Resource monitoring for Kubernetes and 3.x Environments

Note:- For VM-based / Bare metal deployments such as **1.x, vsilb, R3** etc, use **nmon** for resource monitoring - Refer Confluence Page: Capture CPU and memory usage Using **nmon**

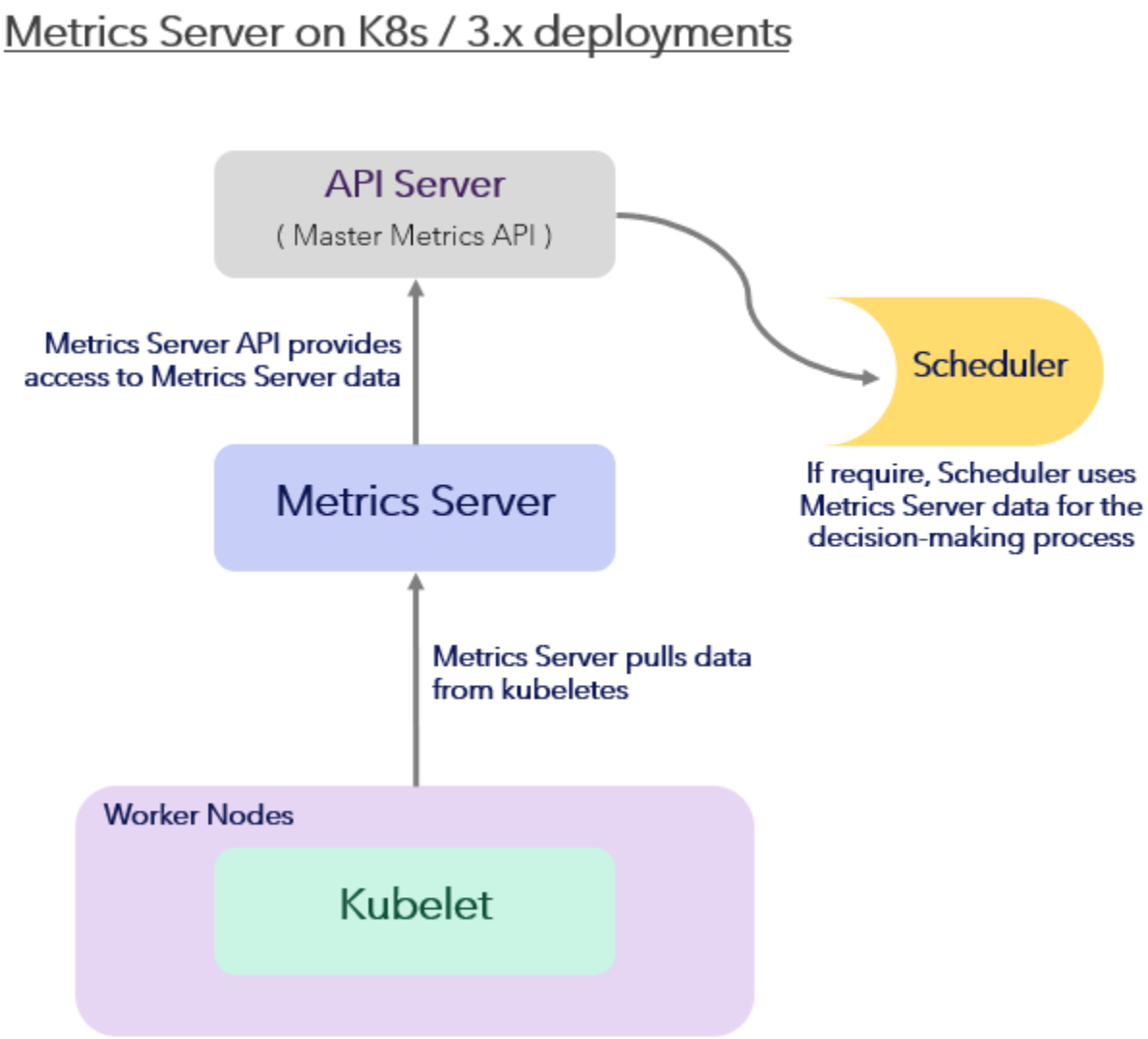
Resource monitoring on Kubernetes can achieve in many ways, listed couple of options below.

1. **On Demand - CLI monitoring via Kubernetes Metrics Server**
2. **Time Series, HTTP/Web based monitoring via Prometheus**

1. On Demand - CLI monitoring via Kubernetes Metrics Server

Kubernetes Metrics server is a cluster add-on that allows you to collect resource metrics for on-demand monitoring, autoscaling pipelines from Kubernetes. After getting metrics, it delivers the aggregated metrics to the Kubernetes API server via the Metric API.

*** Metrics Server does not store metrics, so we can't use it to retrieve historical values and predict tendencies.



In our Comtech's Environment Metrics Server is available by default and its part of all 3.x products/flavors (R4, 3.x VzW, R2 and including WEA)

Make sure *metrics-server* is running in your cluster.

```
[centos@zc429clsg01kma001 dm ~]$ kubectl get deploy,svc -n kube-system | egrep metrics-server
deployment.apps/metrics-server            1/1     1            1           2d14h
service/metrics-server      ClusterIP   10.102.96.80     <none>        443/TCP                      2d14h
```

View metric snapshots using kubectl top

Using Metrics Server you can retrieve compact metric snapshots from the Metrics API using **kubectl top**. The kubectl top command returns current CPU and memory usage for a cluster's pods or nodes, or for a particular pod or node if specified.

For example, you can run the following command to display a snapshot of near-real-time resource usage of all cluster nodes:

```
[centos@zc429clsg01kma001 dm]$ kubectl top node
NAME                CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
zc429clsg01kma001   321m         16%    4008Mi          54%
zc429clsg01kwo001   2121m        13%    11810Mi         12%
zc429clsg01kwo002   1913m        11%    6078Mi          6%
zc429clsg01kwx001   1967m        12%    14545Mi         15%
zc429clsg01kwx002   1128m        7%     13950Mi         14%
[centos@zc429clsg01kma001 dm]$
```

This output shows three worker nodes in a cluster. Each line displays the total amount of CPU (in cores, or in this case m for millicores) and memory (in MiB) that the node is using, and the percentages of the node's allocatable capacity those numbers represent.

Likewise, to query resource utilization by pods in the namespace, run the command below (note that if you do not specify a namespace, the `default` namespace will be used):

```
[centos@zc429clsg01kma001 dm]$ kubectl top pod
NAME                                              CPU(cores)   MEMORY(bytes)
gmlccli-85864d9f6b-xjrcf                          1m           0Mi
healthcheck-pod-5db9d8f96-t8p8t                   418m         69Mi
healthproxy-67979bc4db-mqskt                      6m           28Mi
kafka-0                                           71m          6445Mi
kafka-1                                           231m         6495Mi
kafka-2                                           337m         6247Mi
kafka-zookeeper-0                                 8m           365Mi
kafka-zookeeper-1                                 5m           358Mi
kafka-zookeeper-2                                 6m           362Mi
postgres-operator-6749c45db6-f6x9c                1m           23Mi
prometheus-kube-prometheus-operator-5654998cff-dm6w5   1m      37Mi
prometheus-prometheus-kube-prometheus-prometheus-0     8m      603Mi
reloader-reloader-5b94676964-lxzsx                1m           21Mi
vmlc-redis-node-0                                 44m          9Mi
vmlc-redis-node-1                                 26m          5Mi
vmlc-redis-node-2                                 24m          5Mi
zc429clsg01mlc-asla-consumer-76664bd7c4-nnq8j     11m          803Mi
zc429clsg01mlc-connectionconsumer-7cbd5c9868-s92tg   13m       418Mi
zc429clsg01mlc-dashboard-6c79dccfb-lj7rd          11m          550Mi
zc429clsg01mlc-egmlc1-79d6754974-cvfh7            233m         54Mi
zc429clsg01mlc-egmlc2-768c7f5bbc-558d5            28m          39Mi
.....
```

We can also Sort pods by **memory**

```
[centos@zc429clsg01kma001 dm]$ kubectl top pods --sort-by=memory
NAME                                              CPU(cores)   MEMORY(bytes)
kafka-1                                           227m         6536Mi
kafka-0                                           79m          6482Mi
kafka-2                                           324m         6281Mi
```

```
zc429clsg01mlc-asla-consumer-76664bd7c4-nnq8j          10m          804Mi
zc429clsg01mlc-trace-consumer-74b94bb958-gqdcb         54m          761Mi
zc429clsg01mlc-key-trace-consumer-7f646b9bbf-v58x7     10m          726Mi
zc429clsg01mlc-k8sevent-consumer-6cdbc5958-mgt2f       12m          709Mi
prometheus-prometheus-kube-prometheus-prometheus-0     82m          586Mi
zc429clsg01mlc-dashboard-6c79dccfb-lj7rd               10m          550Mi
dash-redis-node-0                                      54m          548Mi
dash-redis-node-1                                      28m          545Mi
dash-redis-node-2                                      36m          513Mi
dlp-postgresql-0                                       22m          491Mi
zc429clsg01mlc-eventservice-54d86f6968-mb7xd           182m         456Mi
zc429clsg01mlc-vessyslogtranslator-6c6f54c8cb-g2rkb    14m          448Mi
zc429clsg01mlc-vesconsumer-76db57cf9b-flgct            12m          425Mi
.....
```

Sort pods by **cpu** usage

```
[centos@zc429clsg01kma001 ~]$ kubectl top pods --sort-by=cpu
NAME                                               CPU(cores)   MEMORY(bytes)
kafka-2                                            683m         6052Mi
zc429clsg01mlc-slc-5b7785d865-jwf75                576m         61Mi
zc429clsg01mlc-slc-5b7785d865-xt6gv                576m         57Mi
healthcheck-pod-5db9d8f96-t8p8t                    403m         69Mi
zc429clsg01mlc-spc-6b8c9fb85-4ckqr                 362m         42Mi
zc429clsg01mlc-spc-6b8c9fb85-tzlm2                 354m         41Mi
kafka-1                                            283m         6139Mi
zc429clsg01mlc-egmlc1-79d6754974-cvfh7             226m         56Mi
zc429clsg01mlc-leagent-77b5768c97-mjf2k            225m         793Mi
zc429clsg01mlc-leagent-77b5768c97-7dwsn            223m         797Mi
zc429clsg01mlc-geodeticldrconsumer-57ddf87d45-9s58d   207m         956Mi
zc429clsg01mlc-msidldrconsumer-69664c485d-z5578    207m         923Mi
zc429clsg01mlc-locctl-6f5945457-mtkzk              193m         45Mi
zc429clsg01mlc-omproducer-bf555fc75-qtvr4          185m         419Mi
.....
```

we can also get pods/nodes on all namespaces using

`kubectl top pods --all-namespaces --sort-by=memory`

Query resource allocations with kubectl describe

If you want to see details about the resources that have been *allocated* to your nodes, rather than the current resource usage, the `kubectl describe` command provides a detailed breakdown of a specified pod or node. This can be particularly useful to list the resource requests and limits of all of the pods on a specific node. For example, to view details on one of the hosts returned by the `kubectl top node` command above, you would run the following:

```
[centos@zc429clsg01kma001 dm]$ kubectl describe node zc429clsg01kwx001
Name:               zc429clsg01kwx001
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=zc429clsg01kwx001
.....
  Namespace                  Name                                                  CPU Requests  CPU Limits
Memory Requests  Memory Limits  Age
  ---------                  ----                                                  ------------  ----------
---------------  -------------  ---
..
  default                    healthcheck-pod-5db9d8f96-t8p8t                       250m (1%)     1 (6%)
100Mi (0%)       300Mi (0%)     2d14h
  default                    zc429clsg01mlc-connectionconsumer-7cbd5c9868-s92tg    50m (0%)      200m (1%)
700Mi (0%)       2000Mi (2%)    2d14h
  default                    zc429clsg01mlc-dashboard-6c79dccfb-lj7rd              200m (1%)     1 (6%)
512Mi (0%)
 .....
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
```

```
  Resource           Requests     Limits
  --------           --------     ------
  cpu                1635m (10%)  6500m (40%)
  memory             8171Mi (8%)  22018Mi (22%)
  ephemeral-storage  15Gi (6%)    15Gi (6%)
  hugepages-2Mi      0 (0%)       0 (0%)
Events:              <none>
```

Note that `kubectl describe` returns the percent of total available capacity that each resource request or limit represents. These statistics are not a measure of *actual* CPU or memory utilization, as is returned by `kubectl top`. (Because of this difference, the `kubectl describe` command will work even in the absence of Metrics Server.)

Inside Containers with top

To peek inside your containers for monitoring the processes running inside them, we can use the popular Linux command: **`top`**. The **`top`** command allows you to monitor the processes and their resource usage on Linux, and it is installed by default on every Linux distro. Our plan to peek inside the containers of a pod is straightforward. We will get a shell to a running container and run the **`top`** command in the non-interactive mode in it as follows:

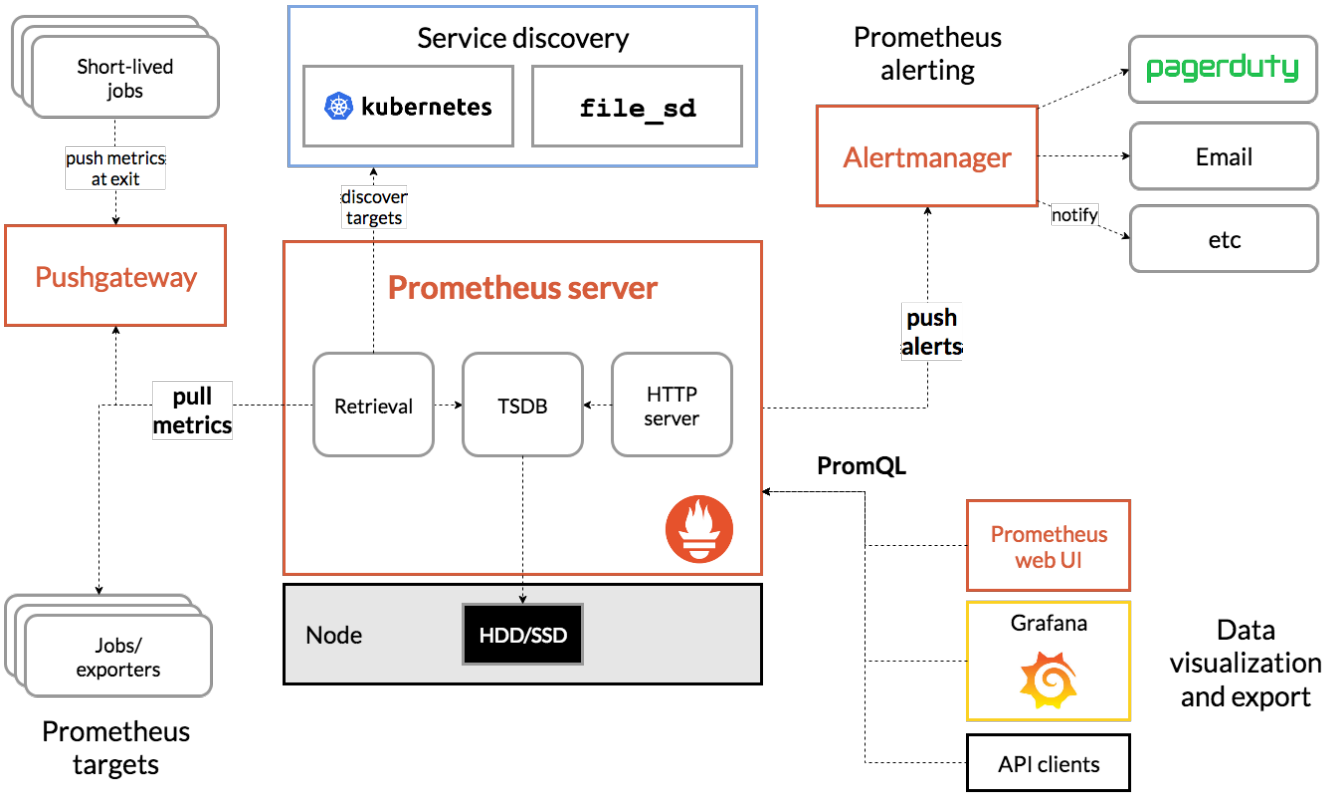
```
[centos@zc429clsg01kma001 ~]$ kubectl exec zc429clsg01mlc-egmlc1-79d6754974-cvfh7 -- top -
bn1
Defaulted container "egmlc" out of: egmlc, sdmgr, config, sdmgr-init (init), init (init)
top - 15:17:31 up 2 days, 15:55,  0 users,  load average: 1.89, 1.69, 1.97
Tasks:   8 total,   1 running,   7 sleeping,   0 stopped,   0 zombie
%Cpu(s): 10.0 us,  4.2 sy,  0.0 ni, 84.6 id,  0.0 wa,  0.0 hi,  1.2 si,  0.0 st
KiB Mem : 98834544 total, 75991344 free,  7730764 used, 15112436 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 90394496 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
31008 tcsapp    20   0 7171952  35368  17812 S  20.0  0.0 626:04.39 egmlc
    1 65535     20   0     956      4      0 S   0.0  0.0   0:00.04 pause
  953 root      20   0  295996  49064  19064 S   0.0  0.0  33:41.54 node
  973 root      20   0   11828   1532   1220 S   0.0  0.0   0:00.01 container_+
  984 root      20   0    6532    424    324 S   0.0  0.0   0:00.00 inotifywait
 3926 tcsapp    20   0   56060   1896   1432 R   0.0  0.0   0:00.01 top
30991 tcsapp    20   0   11832   1648   1300 S   0.0  0.0   0:00.02 container_+
31006 tcsapp    20   0   13436   1640   1304 S   0.0  0.0   0:00.00 Cloud.zc42+
```

We can also execute the following command that **runs the `top` command for each pod of the cluster**

```
kubectl get pods -n default -o custom-columns=name:metadata.name --no-headers | xargs -I{} sh -c 'echo {};
kubectl exec {} -- top -bn1'
```

2. Time Series, HTTP based monitoring via Prometheus

Prometheus is an open-source technology designed to provide monitoring and alerting functionality for cloud-native environments, including Kubernetes. It can collect and store metrics as time-series data, recording information with a timestamp. It can also collect and record labels, which are optional key-value pairs.



The fundamental data unit is a “metric.” Each metric is assigned a name it can be referenced by as well and a set of labels. Labels are arbitrary key-value data pairs that can be used to filter the metrics in your database.

Metrics are always based on one of four core instrument types:

- **Counter** – A value that steadily increments, never decreasing or resetting.
- **Gauge** – A value that can change in any direction at any time.
- **Histogram** – A sampling of multiple values that provides a sum of all the stored values, as well as the count of recorded events.
- **Summary** – A summary functions similarly to a histogram but supports configurable quantiles for aggregate monitoring over sliding time periods.

Expose Metrics via Exporters

Exporters are responsible for exposing application’s metrics ready for Prometheus to collect. for our CPU, Memory etc metrics we can collect using a simple deployment of the Node Exporter which collects basic system metrics from the Linux host it’s installed on.

Enable Nodeport Service

In our 3.x deployments, Prometheus images are available by default, but **to suppress unintended system metrics, events to go out, by default we have disabled** the ***Nodeport*** service. please use caution when enabling on customer sites, this will raise security concerns.

To Enable it, add Nodeport "***Service Section***" to Prometheus Manifest.

vi /home/centos/manifests/helm/vmlc-prometheus-values.yaml

```
prometheus:
  scrapeInterval: 60s
  retention: 5d
  persistence:
    enabled: true
    storageClass: local-prometheus
    size: 0.5Gi
  service:
    type: NodePort
    nodePort: 30555
```

Uninstall prometheus and Install it to get the changes

```
[centos@zc429clsg01kma001 dm]$ sudo helm uninstall prometheus
release "prometheus" uninstalled
[centos@zc429clsg01kma001 dm]$
```

Check the prometheus installation is completely removed before re-installing

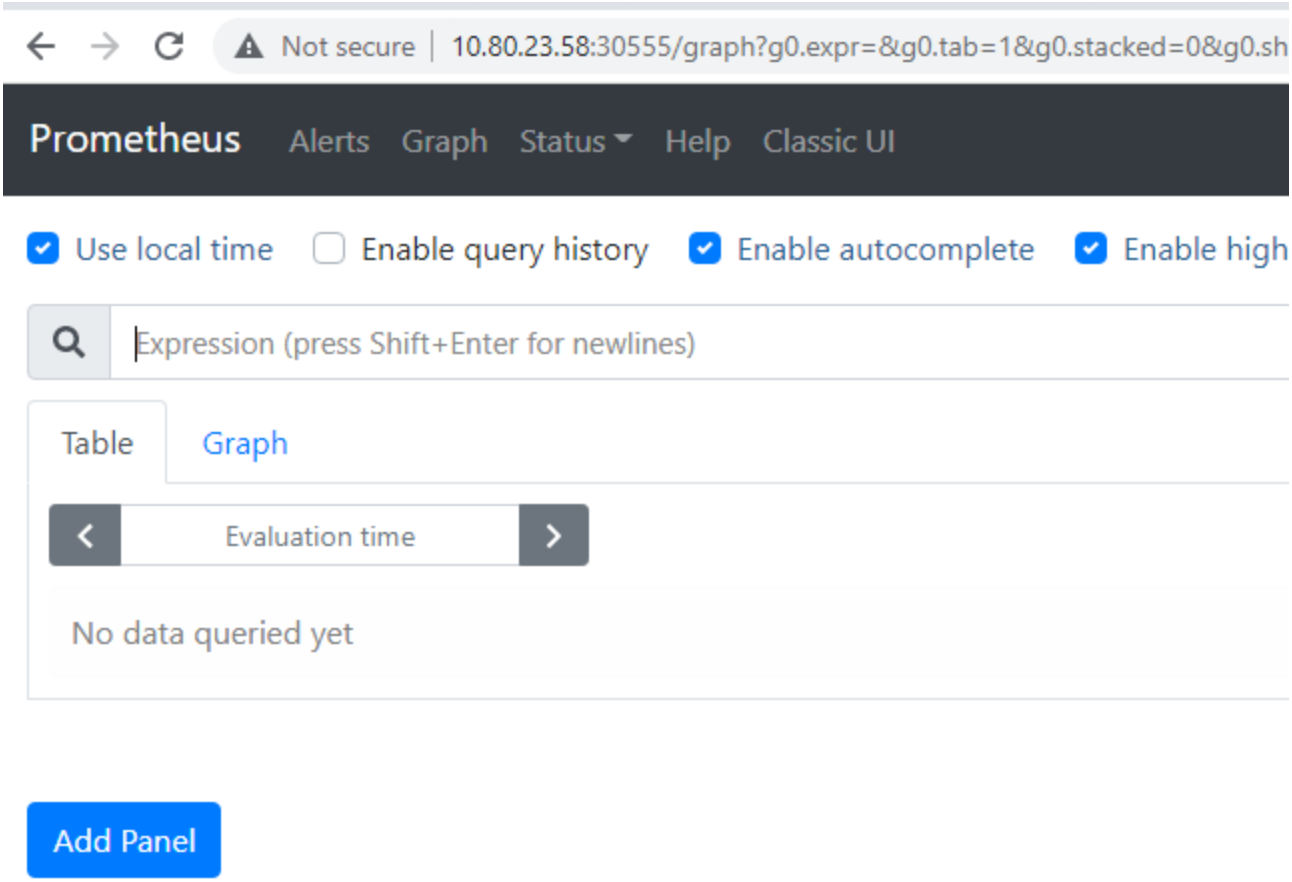
```
[centos@zc429clsg01kma001 dm]$ kubectl get pods | grep -i prometheus
[centos@zc429clsg01kma001 dm]$
```

Re-install Prometheus vi ***helm localrepo***

```
[centos@zc429clsg01kma001 dm]$ sudo helm install prometheus -f /home/centos/manifests/helm/vmlc-prometheus-
values.yaml localrepo/kube-prometheus
NAME: prometheus
LAST DEPLOYED: Thu Aug 25 21:20:51 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: kube-prometheus
CHART VERSION: 6.6.5
APP VERSION: 0.54.0

** Please be patient while the chart is being deployed **

Watch the Prometheus Operator Deployment status using the command:

    kubectl get deploy -w --namespace default -l app.kubernetes.io/name=kube-prometheus-operator,app.
kubernetes    .io/instance=prometheus

Watch the Prometheus StatefulSet status using the command:

    kubectl get sts -w --namespace default -l app.kubernetes.io/name=kube-prometheus-prometheus,app.
kubernetes.    io/instance=prometheus

Prometheus can be accessed via port "9090" on the following DNS name from within your cluster:

    prometheus-kube-prometheus-prometheus.default.svc.cluster.local

To access Prometheus from outside the cluster execute the following commands:

    echo "Prometheus URL: http://127.0.0.1:9090/"
    kubectl port-forward --namespace default svc/prometheus-kube-prometheus-prometheus 9090:9090
```

Access Prometheus Web UI / Dashboard

Prometheus dashboard should be accessible from **http://<k8sMaster-IP:3055>**



Querying for Resource Metrics

< work in progress >