

Capture Traefik Metrics for Apps on Kubernetes with Prometheus



Neil McAllister • [How To](#) • [Kubernetes](#) • February 23, 2021

Capture Traefik Metrics for Apps on Kubernetes with Prometheus



Monitoring distributed systems is one of the core precepts of [site reliability engineering \(SRE\)](#), as defined by Google. When Traefik is deployed as a Kubernetes ingress controller, it becomes an integral part of this practice.

This is the second in a series of blog posts on using Traefik to help enable SRE practices. The [previous entry](#) discussed how the Elastic stack of monitoring software can connect

Prerequisites

As in the previous installment, if you want to follow along with this tutorial, you'll need to have a few things set up first.

1. A Kubernetes cluster running at `localhost`. The Traefik Labs team often uses [k3d](#) for this purpose, which creates a local cluster in Docker containers. However, `k3d` comes bundled with the latest version of `k3s`, and `k3s` comes packaged with Traefik ver 1.7 and `metrics-server`. You'll want to disable both so that you can work with Prometheus and the latest version of Traefik:

```
k3d cluster create dev -p "8081:80@loadbalancer" --k3s-server-arg --disable
```

2. The `kubectl` command-line tool, configured to point to your cluster. (If you created your cluster using `K3d` and the instructions above, this will already be done for you.)
3. A recent version of the [Helm](#) package manager for Kubernetes.
4. The set of configuration files that accompany this article, which are [available on GitHub](#):

```
git clone https://github.com/traefik-tech-blog/traefik-sre-metrics/
```

You **do not** need to have Traefik 2.x preinstalled, as you'll do that in the next step.

The easiest way to deploy Traefik on Kubernetes is to use the official [Helm chart](#). Add Traefik to Helm's repositories using the below commands:

```
helm repo add traefik https://helm.traefik.io/traefik
helm repo update
```

Lastly, deploy the latest version of Traefik in the `kube-system` namespace. For this example, however, you'll want to ensure that Prometheus metrics are enabled in your cluster. This is done by passing Helm the `--metrics.prometheus=true` configuration flag, which you can do by applying the supplied `traefik-values.yaml` file when installing Traefik with Helm:

```
helm install traefik traefik/traefik -n kube-system -f ./traefik-values.yaml
```

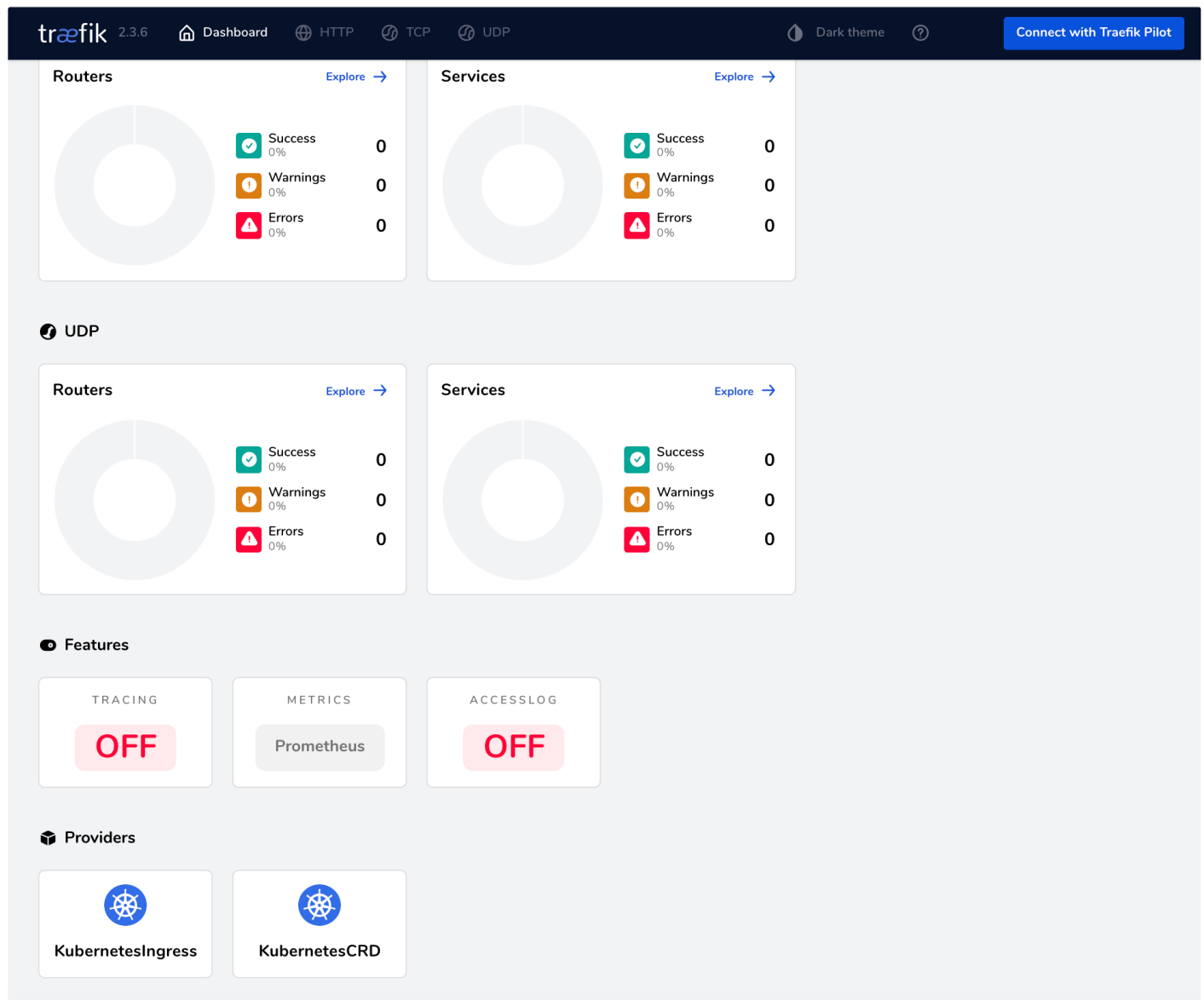
You should also create a `traefik-dashboard` service for the `traefik` endpoint, which Prometheus will use to monitor the Traefik metrics:

```
kubectl apply -f traefik-dashboard-service.yaml
```

The Traefik Dashboard is not exposed by default, but you can make it accessible using port forwarding:

```
kubectl port-forward service/traefik-dashboard 9000:9000 -n kube-system
```

With the Traefik Dashboard accessible from your web browser, you should now see that Prometheus metrics are enabled in the "Features" section of the dashboard, which you



Additionally, you can access the <http://localhost:9000/metrics> endpoint to see some generated metrics:

```
go_gc_duration_seconds{quantile="0.5"} 0.0003403
go_gc_duration_seconds{quantile="0.75"} 0.0005775
go_gc_duration_seconds{quantile="1"} 0.0017605
go_gc_duration_seconds_sum 0.023321
go_gc_duration_seconds_count 52
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 197
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.15.6"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.274876e+07
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 2.29043952e+08
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.514089e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 3.074456e+06
# HELP go_memstats_gc_cpu_fraction The fraction of this program's available CPU time used by the GC since the program started.
# TYPE go_memstats_gc_cpu_fraction gauge
go_memstats_gc_cpu_fraction 1.0996479662251051e-05
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 5.780808e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
```

Deploy Prometheus Stack

The Prometheus metrics stack consists of number of components. Deployment of all these components, along with the required configuration, is beyond the scope of this blog. Instead, you'll use [Prometheus Community Kubernetes Helm Charts](https://github.com/prometheus-community/helm-charts) to deploy the following components:

- Prometheus Metrics server
- Alert Manger
- Metrics Exporter
- Grafana

To add the repository:

```
helm repo add prometheus-community https://github.com/prometheus-community/helm-
helm repo update
```

```
helm search repo prometheus-community
```

From this list, you should install the `kube-prometheus-stack` chart, which will deploy the required components. (This process could take a few moments, so be patient.)

```
$ helm install prometheus-stack prometheus-community/kube-prometheus-stack
```

```
NAME: prometheus-stack
```

```
LAST DEPLOYED: Fri Jan 22 13:09:15 2021
```

```
NAMESPACE: default
```

```
STATUS: deployed
```

```
REVISION: 1
```

```
NOTES:
```

```
kube-prometheus-stack has been installed. Check its status by running:
```

```
kubectl --namespace default get pods -l "release=prometheus-stack"
```

```
visit https://github.com/prometheus-operator/kube-prometheus for instructions on
```

Configure Traefik Monitoring

The Prometheus custom resource definition (CRD) will be used to configure Traefik metrics. You also need to add a ServiceMonitor, which will be used to read the data.

```
# traefik-service-monitor.yaml
jobLabel: traefik-metrics
selector:
  matchLabels:
    app.kubernetes.io/instance: traefik
    app.kubernetes.io/name: traefik-dashboard
namespaceSelector:
  matchNames:
    - kube-system
```

Per the above configuration, Prometheus will look at the `/metrics` endpoint of the `traefik-dashboard` service. The `traefik-dashboard` service is created in the `kube-system` namespace, while the ServiceMonitor is deployed in the `default` namespace:

```
kubectl apply -f traefik-service-monitor.yaml
```

Lets now validate whether Prometheus has started scraping Traefik metrics. The service should be available in the Prometheus Dashboard. To enable it, forward the `9090` port to `localhost:9090` :

```
kubectl port-forward service/prometheus-stack-kube-prom-prometheus 9090:9090
```

You can now open the Service Discovery dashboard:

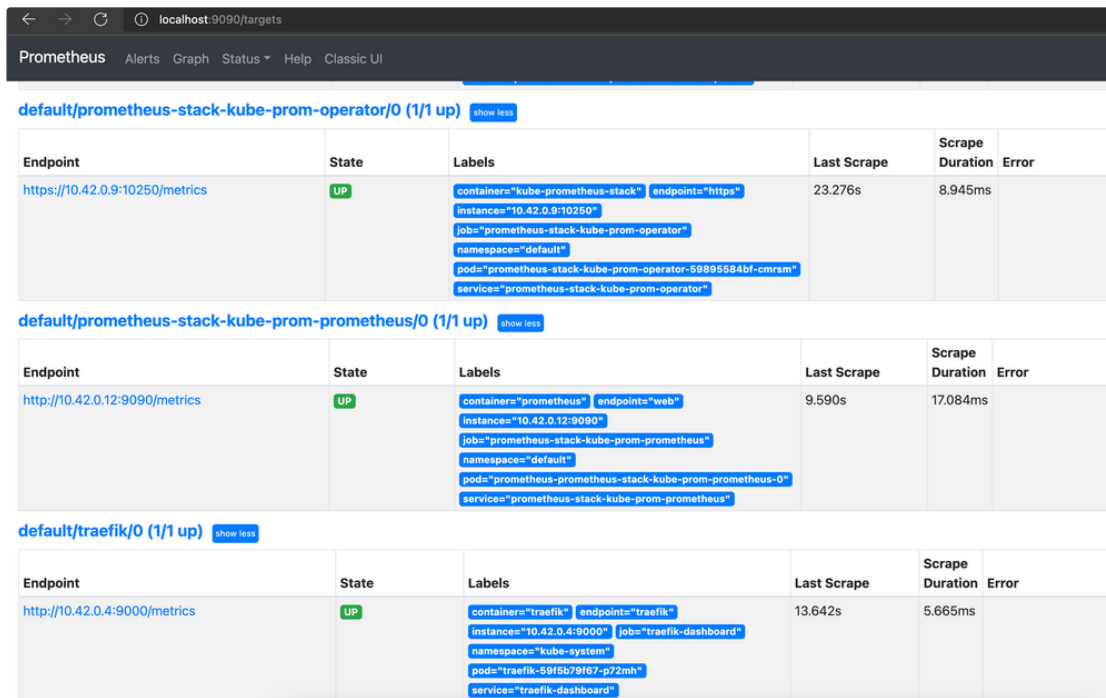
Service Discovery

- [default/prometheus-stack-kube-prom-alertmanager/0 \(1 / 18 active targets\)](#)
- [default/prometheus-stack-kube-prom-apiserver/0 \(1 / 18 active targets\)](#)
- [default/prometheus-stack-kube-prom-coredns/0 \(1 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-grafana/0 \(1 / 18 active targets\)](#)
- [default/prometheus-stack-kube-prom-kube-state-metrics/0 \(1 / 18 active targets\)](#)
- [default/prometheus-stack-kube-prom-kubelet/0 \(1 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-kubelet/1 \(1 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-kubelet/2 \(1 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-node-exporter/0 \(1 / 18 active targets\)](#)
- [default/prometheus-stack-kube-prom-operator/0 \(1 / 18 active targets\)](#)
- [default/prometheus-stack-kube-prom-prometheus/0 \(1 / 18 active targets\)](#)
- [default/traefik/0 \(1 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-kube-controller-manager/0 \(0 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-kube-etcd/0 \(0 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-kube-proxy/0 \(0 / 15 active targets\)](#)
- [default/prometheus-stack-kube-prom-kube-scheduler/0 \(0 / 15 active targets\)](#)

[default/prometheus-stack-kube-prom-alertmanager/0](#) [show more](#)

[default/prometheus-stack-kube-prom-apiserver/0](#) [show more](#)

This should show the `default/traefik` service. Details for the service should also be available under the `status > Targets` view.



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://10.42.0.9:10250/metrics	UP	containers="kube-prometheus-stack" endpoint="https" instances="10.42.0.9:10250" jobs="prometheus-stack-kube-prom-operator" namespace="default" pods="prometheus-stack-kube-prom-operator-59895584bf-cm8sm" services="prometheus-stack-kube-prom-operator"	23.276s	8.945ms	
http://10.42.0.12:9090/metrics	UP	containers="prometheus" endpoints="web" instances="10.42.0.12:9090" jobs="prometheus-stack-kube-prom-prometheus" namespace="default" pods="prometheus-prometheus-stack-kube-prom-prometheus-0" services="prometheus-stack-kube-prom-prometheus"	9.590s	17.084ms	
http://10.42.0.4:9000/metrics	UP	containers="traefik" endpoints="traefik" instances="10.42.0.4:9000" jobs="traefik-dashboard" namespace="kube-system" pods="traefik-59f6b79f67-p72mh" services="traefik-dashboard"	13.642s	5.665ms	

To see what else Prometheus can do you, can add a rule to raise alerts under matching conditions. Details about Prometheus rule expressions is beyond scope of the blog, but you can read more about it in the [official documentation](#).

```
rules:
- alert: TooManyRequest
  expr: avg(traefik_entrypoint_open_connections{job="traefik-dashboard",namespace="kube-system"}) > 5
  for: 1m
  labels:
    severity: critical
```

The above rule while will raise a `TooManyRequest` alert if there are more than 5 open requests for 1 minute. Go ahead and apply the rule:

```
kubectl apply -f traefik-rules.yaml
```

The Prometheus dashboard should show the newly created rule under `status > Rules` :

Prometheus Alerts Graph Status ▾ Help Classic UI				
Traefik			15.607s ago	1.393ms
Rule	State	Error	Last Evaluation	Evaluation Time
<code>alert: TooManyRequests</code> <code>expr: avg(traefik_entrypoint_open_connections{job="traefik-dashboard",namespace="kube-system"}) > 5</code> <code>for: 1m</code> <code>labels:</code> <code>severity: critical</code>	OK		15.607s ago	1.332ms

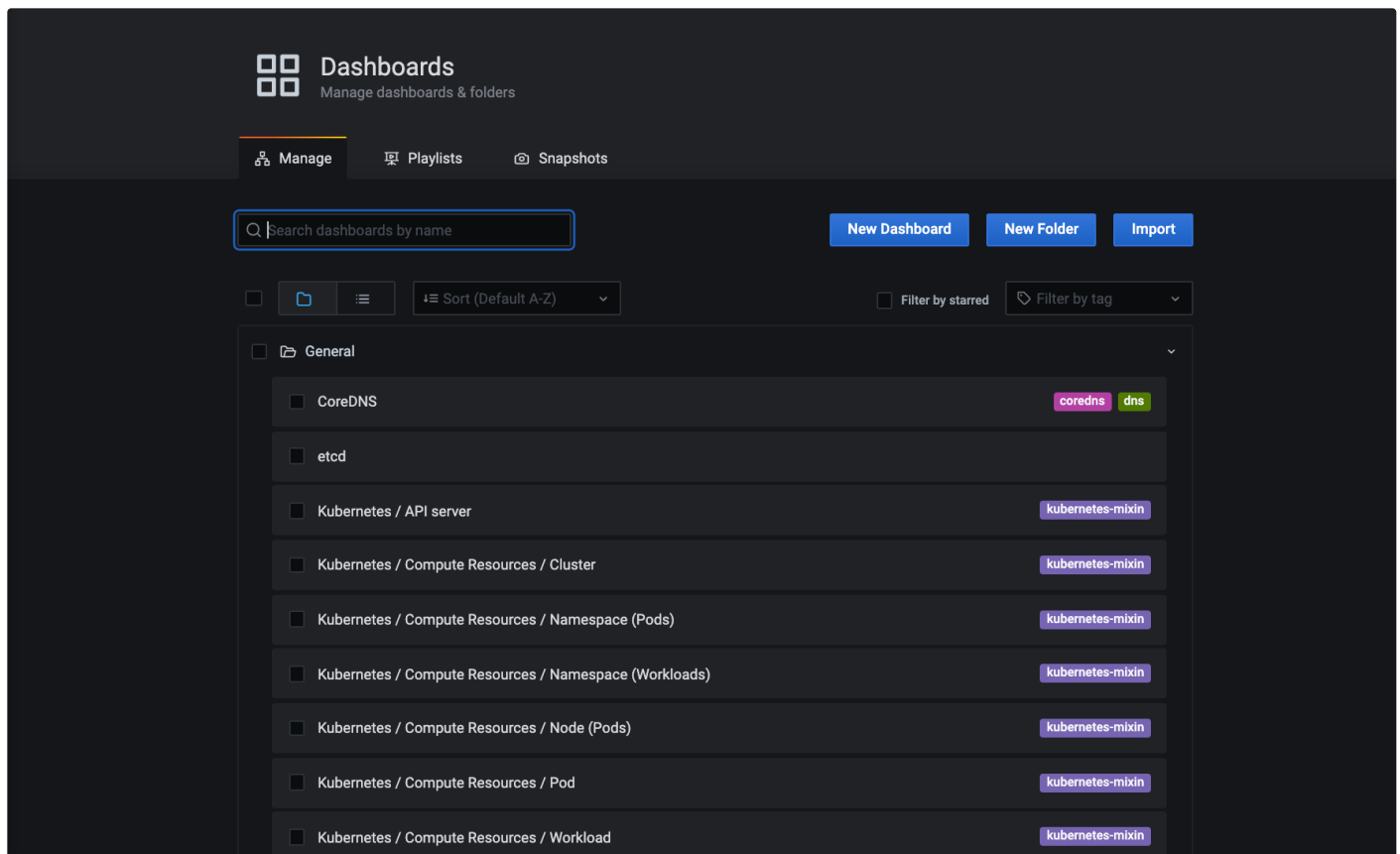
Grafana Charts

Previously you deployed Grafana using the `kube-prometheus-stack` Helm chart. Now you can configure a dashboard for Traefik metrics. But first, you'll need to forward port `80`

```
kubect1 port-forward service/prometheus-stack-grafana 10080:80
```

When you access the Grafana GUI at `http://localhost:10080`, it asks for a login and password. The default login username is `admin` and its password is `prom-operator`. The password can be read from the `prometheus-operator-grafana` Kubernetes secret.

Rather than building new Grafana dashboards from scratch, you can import them from Grafana's marketplace, which hosts community-created dashboards. Add a dashboard by navigating to `Dashboards > Manage` by clicking the four-square icon on the left navigation bar.



Click the **Import** button and input `11462` as the dashboard ID, which corresponds to the **Traefik 2** dashboard contributed by user `timoreymann`.

⬆ Upload JSON file

Import via grafana.com

11462

Load

Import via panel json

Load

After clicking **Load**, you should see the summary of the imported dashboard.

Import Dashboard from [Grafana.com](https://grafana.com)

Published by	timoreymann
Updated on	2019-12-24 01:17:59

Options

Name

Folder

Unique identifier (uid)

The unique identifier (uid) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The uid allows having consistent URL's for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

Prometheus

There is a dropdown at the bottom, select the `Prometheus` datasource and click **Import** to generate the following dashboard:



Deploy Application

Now that the cluster is working and metrics are being fed to Prometheus and Grafana, you'll need an application to monitor. For this, deploy the [HTTPBin](#) service, which provides many endpoints that can be used to generate different types of synthetic user traffic. The Service and the IngressRoute can be deployed using a single configuration file:

```
kubectl apply -f httpbin.yaml
```

The `httpbin` route will match the hostname for `httpbin.local` and forward the requests to the `httpBin` service. Lets lookup the service using `curl` command :

```
$ curl -I http://localhost:8081/ -H "host:httpbin.local"
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Length: 9593
Content-Type: text/html; charset=utf-8
```

As part of your setup, you attached `k3d LoadBalancer port 80 to 8081`. Thus, in the above command, the address to look up is `localhost:8081`.

Simulate User Traffic

You can hit HTTPBin with `ab` to generate some traffic. These requests will in turn generate metrics. Execute the following scripts:

```
ab -c 5 -n 10000 -m PATCH -H "host:httpbin.local" -H "accept: application/json"
ab -c 5 -n 10000 -m GET -H "host:httpbin.local" -H "accept: application/json" h
ab -c 5 -n 10000 -m POST -H "host:httpbin.local" -H "accept: application/json"
```

If you consult the Grafana dashboard after a while, you'll notice it shows rich information:



Some salient data points include:

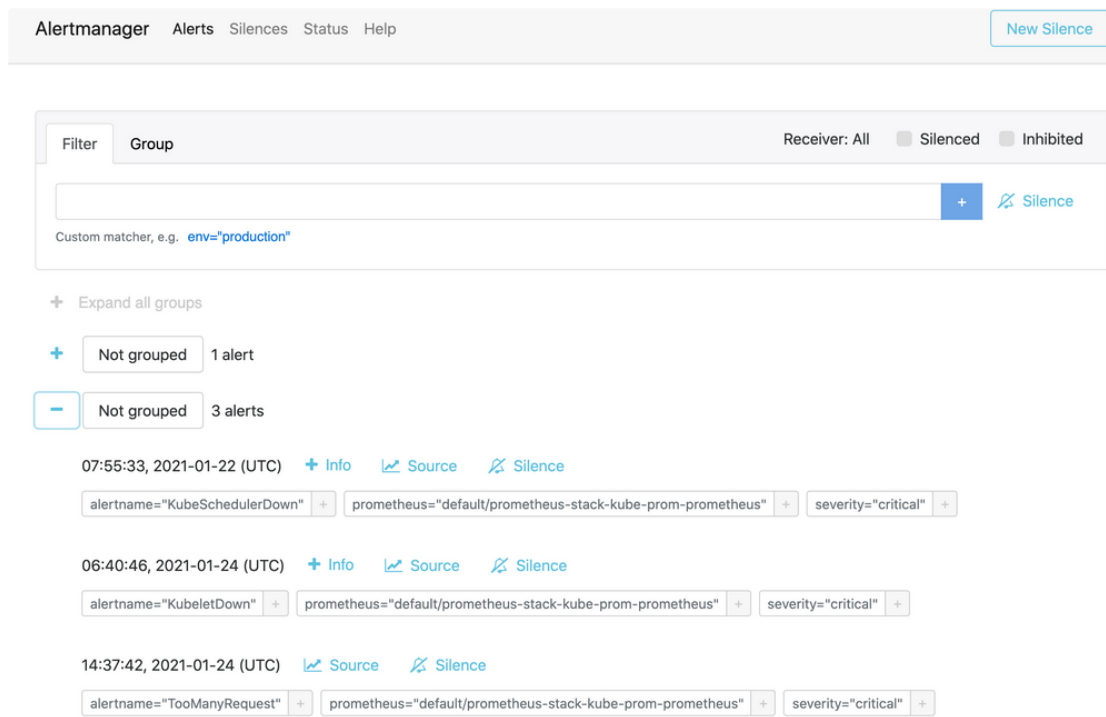
- Average response time
- Total requests
- Request counts based on HTTP method and service.

The above information is provided by the community chart, but you can also generate more charts for your observability needs.

Alerts

Finally, as you simulate application traffic, Prometheus may raise alerts. For the example, if you generate a lots of traffic, the alert for `TooManyRequest` that you created earlier will be shown on the Alertmanager dashboard.

```
$ kubectl port-forward service/prometheus-stack-kube-prom-alertmanager 9093:9093
Forwarding from 127.0.0.1:9093 -> 9093
```



The screenshot shows the Alertmanager web interface. At the top, there's a navigation bar with 'Alertmanager', 'Alerts', 'Silences', 'Status', and 'Help'. A 'New Silence' button is on the right. Below the navigation bar, there's a 'Filter' and 'Group' section. The 'Receiver' is set to 'All'. There are checkboxes for 'Silenced' and 'Inhibited'. A search bar is present with a '+' button and a 'Silence' link. Below the search bar, there's a text input field with the placeholder 'Custom matcher, e.g. env="production"'. Underneath, there's a '+ Expand all groups' link. The main content area shows a list of alerts. The first alert is 'KubeSchedulerDown' with a severity of 'critical', timestamped '07:55:33, 2021-01-22 (UTC)'. The second alert is 'KubeletDown' with a severity of 'critical', timestamped '06:40:46, 2021-01-24 (UTC)'. The third alert is 'TooManyRequest' with a severity of 'critical', timestamped '14:37:42, 2021-01-24 (UTC)'. Each alert has links for 'Info', 'Source', and 'Silence'.

Alertmanager Alerts Silences Status Help [New Silence](#)

Filter Group Receiver: All ☐ Silenced ☐ Inhibited

+ [Silence](#)

Custom matcher, e.g. `env="production"`

+ Expand all groups

+ Not grouped 1 alert

- Not grouped 3 alerts

07:55:33, 2021-01-22 (UTC) [Info](#) [Source](#) [Silence](#)

alertname="KubeSchedulerDown" + prometheus="default/prometheus-stack-kube-prom-prometheus" + severity="critical" +

06:40:46, 2021-01-24 (UTC) [Info](#) [Source](#) [Silence](#)

alertname="KubeletDown" + prometheus="default/prometheus-stack-kube-prom-prometheus" + severity="critical" +

14:37:42, 2021-01-24 (UTC) [Source](#) [Silence](#)

alertname="TooManyRequest" + prometheus="default/prometheus-stack-kube-prom-prometheus" + severity="critical" +

The more visibility you have into the services running on your Kubernetes clusters, the better-equipped you will be to take action in the event of anomalous performance. In this tutorial, you've seen how easy it is to connect Traefik to Prometheus and Grafana to create visualizations from Traefik metrics.

As you familiarize yourself with these tools, you'll be able to create unique dashboards that expose the data points that are most critical for your environment.

The next entry in this series on SRE techniques will focus on application tracing with Traefik and Jaeger. Until then, if you'd like to learn more about gaining visibility and control over your Traefik instances, check out [Traefik Pilot](#), our SaaS monitoring and management platform for Traefik.

1 reply



jbtruffault

10h

Dear ! Thank you for your tutorial ! I had some issues when I tried to access the traefik metrics even after following your instructions. If you have a couple of minutes, could you give me some advice ?

I described my problem here: [Can you help me getting traefik metrics ? \(version 2.2.8\)](#)

Regards

[Continue Discussion](#)

Related Posts

Log Aggregation for Traefik and Kubernetes with the Elastic Stack



Log Aggregation for Traefik and Kubernetes with the Elastic Stack



How To

How To

February 16, 2021

When combined with the set of open-source projects known as the Elastic Stack – including Elasticsearch, Kibana, and Filebeat, among others – Traefik becomes part of a rich set of tools for network log analysis and visualization.

[Read more](#)

Application Request Tracing with Traefik and Jaeger on Kubernetes



How to integrate Traefik with Jaeger, an open-source tracing application, to capture traces for user requests across the various components of a hypothetical application running on a Kubernetes cluster.

[Read more](#)



Six Ways to Kickstart Development with Kubernetes and Traefik



Neil McAllister • [Kubernetes](#) • March 9, 2021

You can start experimenting with Kubernetes and Traefik in minutes and in your choice of environment, which can even be the laptop in front of you.

[Read more](#)

Your email

☐ I agree to receive communications
from Traefik Labs

Products

[Traefik Proxy](#)

[Traefik Enterprise](#)

[Traefik Mesh](#)

[Traefik Pilot](#)

[Request a Demo](#)

[Pricing](#)

[Support Center](#)

Learn

[Blog](#)

[Resource Library](#)

[Success Stories](#)

[Docs](#)

[Community](#)

[Events](#)

Company

[About](#)

[Partners](#)

[Careers](#)

[Press](#)

[Contact Us](#)

Connect

[Forum](#)

[Github](#)

[Twitter](#)

[LinkedIn](#)

[Youtube](#)

