Measuring Containers Memory and CPU Usage

In this lesson, we will see how to measure memory and CPU usage of an individual container.

WE'LL COVER THE FOLLOWING

Observe metrics over time

If you are familiar with Kubernetes, you understand the importance of defining resource requests and limits. Since we already explored `kubectl top pods` command, you might have set the requested resources to match the current usage, and you might have defined the limits to be above the requests. That approach might work on the first day. But, with time, those numbers will change, and we will not be able to get the full picture through `kubectl top pods`. We need to know how much memory and CPU containers use when on their peak loads and how much they use when they are under less stress. We should observe those metrics over time and adjust periodically.

Even if we do somehow manage to guess how much memory and CPU a container needs, those numbers might change from one release to another. Maybe we introduced a feature that requires more memory or CPU?

What we need is to observe resource usage over time and make sure that it does not change with new releases or with increased (or decreased) numbers of users. For now, we'll focus on the former case and explore how to see how much memory and CPU our containers have used over time.

Measure memory and CPU over time

As usual, we'll start by opening the `Prometheus`'s graph screen.

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

We can retrieve container memory usage through the `container_memory_usage_bytes`.

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

```
container_memory_usage_bytes
```

If you take a closer look at the top usage, you'll probably end up confused. It seems that some containers are using way more than the expected amount of memory.

The truth is that some of the `container_memory_usage_bytes` records contain cumulative values, and we should exclude them so that only memory usage of individual containers is retrieved. We can do that by retrieving only the records that have a value in the `container_name` field.

Retrieve memory usage of individual containers

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

```
container_memory_usage_bytes{
  container_name!=""
}
```

Now, the result makes much more sense. It reflects the memory usage of the containers running inside our cluster.

We'll get to alerts based on container resources a bit later. For now, we'll imagine that we'd like to check the memory usage of a specific container (e.g., `prometheus-server`). Since we already know that one of the available labels is `container_name`, retrieving the data we need should be straightforward.

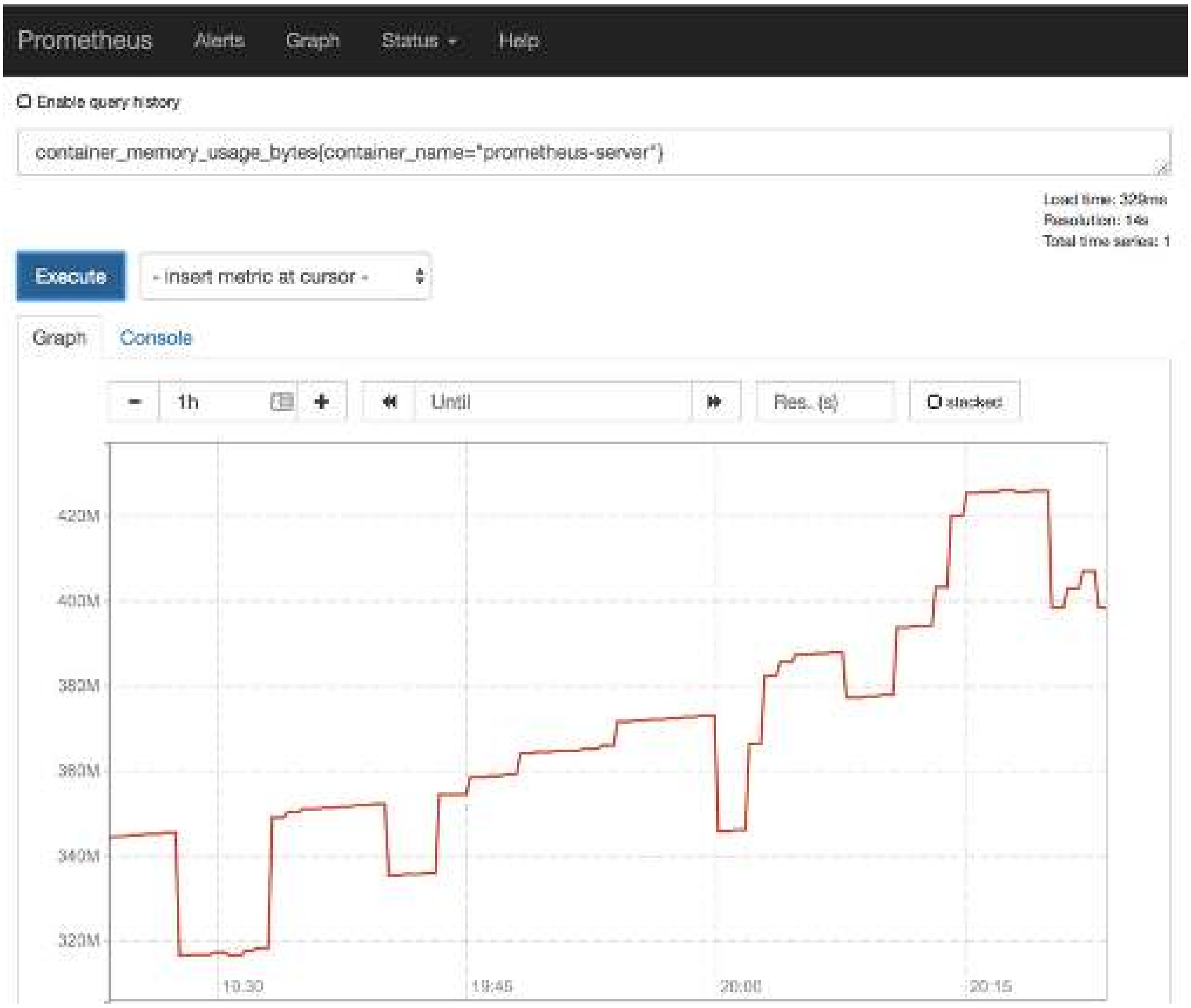
Retrieve memory usage of `prometheus-server`

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

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

```
container_memory_usage_bytes{
  container_name="prometheus-server"
}
```

We can see the oscillations in memory usage of the container over the last hour. Normally, we'd be interested in a longer period like a day or a week. We can accomplish that by clicking the - and + buttons above the graph, or by typing the value directly in the field between them (e.g., *1w*). However, changing the duration might not help much since we haven't been running the cluster for too long. We might not be able to squeeze more data than a few hours unless you are a slow reader.



```
container_memory_usage_bytes{container_name="prometheus-server"}
```

Load time: 329ms
Resolution: 14s
Total time series: 1

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

Retrieve CPU usage of individual containers

Similarly, we should be able to retrieve the CPU usage of a container as well.

In that case, the metric we're looking for could be

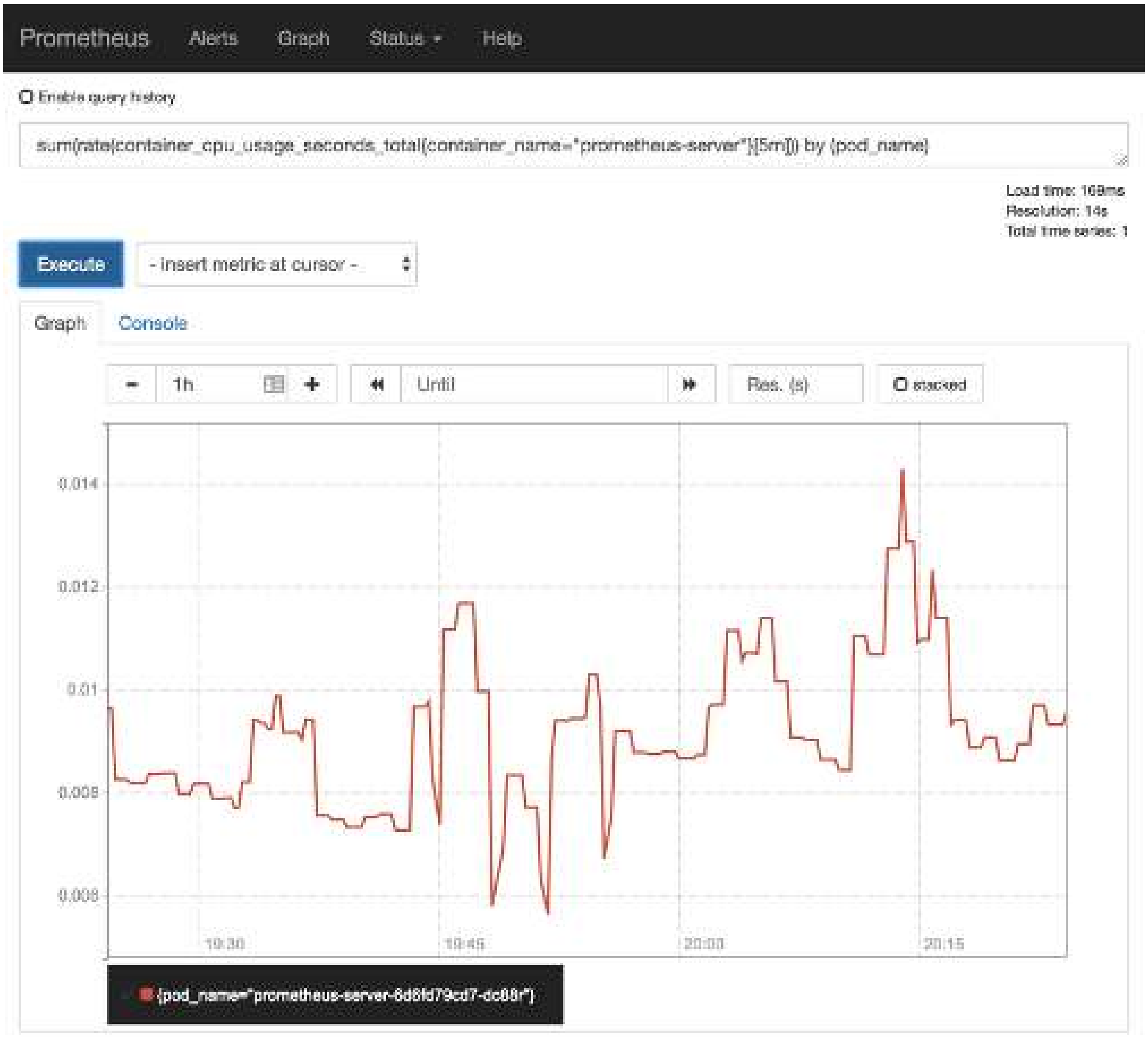
`container_cpu_usage_seconds_total`. However, unlike `container_memory_usage_bytes` which is a gauge, `container_cpu_usage_seconds_total` is a counter, and we'll have to combine `sum` and `rate` to get the changes in values over time.

Retrieve CPU usage of `prometheus-server`

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

```
sum(rate(
  container_cpu_usage_seconds_total{
    container_name="prometheus-server"
  }[5m]
))
by (pod_name)
```

The query shows a summed CPU seconds rate over five minutes intervals. We added `by (pod_name)` to the mix so that we can distinguish different Pods and see when one was created, and the other was destroyed.



```
sum(rate(container_cpu_usage_seconds_total{container_name="prometheus-server"}[5m])) by (pod_name)
```

Load time: 169ms
Resolution: 14s
Total time series: 1

{pod_name="prometheus-server-6d6fd79cd7-dc88r"}

Prometheus' graph screen with the rate of container CPU usage limited to prometheus-server

Q

Some of the `container_memory_usage_bytes` records contain cumulative values, and we should include them so that only memory usage of individual containers is retrieved.

COMPLETED 0%

1 of 1

If that were a “real world” situation, our next step would be to compare actual resource usage with what we defined as `Prometheus`'s `resources`. If what we defined differs considerably compared to what it actually is, we should probably update our Pod definition (the `resources` section).

The problem is that using “real” resource usage to define Kubernetes `resources` better will provide valid values only temporarily. Over time, our resource usage will change. The load might increase, new features might be more resource-hungry, and so on. No matter the reasons, the critical thing to note is that everything is dynamic and that there is no reason to think otherwise for resources.

In that spirit, our next challenge is to figure out how to get a notification when the actual resource usage differs too much from what we defined in the container `resources`. We will see this in the next lesson.