

## Contexte :

Il est question dans ce projet de monitorer le ping grâce à prometheus sur un container alpine, a la fin de ce projet nous devons être capable de mettre en place un environnement devops pour pouvoir atteindre l'objectif cité précédemment.

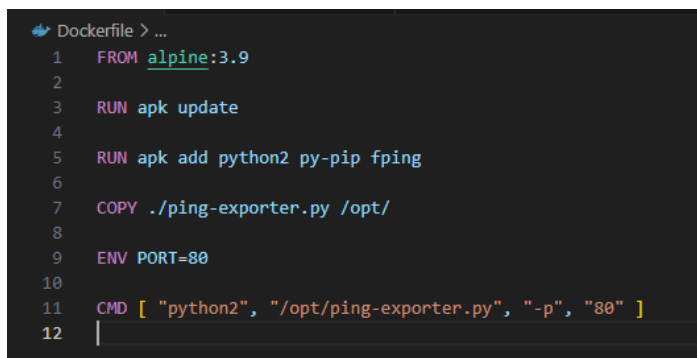
**Ressources** : prometheus.yml et ping-exporter.py

## Réalisation :

Afin de mener a bien notre projet, nous commençons par créer un Dockerfile pour pouvoir build l'image alpine et lancer l'exporter attendu :

### 1) Building the exporter & add a env variable

A partir de la description nous pouvons ressortir le Dockerfile suivant pour pouvoir faire un build de l'image, et aussi nous avons ajouter une variable d'environnement PORT de tel sorte que l'équipe pourra modifier ce port au moment du run



```
Dockerfile > ...
1 FROM alpine:3.9
2
3 RUN apk update
4
5 RUN apk add python2 py-pip fping
6
7 COPY ./ping-exporter.py /opt/
8
9 ENV PORT=80
10
11 CMD [ "python2", "/opt/ping-exporter.py", "-p", "80" ]
12 |
```

Avec la commande comme

- **docker build -t ping-exporter .**

Nous parvenons à ressortir une image builder de l'exporter.

Et avec la commande

- **docker run -e "PORT=80" ping-exporter**

Nous parvenons à lancer le container.

### 2) Run the exporter from docker-compose

Maintenant pour des raisons de clarté et de flexibilité nous allons lancer le container a partir d'un docker-compose.

Il suffit de créer une fichier docker-compose.yml et de suivre la structure définie

```

🔥 docker-compose.yml > {} services > {} prometheus > [ ] ports
1  version: '3'
2
3  services:
4  exporter:
5    container_name: ping-exporter
6    build:
7      context: .
8      dockerfile: Dockerfile
9    environment:
10     - PORT=80
11    ports:
12     - "8000:80"
13

```

La variable d'environnement reste la même, et le port aussi. Il suffit de lancer la commande : **docker-compose up** de se rendre à l'adresse <http://127.0.0.1:8000/?target=1.1.1.1> pour vérifier si tout s'est bien passé.

```

← → ↻ ⓘ 127.0.0.1:8000/?target=1.1.1.1

ping_avg 47.5
ping_max 76.5
ping_min 27.3
ping_loss 0

```

Effectivement tout s'est bien déroulé car le container retourne 4 métriques

### 3) Running prometheus

Une fois que le ping-exporter fonctionne bien et retourne les métriques, nous allons maintenant lancer prometheus qui est un outil de monitoring. Nous allons utiliser la version 2.62 de l'image de prometheus et le rajouter comme service dans notre docker-compose.yml.

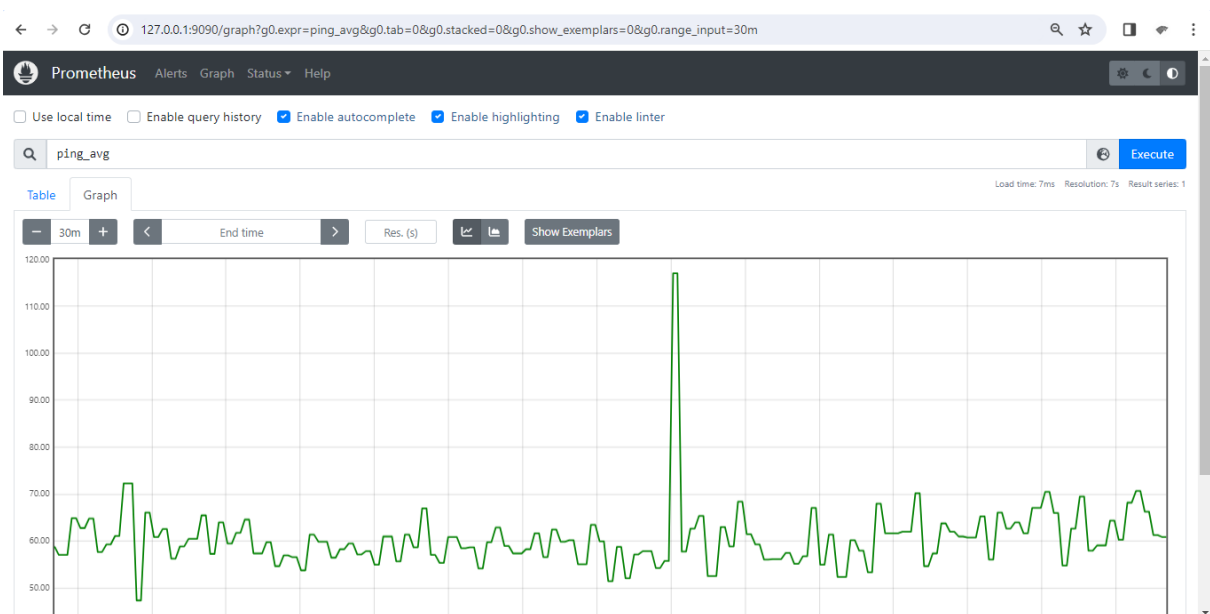
Sachant que prometheus se lance avec un fichier de configuration (prometheus.yml à modifier) il est donc nécessaire de le monter en volume avant d'utiliser les commandes, petit bonus nous ajouterons aussi un fichier prometheus-data pour que les données du futur container soient persistantes.

```

🔥 docker-compose.yml > {} services > {} prometheus > [ ] ports > 0
1  version: '3'
2
3  services:
4  exporter:
5    container_name: ping-exporter
6    build:
7      context: .
8      dockerfile: Dockerfile
9    environment:
10     - PORT=80
11    ports:
12     - "8000:80"
13
14  prometheus:
15    image: prom/prometheus:v2.36.2
16    volumes:
17     - ./conf/prometheus.yml:/etc/prometheus/prometheus.yml
18     - ./prometheus_data:/prometheus
19    command:
20     - "--config.file=/etc/prometheus/prometheus.yml"
21     - "--storage.tsdb.path=/prometheus"
22     - "--web.console.libraries=/usr/share/prometheus/console_libraries"
23     - "--web.console.templates=/usr/share/prometheus/consoles"
24    ports:
25     - "9090:9090"
26

```

Avec la commande : `docker-compose up`. Nous pouvons voir nos conteneurs se lancer et à l'adresse `http :127.0.0.1 :9000` nous avons la page d'accueil de prometheus



Le target a été bien configuré sur le service "exporter" avec le bon port. Dès à présent prometheus récupère les métriques du container alpine.

## Bonus : Grafana

Afin d'étendre le sujet, nous avons ajouté un service Grafana qui est un complémentaire à prometheus dans le monitoring. Grafana récupérera les sources de données via l'adresse ip de prometheus et constituera un dashboard personnalisé :

### 1) Installation

- On récupère l'image via [docker.io](https://hub.docker.com/r/prom/prometheus), ici on utilise la version 10.4
- On ajoute le service dans le `docker-compose.yml`
- Le port exposé est 3000

Ce qui nous donne alors :

```
docker-compose.yml > {} services > {} prometheus > [ ] command > 0
3  services:
4    exporter:
5      image: prom/ping-exporter
6      build:
7        context: .
8        dockerfile: Dockerfile
9      environment:
10       - PORT=80
11      ports:
12       - "8000:80"
13
14    prometheus:
15      image: prom/prometheus:v2.36.2
16      volumes:
17       - ./conf/prometheus.yml:/etc/prometheus/prometheus.yml
18       - ./prometheus_data:/prometheus
19      command:
20       - "--config.file=/etc/prometheus/prometheus.yml"
21       - "--storage.tsdb.path=/prometheus"
22       - "--web.console.libraries=/usr/share/prometheus/console_libraries"
23       - "--web.console.templates=/usr/share/prometheus/consoles"
24      ports:
25       - "9090:9090"
26
27    grafana:
28      image: docker.io/grafana/grafana-oss:10.4.0
29      container_name: grafana
30      ports:
31       - "3000:3000"
32      volumes:
33       - ./grafana_data:/var/lib/grafana
34      restart: unless-stopped
35
```

```

prometheus-1 | ts=2024-03-17T04:24:48.125Z caller=head.go:542 level=info component=tsdb msg="Replaying WAL, this may take a while"
grafana      | logger-migrator t=2024-03-17T04:24:48.124673364Z level=info msg="Starting DB migrations"
prometheus-1 | ts=2024-03-17T04:24:48.184Z caller=head.go:578 level=info component=tsdb msg="WAL checkpoint loaded"
prometheus-1 | ts=2024-03-17T04:24:48.216Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=2 maxSegment=5
prometheus-1 | ts=2024-03-17T04:24:48.273Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=3 maxSegment=5
prometheus-1 | ts=2024-03-17T04:24:48.285Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=4 maxSegment=5
prometheus-1 | ts=2024-03-17T04:24:48.295Z caller=head.go:613 level=info component=tsdb msg="WAL segment loaded" segment=5 maxSegment=5
prometheus-1 | ts=2024-03-17T04:24:48.295Z caller=head.go:619 level=info component=tsdb msg="WAL replay completed" checkpoint_replay_durati
on=59.174378ms wal_replay_duration=110.961258ms total_replay_duration=170.880236ms
prometheus-1 | ts=2024-03-17T04:24:48.388Z caller=main.go:993 level=info fs_type=1021997
prometheus-1 | ts=2024-03-17T04:24:48.388Z caller=main.go:996 level=info msg="TSDB started"
prometheus-1 | ts=2024-03-17T04:24:48.388Z caller=main.go:1177 level=info msg="Loading configuration file" filename=/etc/prometheus/prometh
eus.yml
grafana      | logger-migrator t=2024-03-17T04:24:48.309513895Z level=info msg="migrations completed" performed=0 skipped=547 duration=3.58
0299ms
prometheus-1 | ts=2024-03-17T04:24:48.324Z caller=main.go:1214 level=info msg="Completed loading of configuration file" filename=/etc/prom
etheus/prometheus.yml totalDuration=15.491194ms db_storage=2.3µs remote_storage=2.8µs web_handler=800ms query_engine=1.8µs scrape=3.549199ms s
crape_sd=1.024899ms notify=372.9µs notify_sd=41.1µs rules=453µs tracing=1.262699ms
prometheus-1 | ts=2024-03-17T04:24:48.324Z caller=main.go:957 level=info msg="Server is ready to receive web requests."
prometheus-1 | ts=2024-03-17T04:24:48.324Z caller=manager.go:937 level=info component="rule manager" msg="Starting rule manager..."
grafana      | logger-secrets t=2024-03-17T04:24:48.359509076Z level=info msg="Envelope encryption state" enabled=true currentprovider=secre
etKey.v1
grafana      | logger-plugin.store t=2024-03-17T04:24:48.576650595Z level=info msg="Loading plugins..."
grafana      | logger-local.finder t=2024-03-17T04:24:48.900608974Z level=warn msg="Skipping finding plugins as directory does not exist" p
ath=/usr/share/grafana/plugins-bundled
grafana      | logger-plugin.store t=2024-03-17T04:24:48.900652974Z level=info msg="Plugins loaded" count=55 duration=324.005779ms
grafana      | logger-query.data t=2024-03-17T04:24:48.900525171Z level=info msg="Query Service Initialization"
grafana      | logger-live.push.http t=2024-03-17T04:24:48.915336868Z level=info msg="Live Push Gateway Initialization"
grafana      | logger-ngalert.migration t=2024-03-17T04:24:48.986582142Z level=info msg="Starting
grafana      | logger-ngalert.state.manager t=2024-03-17T04:24:49.20450096Z level=info msg="Running in alternative execution of Error/NoDat
a mode"
grafana      | logger-infra.usagstats.collector t=2024-03-17T04:24:49.23987474Z level=info msg="registering usage stat providers" usageSt

```

## 2) Configuration du dashboard

Après un docker-compose up, on obtient l'interface de Grafana, et dessus nous configurons une source de donnée prometheus dont le server sera l'adresse ip de prometheus. On configure le dashboard et dessus on récupère toutes les métriques



Super on vient de configurer parfaitement Grafana

## 3) Configuration d'une alerte et envoi de mail

### - Configurer une alerte

Nous allons dès à présent configurer une alerte selon une condition bien établie, il suffit de se rendre dans la section alert de Grafana :

Rule type

Select where the alert rule will be managed. [Need help?](#)

Grafana-managed

Data source-managed

The alert rule type cannot be changed for an existing rule.

Expressions

Manipulate data returned from queries with math and other operations.

C Reduce

Set as alert condition

Takes one or more time series returned from a query or an expression and turns each series into a single number.

Input

A

Function

Last

Mode

Strict

Add expression

Preview

D Threshold

Alert condition

Takes one or more time series returned from a query or an expression and checks if any of the series match the threshold condition.

Input

C

IS BELOW

65

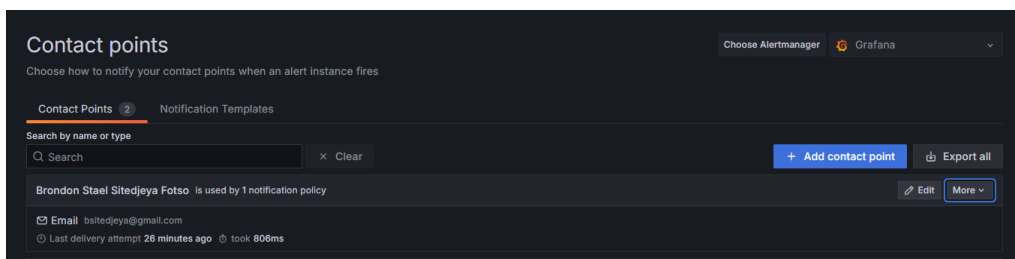
Custom recovery threshold

La première expression récupère la valeur de la métrique ping\_avg. Et la deuxième la compare à 65. Si cette métrique est inférieure à 65 alors Grafana déclenchera une alerte. A noter qu'on pourrait aussi dire tout simplement que si on ne réussit plus à avoir de métrique ce qui signifierait que le serveur est HS alors on déclenche aussi une alerte.

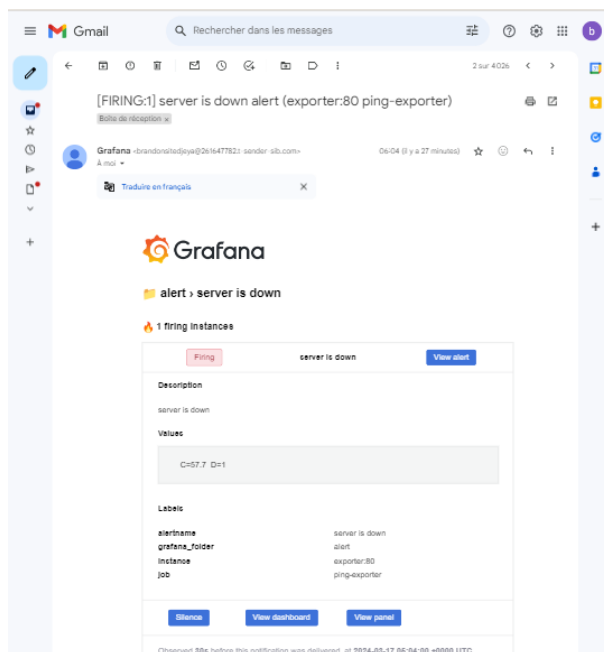
### - Configurer les emails

Après avoir configuré une alerte, il est important de configurer les emails pour notifier les responsables en cas de panne. Pour cela, nous allons configurer un serveur SMTP et ajouter un contact point sur Grafana.

- Modification du fichier grafana.ini et remplacement des conf SMTP de Brevo et enfin on ajoute un contact point et une notification sur grafana



Pour tester il suffit de se rendre sur le dashboard et selon les alertes on vérifie le mail



Nous avons bien reçu le mail pour la valeur 57.

C'est comme ça qu'on monitorise un service avec Docker, docker-compose, prometheus, grafana.

Brondon Stael Sitedjeya Fotso.