Alerting on Error-related Issues

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

WE'LL COVER THE FOLLOWING

- Monitor the rate of errors compared to the total number of requests
  - Retrieve and separate requests from their statuses
  - Using `/demo/random-error` endpoint to generate random error responses
  - Write an expression to retrieve error rate

Monitor the rate of errors compared to the total number of requests

We should always be aware of whether our applications or the system is producing errors. However, we cannot start panicking at the first occurrence of an error since that would generate too many notifications that we'd likely end up ignoring. Errors happen often, and many are caused by issues that are fixed automatically or are due to circumstances that are out of our control. If we are to perform an action on every error, we'd need an army of people working 24/7 only on fixing issues that often do not need to be fixed. As an example, entering into a "panic" mode because there is a single response with code in 500 range would almost certainly produce a permanent crisis. Instead, we should monitor the rate of errors compared to the total number of requests and react only if it passes a certain threshold. After all, if an error persists, that rate will undoubtedly increase. On the other hand, if it continues being low, it means that the issue was fixed automatically by the system (e.g., Kubernetes rescheduled the Pods from the failed node) or that it was an isolated case that does not repeat.

Retrieve and separate requests from their statuses

Our next mission is to retrieve requests and separate them from their

statuses. If we can do that, we should be able to calculate the rate of errors.

We'll start by generating a bit of traffic.

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

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

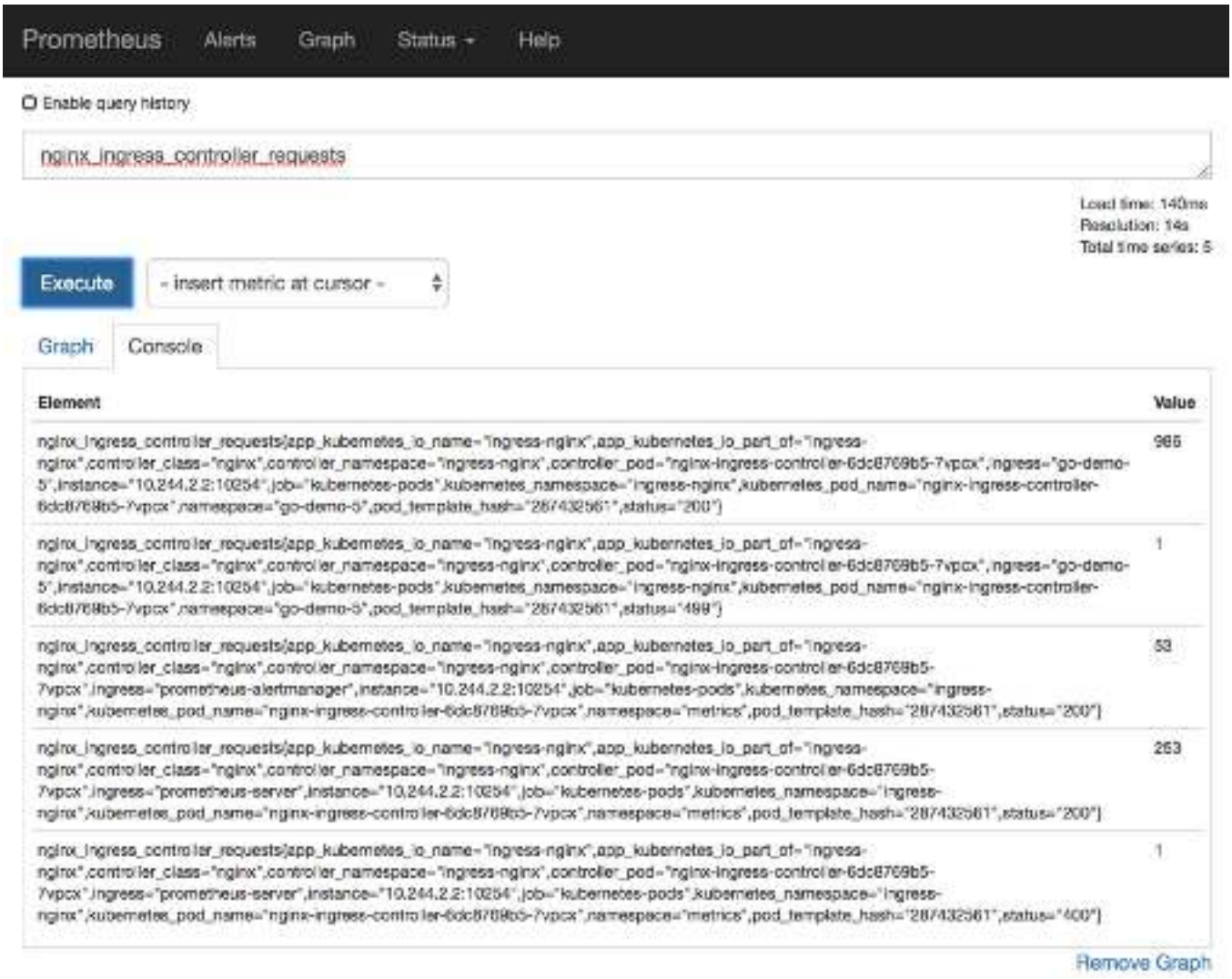
We sent a hundred requests and opened the `Prometheus`'s graph screen.

Let's see whether the `nginx_ingress_controller_requests` metric we used previously provides the statuses of the requests.

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

```
nginx_ingress_controller_requests
```

We can see all the data recently scraped by `Prometheus`. If we pay closer attention to the labels, we can see that, among others, there is `status`. We can use it to calculate the percentage of those with errors (e.g., 500 range) based on the total number of requests. We already saw that we can use the `ingress` label to separate calculations per application, assuming that we are interested only in those that are public-facing.



Prometheus' console view with requests entering through Ingress

Using `/demo/random-error` endpoint to generate random error responses

The `go-demo-5` app has a special endpoint `/demo/random-error` that will generate random error responses. Approximately, one out of ten requests to that address will produce an error. We can use that to test our expressions.

```
for i in {1..100}; do
  curl "http://$GD5_ADDR/demo/random-error"
done
```

We sent a hundred requests to the `/demo/random-error` endpoint and approximately ten percent of those responded with errors (HTTP status code `500`).

Write an expression to retrieve error rate

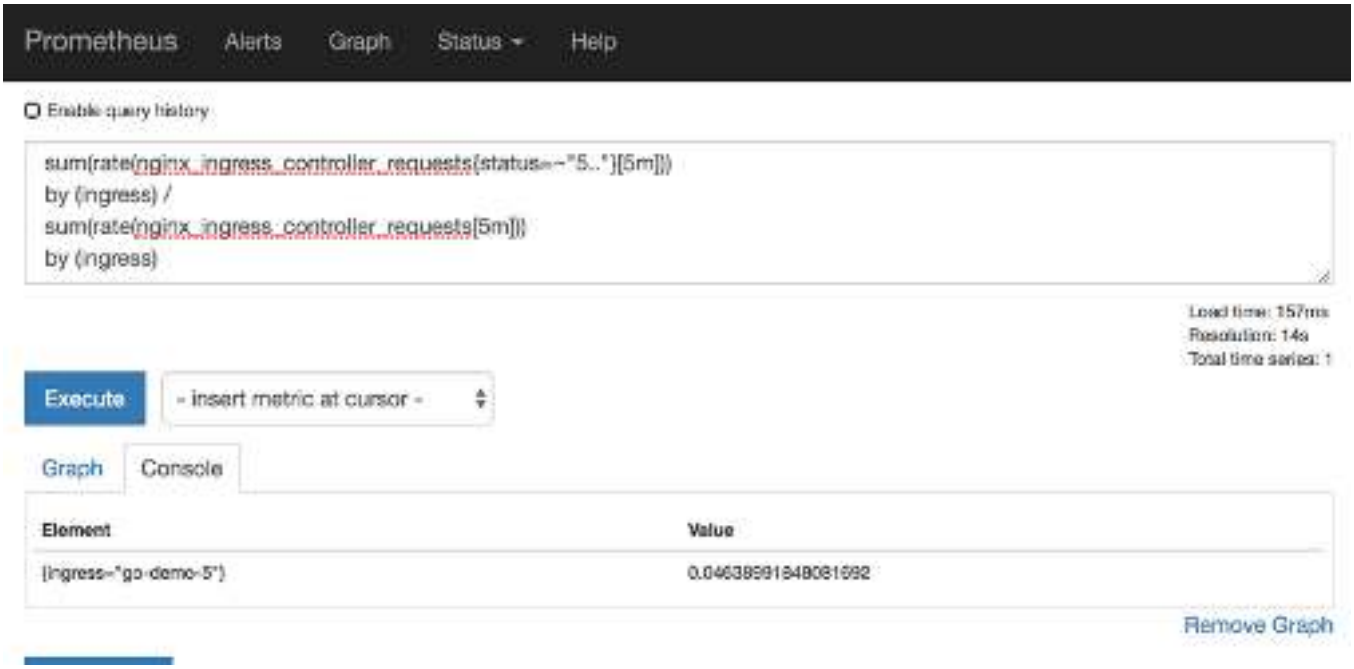
Next, we'll have to wait for a few moments for `Prometheus` to scrape the new batch of metrics. After that, we can open the graph screen and try to write an expression that will retrieve the error rate of our applications.

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

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

```
sum(rate(
  nginx_ingress_controller_requests{
    status=~"5.."
  }[5m]
))
by (ingress) /
sum(rate(
  nginx_ingress_controller_requests[5m]
))
by (ingress)
```

We used `5..` RegEx to calculate the rate of the requests with errors grouped by `ingress`, and we divided the result with the rate of all the requests. The result is grouped by `ingress`. In my case (screenshot below), the result is approximately 4 percent (`0.04`). `Prometheus` did not yet scrape all the metrics, and I expect that number to get closer to ten percent in the next scraping iteration.



Prometheus' graph screen with the percentage with the requests with error responses

Q

We used `7..` RegEx to calculate the rate of the requests with errors grouped by `ingress`.

COMPLETED 0%

1 of 1

Let's compare the updated version of the Chart's values file with the one we used previously.

```
diff mon/prom-values-cpu-memory.yml \
    mon/prom-values-errors.yml
```

The **output** is as follows.

```
127a128,136
> - name: errors
>   rules:
>   - alert: TooManyErrors
>     expr: sum(rate(nginx_ingress_controller_requests{status=~"5.."}[5m
])) by (ingress) / sum(rate(nginx_ingress_controller_requests[5m])) by (in
gress) > 0.025
>     labels:
>       severity: error
>     annotations:
>       summary: Too many errors
>       description: At least one application produced more then 5% of err
or responses
```

The alert will fire if the rate of errors is over 2.5% of the total rate of requests.

Now we can upgrade our `Prometheus`'s Chart.

```
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-errors.yml
```

There's probably no need to confirm that the alerting works. We already saw that `Prometheus` sends all the alerts to `Alertmanager` and that they are forwarded from there to Slack.

Next, we'll move to Saturation metrics and alerts.