TD4
Ressources des conteneurs

Documentation utiles

`https://docs.docker.com/build/building/multi-stage/`
`https://docs.docker.com/develop/dev-best-practices/`
`https://docs.docker.com/develop/develop-images/dockerfile_best-practices/`
`https://docs.docker.com/develop/security-best-practices/`

1 Monitoring

Lors de la dernière séance, nous avons vu qu'une même machine peut héberger plusieurs applications (ou plusieurs versions d'une même application). Nous avons aussi vu comment centraliser les logs.

Aujourd'hui, nous allons monitorer les conteneurs, c.a.d. surveilleurs leur consommation de ressources cpu et ram.

1.1 Installation ctop sous docker

ctop est la version docker de la commande top. On peut l'utiliser via un conteneur.

```
docker run --rm -ti \
  --name=ctop \
  --volume /var/run/docker.sock:/var/run/docker.sock:ro \
  quay.io/vektorlab/ctop:latest
```

1.2 Installation sur la machine Hote

Vous pouvez aussi installer ctop sur votre machine en suivant la documentation
`https://github.com/bcicen/ctop`

1.3 Cadvisor

Il existe aussi une stack plsu évoluée pour monitorer vos applications.

a. Récupérer le docker-compose pour cadvisor dans les ressources du TP
b. Lancer le compose
c. Sur quel port accéder à cadvisor ?
d. Mettre en place une route traefic pour y accéder via `http://monitoring.td.anthonymoll.fr`
e. Un client vous signale une lenteur d'accès hier soir (dans ce cours, il y a une minute), vérifier dans la charge (cpu et ram) à ce moment la. Regardez aussi dans logs via Grafana.
f. Quel intérêt de cadvisor sur ctop ? à quoi sert le conteneur Redis dans ce compose ?

2 Montée en charge de votre application

Vous pouvez simuler des clients à l'aide de requêtes curl ou utiliser une des solutions de benchmarking référencés ici `https://github.com/denji/awesome-http-benchmark`

Ici, nous avons choisi l'utilitaire `bombardier` `https://github.com/codesenberg/bombardier`

g. Installer bombardier sur la machine hôte ou utiliser le via docker `https://hub.docker.com/r/alpine/bombardier`
h. Lancer le compose votre application web issue du TP1.
i. Ouvrez un outil de monitoring (ctop ou cadvisor)
j. Lancer 100 connections effectuant 1000 reqûetes sur la racine du site. Qu'observez vous ?

k. Verifier a bien une BD remplie (COPY dans le docker file inline ou appel via curl ou php au script d'initialisation des données)

l. Lancer le même test sur la page readAll.php. Qu'observez vous ?

m. Créer vous un tableau de la consommation (cpu et ram) de vos conteners en fonction du nb de clients et de requetes.

n. A l'aide d'un conteneur phpmyadmin ou d'un script php, augmentez la volumétrie de la base de données (ridiculement basse dans notre exemple de base).

o. Completer votre tableau de la consommation avec ce nouvel axe.

3 Mise en place de limitations des ressources

Sur un vrai serveur en production, on ne peut pas laisser croitre indéfiniment les ressources d'un seul conteneur. En effet, vous prendriez le risque de planter toute votre infra.

Il est préférable de "sacrifier" un seul service / conteneur.

En se basant sur les données recueillies dans le tableau ci-dessus, on va donc limiter les ressources allouées à chaque service.

p. En se basant sur la documentation du docker compose, allouez 1 cpu et 50M de Ram au conteneur *app*

q. Relancer votre test de charge et suivi la consommation de ressources grâce à l'outil de monotoring de votre choix, en augmentant progressivement la charge.

r. Qu'observez vous sur la latence quand on se rapproche des limites ?

s. Augmenter encore la charge, puis encore. Que finit il par se produire ?

4 Cycle de vie du conteneur

En lisant la documentation de docker compose, faites en sorte que que votre conteneur app redémarre automatiquement après avoir crash.

A ce stade votre stack est capable de s'auto relancer, mais elle ne tiendra pas un grosse charge. Lors de la prochaine séance, nous verrons comment lancer plusieurs instances d'un même conteneur.