# Install Prometheus

Now, we are going to deploy a single instance of Prometheus. Normally, you would/should deploy multiple instances spread throughout the cluster. For example, one instance dedicated to monitor just Kubernetes API, the next dedicated to monitor nodes, and so on... As with many things in the Kubernetes world, there is no specific way things should look 🙂, so to save resources, we will deploy just one.

We should be back in our `monitoring` folder. Create new folder called `prometheus`, and in it create following files:

## prometheus.yaml

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus-persistant
  namespace: monitoring
spec:
  replicas: 1
  retention: 7d
  resources:
    requests:
      memory: 400Mi
  nodeSelector:
    node-type: worker
  securityContext:
    fsGroup: 2000
    runAsNonRoot: true
    runAsUser: 1000
  serviceAccountName: prometheus
  serviceMonitorSelector:
    matchExpressions:
    - key: name
      operator: In
      values:
      - longhorn-prometheus-servicemonitor
      - kube-state-metrics
      - node-exporter
      - kubelet
      - traefik
  storage:
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        storageClassName: longhorn
        resources:
          requests:
            storage: 20Gi
```

***Interesting parts***

`retention` - How long to keep data.

```
nodeSelector:
  node-type: worker
```

If you followed my setup, I set some tags for worker nodes and control planes. Here, I just say to prefer nodes with the tag: worker. I use this just because I didn't want to tax the control nodes more. If it does not matter to you where the Prometheus is running, remove these two lines.

```
  serviceMonitorSelector:
    matchExpressions:
    - key: name
      operator: In
      values:
      - longhorn-prometheus-servicemonitor
      - kube-state-metrics
      - node-exporter
      - kubelet
      - traefik
```

Here are the Service Monitors we created; this part tells Prometheus to go and take data from them. If you add more or less, edit this part.

`storage:` We will just tell it what provisioner to use, and how many GB to provision; our Prometheus Operator will take care of mounting and assigning the storage for persistent data. Make sure that longhorn is default (I mentioned how to make it the default storage provider in longhorn section).

I will keep data for 7 days (I didn't know how much data it will generate at the time), but logging for a full 7 days, it produced 6.62GB in my case, so 20GB is a safe bet. So, we create a physical volume claim from our Longhorn (man, I love Longhorn, so easy to work with).

## prometheus-service-ext.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus-external
  namespace: monitoring
spec:
  selector:
    prometheus: prometheus-persistant
  type: LoadBalancer
  ports:
    - name: web
      protocol: TCP
      port: 9090
      targetPort: web
  loadBalancerIP: 192.168.0.235
```

If you followed my guides, you know I like to keep most of the services on their own IP, so above I told MetalLB to give Prometheus instance IP 192.168.0.235.

## prometheus-service-local.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoring
spec:
  ports:
  - name: web
    port: 9090
    targetPort: web
  selector:
    prometheus: prometheus-persistant
  sessionAffinity: ClientIP
```

Make the Prometheus also available locally in a cluster under the name `prometheus`, and a port called `web`.

## prometeus-serviceacount.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: monitoring
```

This is just a service account for Prometheus; this provides our pod with an "identity". Later below, we add permissions for this account to look into other namespaces, and collect data etc...

## prometeus-rbac-role-binding.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: monitoring
```

## prometeus-rbac-clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
  namespace: monitoring
rules:
- apiGroups: [""]
  resources:
  - nodes
  - services
  - endpoints
  - pods
  - nodes/metrics
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources:
  - configmaps
  verbs: ["get"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

RBAC gives permissions to access various network resources in clusters, more here: RBAC

Jump out of the `prometheus` folder, and apply it to Cluster.

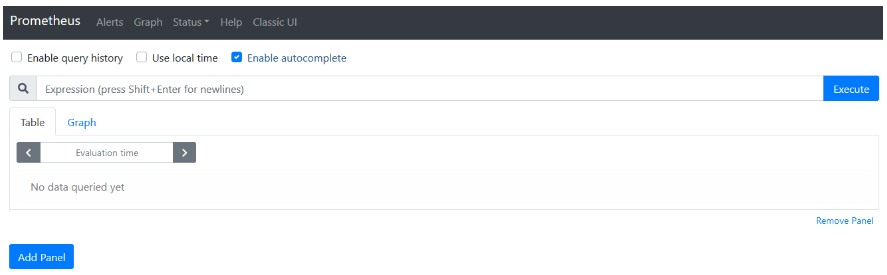
```
cd ..
kubectl apply -f prometheus/
```

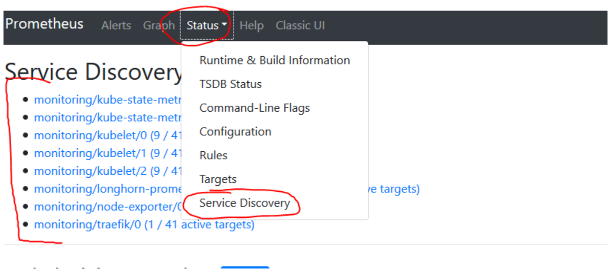
Check if Prometheus deployed. I have called my deployment `prometheus-persistant`, but didn't know that Prometheus Operator adds prometheus- before and -(number) after.

```
root@control01:/home/ubuntu/monitoring# kubectl get pods -n monitoring
NAME                                   READY   STATUS    RESTARTS   AGE
.
.
.
prometheus-prometheus-persistant-0     2/2     Running   1          13d
```

## UI

You should now be able to connect via a browser to the IP of Prometheus, in my case, 192.168.0.235:9090, and see something like this:

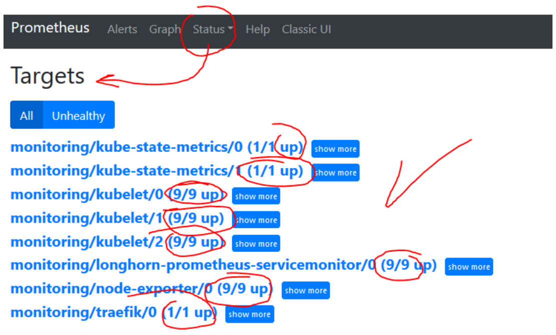




Service Discovery

- monitoring/kube-state-metr
- monitoring/kube-state-metr
- monitoring/kubelet/0 (9 / 41
- monitoring/kubelet/1 (9 / 41
- monitoring/kubelet/2 (9 / 41
- monitoring/longhorn-prome ve targets)
- monitoring/node-exporter/(
- monitoring/traefik/0 (1 / 41 active targets)

Also look into `Status -> Targets`; there all metrics should be `UP`. It can take a few minutes before they are scraped but certainly no longer than 5min. I had configured my permissions wrongly, and got a 403 error for Kubelet. When I fixed it, it got back to UP mode within a minute.



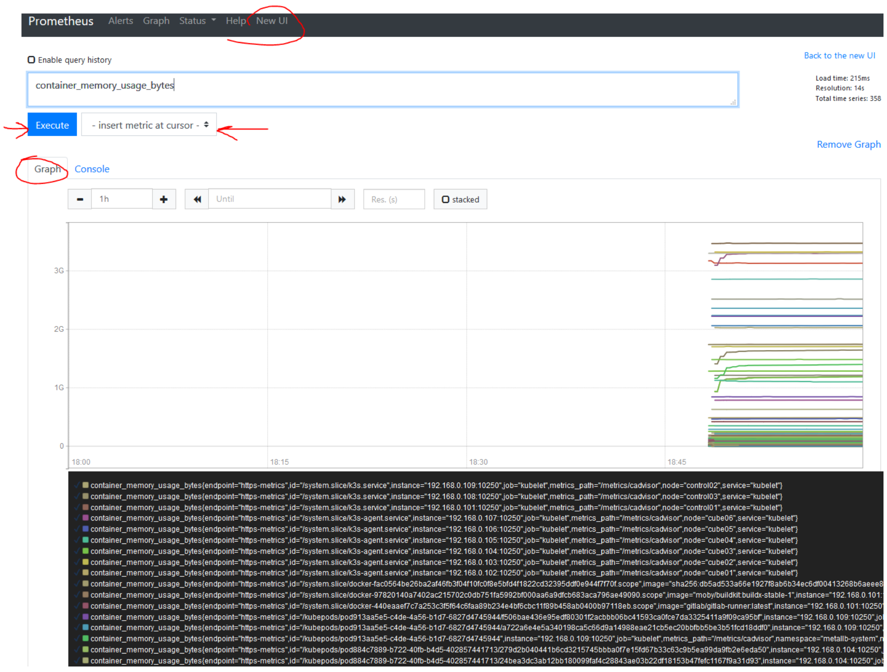
Targets

monitoring/kube-state-metrics/0 (1/1 up)
monitoring/kube-state-metrics/1 (1/1 up)
monitoring/kubelet/0 (9/9 up)
monitoring/kubelet/1 (9/9 up)
monitoring/kubelet/2 (9/9 up)
monitoring/longhorn-prometheus-servicemonitor/0 (9/9 up)
monitoring/node-exporter/0 (9/9 up)
monitoring/traefik/0 (1/1 up)

## Firs graphs

For our graphing purposes we will use Grafana, but you can also get graphs in Prometheus.

It is easiest to switch to `Classic UI`, and under the top input box, right next to `Execute`, is a drop-down menu where you can choose any of the metrics. Next, hit execute, and you should have some data.



container_memory_usage_bytes

Load time: 215ms
Resolution: 14s
Total time series: 358

```
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s.service",instance="192.168.0.109:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="control02",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s.service",instance="192.168.0.108:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="control03",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s.service",instance="192.168.0.101:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="control01",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s-agent.service",instance="192.168.0.107:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="cube06",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s-agent.service",instance="192.168.0.106:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="cube05",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s-agent.service",instance="192.168.0.105:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="cube04",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s-agent.service",instance="192.168.0.104:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="cube03",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s-agent.service",instance="192.168.0.103:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="cube02",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/k3s-agent.service",instance="192.168.0.102:10250",job="kubelet",metrics_path="/metrics/cadvisor",node="cube01",service="kubelet"}
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/docker-fac0564be26ba2af46fb3f04f10fc0f8e5bfd4f1822cd32395ddf0e944f7f70f.scope",image="sha256:db5ad533a66e1927f8ab6b34ec6df00413268b6aeee8
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/docker-97820140a7402ac215702c0db751fa5992bf000aa6a9dfcb683aca796ae49090.scope",image="moby/buildkit:buildx-stable-1",instance="192.168.0.101:
container_memory_usage_bytes{endpoint="https-metrics",id="/system.slice/docker-440eaaef7c7a253c3f5f64c6faa89b234e4bf6cbc11f89b458ab0400b97118eb.scope",image="gitlab/gitlab-runner:latest",instance="192.168.0.101:10250"
container_memory_usage_bytes{endpoint="https-metrics",id="/kubepods/pod913aa5e5-c4de-4a56-b1d7-6827d4745944/f506bae436e95edf80301f2acbbb06bc41593ca0fce7da3325411a9f09ca95bf",instance="192.168.0.109:10250",jo
container_memory_usage_bytes{endpoint="https-metrics",id="/kubepods/pod913aa5e5-c4de-4a56-b1d7-6827d4745944/a722a6e4e5a340198ca5c66d9a14988eae21cb5ec20bbfbb5be3b51fcd18ddf0",instance="192.168.0.109:10250",
container_memory_usage_bytes{endpoint="https-metrics",id="/kubepods/pod913aa5e5-c4de-4a56-b1d7-6827d4745944",instance="192.168.0.109:10250",job="kubelet",metrics_path="/metrics/cadvisor",namespace="metallb-system",n
container_memory_usage_bytes{endpoint="https-metrics",id="/kubepods/pod884c7889-b722-40fb-b4d5-402857441713/279d2b040441b6cd3215745bbba0f7e15fd67b33c63c9b5ea99da9fb2e6eda50",instance="192.168.0.104:10250",
container_memory_usage_bytes{endpoint="https-metrics",id="/kubepods/pod884c7889-b722-40fb-b4d5-402857441713/24bea3dc3ab12bb180099faf4c28843ae03b22df18153b47fefc1167f9a31d93",instance="192.168.0.104:10250",job
```

We are almost there, the last step is Grafana for some kick ass dashboards. Grafana

Did it work for you? Take a break, drink a good beverage of your choice, and if you think I was helpful, maybe get a drink for me as well 🙂.

Last update: May 26, 2022

## Comments

What do you think?

2 Responses

Upvote Funny Love Surprised Angry Sad

4 Comments https://rpi4cluster.com Disqus' Privacy Policy Login

Favorite Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

**Rich Durso** • 3 months ago • edited

Thanks for the helpful guide! In my Service Discovery, the kube-state-metrics shows (0/8 active targets) and Traefik shows (0 / 11 active targets). (the kubelet and node-exporter seems to be working fine). Regarding the Traefik I wonder if things change since you tried, this. This is the useful info about Traefik:

```
Ports:
metrics: 9100
traefik: 9000
Namespace: kube-system

Labels:
app.kubernetes.io/instance=traefik
app.kubernetes.io/managed-by=Helm
app.kubernetes.io/name=traefik
helm.sh/chart=traefik-10.9.100
pod-template-hash=6bb96f9bd8

Annotations:
prometheus.io/path=/metrics
prometheus.io/port=9100
```

see more

• Reply • Share ›

**slaecker** • 5 months ago

When applying prometheus.yaml the CRD object prometheus-persistant gets created but no Prometheus pod. What's missing?

• Reply • Share ›

**Daniel Wood** • 5 months ago

I'm getting "no matches for kind "prometheus" in version "monitoring.coreos.com/v1"

When I try to create the promettheus.yaml - any ideas why? Everything else works, although the prometheus-operator pod it regularly restarting?

• Reply • Share ›