← Back to All Blogs

Monitoring Kubernetes with Prometheus | Prometheus Docker Monitoring

August 03, 2016 • By Joe Bowers

Tags: Prometheus technical posts

Monitoring is one of the pillars of successful infrastructure. It has been called the base of the hierarchy of reliability. Monitoring is a must have for responding to incidents, detecting and debugging systemic problems, planning for the future, and generally understanding your infrastructure.

Monitoring modern infrastructure

Contemporary cluster architectures like those using Kubernetes, where loosely coupled containers come and go depending on when and where they are needed, require a new approach to effective monitoring. It's not enough to watch load and activity on a particular machine when the work assigned to that machine may change with the requirements of the cluster and application as a whole. If your architecture is designed to allow for processes to fail, you can't tell if a failure is critical by monitoring individual processes.

Effective use of Kubernetes requires architects to think about applications and workloads rather than machines and processes. Monitoring applications deployed to Kubernetes requires the same shift in thinking. Like using Kubernetes for your workloads, taking a new approach to monitoring doesn't just give you today's benefits and features in a different form. Cluster-centric monitoring allows us to observe and react to trends and events at the application level; we can get more actionable insight and less noise from our new tools than from the old tools they're replacing.

Prometheus is an open source monitoring and alerting toolkit made for monitoring applications in clusters. Inspired by Google's *Borgmon*, it was designed from day one for the Kubernetes model of assigning and managing units of work. Prometheus also has beta support for using the Kubernetes API itself to discover cluster services and monitoring targets.

Kubernetes features built in support for Prometheus metrics and labels as well, and Prometheus support and integration continues to advance on both sides. This post shows how to use Prometheus to monitor the essential workers running in your Kubernetes cluster.

In this blog post, we'll set up a basic installation of Prometheus on your Kubernetes cluster with just a few manifests, and then use it to demonstrate how to inspect containers, pods, and whole namespaces in Kubernetes.

The setup

Like the jobs it monitors, Prometheus runs as a pod in your Kubernetes cluster. Deploying Prometheus requires two Kubernetes objects: a Deployment for the Prometheus pod itself, and a ConfigMap that tells Prometheus how to connect to your cluster. This manifest file contains descriptions of both objects. To deploy this manifest you can use kubectl with an existing Kubernetes cluster:

```
$ kubectl create -f https://raw.githubusercontent.com/coreos/blog-examples/master/monitoring-kubernetes-with-prometheus/prometheu
s.yml
```

The configuration in the example manifest follows the default Prometheus Kubernetes example fairly closely. As soon as this manifest is loaded into your cluster, Prometheus will start collecting data. Your queries will return more interesting results if you wait a bit, to give Prometheus time to collect information.

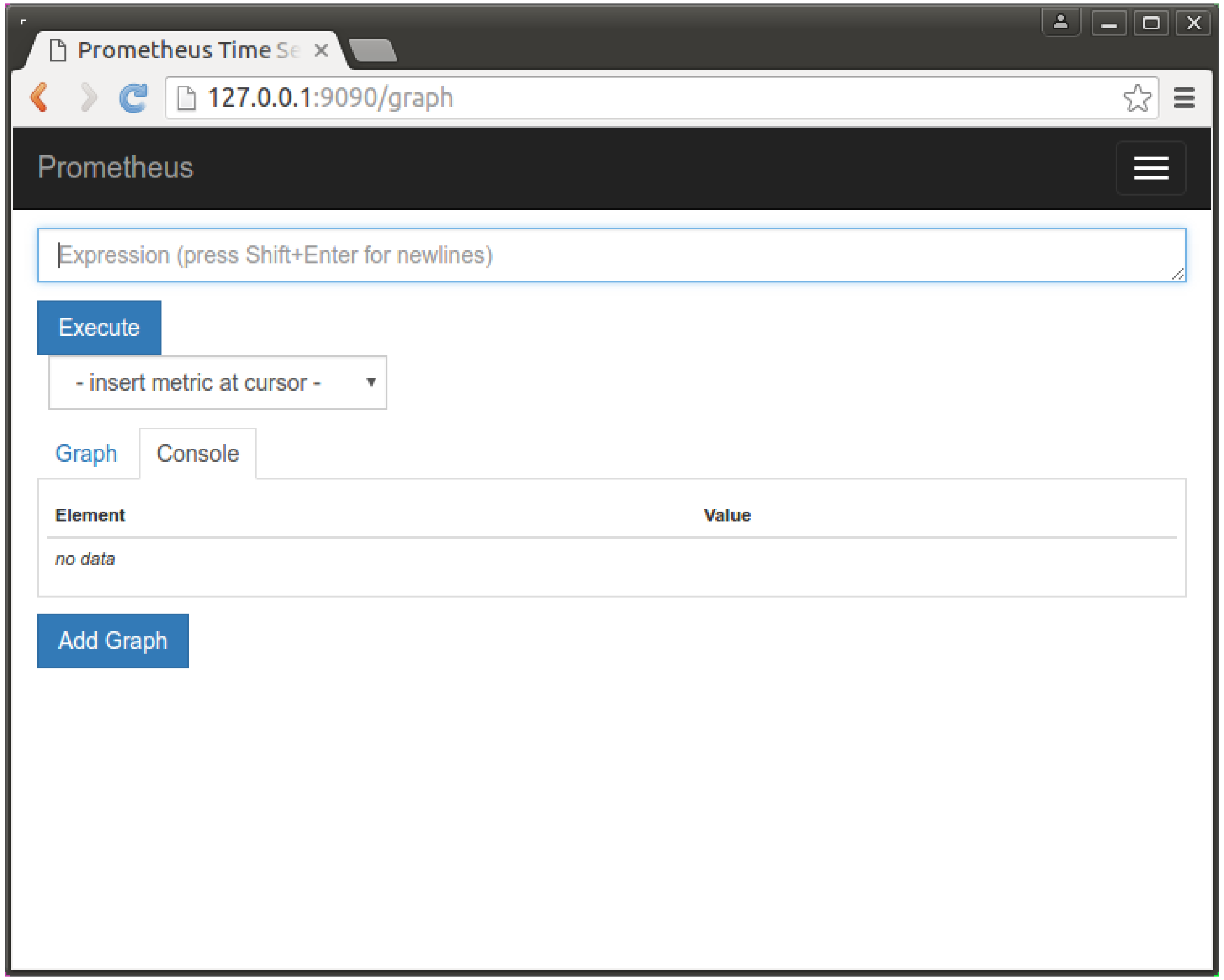
This simple setup stores the results of Prometheus's data collection in an ephemeral emptyDir volume. If the pod is killed or rescheduled, your data will start over from zero.

Viewing cluster metrics with Prometheus queries

To query and view your metrics, you can use `kubectl port-forward` to proxy connections to the Prometheus web UI:

```
$ kubectl get pods -l app=prometheus -o name | \
        sed 's/^.*\///' | \
        xargs -I{} kubectl port-forward {} 9090:9090
```

This command line finds the name of the Prometheus pod, cleans the name up a bit with sed, and then forwards port 9090 on your workstation to that pod. While it’s running, you’ll be able to see the Prometheus query UI at http://127.0.0.1:9090.

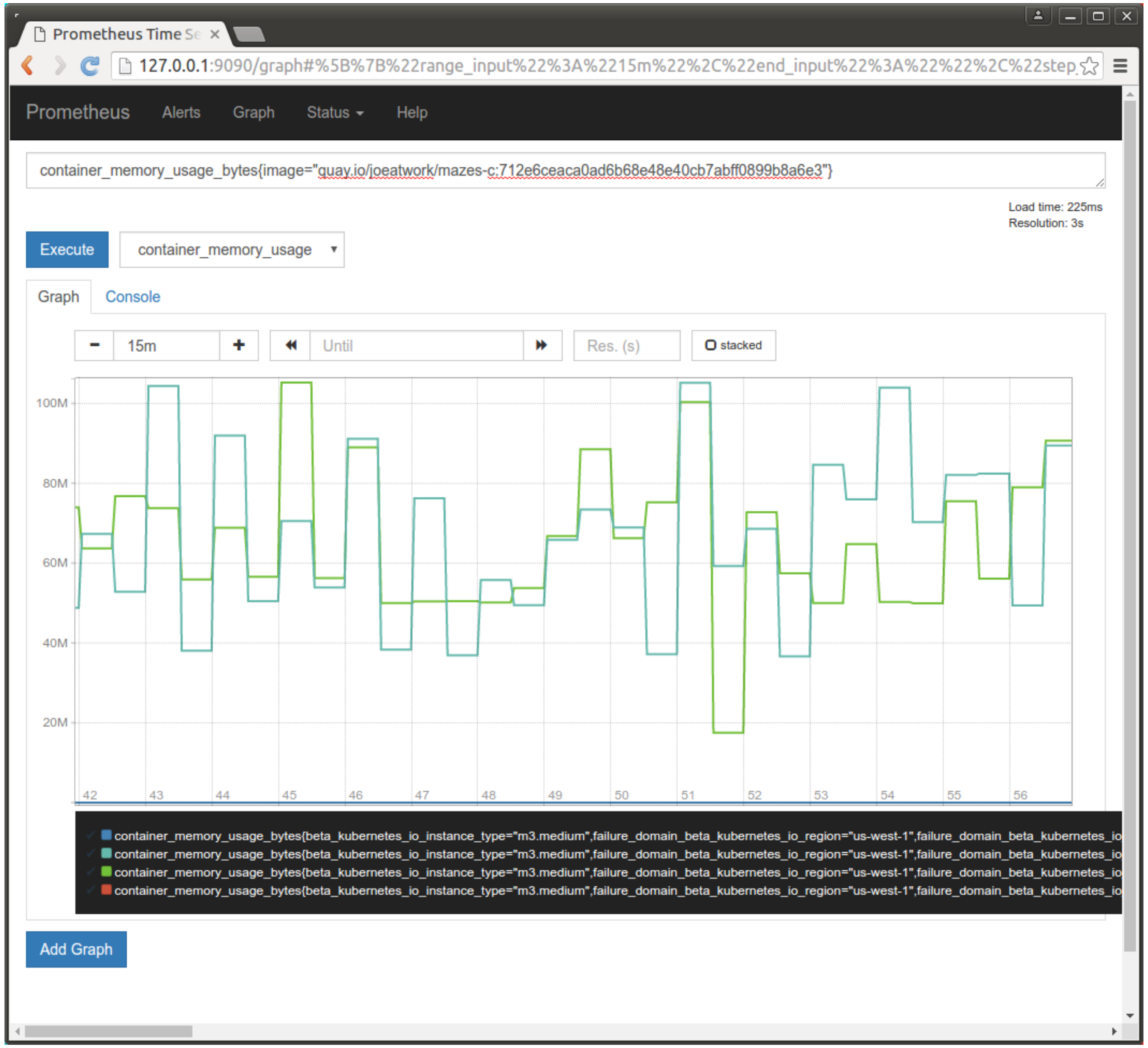


Element	Value
no data	

The Prometheus query UI

Querying

As soon as it starts, the Prometheus pod will be accumulating data. After a few minutes, you'll have some records that you can query using Prometheus's powerful query language. Kubernetes collects container metrics through cAdvisor. You can see all of the metrics that Kubernetes and cAdvisor provide out of the box in the dropdown menu on the query interface.



```
container_memory_usage_bytes{image="quay.io/joeatwork/mazes-c:712e6ceaca0ad6b68e48e40cb7abff0899b8a6e3"}
```

```
container_memory_usage_bytes{beta_kubernetes_io_instance_type="m3.medium",failure_domain_beta_kubernetes_io_region="us-west-1",failure_domain_beta_kubernetes_io
container_memory_usage_bytes{beta_kubernetes_io_instance_type="m3.medium",failure_domain_beta_kubernetes_io_region="us-west-1",failure_domain_beta_kubernetes_io
container_memory_usage_bytes{beta_kubernetes_io_instance_type="m3.medium",failure_domain_beta_kubernetes_io_region="us-west-1",failure_domain_beta_kubernetes_io
container_memory_usage_bytes{beta_kubernetes_io_instance_type="m3.medium",failure_domain_beta_kubernetes_io_region="us-west-1",failure_domain_beta_kubernetes_io
```

List of all default Kubernetes metrics

The CPU, memory, I/O, and network activity of your running containers are all measured, collected, and made available for queries and alerts. For example, you can see the memory usage of all of the containers running a particular image with a query like:

```
container_memory_usage_bytes{image="CONTAINER:VERSION"}
```

Querying a container by image

It might not always be convenient to know the precise version of your container images - or perhaps you'd like to see memory usage across all versions of your image. For cases like these, the Prometheus query language supports regular expressions when matching events. You can rewrite your memory usage query as:

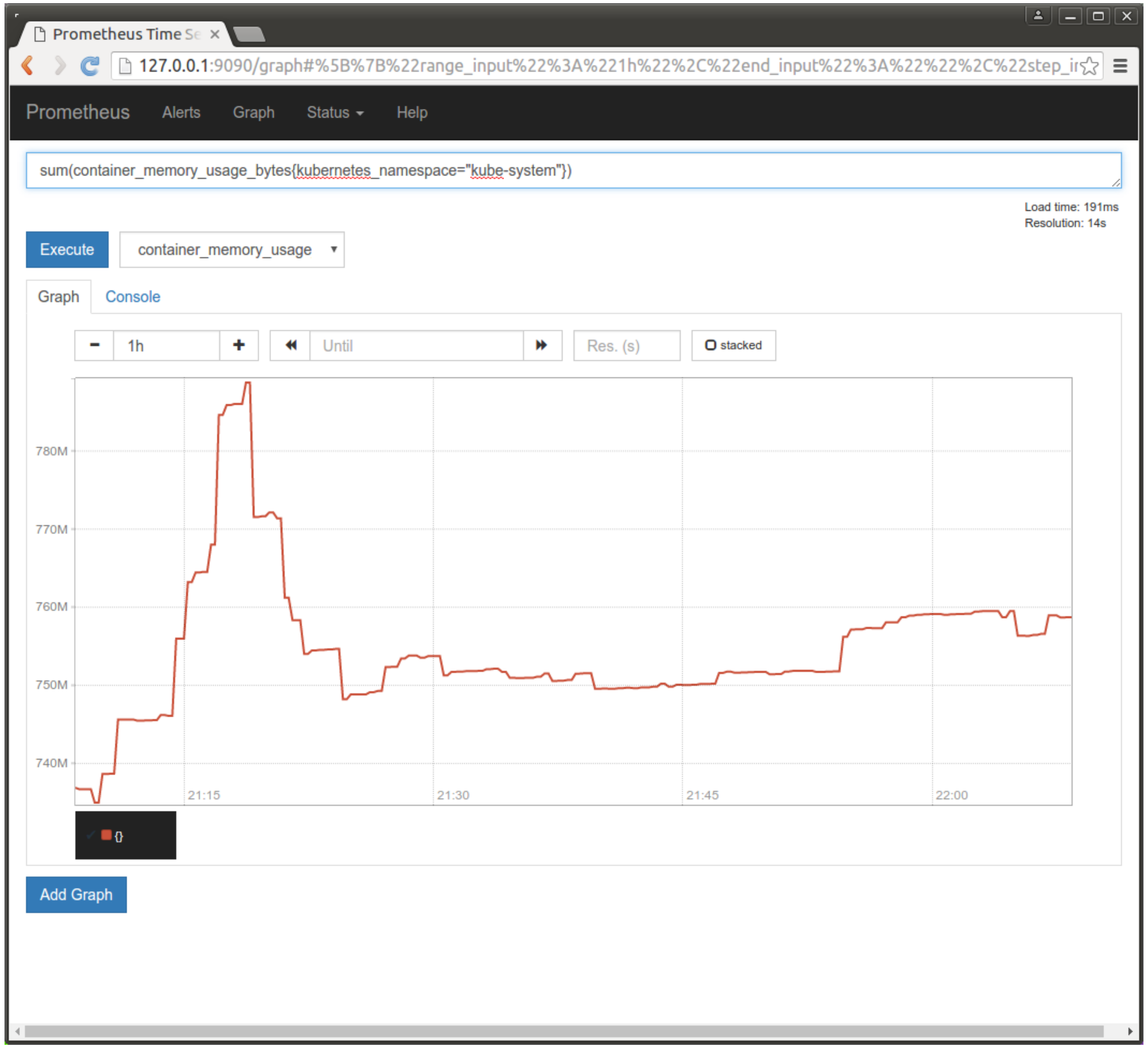
```
container_memory_usage_bytes{image=~"CONTAINER:.*"}
```

This is more convenient to type and more flexible across changes. Note that the equals sign after `image`, "=", has been changed to an equals sign and tilde, "=~", to indicate that the value in quotation marks is a regular expression.

With Prometheus monitoring Kubernetes, container- and image-level querying is just the beginning. Kubernetes annotates your container metrics with metadata that you can use to get a refined, application-level view of the performance of your workloads. Container-level metrics are labeled with the Kubernetes pod and namespace they are associated with. That means you can "zoom out" from the container to the application and the cluster layers, to answer increasingly global questions.

For example, how much memory are the tools in the `kube-system` namespace using? That's a question you can answer with a query like:

```
sum(container_memory_usage_bytes{kubernetes_namespace="kube-system"})
```

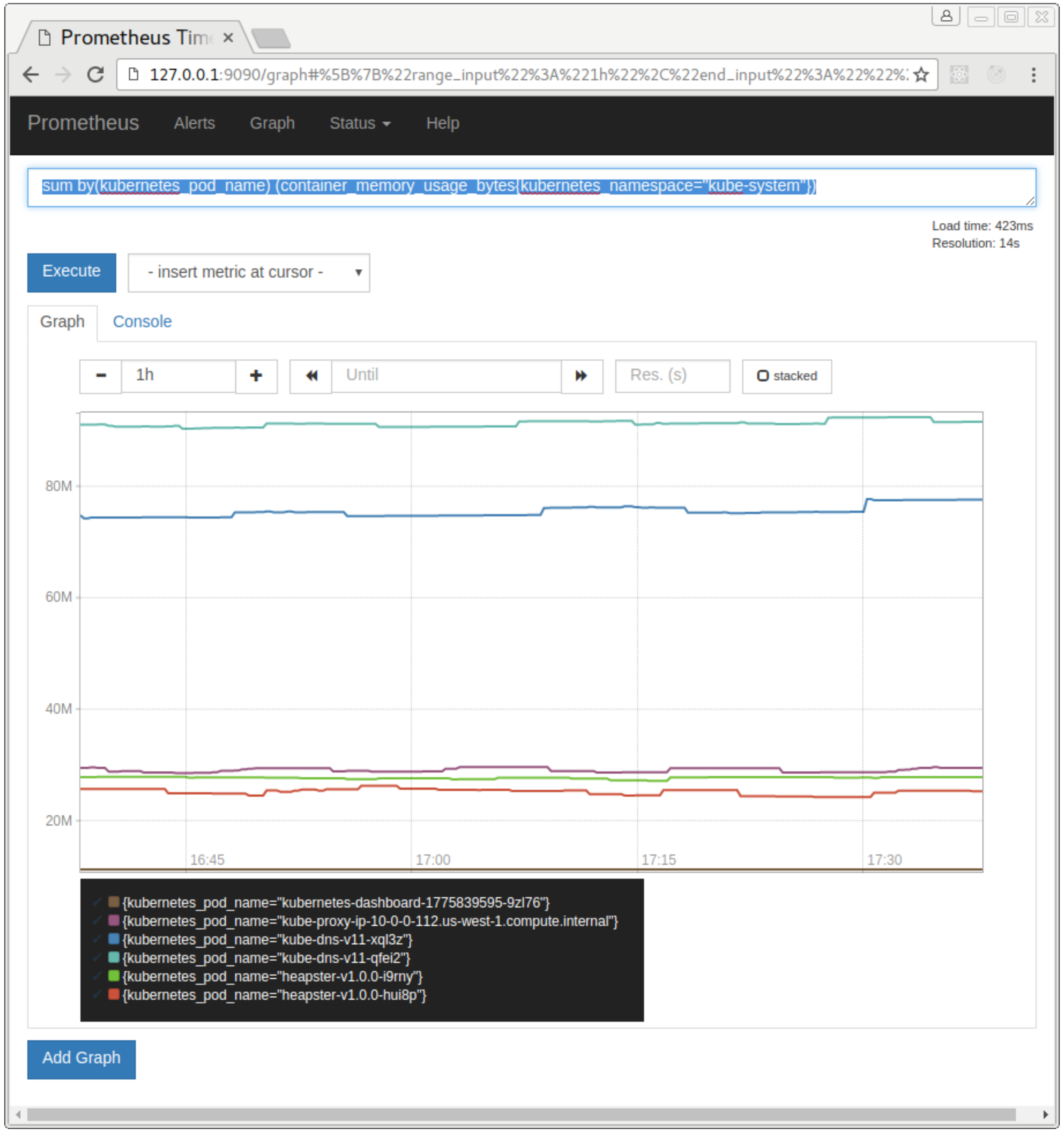


```
sum(container_memory_usage_bytes{kubernetes_namespace="kube-system"})
```

Zoom out to answer questions about namespaces as a whole

Container metrics are labeled by pod, as well as by namespace, so you can break down your results by individual pods using the `by` operator:

```
sum by(kubernetes_pod_name) (container_memory_usage_bytes{kubernetes_namespace="kube-system"})
```



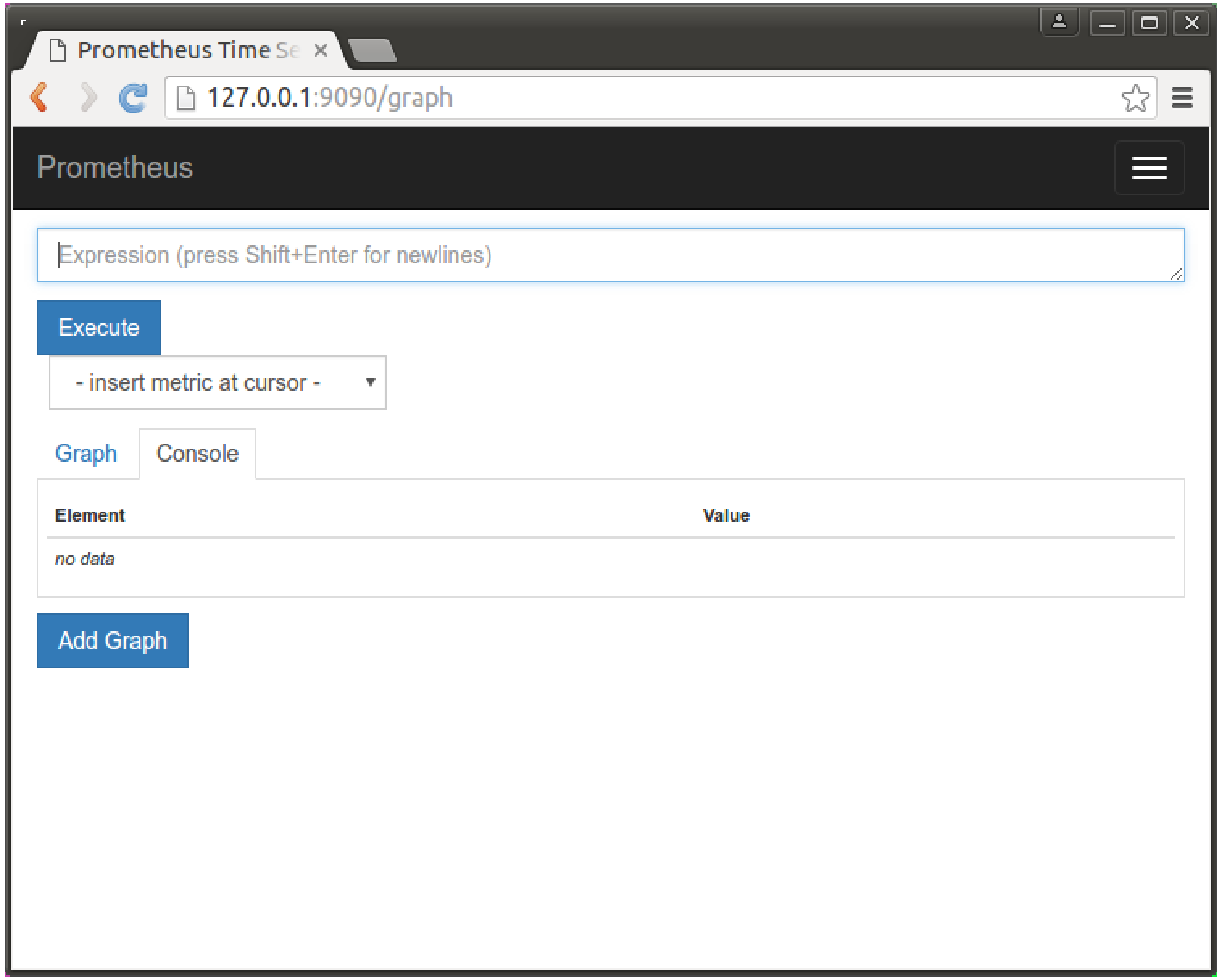
```
sum by(kubernetes_pod_name) (container_memory_usage_bytes{kubernetes_namespace="kube-system"})
```

```
{kubernetes_pod_name="kubernetes-dashboard-1775839595-9zl76"}
{kubernetes_pod_name="kube-proxy-ip-10-0-0-112.us-west-1.compute.internal"}
{kubernetes_pod_name="kube-dns-v11-xql3z"}
{kubernetes_pod_name="kube-dns-v11-qfei2"}
{kubernetes_pod_name="heapster-v1.0.0-i9rny"}
{kubernetes_pod_name="heapster-v1.0.0-hui8p"}
```

Break down your results using the `by()` operator

In Kubernetes, a single deployment or replication controller can be associated with multiple pods. To see all of the metrics associated with a particular deployment or replica set, you can use the pod naming convention and a regular expression to see all of the pods together. For example, all of the pods scheduled by the `kube-dns-v11` ReplicationController will have names beginning with “kube-dns-v11-”. Since Prometheus supports regular expression queries, we can see the memory in use by our Kubernetes system DNS with

```
sum(container_memory_usage_bytes{kubernetes_namespace="kube-system", kubernetes_pod_name=~"kube-dns-v11.*"})
```

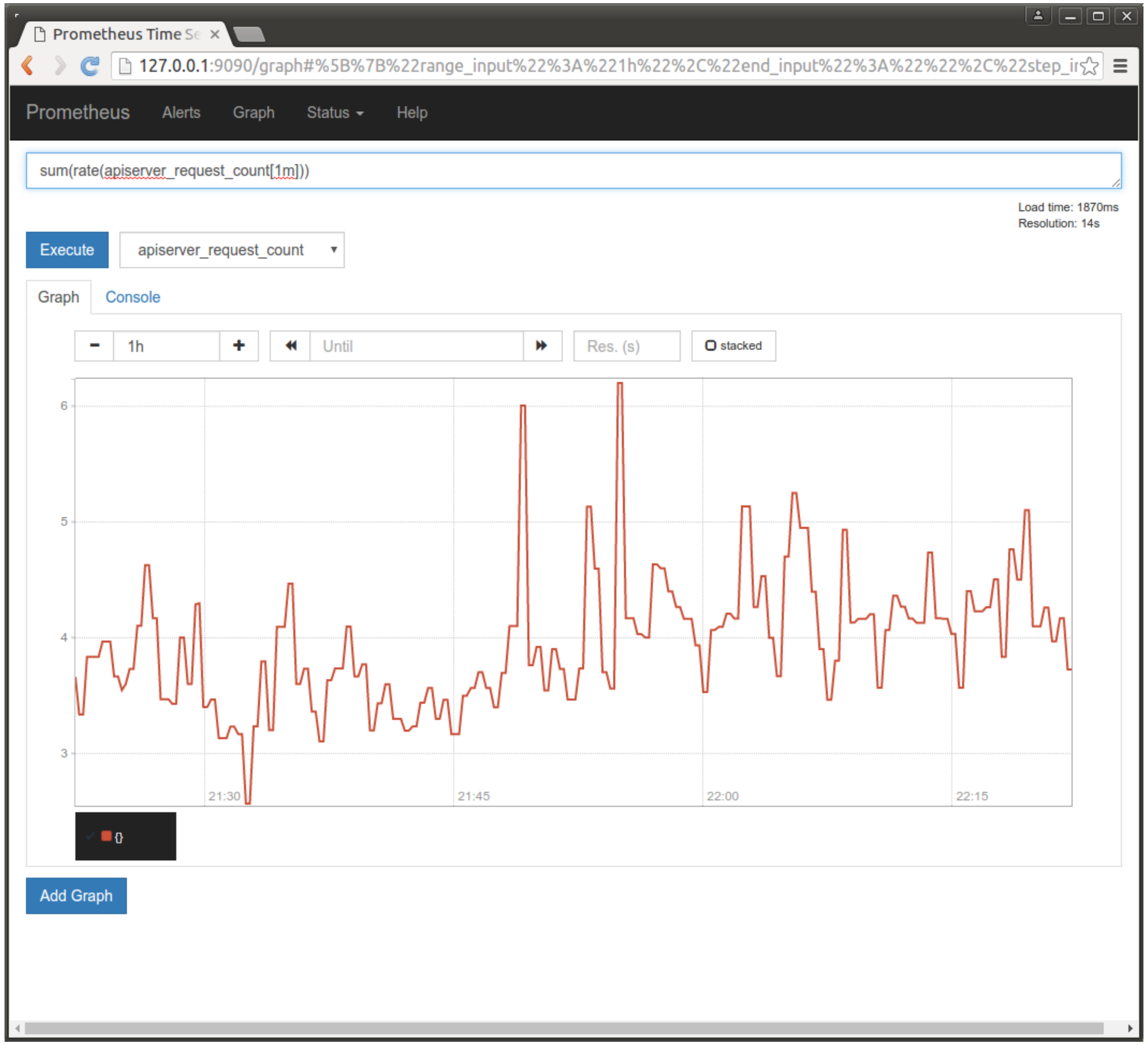


Zooming in on the Pods associated with the kube-dns-v11 ReplicationController

You can see the full list of labels provided by Kubernetes on the “Console” tab of the query tool.

There are also Kubernetes-specific metrics you can query – in particular, the API server and the Kubelet on each worker node exposes information in addition to the cAdvisor container metrics. For example, the API server reports its overall request count.

```
sum(rate(apiserver_request_count[1m]))
```


Prometheus Time Se
127.0.0.1:9090/graph#%5B%7B%22range_input%22%3A%221h%22%2C%22end_input%22%3A%22%22%2C%22step_ir
Prometheus
Alerts
Graph
Status
Help
Execute
Graph
Console
stacked
Add Graph

```
sum(rate(apiserver_request_count[1m]))
```

Load time: 1870ms
Resolution: 14s

apiserver_request_count

1h | Until | Res. (s)

Everybody wins

Something to note here is that we've been using the same query tools to inspect the behavior and performance of kube-dns, a core Kubernetes service, that you might use to inspect the applications you deploy to Kubernetes yourself. Right now, as long as a pod includes at least one container that exposes at least one port, Prometheus's automatic service discovery will include it in the data collection.

At CoreOS, we believe that effective monitoring is one of the keys to successful infrastructure, and we believe that Prometheus is the right tool for the job. That's why we're helping to fund Prometheus development, employing our own developers on upstream Prometheus, and including Prometheus by default in Tectonic, the enterprise Kubernetes distribution.

Try out Tectonic Starter to quickly get the benefits of GIFEE, Kubernetes and the CoreOS stack.

Related Posts:

+ The Prometheus Operator: Managed Prometheus setups for Kubernetes
+ CoreOS and Prometheus: Building monitoring for the next generation of cluster infrastructure

IRC #coreos

@CoreOS