Creating Semaphore Dashboards

In this lesson, we will see how to create Semaphore dashboards.

Value of dashboards

If I'm claiming that the value dashboards bring to the table is lower than we think, you might be asking yourself the same question from the beginning of this chapter. Why are we talking about dashboards? Well, I already changed my statement from "dashboards are useless" to "there is some value in dashboards." They can serve as a registry for queries. Through dashboards, we do not need to memorize expressions that we would need to write in `Prometheus`. They might be a good starting point of our search for the cause of an issue before we jump into `Prometheus` for some deeper digging into metrics. But, there is another reason I am including dashboards into the solution.

I love big displays. It's very satisfying to enter into a room with large screens showing stuff that seem to be important. There is usually a room where operators sit surrounded with monitors on all four walls. That's usually an impressive sight. However, there is a problem with many such situations. A bunch of monitors displaying a lot of graphs might not amount to much more than a pretty sight. After the initial few days, nobody will stare at graphs. If that's not true, you can just as well fire that person knowing that he was faking his work.

Let me repeat it one more time.

> 🔍 Dashboards are not designed for us to stare at them, especially not when they are on big screens where everyone can see them.

`Semaphores` for big screens

So, if it's a good idea to have big screens, but graphs are not a good candidate to decorate them, what should we do instead? The answer lies in `semaphores`. They are similar to alerts, and they should provide a clear indication of the status of the system. If everything on the screen is green, there is no reason for us to do anything. One of them turning red is a cue that we should do something to correct the problem. Therefore, it is imperative that we try to avoid false positives. If something turns red, and that does not require any action, we are likely to start ignoring it in the future. When that happens, we are risking the situation in which we ignore a real issue, thinking that it is just another false positive. Hence, every appearance of an alarm should be followed by an action. That can be either a fix that will correct the system or a change in the conditions that turned one of the `semaphores` red. In either case, we should not ignore it.

The main problem with `semaphores` is that they are not as appealing to CTOs and other decision-makers. They are not colorful, nor do they show a lot of boxes, lines, and numbers. People often confuse usefulness with how pleasing something is to look at. Nevertheless, we are not building something that should be sold to CTOs, but something that can be helpful in our day-to-day work.

> 🔍 `Semaphores` are much more useful than graphs as a way to see the status of the system, even though they do not look as colorful and eye-pleasing as graphs.

Creating our first `Semaphore`

Please click the *Add panel* icon from the top-right part of the screen, followed

Please click the *Add panel* icon from the top-right part of the screen, followed by *Choose Visualization". Select *Singlestat*. Click the arrow icon next to the

Panel Title, and select *Edit*.

For the most part, creating a single stat (a `semaphore`) is not much different from creating a graph. The significant difference is in the metric (query) that should produce a single value. We'll get there soon. For now, we'll change some general info of the panel.

Please select the *General* tab.

Type "*Pods with < $cpuReqPercentMin% || > $cpuReqPercentMax% actual compared to reserved CPU*" as the *Title* and the text that follows as the *Description*.

```
The number of Pods with less than $cpuReqPercentMin% or more than $cpuReqP
ercentMax% actual compared to reserved CPU
```

This single stat will use a similar query as the graph we made earlier. However, while the graph is displaying current usage compared to reserved CPU, this panel is supposed to show how many Pods have actual CPU usage outside of the boundaries based on reserved CPU. That is reflected in the title and the description we just entered. As you can see, this time we're relying on more variables to formulate our intentions.

Write the query

Now, let's turn our attention to the query. Please click the *Queries* tab and type the expression that follows into the field next to *A*.

> 🔍 For your convenience, the query is available in the grafana-single-stat-actual-vs-reserved-cpu Gist.

```
sum(
    (
        sum(
            label_join(
                rate(container_cpu_usage_seconds_total{
                    namespace!="kube-system",
                    pod_name!=""}[5m]),
                    pod_name!=""}[5m]),
                    "pod",
                    ",",
                    "pod_name"
            )
        ) by (pod) /
        sum(
            kube_pod_container_resource_requests_cpu_cores{
                namespace!="kube-system",
                namespace!="ingress-nginx"
            }
        ) by (pod) and
        sum(
            kube_pod_container_resource_requests_cpu_cores{
                namespace!="kube-system",
                namespace!="ingress-nginx"
            }
        ) by (pod) > $minCpu
    ) < bool ($cpuReqPercentMin / 100)
) +
sum(
    (
        sum(
            label_join(
                rate(
                    container_cpu_usage_seconds_total{
                        namespace!="kube-system",
                        pod_name!=""
                    }[5m]
                ),
                "pod",
                ",",
                "pod_name"
            )
        ) by (pod) /
        sum(
            kube_pod_container_resource_requests_cpu_cores{
                namespace!="kube-system",
                namespace!="ingress-nginx"
            }
        ) by (pod) and
        sum(
            kube_pod_container_resource_requests_cpu_cores{
                namespace!="kube-system",
                namespace!="ingress-nginx"
            }
        ) by (pod) > $minCpu
    ) > bool ($cpuReqPercentMax / 100)
)
```

That query is similar to one of the ones we used for the `Prometheus alert`. To be more precise, it is a combination of the two `Prometheus alerts`. The first half returns the number of Pods with more than `$minCpu` (5 CPU milliseconds) of reserved CPU and with actual CPU usage lower than `$cpuReqPercentMin` (50%). The second half is almost the same as the first, except that it returns Pods with CPU usage higher than `$cpuReqPercentMax` (150%).

Using `sum` to return the number of pods

Since our goal is to return a single stat which, in this case, the number of Pods, you might be surprised that we used `sum` instead of `count`. Counting Pods would indeed make more sense, except that would return `N/A` if there are no results. To avoid that, we're using a trick with `bool`. By putting it in front of expression, it returns `1` if there is a match, and `0` if there isn't. That way, if none of the Pods match the conditions, we won't get an empty result, but `0`, which is a better representation of the number of problematic Pods.

All in all, we are retrieving a sum of all the Pods with the actual CPU below `$cpuReqPercentMin` (50%) of the reserved CPU, plus the sum of all the Pods with the actual CPU above `$cpuReqPercentMax` (150%) of the reserved CPU. In both cases, only the Pods with more than `$minCpu` (five CPU milliseconds) are included. The query itself is not the simplest one we could write but, considering that we already spent a lot of time with `Prometheus` queries, I thought that I should not "insult" you with something trivial.

Defining conditions to trigger the change of colors

Next, please click the *Visualization* icon. That is where we'll define the conditions that should trigger the change of colors.

We do NOT want average value over the specified period, but the current number of problematic Pods. We'll accomplish that by changing the value of the *Show* drop-down list to *Current*.

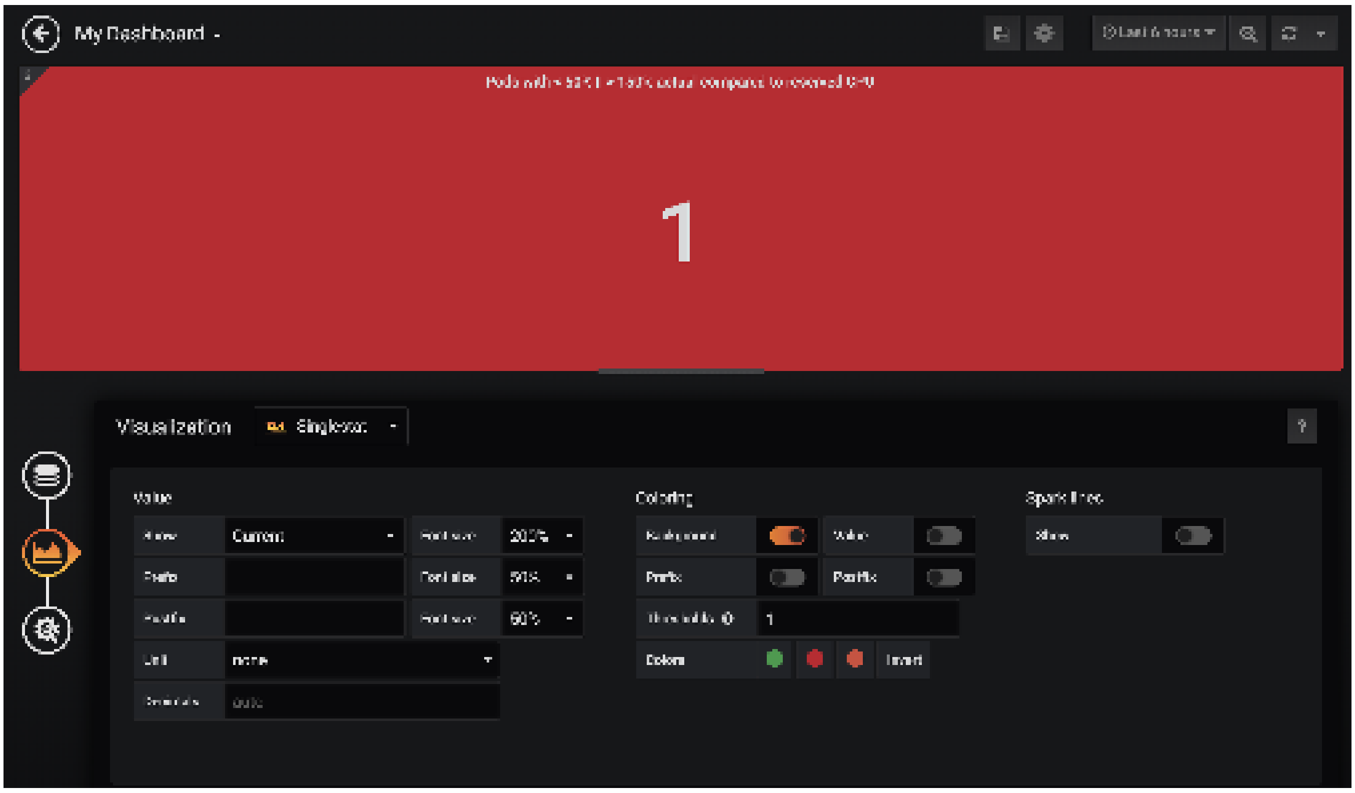
We want this panel to be very visible, so we'll change the *Show Font size* to *200%*. I'd prefer even bigger font, but Grafana does not allow us to go higher than that.

Next, we want to change the background color of the panel, so please check

Next, we want to change the background color of the panel, so please check the *Coloring Background* checkbox.

We could use up to three colors, but I believe we need only two. Either one or more of the Pods meet the conditions, or none of them do. We should be notified as soon as the query returns `1`, or a higher number. Please type *1* as the *Coloring Thresholds*. If we had more, we'd separate them with commas.

Finally, since we only have two conditions, green and red, we'll need to change the second color from orange to red. Please click the orange icon in *Coloring Colors*, and select a red color. The third color is not used, so we'll leave it intact.



Grafana's single stat panel

We're finished with our panel, so go *Back to dashboard*.

Before we proceed, please click the *Save Dashboard* icon.

So far, we've created a dashboard with a graph and a single stat (semaphore). The former shows the deviation of CPU usage compared to reserved CPU over time. It has alerts (red areas) that tell us whether one of the vectors is outside predefined boundaries. The single stat (semaphore) shows a single number with a green or red background depending on whether that number reached a threshold which, in our case, is set to *1*.

We just started, and we need many additional panels before this dashboard

We just started, and we need many additional panels before this dashboard becomes useful. I'll save you from repetitive instructions for defining the

others. I feel that you got a grip on how `Grafana` works. You should at least have the base knowledge that you can expand on your own.

We'll fast forward. We'll import a dashboard I prepared and discuss the design choices. Let's see that in the next lesson.