Upgrading Old Pods

In this lesson, we will see how to detect the Old Pods running in our System and how to generate an alert for it.

WE'LL COVER THE FOLLOWING

Upgrading our system

Our primary goal should be to prevent issues from happening by being proactive. In cases when we cannot predict that a problem is about to materialize, we must, at least, be quick with our reactive actions that mitigate the issues after they occur. Still, there is a third category that can only loosely be characterized as being proactive. We should keep our system clean and up-to-date.

Among many things we could do to keep the system up-to-date is making sure that our software is relatively recent (patched, updated, and so on). A reasonable rule could be to try to renew software after ninety days, if not earlier. That does not mean that everything we run in our cluster should be newer than ninety days, but that might be a good starting point. Further on, we might create finer policies that would allow some kinds of applications (usually third-party) to live up to, let's say, half a year without being upgraded. Others, especially software we're actively developing, will probably be upgraded much more frequently. Nevertheless, our starting point is to detect all the applications that were not upgraded in ninety days or more.

Detect old applications

Just as in almost all other exercises in this chapter, we'll start by opening the `Prometheus` graph screen and explore the metrics that might help us reach our goal.

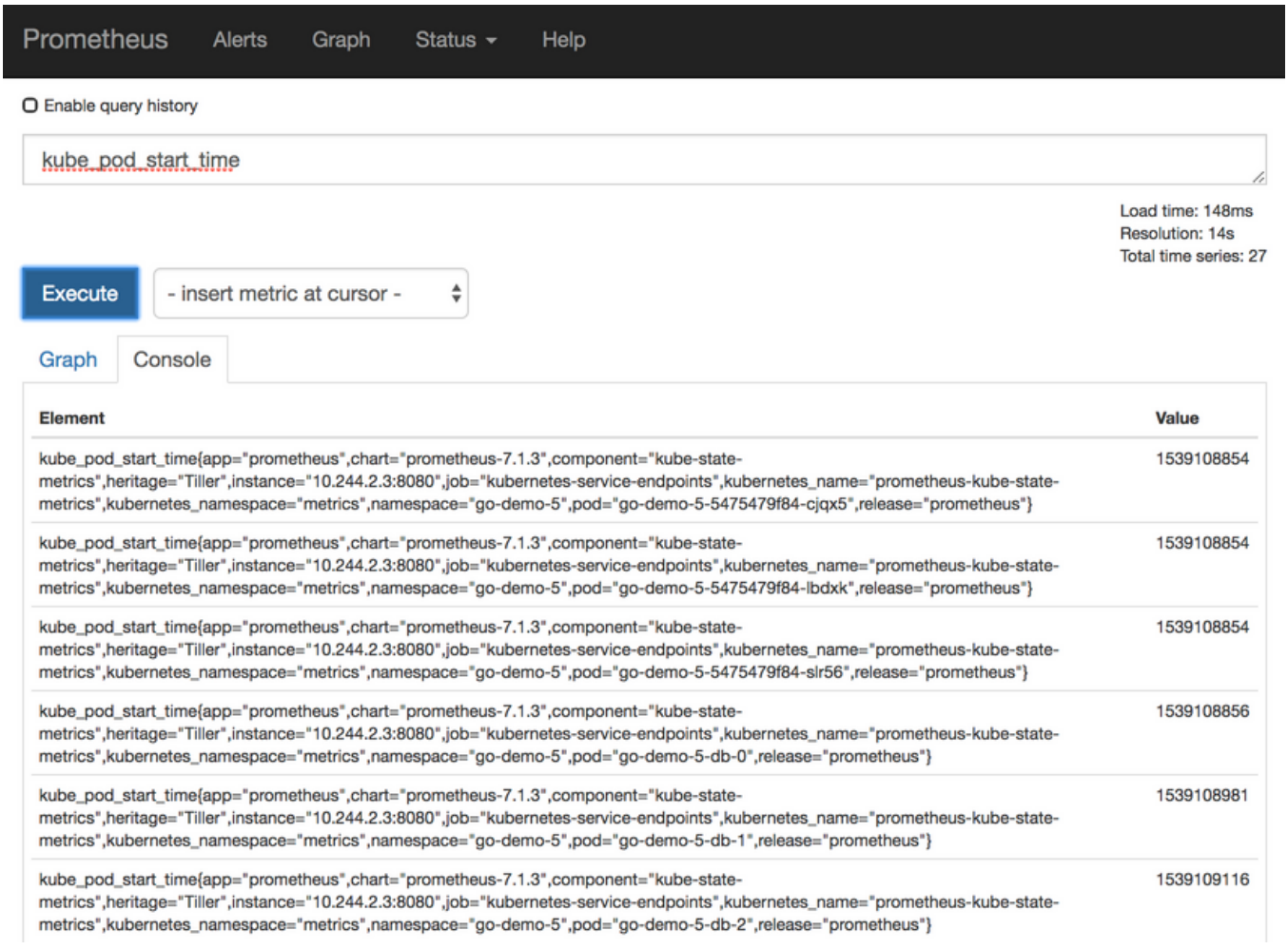
```
open "http://$PROM_ADDR/graph"
```

If we inspect the available metrics, we'll see that there is `kube_pod_start_time`. Its name provides a clear indication of its purpose. It provides the Unix timestamp that represents the start time of each Pod in the form of a Gauge. Let's see it in action.

Please type the expression that follows and click the *Execute* button.

```
kube_pod_start_time
```

Those values alone are of no use, and there's no point in teaching you how to calculate the human date from those values. What matters, is the difference between now and those timestamps.



Prometheus' console view with the start time of the Pods

Prometheus' console view with the start time of the Pods

We can use the `Prometheus`'s time() function to return the number of seconds since January 1, 1970, UTC (or Unix timestamp).

Please type the expression that follows and click the *Execute* button.

```
time()
```

Just as with the `kube_pod_start_time`, we got a long number that represents seconds since 1970. The only noticeable difference, besides the value, is that there is only one entry, while with `kube_pod_start_time` we got a result for each Pod in the cluster.

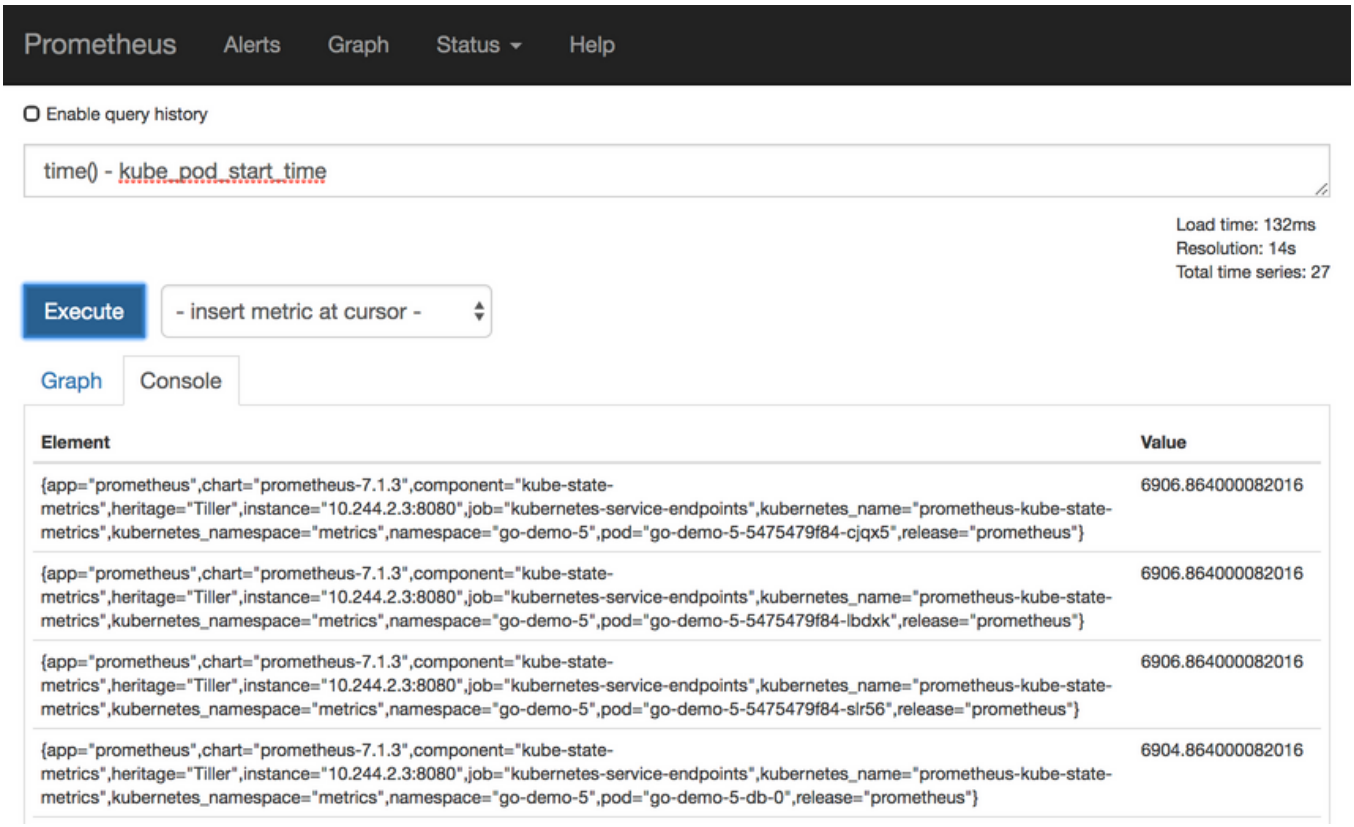
Retrieve the age of each of the pods

Now, let's combine the two metrics in an attempt to retrieve the age of each of the Pods.

Please type the expression that follows and click the *Execute* button.

```
time() -
kube_pod_start_time
```

The results are much smaller numbers, representing the seconds between now and the creation of each of the Pods. In my case (screenshot below), the first Pod (one of the `go-demo-5` replicas), is slightly over six thousand seconds old. That would be around a hundred minutes (6096 / 60), or less than two hours (100 min / 60 min = 1.666 h).



Prometheus' console view with the time passed since the creation of the Pods

Q

With `Prometheus' time()` there is only one entry, while with `kube_pod_start_time` we got a result for each Pod in the cluster.

COMPLETED 0%

1 of 1

Lower the target time

Since there are probably no Pods older than our target of ninety days, we'll lower it temporarily to a minute (sixty seconds).

Please type the expression that follows and click the *Execute* button.

```
(
```

```
  time() -
  kube_pod_start_time{
    namespace!="kube-system"
  }
) > 60
```

Increase the threshold

In my case, all the Pods are older than a minute (as yours are probably as well). We confirmed that it works so we can increase the threshold to ninety days. To get to ninety days, we should multiply the threshold by sixty to get minutes, by another sixty to get hours, by twenty-four to get days, and, finally, by ninety. The formula would be `60 * 60 * 24 * 90`. We could use the final value of `7776000`, but that would make the query harder to decipher. I prefer using the formula instead.

Please type the expression that follows and click the *Execute* button.

```
(
  time() -
  kube_pod_start_time{
    namespace!="kube-system"
  }
) >
(60 * 60 * 24 * 90)
```

It should come as no surprise that there are (probably) no results. If you created a new cluster for this chapter, you'd need to be the slowest reader on earth if it took you ninety days to get here. This might be the longest chapter I've written so far, but it's still not worth ninety days of reading.

Adding an alert

Now that we know which expression to use, we can add one more alert to our setup.

```
diff mon/prom-values-phase.yml \
    mon/prom-values-old-pods.yml
```

The **output** is as follows.

```
146a147,154
```

```
> - alert: OldPods
>   expr: (time() - kube_pod_start_time{namespace!="kube-system"}) > 60
>   labels:
>     severity: notify
>     frequency: low
>   annotations:
>     summary: Old Pods
>     description: At least one Pod has not been updated to more than 90 d
ays
```

We can see that the difference between the old and the new values are in the `OldPods` alert. It contains the same expression we used a few moments ago. We kept the low threshold of `60` seconds so that we can see the alert in action. Later on, we'll increase that value to ninety days.

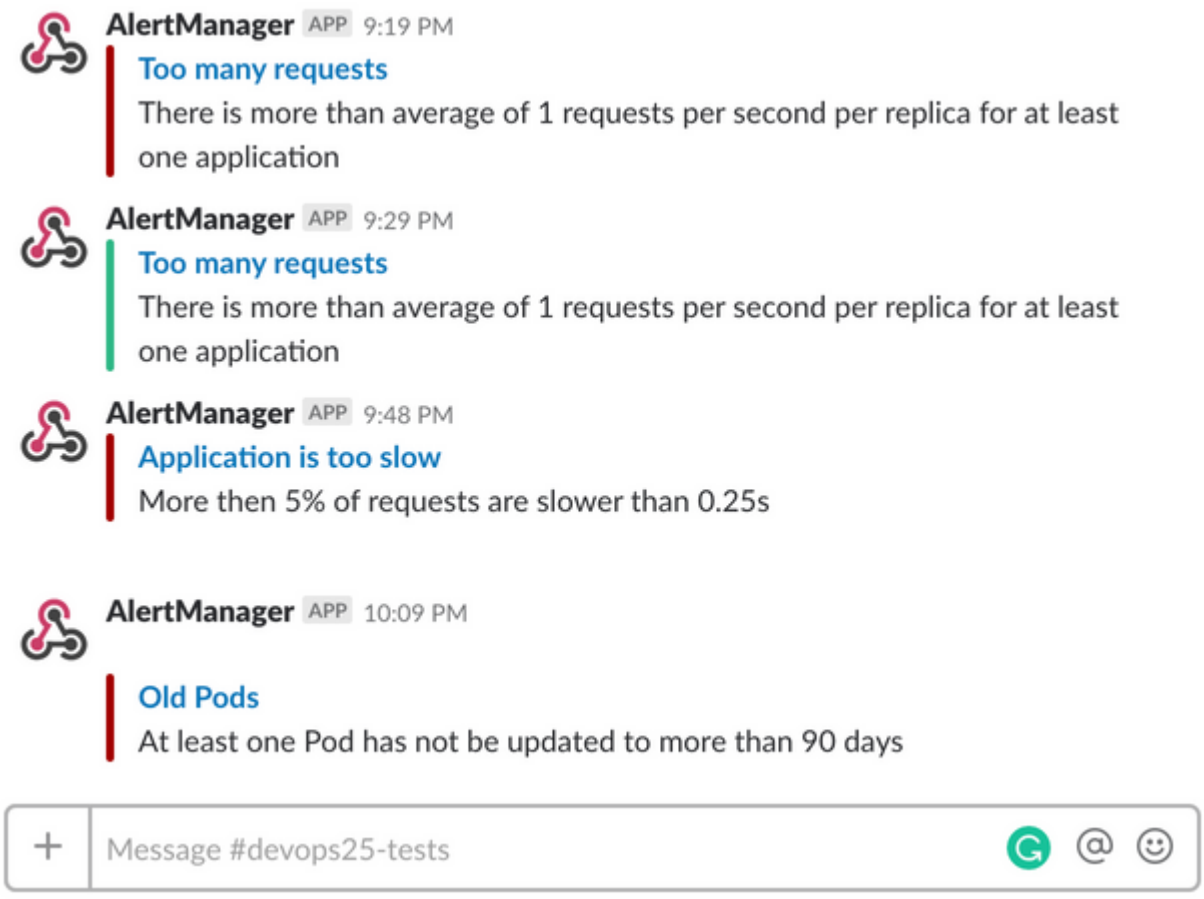
There was no need to specify `for` duration. The alert should fire the moment the age of one of the Pods reaches three months (give or take).

Let's upgrade our `Prometheus`'s Chart with the updated values and open the Slack channel where we should see the new message.

```
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-old-pods.yml

open "https://devops20.slack.com/messages/CD8QJA8DS/"
```

All that's left is to wait for a few moments until the new message arrives. It should contain the title *Old Pods* and the text stating that *at least one Pod has not been updated to more than 90 days*.



Slack with multiple fired and resolved alert messages

Such a generic alert might not work for all your use-cases. But, I'm sure that you'll be able to split it into multiple alerts based on Namespaces, names, or something similar.

Now that we have a mechanism to receive notifications when our Pods are too old and might require upgrades, we'll jump into the next topic and explore how to retrieve memory and CPU used by our containers in the next lesson.