

Alerting on Latency-related Issues

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

WE'LL COVER THE FOLLOWING

- Measure latency
 - Get the duration of requests entering the system
 - Calculate the rate of requests
 - Percentage of requests
 - Limit the results to `go-demo-5` Ingress
 - Filtering metrics
 - Define an alert
 - Generate a few slow requests
 - Confirm Slack notification
 - Using regular expressions to exclude applications from alert
 - Specifying `ingress!~"prometheus.+|jenkins.+"` as filter

Measure latency

We'll use the `go-demo-5` application to measure latency, so our first step is to install it.

```
GD5_ADDR=go-demo-5.$LB_IP.nip.io

kubectl create namespace go-demo-5

helm install go-demo-5 \
  https://github.com/vfarcic/go-demo-5/releases/download/0.0.1/go-demo-5-0.0.1.tgz \
  --namespace go-demo-5 \
  --set ingress.host=$GD5_ADDR
```

We generated an address that we'll use as an Ingress entry-point, and we deployed the application using Helm. Now we should wait until it rolls out.

```
kubectl -n go-demo-5 \  
  rollout status \  
  deployment go-demo-5
```

Before we proceed, we'll check whether the application is indeed working correctly by sending an HTTP request.

```
curl "http://$GD5_ADDR/demo/hello"
```

The **output** should be the familiar `hello, world!` message.

Get the duration of requests entering the system

#

Now, let's see whether we can, for example, get the duration of requests entering the system through Ingress.

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


If you click on the - *insert metrics at cursor* - drop-down list, you'll be able to browse through all the available metrics. The one we're looking for is `nginx_ingress_controller_request_duration_seconds_bucket`. As its name implies, the metric comes from NGINX Ingress Controller, and provides request durations in seconds and grouped in buckets.

Please type the expression that follows and click the *Execute* button.

```
nginx_ingress_controller_request_duration_seconds_bucket
```

In this case, seeing the raw values might not be very useful, so please click the *Graph* tab.

You should see graphs, one for each Ingress. Each is increasing because the metric in question is a `counter`. Its value is growing with each request.

 A `Prometheus counter` is a cumulative metric whose value can only increase, or be reset to zero on restart.

Calculate the rate of requests

What we need is to calculate the rate of requests over a period of time. We'll accomplish that by combining `sum` and `rate` functions. The former should be self-explanatory.

 `Prometheus` 's rate function calculates the per-second average rate of increase of the time series in the range vector.

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

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

The resulting graph shows us the per-second rate of all the requests entering the system through Ingress. The rate is calculated based on five minutes intervals. If you hover one of the lines, you'll see the additional information like the value and the Ingress. The `by` statement allows us to group the results by `ingress`.

Still, the result by itself is not very useful, so let's redefine our requirement. We should be able to find out how many of the requests are slower than 0.25 seconds. We cannot do that directly. Instead, we can retrieve all those that are 0.25 second or faster.

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

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.25"
  }[5m]
))
by (ingress)
```

Percentage of requests

What we really want is to find the percentage of requests that fall into 0.25 seconds bucket. To accomplish that, we'll get the rate of the requests faster

than or equal to 0.25 seconds and divide the result with the rate of all the requests.

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

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.25"
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_request_duration_seconds_count[5m]
))
by (ingress)
```

Since we did not yet generate much traffic, you probably won't see much in the graph beyond occasional interaction with `Prometheus` and `Alertmanager` and a single request we sent to `go-demo-5`. Nevertheless, the few lines you can see display the percentage of the requests that responded within 0.25 seconds.

Limit the results to `go-demo-5` Ingress

For now, we are interested only in `go-demo-5` requests, so we'll refine the expression further to limit the results only to `go-demo-5` Ingress.

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

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.25",
    ingress="go-demo-5"
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_request_duration_seconds_count{
    ingress="go-demo-5"
  }[5m]
))
by (ingress)
```

The graph should be almost empty since we sent only one request. Or, maybe you received the *no datapoints found* message. It's time to generate some traffic.

```
for i in {1..30}; do
  DELAY=$(( $RANDOM % 1000 ))
  curl "http://$GD5_ADDR/demo/hello?delay=$DELAY"
done
```

We sent thirty requests to `go-demo-5`. The application has a “hidden” feature to delay response to a request. Given that we want to generate traffic with random response time, we used the `DELAY` variable with a random value of up to a thousand milliseconds. Now we can re-run the same query and see whether we can get some more meaningful data.

Please wait for a while until data from new requests are gathered, then type the expression that follows (in `Prometheus`), and press the *Execute* button.

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.25",
    ingress="go-demo-5"
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_request_duration_seconds_count{
    ingress="go-demo-5"
  }[5m]
))
by (ingress)
```

This time, we can see the emergence of a new line. In my case (screenshot below), around twenty-five percent of requests have durations that are within 0.25 seconds. Or, to put it into different words, around a quarter of the requests are slower than expected.

☐ Enable query history

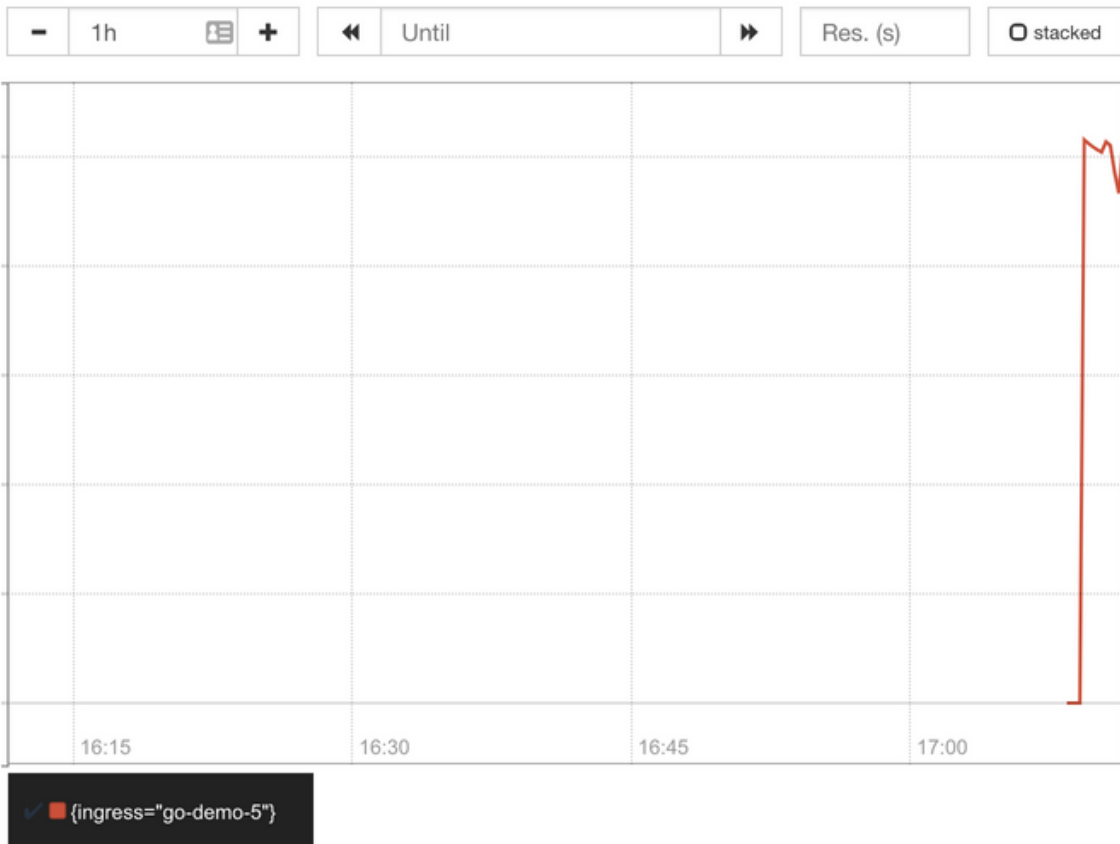
```
sum(rate(nginx_ingress_controller_request_duration_seconds_bucket{le="0.25", ingress="go-demo-5"}[5m]))  
by (ingress) /  
sum(rate(nginx_ingress_controller_request_duration_seconds_count{ingress="go-demo-5"}[5m]))  
by (ingress)
```

Load time: 409ms
Resolution: 14s
Total time series: 1

Execute

- insert metric at cursor -

Graph Console

[Remove Graph](#)

Prometheus' graph screen with the percentage of requests with 0.25 seconds duration

Q

Ingress ' rate function calculates the per-second average rate of increase of the time series in the range vector.

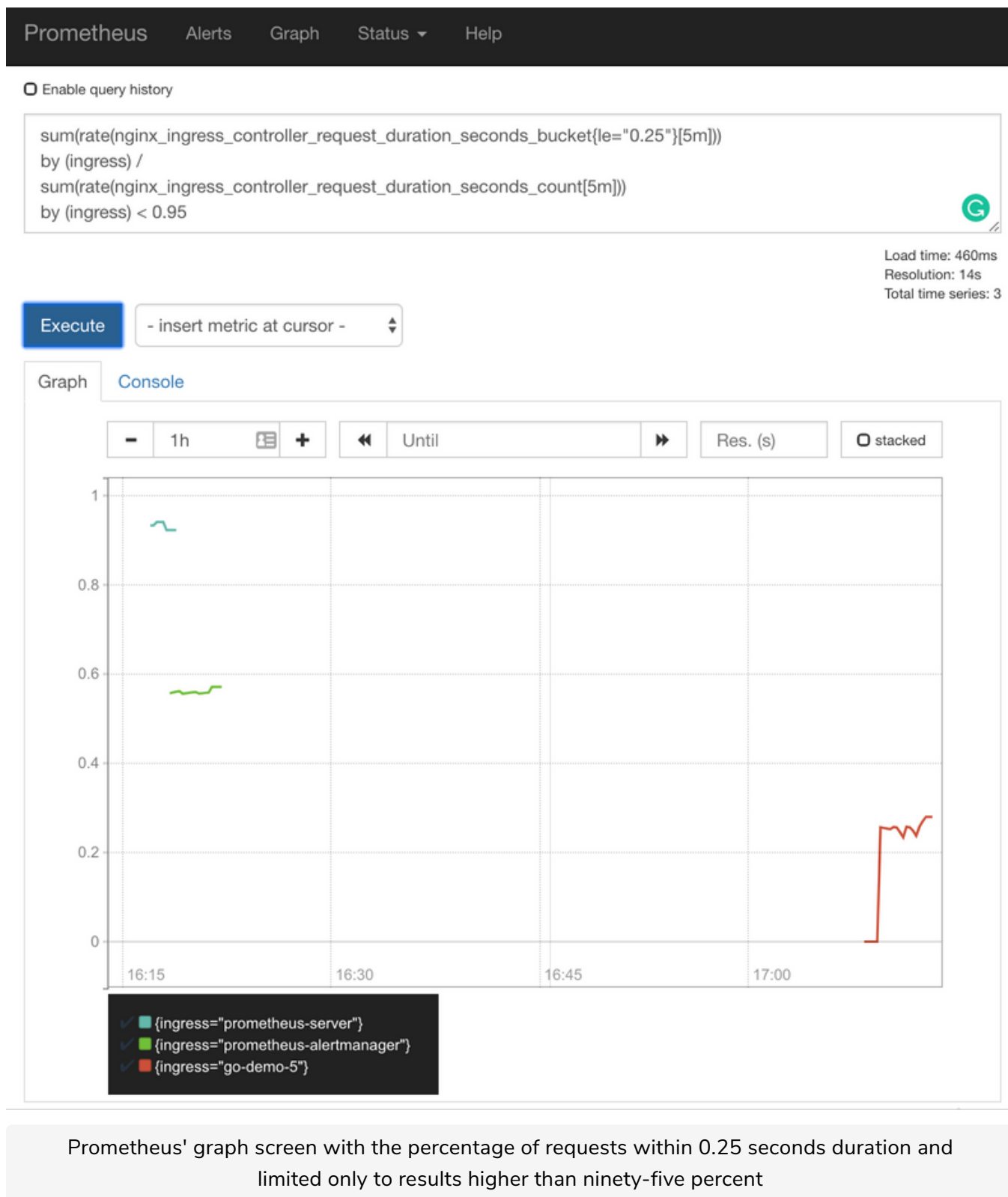
Filtering metrics

Filtering metrics for a specific application (Ingress) is useful when we know that there is a problem and we want to dig further into it. However, we still need an alert that will tell us that there is an issue. For that, we'll execute a similar query, but this time without limiting the results to a specific application (Ingress). We'll also have to define a condition that will fire the alert, so we'll set the threshold to ninety-five percent (0.95). Without such a threshold, we'd get a notification every time a single request is slow. As a result, we'd get swarmed with alarms and would probably start ignoring them soon afterward. After all, no system is in danger if a single request is slow, but only if a considerable number of them are. In our case, that is five percent of slow requests or, to be more precise, less than ninety-five percent of fast requests.

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

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.25"
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_request_duration_seconds_count[5m]
))
by (ingress) < 0.95
```

We can see occasional cases when less than ninety-five percent of requests are within 0.25 seconds. In my case (screenshot below), we can see that `Prometheus`, `Alertmanager`, and `go-demo-5` are occasionally slow.



Define an alert

The only thing missing is to define an alert based on the previous expression. As a result, we should get a notification whenever less than ninety-five percent of requests have a duration of less than 0.25 seconds.

I prepared an updated set of **Prometheus**'s Chart values, so let's take a look at the differences when compared with the one we're using currently.


```
diff mon/prom-values-nodes-am.yml \
    mon/prom-values-latency.yml
```

The **output** is as follows.

```
53a54,62
> - name: latency
>   rules:
>     - alert: AppTooSlow
>       expr: sum(rate(nginx_ingress_controller_request_duration_seconds_bucket{le="0.25"}[5m])) by (ingress) / sum(rate(nginx_ingress_controller_request_duration_seconds_count[5m])) by (ingress) < 0.95
>       labels:
>         severity: notify
>       annotations:
>         summary: Application is too slow
>         description: More than 5% of requests are slower than 0.25s
57c66
<     expr: count(kube_node_info) > 0
---
>     expr: count(kube_node_info) > 3
```

We added a new alert **AppTooSlow**. It'll trigger if the percentage of requests with a duration of 0.25 seconds or less is smaller than ninety-five percent (**0.95**).

We also reverted the threshold of the **TooManyNodes** back to its original value of **3**.

Next, we'll update the **Prometheus**'s Chart with the new values and open the alerts screen to confirm whether the new alert was indeed added.

```
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-latency.yml

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

If the *AppTooSlow* alert is still not available. Please wait a few moments and refresh the screen.

Prometheus Alerts Graph Status ▾ Help

Alerts

☐ Show annotations

AppTooSlow (0 active)

```
alert: AppTooSlow
expr: sum
  by(ingress) (rate(nginx_ingress_controller_request_duration_seconds_bucket{le="0.25"}[5m]))
  / sum by(ingress) (rate(nginx_ingress_controller_request_duration_seconds_count[5m]))
  < 0.95
labels:
  severity: notify
annotations:
  description: More then 5% of requests are slower than 0.25s
  summary: Application is too slow
```

TooFewNodes (0 active)

TooManyNodes (0 active)

Prometheus' alerts screen

Generate a few slow requests

The newly added alert is (probably) green (not triggering). We need to generate a few slow requests to see it in action.

Please execute the command that follows to send thirty requests with a random response time of up to ten thousand milliseconds (ten seconds).

```
for i in {1..30}; do
  DELAY=$(( $RANDOM % 10000 ))
  curl "http://$GD5_ADDR/demo/hello?delay=$DELAY"
done
```

It'll take a few moments until **Prometheus** scrapes new metrics and for the alert to detect that the threshold is reached. After a while, we can open the alerts screen again and check whether the alert is indeed firing.

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

We can see that the state of the alert is *firing*. If that's not your case, please wait a while longer and refresh the screen. In my case (screenshot below), the value is 0.125, meaning that only 12.5 percent of requests have a duration of 0.25 seconds or less.

🔍 There might be two or more active alerts inside *AppTooSlow* if `prometheus-server`, `prometheus-alertmanager`, or some other application is responding slow.

Prometheus Alerts Graph Status ▾ Help

Alerts

🔍 Show annotations

AppTooSlow (1 active)

```
alert: AppTooSlow
expr: sum
  by(ingress) (rate(nginx_ingress_controller_request_duration_seconds_bucket{le="0.25"}[5m]))
  / sum by(ingress) (rate(nginx_ingress_controller_request_duration_seconds_count[5m]))
  < 0.95
labels:
  severity: notify
annotations:
  description: More then 5% of requests are slower than 0.25s
  summary: Application is too slow
```

Labels	State	Active Since	Value
<code>alertname="AppTooSlow"</code> <code>ingress="go-demo-5"</code> <code>severity="notify"</code>	FIRING	2018-10-09 18:55:04.914947075 +0000 UTC	0.125

TooFewNodes (0 active)

TooManyNodes (0 active)

Prometheus' alerts screen with one alert firing

Confirm Slack notification

The alert is red meaning that `Prometheus` sent it to `Alertmanager` which, in turn, forwarded it to Slack. Let's confirm that.

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

As you can see (screenshot below), we received two notification. Since we reverted the threshold of the *TooManyNodes* alert back to greater than three

reverted the threshold of the *TooManyNodes* alert back to greater than three nodes, and our cluster has less, **Prometheus** sent a notification to **Alertmanager** that the problem is resolved. As a result, we got a new notification in Slack. This time, the color of the message is green.

Further on, a new red message appeared indicating that an *application is too slow*.

devops25-tests

You created this channel on October 7th. This is the very beginning of the # devops25-tests channel.

[Set a purpose](#) + [Add an app](#) [Invite others to this channel](#)

Today



AlertManager APP 8:09 PM

Cluster increased

The number of the nodes in the cluster increased



AlertManager APP 8:54 PM

Cluster increased

The number of the nodes in the cluster increased

Application is too slow

More then 5% of requests are slower than 0.25s



Message #devops25-tests



Slack with alerts firing (red) and resolved (green) messages

Using regular expressions to exclude applications from alert

We often cannot rely on a single rule that will fit all the applications.

Prometheus and, for example, **Jenkins** would be a good candidate for internal applications which we cannot expect to have less than five percent of response times above 0.25 seconds. So, we might want to further filter the alerts. We can use any number of labels for that. To keep it simple, we'll continue leveraging the **ingress** label but, this time, we'll use regular expressions to exclude some applications (Ingresses) from the alert.

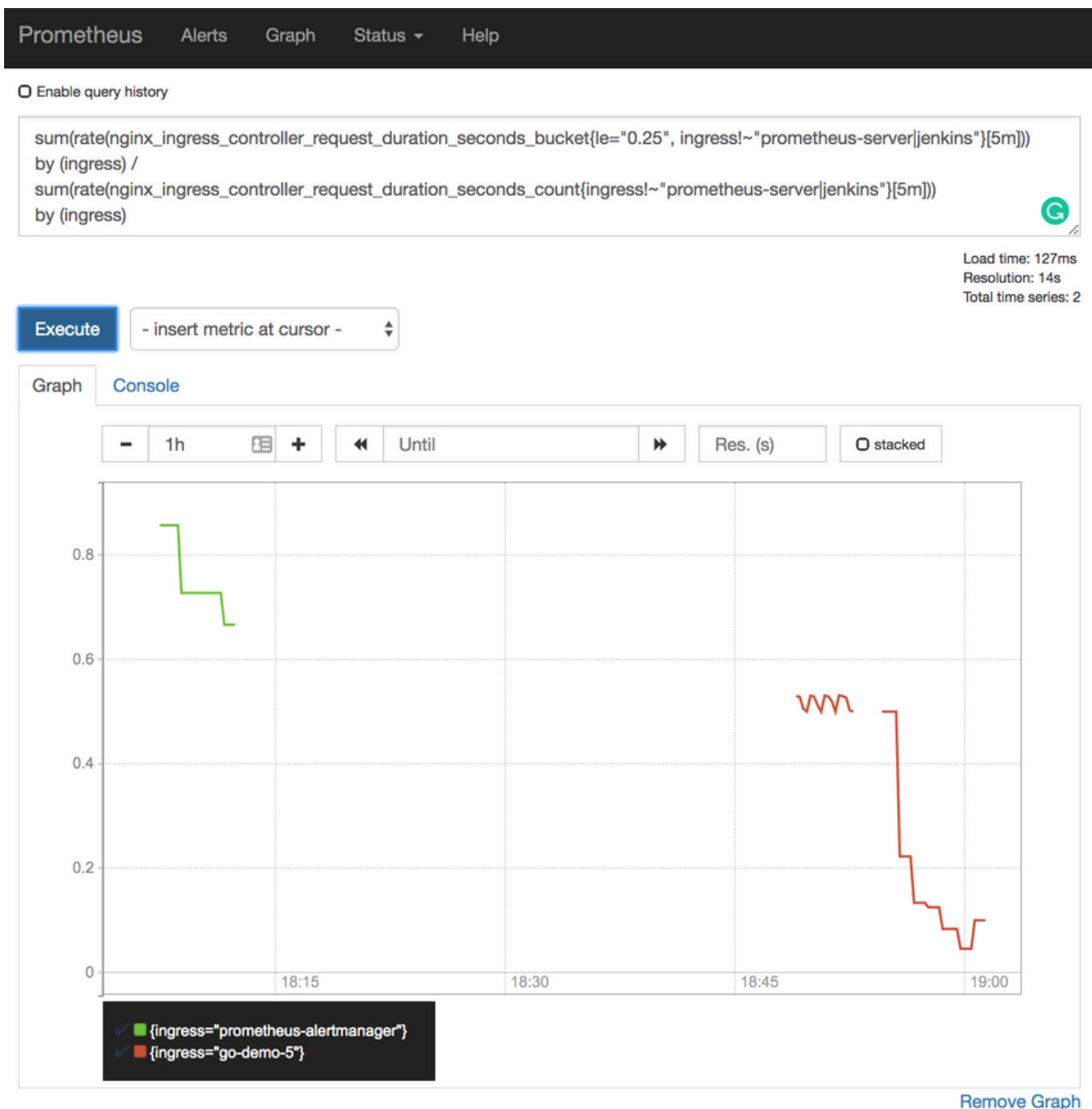
Let's open the graph screen one more time.

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

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

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.25",
    ingress!~"prometheus-server|jenkins"
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_request_duration_seconds_count{
    ingress!~"prometheus-server|jenkins"
  }[5m]
))
by (ingress)
```

The addition to the previous query is the `ingress!~"prometheus-server|jenkins"` filter. The `!~` is used to select metrics with labels that do NOT regex match the `prometheus-server|jenkins` string. Since `|` is equivalent to the `or` statement, we can translate that filter as "everything that is NOT `prometheus-server` or is NOT `jenkins`". We do not have **Jenkins** in our cluster. I just wanted to show you a way to exclude multiple values.



Prometheus graph screen with the percentage of requests with 0.25 seconds duration and the results excluding prometheus-server and jenkins

Specifying `ingress!~"prometheus.+|jenkins.+"` as filter `#`

We could have complicated it a bit more and specified

`ingress!~"prometheus.+|jenkins.+"` as the filter. In that case, it would exclude all Ingresses with the name that starts with `prometheus` and `jenkins`. The key is in the `.+` addition that, in RegEx, matches one or more entries (`+`) of any character (`.`).

We won't go into an explanation of RegEx syntax. I expect you to be already familiar with it. If you're not, you might want to Google it or to visit [Regular expression Wiki page](#)

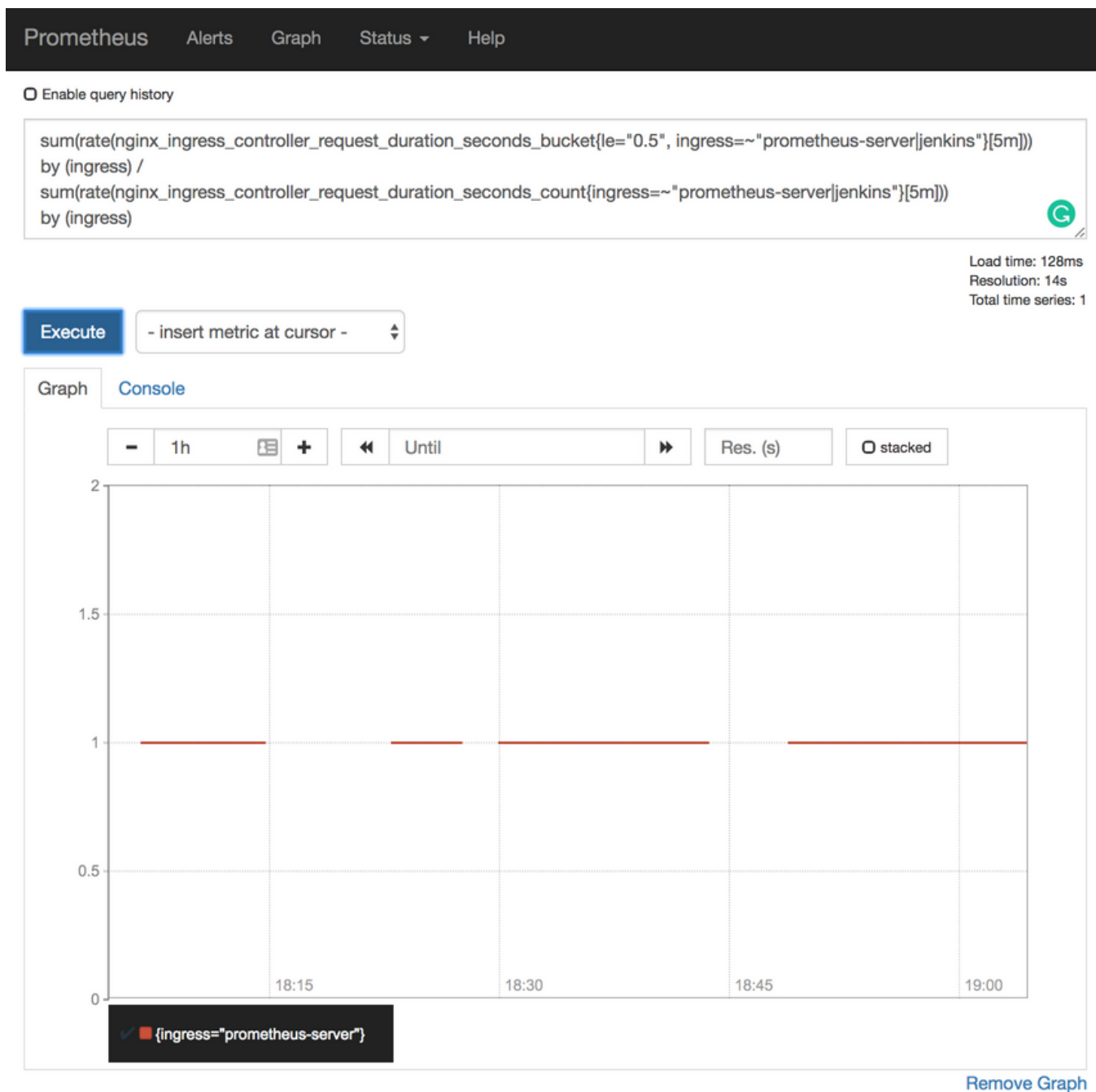
The previous expression retrieves only the results that are NOT `prometheus-server` and `jenkins`. We would probably need to create another one that includes only those two.

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

```
sum(rate(
  nginx_ingress_controller_request_duration_seconds_bucket{
    le="0.5",
    ingress=~"prometheus-server|jenkins"
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_request_duration_seconds_count{
    ingress=~"prometheus-server|jenkins"
  }[5m]
))
by (ingress)
```

The only difference, when compared with the previous expression, is that this time we used the `=~` operator. It selects labels that regex-match the provided string. Also, the bucket (`le`) is now set to `0.5` seconds, given that both applications might need more time to respond and we are OK with that.

In my case, the graph shows `prometheus-server` as having one hundred percent requests with durations within 0.5 seconds (in your case that might not be true).



Prometheus graph screen with the percentage of requests with 0.5 seconds duration and the results including only prometheus-server and jenkins

The few latency examples should be enough to get you going with that type of metrics, so we'll move to traffic.