Install Metrics Server

In this lesson, we will see what Metrics Server is and how to install it?

WE'LL COVER THE FOLLOWING

From Heapster to `Metrics Server`

The critical element in scaling Pods is the **Kubernetes Metrics Server**. You might consider yourself a Kubernetes ninja and yet never heard of the Metrics Server. Don't be ashamed if that's the case. You're not the only one.

If you started observing Kubernetes metrics, you might have used **Heapster**. It's been around for a long time, and you likely have it running in your cluster, even if you don't know what it is. Both the Metrics server and Heapster serve the same purpose, with one being deprecated for a while, so let's clarify things a bit.

Early on, Kubernetes introduced **Heapster** as a tool that enables Container Cluster Monitoring and Performance Analysis for Kubernetes. It's been around since Kubernetes version 1.0.6. You can say that **Heapster** has been part of Kubernetes' life since its toddler age. It collects and interprets various metrics like resource usage, events, and so on. **Heapster** has been an integral part of Kubernetes and enabled it to schedule Pods appropriately. Without it, Kubernetes would be blind. It would not know which node has available memory, which Pod is using too much CPU, and so on. But, just as with most other tools that become available early, its design was a "failed experiment".

As Kubernetes continued growing, we (the community around Kubernetes)

started realizing that a new, better, and, more importantly, a more extensible design is required. Hence, the `Metrics Server` was born. Right now, even though **Heapster** is still in use, it is considered deprecated, even though today the `Metrics Server` is still in beta state.

What is `Metrics Server`?

A simple explanation is that it collects information about used resources (memory and CPU) of nodes and Pods. It does not store metrics, so do not think that you can use it to retrieve historical values and predict tendencies. There are other tools for that, and we'll explore them later. Instead, `Metrics Server's` goal is to provide an API that can be used to retrieve current resource usage. We can use that API through `kubectl` or by sending direct requests with, let's say, `curl`. In other words, the `Metrics Server` collects cluster-wide metrics and allows us to retrieve them through its API. That, by itself, is very powerful, but it is only part of the story.

I already mentioned extensibility. We can extend the `Metrics Server` to collect metrics from other sources. We'll get there in due time. For now, we'll explore what it provides out of the box and how it interacts with some other Kubernetes resources that will help us make our Pods scalable and more resilient.

Installation of `Metrics Server`

`Helm` makes the installation of almost any publicly available software very easy if there is a *Chart* available. If there isn't, you might want to consider an alternative since that is a clear indication that the vendor or the community behind it does not believe in Kubernetes. Or, maybe they do not have the skills necessary to develop a *Chart*. Either way, the best course of action is to run away from it and adopt an alternative. If that's not an option, develop a `Helm` *Chart* yourself. In our case, there won't be a need for such measures. `Metrics Server` does have a `Helm` *Chart*, and all we need to do is to install it.

> A note to GKE and AKS users

Google and Microsoft already ship Metrics Server as part of their managed Kubernetes clusters (GKE and AKS). There is no need to install it, so please skip the commands that follow.

A note to minikube users

Metrics Server is available as one of the plugins. Please execute `minikube addons enable metrics-server` and `kubectl -n kube-system rollout status deployment metrics-server` commands instead of those below.

A note to Docker For Desktop users

Recent updates to the Metrics Server do not work with self-signed certificates by default. Since Docker For Desktop uses such certificates, you'll need to allow insecure TLS. Please add `--set args={"--kubelet-insecure-tls=true"}` argument to the `helm install` command that follows.

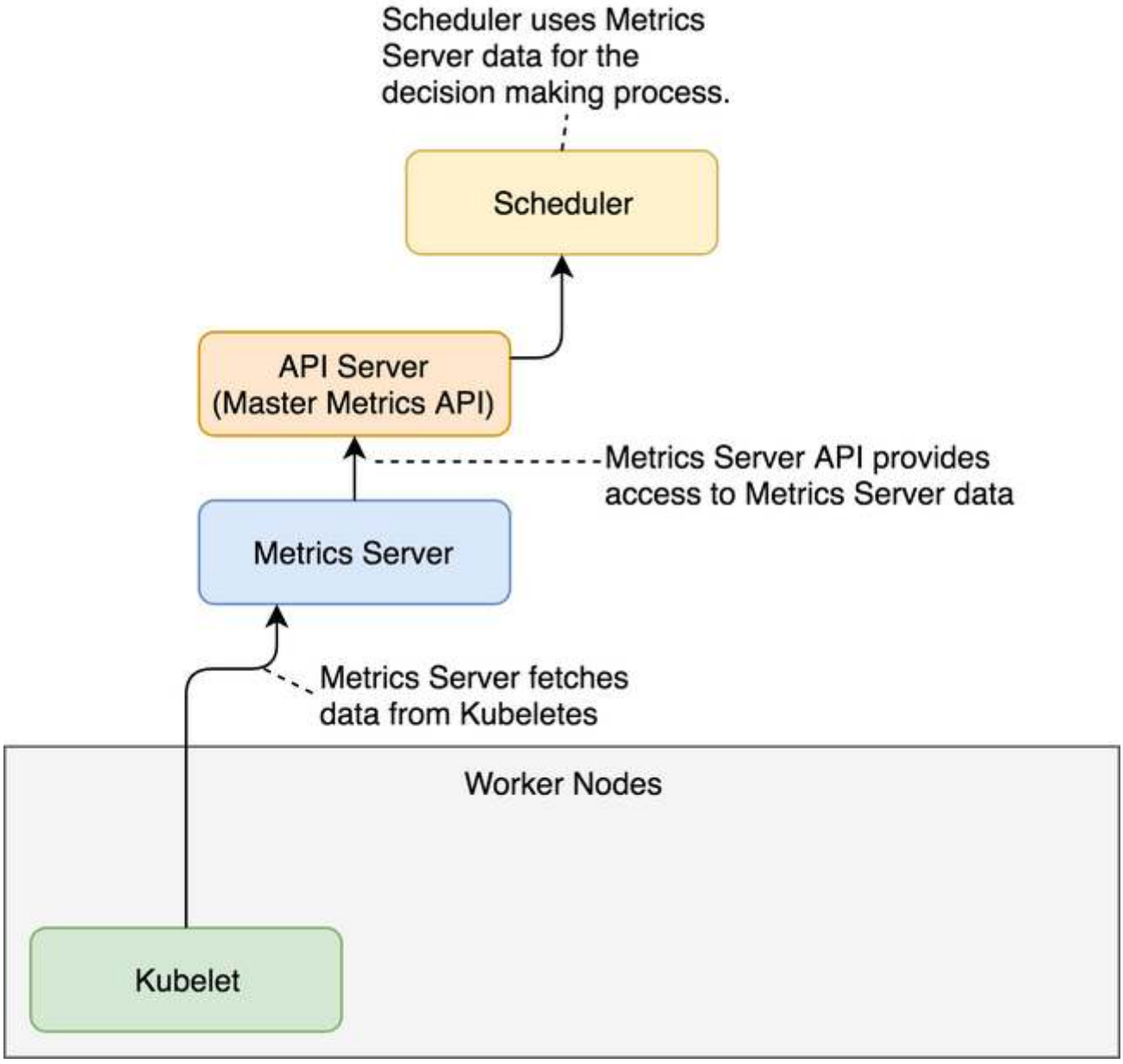
```
helm repo update

kubectl create namespace metrics

helm install metrics-server \
    stable/metrics-server \
    --version 2.0.2 \
    --namespace metrics

kubectl -n metrics \
    rollout status \
    deployment metrics-server
```

We used `Helm` to install `Metrics Server`, and we waited until it rolled out.

Flow of data in `Metrics Server`

`Metrics Server` will periodically fetch metrics from Kubeletes running on the nodes. Those metrics, for now, contain memory and CPU utilization of the Pods and the nodes. Other entities can request data from the `Metrics Server`

through the API Server which has the **Master Metrics API**. An example of those entities is the Scheduler that, once `Metrics Server` is installed, uses its data to make decisions. As you will see soon, the usage of the `Metrics Server` goes beyond the Scheduler but, for now, the explanation should provide an image of the basic flow of data.



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

Q

`Metrics Server` will periodically fetch metrics from Kubeletes running on _____?

Retrieve the metrics of nodes

Now we can explore one of the ways we can retrieve the metrics. We'll start with those related to nodes.

```
kubectl top nodes
```

If you were fast, the output should state that `metrics` are `not available yet`. That's normal. It takes a few minutes before the first iteration of metrics retrieval is executed. The exception is GKE and AKS that already come with the `Metrics Server` baked in.

Fetch some coffee before we repeat the command.

```
kubectl top nodes
```

This time, the **output** is different.

> In this chapter, I'll show the outputs from Docker For Desktop. Depending on the Kubernetes flavor you're using, your outputs will be different. Still, the logic is the same and you should not have a problem following along.

My **output** is as follows.

```
NAME                 CPU(cores)  CPU%  MEMORY(bytes)  MEMORY%
docker-for-desktop   248m        12%   1208Mi         63%
```

We can see that I have one node called `docker-for-desktop`. It is using 248 CPU

milliseconds. Since the node has two cores, that's 12% of the total available

CPU. Similarly, 1.2GB of RAM is used, which is 63% of the total available memory of 2GB.

In the next lesson, we will observe how much memory each of our Pods is using.