Sleep Soundly: Reliable Alerting with Unit Testing in Prometheus

Rubén Cougil Grande, DevBcn 2023

Being on-call

- **Availability** outside working hours (nights, weekends, holidays…).
- 🔥 **False Positives** due to flawed alerting rules.
- Needs of **Real Data**, hard to test these rules in advance.

Unit Testing in Prometheus

- 🔧 built-in cli tool: **promtool**
- **Assert** the alerting rule perform as intended.
- **Early detection** of issues.
- Integrate it in **CI** pipeline.
- == sleep better 😴

How Alerting Rules works in Prometheus

- Evaluation of expressions written in **PromQL**
  - `sum(up{job="app"}) == 0`
- **States**: 🟢 Inactive 🟡 Pending 🔴 Firing.
- Custom **Labels**, such as “severity”.
- **Annotations**, such as “summary” or “runbook URL”.
- 🤯 **PromQL** can be a tricky DSL.

PromQL complexity

- 📈 The complexity of the query increases the chances of **errors**.
- **Unit Tests** will help.

```
sum by(namespace, application) (increase(http_server_requests_seconds_count{namespace="live",
application="backend", status=~"500|503"}[10m]))
/
sum by(namespace, application) (increase(http_server_requests_seconds_count{namespace="live",
application="backend"}[10m])) * 100 > 10
and ON()
sum by(namespace, application) (increase(http_server_requests_seconds_count{namespace="live",
application="backend"}[10m])) > 100
```

How Alerting Rules works in Prometheus

```yaml
- alert: InstancesDownV1
  expr: sum(up{job="app"}) == 0
  labels:
      severity: sev1
  annotations:
      summary: "All instances of App are down"
      description: "All instances of App are down"
```

How to unit test a Rule

- ☝️ Generate data for the Test Case (**Arrange**):

```yaml
interval: 1m
input_series:
    - series: 'up{job="app", instance="app-1:2223"}'
      # 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      values: "0x14"

    - series: 'up{job="app", instance="app-2:2223"}'
      # 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
      values: "1x4 0x9 1x4"
```

- ☝️ Set the expectations (**Assert**):

```yaml
- eval_time: 4m
  alertname: InstancesDownV1

- eval_time: 5m
  alertname: InstancesDownV1
  exp_alerts:
    - exp_labels:
        severity: page
      exp_annotations:
        summary: "All instances of the App are down"
        description: "All instances of the App are down"

- eval_time: 15m
  alertname: InstancesDownV1
```

- ☝️ Run the test (using **promtool**):
  - **$>** *promtool test rules instances_down_v1_test.yml*

    SUCCESS

- ❗ How output would looks like **if assertion fails**:
  - **$>** promtool test rules instances_down_v1_test.yml

    FAILED:

    alertname: InstancesDownV1, time: **4m**,

    exp: [

    Labels: {alertname="InstancesDownV1", severity="page"}

    Annotations: {description="All instances of the App are down", summary="All instances of the App are down"}

    ],

    got:[ ]

Samples can be empty (“_”) or stale

- In previous example, we’ve set **“0”** when target is down in the test.
- In real world, target down will report **empty sample (“_”)**
- More accurate example:

```
input_series:
      - series: 'up{job="app", instance="app-1:2223"}'
        # _ _ _ _ _ _ _ _ _ _ _ _ _ _
        values: "_x14"

      - series: 'up{job="app", instance="app-2:2223"}'
        # 1 1 1 1 1 _ _ _ _ _ _ _ _ _ _ 1 1 1 1 1
        values: "1x4 _x9 1x4"
```

Samples can be empty ("_") or stale

- 😮 Now the **test fails**, because:
  - `expr: sum(up{job="app"}) == 0`
  - Sum of empty state sample (NaN) will never be equals to zero.

- ☝️ Expression should be:
  - `expr: sum(up{job="app"} OR on() vector(0)) == 0`
  - **`vector(0)`** `returns zero when sample is empty.`
  - ✅ `Now the test passes`

Usage of “for” in rules

- **States**: 🟢 Inactive 🟡 Pending 🔴 Firing.
- 🟡 Pending state depends on **“for”** clause
- ❗ It will **affect** test **expectations**

```yaml
- alert: InstancesDownV1
    expr: sum(up{job="app"}) == 0
    for: 5m
    labels:
        severity: sev1
    annotations:
        summary: "All instances of App are down"
        description: "All instances of App are down"
```

Scalability and Alerting Rules

- 😮 Prometheus server **does NOT scale horizontally**
- Configure Prometheus to send metrics to an **external service** (*Thanos*, *Cortex*…)
- Alerting Rules are evaluated using Prometheus internal TSBD.
- 15 days of **retention** by default.
- ☝️ Do not create rules that go beyond that.
- Keep this in mind in tests.

Benefits of Unit Testing Rules

- 👮‍♀️ Ensuring rule correctness.
- 🕵️‍♀️ Early detection of issues.
- 🛠️ Facilitating code refactoring.
- 🏄 Supporting continuous integration.
- 🙌 Collaboration and documentation.

Resources

- Github Repository with examples:
  - https://github.com/rubencougil/prometheus-testing
- Unit Testing in Prometheus official page.
- Twitter: @rcougil
- Talk resources @ Whova
- Awesome Prometheus Alerts in Github