Observe Metrics Server Data

In this lesson, we will observe the data contained in Metrics Server.

Resource usage of the nodes is useful but is not what we're looking for. In this chapter, we're focused on auto-scaling Pods. But, before we get there, we should observe how much memory each of our Pods is using. We'll start with those running in the `kube-system` Namespace.

Memory consumption of pods running in `kube-system`

Execute the following from your command line to see the memory consumption of all the pods running in the Kube-system.

```
kubectl -n kube-system top pod
```

The **output** (on Docker For Desktop) is as follows.

```
NAME                                       CPU(cores) MEMORY(bytes)
etcd-docker-for-desktop                    16m        74Mi
kube-apiserver-docker-for-desktop          33m        427Mi
kube-controller-manager-docker-for-desktop 44m        63Mi
kube-dns-86f4d74b45-c47nh                  1m         39Mi
kube-proxy-p56kd                           2m         33Mi
```

```
kube-proxy-r56kd                        2m          22Mi
kube-scheduler-docker-for-desktop       13m         23Mi
```

We can see resource usage (CPU and memory) for each of the Pods currently running in `kube-system`. If we do not find better tools, we could use that information to adjust the `requests` of those Pods to be more accurate. However, there are better ways to get that info, so we'll skip adjustments for now. Instead, let's try to get current resource usage of all the Pods, no matter the Namespace.

Get resource usage of pods using `--all-namespaces`

```
kubectl top pods --all-namespaces
```

The **output** (on Docker For Desktop) is as follows.

```
NAMESPACE   NAME                                     CPU(cores) MEMORY(b
ytes)
docker      compose-7447646cf5-wqbwz                 0m         11M
i
docker      compose-api-6fbc44c575-gwhxt             0m         14M
i
kube-system etcd-docker-for-desktop                  16m        74M
i
kube-system kube-apiserver-docker-for-desktop        33m        427M
i
kube-system kube-controller-manager-docker-for-desktop 46m      63M
i
kube-system kube-dns-86f4d74b45-c47nh                1m         38M
i
kube-system kube-proxy-r56kd                         3m         22M
i
kube-system kube-scheduler-docker-for-desktop        14m        23M
i
metrics     metrics-server-5d78586d76-pbqj8          0m         10M
i
```

That **output** shows the same information as the previous one, only extended to all Namespaces. There should be no need to comment on it.

Get resource usage of pods using `--containers`

Often, metrics of a Pod are not granular enough, and we need to observe the resources of each of the containers that constitute a Pod. All we need to do to get container metrics is to add `--containers` argument.

```
kubectl top pods \
  --all-namespaces \
  --containers
```

The **output** (on Docker For Desktop) is as follows.

```
NAMESPACE   POD                                         NAM
E                     CPU(cores) MEMORY(bytes)
docker      compose-7447646cf5-wqbwz                    compos
e                  0m          11Mi
docker      compose-api-6fbc44c575-gwhxt                compos
e                  0m          14Mi
kube-system etcd-docker-for-desktop                     etc
d                     16m         74Mi
kube-system kube-apiserver-docker-for-desktop           kube-apiserver
          33m         427Mi
kube-system kube-controller-manager-docker-for-desktop kube-controller-man
ager 46m          63Mi
kube-system kube-dns-86f4d74b45-c47nh                   kubedn
s                  0m          13Mi
kube-system kube-dns-86f4d74b45-c47nh                   dnsmas
q                  0m          10Mi
kube-system kube-dns-86f4d74b45-c47nh                   sideca
r                  1m          14Mi
kube-system kube-proxy-r56kd                            kube-proxy
              3m         22Mi
kube-system kube-scheduler-docker-for-desktop           kube-scheduler
          14m         23Mi
metrics     metrics-server-5d78586d76-pbqj8             metrics-server
          0m          10Mi
```

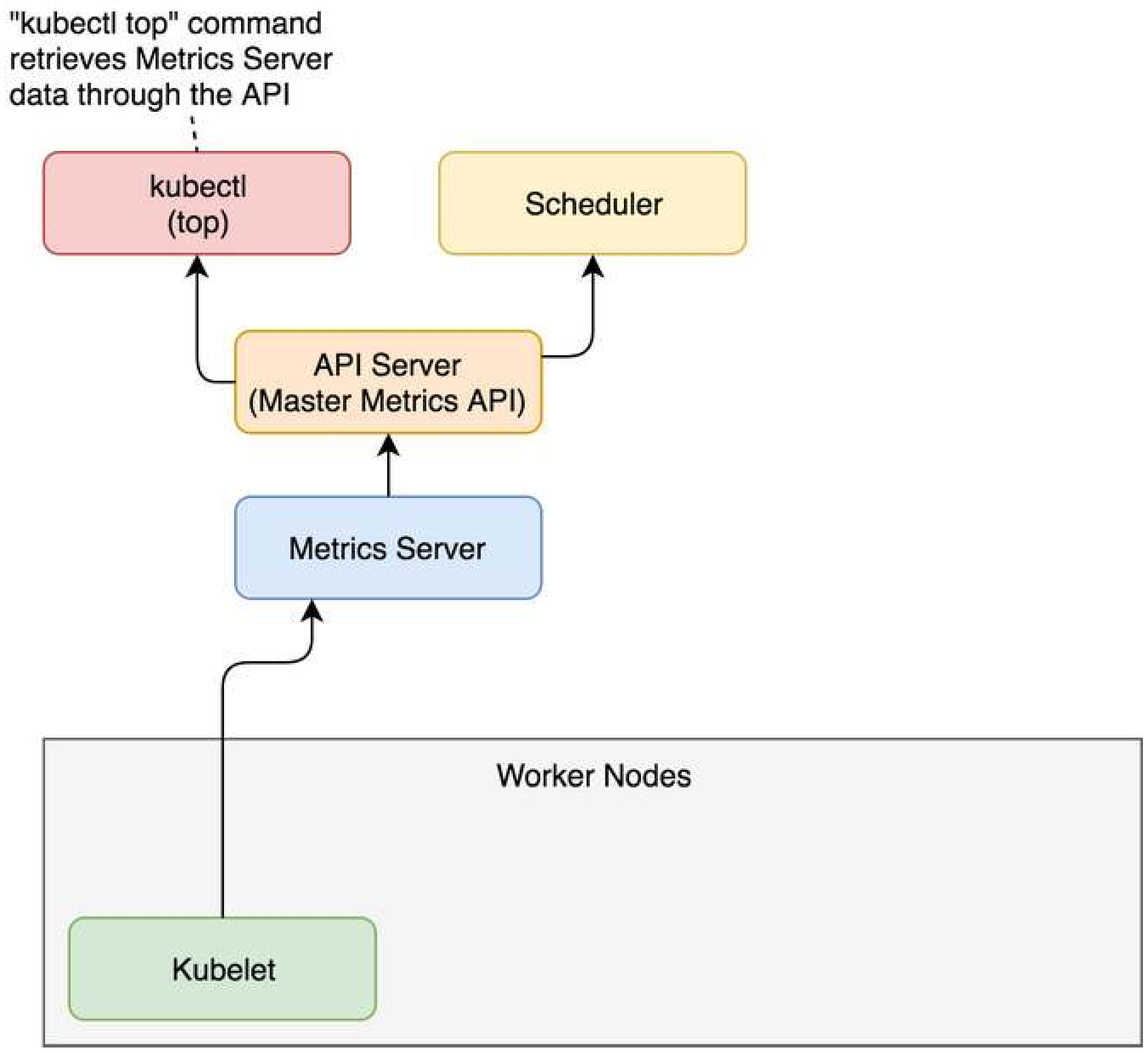
We can see that this time, the output shows each container separately. We can, for example, observe metrics of the `kube-dns-*` Pod separated into three containers (`kubedns`, `dnsmasq`, `sidecar`).

Flow of data using `kubectl top`

When we request metrics through `kubectl top`, the flow of data is almost the same as when the scheduler makes requests. A request is sent to the **API**

Server (Master Metrics API), which gets data from the `Metrics Server`

which, in turn, was collecting information from Kubeletes running on the nodes of the cluster.


"kubectl top" command retrieves Metrics Server data through the API
kubectl (top)
Scheduler
API Server (Master Metrics API)
Metrics Server
Worker Nodes
Kubelet

The flow of the data to and from the Metrics Server (arrows show directions of data flow)

Q

`kubectl` command retrieves data from Metrics Server.

Scrape the metrics using `JSON`

While `kubectl top` command is useful to observe current metrics, it is pretty useless if we'd like to access them from other tools. After all, the goal is not for us to sit in front of a terminal with watch `kubectl top pods` command. That would be a waste of our (human) talent. Instead, our goal should be to scrape those metrics from other tools and create alerts and (maybe) dashboards based on both real-time and historical data. For that, we need output in `JSON` or some other machine-parsable format. Luckily, `kubectl` allows us to invoke its API directly in raw format and retrieve the same result as if a tool would query it.

```
kubectl get \
    --raw "/apis/metrics.k8s.io/v1beta1" \
    | jq '.'
```

The **output** is as follows.

```
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "metrics.k8s.io/v1beta1",
  "resources": [
    {
      "name": "nodes",
      "singularName": "",
      "namespaced": false,
      "kind": "NodeMetrics",
      "verbs": [
        "get",
        "list"
      ]
    },
    {
      "name": "pods",
      "singularName": "",
      "namespaced": true,
      "kind": "PodMetrics",
      "verbs": [
        "get",
        "list"
      ]
    }

  ]
}
```

We can see that the `/apis/metrics.k8s.io/v1beta1` endpoint is an index API that has two resources (`nodes` and `pods`).

Let's take a closer look at the `pods` resource of the metrics API.

```
kubectl get \
    --raw "/apis/metrics.k8s.io/v1beta1/pods" \
    | jq '.'
```

The **output** is too big to display here, so I'll leave it up to you to explore it. You'll notice that the output is `JSON` equivalent of what we observed through the `kubectl top pods --all-namespaces --containers` command.

`Metrics Server` for machines

There are two important things to note. First of all, it provides current (or short-term) memory and CPU utilization of the containers running inside a cluster. The second and more important note is that we will not use it directly. `Metrics Server` was not designed for humans but for machines. We'll get there later. For now, remember that there is a thing called `Metrics Server` and that you should not use it directly (once you adopt a tool that will scrape its metrics).

Now that we explored `Metrics Server`, we'll try to put it to good use and learn how to auto-scale our Pods based on resource utilization, in the next lesson.