Alerting System

Uptime is important for a backend, we need to be notified when shit hits the fan. The first step of figuring out the service isn't working is to check its status regularly.

However, placing the alerting system on the same machine as the service is a bit unwise. If the server crashes, it takes you alerting system with it. The only way to efficiently achieve this is to implement distributed alerting. We're cheap, so we don't want to pay for someone like PagerDuty/Datadog/etc, enter DIY solutions with my Raspberry Pi at home!

The problems we need to solve:

- Service to send HTTP(S) requests to an endpoint and interpret the result

- Serivice to store that information in a time series database - makes it easier to query

- Service to read time series database and fire an alert when rules are met

- Service to detect the alert and send it to a phone/email/human shock collar

I will provide a TL;DR since its quite a lengthy page and most people will loose interest.

Blackbox_Exporter sends GET requests to the Melton API and interprets the result. It stores the result in Prometheus time series database. Alertmanager reads the data and detects when an alert needs to be fired. Telegram takes that alert and sends it to a Telegram group.

image-1597596457463.png

Proceed furthur if you want a more detailed explanation, i promise i tried to make it funny.

Service to send HTTP(S) requests to an endpoint and interpret the result:

> Blackbox_exporter from prometheus is a great service that does exactly what we need, and its part of the prometheus ecosystem that I explain later on. The config simply specifies the module to use, method, the timeout and if the exporter needs to perform HTTP/HTTPS.

```
modules:
  http_2xx:
    prober: http
```

```
    timeout: 10s
    http:
      valid_status_codes: []
      method: GET
```

A perceptive individual might notice a lack of specification with regard to which URLs to test, I say this individual should chill a bit. I'm getting to it. Snitches get stitches.

Serivice to store that information in a time series database:

Prometheus is supposed to have given humanity the gift of fire, well, the software gives us the gift of monitoring - Clearly the better gift. Prometheus takes metrics and organizes the info into a time series database. It then exposes a port that is ready to be scraped by the monitoring system.

The prometheus service is the glue that bring all the other services together, its the central hub that organizes everything. Prometheus performs its tasks based on the `scrape_configs`. All the jobs described here are the metrics that Prometheus scrapes and organizes into a time series database.

Prometheus also supports alerting, One needs to just specify the endpoint to the alertmanger. We will look into that in the next step.

One can specify the scrape interval - this defines the minimum resolution of the metrics being scraped.

If one looks at the `job_name: 'blackbox'` then you see the URLs. Essentially prometheus specifies that it uses the blackbox_exporter module and run against those URLs and saves the information in prometheus with the labels.

prometheus.yml:

```
global:
  scrape_interval:     30s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 30s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

  # Attach these labels to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
      monitor: 'example'

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
```

```
  # - "first_rules.yml"
  # - "second_rules.yml"
  - alert.rules.yml

alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 30s
    scrape_timeout: 10s

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['localhost:9090']

  - job_name: node
    # If prometheus-node-exporter is installed, grab stats about the local
    # machine by default.
    static_configs:
      - targets: ['localhost:9100']

  - job_name: 'blackbox'
    metrics_path: /probe
    params:
      module: [http_2xx]
    static_configs:
      - targets:
        - http://httpbin.org/get
        - https://melton.indenwolken.tech/api/
```

```
      - https://hathibox.indenwolken.tech/status.php
      - https://grafana.indenwolken.tech/api/health
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: localhost:9115
```

Service to read time series database and fire an alert when rules are met:

Alertmanager is also part of the prometheus ecosystem, and as defined in the name, it manages alerts. The alerts themselves are defined in the `alert_rules.yml` file. The only alert we care about at the moment is when the endpoint is unreachable for a defined duration of time. That alert looks like this:

```
groups:
- name: alert.rules
  rules:
  - alert: EndpointDown
    expr: probe_success == 0
    for: 300s
    labels:
      severity: "critical"
    annotations:
      summary: "Endpoint {{ $labels.instance }} down"
```

The alertmanager simply fires the alert and nothing else. When the issue is resolved, then the alert no longer fires. Its the job of other services or integrations to perform some action on these alerts.

Service to detect the alert and send it to a phone/email/human shock collar:

Since whatsapp doesn't have a pluggable API, I had to find an integration that works with telegram or signal. I decided to go with telegram since I could find an easily pluggable telegram bot that does this job.

The telegram bot takes a token, and uses that token to post the alert from alertmanager in a specified telegram group. I found it on github, no code audit or anything, but hey - it works.

`https://github.com/inCaller/prometheus_bot`

Revision #7
Created 2 August 2020 11:35:47 by pari
Updated 16 August 2020 16:50:19 by pari