**KUBERNETES CLUSTER MONITORING**

We are going to use various monitoring , visualization and their exporter integrations which are needed to monitor Kubernetes cluster in this article

We are going to install and configure below components as part of Kubernetes monitoring stack

* PROMETHEUS
* KUBE-STATE-METRICS
* NODE-EXPORTER
* GRAFANA
* ALERTMANAGER

***PREREQUISITES:***

* A Kube cluster with Administrator access on the cluster. if you have the kube config setup after installation of kube cluster on master. It should work
* Make sure you configure firewalls /security groups accordingly so that you can access the Grafana and prometheus UI

***INSTALLATION OF PROMETHEUS***

* Login to the Kubernetes cluster(exporting kubeconfig or setting kube config)
* git clone <https://github.com/leaddevops/prometheus-grafana.git>
* cd prometheus-grafana/**kubernetes-prometheus**/
* kubectl create -f .

This will create the following Kubernetes objects

1. **Namaspace** – monitoring
2. **Clusterrole** -- Which will help Prometheus pod to retrieve the metrics related to kubrrnetes cluster objects.
3. **Clusterrolebinding** – This will bind the cluster role to the default service account of monitoring namespace so that the prometheus pod will be able to retrieve metrics
4. **Deployment** – To keep the prometheus pod up and running. We are also restricting the pod to worker 1 in this demo since we are using local host volumes. This is done for persisting prometheus data

Few of the important arguments for prometheus deployment are

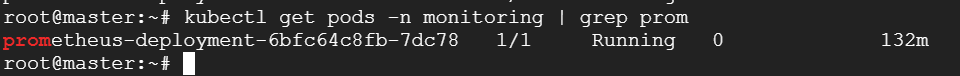
* storage.tsdb.retention.time -- timeframe prometheus need to keep the data..Prometheus will be cleaning up the data older than 6 hours. This will also help clean up the space in persistent volume
* config.file -- config file path of prometheus within the pod
* storage.tsdb.path -- path where scrapped data by prometheus is stored within the pod

1. **Service** – This is a Nodeport service which will allow us to access prometheus UI on the node port 30000
2. **Config map** – This is the prometheus configmap is created to externalize the prometheus configurations. We will understand deep dive about it once all the stack is installed.For extended explanation of prometheus configurations..Please check [here](https://github.com/leaddevops/prometheus-grafana/blob/main/kubernetes-prometheus/config.md)
3. **Persistent Volume and Persistent Volume claim** – We are creating manual persistent volume which will use host path of worker node and using static provisioning of PV’s. Once PV is created, we attach PV to the pod using PVC

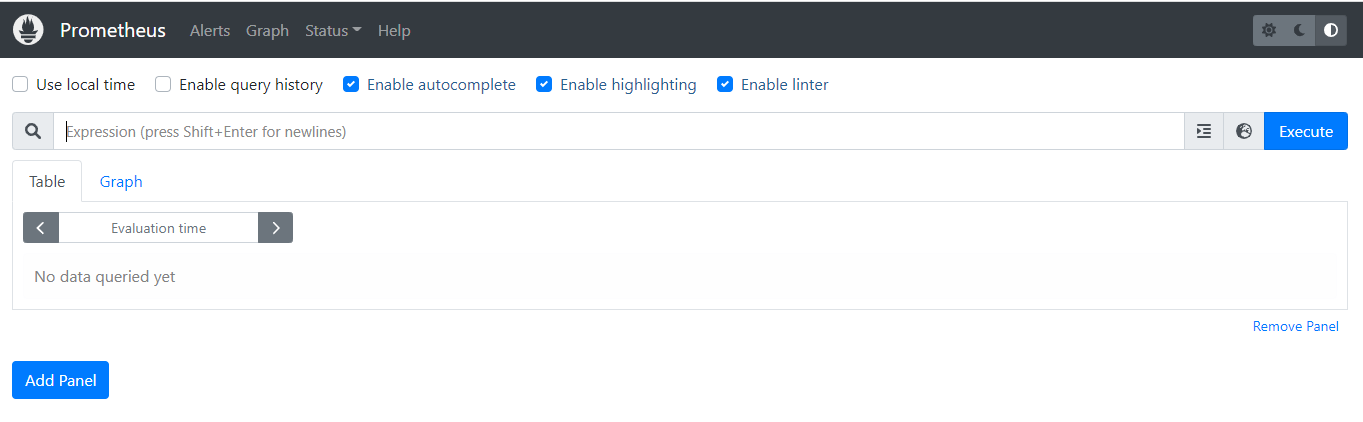
**Note**: In production environment you may be using dynamic provisioning for persistent storage

* Check if the prometheus pod is up and running

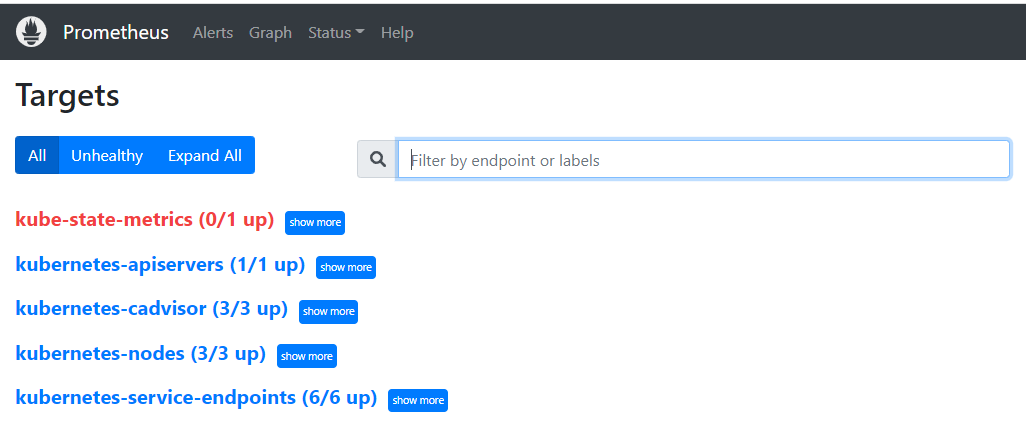
kubectl get pods -n monitoring



* At this point you should be able to access promtheus UI using below URL if the pod is up and running
  + URL : http://{Any kube node public IP}:30000/



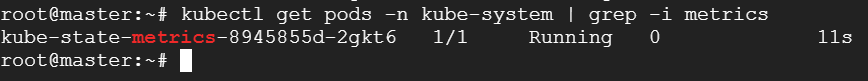
* You can also access the target endpoints to see the jobs at highlevel which are been scrapped by prometheus..You might notice that the kube-state-metrics is unhealthy which takes us to the second component which we are installing as part of this monitoring stack
  + URL : http://{Any kube node public IP}:30000/targets

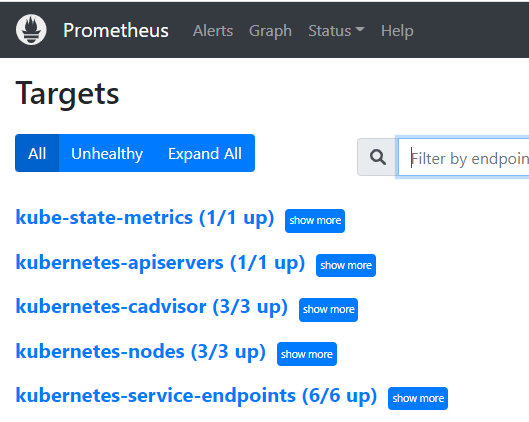


* Now that prometheus is installed on our kube cluster We have prometheus.yml connfigurations to scrape Kubernetes-apiserver caadvisor metrics, node level metrics in Kubernetes, services and pods. But kube-state-metrics component helps us gather the metrics of many other Kubernetes objects such as deployments, cronjobs, statefulsets etc.. which are not available by default

***INSTALLATION OF KUBE\_STATE\_METRICS***

* git clone <https://github.com/leaddevops/prometheus-grafana.git>
* cd prometheus-grafana/ kube-state-metrics
* kubectl create -f .
* This will create the following Kubernetes objects. All the objects with namesapces scope install in the kube-system namespace
  1. A Service Account
  2. Cluster Role – For kube state metrics to access all the Kubernetes API objects.
  3. Cluster Role Binding – Binds the service account with the cluster role.
  4. Kube State Metrics Deployment
  5. Service – To expose the metrics
* By default kube-state-metrics expose metrics endpoint on /metrics and the prometheus scrapes the metrics endpoint using static configs mentioned In prometheus.yml.
* Check for the pod status in kube-system namespace and once its up and running You should be seeing targets healthy in prometheus UI



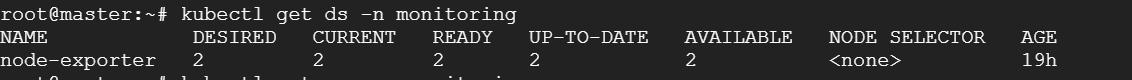


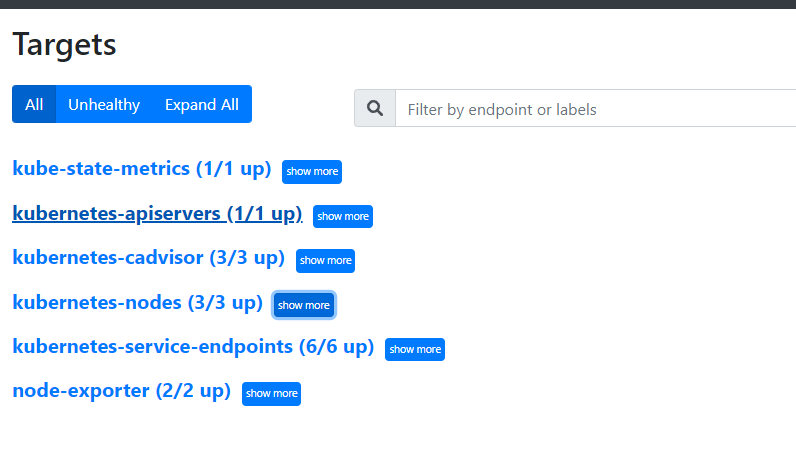
***INSTALLATION OF NODE\_EXPORTER***

* Node Exporter will provide all the Linux system-level metrics of all Kubernetes nodes. The metrics which we have been collecting so far tell us a lot about node at Kubernetes stand point but not really at the OS level..If you wish to see the metrics at OS level like network I/O, disk utilization, uptime etc.. We need to install this node exporter extension
* git clone <https://github.com/leaddevops/prometheus-grafana.git>
* cd prometheus-grafana/ kubernetes-node-exporter/
* kubectl create -f .
* This will create the following Kubernetes objects.

1. Daemonset in the namespace monitoring which will collects all the Linux system metrics and exposes them via /metrics endpoint on port 9100
2. Service that listens on port 9100 and points to all the daemonset node exporter pods. We would be monitoring the service endpoints (Node exporter pods) from Prometheus

* Check for the pod and service status under namespace monitoring and you should also be seeing node-exporter in targets now

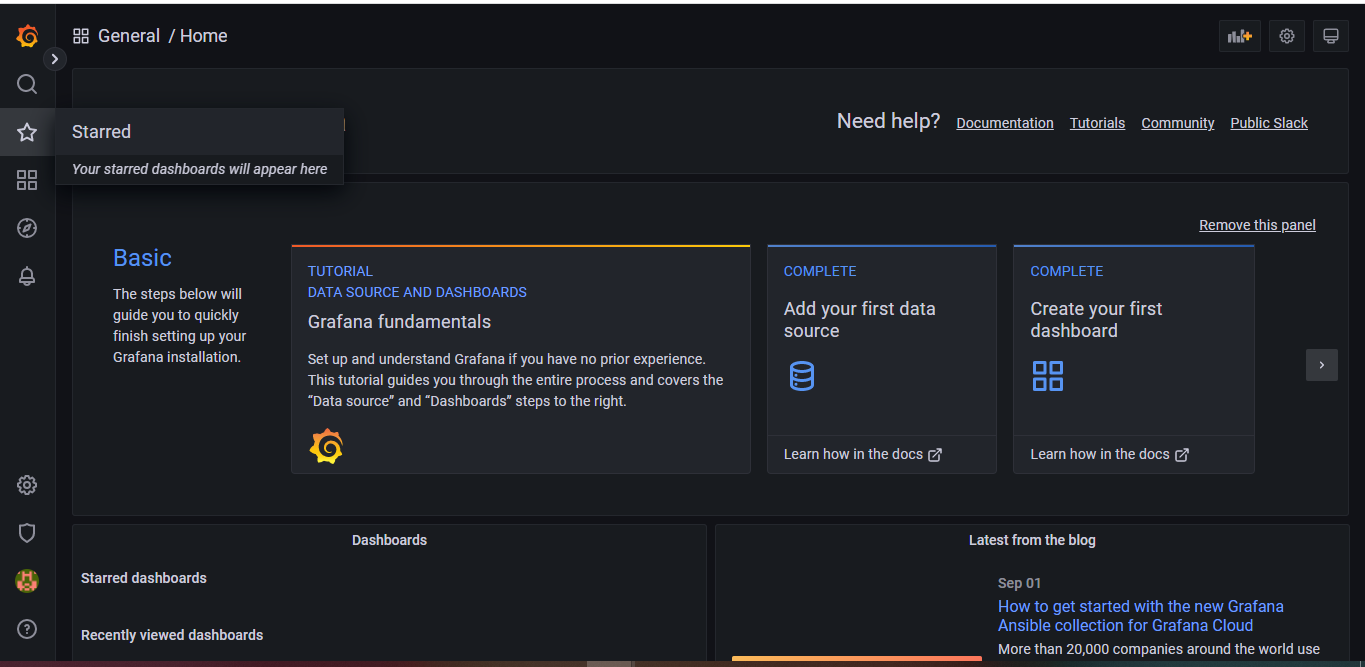




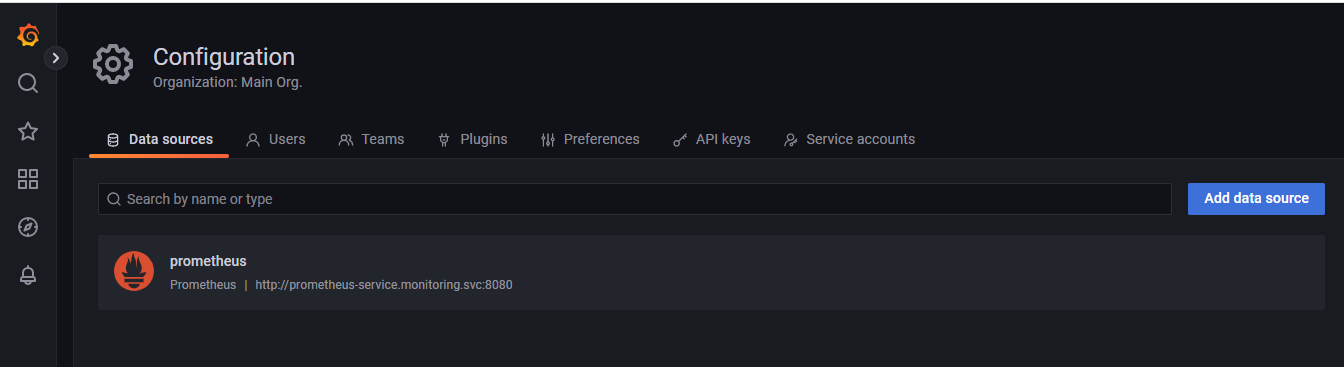
***INSTALLATION OF GRAFANA***

* Now that we have all the metrics scrapped by prometheus lets install Grafana for better and simplified visualization of these metrics
* git clone <https://github.com/leaddevops/prometheus-grafana.git>
* cd prometheus-grafana/ kubernetes-grafana
* kubectl create -f .
* This will create the following Kubernetes objects.
* **Deployment** – To keep the Grafana pod up and running. We are also restricting the pod to worker 2 in this demo since we are using local host volumes. This is done for persisting Grafana db which consists of dashboards
* **Service** – This is a Nodeport service which will allow us to access Grafana UI on the node port 32000
* **Config map** – This is the datasource configmap which holds the configurations of prometheus which we wish to connect In this case we are doing an inservice connection to prometheus as they are within same cluster.Note that we can also give multiple datasource configurations ..This will help persist the datassource configurations even if the pod restarts
* **Persistent Volume and Persistent Volume claim** – We are creating manual persistent volume which will use host path of worker node and using static provisioning of PV’s. Once PV is created, we attach PV to the pod using PVC.This will persist the Grafana db data even after pod restarts
  + - **Note**: In production environment you may be using dynamic provisioning for persistent storage
* Upon succsefull installation you should see the Grafana pod up and running and should be able to access Grafana UI
  + URL : http://{Any kube node public IP}:32000/

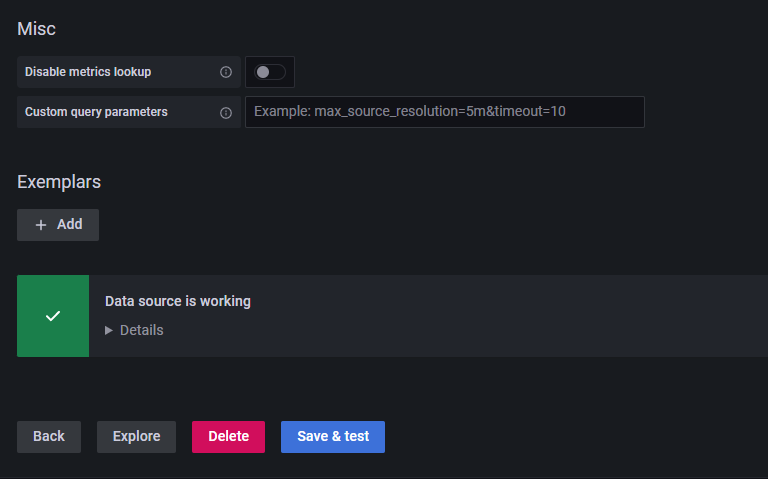




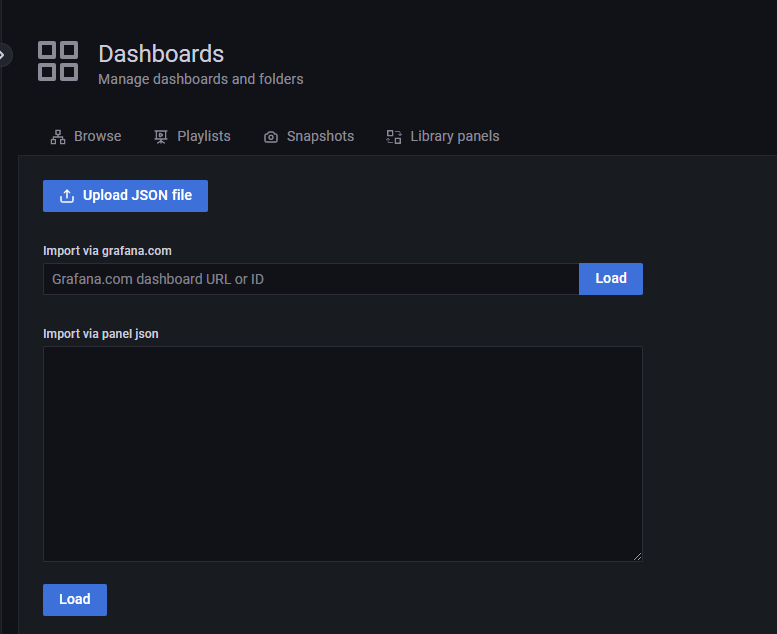
* You should see that the datasource for prometheus is already configured and you do not have to add it
* Expand the left nav under configuration -> Datasources you should prometheus data source as below which is been picked up from config map



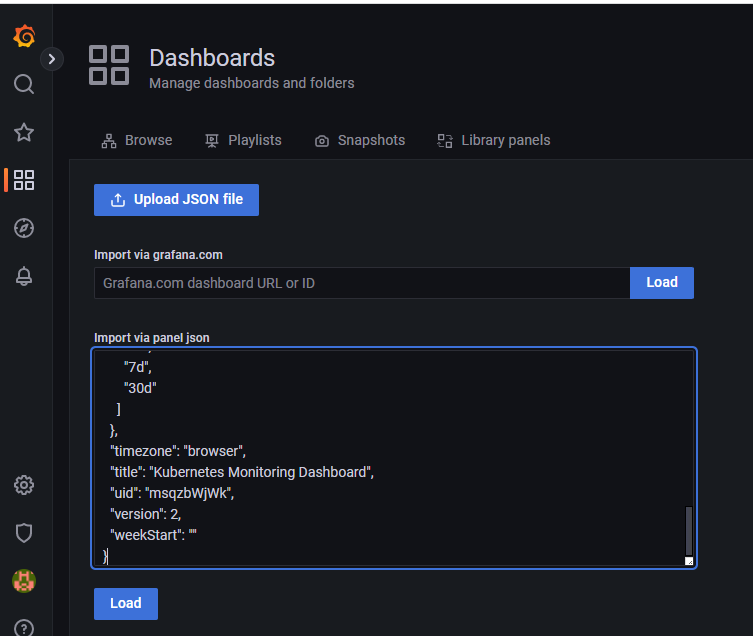
* Click on the datasource to do a do a quick test to check if Grafana is able to connect to prometheus which we have installed. You should see a working response as shown below



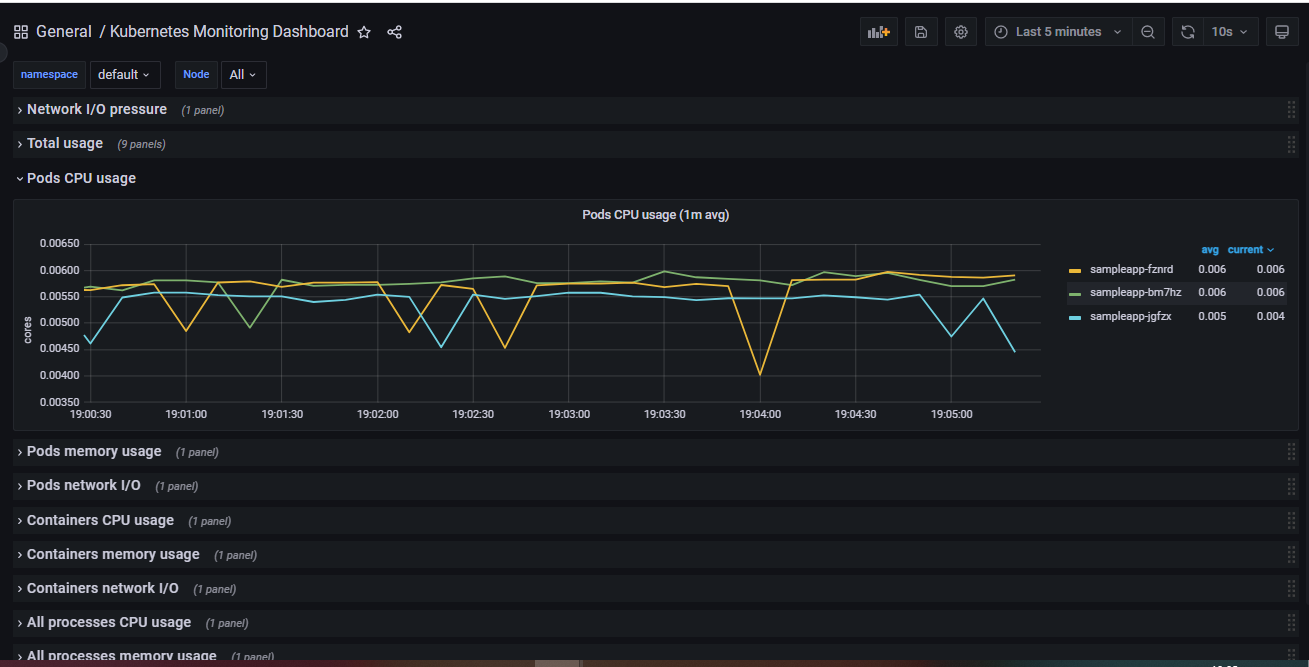
* Lets go ahead and import some helpful Kubernetes dashboards which are in the github repo. From the left Nav go to Dashboards -> Import

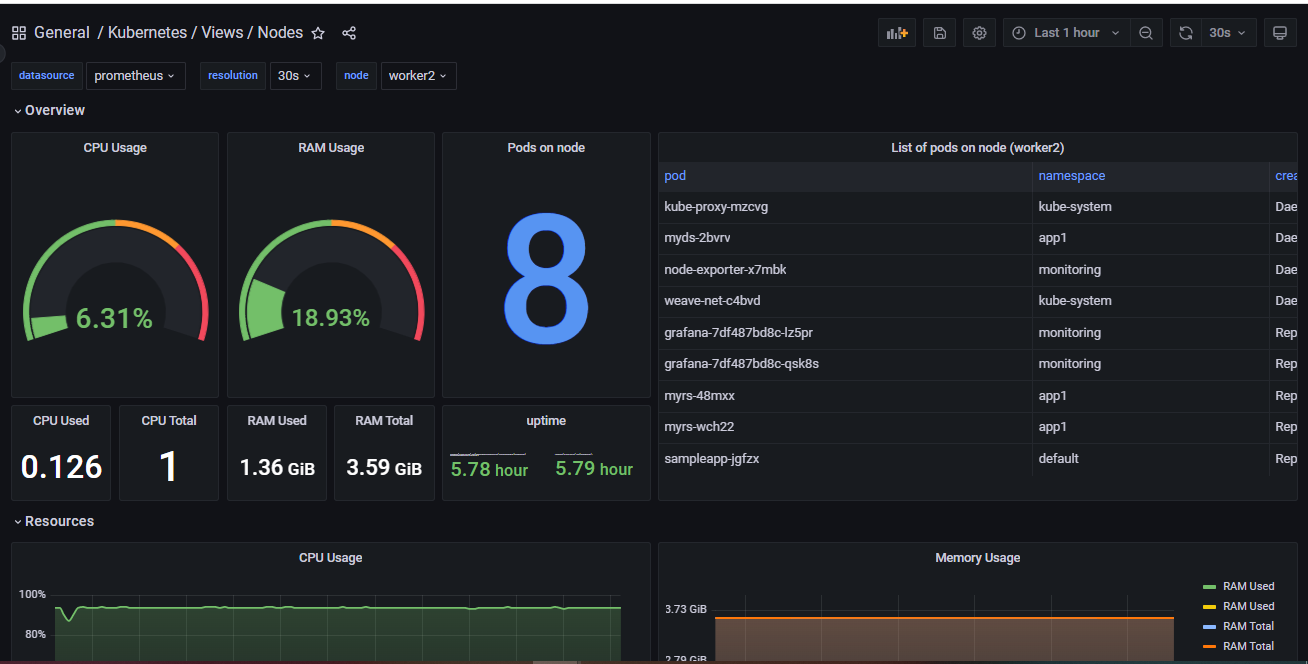


* We are going to copy the json files which are available in gihub repo under below folder
  + <https://github.com/leaddevops/prometheus-grafana/tree/main/kubernetes-grafana/dashboards>
* Copy the json and paste into the Json panel as shown and click on Load and the Import
  + Tip: Try taking the raw content of the json.It helps with indentation



* You can import all the thres dashboards using the same process and there are lot more available in Grafana labs if you would wish to try
* After importing all the three dashboards you should see data related to nodes/cluster usage/pods usage and lot more interesting data about Kubernetes clusters and pods within it



* 
* 

***INSTALLATION OF ALERT MANEGER***

* git clone <https://github.com/leaddevops/prometheus-grafana.git>
* cd prometheus-grafana/ Kubernetes-alert-manager
* kubectl create -f .

This will create the following Kubernetes objects.

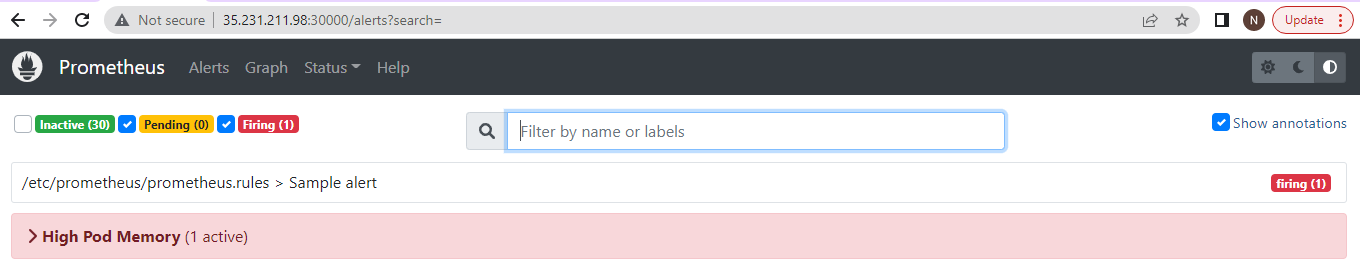
* + Deployment for alert manager
  + Config map holding the receivers configurations of alert manager.Alert manager supports multiple receivers like email,opsgenie,slack,pagerduty etc..In this demo we will be seeing email receiver(GMAIL)
  + Config map holding the default email templates.This can be modified with a little knowledge in HTML
  + Service to expose alert manager configs
* Once the installation is success you should be able to access alert manager service via below URL



* + URL : http://{Any kube node public IP}:31000/



* We will not be using individual alert URL much here. We will be mostly using prometheus UI to visualize the alerts which will look like below.To navigateto this click on Alerts on top left



* There are three types of alerts here

**Inactive**: These are the alerts which were fired/pending earlier and are no longer active now

**Pending**: This is that state of an alert that has been active for less than the configured threshold duration for example I configured my alert with time period of 10m if the conditions hasn’t met 10m yet.we see the alert here

**Firing:** This is the alert that has been active for longer than the configured threshold duration and is currently notifying via receivers you configured.

**Note:** This high pod memory alert is asample alert to test if the receiver configs are working as expected

***PROMETHEUS CONFIGURATIONS TO SCRAPE ALERT-MANAGER :***

Once we have the above installation complete we need to make sure the below data is in prometheus config map which we created as part of prometheus installation

***alerting:***

***alertmanagers:***

***- scheme: http***

***static\_configs:***

***- targets:***

***- "alertmanager.monitoring.svc:9093"***

***rule\_files:***

***- /etc/prometheus/prometheus.rules***

We are essentially asking promethus to scrape alert manager service via static configs

All the alerting rules have to be present on Prometheus config based on your needs. It should be created as part of [the Prometheus config map](https://github.com/bibinwilson/kubernetes-prometheus/blob/master/config-map.yaml) with a file named prometheus.rules..This is the same configuration we are adding under rule\_files

Sample config map can be found under the kubernetes-prometheus directory. A detailed explanation of each rule can be seen [here](https://github.com/leaddevops/prometheus-grafana/blob/main/kubernetes-alert-manager/AlertManagerConfigmap.yaml)

***ALERT MANAGER RECEIVER CONFIGS:***

As discussed earlier there are many receivers alert mnager supports for notifications

Out of which we are choosing email and Im going to use GMAIl in this demo

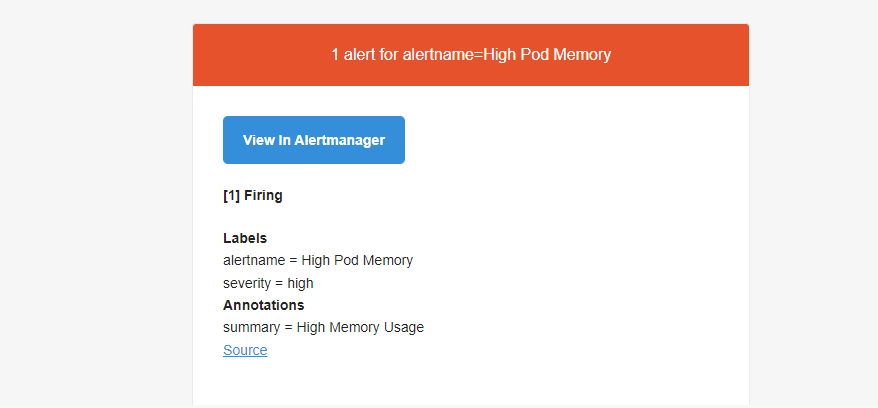
These configs are driven through a config map named alertmanager-config in this demo

Detailed explanation of each field is mentioned [here](https://github.com/leaddevops/prometheus-grafana/blob/main/kubernetes-prometheus/config.md)

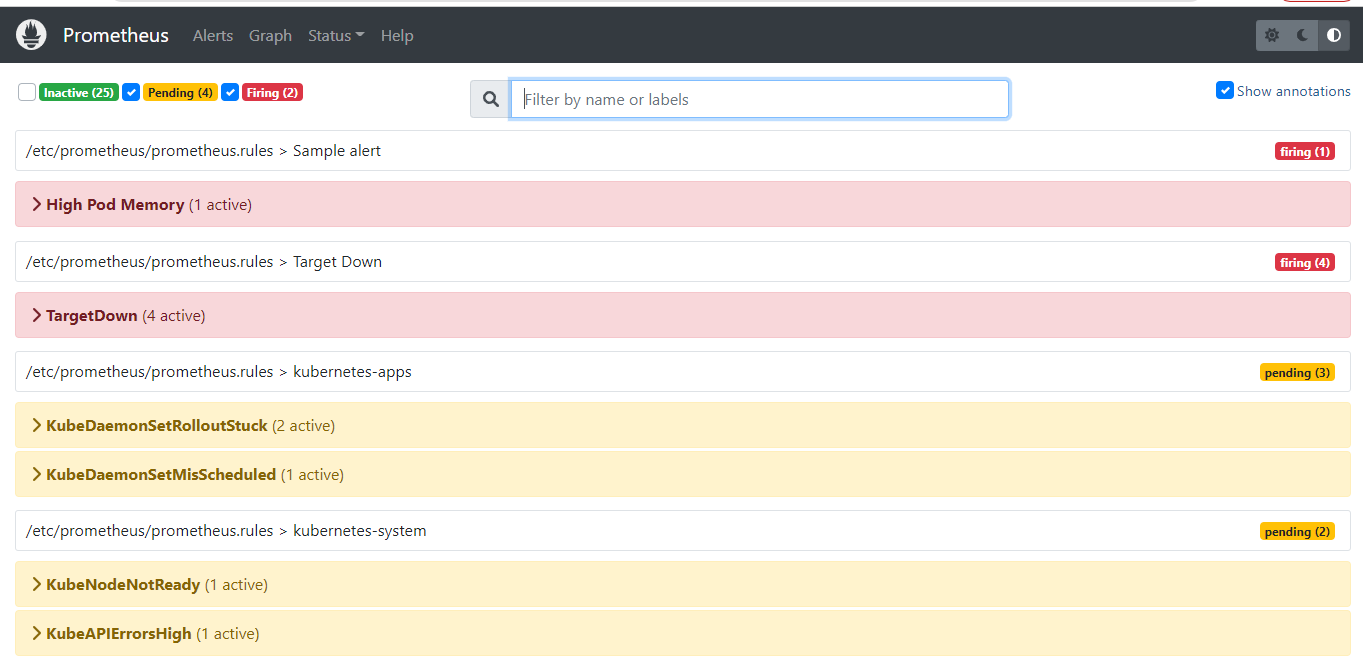
Upon successful configurations of receiver snad installation of Alert manager you should now receive a sample alert via gmail upon configured email.More about receiver configs can be seen [here](https://github.com/leaddevops/prometheus-grafana/blob/main/kubernetes-alert-manager/AlertManagerConfigmap.yaml)

**Note**: We used Google cloud instance to make it easier in communication

Snippets of mail for sample alert setup are as follws



To see more alerts or test few of the configs given. You can shut down one of the worker node and you would see many alerts in Pending state or firing state depending on time interval configured for alerts



**Types of RULES**

Prometheus supports two types of rules which may be configured and then evaluated at regular intervals: recording rules and [alerting rules](https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/).

***Recording rules*** allows us to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series. Querying the precomputed result will then often be much faster than executing the original expression every time it is needed.This is especially useful for dashboards, which need to query the same expression repeatedly every time they refresh. Recording and alerting rules exist in a rule group.

**Alerting rules** allow you to define alert conditions based on Prometheus expression language expressions and to send notifications about firing alerts to an external service.

More info on the syntax and configuration is explained [here](https://github.com/leaddevops/prometheus-grafana/blob/main/kubernetes-prometheus/config.md)

All the rules can be seen if we navigate to status -> Rules under the UI

