

Seeing Green Alerts in Action

In this lesson, we will see green alert in action.

WE'LL COVER THE FOLLOWING

- Retrieve node information using `kube_node_info`
 - `Prometheus Gauge` metric
- Writing query using `Prometheus` 's functions
- Define an alert to notify us in case of failures
 - Group of rules
 - Green alerts

Next, we'll take a look at the alerts screen.

```
open "http://$PROM_ADDR/alerts"
```

The screen is empty. Do not despair. We'll get back to that screen quite a few times. The alerts will be increasing as we progress. For now, just remember that's where you can find your alerts.

Finally, we'll open the graph screen.

```
open "http://$PROM_ADDR/graph"
```

That is where you'll spend your time debugging issues you'll discover through alerts.


Retrieve node information using `kube_node_info`

As our first task, we'll try to retrieve information about our nodes. We'll use `kube_node_info` so let's take a look at its description (help) and its type.

```
kubectl -n metrics run -it test \
  --image=appropriate/curl \
  --restart=Never \
  --rm \
  -- prometheus-kube-state-metrics:8080/metrics \
  | grep "kube_node_info"
```

The **output**, limited to the **HELP** and **TYPE** entries, is as follows.

```
# HELP kube_node_info Information about a cluster node.
# TYPE kube_node_info gauge
...
```

 You are likely to see variations between your results and mine. That's normal since our clusters probably have different amounts of resources, my bandwidth might be different, and so on. In some cases, my alerts will fire, and yours won't, or the other way around. I'll do my best to explain my experience and provide screenshots that accompany them. You'll have to compare that with what you see on your screen.

Now, let's try using that metric in **Prometheus**.

Please type the following query in the expression field.

```
kube_node_info
```

Click the *Execute* button to retrieve the values of the **kube_node_info** metric.

 Unlike previous chapters, the Gist from this one [03-monitor.sh](#) contains not only the commands but also **Prometheus expressions**. They are all commented (with **#**). If you're planning to copy & paste the expressions from the Gist, please exclude the comments. Each expression has a **# Prometheus expression** comment on top to help you identify it. As an example, the one you just executed is written in the Gist as follows.

```
#Prometheus expression #kube_node_info
```

If you check the **HELP** entry of the **kube_node_info**, you'll see that it provides **information about a cluster node** and that it is a **gauge**. "A **gauge** is a metric that represents a single numerical value that can arbitrarily go up and down". That makes sense for information about nodes since their number can increase or decrease over time.

Prometheus Gauge metric

📌 A **Prometheus gauge** is a metric that represents a single numerical value that can arbitrarily go up and down.

If we focus on the output, you'll notice that there are as many entries as there are worker nodes in the cluster. The value (**1**) is useless in this context. Labels, on the other hand, can provide some useful information. For example, in my case, the operating system (**os_image**) is **Ubuntu 16.04.5 LTS**. Through that example, we can see that we can use the metrics not only to calculate values (e.g., available memory) but also to get a glimpse into the specifics of our system.

The screenshot shows the Prometheus web interface. At the top, there are navigation links: Prometheus, Alerts, Graph, Status, and Help. Below these, there is a search bar with the text "kube_node_info". To the right of the search bar, it says "Load time: 151ms", "Resolution: 14s", and "Total time series: 3". Below the search bar, there is a button labeled "Execute" and a dropdown menu with the text "Insert metric at cursor". Below this, there are two tabs: "Graph" and "Console". The "Console" tab is selected, and it shows the HELP output for the kube_node_info metric. The output is a table with two columns: "Element" and "Value". The "Value" column contains the number "1". The "Element" column contains the following text: "kube_node_info(app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",container_runtime_version="docker/1.13.1",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kernel_version="4.15.0-1023-azure",kubernetes_version="v1.11.3",kubeproxy_version="v1.11.3",kubernetes_namespace="prometheus-kube-state-metrics",kubernetes_namespace="metrics",node="aks-nodepool1-29770171-0",os_image="Ubuntu 16.04.5 LTS",provider_id="azure://subscriptions/7169b08b-7d00-43e9-8e0c-f10bb78e9d11/resourceGroups/MC_devops25-group_devops25-cluster_azure/providers/Microsoft.Compute/virtualMachines/aks-nodepool1-29770171-0",release="prometheus")".

Element	Value
kube_node_info(app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",container_runtime_version="docker/1.13.1",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kernel_version="4.15.0-1023-azure",kubernetes_version="v1.11.3",kubeproxy_version="v1.11.3",kubernetes_namespace="prometheus-kube-state-metrics",kubernetes_namespace="metrics",node="aks-nodepool1-29770171-0",os_image="Ubuntu 16.04.5 LTS",provider_id="azure://subscriptions/7169b08b-7d00-43e9-8e0c-f10bb78e9d11/resourceGroups/MC_devops25-group_devops25-cluster_azure/providers/Microsoft.Compute/virtualMachines/aks-nodepool1-29770171-0",release="prometheus")	1
kube_node_info(app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",container_runtime_version="docker/1.13.1",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kernel_version="4.15.0-1023-azure",kubernetes_version="v1.11.3",kubeproxy_version="v1.11.3",kubernetes_namespace="prometheus-kube-state-metrics",kubernetes_namespace="metrics",node="aks-nodepool1-29770171-1",os_image="Ubuntu 16.04.5 LTS",provider_id="azure://subscriptions/7169b08b-7d00-43e9-8e0c-f10bb78e9d11/resourceGroups/MC_devops25-group_devops25-cluster_azure/providers/Microsoft.Compute/virtualMachines/aks-nodepool1-29770171-1",release="prometheus")	1
kube_node_info(app="prometheus",chart="prometheus-7.1.3",component="kube-state-metrics",container_runtime_version="docker/1.13.1",heritage="Tiller",instance="10.244.2.3:8080",job="kubernetes-service-endpoints",kernel_version="4.15.0-1023-azure",kubernetes_version="v1.11.3",kubeproxy_version="v1.11.3",kubernetes_namespace="prometheus-kube-state-metrics",kubernetes_namespace="metrics",node="aks-nodepool1-29770171-2",os_image="Ubuntu 16.04.5 LTS",provider_id="azure://subscriptions/7169b08b-7d00-43e9-8e0c-f10bb78e9d11/resourceGroups/MC_devops25-group_devops25-cluster_azure/providers/Microsoft.Compute/virtualMachines/aks-nodepool1-29770171-2",release="prometheus")	1

Remove Graph

Writing query using Prometheus's functions

Let's see if we can get a more meaningful query by combining that metric with one of the Prometheus's functions. We'll `count` the number of worker nodes in our cluster. The `count` is one of Prometheus's [aggregation operators](#).

Please execute the expression that follows.

```
count(kube_node_info)
```

The **output** should show the total number of worker nodes in your cluster. In my case (AKS) there are `3`. At first glance, that might not be very helpful. You might think that you should know without Prometheus how many nodes you have in your cluster. But that might not be true. One of the nodes might have failed, and it did not recuperate. That is especially true if you're running your cluster on-prem without scaling groups. Or maybe [Cluster Autoscaler](#) increased or decreased the number of nodes. Everything changes over time, either due to failures through human actions or through a system that adapts itself. No matter the reasons for volatility, we might want to be notified when something reaches a threshold. We'll use nodes as the first example.

Our mission is to define an alert that will notify us if there are more than three or less than one node in the cluster. We'll imagine that those are our limits and that we want to know if the lower or the upper thresholds are reached due to failures or [Cluster Autoscaling](#).

Define an alert to notify us in case of failures

We'll take a look at a new definition of the Prometheus's Chart's values. Since the definition is big and it will grow with time, from now on, we'll only look at the differences.

```
diff mon/prom-values-bare.yml \  
mon/prom-values-nodes.yml
```

The **output** is as follows.

```
> serverFiles:
>   alerts:
>     groups:
>       - name: nodes
>         rules:
>           - alert: TooManyNodes
>             expr: count(kube_node_info) > 3
>             for: 15m
>             labels:
>               severity: notify
>             annotations:
>               summary: Cluster increased
>               description: The number of the nodes in the cluster increased
>           - alert: TooFewNodes
>             expr: count(kube_node_info) < 1
>             for: 15m
>             labels:
>               severity: notify
>             annotations:
>               summary: Cluster decreased
>             description: The number of the nodes in the cluster decreased
```

We added a new entry `serverFiles.alerts`. If you check `Prometheus` Helm documentation, you'll see that it allows us to define alerts (hence the name). Inside it, we're using the "standard" `Prometheus` syntax for defining alerts.

Please consult [Alerting Rules documentation](#) for more info about the syntax.

Group of rules

We defined only one group of rules called `nodes`. Inside it are two `rules`. The first one (`TooManyNodes`) will notify us if there are more than `3` nodes `for` more than `15` minutes. The other (`TooFewNodes`) will do the opposite. It'll tell us if there are no nodes (`<1`) for `15` minutes. Both `rules` have `labels` and `annotations` that, for now, serve only informational purposes. Later on, we'll see their real usage.

Let's upgrade our `Prometheus`'s Chart and see the effect of the new alerts.

```
helm upgrade prometheus \
  stable/prometheus \
  --namespace metrics \
  --version 9.5.2 \
  --set server.ingress.hosts=${PROM_ADDR} \
  --set alertmanager.ingress.hosts=${AM_ADDR} \
  -f mon/prom-values-nodes.yml
```

It'll take a few moments until the new configuration is “discovered” and **Prometheus** is reloaded. After a while, we can open the **Prometheus**'s alerts screen and check whether we got our first entries.

🔍 From now on, I won't comment (much) on the need to wait for a while until the next config is propagated. If what you see on the screen does not coincide with what you're expecting, please wait for a while and refresh it.

```
open "http://$PROM_ADDR/alerts"
```

You should see two alerts.

Green alerts

Both alerts are green since none evaluate to **true**. Depending on the Kubernetes flavor you choose, you either have only one node (e.g., Docker For Desktop and minikube) or you have three nodes (e.g., GKE, EKS, AKS). Since our alerts are checking whether we have less than one, or more than three nodes, neither of the conditions are met, no matter which Kubernetes flavor you're using.

🔍 If your cluster was not created through one of the **Gists** provided at the beginning of this chapter, then you might have more than three nodes in your cluster, and the alert will fire. If that's the case, I suggest you modify the *mon/prom-values-nodes.yml* file to adjust the threshold of the alert.

Alerts

☐ Show annotations

TooFewNodes (0 active)

```
alert: TooFewNodes
expr: count(kube_node_info)
      < 1
for: 15m
labels:
  severity: notify
annotations:
  description: The number of the nodes in the cluster decreased
  summary: Cluster decreased
```

TooManyNodes (0 active)

Prometheus' alerts screen



`serverFiles.alerts` allows us to define alerts.

COMPLETED 0%

1 of 1



In the next lesson, we will create a red alert and define rules to forward alerts.