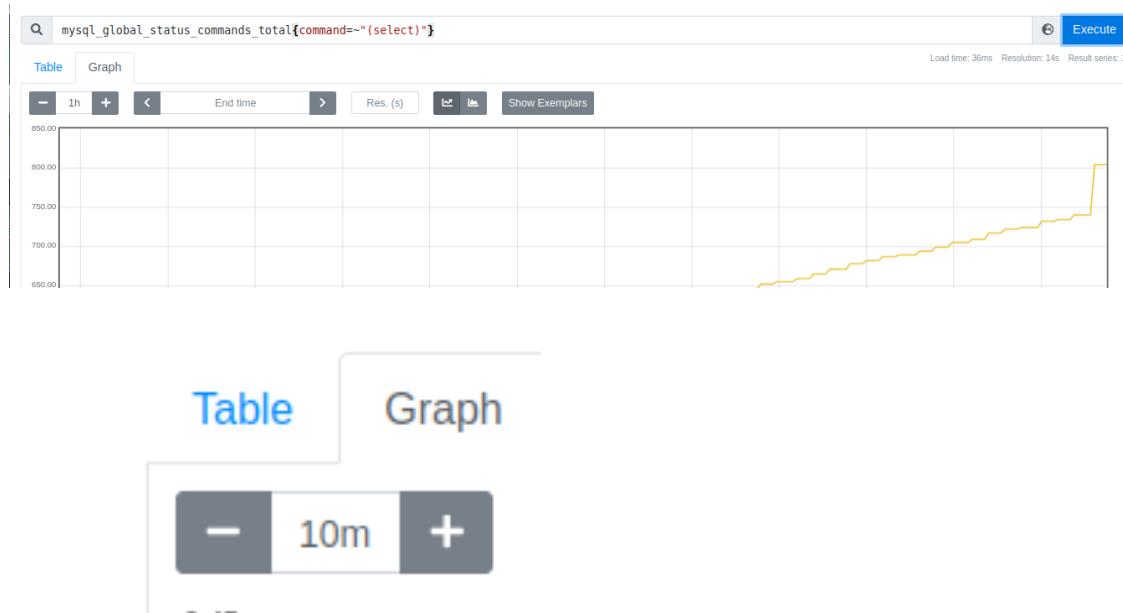
mysql_global_status_commands_total{command=~"(select)"}



9. Now let's simulate some query traffic to the database. I have a simple shell script that randomly queries the database in our application x times while waiting a certain interval between queries. It's called ping-db.sh. Run this in a terminal for 30 times with an interval of 1 second and then go back and refresh the graph again by clicking on the blue **Execute** button. (Note that you may need to wait a bit and refresh again to see the spike.)

```
$ ~/ping-db.sh roar 30 1
```

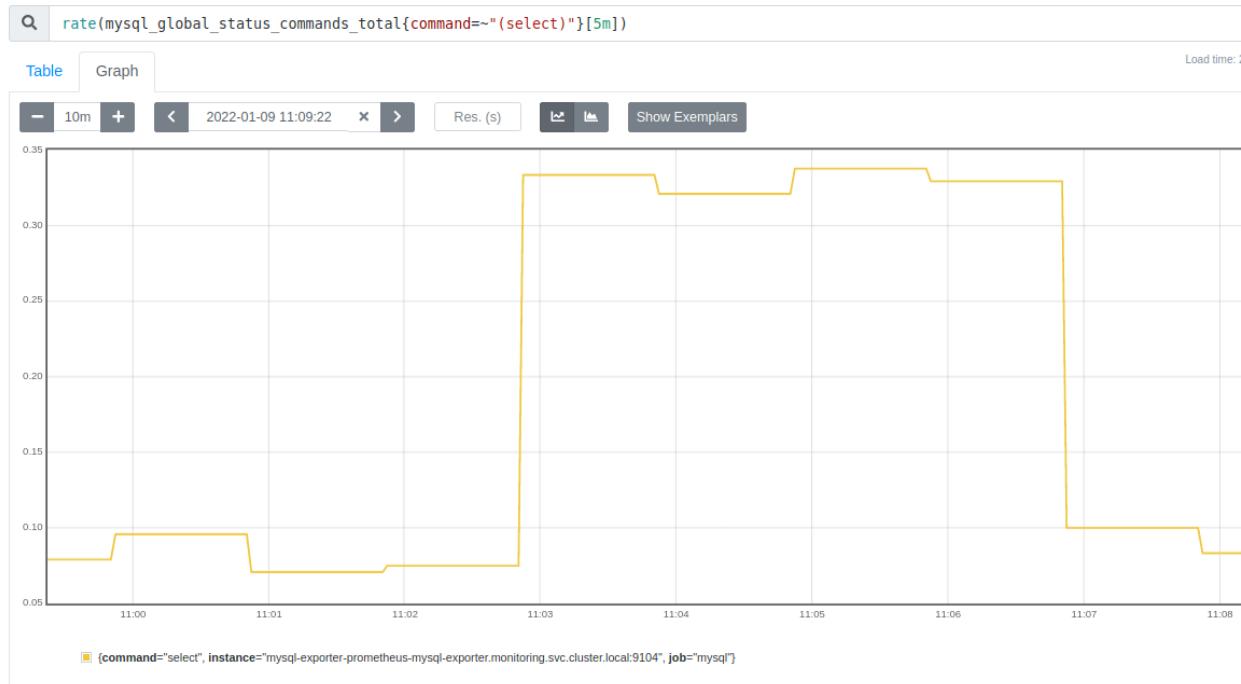
10. After clicking on the Execute button to refresh, you should see a small spike on the graph from our monitoring. (It may take a minute for the monitoring to catch up.) This is something we could key off of to know there was a load, but it will always just be an increasing value. Let's focus in on a smaller timeframe so we can see the changes easier. In the upper left of the Prometheus Graph tab, change the interval selector down to 10m. (Note that you can type in this field too.)



11. What we really need here is a way to detect any significant increase over a point in time regardless of the previous value. We can use the rate function we saw before for this. Change the query in Prometheus to be one that shows us the rate of change over the last 5 minutes and click on the **Execute** button again.

```
rate(mysql_global_status_commands_total{command=~"(select)"})[5m]
```

12. After clicking on the Execute button to refresh, you should see a different representation of the data. After you refresh, you'll be able to see that we no longer just see an increasing value, we can see where the highs and lows are.



END OF LAB

Lab 4 - Alerts and AlertManager

Purpose: In this lab, we'll see how to construct some simple alerts for Prometheus based on queries and conditions and use AlertManager to see them.

- Let's suppose that we want to get alerted when the "select" traffic spikes to high levels. We have a working "rate" query for our mysql instance gives us that information from the last lab. Take another look at that one to refresh your memory. Now let's change it to only show when our rate is above .35. And, let's also change it to use a scale of 0 to 100. We do this by multiplying the result by 100. Change the query to add the multiplier and "> 35" at the end and click on the blue "Execute" button. The query to use is shown below.

```
rate(mysql_global_status_commands_total{command=~"(select)"}[5m]) * 100 > 30
```

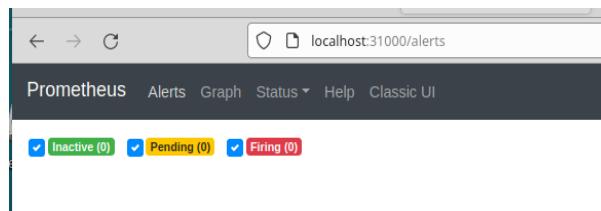
- After clicking on the Execute button to refresh, you will probably see an empty query result on the page. This is because we are targeting a certain threshold of data and that threshold hasn't been hit in the time range of the query (5 minutes). To have some data to look at, let's run our program to simulate the load again with the rate query in effect. Execute the same script we used before again with 30 iterations and a 2 second wait in-between.

```
$ ~/ping-db.sh roar 30 2
```

- After a couple of minutes and a couple of refreshes, you'll be able to see that we no longer just see an increasing value, we can see where the highs and lows are.



- While we can monitor by refreshing the graph and looking at it, it would work better to have an alert setup for this. Let's see what alerts we have currently. Switch to the alerts tab of Prometheus by clicking on "Alerts" in the dark bar at the top (or go to localhost:31000/alerts). Currently, you will not see any configured.



- Now let's configure some alert rules. We already have a configmap with some basic rules in it. gedit or cat the file extra/ps-cm-with-rules.yaml and look at the "alerting-rules.yml" definition under "data:".

```
$ cat (or gedit ) extra/ps-cm-with-rules.yaml
```

```

1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   labels:
5     app: prometheus
6     app.kubernetes.io/managed-by: Helm
7     chart: prometheus-15.0.2
8     component: server
9     heritage: Helm
10    release: pc
11  name: pc-prometheus-server
12  namespace: monitoring
13 data:
14   alerting_rules.yml: |->
15     groups:
16       - name: demo alerts
17         rules:
18           - alert: Cluster Memory Usage
19             expr: sum(container_memory_working_set_bytes{id="/"}) / sum(machine_memory_bytes{}) * 100 > 90
20             for: 5m
21             labels:
22               severity: page
23             annotations:
24               summary: Cluster Memory Usage > 90%
25             - alert: Cluster CPU Usage
26               expr: sum(rate(container_cpu_usage_seconds_total{id="/"}) [5m]) / sum(machine_cpu_cores{}) * 100 > 90
27               for: 5m
28               labels:
29                 severity: page
30               annotations:
31                 summary: "Cluster CPU Usage > 90%"
32               description: "Cluster CPU Usage on host {{\$labels.instance}} : (current value: {{ \$value }})."
33             - alert: Pod Memory usage
34               expr: sum(container_memory_working_set_bytes{image!

```

- Now, go ahead and apply that configmap definition. Then refresh your view of the Alerts and you should see a set of alerts that have been defined. (This is in the "extra" directory)

\$ k apply -n monitoring -f ps-cm-with-rules.yaml

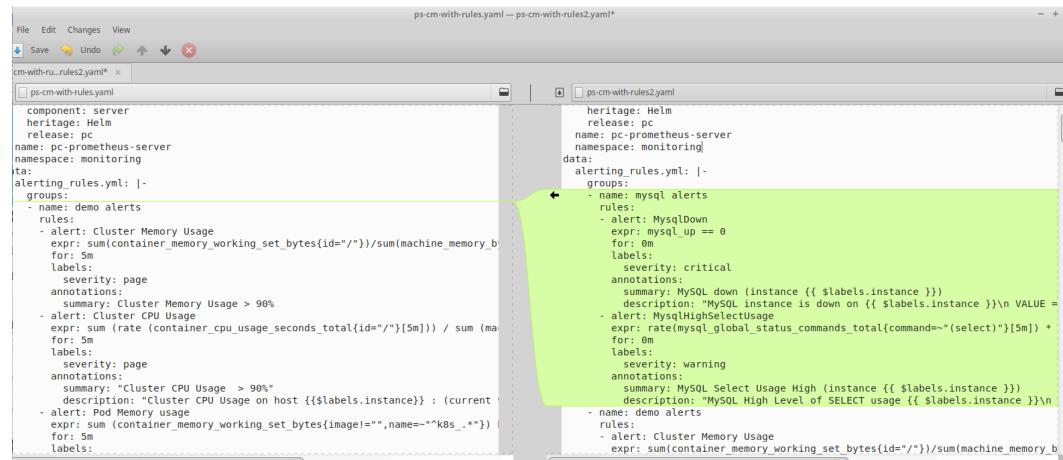
Alert Type	Status
Cluster Memory Usage	Inactive (0 active)
Cluster CPU Usage	Inactive (0 active)
Pod Memory usage	Inactive (0 active)

You can then expand those to see the alert definitions. Notice that each alert uses a PromQL query like we might enter in the main Prometheus query area.

Alert Type	PromQL Query
Cluster Memory Usage	name: Cluster Memory Usage expr: sum(container_memory_working_set_bytes{id="/"}) / sum(machine_memory_bytes{}) * 100 > 90 for: 5m labels: severity: page annotations: summary: Cluster Memory Usage > 90%
Cluster CPU Usage	name: Cluster CPU Usage expr: sum(rate(container_cpu_usage_seconds_total{id="/"}) [5m]) / sum(machine_cpu_cores{}) * 100 > 90 for: 5m labels: severity: page annotations: description: Cluster CPU Usage on host {{\\$labels.instance}} : (current value: {{ \\$value }}). summary: Cluster CPU Usage > 90%
Pod Memory usage	name: Pod Memory usage expr: sum by(container_label_io_kubernetes_pod_name) (container_memory_working_set_bytes{image!="",name=~"^k8s_.+"}) > 4.026531841e+09 for: 5m labels: severity: page annotations: description: Pod Memory usage on host {{\\$labels.instance}} : (current value: {{ \\$value }}). summary: Pod Memory usage > 90%

7. Let's add some custom alert rules for a group for mysql. These will follow a similar format as the other rules but using mysql PromQL queries, names, etc. There is already a file with them added - **ps-cm-with-rules2.yaml**. You can do a meld on that and our previous version of the cm data to see the new rules.

```
$ meld ps-cm-with-rules.yaml ps-cm-with-rules2.yaml
```



8. When you're done, just close the meld application. Now, let's apply the updated configmap manifest and add the new mysql alerts. Then refresh the Alerts view (and after a period of time) you should see the new mysql rules.

```
$ k apply -n monitoring -f ps-cm-with-rules2.yaml
```



9. Let's see if we can get our alert to fire now. Run our loading program to simulate the load again with the rate query in effect. Execute the same script we used before again with 60 iterations and a 0.5 second wait in-between.

```
$ ~/ping-db.sh roar 60 0.5
```

10. After this runs, after you refresh, on the Alerts tab, you should be able to see that the alert was fired. You can expand it to see details.

The screenshot shows the Prometheus UI at localhost:31000/alerts. The 'Alerts' tab is selected. There are two sections of alerts:

- /etc/config/alerting_rules.yml > demo alerts** (inactive): Contains three alerts: Cluster Memory Usage (0 active), Cluster CPU Usage (0 active), and Pod Memory usage (0 active).
- /etc/config/alerting_rules.yml > mysql alerts** (firing (1)): Contains one alert: MysqlDown (0 active). This alert is expanded to show its configuration:


```
name: MysqlHighSelectUsage
expr: rate(mysql_global_status_commands_total[command=~"(select)"]{5m}) * 100 > 35
labels:
  severity: warning
annotations:
  description: MySQL High Level of SELECT usage {{ $labels.instance }}
  VALUE = {{ $value }}
  LABELS = {{ $labels }}
  summary: MySQL Select Usage High (instance {{ $labels.instance }})
```

Labels	State	Active Since	Value
alertrname=MysqlHighSelectUsage command=select instance=mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104 job=mysql severity=warning	FIRING	2022-01-09T19:19:47.065791371Z	51.11111111111112

Annotations

```
description
MySQL High Level of SELECT usage mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104 VALUE = 51.11111111111112 LABELS = map[command:select instance:mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104 job:mysql]
summary
MySQL Select Usage High (instance mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104)
```

11. Now, that our alert has fired, we should be able to see it in the Alert Manager application. On this machine, it is exposed at node port 31500. Open up that location and take a look.

The screenshot shows the Alertmanager UI at localhost:31500/#/alerts. The 'Alerts' tab is selected. A single alert is listed:

- Not grouped**: 1 alert
- 2022-01-09T19:19:47.065Z**: + Info ↗ Source ✖ Silence
- alertrname="MysqlHighSelectUsage"**: +
- command="select"**: +
- instance="mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104"**: +
- job="mysql"**: +
- severity="warning"**: +

12. If you click on the "Source" link next to the timestamp and "+ Info" it will take you back to the main Prometheus query screen. When you get there, click on Execute and open up the Graph view.



The screenshot shows the Prometheus web interface. At the top, there are several checkboxes: 'Use local time' (checked), 'Enable query history' (checked), 'Enable autocomplete' (checked), 'Enable highlighting' (checked), and 'Enable linter' (checked). Below the checkboxes is a search bar containing the query: `rate(mysql_global_status_commands_total{command=~"(select)"}[5m]) * 100 > 35`. To the right of the search bar is an 'Execute' button. Under the search bar, there are two tabs: 'Table' (selected) and 'Graph'. Below the tabs is an 'Evaluation time' dropdown with arrows. The main content area displays the results of the query, showing a single row: `(command="select", instance="mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104", job="mysql")`. At the bottom right of the results area is a 'Remove Panel' link.

13. You can also Silence alerts for some period of time. Go back to the Alert Manager screen. Click on the Silence icon and enter the information for a temporary silence, such as 10m. You'll also need to add a Creator (author) and Comment for the silence. Then you can click on the Create button to save your changes.

The screenshot shows the Alertmanager 'Silences' page with a 'New Silence' form. The browser address bar shows `localhost:31500/#/silences/new?filter={alertname%3D"MySqlHighSelectUsage"%2C command%3D"select"%2C severity%3D"warning"}`. The page has a header with links for Alertmanager, Alerts, Silences, Status, and Help. The main section is titled 'New Silence'. It contains fields for 'Start' (2022-01-09T19:32:07.516Z), 'Duration' (10m), and 'End' (2022-01-09T19:42:07.516Z). Below these are 'Matchers' for alerts affected by the silence, including alertname="MySqlHighSelectUsage", command="select", Instance="mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104", job="mysql", and severity="warning". There is a 'Custom matcher, e.g. env="production"' field with a '+' button. The 'Creator' field is set to 'User 1'. The 'Comment' field contains 'Demo silence'. At the bottom are buttons for 'Preview Alerts' (green), 'Create' (blue), and 'Reset' (red).

14. You'll then have a new Silence saved. You can Expire it in advance if needed, but while its active, if you repeat the load example, you should not get alerted in Alert Manager.

The screenshot shows a browser window with the URL `localhost:31500/#/silences/97ab3bff-3dcd-4e5e-8ca9-bd5ec26ec9de`. The page title is "Silence". The main content displays the following details:

- ID:** 97ab3bff-3dcd-4e5e-8ca9-bd5ec26ec9de
- Starts at:** 2022-01-09T19:35:45.729Z
- Ends at:** 2022-01-09T19:42:07.516Z
- Updated at:** 2022-01-09T19:35:45.729Z
- Created by:** User 1
- Comment:** Demo silence
- State:** active
- Matchers:**
 - alertname=MysqlHighSelectUsage
 - command=select
 - instance=mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104
 - job=mysql
 - severity=warning

Below the details, it says "No affected alerts".

END OF LAB

Lab 5 - Grafana

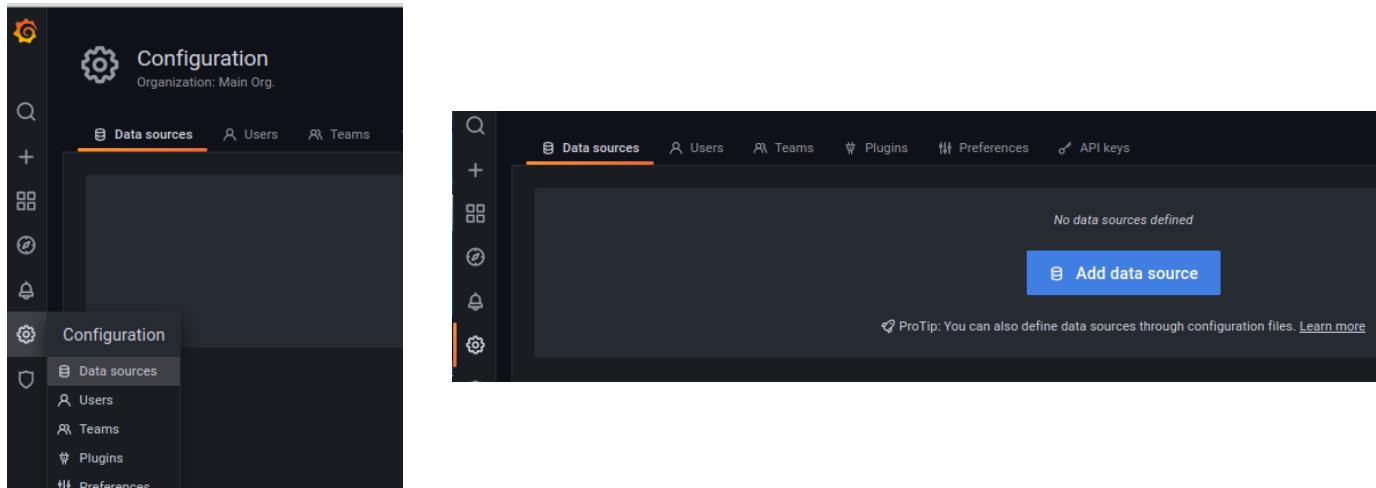
Purpose: In this lab, we'll see how to use Grafana to display custom graphs and dashboards for Prometheus data.

1. We already have an instance of Grafana running on this system. Open up a browser to the home page at <http://localhost:31750>. If you are prompted about changing your password, just select to skip that option. The default admin userid and password are both "admin".

The screenshot shows the Grafana home dashboard with the following sections:

- Welcome to Grafana:** A central panel with links to "TUTORIAL", "DATA SOURCE AND DASHBOARDS", and "Grafana fundamentals". It also includes a "Basic" sidebar with instructions for setting up Grafana.
- DATA SOURCES:** A panel titled "Add your first data source" with a "Create" button and a link to "Learn how in the docs".
- DASHBOARDS:** A panel titled "Create your first dashboard" with a "Create" button and a link to "Learn how in the docs".
- Dashboards:** A section showing "Starred dashboards" and "Recently viewed dashboards".
- Latest from the blog:** A section featuring a blog post by "Viktor Tasevski" with the title "Joining a company in its hypergrowth phase comes with its own challenges and looking back, I'm really grateful for the support I've received from Day 1".
- Jan 06:** A post titled "Building an effective remote-first team during the pandemic".
- Jan 05:** A post titled "I'm an engineering manager at Grafana Labs serving on the Grafana Enterprise Operations team. I joined Grafana Labs in December 2020 and I just celebrated my first year at the company. The last 12+ months have been filled with the most exciting and rewarding experiences in my career, full of new opportunities and learnings. More importantly, I am lucky enough to meet and work with the wonderful people at Grafana Labs."
- Jan 05:** A post titled "Don't miss our webinars on Grafana Tempo, Grafana Enterprise Traces, and the new Sentry plugin".

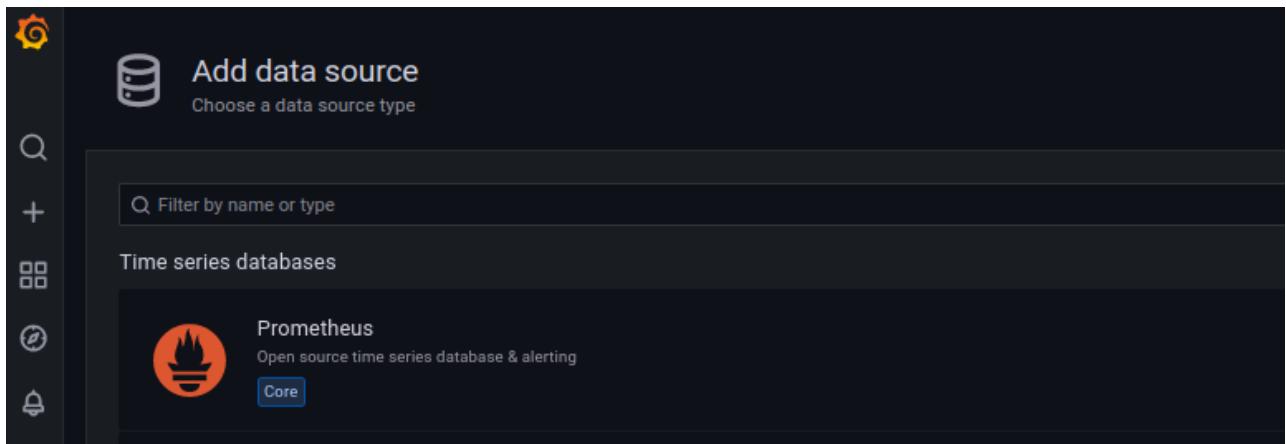
- Let's first add our Prometheus instance as a Data Source. Hover over or click on the "gear icon" on the side and select "Data Sources". Then click on the blue button for "Add data source".



- Select "Prometheus" and then for the HTTP URL field, enter

<http://pc-prometheus-server.monitoring.svc.cluster.local:80>

Then click on "Save and Test". After a moment, you should get a response that indicates the data source is working.

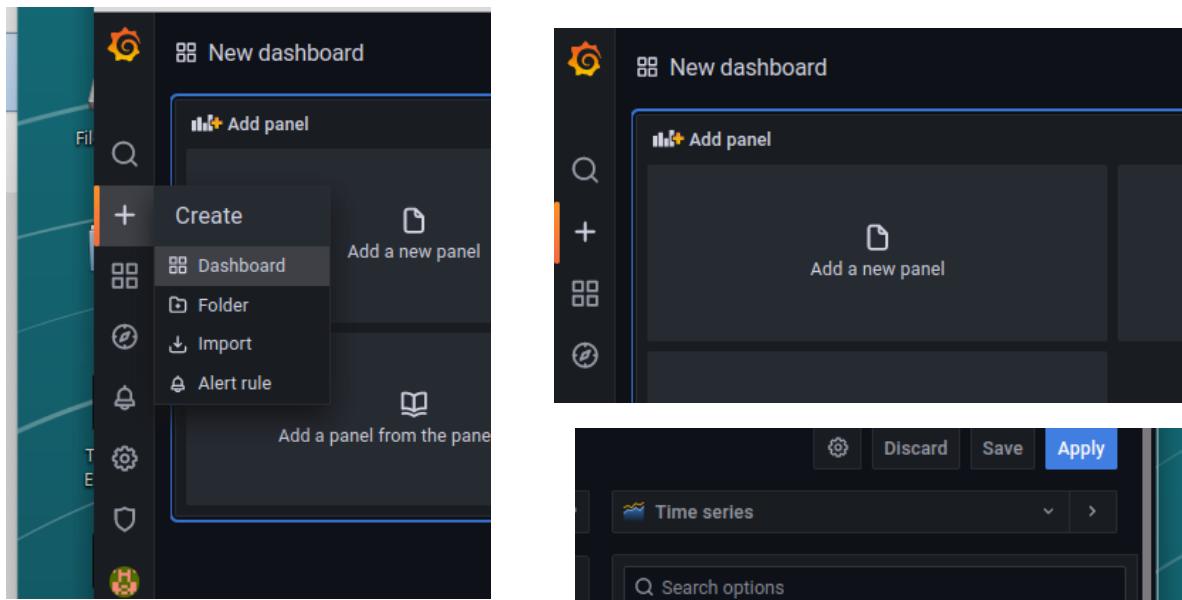


The screenshot shows the Grafana interface for managing data sources. The top navigation bar indicates the URL is `localhost:31750/datasources/edit/F2bVzWJnk`. The main title is "Data Sources / Prometheus" with a "Type: Prometheus" subtitle. On the left, a sidebar menu includes icons for Home, Settings, Dashboards, and others. The "Settings" tab is active. A table lists a single data source entry:

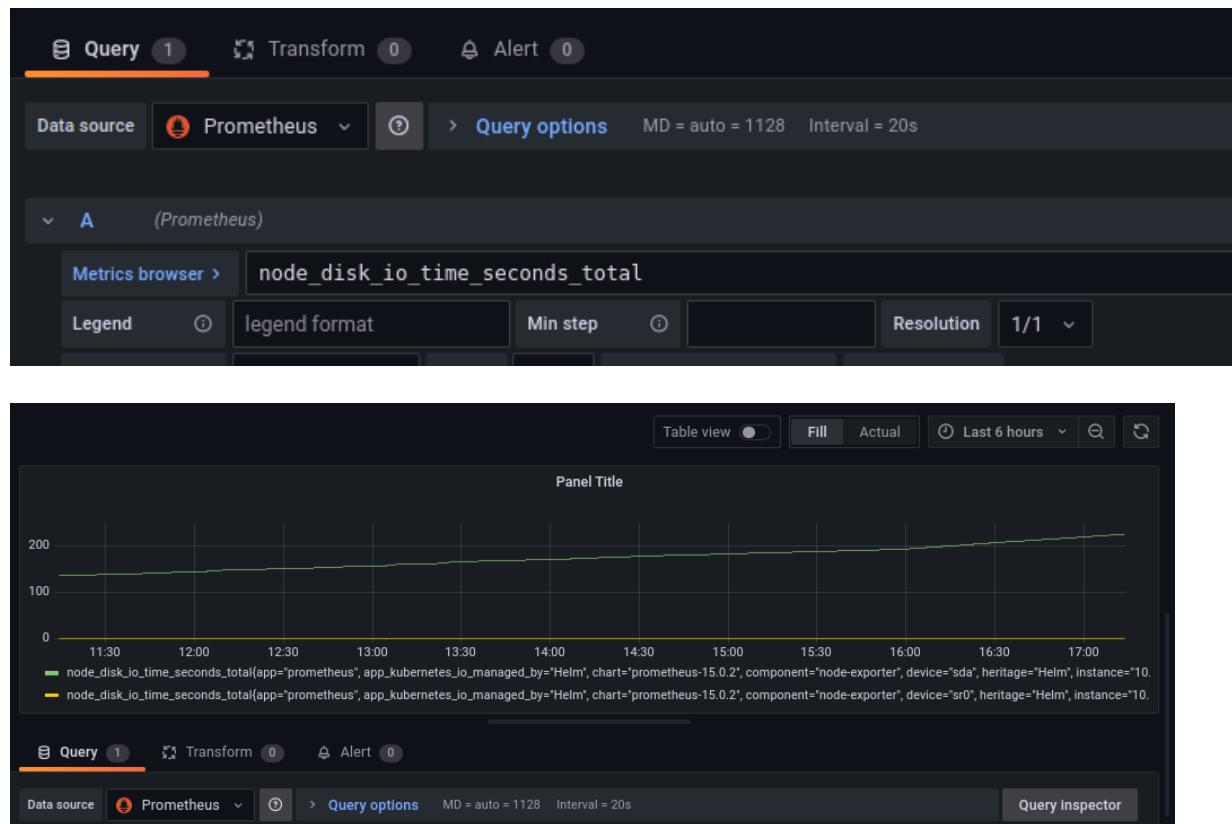
Name	URL	Access	Allowed cookies
Prometheus	rometheus-server.monitoring.svc.cluster.local:80	Server (default)	New tag (enter key to add)

Below the table, the "Exemplars" section contains buttons for Back, Explore, Delete (highlighted in pink), and Save & test. A success message "Data source is working" is shown with a green checkmark icon. At the bottom, there are Back, Explore, Delete, and Save & test buttons.

4. Now, let's create a simple dashboard for one of our mysql metrics. Click on the "+" sign on the left and select Dashboard from the menu. Then click on "Add a new panel". In the upper right, make sure "Time series" is selected for the type of visualization.



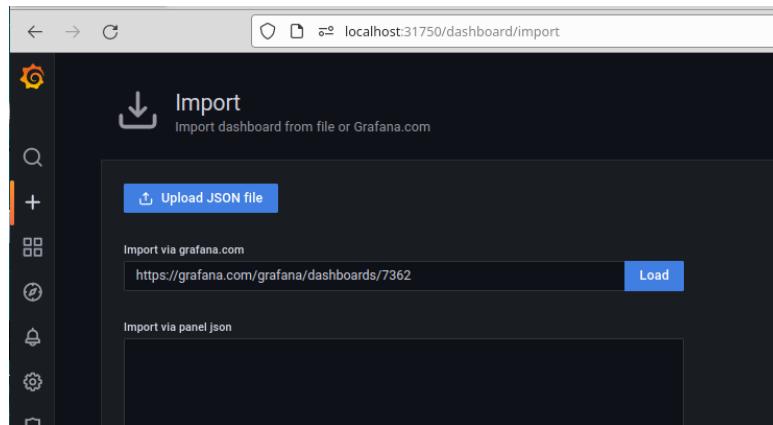
5. In the Metrics Browser section, start typing "node" and then pick a sample metric, such as "node_disk_io_time_seconds_total". Then click in the Panel and you should see a new chart.



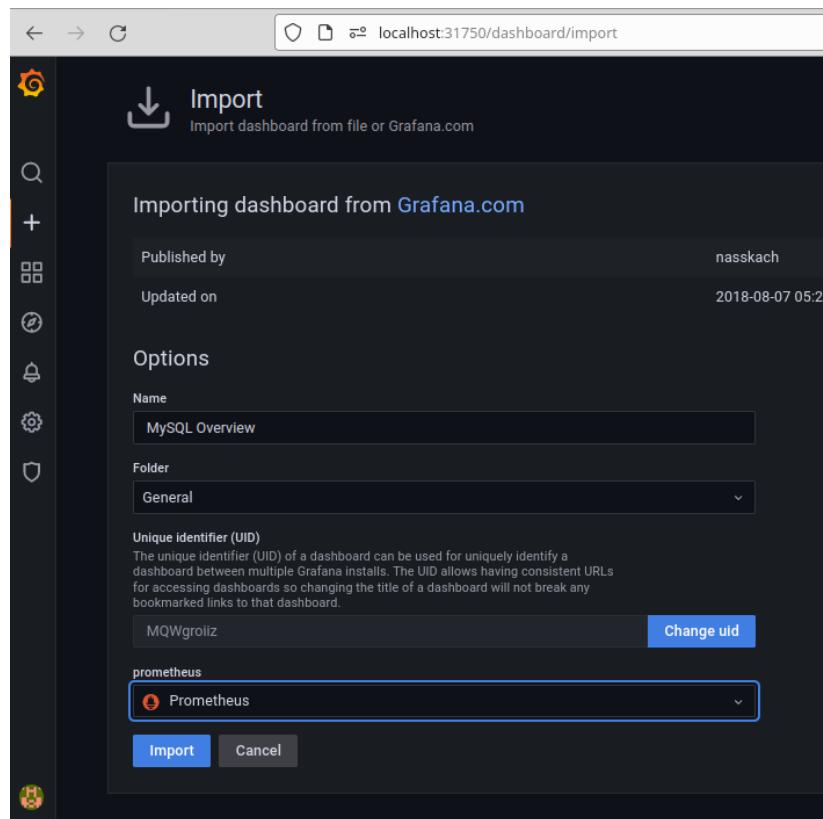
6. While we can create individual dashboards with Grafana, that can take a lot of time and effort. The community has already created a number of dashboards that we can just import and use.

Let's grab one for mysql. Back on the main page, click on the "+" on the left side, then select "Import". (You can save or discard the previous one if prompted.) In the field that says "Grafana.com dashboard URL or ID", enter the location below and click the blue "Load" button.

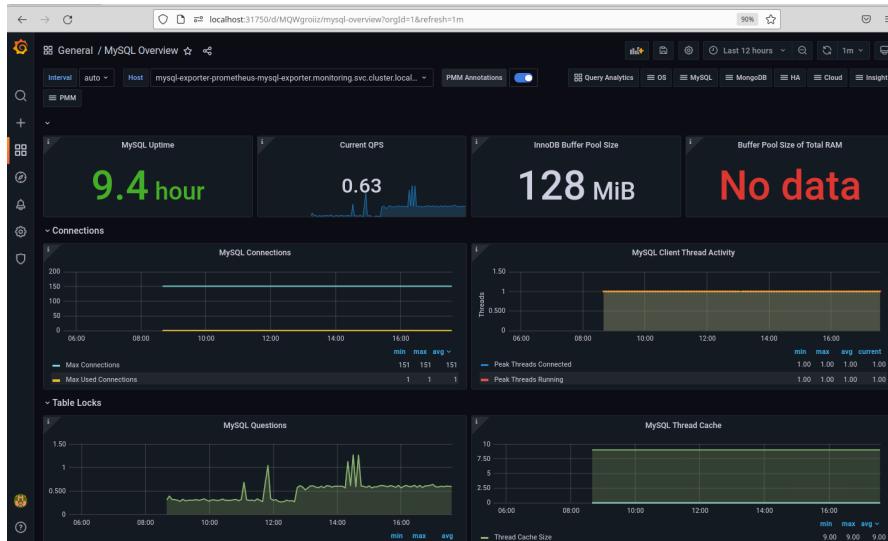
<https://grafana.com/grafana/dashboards/7362>



7. On the next page, you can leave everything as-is, except at the bottom for the Prometheus source, click in that box and select our default Prometheus data source that we setup. Then click the blue "Import" button at the bottom.

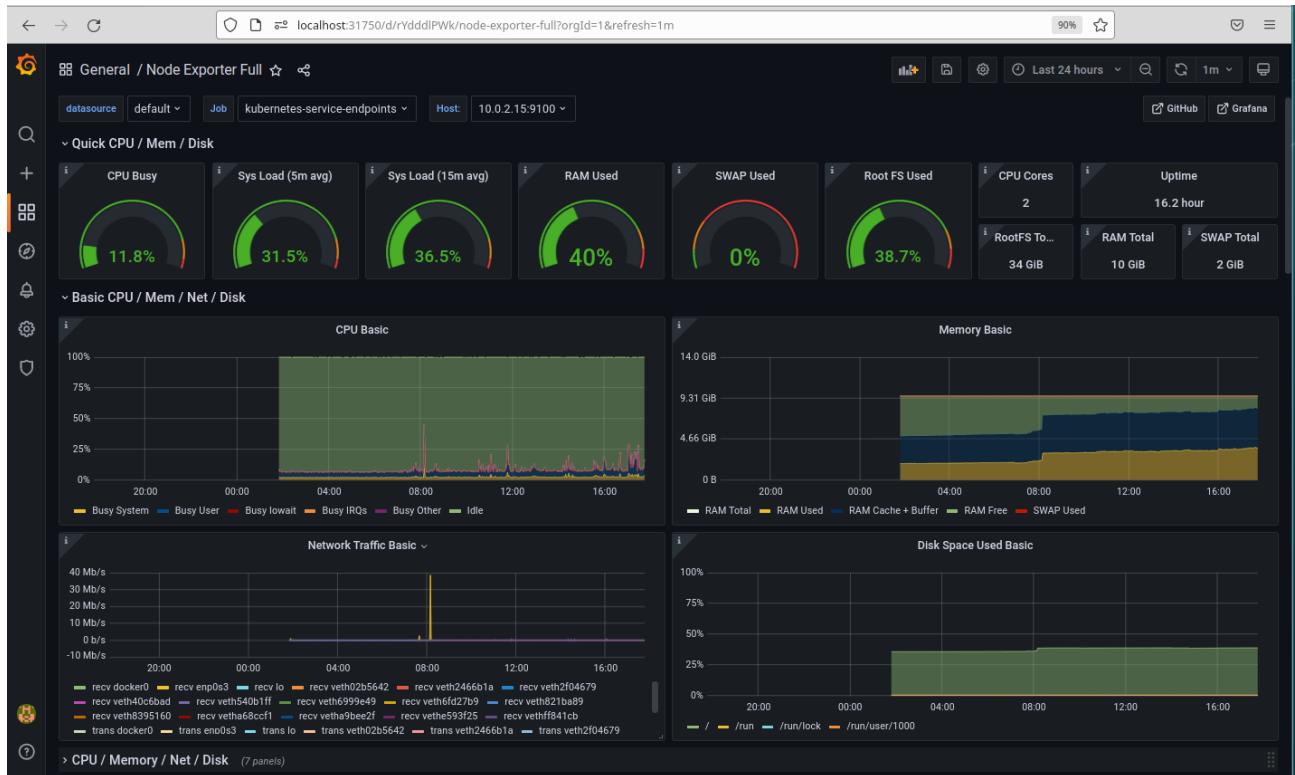


8. At this point, you should see a populated dashboard with a number of panels looking at the mysql exporter data from our system through Prometheus. You can scroll around and explore if you want.



9. Another cool one to import (via the same process) is the "Node Exporter Full" one. It's available from the link below. A screenshot is also included.

<https://grafana.com/grafana/dashboards/1860>



END OF LAB