Prometheus: understanding the delays on alerting

16 NOV 2016 *by Marco Pracucci* 4 COMMENTS

Prometheus is a great monitoring and alerting system. At Spreaker (self promotion - we're hiring!) we're slowly migrating to Prometheus, and thus a correct understanding in how it works is very important.

Today, me and my colleague Rocco were experimenting on the delays introduced by our Prometheus setup. In particular, it was our intention to **measure how much time it takes to get an event**, occured on a monitored target, **notified** through our notification channels. This article sums up what we've learned.

Scraping, Evaluation and Alerting

Prometheus **scrape metrics** from monitored targets at regular intervals, defined by the `scrape_interval` (defaults to `1m`). The scrape interval can be configured globally, and then overriden per job. Scraped metrics are then stored persistently on its local storage.

Prometheus has another loop, whose *clock* is independent from the scraping one, that **evaluates alerting rules** at a regular interval, defined by `evaluation_interval` (defaults to `1m`). At each

evaluation cycle, Prometheus runs the expression defined in each alerting rule and updates the alert state.

An alert can have the following **states**:

- `inactive`: the state of an alert that is neither firing nor pending
- `pending`: the state of an alert that has been active for less than the configured threshold duration
- `firing:` the state of an alert that has been active for longer than the configured threshold duration

An alert transitions from a state to another only during the evaluation cycle. How the transition occurs, depends if the alert's `FOR` clause has been set or not. From the doc:

```
The optional FOR clause causes Prometheus to wait for a certain duration
between first encountering a new expression output vector element (like an
instance with a high HTTP error rate) and counting an alert as firing for this
element.
```

- Alerts **without** the `FOR` clause (or set to `0`) will immediately transition to `firing`.
- Alerts **with** the `FOR` clause will transition first to `pending` and then `firing`, thus it will take at least two evaluation cycles before an alert with the `FOR` clause is fired.

The Alert Lifecycle

Let's do an example to better explain the lifecycle of an alert. We do have a simple alert that monitors the load 1m of a node, and fires when it's higher than 20 for at least 1 minute.

```
ALERT NODE_LOAD_1M
  IF node_load1 > 20
  FOR 1m
```

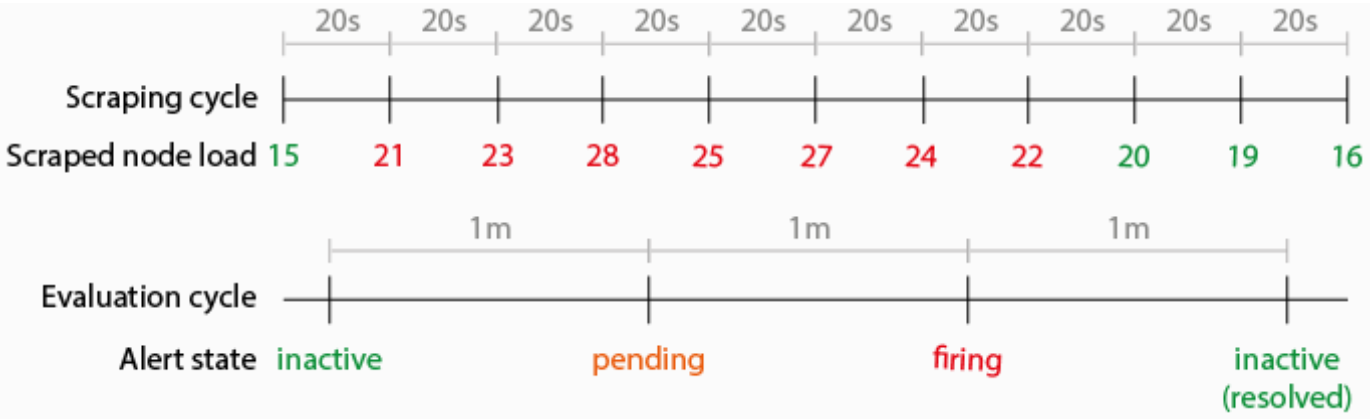
Prometheus is configured to scrape metrics every 20 seconds, and the evaluation interval is 1 minute.

```
global:
  scrape_interval: 20s
  evaluation_interval: 1m
```

Question: how much time does it take to fire `NODE_LOAD_1M`, once the avgload on the machine is higher than 20?

Answer: it takes a time between `1m` and `20s + 1m + 1m`. The upper bound is probably higher than you may expect when you set `FOR 1m`, but it actually makes much sense in the Prometheus architecture.

The reason of such worst-case timing is explained by the lifecycle of an alert. The following diagram shows the sequence of events over a timeline:



1. The load of a node constantly changes, but it gets scraped by Prometheus every `scrape_interval` (ie. `20s`)
2. Alert rules are then evaluated against the scraped metrics every `evaluation_interval` (ie. `1m`)
3. When an alert rule expression is true (ie. `node_load1 > 20`), the alert switches to `pending`, in order to honor the `FOR` clause
4. At the next evaluation cycles, if the alert expression is still true, once the `FOR` clause is honored the alert finally switches to `firing` and a notification is pushed to the **alert manager**

The Alert Manager

The Alert Manager is the last piece in the pipeline, before an alert notification gets dispatched. It receives in input the alerts once switched to `firing` or switched back to `inactive`, and then it dispatch notifications about the alert (please note that `pending` alerts are not forwarded to the Alert Manager).

The Alert Manager has several features, including **grouping** similar alerts together into a single notification. The grouping feature is particularly important to avoid bombing notification receivers when multiple alerts of the same type occur (*ie. when the same alert condition occurring on multiple nodes, we may want to receive just one notification that groups all nodes together instead of a single notification per node*).

The downside of grouping notifications is that **it may introduce further delays**.

The Alert Manager notifications grouping is basically controlled by the following settings:

```
group_by: [ 'a-label', 'another-label' ]
group_wait: 30s
group_interval: 5m
```

When a new alert is fired, it waits for `group_wait` time until the notification gets dispatched. This waiting time is necessary to eventually group further upcoming alerts that match the same `group_by` condition (in other terms, two alerts are grouped together if they have the same exact values for the `group_by` labels).

This means that if `group_wait` is set to `30s`, the alert manager will buffer the alert notification for 30 seconds, waiting for other potential notifications to be appended to the buffered notification. Once the `group_wait` time expires, the notification is finally sent.

Great! The notification is finally sent.

But what happens if at the next evaluation cycle **further alerts of the same group get fired**?

They will not wait for `group_wait` before getting dispatched, but they will wait for `group_interval`.

```
# How long to wait before sending notification about new alerts that are
# in are added to a group of alerts for which an initial notification
# has already been sent. (Usually ~5min or more.)
group_interval: 5m
```

The `group_interval` setting controls how long to wait before dispatching further notifications of the same group, and the time interval is calculated starting from the last notification sent.

References

- Illustrator file for the diagram above

Related articles

- KubeCon Europe 2018 - Memo and Takeaways
- Migrating to Prometheus: what we learned running it in production (the video)
- Migrating to Prometheus: what we learned running it in production
- Rounding values in Prometheus alerts annotations
- KubeCon 2017 - Prometheus Takeaways
- My take on the future of applications development and operability
- PHP realpath cache and Kubernetes secrets / configmap updates

- Kubernetes pods /etc/resolv.conf ndots:5 option and why it may negatively affect your application performances
- AWS re:invent 2017 annoucements
- Kubernetes RBAC with kops
- Stepping back from CTO and jumping into operations
- KubeCon 2017 - Kubernetes Takeaways
- Graceful shutdown of pods with Kubernetes
- Display the current kubectl context in the Bash prompt
- Distributed Matters Conf: Takeaways

Name