# Creating Custom Dashboards

In this lesson, we will create our very own custom dashboard.

WE'LL COVER THE FOLLOWING

## Creating a custom dashboard

It would be great if all our needs could be covered by existing dashboards. But, that is probably not the case. Each organization is "special" and our needs have to be reflected in our dashboards. Sometimes we can get away with dashboards made by others, and sometimes we need to change them. In other cases, we need to create our own dashboards. That's what we'll explore next.

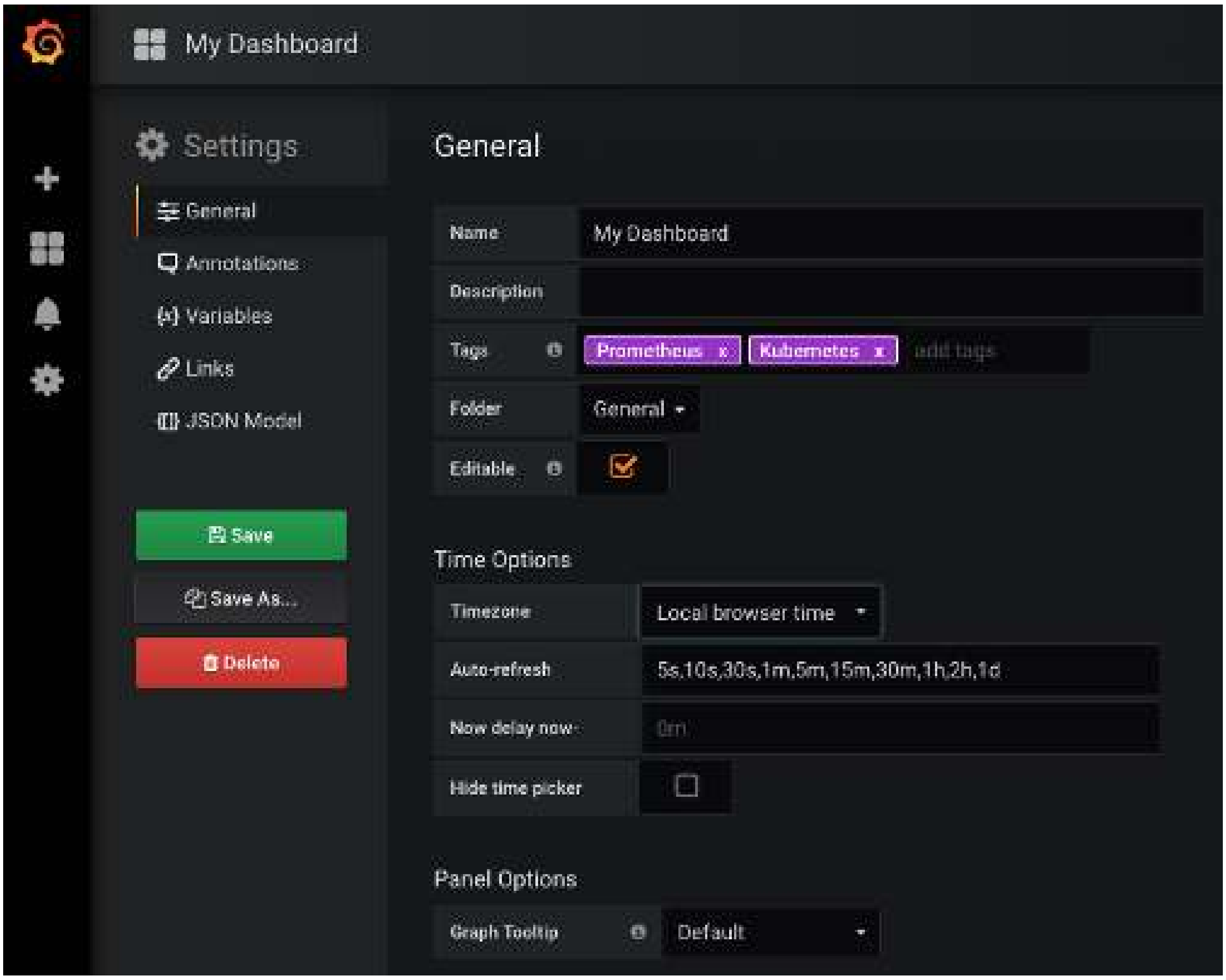
Please click the + icon in the left-hand menu and choose to *Create Dashboard*. You'll be presented with the choice of a few types of panels. Select *Choose Visualization*, you'll be presented with the choice of a few types of panels. Select *Graph*.

### Dashboard settings

Before we define our first visualization, we'll change a few dashboard settings. Please click the *Settings* icon in the top-right part of the screen.

Inside the *General* section, type the *Name* of the dashboard. If you are not inspired today, you can call it *My Dashboard*. Set the *Tags* to `Prometheus` and

*Kubernetes*. You'll have to press the enter key after typing each tag. Finally, change the *Timezone* to *Local browser time*.



General

| | |
|---|---|
| Name | My Dashboard |
| Description | |
| Tags | Prometheus x Kubernetes x add tags |
| Folder | General |
| Editable | ✓ |

Time Options

| | |
|---|---|
| Timezone | Local browser time |
| Auto-refresh | 5s,10s,30s,1m,5m,15m,30m,1h,2h,1d |
| Now delay now- | 0m |
| Hide time picker | ☐ |

Panel Options

| | |
|---|---|
| Graph Tooltip | Default |

Grafana's dashboard general settings screen

## Convert an alert into graph

That was the boring part. Now let's switch to something more interesting. We are about to convert one of the alerts we created in `Prometheus` into a graph. We'll use the one that tells us the percentage of actual vs. reserved CPU. For that, we'll need a few variables. To be more precise, we don't really need them since we could hard-code the values. But, that would cause problems later on if we decide to change them. It is much easier to modify variables than to change the queries.

Specifically, we'll need variables that will tell us the minimum CPU so that we can ignore the thresholds for the applications that are set to use very low reservations. Also, we'll define variables that will act as lower and upper boundaries. Our goal is to be notified if reserved CPU is too low or too high when compared with the actual usage, just as we did with `Prometheus` alerts.

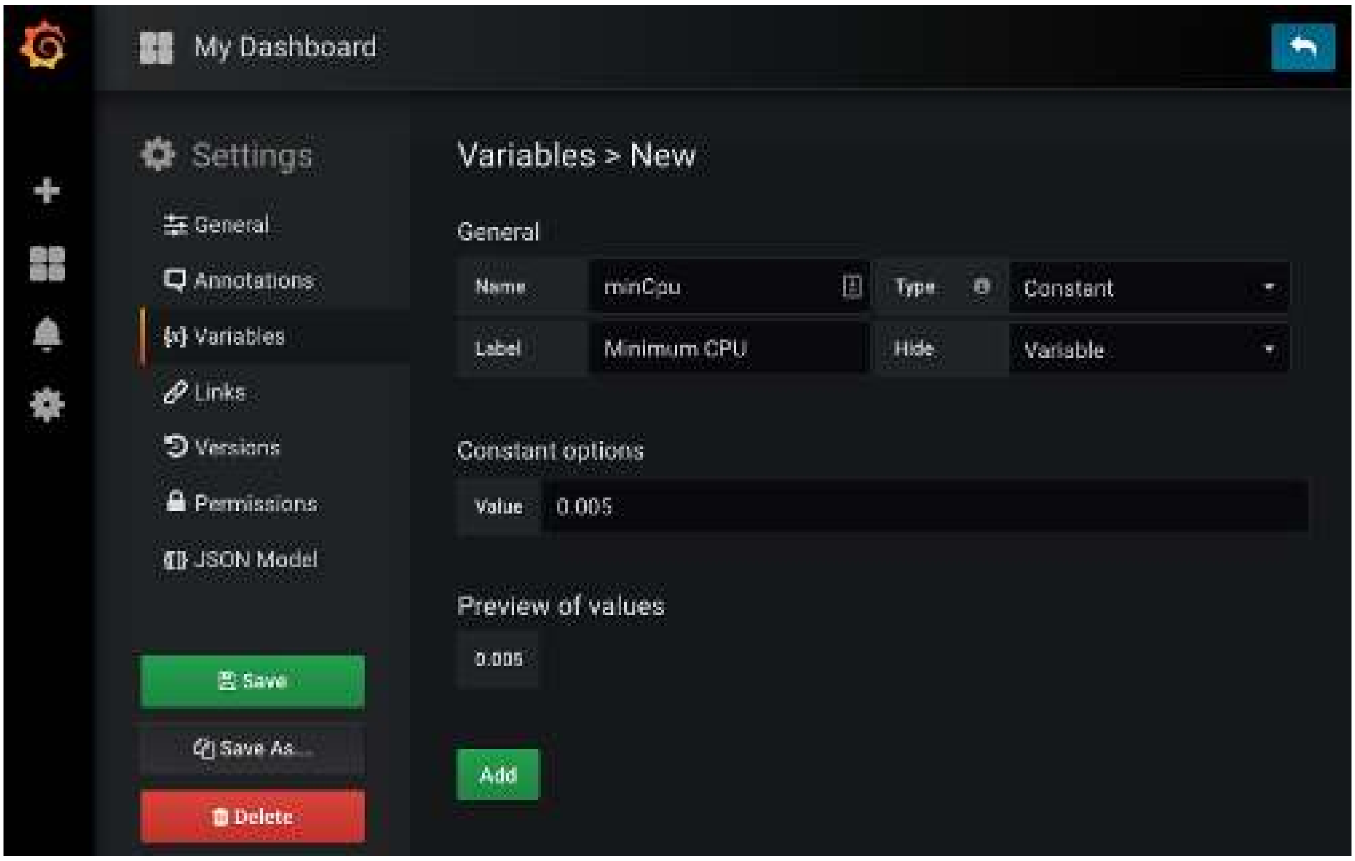
Please select the *Variables* section from the left-hand menu, and click the *Add Variable* button.

You already saw the screen with **Grafana variables** when we created a new one for the dashboard we imported. This time, however, we'll use slightly different settings.

## Specifying settings #

Type *minCpu* as the *Name* and choose *Constant* as the *Type*. Unlike the *device* variable we created earlier, this time we do not need `Grafana` to query the values. By using that type, we are going to define a constant value. Please set the *Value* to *0.005* (five CPU milliseconds). Finally, we do not need to see that variable in the dashboard, since the value is not likely to change often. If we do need to change it in the future, we can always come back to this screen and update it. Therefore, change the *Hide* value to *Variable*.

All that's left is to click the *Add* button.



General

| | | | |
|---|---|---|---|
| Name | minCpu | Type | Constant |
| Label | Minimum CPU | Hide | Variable |

Constant options

Value 0.005

Preview of values

0.005

Grafana's dashboard new variable screen

Q

Variables tell us what is the maximum CPU so that we can ignore the thresholds for the applications that are set to use very low reservations.

COMPLETED 0%

1 of 1

We need two more variables. There's probably no need to repeat the same instructions, so please use the following information to create them.

```
Name:  cpuReqPercentMin
Type:  Constant
Label: Min % of requested CPU
Hide:  Variable
Value: 50

Name:  cpuReqPercentMax
Type:  Constant
Label: Max % of requested CPU
Hide:  Variable
Value: 150
```

Now we can go back and define our graph. Please click the *Back to dashboard* icon from the top-left part of the screen.

We'll start with the *General* section. Please select it. It's one of the icons on the bottom-left side of the screen.

Next, write "*% of actual vs reserved CPU*" as the *Title* and the text that follows as the *Description*.

```
The percentage of actual CPU usage compared to reserved. The calculation e
xcludes Pods with reserved CPU equal to or smaller than $minCpu. Those wit
h less than $minCpu of requested CPU are ignored.
```

Please note the usage of *$minCpu* variable in the description. When we go back to the dashboard, it will expand to its value.

## Switching to Metrics tab

Next, please click the *Queries* icon. That's where the real action is happening.

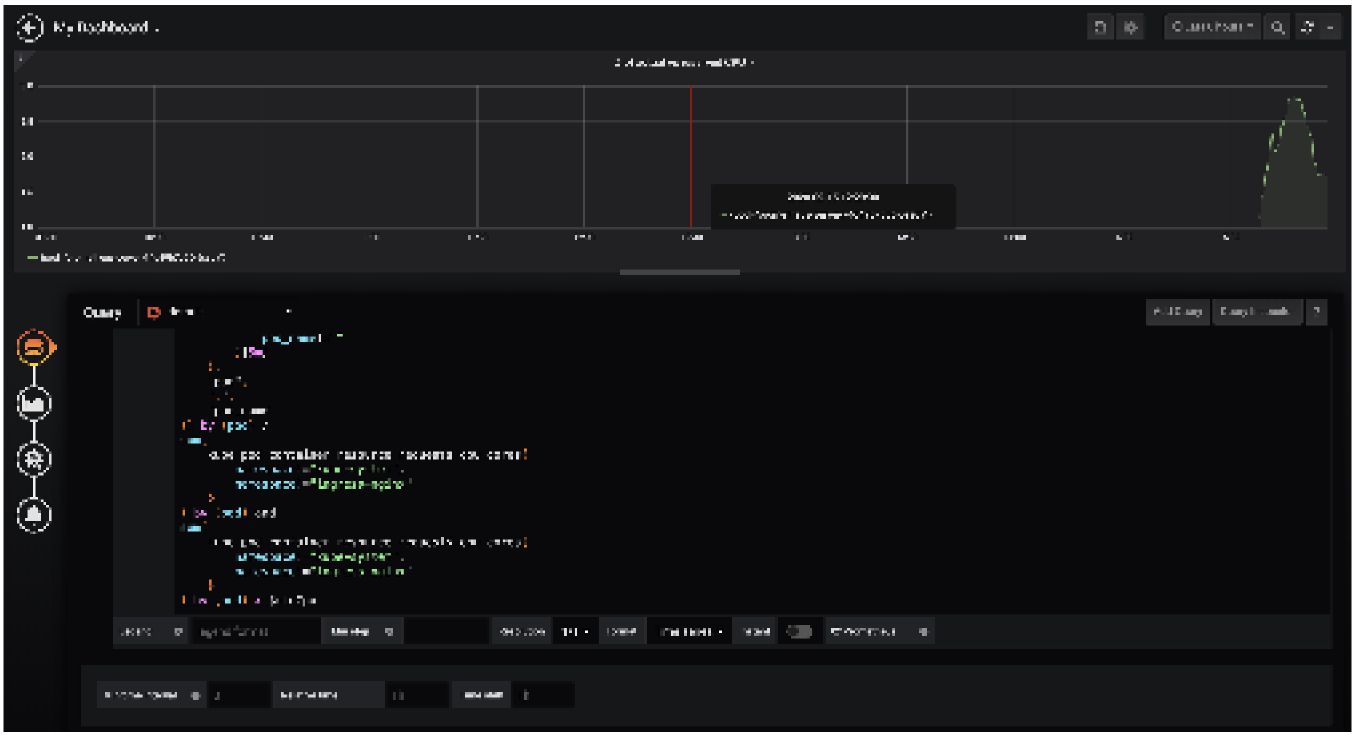
We can define multiple queries but, for our use case, one should be enough. Please type the query that follows in the field to the right of *A*.

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

```
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) > 0.005
```

That query is almost the same as one of those we used in the Collecting And Querying Metrics And Sending Alerts chapter.

A few moments after entering the query, we should see the graph come alive. There is probably only one Pod included since many of our applications are defined to use five CPU milliseconds or less.



Grafana's panel based on a graph

Next, we'll adjust the units on the left side of the graph. Please click the *Visualization* tab and scroll to the *Axes* section.

Inside the *Left Y Unit*, select *Misc*, followed by *percent (0.0-1.0)*. Since we're not using the *Right Y* axis, please uncheck the *Show* checkbox.

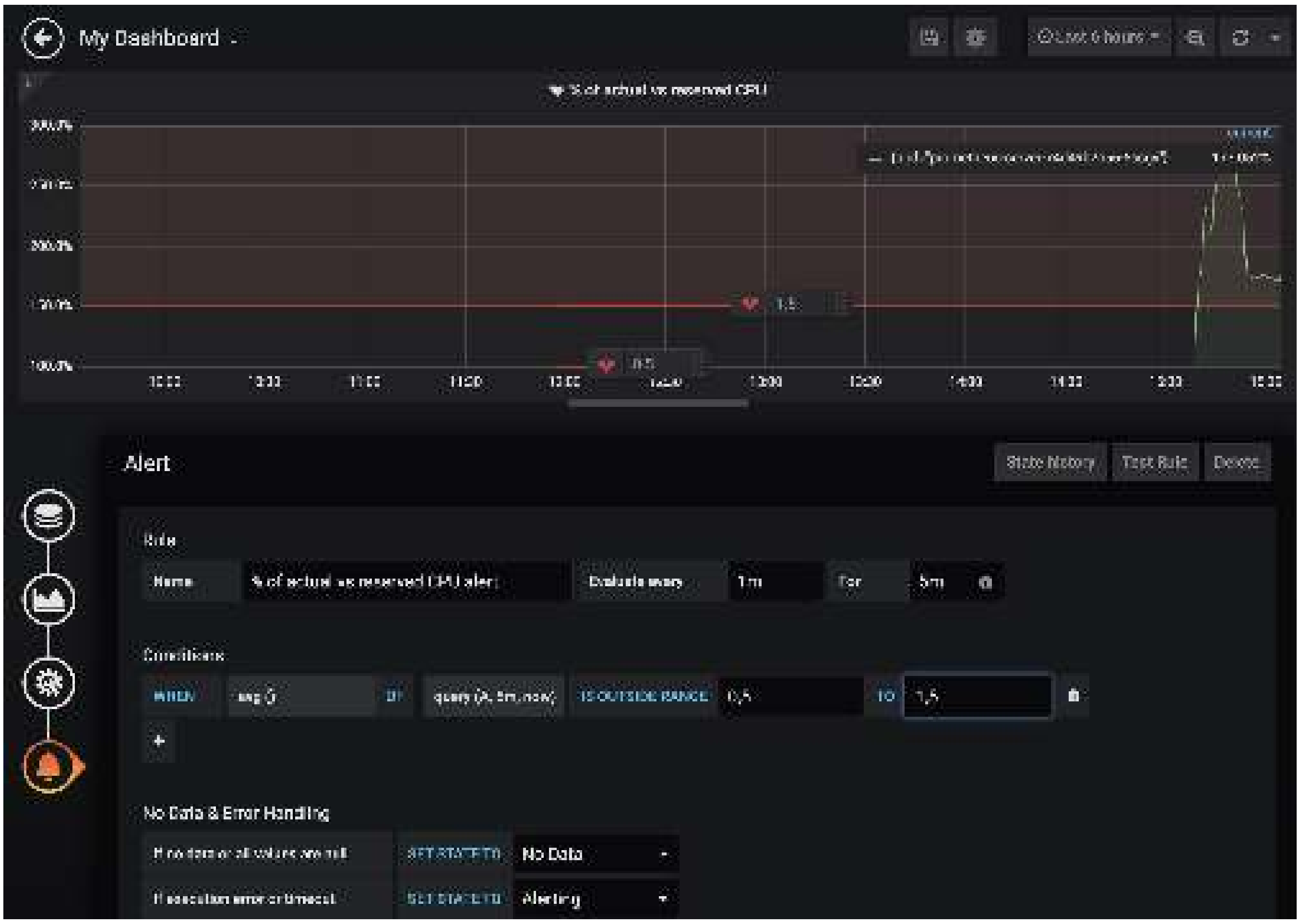
The next section in line is *Legend*.

Check the *Options As Table*, *Options To the right*, and *Values > Current* checkboxes. The changes are applied to the graph immediately, and you should not have trouble deducing what each of those does.

There's only one more thing missing. We should define upper and lower thresholds that will provide a clear indication that the results are outside expected boundaries.

Please click the *Alert* icon.

Click the *Create Alert* button and change the *IS ABOVE* condition to *IS OUTSIDE RANGE*. Set the values of the next two fields to *0.5* and *1.5*. That should notify us if the actual CPU usage is below 50% or above 150% when compared to the reserved value.



Rule

Name: % of actual vs reserved CPU alert
Evaluate every: 1m
For: 5m

Conditions

WHEN avg () OF query (A, 5m, now) IS OUTSIDE RANGE 0,5 TO 1,5

No Data & Error Handling

If no data or all values are null SET STATE TO No Data
If execution error or timeout SET STATE TO Alerting

Grafana's graph with alerts

We're done with the graph, so please go *Back to dashboard* and enjoy "pretty colors". You might want to drag the bottom-right corner of the graph to adjust its size.

We can see the difference between the requested and the actual CPU usage. We also have the thresholds (marked in red) that will tell us whether the usage goes outside the established boundaries.

## Graph's usefulness based on its usage #

Now comes the big question. Is such a graph useful? The answer depends on what we're going to use it for.

If the goal is to stare at the graph waiting for one of the Pods to start using too much or too little CPU, I can only say that you're wasting your talent that can be used on more productive tasks. After all, we already have a similar alert in `Prometheus` that will send us a Slack notification when the criteria are met. It is more advanced than what we have in that graph because it will notify us only if the CPU usage spikes for a given period, thus avoiding temporary

issues that might be resolved a few seconds or a few minutes later. We should discard those cases as false alarms.

Another usage of the graph could be more passive. We could ignore it (close `Grafana`) and come back to it only if the above mentioned `Prometheus` alert is fired. That might make more sense. Even though we could run a similar query in `Prometheus` and get the same results, having a predefined graph could save us from writing such a query. You can think of it as a way to have a query registry with corresponding graphical representations. That is something that does make more sense. Instead of staring at the dashboard (choose Netflix instead), we can come back to it in time of need. While in some situations that might be a reasonable strategy, it will work only in very simple cases. When there is an issue, and a single pre-defined graph solves the problem or, to be more precise, provides a clear indication of the cause of the issue, graphs do provide significant value. However, more often than not, finding the cause of a problem is not that simple and we'll have to turn to `Prometheus` to start digging deeper into metrics.

> 🔍 Looking at dashboards with graphs is a waste of time. Visiting dashboards after receiving a notification about an issue makes a bit more sense. Still, all but trivial problems require deeper digging through Prometheus metrics.

Nevertheless, the graph we just made might prove itself useful, so we'll keep it. What we might want to do, in such a case, is to change the link of the **Prometheus alert** (the one we're currently receiving in Slack) so that it takes us directly to the Graph (not the dashboard). We can get that link by clicking the arrow next to the panel name, and choosing the *View* option.

I believe that we can make our dashboard more useful if we change the type of panels from graphs to something less colorful, with fewer lines, fewer axes, and without other pretty things.

---

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