**for observability we have installed:**

#### **Kube-state-metrics:**

#### **✅ Purpose:**

Provides **metadata and state** of Kubernetes objects. It's about *what exists* and *what state it's in*, **not** about actual CPU or memory usage.

#### **✅ Exposes metrics like:**

* Pod status, restart counts
* Deployment replicas (desired vs available)
* StatefulSet/DaemonSet status
* Job completions/failures
* Node conditions
* PersistentVolume and PersistentVolumeClaim info
* HPA status

#### **✅ Used by:**

* **Prometheus** to collect state-level metrics
* **Grafana dashboards** focused on cluster health, pod lifecycles, etc.

**metrics-server**

#### **✅ Purpose:**

Provides **resource usage metrics**: CPU and memory **usage** of **pods and nodes**.

#### **✅ It collects:**

* Live resource usage (NOT historical)
* Data from kubelet /metrics/resource
* It's used by the **HPA (Horizontal Pod Autoscaler)** and kubectl top

important:

**kube-state-metrics** → for **Kubernetes object health**

**metrics-server or cAdvisor** → for **resource usage**

—----------------------------------------------------------------------------------------------------------------------------

**Install cAdvisor:**

**create a cadvisor.yaml**

This will deploy **cAdvisor as a DaemonSet** so it runs on all nodes in your Kubernetes cluster.

Note: for containerd-based Kubernetes nodes

File name: cadvisor.yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: cadvisor

labels:

app: cadvisor

spec:

selector:

matchLabels:

name: cadvisor

template:

metadata:

labels:

name: cadvisor

spec:

containers:

- name: cadvisor

image: gcr.io/cadvisor/cadvisor:v0.47.2

ports:

- containerPort: 8080

hostPort: 8080

name: http

volumeMounts:

- name: rootfs

mountPath: /rootfs

readOnly: true

- name: var-run

mountPath: /var/run

readOnly: true

- name: sys

mountPath: /sys

readOnly: true

- name: docker

mountPath: /var/lib/docker

readOnly: true

- name: disk

mountPath: /dev/disk

readOnly: true

volumes:

- name: rootfs

hostPath:

path: /

- name: var-run

hostPath:

path: /var/run

- name: sys

hostPath:

path: /sys

- name: docker

hostPath:

path: /var/lib/docker

- name: disk

hostPath:

path: /dev/disk

hostNetwork: true

hostPID: true

tolerations:

- key: node-role.kubernetes.io/master

operator: Exists

effect: NoSchedule

Step 2: Apply it to your cluster:

kubectl apply -f cadvisor.yaml

Step 3: Check pods:

kubectl get pods -l name=cadvisor

Port-forward or expose via NodePort to access the cAdvisor UI, or just let Prometheus scrape it on port 8080.

—----------------------------------------------------------------------------------------------------------------------------

**Install metrics-server:**

**Step 1: Download and modify the official manifest**

**Wget https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml -O metrics-server.yaml**

**Step 2: Edit the Deployment section**

**Open metrics-server.yaml in your editor and find the Deployment named metrics-server. Under containers.args, add the following flags:**

**- --kubelet-insecure-tls**

**- --kubelet-preferred-address-types=InternalIP,Hostname,ExternalIP**

**Step 3: Apply the updated manifest**

**kubectl apply -f metrics-server.yaml**

**Step 4: Check if it's running**

**kubectl get pods -n kube-system | grep metrics-server**

**Step 5: Test Metrics API**

**kubectl top nodes**

**kubectl top pods -A**

—----------------------------------------------------------------------------------------------------------------------------

**For CPU/Memory usage:** prometheus  
CopyEdit  
sum(rate(container\_cpu\_usage\_seconds\_total{container!="",pod!="",namespace!="kube-system"}[5m])) by (pod, namespace)

sum(container\_memory\_usage\_bytes{container!="",pod!="",namespace!="kube-system"}) by (pod, namespace)

**For node CPU/Memory:**prometheus  
CopyEdit  
sum(node\_cpu\_seconds\_total{mode="user"}) by (instance)

sum(node\_memory\_MemTotal\_bytes) by (instance)

The reason we deploy **cAdvisor as a DaemonSet** and **metrics-server as a Deployment (Service)** comes down to **how they gather metrics** and **what they're intended to monitor**.

### **🔹 Why cAdvisor is a DaemonSet**

#### **🔍 What it does:**

* cAdvisor collects **container-level metrics** (CPU, memory, I/O, etc.) directly from each node.
* It's meant to run **on every node** to monitor the containers **local to that node**.

#### **✅ Why DaemonSet?**

* A **DaemonSet ensures one Pod per node** — perfect for something that must run everywhere like node-level monitoring tools.
* Without a DaemonSet, you'd have to manually create one pod per node.

#### **📦 Example metrics from cAdvisor:**

* container\_cpu\_usage\_seconds\_total
* container\_memory\_usage\_bytes
* Disk I/O, network, filesystem metrics per container

### **🔹 Why metrics-server is a Deployment + Service**

#### **🔍 What it does:**

* metrics-server collects **summary metrics** from **each kubelet** via the /metrics/resource endpoint.
* It's used by Kubernetes for things like:  
  + kubectl top commands
  + Horizontal Pod Autoscalers (HPA)

#### **✅ Why Deployment?**

* It **doesn’t need to run on every node** — it aggregates from the kubelets.
* It’s a **centralized service**, not node-specific.

#### **🧠 Access pattern:**

* Metrics Server scrapes the kubelets remotely over the network.

### **🔁 Summary**

| **Component** | **Type** | **Why?** |
| --- | --- | --- |
| **cAdvisor** | DaemonSet | Needs to run on every node to gather per-container/node metrics |
| **metrics-server** | Deployment | Centralized, scrapes metrics from kubelets, not tied to specific nodes |

### **🚀 Typical Setup in Monitoring Stack:**

* **🧠 metrics-server = lightweight live metrics for scaling**
* **💡 kube-state-metrics = object state metrics (good for Grafana dashboards)**
* **🔍 cAdvisor = in-depth per-container resource usage (best for detailed monitoring)**

Here's a detailed **comparison table** between **cAdvisor**, **metrics-server**, and **kube-state-metrics**, focusing on their **purpose**, **deployment type**, **metrics exposed**, and **typical usage** in a Kubernetes monitoring stack:

### 

| **📊 Monitoring Tools Comparison: cAdvisor vs metrics-server vs kube-state-metrics** | | |  |
| --- | --- | --- | --- |
| **Feature / Tool** | **cAdvisor 🧱** | **metrics-server 📡** | **kube-state-metrics 📦** |
| Primary Purpose | Collect per-container resource usage | Aggregate node/pod usage for K8s APIs | Expose Kubernetes object states as metrics |
| Common Metrics | container\_cpu\_usage\_seconds\_total, container\_memory\_usage\_bytes | node\_cpu\_usage, pod\_memory\_usage (summary only) | kube\_pod\_status\_phase, kube\_deployment\_status\_replicas\_available |
| Data Source | Docker/container runtime on each node | Kubelets’ /metrics/resource endpoint | Kubernetes API server |
| Scraped by | Prometheus | Kubernetes API consumers (e.g. kubectl top, HPA) | Prometheus |
| Installation Type | DaemonSet (runs on each node) | Deployment (central aggregator) | Deployment (central) |
| Resource Usage | Moderate (runs on every node) | Lightweight | Very lightweight |
| Used by HPA? | ❌ No | ✅ Yes | ❌ No |
| Used by Prometheus? | ✅ Yes | ❌ Not directly | ✅ Yes |
| Key Use Cases | Deep container monitoring, per-node metrics | Autoscaling, kubectl top, live CPU/mem | K8s health dashboards, object tracking |
| Access Mode | NodePort or Pod metrics path (/metrics) | Internal API calls to Kubelet (:10250) | HTTP endpoint (:8080/metrics) |
| Authentication Required? | ❌ No | ✅ Yes (via kubelet TLS) | ❌ No |

### 

### 

### **🔧 Example Use Cases:**

* **cAdvisor**: Build detailed dashboards in Grafana showing CPU, memory, disk I/O per container.
* **metrics-server**: Enable kubectl top pods/nodes and horizontal pod autoscaling.
* **kube-state-metrics**: Monitor desired vs current replicas, pod restarts, job completions, etc.