Alerting on Traffic-related Issues

In this lesson, we will discuss the issues related to the Traffic Key Metric.

WE'LL COVER THE FOLLOWING

Measuring traffic

So far, we measured the latency of our applications, and we created alerts that fire when certain thresholds based on request duration are reached. Those alerts are not based on the number of requests coming in (traffic), but on the percentage of slow requests. The *AppTooSlow* would fire even if only one single request enters an application, as long as the duration is above the threshold. For completeness, we need to start measuring traffic or, to be more precise, the number of requests sent to each application and the system as a whole. Through that, we can know if our system is under a lot of stress and make a decision on whether to scale our applications, add more workers, or apply some other solution to mitigate the problem. We might even choose to block part of the incoming traffic if the number of requests reaches abnormal numbers providing a clear indication that we are under Denial of Service (DoS) attack.

We'll start by creating a bit of traffic that we can use to visualize requests.

```
for i in {1..100}; do
    curl "http://$GD5_ADDR/demo/hello"
```

```
done


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

We sent a hundred requests to the `go-demo-5` application and opened the `Prometheus`'s graph screen.

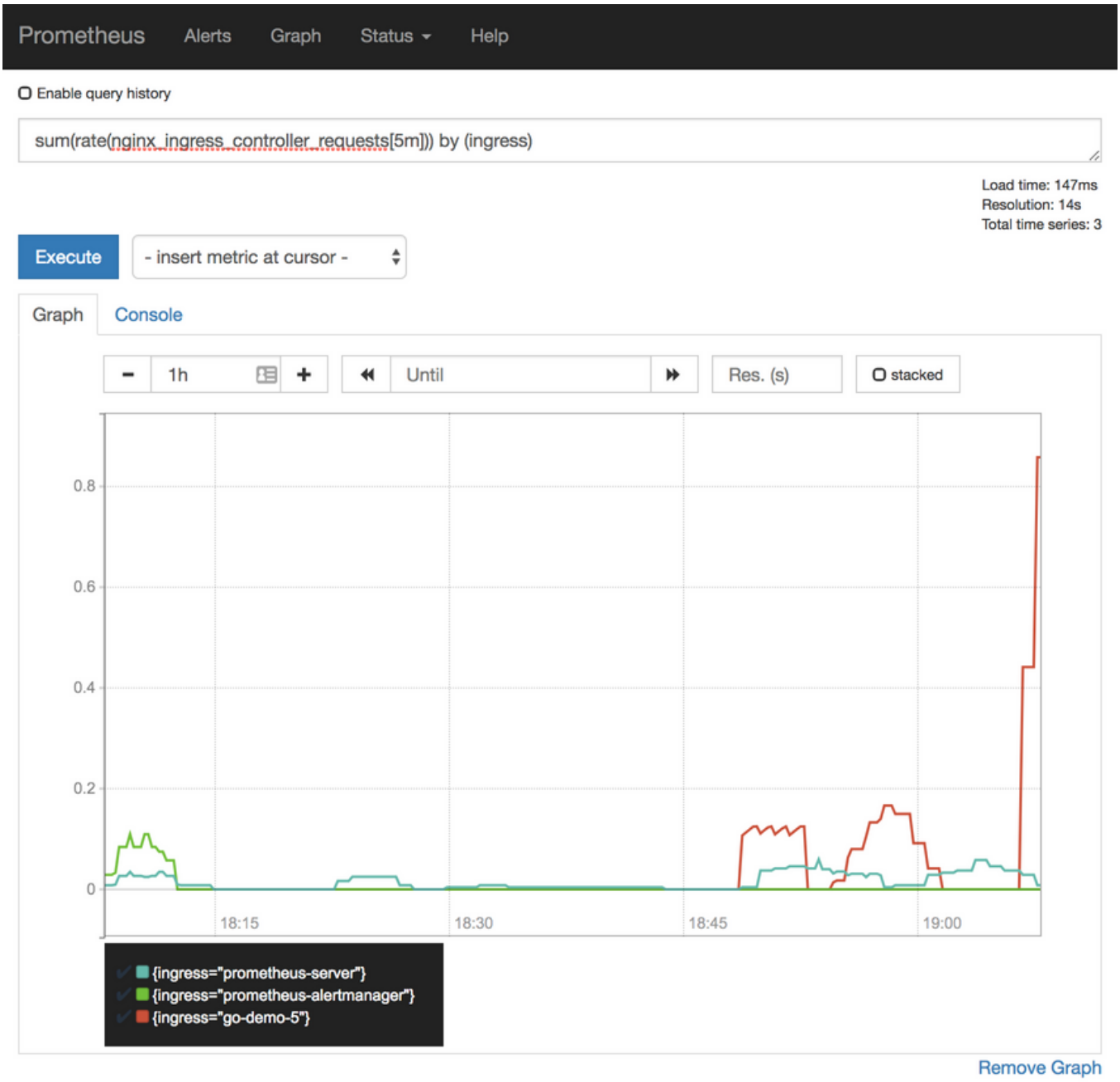
Retrieve the number of requests

We can retrieve the number of requests coming into the Ingress controller through the `nginx_ingress_controller_requests`. Since it is a counter, we can continue using the `rate` function combined with `sum`. Finally, we probably want to know the rate of requests grouped by the `ingress` label.

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

```
sum(rate(
  nginx_ingress_controller_requests[5m]
))
by (ingress)
```

We can see a spike on the right side of the graph. It shows the requests that went to the `go-demo-5` applications through the Ingress with the same name. In my case (screenshot below), the peak is close to one request per second (yours will be different).



Prometheus' graph screen with the rate of the number of requests

We are probably more interested in the number of requests per second per replica of an application, so our next task is to find a way to retrieve that data. Since `go-demo-5` is a Deployment, we can use `kube_deployment_status_replicas`.

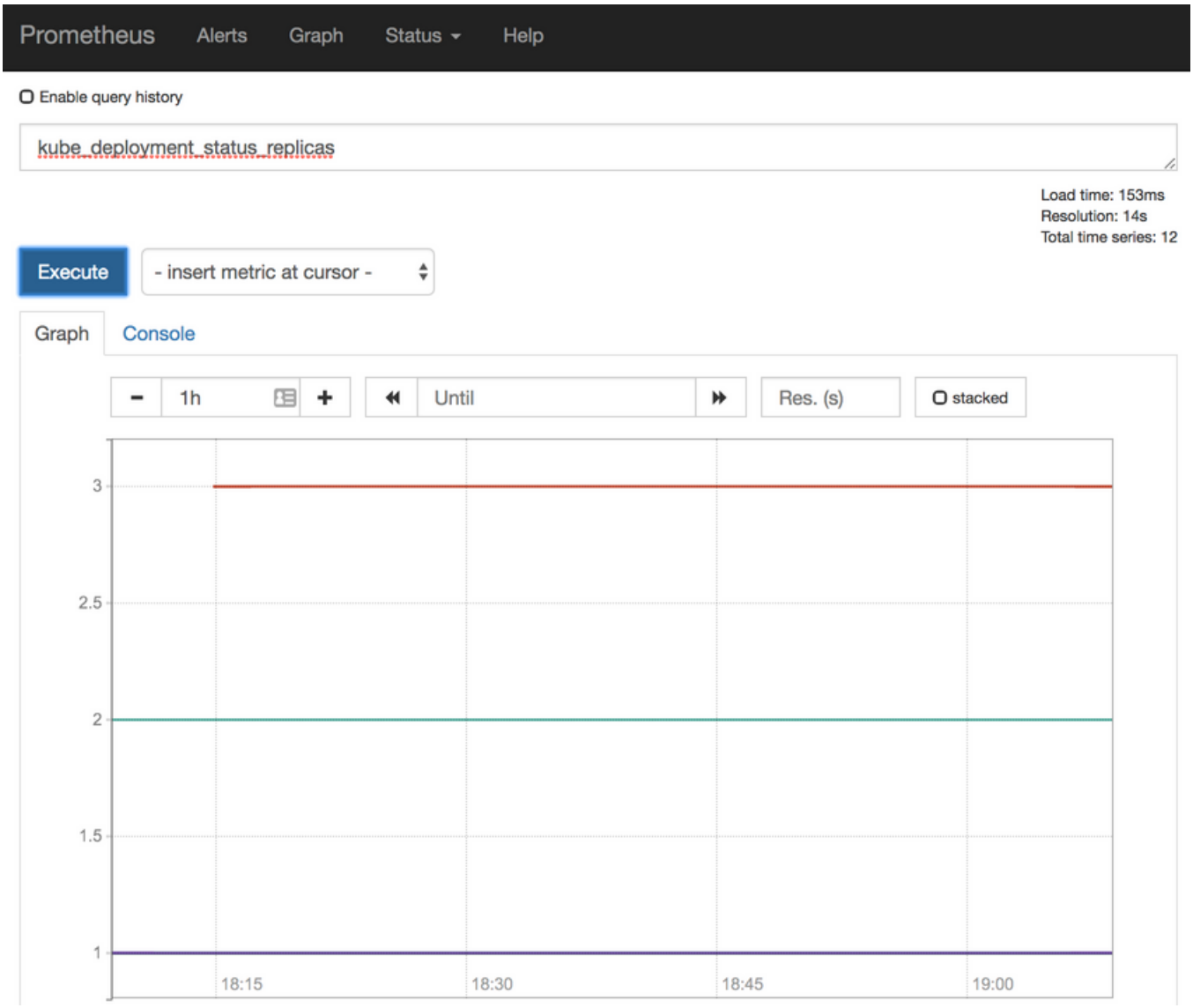
Retrieve requests per second per replica

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

```
kube_deployment_status_replicas
```

We can see the number of replicas of each Deployment in the system. The `go-`

`demo-5` application, in my case painted in red (screenshot below), has three replicas.



Prometheus' graph screen with the number of replicas of Deployments

Join the two metrics

Next, we should combine the two expressions to get the number of requests per second per replica. However, we are facing a problem. For two metrics to join, they need to have matching labels. Both the Deployment and the Ingress of `go-demo-5` have the same name; we can use that to our benefit, given that we can rename one of the labels. We'll do that with the help of the label_join function.

> 🔍 For each time-series in v, `label_join(v instant-vector, dst_label string, separator string, src_label_1 string, src_label_2 string, …)` joins all the values of all the `src_labels` using the separator and returns the time-series with the label `dst_label` containing the joined value.

Transform `kube_deployment_status_replicas` by adding `ingress` label

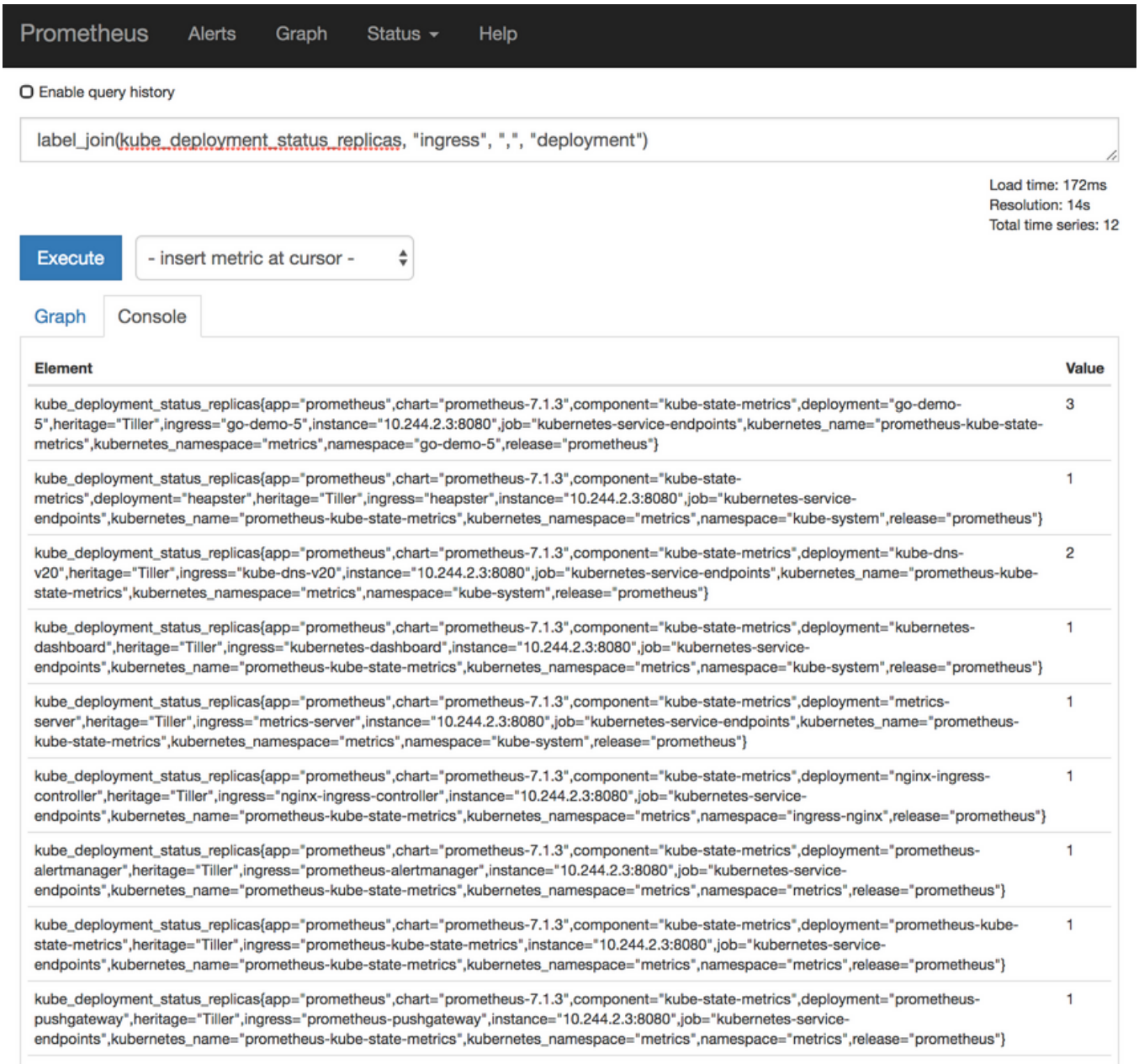
If the previous explanation of the `label_join` function was confusing, you're not alone. Instead, let's go through an example that will transform `kube_deployment_status_replicas` by adding `ingress` label that will contain values from the `deployment` label. If we are successful, we'll be able to combine the result with `nginx_ingress_controller_requests` since both will have the same matching labels (`ingress`).

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

```
label_join(
  kube_deployment_status_replicas,
  "ingress",
  ",",
  "deployment"
)
```

Since we are, this time, interested mostly in values of the labels, please switch to the *Console* view by clicking the tab.

As you can see from the output, each metric now contains an additional label `ingress` with the same value as `deployment`.



Prometheus' console view of Deployment replicas status and a new label ingress created from the deployment label

Q

We can retrieve the number of requests coming into the Ingress controller through the `nginx_ingress_controller_requests`.

Now we can combine the two metrics.

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

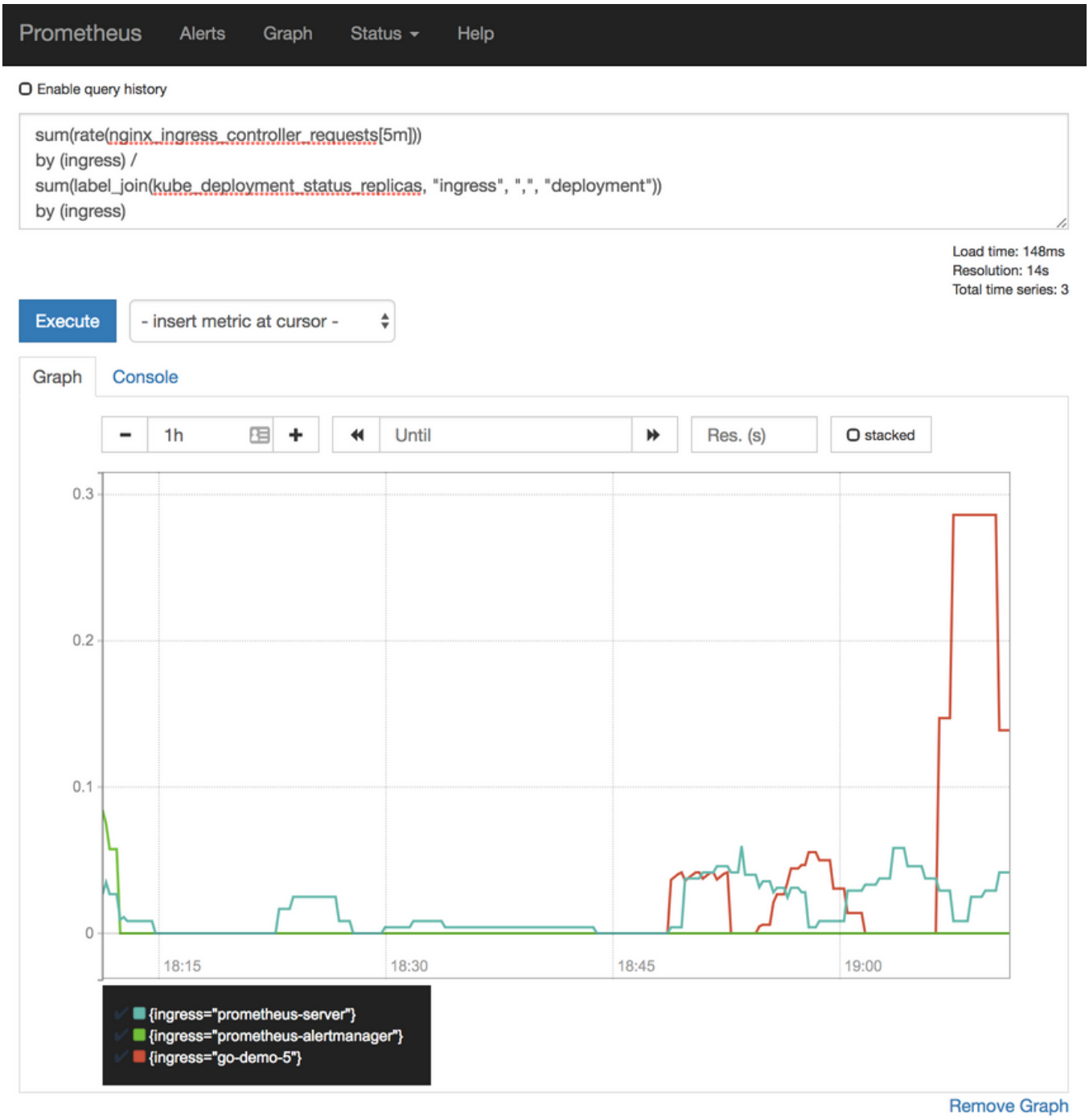
```
sum(rate(
  nginx_ingress_controller_requests[5m]
))
by (ingress) /
sum(label_join(
  kube_deployment_status_replicas,
  "ingress",
  ",",
  "deployment"
))
by (ingress)
```

Switch back to the *Graph* view.

We calculated the rate of the number of requests per application (`ingress`) and divided it with the total number of replicas per application (`ingress`). The end result is the rate of the number of requests per application (`ingress`) per replica.

It might be worth noting that we can not retrieve the number of requests for each specific replica, but rather the average number of requests per replica. This method should work given that Kubernetes networking, in most cases, performs round-robin that results in more or less the same amount of requests being sent to each replica.

All in all, now we know how many requests our replicas are receiving per second.



Prometheus' graph screen with the rate of requests divided by the number of Deployment replicas

Convert the expression into an alert

Now that we have learned how to write an expression to retrieve the rate of the number of requests per second per replica, we should convert it into an alert. So, let's take a look at the diff between the old and the new definition of `Prometheus`'s Chart values.

```
diff mon/prom-values-latency.yml \
    mon/prom-values-latency2.yml
```

The **output** is as follows.

The **output** is as follows.

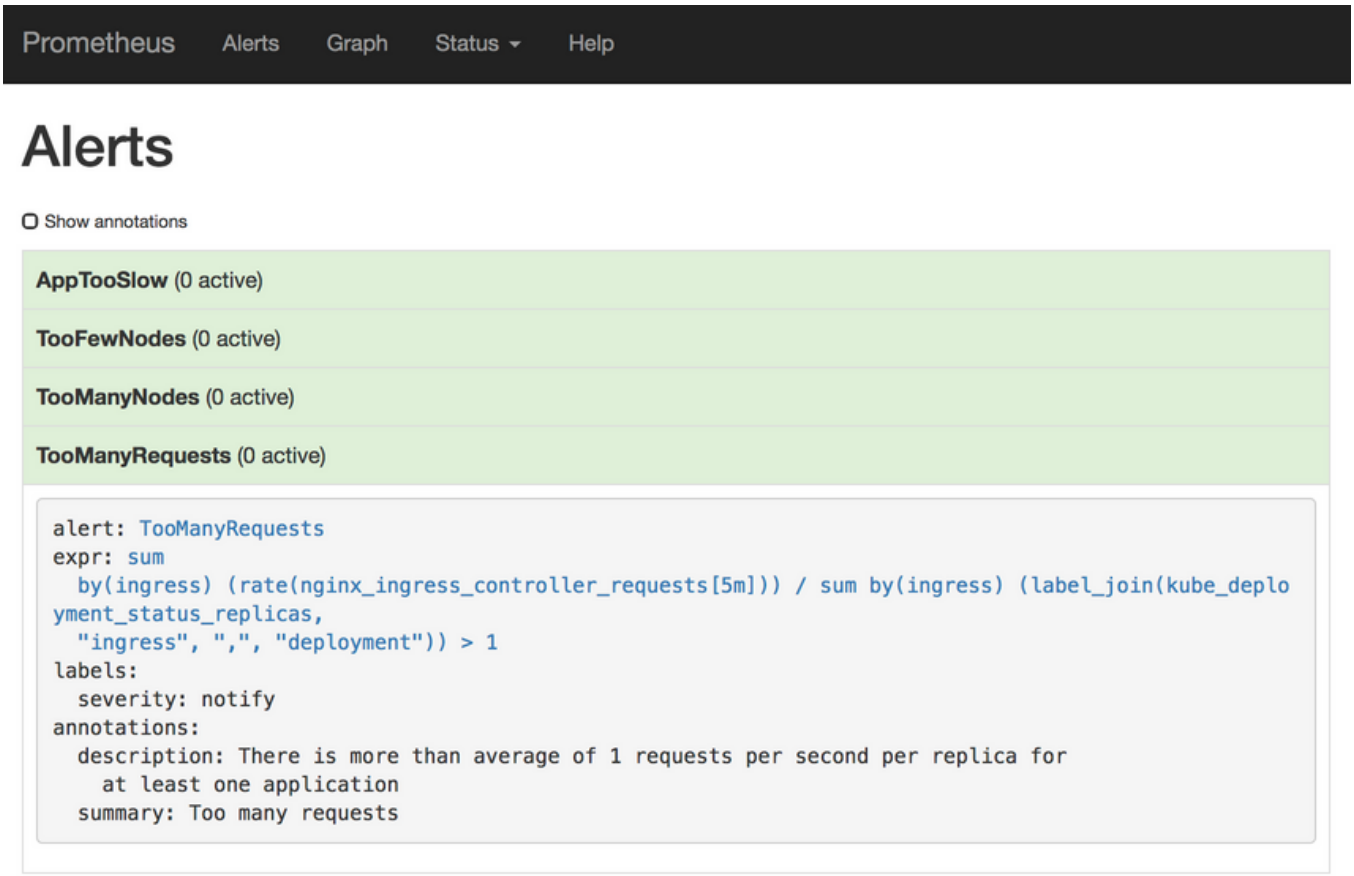
```
62a63,69
> - alert: TooManyRequests
>   expr: sum(rate(nginx_ingress_controller_requests[5m])) by (ingress) /
 sum(label_join(kube_deployment_status_replicas, "ingress", ",", "deployme
nt")) by (ingress) > 0.1
>   labels:
>     severity: notify
>   annotations:
>     summary: Too many requests
>     description: There is more than average of 1 requests per second pe
r replica for at least one application
```

We can see that the expression is almost the same as the one we used in the `Prometheus`'s graph screen. The only difference is the threshold which we set to 0.1. As a result, that alert should notify us whenever a replica receives more than a rate of `0.1` requests per second, calculated over the period of five minutes (`[5m]`). As you might have guessed, `0.1` requests per second is too low of a figure to use in production. However, it'll allow us to trigger the alert easily and see it in action.

Now, let's upgrade our Chart and open `Prometheus`'s alerts 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-latency2.yml

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

Please refresh the screen until the *TooManyRequests* alert appears.



Prometheus' alerts screen

Next, we'll generate some traffic so we can see that the alert is generated and sent through `Alertmanager` to Slack.

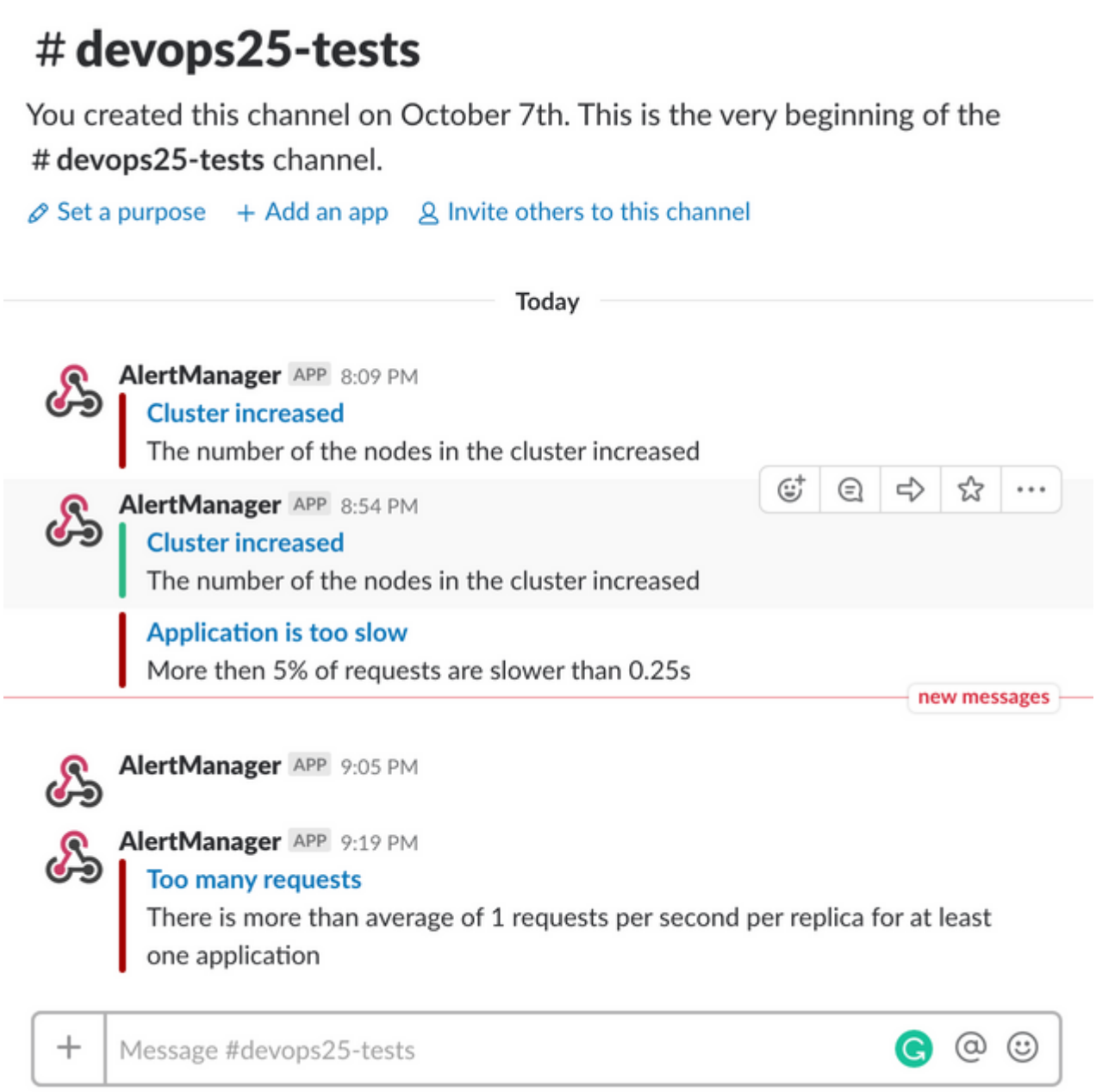
```
for i in {1..200}; do
    curl "http://$GD5_ADDR/demo/hello"
done

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

We sent two hundred requests and reopened the `Prometheus`'s alerts screen. Now we should refresh the screen until the *TooManyRequests* alert becomes red.

When `Prometheus` fired the alert, it was sent to `Alertmanager` and, from there, forwarded to Slack. Let's confirm that.

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

We can see the *Too many requests* notification, thus proving that the flow of this alert works.



Slack with alert messages

Next, we'll jump into errors-related metrics.