Face a Disaster

In this lesson, we will first simulate an issue and try to find out the issue by executing different expressions.

WE'LL COVER THE FOLLOWING

- Simulate an issue
  - Application is too slow
- Find out the issue
  - Requests are getting slow responses
- Need for detailed application-specific metrics

Let's explore one disaster scenario. Frankly, it's not going to be a real disaster, but it will require us to find a solution to an issue.

We'll start by installing the already familiar `go-demo-5` application.

```
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

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

We declared `GD5_ADDR` with the address through which we'll be able to access the application. We used it as an `ingress.host` variable when we installed the `go-demo-5` Chart. To be on the safe side, we waited until the app rolled out, and all that's left, from the deployment perspective, is to confirm that it is

running by sending an HTTP request.

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

The **output** is the developer's favorite message `hello, world!`.

Simulate an issue

Next, we'll simulate an issue by sending twenty slow requests with up to ten seconds duration. That will be our simulation of a problem that might need to be fixed.

```
for i in {1..20}; do
    DELAY=$[ $RANDOM % 10000 ]
    curl "http://$GD5_ADDR/demo/hello?delay=$DELAY"
done
```

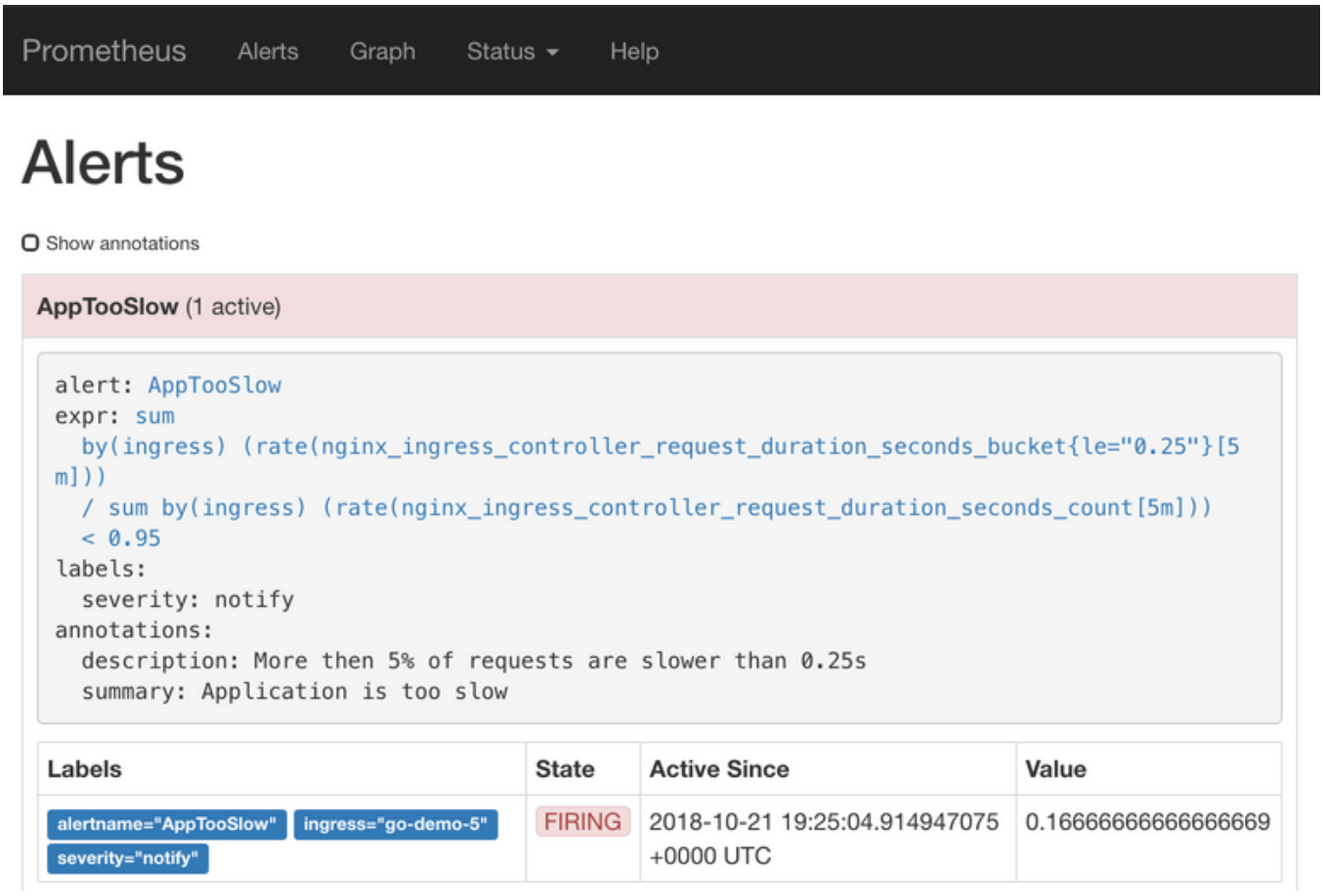
Application is too slow

Since we already have `Prometheus`'s alerts, we should receive a notification on Slack stating that the application is too slow. However, many readers might be using the same channel for those exercises, and it might not be clear whether the message comes from us. Instead, we'll open the `Prometheus' alerts screen` to confirm that there is a problem. In the "real" setting, you wouldn't be checking `Prometheus alerts`, but wait for notifications on Slack, or whichever notifications tool you chose.

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

A few moments later (don't forget to refresh the screen), the ***AppTooSlow*** alert should fire, letting us know that one of our applications is slow and that we should do something to remedy the problem.

> 📌 True to the promise that each chapter will feature outputs and screenshots from a different Kubernetes flavor, this time it's minikube's turn.



One of Prometheus' alerts in the firing state

Q

The ***AppTooSlow*** alert lets us know that one of our applications is dead and that we should do something to remedy the problem.

COMPLETED 0%

1 of 1

Find out the issue

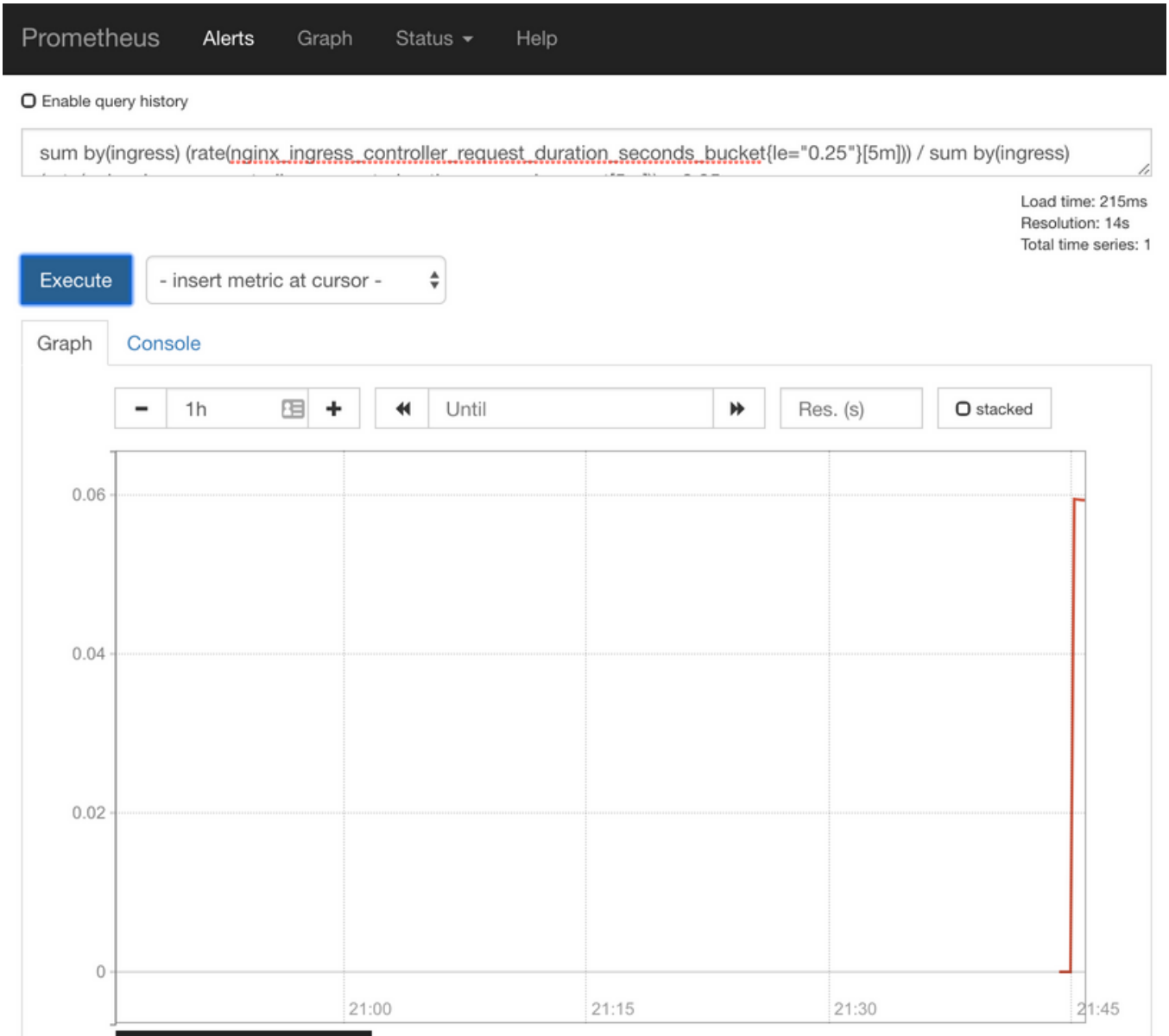
We'll imagine that we did not generate slow requests intentionally, so we'll try to find out what the issue is. Which application is too slow? What useful information can we pass to the team so that they can fix the problem as soon

as possible?

The first logical debugging step is to execute the same expression as the one used by the alert. Please expand the ***AppTooSlow*** alert and click the link of the expression. You'll be redirected to the graph screen with the expression already pre-populated. Click the *Execute* button, and switch to the *Graph* tab.

Requests are getting slow responses

We can see, from the graph, that there was a surge in the number of slow requests. The alert was fired because less than ninety-five percent of responses are within **0.25** seconds bucket. Judging from my Graph (screenshot below), zero percent of responses were inside the **0.25** seconds bucket or, in other words, all were slower than that. A moment later, it improved slightly by jumping to only six percent of fast requests. All in all, we have a situation in which too many requests are getting slow responses, and we should fix that. The main question is how to find out what the cause of that slowness is.



The graph with the percentage of requests with fast responses

Need for detailed application-specific metrics

How about executing different expressions? We can, for example, output the rate of request durations for that `ingress` (application).

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

```
sum(rate(
    nginx_ingress_controller_request_duration_seconds_sum{
        ingress="go-demo-5"
    }[5m]
)) /
sum(rate(
    nginx_ingress_controller_request_duration_seconds_count{
        ingress="go-demo-5"
    }[5m]
))
```

That graph shows us the history of request durations, but it does not get us any closer to revealing the cause of the issue or, to be more precise, to the part of the application that is slow. We could try using other metrics, but they are more or less equally generic and are probably not going to get us anywhere. We need more detailed application-specific metrics.

In the next lesson, we will see data that comes from inside the `go-demo-5` app.