Getting Started with Prometheus

Monitoring Kubernetes infrastructure and applications for reliability

Class Labs

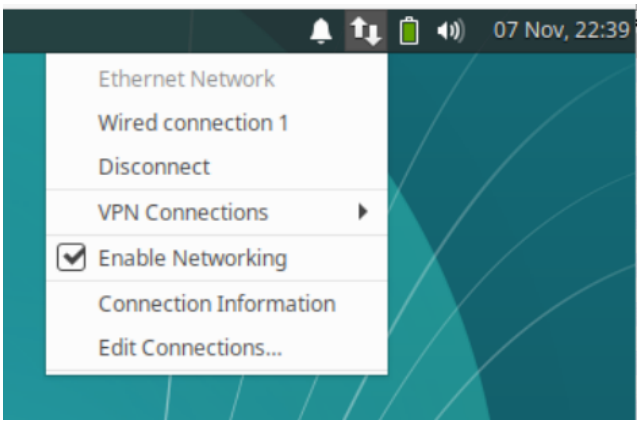
Version 1.1 by Brent Laster on behalf of Tech Skills Transformations

01/09/2022

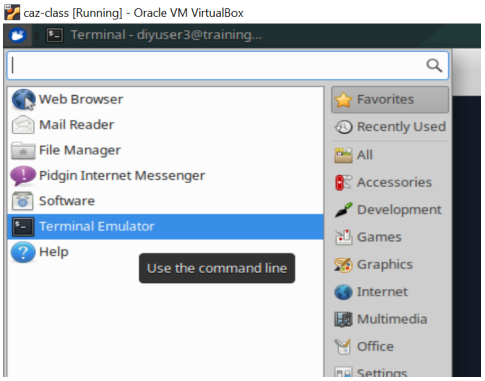
Important Prereq: These labs assume you have already followed the instructions in the separate setup document and have VirtualBox up and running on your system and have downloaded the *prom-start.ova* file and loaded it into VirtualBox. If you have not done that, please refer to the setup document for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

Lab 1 - Monitoring with Prometheus

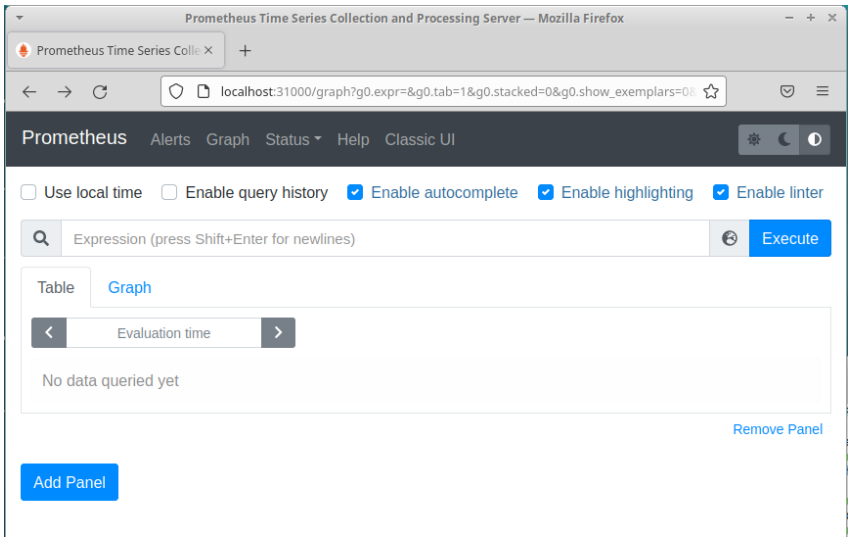
Purpose: In this lab, we'll run an instance of the node exporter and use Prometheus to surface basic data and metrics.

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
```

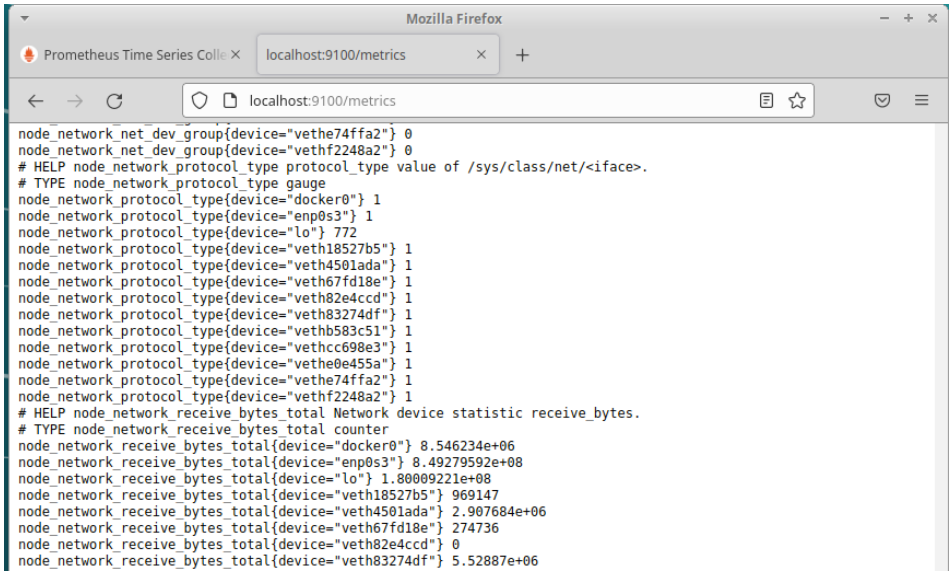
2. Notice that we have both Prometheus itself and the node exporter piece running there (among others). Let's take a look at the Prometheus dashboard in the web browser. Open up a web browser and go to the url below to see it.

http://localhost:31000/

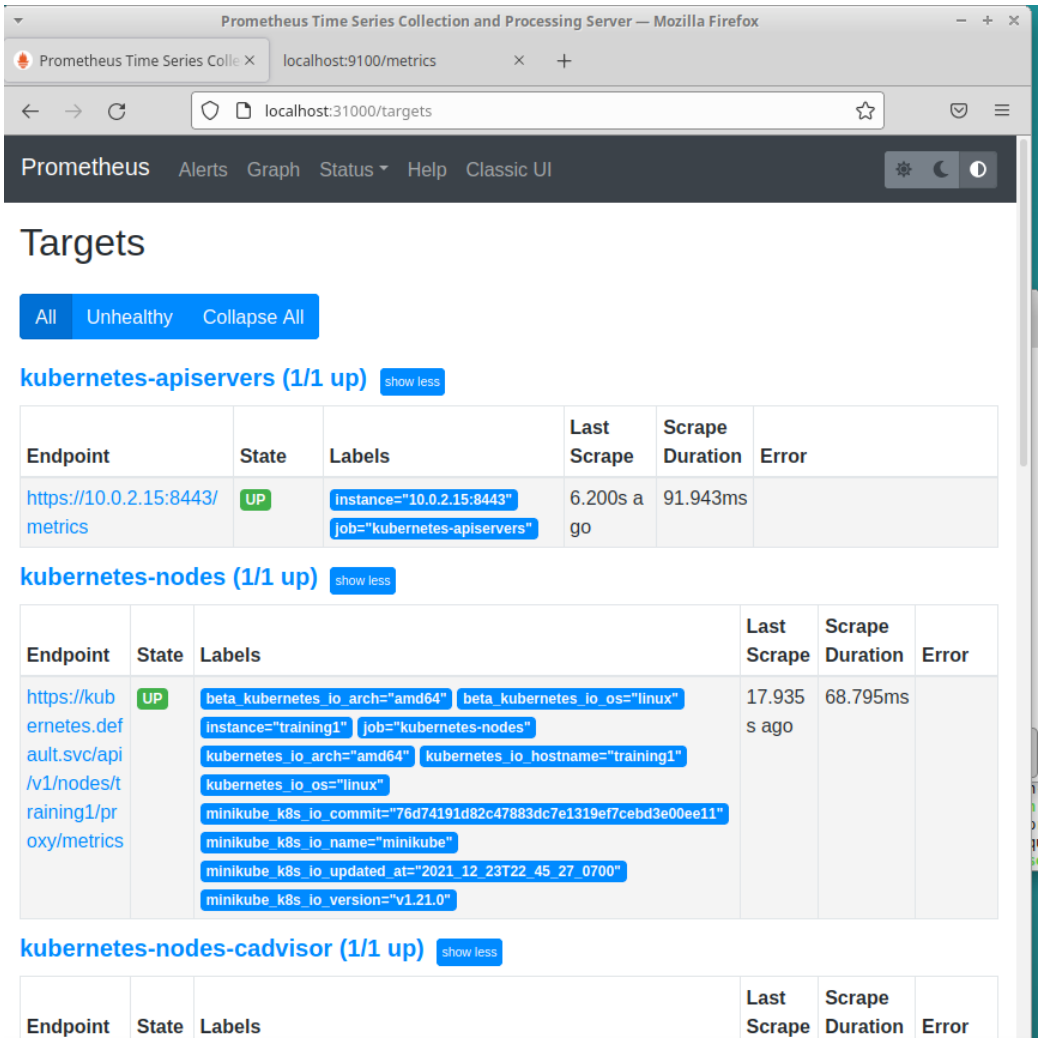


3. We'll get to use the dashboard more in the labs. For now, lets open up the node exporter's metrics page and look at the different information on it. (Note that we only have one node on this cluster.) Once on that page, scan through some of the metrics that are exposed by this exporter. Then see if 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



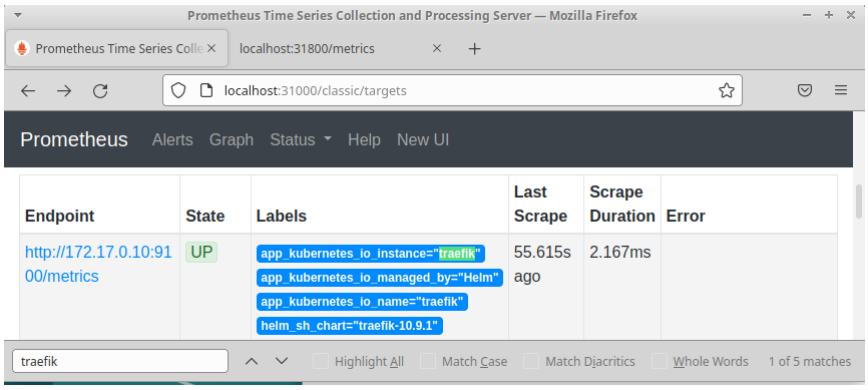
4. Now, let's see which targets Prometheus is automatically scraping from the cluster. Back in the top menu (dark bar) on the main Prometheus page tab, select Status and then Targets (or go to http://localhost:31000/targets). Then see if you can find how long ago the last scraping happened, and how long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



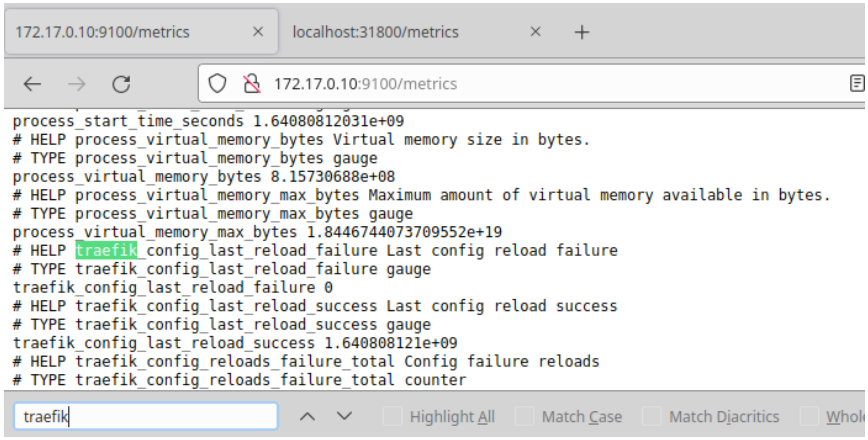
5. Let's setup an application in our cluster that has a built-in Prometheus metrics exporter - traefik - an ingress. The Helm chart is already loaded for you. So, we just need to create a namespace for it and run a script to deploy it. After a few moments, you should be able to see things running in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

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

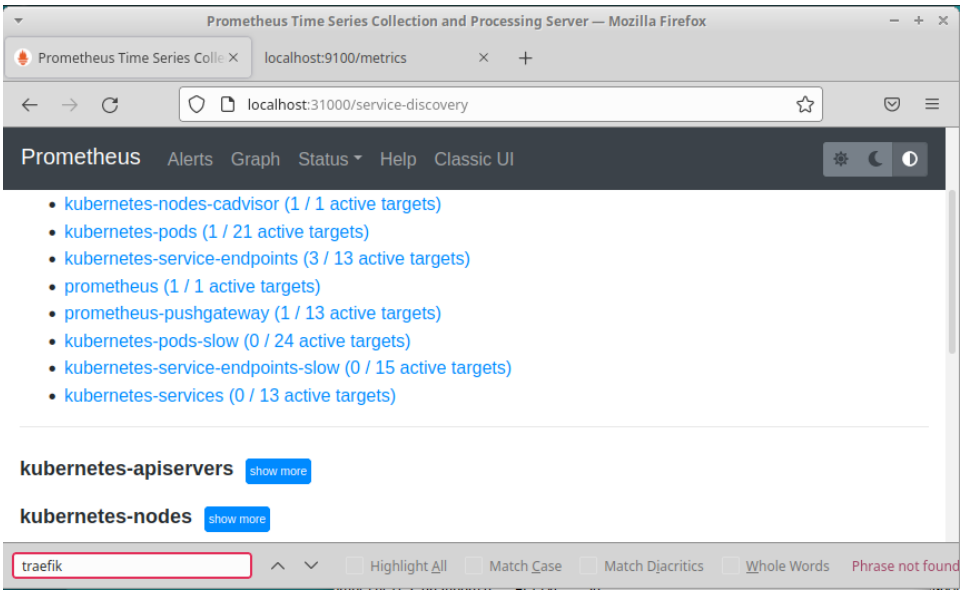
6. You should now be able to see the metrics area that Traefik exposes for Prometheus as a pod endpoint. Take a look in the **Status -> Targets** area of Prometheus and see if you can find it (**localhost:31000/targets**) and use a **Ctrl-F** to try to find the text "traefik". Note that this is the pod 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.)



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



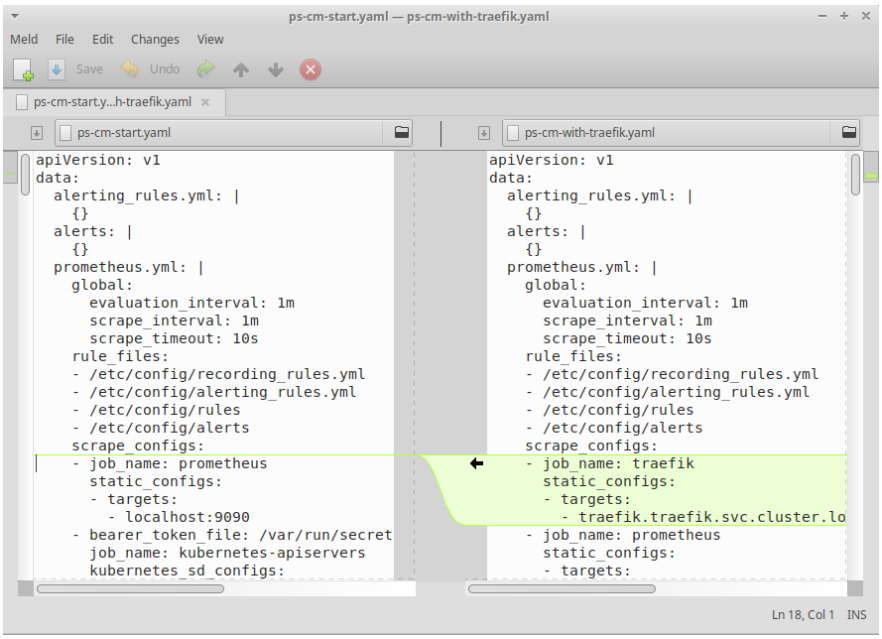
7. While we can find it as a pod endpoint, we don't yet have the traefik metrics established as a standalone "job" being monitored in Prometheus. You can see this because there is no section specifically for "traefik (1/1 up)" in the Targets page. Also, Traefik is not listed if you check the Prometheus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



8. 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)
```

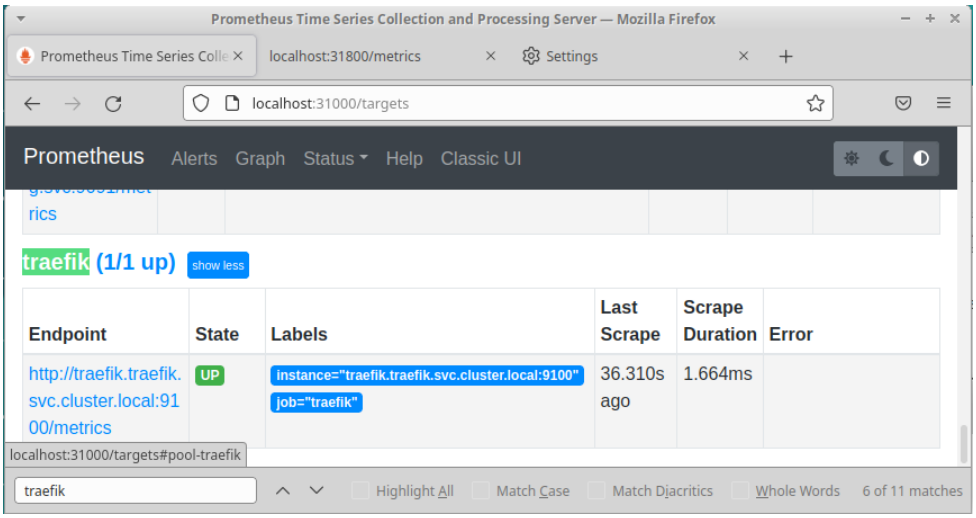
```
$ meld ps-cm-start.yaml ps-cm-with-traefik.yaml
```

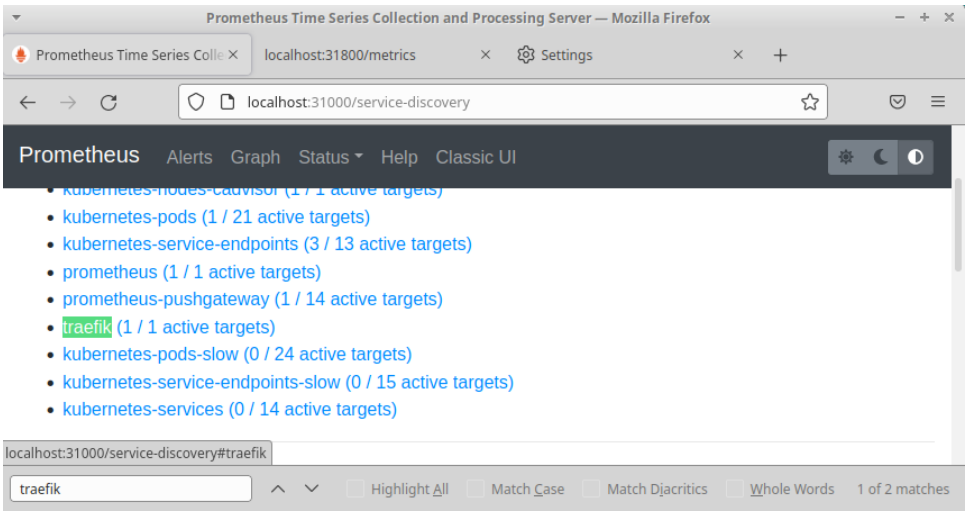


9. It's easy to see the difference here. When you're done viewing, just go ahead and close meld. Now we'll apply the new configmap definition with our additional job. (Ignore the warning.)

```
$ 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.)





END OF LAB

Lab 2- Deploying a separate exporter for an application

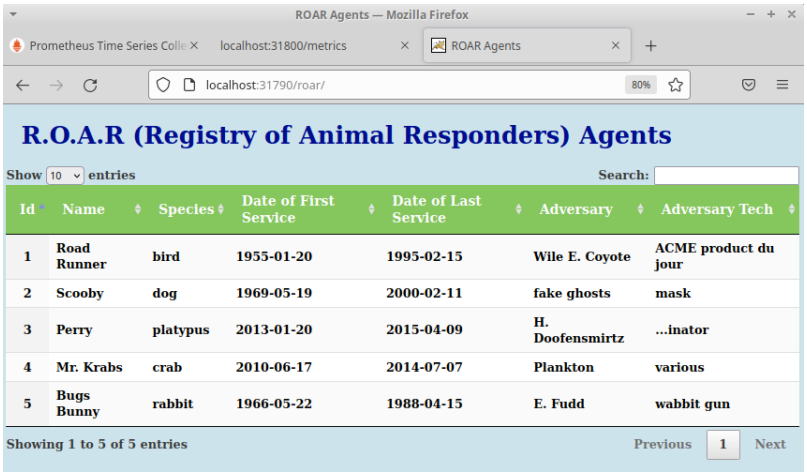
Purpose: In this lab, we'll see how to deploy a separate exporter for a mysql application running in our cluster.

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.

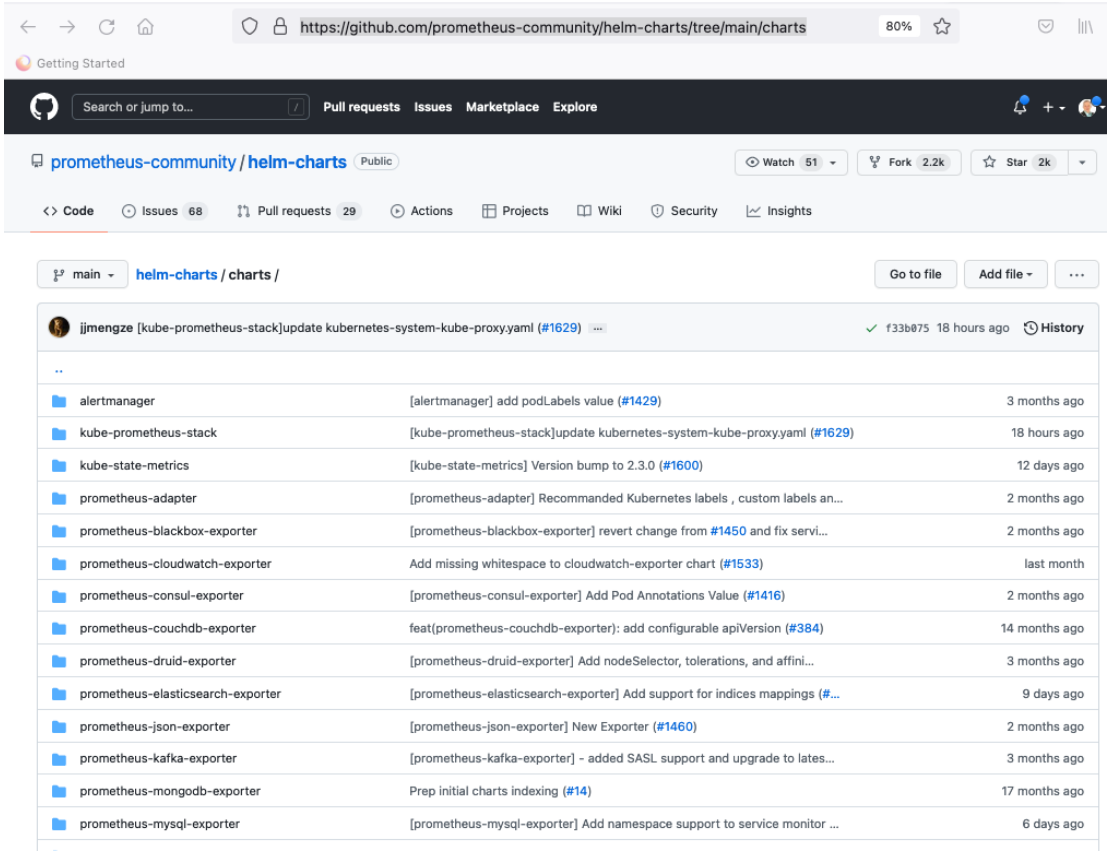
```
$ k create ns roar

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

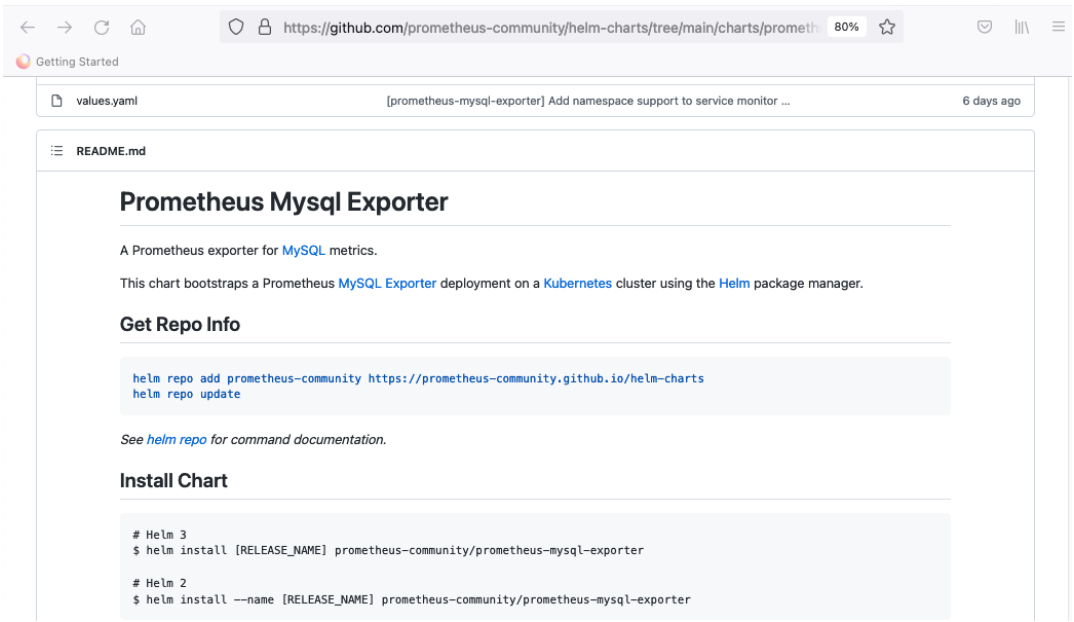
2. After a few moments, you can view the running application in the roar namespace and also in the browser at http://localhost:31790/roar .

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



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.





4. 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)
```

5. 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.

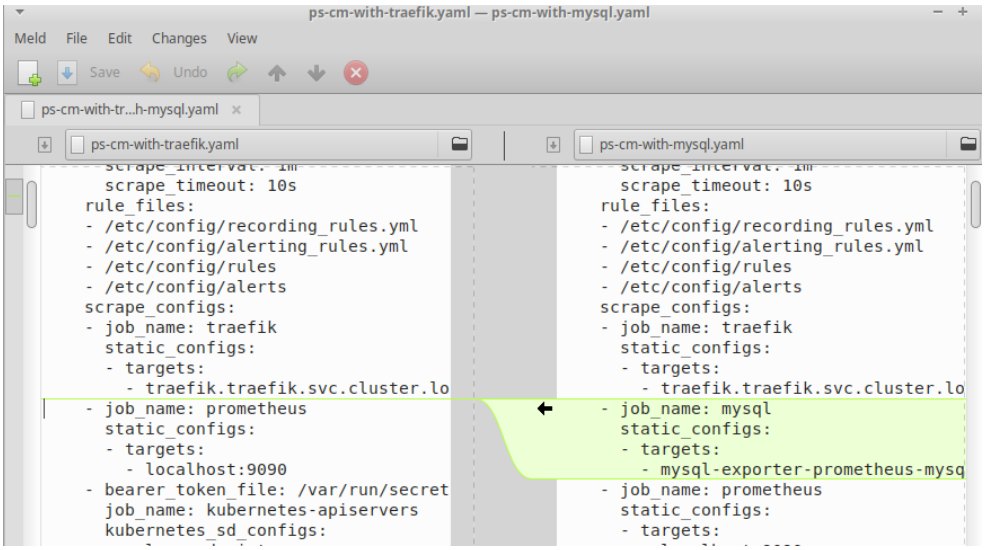
```
$ cat values.yaml

$ cat secret.yaml
```

6. 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
```

7. 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.

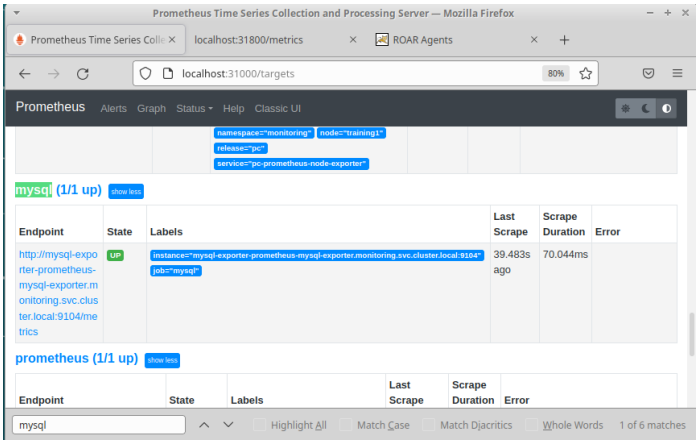
```
$ meld ../extra/ps-cm-with-traefik.yaml ps-cm-with-mysql.yaml
```

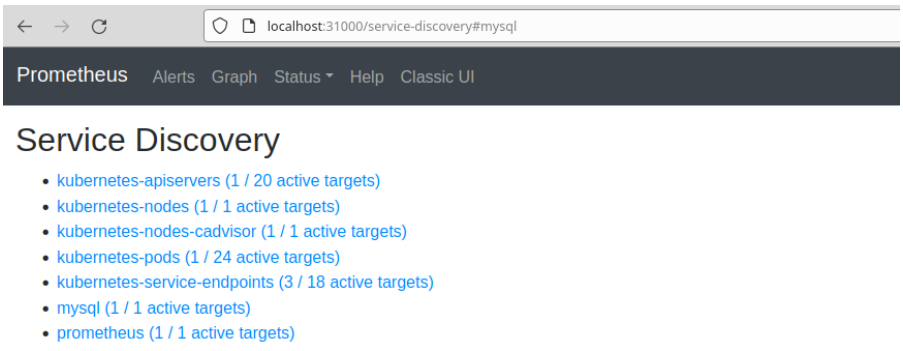


8. Now you can apply the updated configmap definition.

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

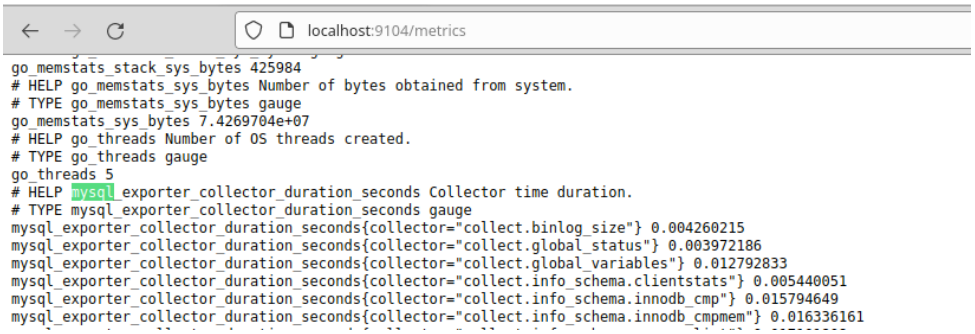
9. You should now be able to see the mysql item in the targets page (localhost:31000/targets) and also in the service-discovery page (localhost:31000/service-discovery#mysql). (Again, it may take a few minutes for the mysql target to appear and reach (1/1 up).)





10. (Optional) If you want to see the metrics that are exposed by this job, there is a small script named **pf.sh** 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
```

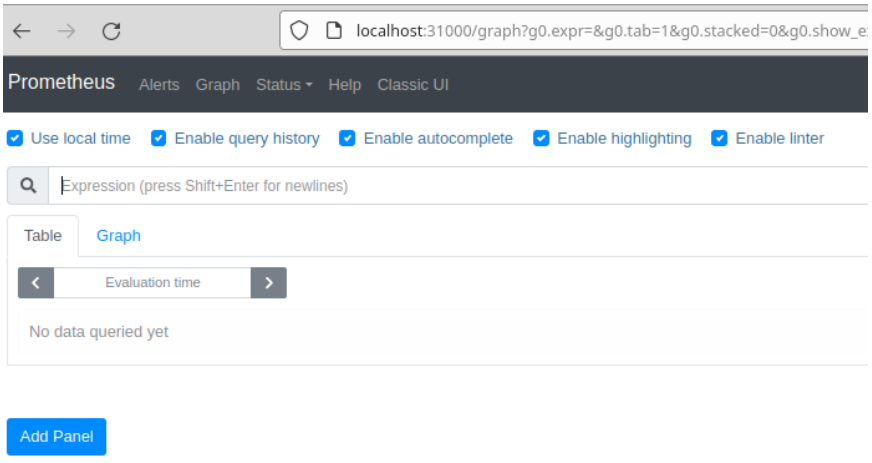


END OF LAB

Lab 3 - Writing queries with PromQL

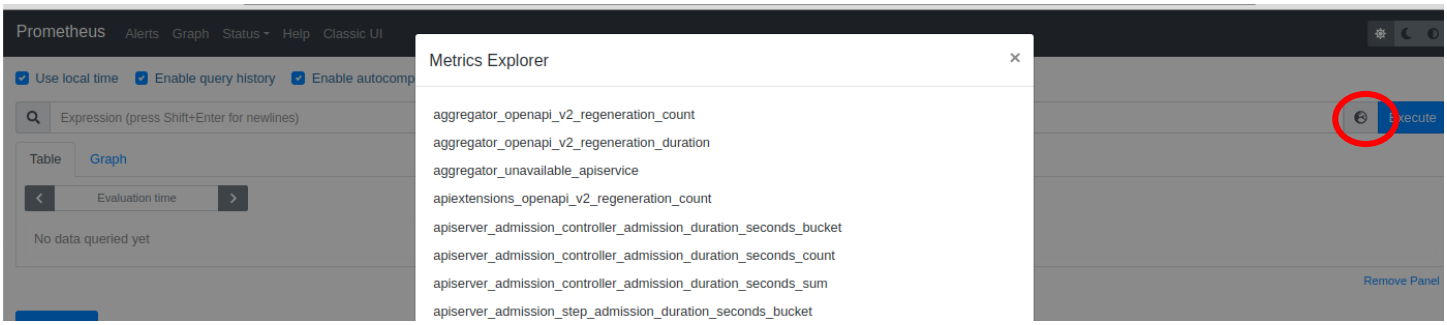
Purpose: In this lab, we'll see how to construct queries with the PromQL language.

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.

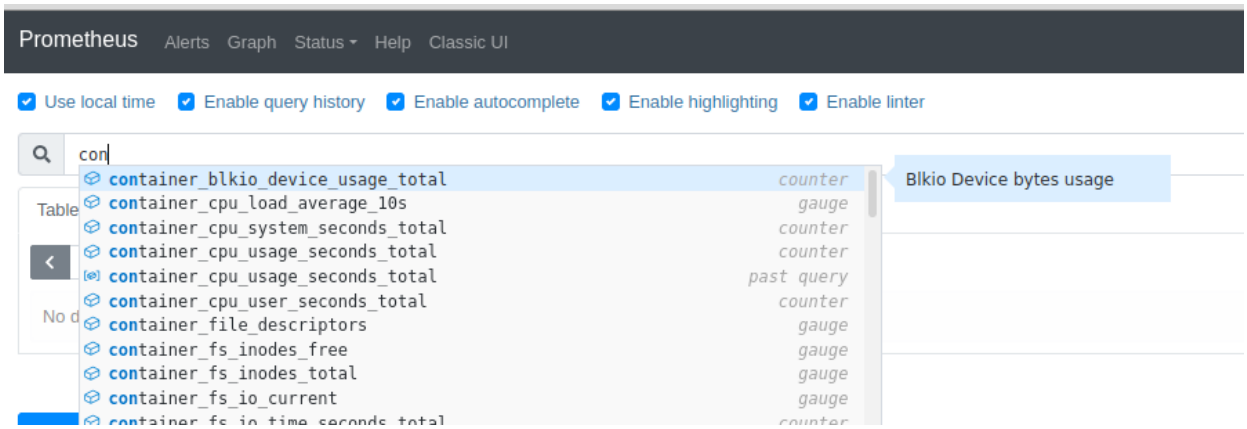


2. There are a couple of different ways to find available metrics to choose from in Prometheus. One way is to click on the query explorer icon next to the blue "Execute" button on the far right. Click on that and

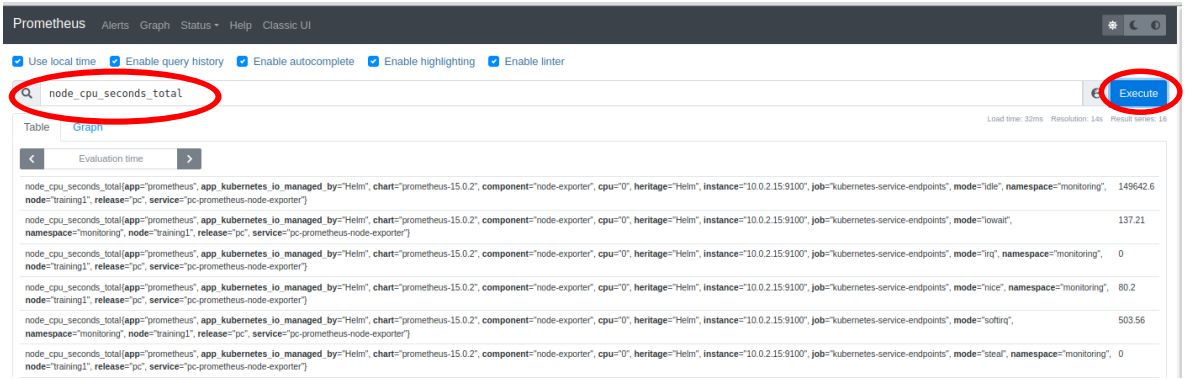
you can scroll through the list that pops up. You don't need to pick any right now and you can close it (via the "x" in the upper right) when done.



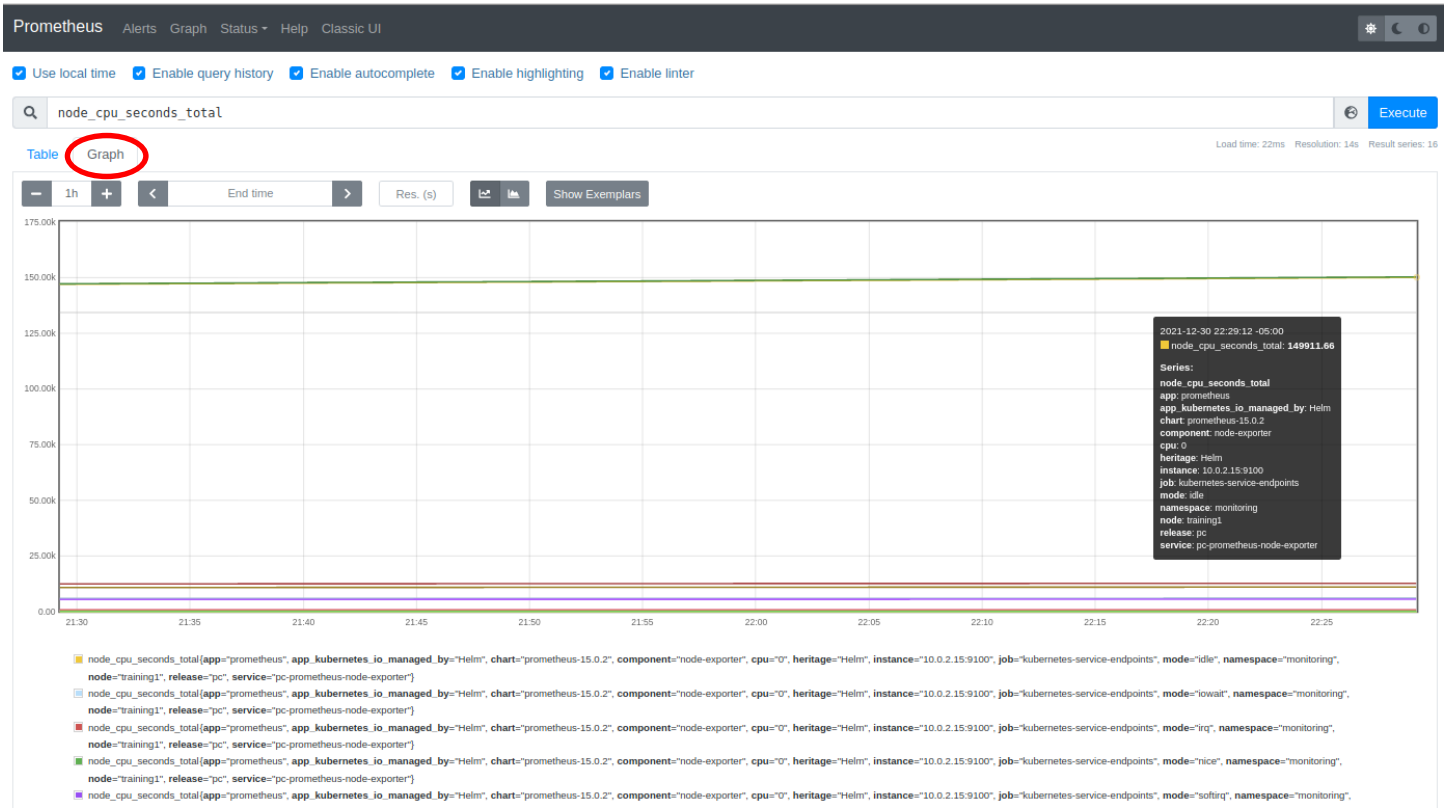
3. Another way to narrow in quickly on a metric you're interested in is to start typing in the "Expression" area and pick from the list that pops-up based on what you've typed. Try typing in the names of some of the applications that we are monitoring and see the metrics available. For example, you can type in "con" to see the ones for containers, "mysql" to see the ones for mysql, "trae" to see the ones for traefik, 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.



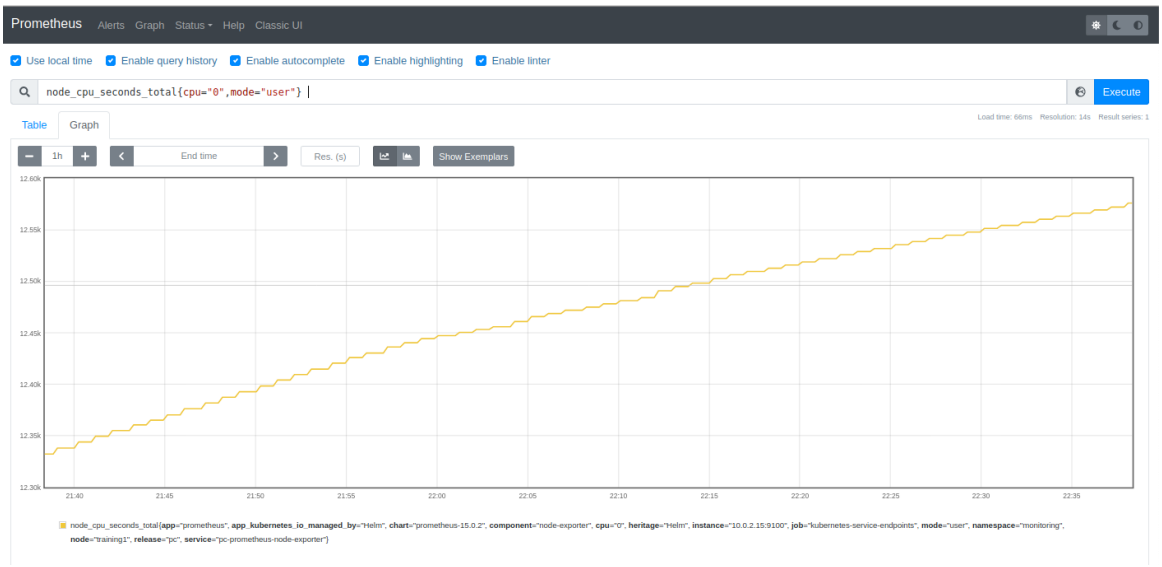
5. Notice that we have a lot of rows of output from this single query. If you look closely, you can see that each row is different in some aspect, such as the cpu number or the mode. Rows of data like this are not that easy to digest. Instead, it is easier to visualize with a graph. So, click on the "Graph" link above the rows of data to see a visual representation. You can then you're 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.



6. 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"}
```

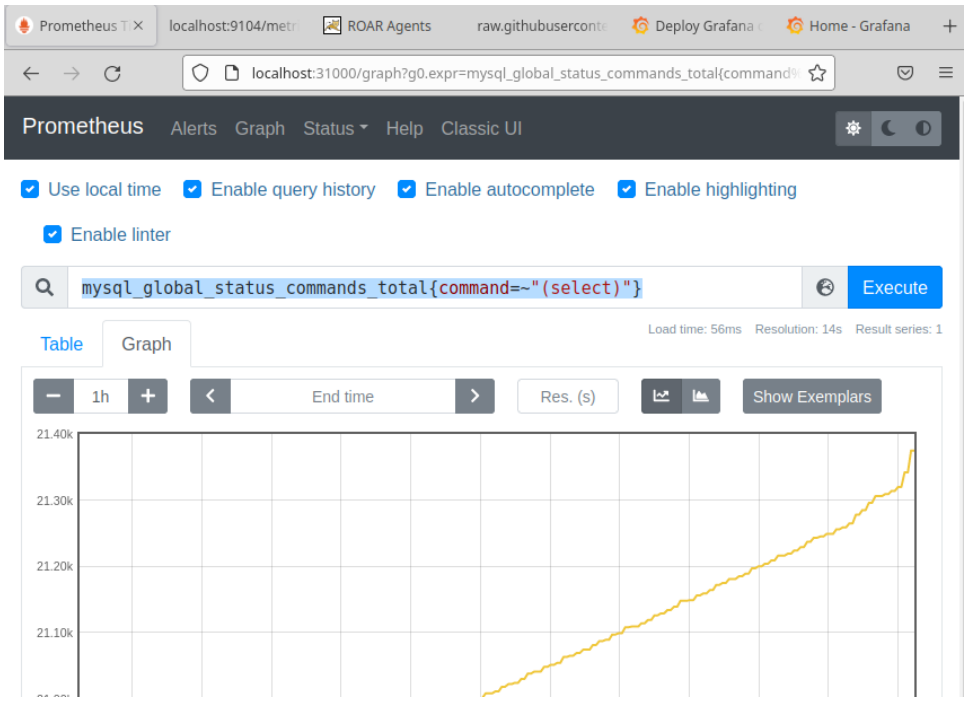


7. So far we have used counters in our queries - a value that increases (or can be reset to 0) as indicated by the "total" in the name. However, there are other kinds of time series such as "gauges" where values can go up or down. Let's see an example of one of those. Change your query to the expression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

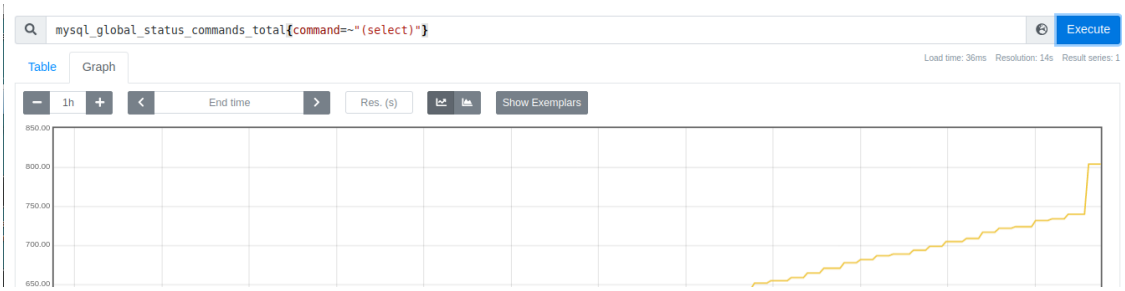
```
mysql_global_status_commands_total{command=~"(select)"}
```

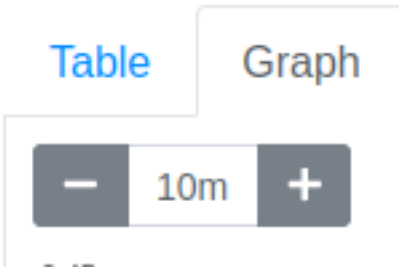


9. Now let's simulate some query traffic to the database. I have a simple shell script that randomly queries the database in our application x times while waiting a certain interval between queries. It's called ping-db.sh. Run it for 30 times with an interval of 1 second and then go back and refresh 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.





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.

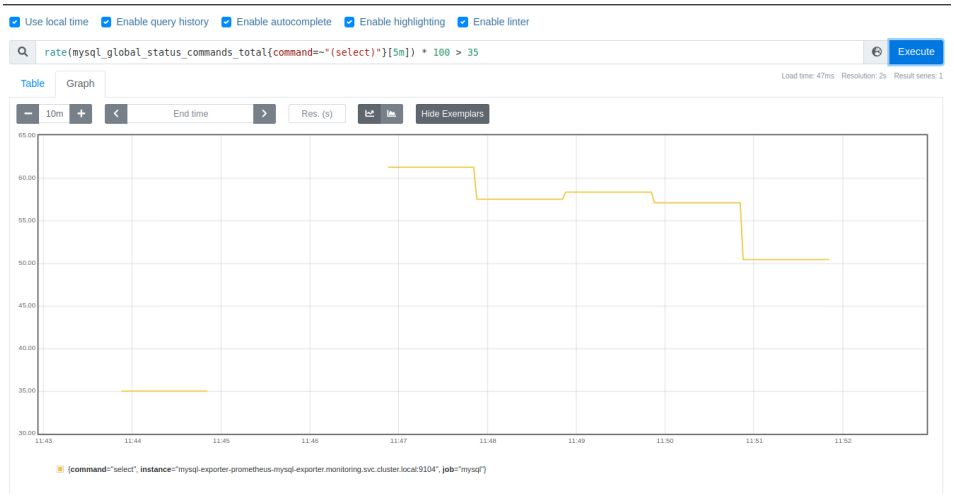
1. Let's suppose that we want to get alerted when the "select" traffic spikes to high levels. We have a working "rate" query for our mysql instance gives us that information from the last lab. Take another look at that one to refresh your memory. Now let's change it to only show when our rate is above .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 > 35
```

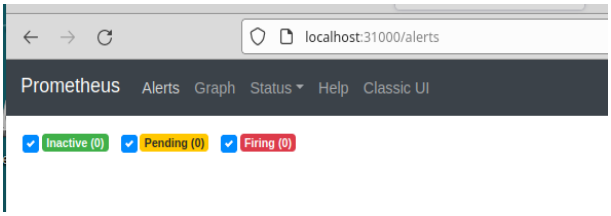
2. After clicking on the Execute button to refresh, you will probably see an empty query result on the page. This is because we are targeting a certain threshold of data and that threshold hasn't been hit in the time range of the query (5 minutes). To have some data to look at, let's run our program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

3. 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.



4. 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 wil not see any configured.

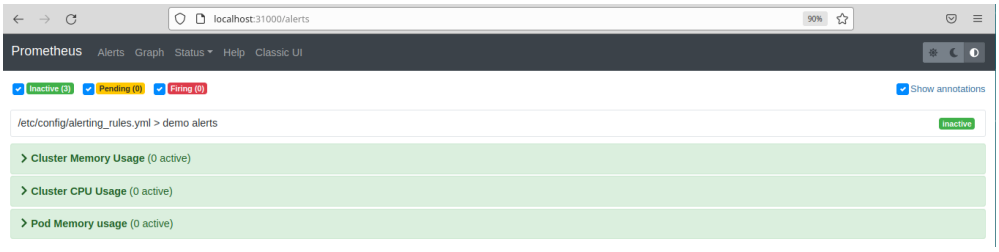


5. Now let's configure some alert rules. We already have a configmap with some basic rules in it. gedit or cat the file extra/ps-cm-with-rules.yaml and look at the "alerting-rules.yml" definition under "data:".

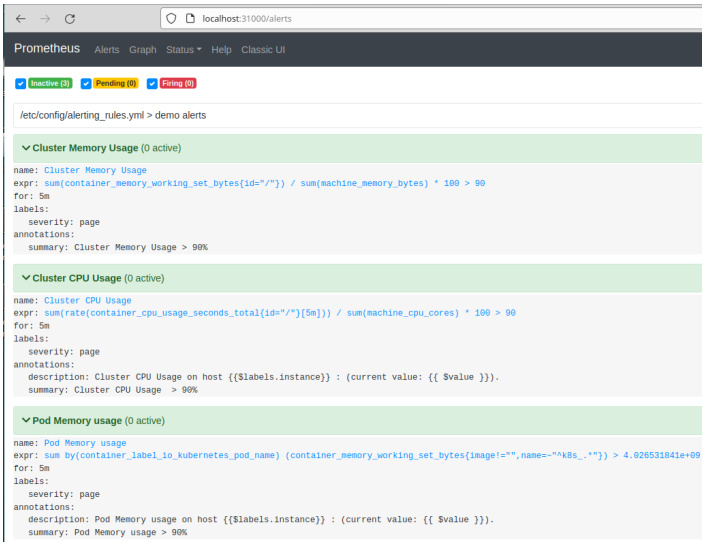
```
$ cat (or gedit ) extra/ps-cm-with-rules.yaml
```

6. 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
```

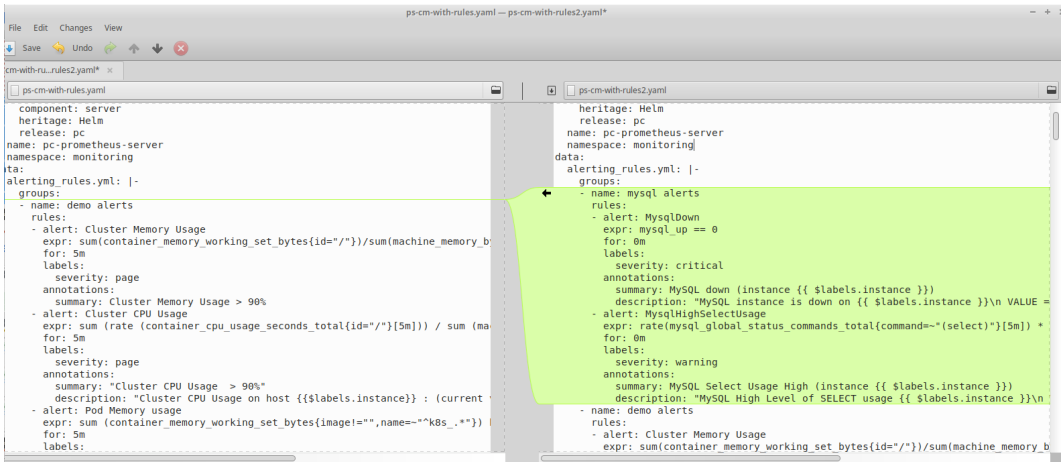


You can then expand those to see the alert definitions. Notice that each alert uses a PromQL query like we might enter in the main Prometheus query area.



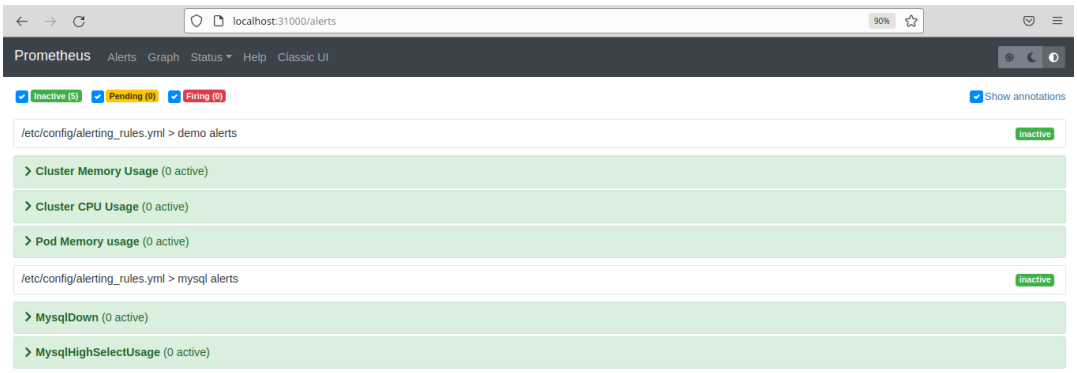
7. Let's add some custom alert rules for a group for mysql. These will follow a similar format as the other rules but using mysql PromQL queries, names, etc. There is already a file with them added - **ps-cm-with-rules2.yaml**. You can do a meld on that and our previous version of the cm data to see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

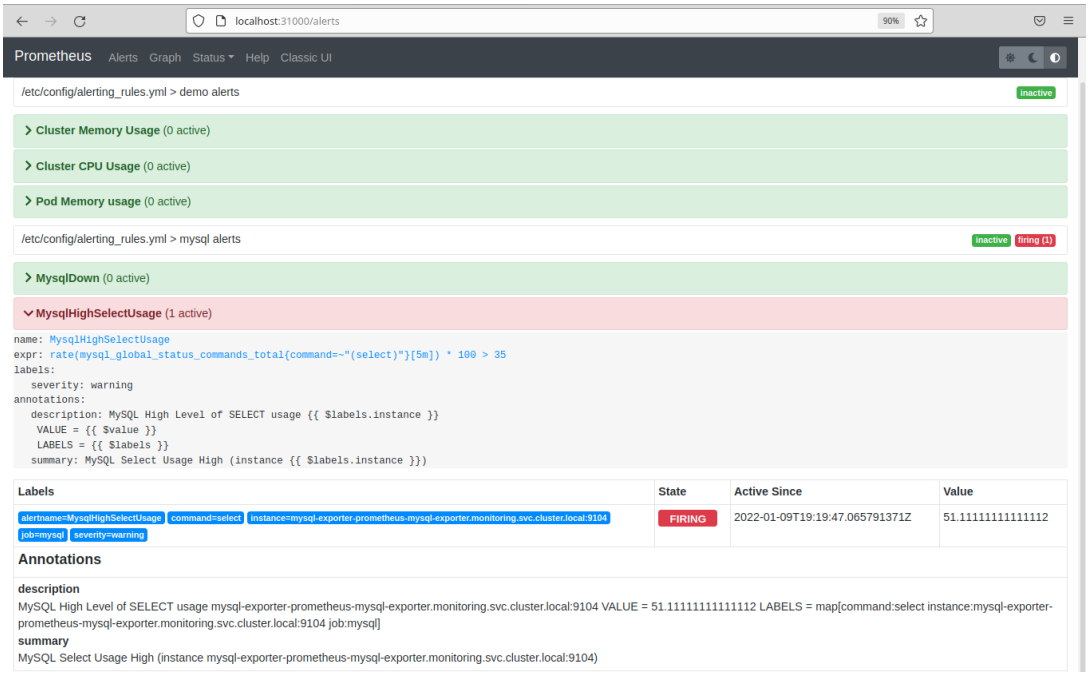
```
$ k apply -n monitoring -f ps-cm-with-rules2.yaml
```



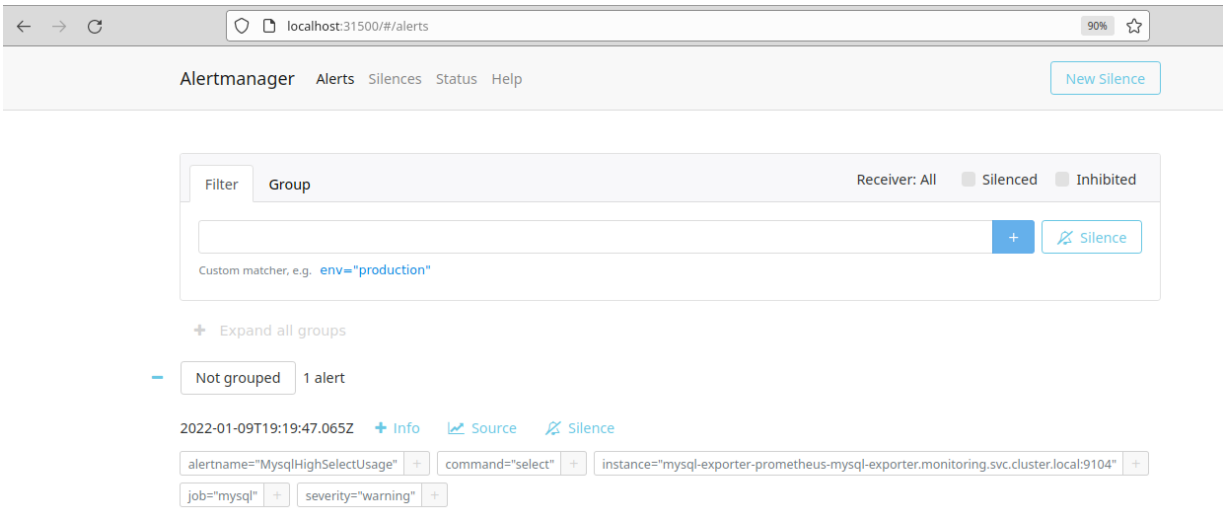
9. Let's see if we can get our alert to fire now. Run our loading program to simulate the load again with the rate query in effect. Execute the same script we used before again with 60 iterations and a 0.5 second wait in-between.

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

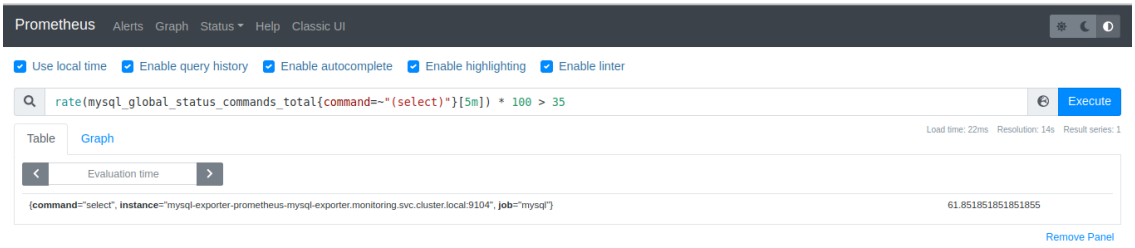
10. After this runs, after you refresh, on the Alerts tab, you should be able to see that the alert was fired. You can expand it to see details.



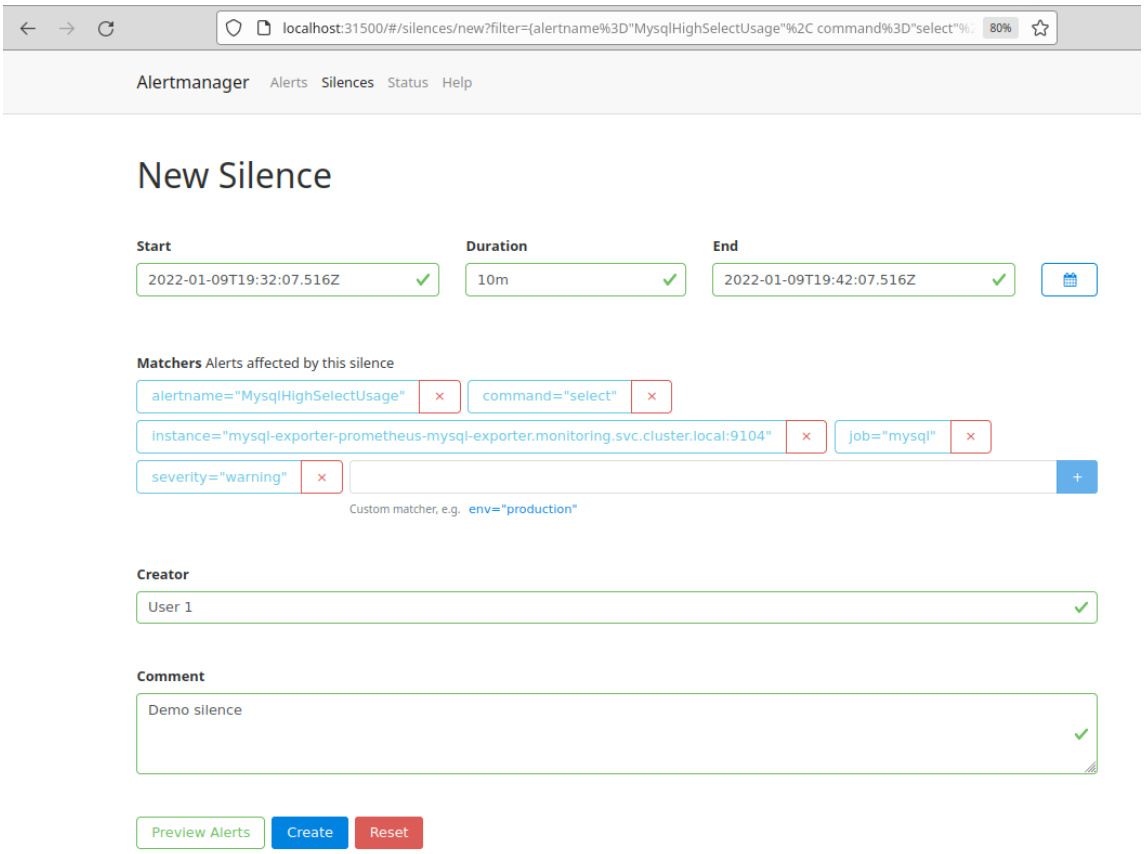
11. Now, that our alert has fired, we should be able to see it in the Alert Manager application. On this machine, it is exposed at node port 31500. Open up that location and take a look.



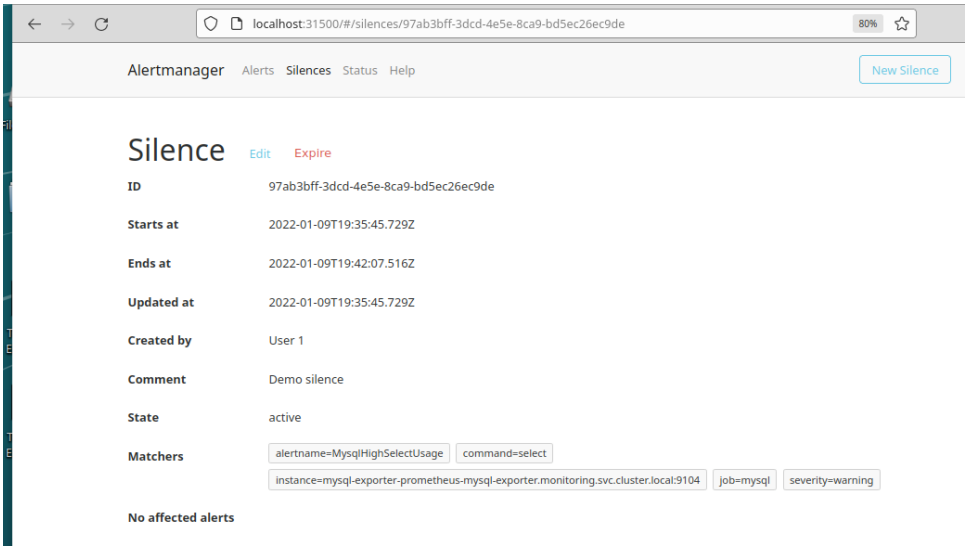
12. If you click on the "Source" link next to the timestamp and "+ Info" it will take you back to the main Prometheus query screen. When you get there, click on Execute and open up the Graph view.



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.



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.

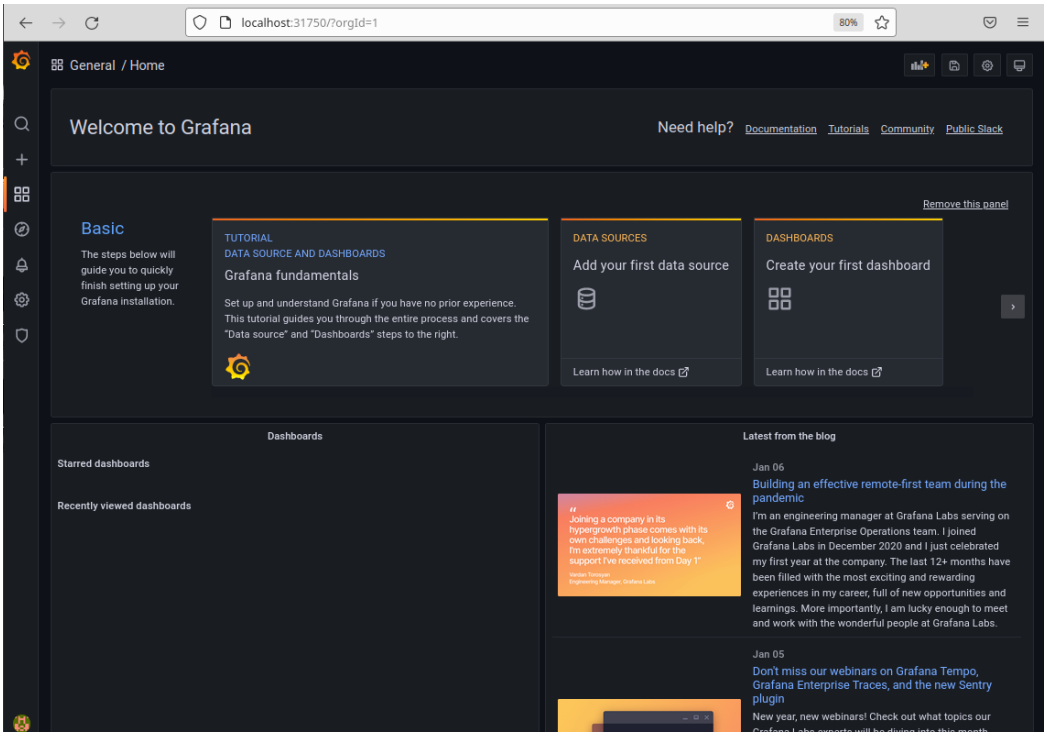


END OF LAB

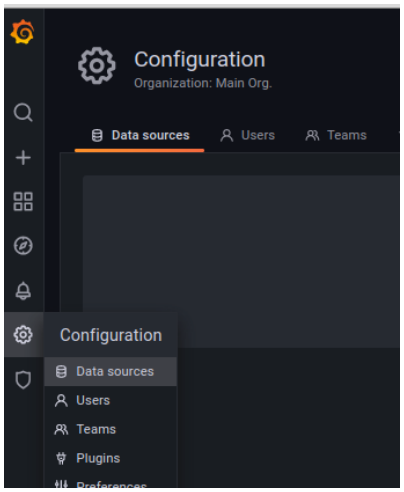
Lab 5 - Grafana

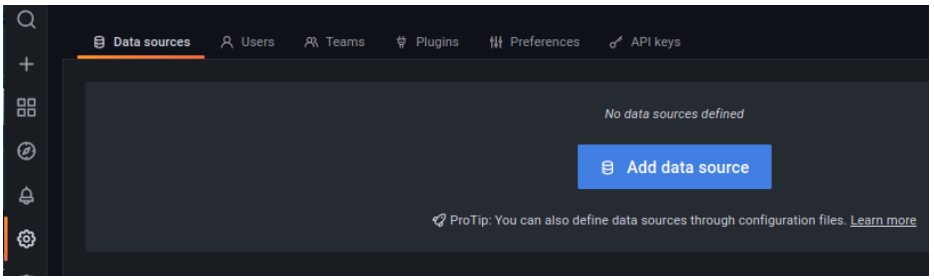
Purpose: In this lab, we'll see how to use Grafana to display custom graphs and dashboards for Prometheus data.

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".



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

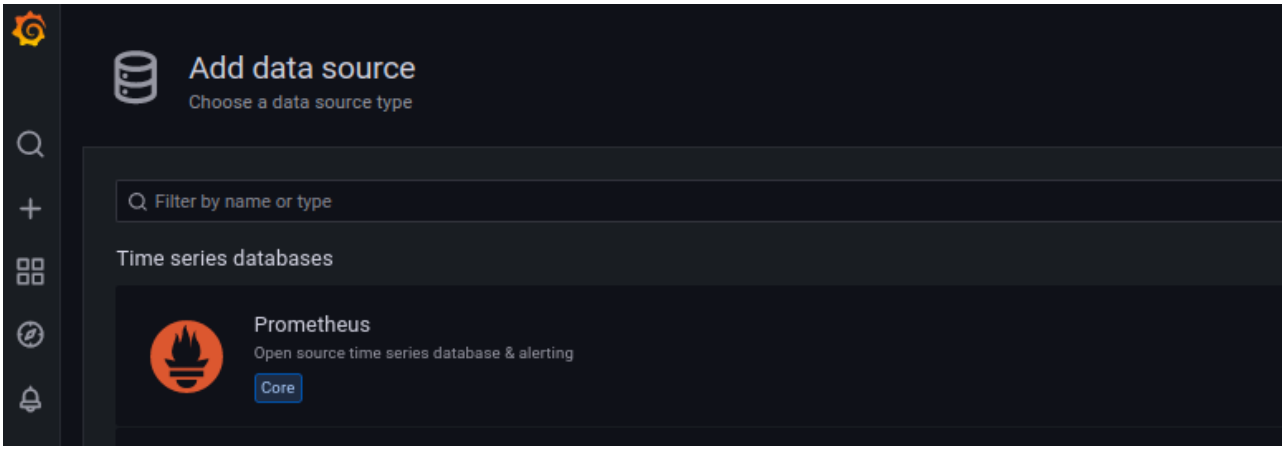


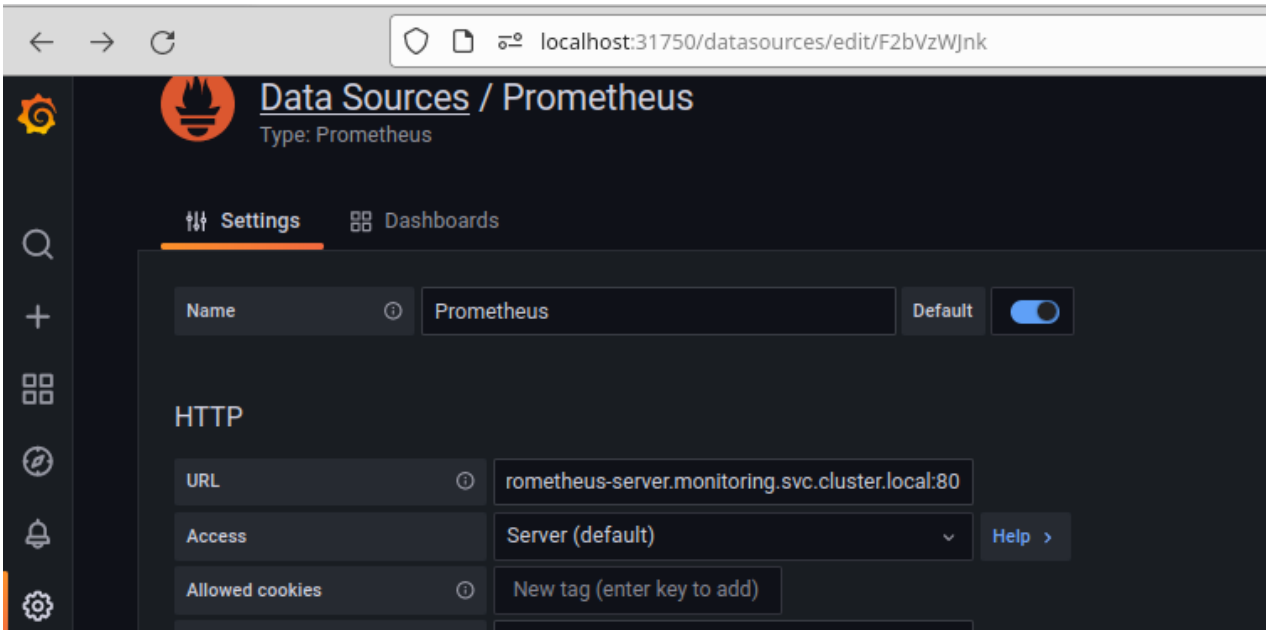


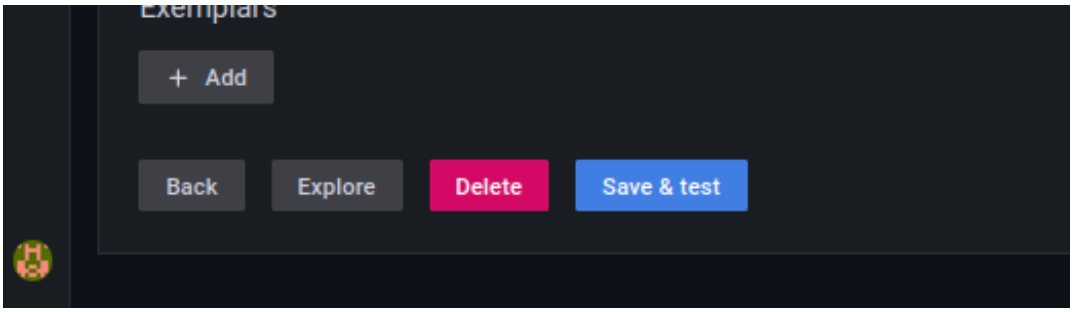
3. Select "Prometheus" and then for the HTTP URL field, enter

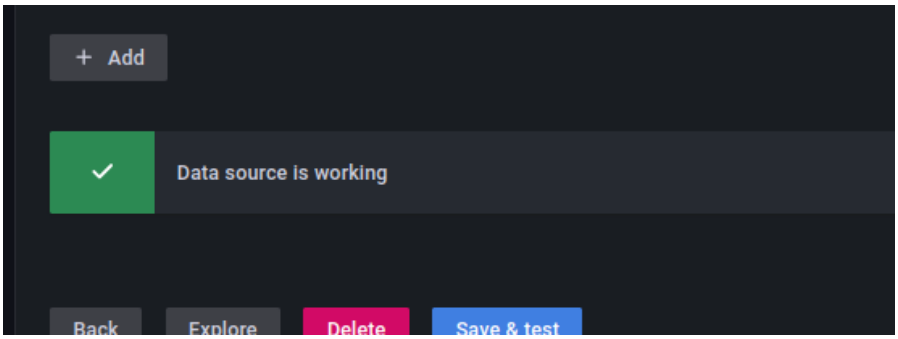
http://pc-prometheus-server.monitoring.svc.cluster.local:80

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

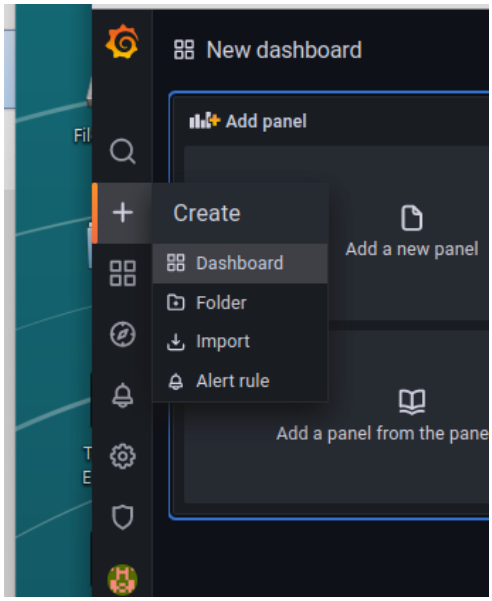


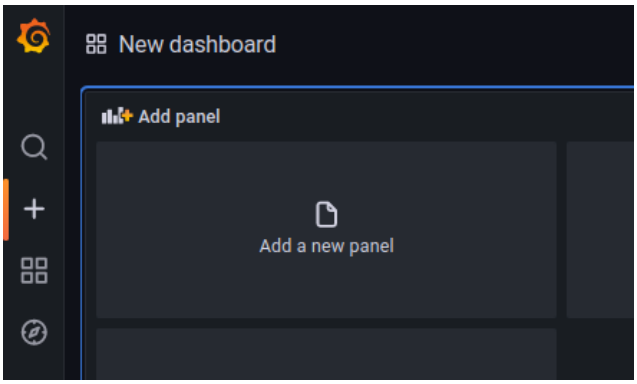


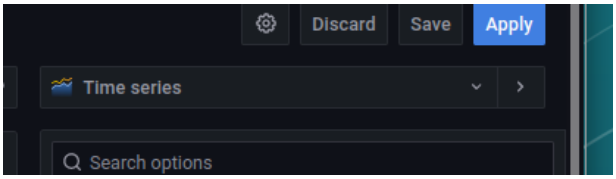




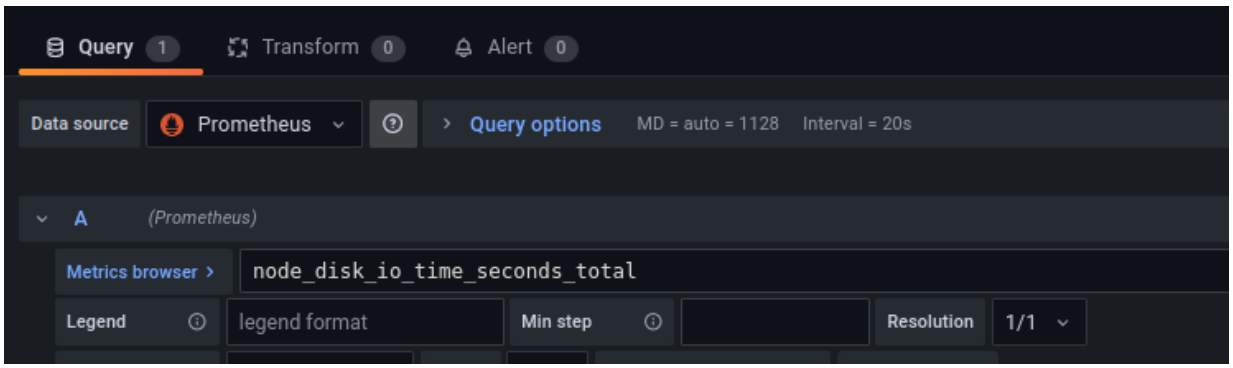
4. Now, let's create a simple dashboard for one of our mysql metrics. Click on the "+" sign on the left and select Dashboard from the menu. Then click on "Add a new panel". In the upper right, make sure "Time series" is selected for the type of visualization.





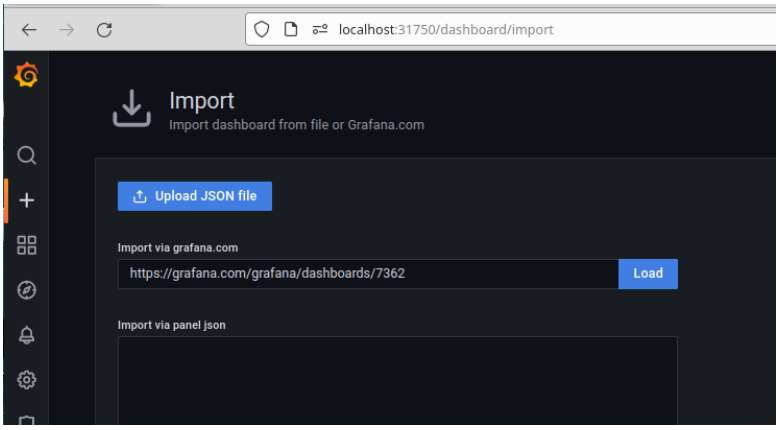


5. In the Metrics Browser section, start typing "node" and then pick a sample metric, such as "node_disk_io_time_seconds_total". Then click in the Panel and you should see a new chart.

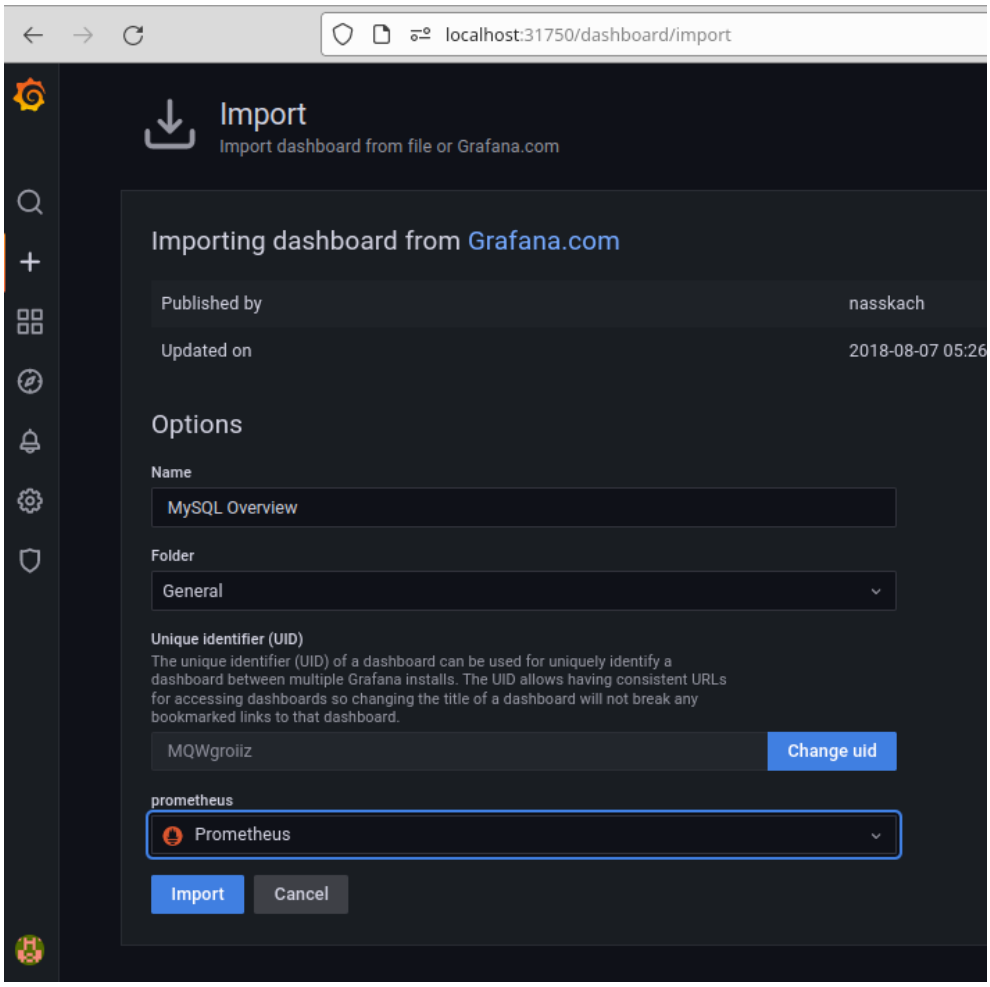


6. While we can create individual dashboards with Grafana, that can take a lot of time and effort. The community has already created a number of dashboards that we can just import and use. Let's grab one for mysql. Click on the "+" on the left side, then select "Import". (You can save or discard the 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.

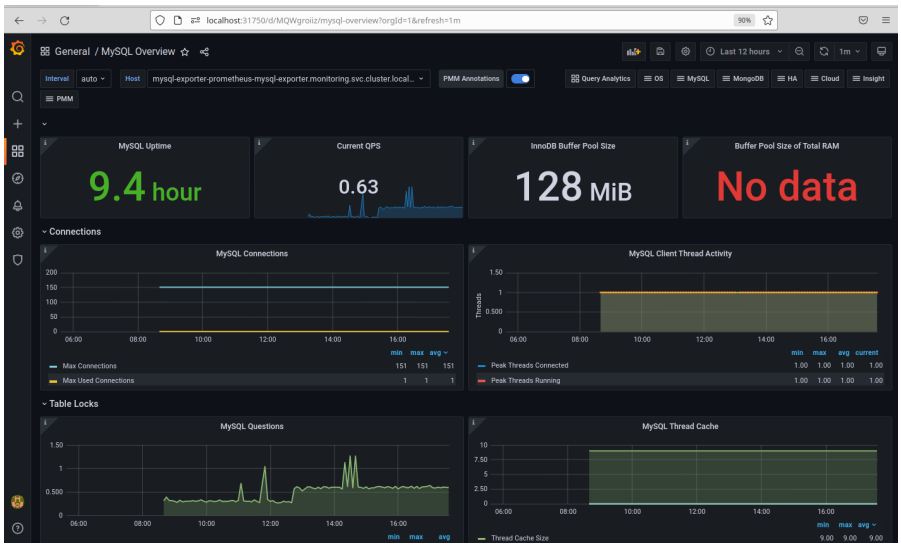
https://grafana.com/grafana/dashboards/7362



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.

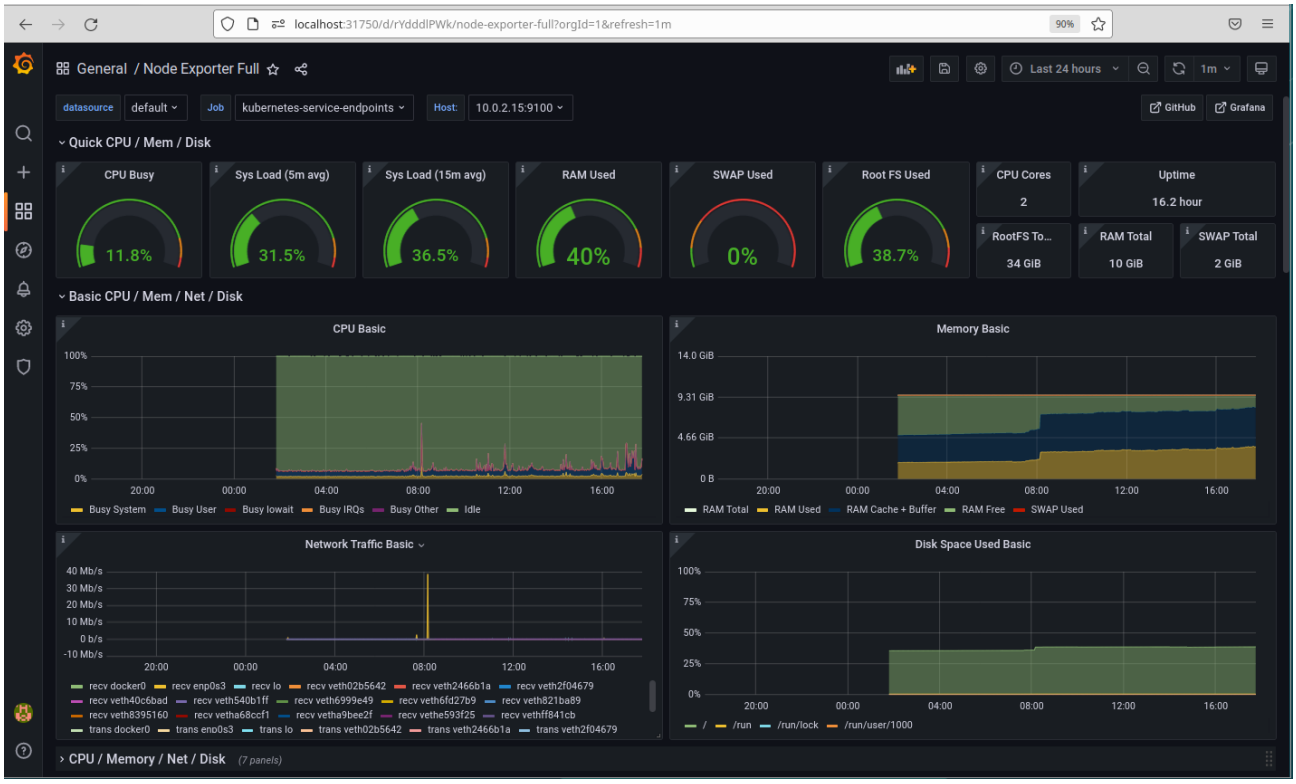


8. At this point, you should see a populated dashboard with a number of panels looking at the mysql exporter data from our system through Prometheus. You can scroll around and explore if you want.



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