

Getting Started with Prometheus

Monitoring Kubernetes infrastructure and applications for reliability

Class Labs

Version 2.2 by Brent Lester on behalf of Tech Skills Transformations

11/01/2023

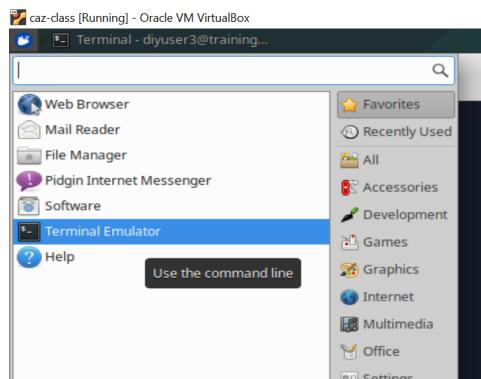
Important Prereq: These labs assume you have already followed the instructions in the separate setup document and have VirtualBox up and running on your system and have downloaded the *prom-start2.ova* file and loaded it into VirtualBox. If you have not done that, please refer to the setup document for 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. Note: At the end of this running, you will see some error messages about a pod not being reading and it "waiting for pod". This is normal and it should finish shortly.

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

Lab 1 - Monitoring with Prometheus

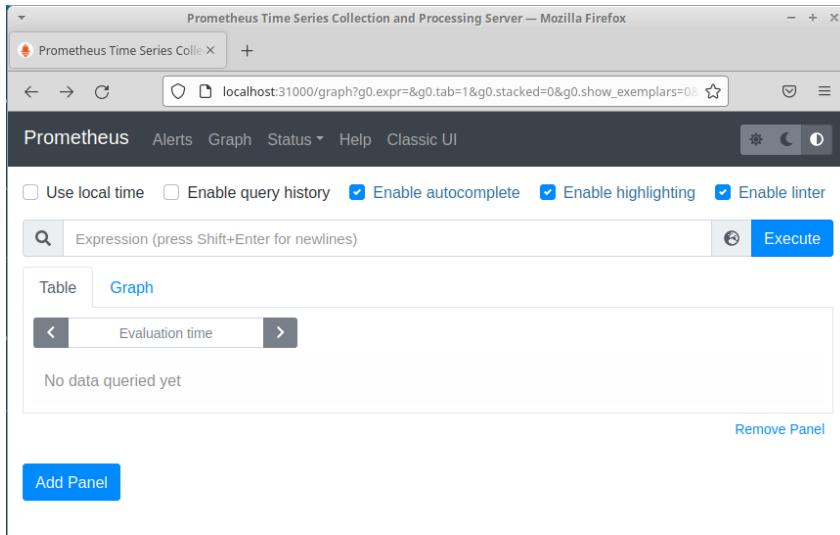
Purpose: In this lab, we'll run an instance of the node exporter and use Prometheus to surface basic data and metrics.

1. 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
```

2. 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/>



3. 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="veth74ffa2"} 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="veth82e4cc"} 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="veth0e455a"} 1
node_network_protocol_type{device="veth74ffa2"} 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="veth82e4cc"} 0
node_network_receive_bytes_total{device="veth83274df"} 5.52887e+06

```

4. 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.

The screenshot shows the Prometheus Targets page in Mozilla Firefox. The URL is `localhost:31000/targets`. The page has a header with tabs for Prometheus, Alerts, Graph, Status, Help, and Classic UI. Below the header, there's a section titled "Targets" with buttons for All, Unhealthy, and Collapse All. There are three main sections: "kubernetes-apiservers (1/1 up)", "kubernetes-nodes (1/1 up)", and "kubernetes-nodes-cadvisor (1/1 up)". Each section has a "show less" button.

kubernetes-apiservers (1/1 up)

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

kubernetes-nodes (1/1 up)

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

kubernetes-nodes-cadvisor (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error

- 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
```

- 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.)

Prometheus Alerts Graph Status Help Classic UI

kubernetes-pods (1/1 up) show more
kubernetes-service-endpoints (3/3 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
----------	-------	--------	-------------	-----------------	-------

Prometheus Time Series Collection and Processing Server — Mozilla Firefox

Prometheus Time Series Collection localhost:31800/metrics

Prometheus Targets localhost:31000/classic/targets

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.17.0.10:9100/metrics	UP	app_kubernetes_io_instance="traefik" app_kubernetes_io_managed_by="Helm" app_kubernetes_io_name="traefik" helm_sh_chart="traefik-10.9.1"	55.615s ago	2.167ms	

traefik

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

172.17.0.10:9100/metrics localhost:31800/metrics

172.17.0.10:9100/metrics

```

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

```

traefik

7. 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 Collector localhost:9100/metrics

localhost:31000/service-discovery

Prometheus Alerts Graph Status Help Classic UI

- 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

Highlight All Match Case Match Diacritics Whole Words Phrase not found

- 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
```

Meld File Edit Changes View

ps-cm-start.yaml — ps-cm-with-traefik.yaml

ps-cm-start...h-traefik.yaml

ps-cm-startyaml

ps-cm-with-traefikyaml

```

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.io/serviceaccount/token
          - job_name: kubernetes-apiservers
            kubernetes_sd_configs:
              - type: Endpoints
                endpoint: kube-apiserver

```

```

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
            bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      - job_name: prometheus
        static_configs:
          - targets:
              - localhost:9090
            bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token

```

Ln 18, Col 1 INS

9. 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
```

10. 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	

Target	Status	Labels	Scrape Duration	Age
traefik (1 / 1 active targets)	1 / 1 active targets			
kubernetes-nodes-cadvisor (1 / 1 active targets)	1 / 1 active targets			
kubernetes-pods (1 / 21 active targets)	1 / 21 active targets			
kubernetes-service-endpoints (3 / 13 active targets)	3 / 13 active targets			
prometheus (1 / 1 active targets)	1 / 1 active targets			
prometheus-pushgateway (1 / 14 active targets)	1 / 14 active targets			
traefik (1 / 1 active targets)	1 / 1 active targets			
kubernetes-pods-slow (0 / 24 active targets)	0 / 24 active targets			
kubernetes-service-endpoints-slow (0 / 15 active targets)	0 / 15 active targets			
kubernetes-services (0 / 14 active targets)	0 / 14 active targets			

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.

```
$ cd ~/prom-start/roar-k8s
```

```
$ 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> .

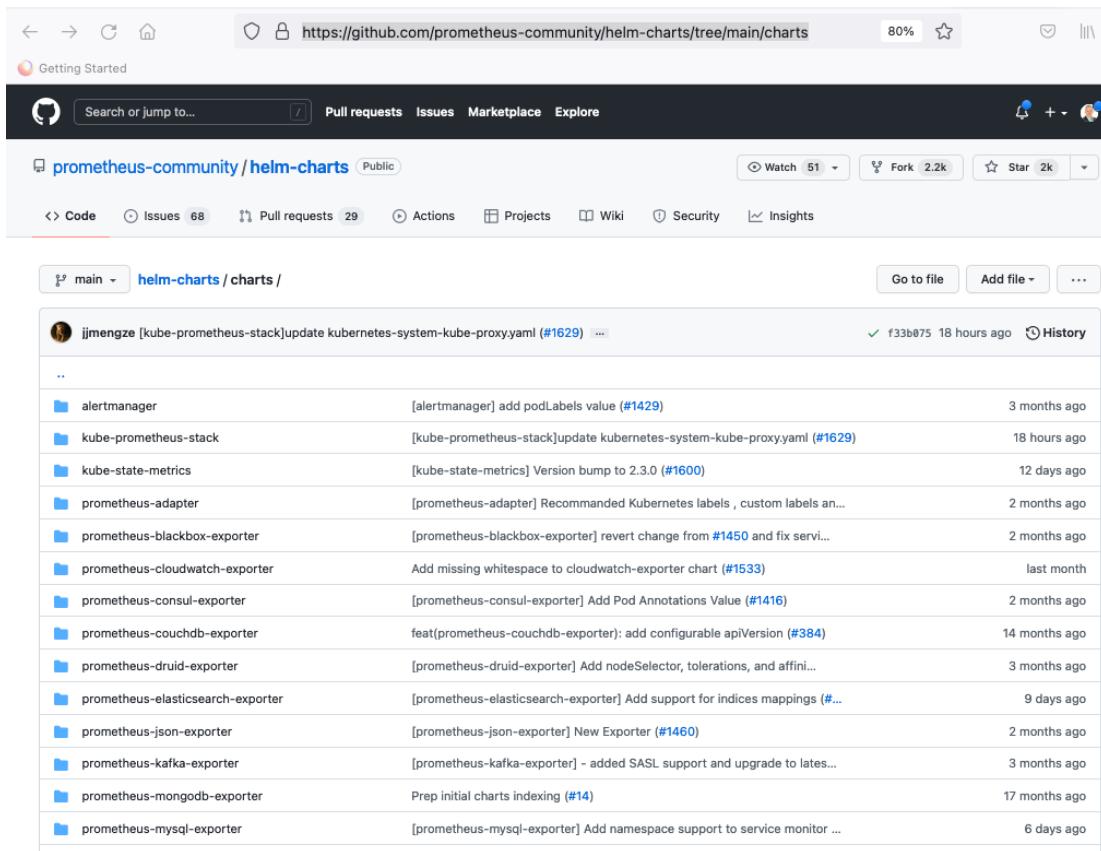
```
$ k get all -n roar
```

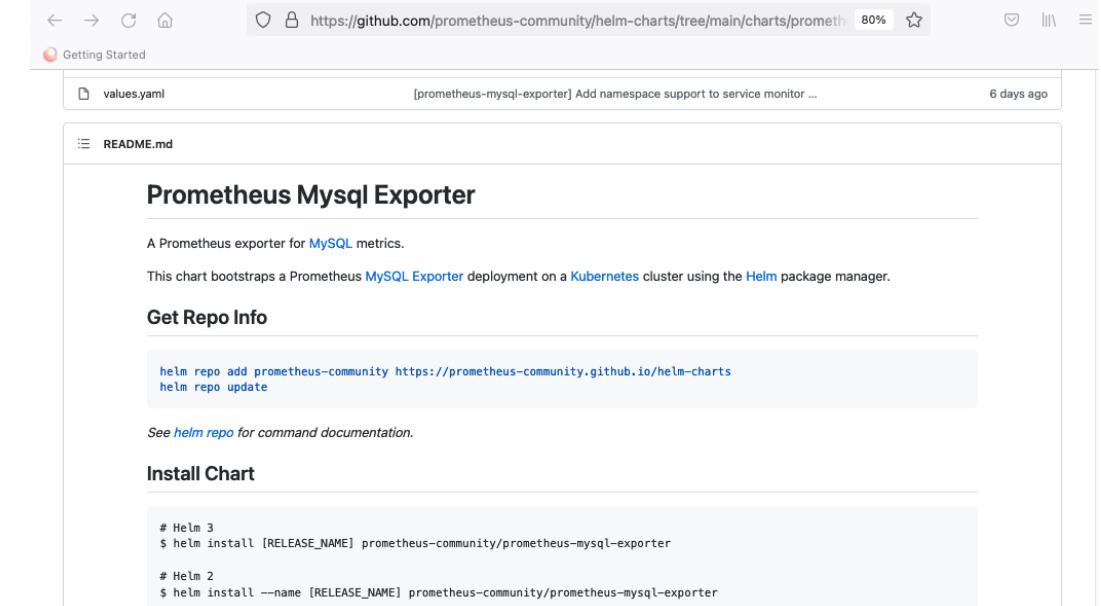
	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	Perry	platypus	2013-01-20	2015-04-09	H. Doofenshmirtz	...inator
4	Mr. Krabs	crab	2010-06-17	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

3. 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.

A screenshot of a GitHub repository page. The URL is https://github.com/prometheus-community/helm-charts/tree/main/charts. The repository name is "prometheus-community / helm-charts" and it is public. There are 68 issues, 29 pull requests, and 2k stars. The main branch is "main". A list of pull requests is shown, with the most recent being "update kubernetes-system-kube-proxy.yaml" by jjmengze, opened 18 hours ago.

A screenshot of a GitHub repository page for the "prometheus-mysql-exporter" chart. The URL is https://github.com/prometheus-community/helm-charts/tree/main/charts/prometheus-mysql-exporter. The README.md file is displayed, which includes instructions for getting repo info and installing the chart using Helm commands.

- 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)

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
```

```

diff --git a/ps-cm-with-traefik.yaml b/ps-cm-with-mysql.yaml
--- a/ps-cm-with-traefik.yaml
+++ b/ps-cm-with-mysql.yaml
@@ -1,10 +1,10 @@
 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:80
- job_name: prometheus
  static_configs:
  - targets:
    - localhost:9090
- bearer_token file: /var/run/secrets/kubernetes-api-servers
  job_name: kubernetes-apiservers
  kubernetes_sd_configs:

```

8. Now you can apply the updated configmap definition.

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

9. 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).)

Targets Page:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mysql-exporter-prometheus.mysql-exporter.monitoring.svc.cluster.local:9104/metrics	Up	instances="mysql-exporter-prometheus mysql-exporter.monitoring.svc.cluster.local:9104" job="mysql"	39.483s ago	70.044ms	

Service Discovery Page:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
mysql					

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
```

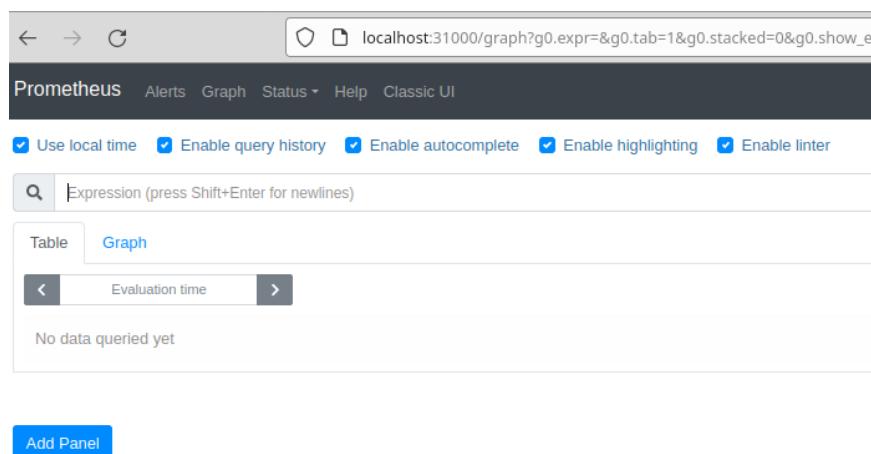
```
go_memstats_stack_sys_bytes 425984
# HELP go_memstats_sys_bytes Number of bytes obtained from system.
# TYPE go_memstats_sys_bytes gauge
go_memstats_sys_bytes 7.4269704e-07
# HELP go_threads Number of OS threads created.
# TYPE go_threads gauge
go_threads 5
# HELP mysql_exporter_collector_duration_seconds 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.



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 Prometheus Metrics Explorer interface. At the top, there are several checkboxes: 'Use local time', 'Enable query history', and 'Enable autocomplete'. Below these is a search bar labeled 'Expression (press Shift+Enter for newlines)'. Underneath the search bar are two tabs: 'Table' and 'Graph', with 'Table' currently selected. A dropdown menu labeled 'Evaluation time' is open. The main area displays a list of metrics, with the first few being: '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'. In the top right corner of the interface, there is a red circle highlighting the 'Execute' button.

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.

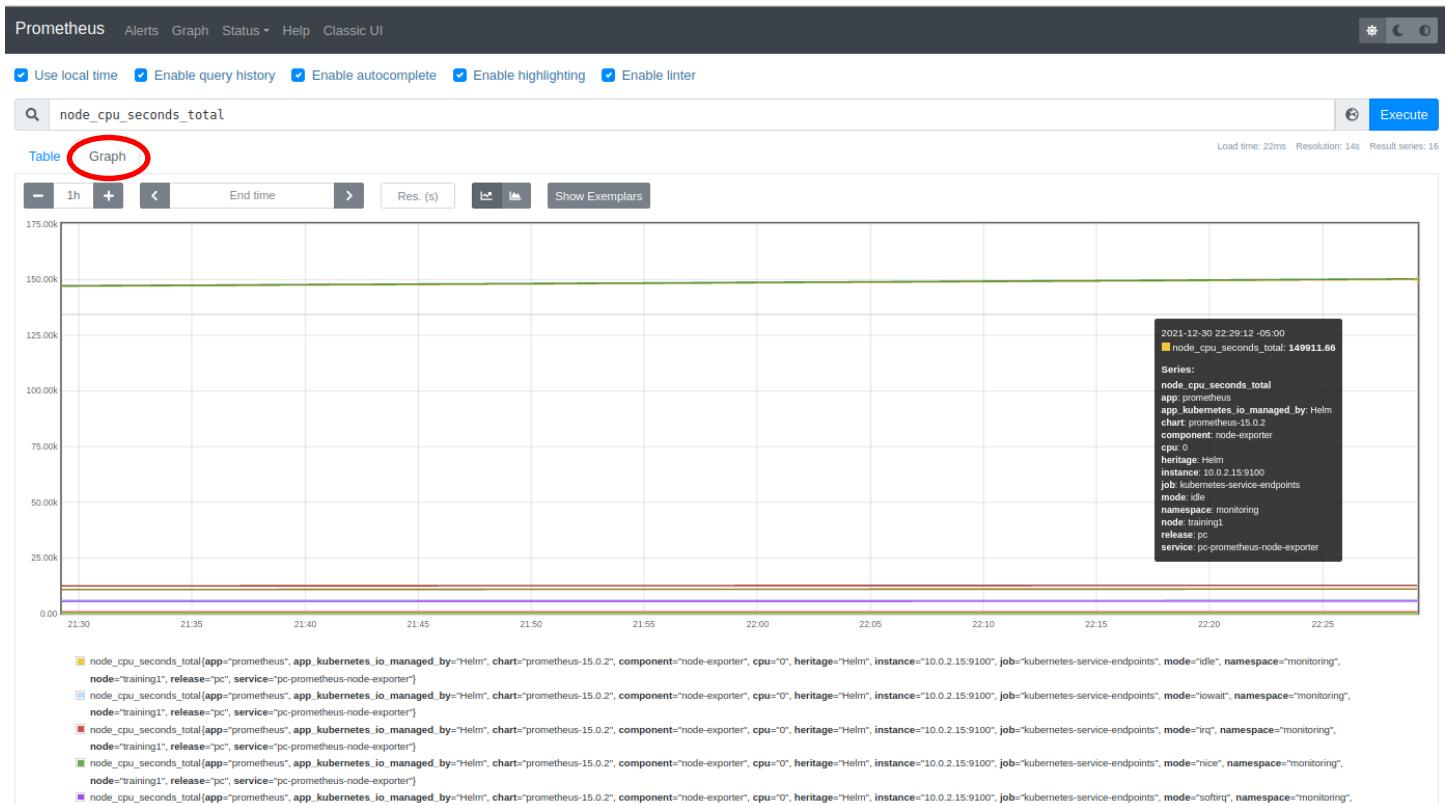
The screenshot shows the Prometheus Metrics Explorer interface with the search bar containing 'con'. The results list includes: 'container_blkio_device_usage_total', 'container_cpu_load_average_10s', 'container_cpu_system_seconds_total', 'container_cpu_usage_seconds_total', 'container_cpu_usage_seconds_total', 'container_cpu_user_seconds_total', 'container_file_descriptors', 'container_fs_inodes_free', 'container_fs_inodes_total', 'container_fs_io_current', and 'container_fc_in_time_seconds_total'. To the right of the results, a tooltip for 'container_blkio_device_usage_total' is displayed, stating 'Blkio Device bytes usage' and 'counter'. Other metrics in the list have similar descriptions: 'gauge', 'counter', 'counter', 'past query', 'counter', 'gauge', 'gauge', 'gauge', and 'counter'.

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.

The screenshot shows the Prometheus interface with the search bar containing 'node_cpu_seconds_total'. The 'Table' tab is selected. The results show multiple rows of data with various metrics like app, component, chart, and mode. The 'Execute' button is circled in red.

app	component	chart	cpu	heritage	instance	job	mode	namespace	node	release	service
prometheus	node-exporter	prometheus-15.0.2	0	Helm	10.0.2.15:9100	kubernetes-service-endpoints	idle	monitoring			pc-prometheus-node-exporter
kubernetes.io	node-exporter	prometheus-15.0.2	0	Helm	10.0.2.15:9100	kubernetes-service-endpoints	lowlat	monitoring	training1	pc	pc-prometheus-node-exporter
kubernetes.io	node-exporter	prometheus-15.0.2	0	Helm	10.0.2.15:9100	kubernetes-service-endpoints	irq	monitoring	training1	pc	pc-prometheus-node-exporter
kubernetes.io	node-exporter	prometheus-15.0.2	0	Helm	10.0.2.15:9100	kubernetes-service-endpoints	nice	monitoring	training1	pc	pc-prometheus-node-exporter
kubernetes.io	node-exporter	prometheus-15.0.2	0	Helm	10.0.2.15:9100	kubernetes-service-endpoints	softirq	monitoring	training1	pc	pc-prometheus-node-exporter
kubernetes.io	node-exporter	prometheus-15.0.2	0	Helm	10.0.2.15:9100	kubernetes-service-endpoints	steal	monitoring	training1	pc	pc-prometheus-node-exporter

- 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"}
```



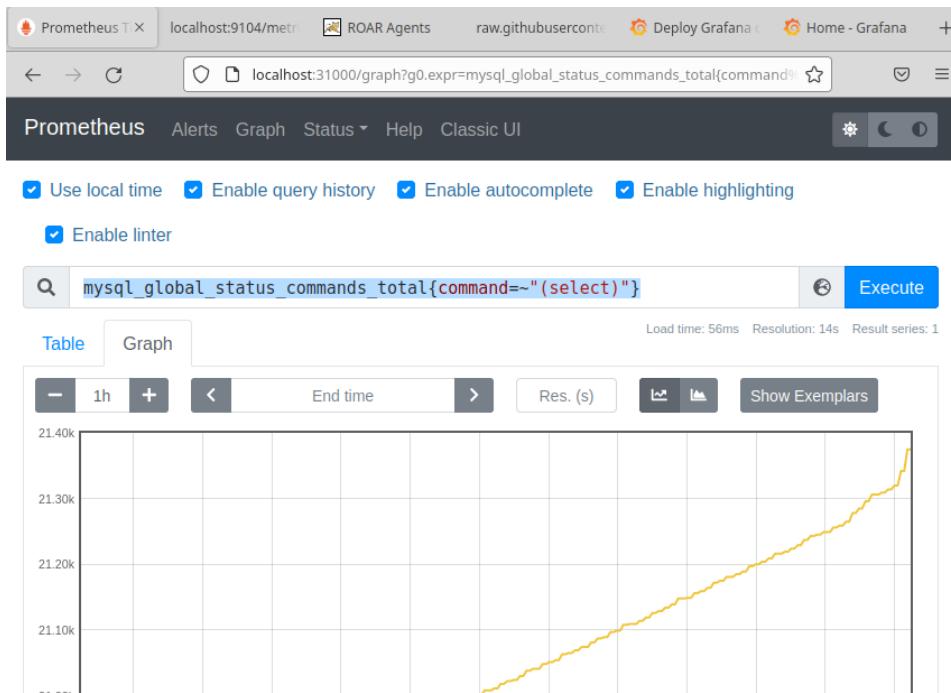
7. So far we have used counters in our queries - a value that increases (or can be reset to 0) as indicated by the "total" in the name. However, there are other kinds of time series such as "gauges" where values can go up or down. Let's see an example of one of those. Change your query to the expression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

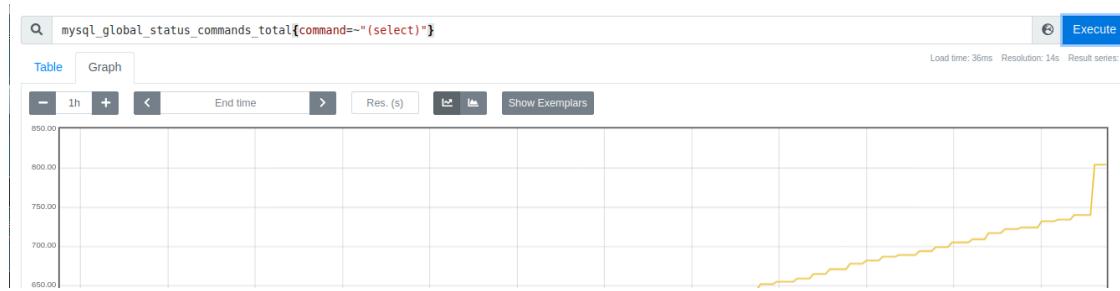
```
mysql_global_status_commands_total{command=~"(select)"}
```



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.)

```
$ ~/prom-start/extra/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 which 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 > 35
```

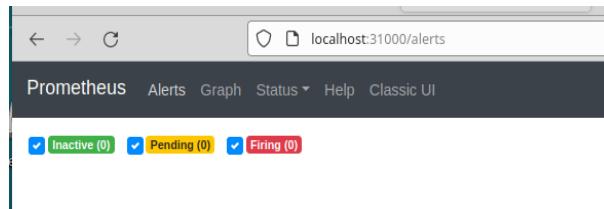
- After clicking on the Execute button to refresh, you will probably see an empty query result on the page. This is because we are targeting a certain threshold of data and that threshold hasn't been hit in the time range of the query (5 minutes). To have some data to look at, let's run our program to 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.

```
$ ~/prom-start/extra/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. cat the file extra/ps-cm-with-rules.yaml with the grep command and look at the "alerting-rules.yml" definition under "data:".

```
$ cd ~/prom-start/extra
```

```
$ cat ps-cm-with-rules.yaml | grep -A20 alerting_rules.yml:
```

```

apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: prometheus
    app.kubernetes.io/managed-by: Helm
    chart: prometheus-15.0.2
    component: server
    heritage: Helm
    release: pc
  name: pc-prometheus-server
  namespace: monitoring
data:
  alerting_rules.yml: |
    groups:
      - name: demo alerts
        rules:
          - alert: 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%
          - alert: 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:
              summary: "Cluster CPU Usage > 90%"
              description: "Cluster CPU Usage on host {{\$labels.instance}} : (current value: {{ \$value }})."
          - alert: Pod Memory usage
            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.

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

The screenshot shows the Prometheus Alerts interface at localhost:31000/alerts. The top navigation bar includes Prometheus, Alerts, Graph, Status, Help, and Classic UI. Below the navigation is a search bar with the path /etc/config/alerting_rules.yml > demo alerts. A green button labeled 'inactive' is visible on the right. The main content area displays three collapsed sections: 'Cluster Memory Usage (0 active)', 'Cluster CPU Usage (0 active)', and 'Pod Memory usage (0 active)'. Each section has a summary message indicating it is inactive.

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.

The screenshot shows the same Prometheus Alerts interface with the expanded alert definitions. The 'Cluster Memory Usage' alert is defined with a query sum(container_memory_working_set_bytes{id="/"}) / sum(machine_memory_bytes) * 100 > 90, a duration of 5m, and a summary 'Cluster Memory Usage > 90%'. The 'Cluster CPU Usage' alert is defined with a query sum(rate(container_cpu_usage_seconds_total{id="/"}[5m])) / sum(machine_cpu_cores) * 100 > 90, a duration of 5m, and a summary 'Cluster CPU Usage > 90%'. The 'Pod Memory usage' alert is defined with a query sum by(container_label_io_kubernetes_pod_name) (container_memory_working_set_bytes[image!="",name=~"^k8s_.*"]) > 4.026531841e+09, a duration of 5m, and a summary 'Pod Memory usage > 90%'. All alerts have a severity of 'page' and no annotations.

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. (This is in the "extra" directory.)

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

The screenshot shows the meld merge tool comparing two YAML files: ps-cm-with-rules.yaml and ps-cm-with-rules2.yaml. The left pane contains the original configuration, while the right pane shows the updated configuration with new MySQL-related alert rules. The right pane is highlighted in green, indicating the changes made in ps-cm-with-rules2.yaml. The new rules include alerts for MySQL down status and MySQL high select usage.

```

# ps-cm-with-rules.yaml
component: server
heritage: Helm
release: pc
name: pc-prometheus-server
namespace: monitoring
...
alerting_rules.yaml: |-
  groups:
    - name: demo alerts
      rules:
        - alert: 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%
        - alert: 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: {{ $value }}).
              summary: Cluster CPU Usage > 90%
        - alert: Pod Memory usage
          expr: sum by(container_label_io_kubernetes_pod_name) (container_memory_working_set_bytes[image!="",name=~"^k8s_.*"]) > 4.026531841e+09
          for: 5m
          labels:
            severity: page
            annotations:
              description: Pod Memory usage on host {{$labels.instance}} : (current value: {{ $value }}).
              summary: Pod Memory usage > 90%
# ps-cm-with-rules2.yaml
component: server
heritage: Helm
release: pc
name: pc-prometheus-server
namespace: monitoring
...
alerting_rules.yaml: |-
  groups:
    - name: demo alerts
      rules:
        - alert: MySQL Down
          expr: mysql_up == 0
          for: 0m
          labels:
            severity: critical
            annotations:
              summary: MySQL down (instance {{$labels.instance}})
              description: "MySQL instance is down on {{$labels.instance}}\n VALUE = 0"
        - alert: MySQLHighSelectUsage
          expr: rate(mysql_global_status_commands_total{command=~"(select)"})[5m] * 100 > 90
          for: 5m
          labels:
            severity: warning
            annotations:
              summary: MySQL Select Usage High (instance {{$labels.instance}})
              description: "MySQL High Level of SELECT usage on {{$labels.instance}}\n VALUE = 100"
        - alert: 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%

```

- 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
```

The screenshot shows the Prometheus UI with the 'Alerts' tab selected. There are five alerts listed under the demo alerts section:

- > Cluster Memory Usage (0 active)
- > Cluster CPU Usage (0 active)
- > Pod Memory usage (0 active)
- /etc/config/alerting_rules.yml > mysql alerts
- > MySQLDown (0 active)
- > MySQLHighSelectUsage (0 active)

The 'MySQLDown' alert is expanded, showing its configuration details. The 'MySQLHighSelectUsage' alert is also expanded, showing its configuration details.

- 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.

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

- 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 with the 'Alerts' tab selected. The 'MySQLHighSelectUsage' alert is expanded, showing its configuration details. The alert state is listed as 'FIRING'.

Labels	State	Active Since	Value
alarmname=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

The 'Annotations' section contains the following information:

```
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 interface at `localhost:31500/#/alerts`. At the top, there are navigation icons, a search bar, and a status bar showing 90%. Below the header is a menu with **Alertmanager**, **Alerts**, **Silences**, **Status**, and **Help**. A **New Silence** button is in the top right. The main area has tabs for **Filter** and **Group**. Under **Filter**, there is a search input with placeholder "Custom matcher, e.g. `env=production`" and a **+ Silence** button. Below the filter is a link to "Expand all groups". A single alert is listed under "Not grouped": **1 alert**. The alert details are as follows:

- Time:** 2022-01-09T19:19:47.065Z
- Info:** + Info
- Source:** + Source
- Silence:** + Silence
- Matchers:**
 - alertname="MysqlHighSelectUsage"
 - command="select"
 - instance="mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104"
 - job="mysql"
 - severity="warning"

12. You can also Silence alerts for some period of time. 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 "New Silence" form at `localhost:31500/#/silences/new?filter=(alertname%3D'MysqlHighSelectUsage'%2C command%3D'select'%2C instance%3D'mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local%3A9104'%2C job%3D'mysql'%2C severity%3D'warning')`. The page title is "Silence". The form fields are:

- Start:** 2022-01-09T19:32:07.516Z
- Duration:** 10m
- End:** 2022-01-09T19:42:07.516Z
- Matchers:**
 - alertname="MysqlHighSelectUsage"
 - command="select"
 - instance="mysql-exporter-prometheus-mysql-exporter.monitoring.svc.cluster.local:9104"
 - job="mysql"
 - severity="warning"
- Custom matcher, e.g. `env=production`**: An input field with a plus sign (+).
- Creator:** User 1
- Comment:** Demo silence
- Buttons:** Preview Alerts, Create (highlighted in blue), and Reset.

13. 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". At the top, there are tabs for "Alerts", "Silences", "Status", and "Help", with "Silences" being the active tab. A "New Silence" button is located in the top right corner. The main content area displays the following details for the silence:

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	<pre>alertname=MysqlHighSelectUsage command=select instance=mysql-exporter-prometheus-mysql-exporter-monitoring.svc.cluster.local:9104 job=mysql severity=warning</pre>

Below the table, a message states "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. For this lab, we need to get the default password that was created when Grafana was installed. Run the command "get-gf-pass.sh" to retrieve it and then copy the password that is displayed.

```
$ ~/prom-start/extra/get-gf-pass.sh
```

2. We already have an instance of Grafana running on this system. Open up a browser to the home page at <http://localhost:31750>. The default admin userid is "admin". The password will be the one that was echoed to your screen from step 1. Just enter/paste the output from that step into the password field.

Welcome to Grafana

Need help? [Documentation](#) [Tutorials](#) [Community](#) [Public Slack](#)

Basic

The steps below will guide you to quickly finish setting up your Grafana installation.

TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

DATA SOURCES
Add your first data source

DASHBOARDS
Create your first dashboard

Learn how in the docs [\[link\]](#)

Learn how in the docs [\[link\]](#)

Dashboards

Starred dashboards

Latest from the blog

Logs Scenes App

Subtitle for this wonderful example app.

Data source: Loki | Source series: place | Stream value: moon

Last 1 hour [\[link\]](#)

- Let's first add our Prometheus instance as a Data Source. Click on the "3 bar" menu at the top left and then select "Connections" and then "Data Sources" from the left menu. Then click on the blue button for "Add data source".

Home

Starred

Dashboards

Explore

Alerting

Connections

Add connection

Data sources

Administration

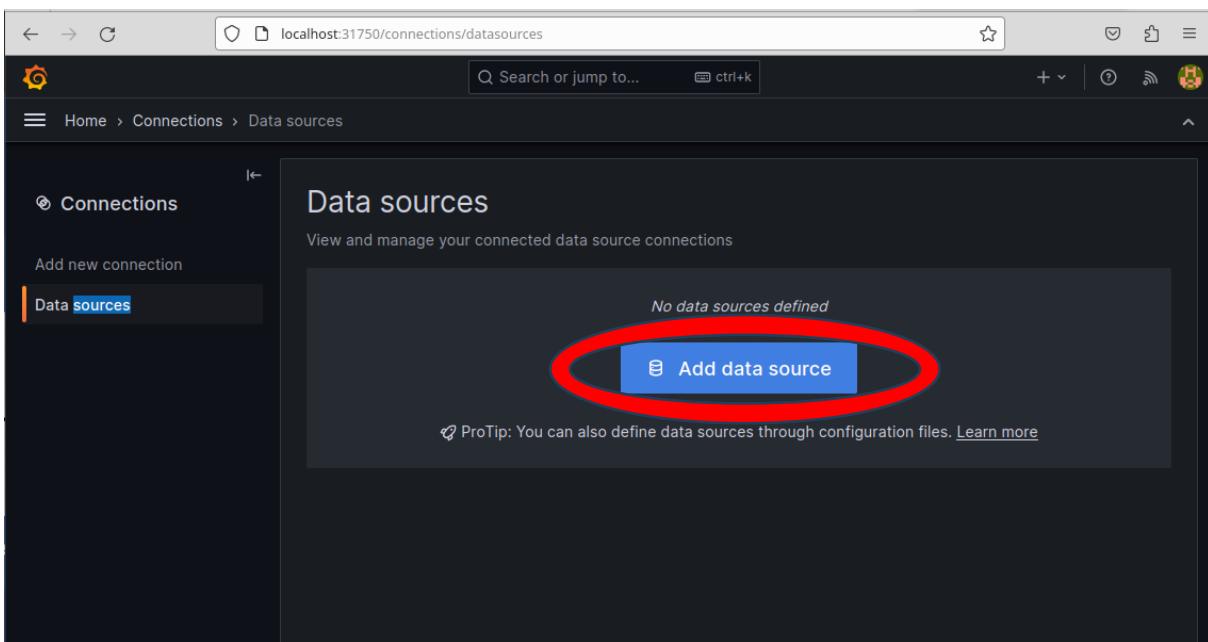
Need help? [Documentation](#) [Tutorials](#) [Community](#) [Public Slack](#)

AND DASHBOARDS
fundamentals

understand Grafana if you have no e. This tutorial guides you through ness and covers the "Data source" ds" steps to the right.

DATA SOURCES
Add your first data source

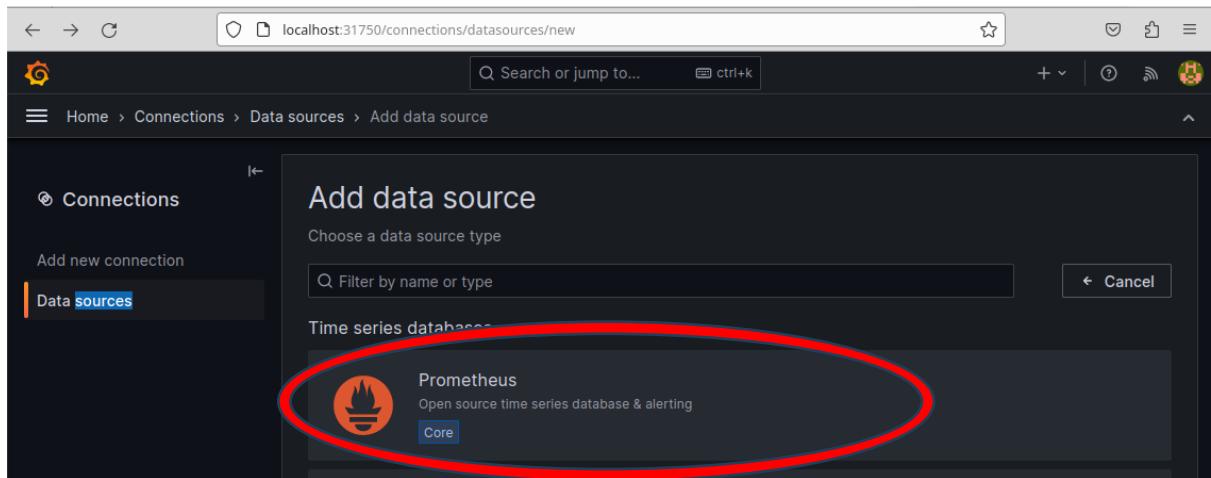
DASHBOARDS
Create your first dashboard



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

<http://prom-start-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.

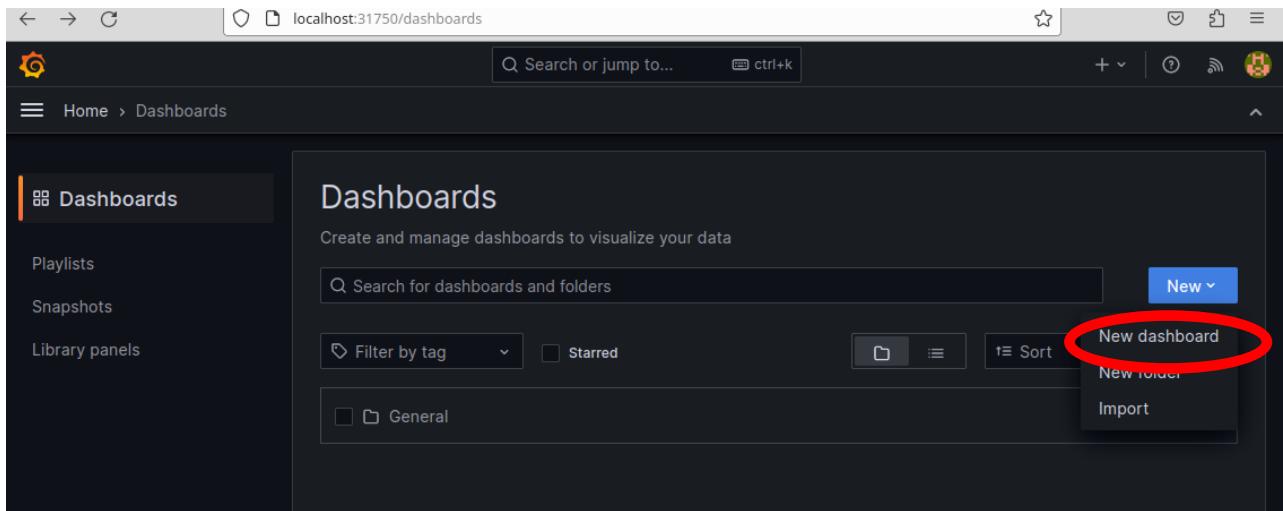


The screenshot shows the 'Settings' tab for the 'Prometheus-1' data source in Grafana. The 'HTTP' section is visible, featuring fields for 'Prometheus server URL' (set to 'prometheus-server.monitoring.svc.cluster.local:80'), 'Allowed cookies' (with a placeholder 'New tag (enter key to add)'), and 'Timeout'. A red oval highlights the 'Prometheus server URL' field.

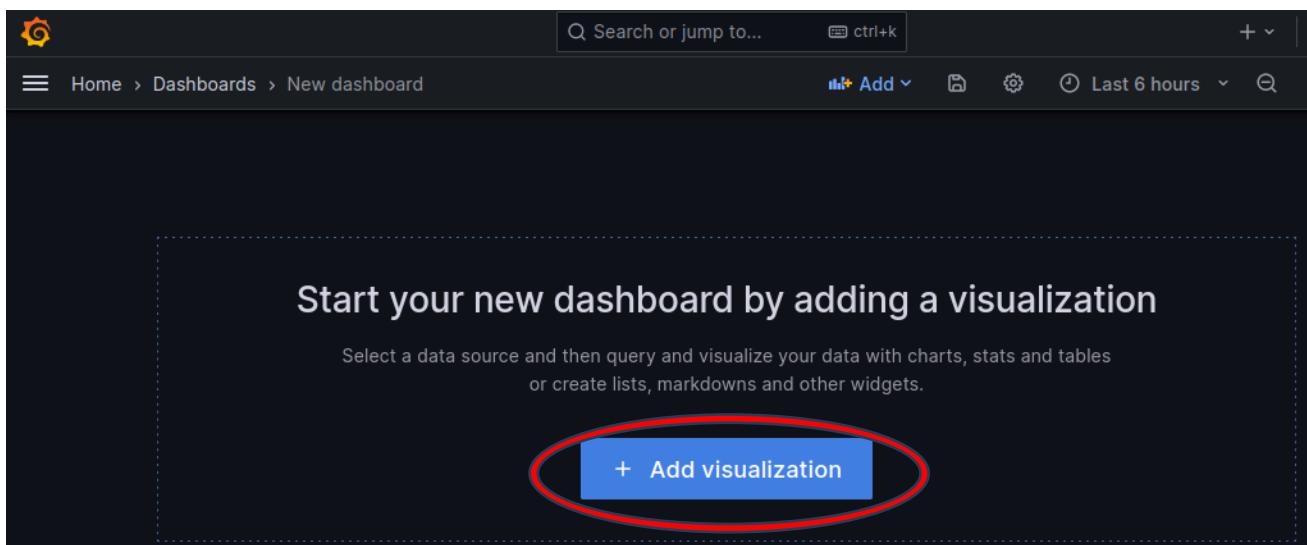
The screenshot shows the 'Exemplars' configuration page. It includes a '+ Add' button, 'Back' and 'Explore' buttons, and a 'Delete' button. A red oval highlights the 'Save & test' button.

The screenshot shows the results of a successful API query. It displays 'Custom query parameters' (example: max_source_resolution=5m&timeout) and 'HTTP method' (POST). Below this, a green success message states 'Successfully queried the Prometheus API.' and suggests starting a dashboard or exploring data. A red oval highlights the 'Save & test' button at the bottom.

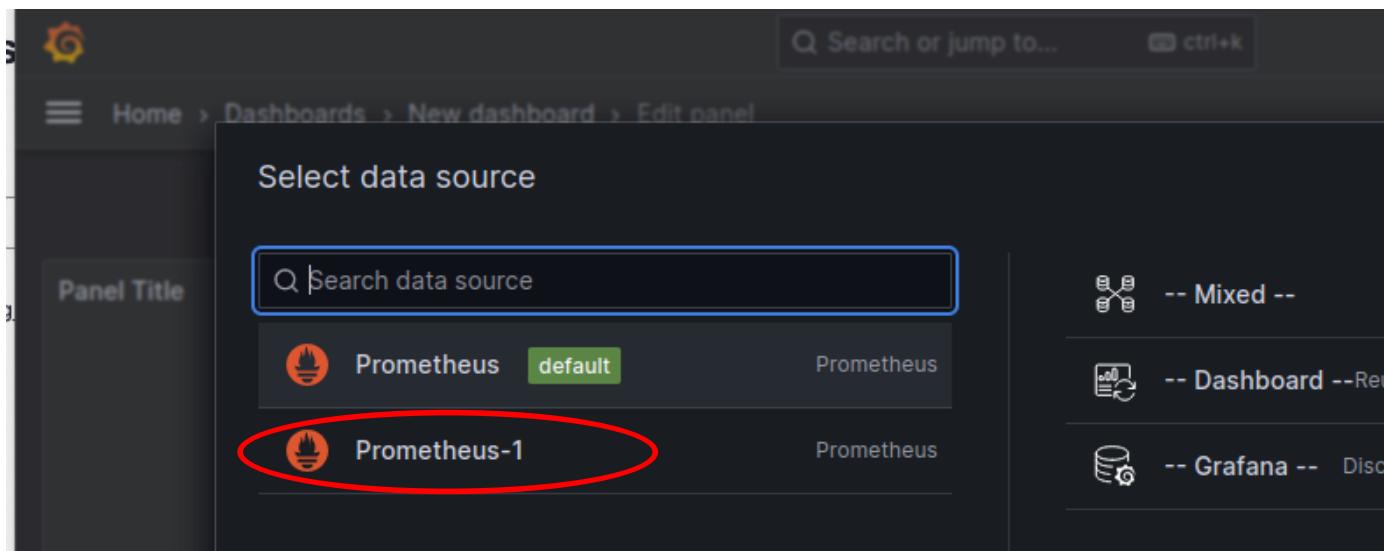
- Now, let's create a simple dashboard for one of our mysql metrics. Click on the "three bars" menu on the left and select "Dashboards". Then on the Dashboards pane, select "New" and "New dashboard" from the menu.



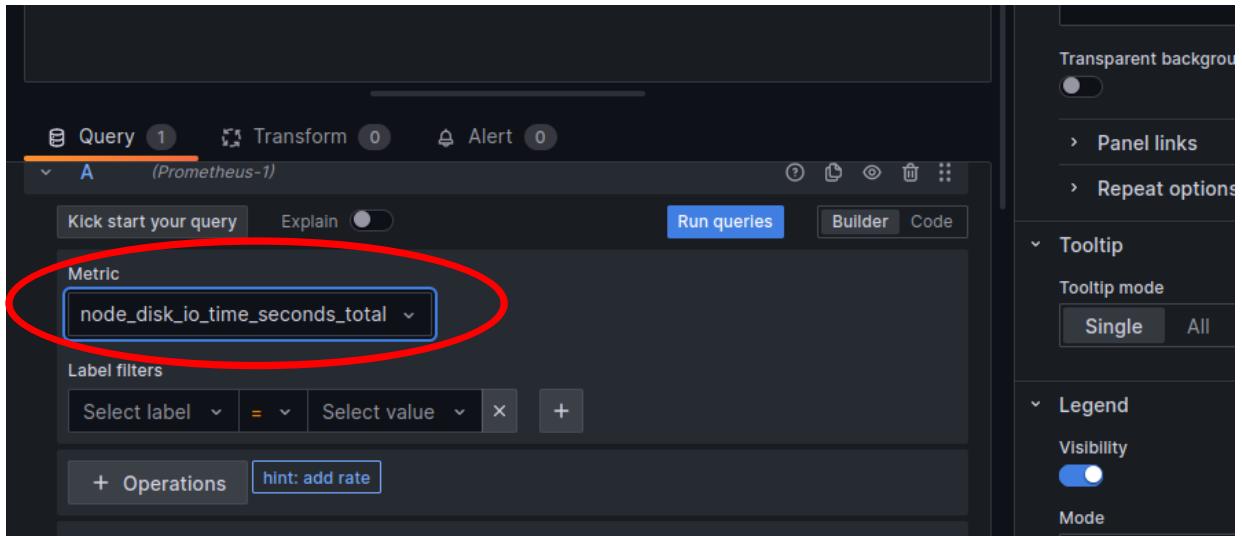
6. Click on “Add visualization”.



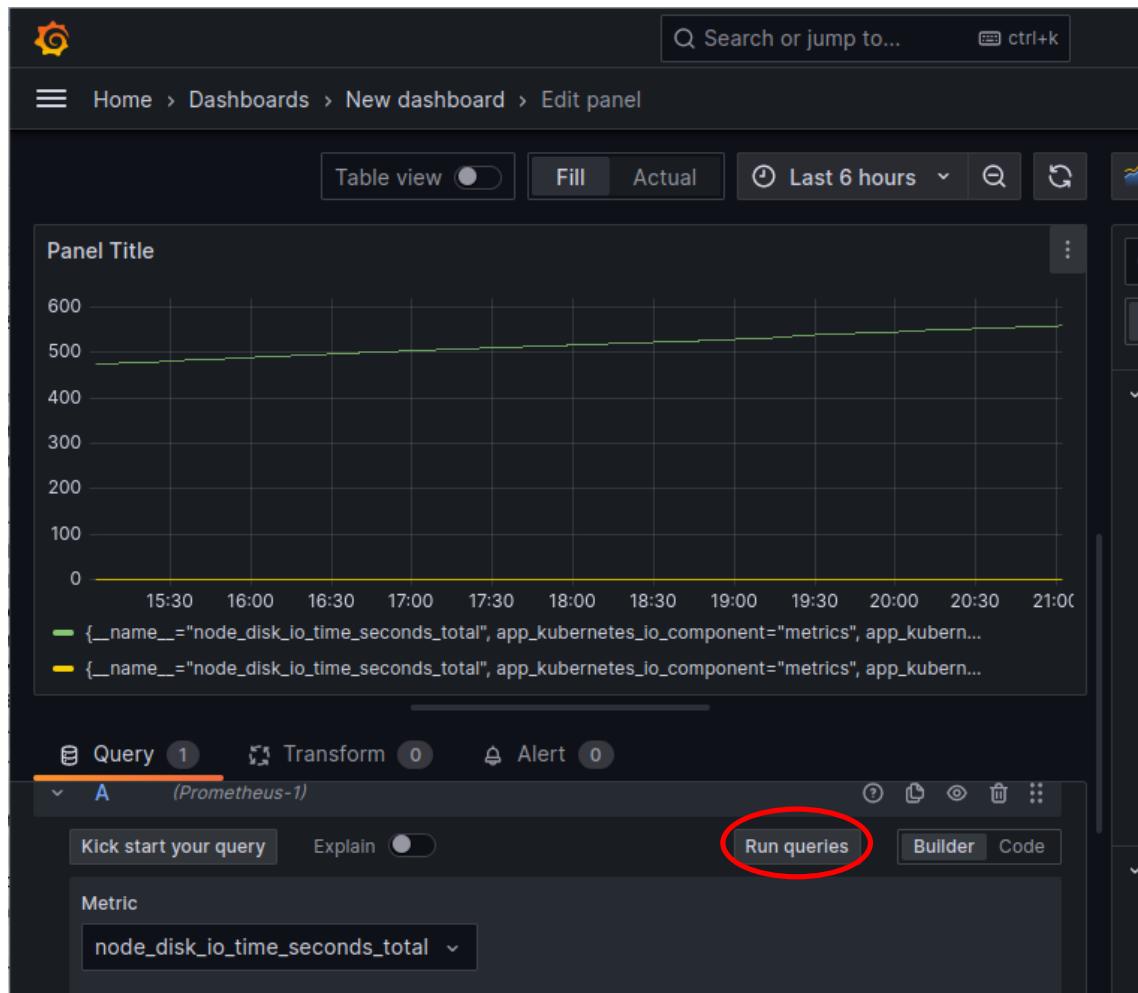
7. Select the “Prometheus-1” data source on the next screen.



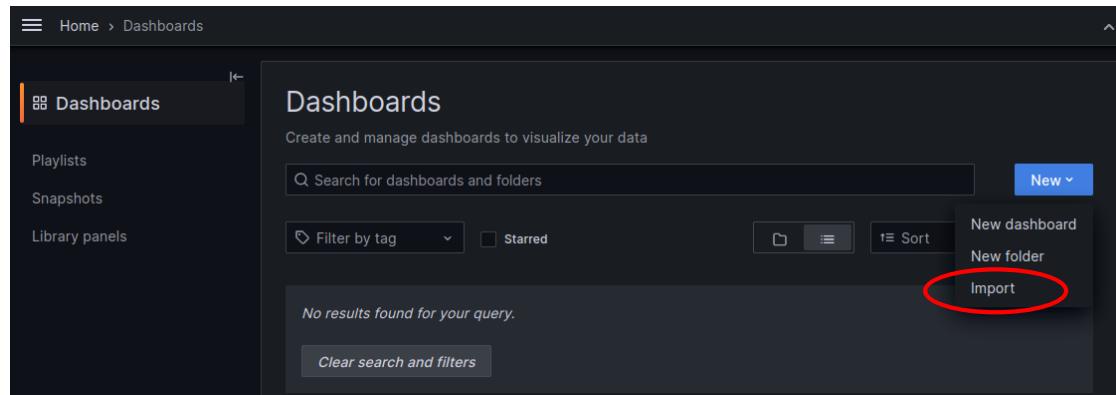
8. Then click on the "Select metric" section and pick a metric to show. For example, you could type in "node_disk_io_time_seconds_total".



9. When done, click on the "Run queries" button and you should see a new graph being shown.

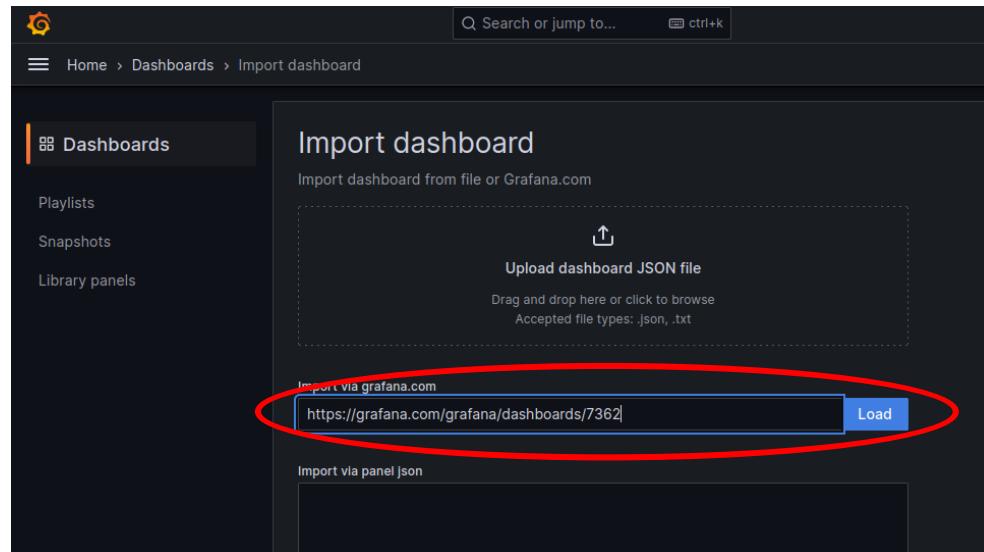


10. While we can create individual dashboards with Grafana, that can take a lot of time and effort. The community has already created a number of dashboards that we can just import and use. So let's grab one for mysql. Back on the main page, click on the "three bars" icon on the left side, then select "Dashboards" then work through to get to "Import".

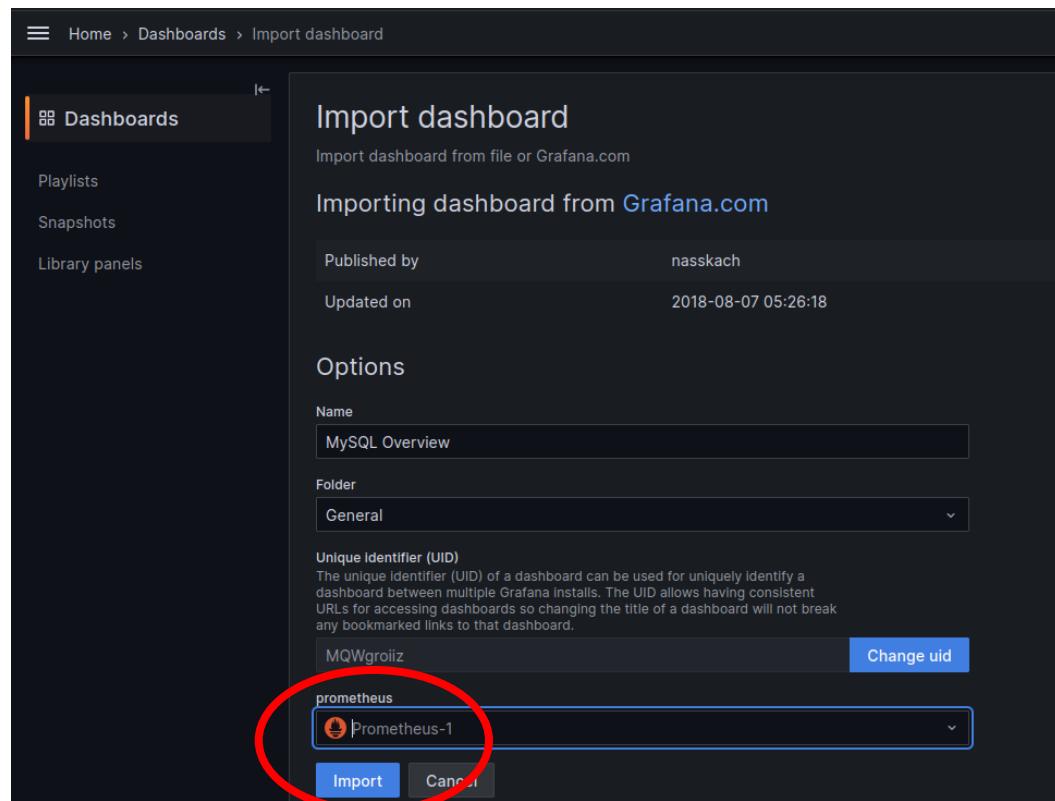


11. In the field that says "Grafana.com dashboard URL or ID", enter the location below and click the blue "Load" button.

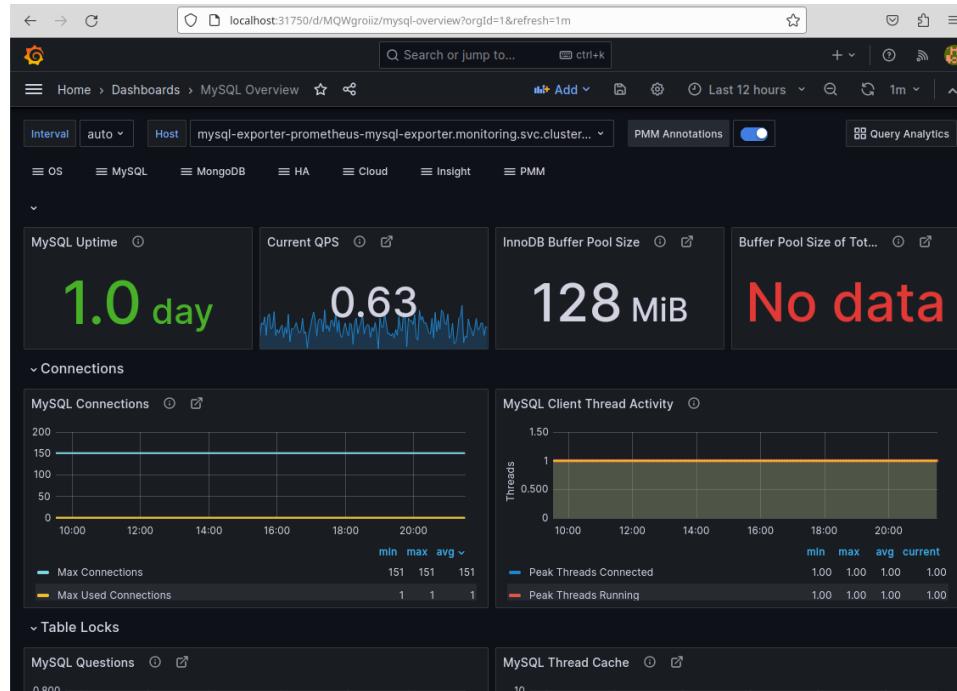
<https://grafana.com/grafana/dashboards/7362>



12. 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-1 data source that we setup. Then click the blue "Import" button at the bottom.

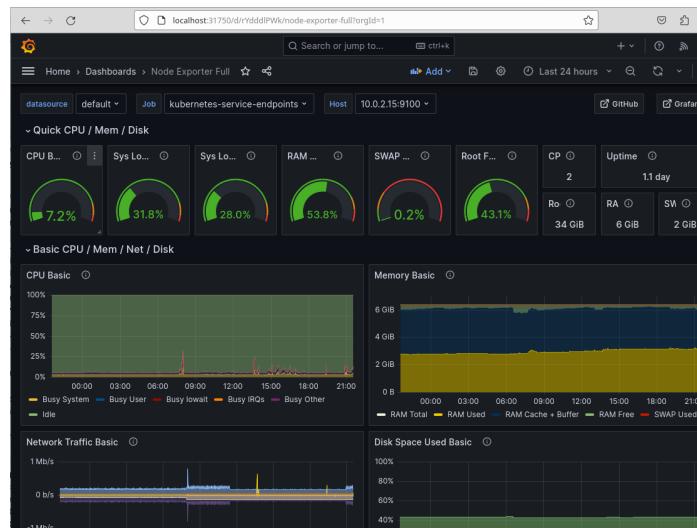


13. At this point, you should see a populated dashboard with a number of panels looking at the mysql exporter data from our system through Prometheus. You can scroll around and explore.



14. 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. (Here again, you'll need to select the Prometheus-1 data source as we did before.)

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



END OF LAB