Comparing Actual Resource Usage with Defined Requests

This lesson focuses on comparing actual resource usage with requested resources.

WE'LL COVER THE FOLLOWING

If we define container `resources` inside a Pod and without relying on actual usage, we are just guessing how much memory and CPU we expect a container to use. I'm sure that you already know why guessing, in the software industry, is a terrible idea, so I'll focus on Kubernetes aspects only.

Why define container `resources`?

Kubernetes treats Pods with containers that do not have specified resources as the *BestEffort Quality Of Service (QoS)*. As a result, if it ever runs out of memory or CPU to serve all the Pods, those are the first to be forcefully removed to leave space for others. If such Pods are short-lived as, for

example, those used as one-shot agents for continuous delivery processes, BestEffort QoS is not a bad idea. But, when our applications are long-lived,

BestEffort QoS should be unacceptable. That means that in most cases, we do have to define container `resources`.

Using guesstimates to measure actual usage

If container `resources` are (almost always) a must, we need to know which values to put. I often see teams that merely guess. “It’s a database; therefore it needs a lot of RAM” and “it’s only an API, it shouldn’t need much” are only a few of the sentences I hear a lot. Those guesstimates are often the result of not being able to measure actual usage. When something would blow up, those teams would just double the allocated memory and CPU. Problem solved!

I never understood why anyone would invent how much memory and CPU an application needs. Even without any “fancy” tools, we always had `top` command in Linux. We could know how much our application uses. Over time, better tools were developed, and all we had to do is Google “how to measure memory and CPU of my applications.” You already saw `kubectl top pods` in action when you need current data, and you are becoming familiar with the power of `Prometheus` to give you much more. You do not have an excuse to guesstimate.

Resource usage Vs Requested resources

But, why do we care about resource usage compared with requested resources? Besides the fact that it might reveal a potential problem (e.g., memory leak), inaccurate resource requests and limits prevent Kubernetes from doing its job efficiently. If, for example, we define the memory request to 1GB RAM, that’s how much Kubernetes will remove from allocatable memory. If a node has 2GB of allocatable RAM, only two such containers could run there, even if each uses only 50MB RAM. Our nodes would use only a fraction of allocatable memory and, if we have `Cluster Autoscaler`, new nodes would be added even if the old ones still have plenty of unused memory.

Alert when requested memory and CPU differs from actual usage

Even though we know how to get actual memory usage, it would be a waste of time to start every day by comparing YAML files with the results in

time to start every day by comparing YAML files with the results in `Prometheus`. Instead, we'll create yet another alert that will send us a notification whenever the requested memory and CPU differs too much from the actual usage. That's our next mission.

First, we'll reopen the `Prometheus`'s graph screen.

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

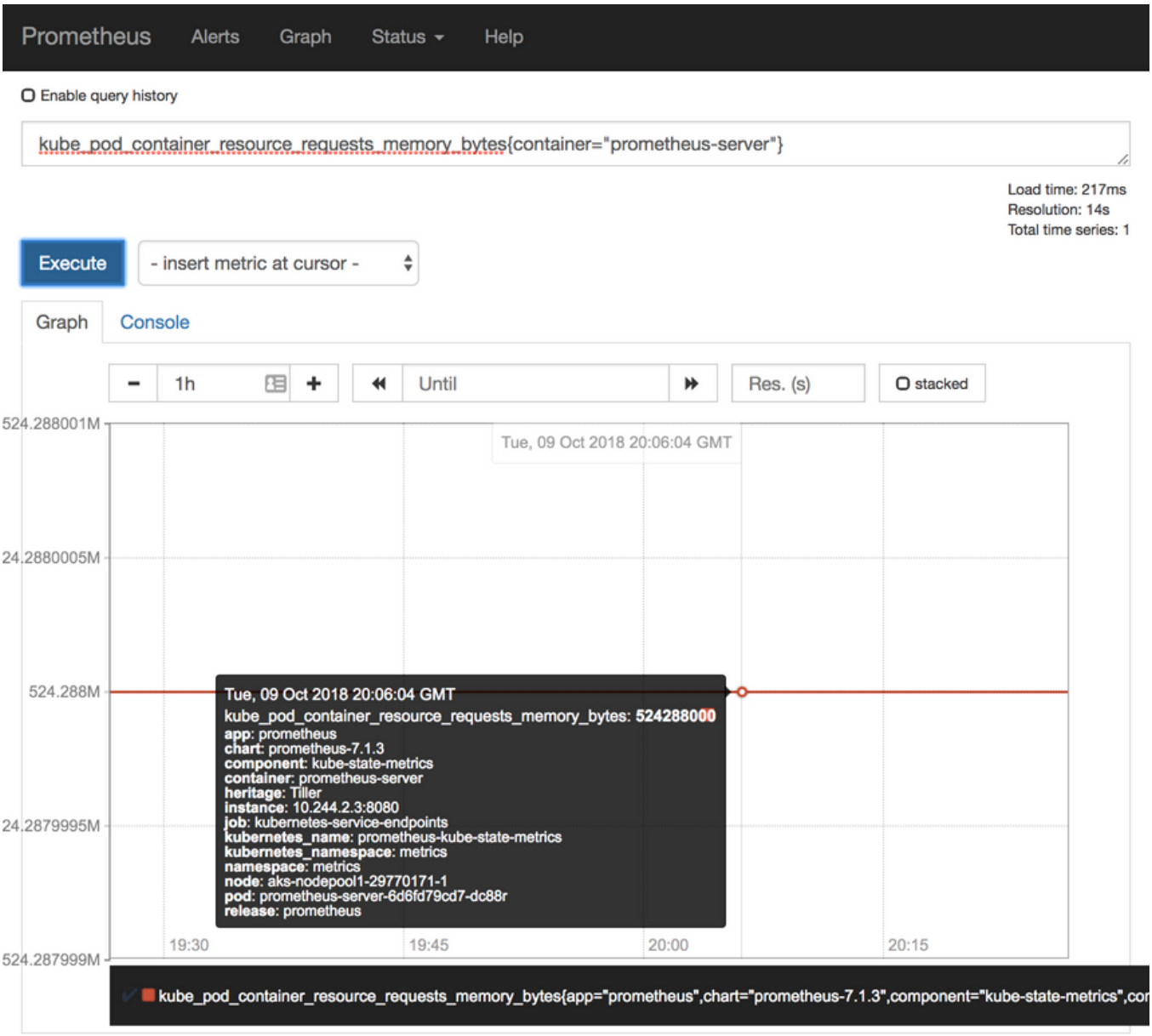
We already know how to get memory usage through `container_memory_usage_bytes`, so we'll jump straight into retrieving requested memory. If we can combine the two, we'll get the discrepancy between the requested and the actual memory usage.

The metric we're looking for is `kube_pod_container_resource_requests_memory_bytes`, so let's take it for a spin with, let's say, `prometheus-server` Pod.

Please type the expression that follows, press the *Execute* button, and switch to the Graph tab.

```
kube_pod_container_resource_requests_memory_bytes{
  container="prometheus-server"
}
```

We can see from the result that we requested 500MB RAM for the `prometheus-server` container.



Prometheus' graph screen with container requested memory limited to prometheus-server

Using `label_join` function

The problem is that the `kube_pod_container_resource_requests_memory_bytes` metric has, among others, a `pod` label while, on the other hand, `container_memory_usage_bytes` uses `pod_name`. If we are to combine the two, we need to transform the label `pod` into `pod_name`. Fortunately, this is not the first time we've faced that problem, and we already know that the solution is to use the `label_join` function to create a new label based on one or more of the existing labels.

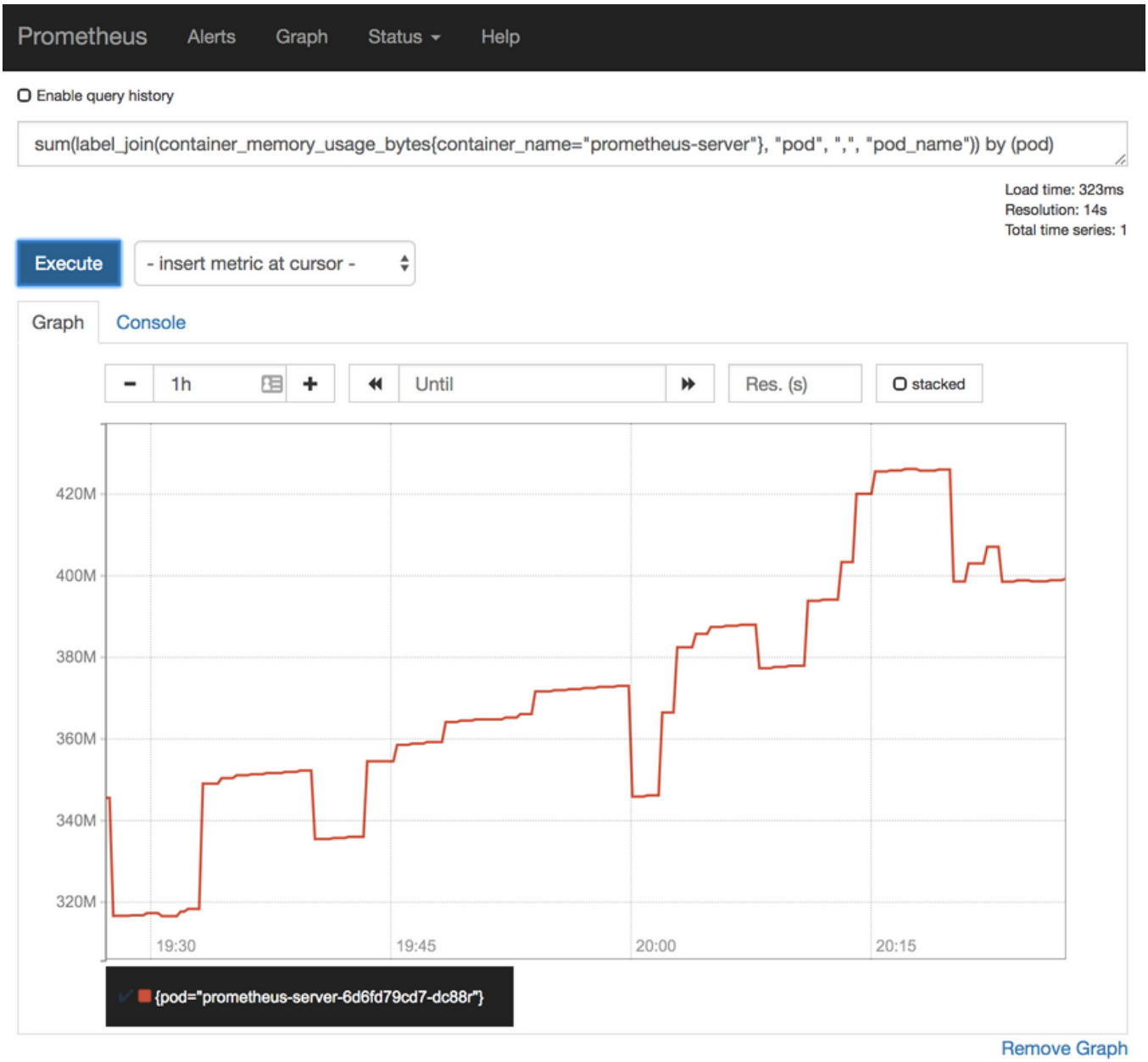
Please type the expression that follows, and press the *Execute* button.

```
sum(label_join(
  container_memory_usage_bytes{
```

```
    container_name="prometheus-server"
  },

  "pod",
  ",",
  "pod_name"
))
by (pod)
```

This time, not only did we add a new label to the metric, but we also grouped the results by that very same label (`by (pod)`).



Prometheus' graph screen with container memory usage limited to prometheus-server and grouped by the pod label extracted from pod_name

Now we can combine the two metrics and find out the discrepancy between the requested and the actual memory usage.

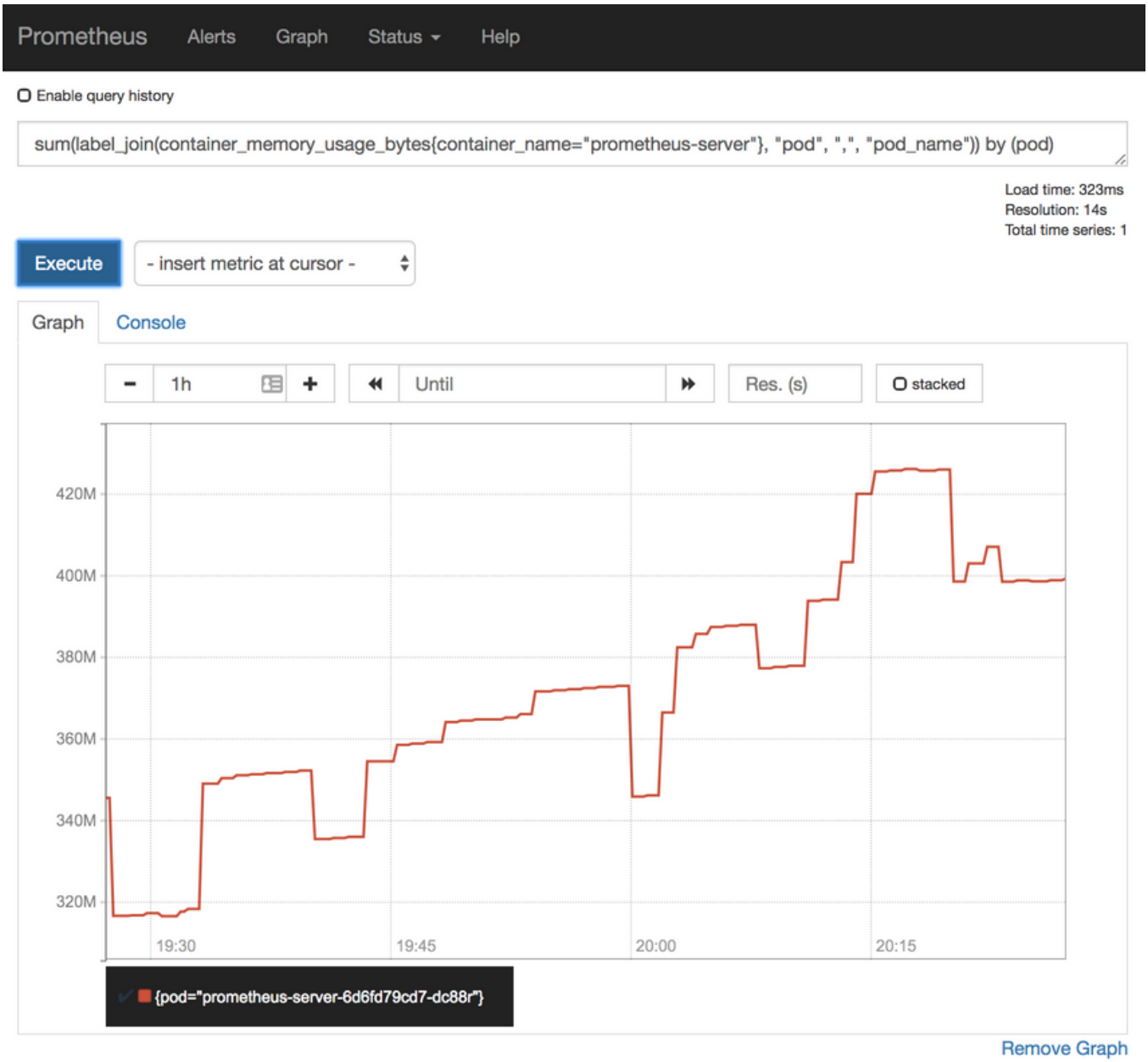
Find the discrepancy between requested and memory usage

Find the discrepancy between requested and memory usage

Please type the expression that follows, and press the *Execute* button.

```
sum(label_join(
  container_memory_usage_bytes{
    container_name="prometheus-server"
  },
  "pod",
  ",",
  "pod_name"
))
by (pod) /
sum(
  kube_pod_container_resource_requests_memory_bytes{
    container="prometheus-server"
  }
)
by (pod)
```

In my case (screenshot below), the discrepancy was becoming gradually smaller. It started somewhere around sixty percent, and now it's approximately seventy-five percent. Such a difference is not big enough for us to take any corrective action.



Prometheus' graph screen with the percentage of container memory usage based on requested memory

Make general expression to get containers

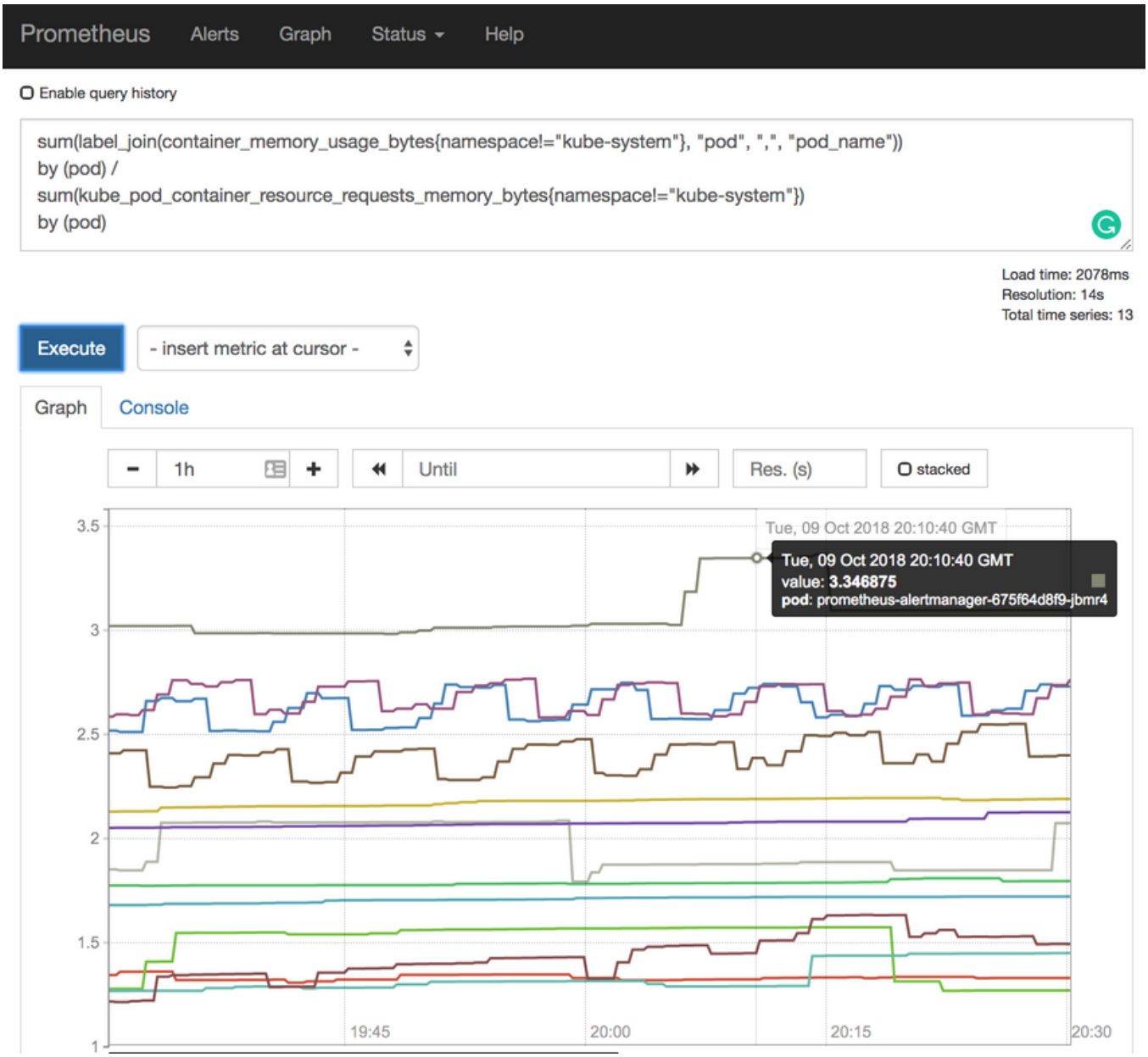
Now that we saw how to get the difference between reserved and actual memory usage for a single container, we should probably make the expression more general and get all the containers in the cluster. However, all might be a bit too much. We probably do not want to mess with the Pods running in the `kube-system` Namespace. They are likely pre-installed in the cluster, and we might want to leave them as they are, at least for now. So, we'll exclude them from the query.

Please type the expression that follows, and press the *Execute* button.

```
sum(label_join(
  container_memory_usage_bytes{
    namespace!="kube-system"
  },
  "pod",
  ",",
  "pod_name"
))
by (pod) /
sum(
  kube_pod_container_resource_requests_memory_bytes{
    namespace!="kube-system"
  }
)
by (pod)
```

The result should be the list of percentages of difference between requested and actual memory, with the Pods in the `kube-system` excluded.

In my case, there are quite a few containers that use a lot more memory than what we requested. The main culprit is `prometheus-alertmanager` which uses more than three times more memory than what we requested. That can be due to several reasons. Maybe, we requested too little memory. Or perhaps, it contains containers that do not have `requests` specified. In either case, we should probably redefine requests not only for the `Alertmanager` but also for all the other Pods that use more than, let's say 50% more memory than requested.



Prometheus' graph screen with the percentage of container memory usage based on requested memory and with those from the kube-system Namespace excluded

Q

The main culprit is `prometheus-alertmanager` which uses more than three times more memory than what we requested.

Alert when requested memory is much more or much less than the actual usage

We are about to define a new alert that will deal with cases when requested memory is much more or much less than the actual usage. But, before we do that, we should discuss the conditions we should use.

Defining conditions

One alert could fire when actual memory usage is over 150% of the requested memory for over an hour. That would remove false positives caused by a temporary spike in memory usage (that's why we have `limits` as well). The other alert could deal with the situation when memory usage is more than 50% below the requested amount. But, in case of that alert, we might add another condition.

Some applications are too small, and we might never be able to fine-tune their requests. We can exclude those cases by adding another condition that will ignore the Pods with only 5MB reserved RAM, or less.

Finally, this alert might not need to fire as frequently as the previous one. We should know relatively quickly if our application uses more memory than we intended to give since that can be a sign of a memory leak, significantly increased traffic, or some other potentially dangerous situation. But if memory uses much less than intended, the issue is not as critical. We should correct it, but there is no need to act urgently. Therefore, we'll set the duration of the latter alert to six hours.

Now that we have set a few rules we should follow, we can take a look at yet another diff between the old and the new set of Chart's values.

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

The **output** is as follows.

```
148c148
<   expr: (time() - kube_pod_start_time{namespace!="kube-system"}) > 60
---
```

```
>   expr: (time() - kube_pod_start_time{namespace!="kube-system"}) > (60 *
 60 * 24 * 90)
154a155,172
> - alert: ReservedMemTooLow
>   expr: sum(label_join(container_memory_usage_bytes{namespace!="kube-sys
tem", namespace!="ingress-nginx"}, "pod", ",", "pod_name")) by (pod) / sum
(kube_pod_container_resource_requests_memory_bytes{namespace!="kube-syste
m"}) by (pod) > 1.5
>   for: 1m
>   labels:
>     severity: notify
>     frequency: low
>   annotations:
>     summary: Reserved memory is too low
>     description: At least one Pod uses much more memory than it reserved
> - alert: ReservedMemTooHigh
>   expr: sum(label_join(container_memory_usage_bytes{namespace!="kube-sys
tem", namespace!="ingress-nginx"}, "pod", ",", "pod_name")) by (pod) / sum
(kube_pod_container_resource_requests_memory_bytes{namespace!="kube-syste
m"}) by (pod) < 0.5 and sum(kube_pod_container_resource_requests_memory_by
tes{namespace!="kube-system"}) by (pod) > 5.25e+06
>   for: 6m
>   labels:
>     severity: notify
>     frequency: low
>   annotations:
>     summary: Reserved memory is too high
>     description: At least one Pod uses much less memory than it reserved
```

First, we set the threshold of the `OldPods` alert back to its intended value of ninety days (`60 * 60 * 24 * 90`). That way we'll stop it from firing alerts only for test purposes.

Defining a new alert `ReservedMemTooLow`

Next, we defined a new alert called `ReservedMemTooLow`. It will fire if used memory is more than `1.5` times bigger than the requested memory. The duration for the pending state of the alert is set to `1m`, only so that we can see the outcome without waiting for the full hour. Later on, we'll restore it back to `1h`.

Defining a new alert `ReservedMemTooHigh`

The `ReservedMemTooHigh` alert is (partly) similar to the previous one, except

that it has the condition that will cause the alert to fire if the difference between the actual and the requested memory is less than `0.5` and continues for over `6m` (we'll change it later to `6h`). The second part of the expression is new. It requires that all the containers in a Pod have more than 5BM of the requested memory (`5.25e+06`). Through that second statement (separated with `and`), we're saving ourselves from dealing with too small applications. If it requires less than 5MB RAM, we should ignore it and, probably, congratulate the team behind it for making it that efficient.

Now, let's upgrade our `Prometheus`'s Chart with the updated values and open the graph screen.

```
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-req-mem.yml
```

We won't wait until the alerts start firing. Instead, we'll try to accomplish similar objectives, but with CPU.

There's probably no need to go through the process of explaining the expressions we'll use. We'll jump straight into the CPU-based alerts by exploring the diff between the old and the new set of Chart's values.

```
diff mon/prom-values-req-mem.yml \
    mon/prom-values-req-cpu.yml
```

The **output** is as follows.

```
157c157
<   for: 1m
---
>   for: 1h
166c166
<   for: 6m
---
>   for: 6h
172a173,190
> - alert: ReservedCPUTooLow
```

```
>   expr: sum(label_join(rate(container_cpu_usage_seconds_total{namespace!
="kube-system", namespace!="ingress-nginx", pod_name!=""}[5m]), "pod", ","

, "pod_name")) by (pod) / sum(kube_pod_container_resource_requests_cpu_cor
es{namespace!="kube-system"}) by (pod) > 1.5
>   for: 1m
>   labels:
>     severity: notify
>     frequency: low
>   annotations:
>     summary: Reserved CPU is too low
>     description: At least one Pod uses much more CPU than it reserved
> - alert: ReservedCPUTooHigh
>   expr: sum(label_join(rate(container_cpu_usage_seconds_total{namespace!
="kube-system", pod_name!=""}[5m]), "pod", ",", "pod_name")) by (pod) / su
m(kube_pod_container_resource_requests_cpu_cores{namespace!="kube-system"
}) by (pod) < 0.5 and sum(kube_pod_container_resource_requests_cpu_cores{n
amespace!="kube-system"}) by (pod) > 0.005
>   for: 6m
>   labels:
>     severity: notify
>     frequency: low
>   annotations:
>     summary: Reserved CPU is too high
>     description: At least one Pod uses much less CPU than it reserved
```

The first two sets of differences define more sensible thresholds for `ReservedMemTooLow` and `ReservedMemTooHigh` alerts we explored previously. Further down, we can see the two new alerts.

The `ReservedCPUTooLow` alert will fire if CPU usage is more than 1.5 times bigger than requested. Similarly, the `ReservedCPUTooHigh` alert will fire only if CPU usage is less than half of the requested and if we requested more than 5 CPU milliseconds. Getting notifications because 5MB RAM is too much would be a waste of time.

Both alerts are set to fire if the issues persist for a short period (`1m` and `6m`) so that we can see them in action without having to wait for too long.

Now, let's upgrade our `Prometheus`'s Chart with the updated values.

```
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-req-cpu.yml
```

I'll leave it to you to check whether any of the alerts fire and whether they are forwarded from `Alertmanager` to Slack. You should know how to do that by now.

Next, we'll move to the last alert in this chapter.