APPRENDRE À DÉBOGUER VOS CONTENEURS ET VOS IMAGES DOCKER

Introduction

Dans ce chapitre, nous allons nous attaquer à la partie Debug dans Docker. Le but de ce chapitre c'est que vous soyez capable de récolter finement des informations sur vos conteneurs afin d'**être capable de réparer vos conteneurs** mais aussi d'utiliser ces données dans vos scripts dans l'intention d'**automatiser vos tâches d'administration Docker**.

Les commandes de débogage

la commande stats

Imaginez que vous utilisez un conteneur (un Apache par exemple), mais malheureusement il n'arrive plus à répondre malgré le fait que son statut soit toujours à l'état `UP`. Que feriez-vous si étiez dans ce cas précis ?

Dans un premier temps, il serait d'abord intéressant de **vérifier les statistiques d'utilisation des ressources de votre conteneur**. Ceci pour se faire à l'aide de la commande Docker stats.

Dans le but de manipuler cette commande, nous allons premièrement télécharger et ensuite lancer l'image docker httpd :

```
docker run -tid --name httpdc -p 80:80 httpd
```

Si vous lancez la commande Docker stats sans argument alors elle vous affichera en temps réel les statistiques de consommation de tous vos conteneurs en cours d'exécution. Exemple :

```
docker stats
```

Résultat :

```
CONTAINER ID    NAME        CPU %       MEM USAGE / LIMIT     MEM %       NET I/O
eaa4f4c869a2    ubuntuc     0.00%       1.777MiB / 11.61GiB   0.01%       2.61kB /
24b9fa633549    httpdc      0.00%       7.082MiB / 11.61GiB   0.06%       4.01kB /
```

Le résultat est sous forme de table, voici ci-dessous une liste d'**explication des différentes colonnes de la table de la commande Docker stats** :

- **CONTAINER ID et Name** : l'identifiant et le nom du conteneur.
- **CPU % et MEM %** : le pourcentage de CPU et de mémoire de l'hôte utilisé par le conteneur.
- **MEM USAGE / LIMIT** : la mémoire totale utilisée par le conteneur et la quantité totale de mémoire qu'il est autorisé à utiliser.
- **NET I/O** : la quantité de données que le conteneur a envoyées et reçues sur son interface réseau.
- **BLOCK I/O** : quantité de données lues et écrites par le conteneur à partir de périphériques en mode bloc sur l'hôte.
- **PIDs** : le nombre de processus ou de threads créés par le conteneur.

Vous pouvez spécifier le nom ou l'id d'un seul ou plusieurs conteneur(s), pour ne visionner que les statistiques propres à vos conteneurs :

```
docker stats httpdc
```

Résultat :

```
CONTAINER ID    NAME        CPU %       MEM USAGE / LIMIT     MEM %       NET I/O
24b9fa633549    httpdc      0.00%       7.082MiB / 11.61GiB   0.06%       4.16kB /
```

Stressons un peu notre conteneur `httpdc` avec un script shell en envoyant plusieurs requêtes, en vue de **visualiser l'augmentation de la consommation du conteneur** :

```
#!/bin/bash
curl_func () {
    curl -s "http://localhost:80/page{1, 2}.php?[1-1000]" &
}

for i in {1..4}
do
    curl_func
done

wait
echo "All done"
```

En lançant le script sur ma machine hôte, j'ai pu constater une augmentation au niveau de la consommation CPU et du flux réseau du conteneur :

```
CONTAINER ID    NAME        CPU %       MEM USAGE / LIMIT       MEM %       NET I/O
24b9fa633549    httpdc      25.24%      17.56MiB / 11.61GiB     0.15%       39.8MB / 7
```

Grâce à l'option `--format` ou `-f` vous pouvez formater le résultat de la commande Docker stats de manière à limiter l'affichage du résultat en ne représentant que les ressources souhaitées. Dans cet exemple je ne vais afficher que la consommation CPU et le flux réseau du conteneur `httpd` sous forme de table :

```
docker stats --format "table {{.Name}}\t{{.CPUPerc}}\t{{.NetIO}}" httpdc
```

Résultat :

```
NAME        CPU %         NET I/O
httpdc      19.17%         39.8MB / 73MB
```

Voici la liste des différents mots résérvés pour l'option `--format` de la commande Docker stats :

- `.Container` : Nom ou ID du conteneur (entrée utilisateur).
- `.Name` : Nom du conteneur.
- `.ID` : Identifiant du conteneur.
- `.CPUPerc` : Pourcentage de CPU.
- `.MemUsage` : Utilisation de la mémoire.
- `.NetIO` : Utilisation du flux réseau Entrant/Sortant.
- `.BlockIO` : Utilisation du disque dur en Lecture/Écriture.
- `.MemPerc` : Pourcentage de mémoire (non disponible pour le moment sous Windows).
- `.PIDs` : Nombre de PID (non disponible pour le moment sous Windows).

Autre chose, si vous désirez récupérer par exemple que la valeur de la consommation CPU à l'instant T de votre conteneur dans votre script pour la stocker ensuite dans une variable et donc par la même occasion d'**éviter le résultat en mode streaming**, alors vous pouvez utiliser l'option `--no-stream`, comme suit :

```
docker stats --no-stream  --format "{{.CPUPerc}}" httpdc
```

Résultat :

```
24.54%
```

La commande Docker inspect

La commande Docker inspect fournit des informations détaillées sous forme de tableau JSON sur les objets Docker (image Docker, conteneur docker, volume docker, etc ...). Nous avons déjà eu l'occasion d'utiliser cette commande, mais dans ce chapitre nous allons plus nous intéresser à la **la partie filtrage de résultat de la commande Docker inspect**.

Si vous tentez de lancer cette commande sur un conteneur (par exemple sur le conteneur `httpdc` créé précédemment), vous allez alors récupérer beaucoup trop d'informations :

```
docker inspect httpdc
```

Résultat (je n'affiche pas tout car c'est vraiment long) :

```
[
    {
        "Id": "24b9fa6335492cb968c000a4221bcb1d503a4befc4cb8770171f5345a350cdca",
        "Created": "2019-07-12T07:29:54.532971612Z",
       ...
            "EndpointID": "290ebd6de9ab4f3139ee93d49462a6ca692a1f18bac7888e14e5f4eba
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "5220c685dc3d77ba5547fd853e055a66f6acffd9cc2f57acde
                    "EndpointID": "290ebd6de9ab4f3139ee93d49462a6ca692a1f18bac7888e14
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
```

```
            }
        }
    }
]
```

Pour amincir le résultat, nous allons une nouvelle fois utiliser l'option de formatage `--format` ou `-f`. Nous allons commencer par récupérer les sous-éléments de l'élément `State` de la façon suivante :

```
docker inspect --format='{{json .State}}' httpdc
```

Résultat :

```
{"Status":"running","Running":true,"Paused":false,"Restarting":false,"OOMKilled":fals
```

Je vais utiliser la bibliothèque json de python (installé par défaut sur Linux) afin d'avoir un affichage plus joli :

```
docker inspect --format='{{json .State}}' httpdc | python3 -m json.tool
```

Résultat :

```
{
    "Status": "running",
    "Running": true,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "Dead": false,
    "Pid": 18532,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2019-07-12T07:29:55.132385698Z",
    "FinishedAt": "0001-01-01T00:00:00Z"
}
```

Essayons d'aller plus loins en récupérant sous un format texte l'élément `Status` de l'élément `State`. De cette manière vous pouvez par exemple **stocker le résultat dans une variable de votre script** :

```
docker inspect --format='{{ .State.Status}}' httpdc
```

Résultat :

```
running
```

Allons encore plus en profondeur et tentons de **récupérer les mappages de port d'un conteneur**. Premièrement, on va créer un conteneur utilisant différents mappages de port :

```
docker run -tid --name ubuntuc -p 9000:8000 -p 9001:8001 -p 9002:8002 ubuntu
```

Nous allons ensuite inspecter notre conteneur afin de **récupérer les informations concernant les ports** :

```
docker inspect --format='{{json .NetworkSettings.Ports}}' ubuntuc | python3 -m json.
```

Nous récupérons ainsi le tableau JSON suivant :

```
{
    "8000/tcp": [
        {
            "HostIp": "0.0.0.0",
            "HostPort": "9000"
        }
    ],
    "0001/tcp": [
        {
            "HostIp": "0.0.0.0",
            "HostPort": "9001"
        }
    ],
    "8002/tcp": [
        {
            "HostIp": "0.0.0.0",
            "HostPort": "9002"
        }
    ]
}
```

Pour récupérer des informations d'un tableau JSON depuis l'option `--format`, il faut au préalable utiliser le mot-clé `range`. Dans cet exemple nous allons d'abord récupérer la clé de chaque élément du tableau dans une variable nommée `$p`, cette clé correspondant à tous les ports exposés par votre conteneur , ensuite nous allons aussi récupérer la valeur de chaque clé dans une variable nommée `$conf` :

```
docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}}{{println "clé
```

Résultat :

```
clé : 8000/tcp =>  | valeur : [{0.0.0.0 9000}]
clé : 8001/tcp =>  | valeur : [{0.0.0.0 9001}]
clé : 8002/tcp =>  | valeur : [{0.0.0.0 9002}]
```

Maintenant gardons la variable `$p` et essayons de ne récupérer que la deuxième valeur de la variable `$conf` correspondant au port cible mapper :

```
docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}}{{println "Port
```

Résultat :

```
Port exposé : 8000/tcp  | Port cible : 9000
Port exposé : 8001/tcp  | Port cible : 9001
Port exposé : 8002/tcp  | Port cible : 9002
```

la commande logs

Il y a des risques que votre conteneur soit constamment à l'état `RESTART`. Dans ce cas il est important de **vérifier les logs de votre conteneur**.

Pour nos tests, nous allons construire une image ou j'ai rajouté exprès une erreur :

```
FROM alpine:latest

RUN apk add --no-cache apache2
```

```
EXPOSE 80

ENTRYPOINT /usr/sbin/http -DFOREGROUND
```

Buildons ensuite notre image :

```
docker build -t alpineerror .
```

Démarrons subséquemment notre conteneur avec les options suivantes :

```
docker run -d --restart always --name alpineerrorc -p 80:80 alpineerror
```

Si on vérifie l'état de notre conteneur, on constatera alors qu'il essaiera toujours de redémarrer mais sans aucun succès :

```
docker ps
```

Résultat :

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
31e4baf228c8        alpineerror         "/bin/sh -c '/usr/sb…"   About a minute ago
```

Vérifions ensuite les logs du conteneur afin de trouver la source du problème :

```
docker logs alpineerrorc
```

Résultat :

```
/bin/sh: /usr/sbin/http: not found
```

Les logs nous indiquent clairement que le chemin de la commande est introuvable. Pour corriger cette erreur il suffit juste de remplacer `/usr/sbin/http` par `/usr/sbin/httpd` dans votre Dockerfile.

la commande history

Dans certains cas il est nécessaire de **diagnostiquer la construction de votre image** en vue de l'optimiser, voire la réparer. Dans ce cas la commande Docker history vous sera d'une grande aide car elle vous indiquera les couches individuelles qui constituent votre image, ainsi que les commandes qui les ont créées, leur taille, et leur temps d'exécution.

Pour utiliser cette commande, nous allons d'abord construire une nouvelle image :

Dockerfile :

```
FROM alpine:latest
RUN apk add --no-cache git
RUN apk add --no-cache mysql-client
```

Buildons ensuite notre image :

```
docker build -t myalpine .
```

Exécutons maintenant la commande Docker History :

```
docker history myalpine
```

Résultat :

```
IMAGE               CREATED             CREATED BY
e0fee090997d        4 seconds ago       /bin/sh -c apk add --no-cache mysql-client
0c9541183535        7 seconds ago       /bin/sh -c apk add --no-cache git
b7b28af77ffe        18 hours ago        /bin/sh -c #(nop)  CMD ["/bin/sh"]
alt;missing>        18 hours ago        /bin/sh -c #(nop) ADD file:0eb5ea35741d23fe3.
```

Par défaut vous n'avez pas d'horodatage de la création de l'image, cette information peut vous être utile si vous souhaitez éxaminer le temps d'éxécution de chaque couche de votre image. Pour afficher cette information, il faut une nouvelle fois utiliser l'option `--format` avec le mot reservé `.CreatedAt` :

Avant de l'utiliser, laissez-moi d'abord vous décrire les différents mots résérvés de l'option `--format` pour la commande Docker history. Par la même occasion ça vous permettra d'en savoir davantage sur chaque colonne retournée par la table de cette commande :

- `.ID` : ID de la couche de l'image.
- `.CreatedSince` : Temps écoulé depuis la création de la couche de l'image.
- `.CreatedAt` : Horodatage de la création de la couche de l'image.
- `.CreatedBy` : Commande utilisée pour créer la couche de l'image.
- `.Size` : Taille du disque de la couche de l'image.
- `.Comment` : Commentaire de la couche de l'image.

Dans l'exemple ci-dessous nous allons dans un premier temps afficher la commande utilisée par nos différentes couches de l'image `myalpine`, et dans un second temps révéler l'horodatage de création pour chaque couche de cette même image.

```
docker history -H --format="table {{.ID}}\t{{.CreatedBy}}\t{{.CreatedAt}}" myalpine
```

Résultat :

```
IMAGE               CREATED BY                                      CREATED AT
e0fee090997d        /bin/sh -c apk add --no-cache mysql-client      2019-07-12T18:14
0c9541183535        /bin/sh -c apk add --no-cache git               2019-07-12T18:14
b7b28af77ffe        /bin/sh -c #(nop)  CMD ["/bin/sh"]              2019-07-12T00:20
&tl;missing>        /bin/sh -c #(nop) ADD file:0eb5ea35741d23fe3…   2019-07-12T00:20
```

Exercice

Énoncé

Il est temps de pratiquer un peu ! Je vous ai préparé **un petit exercice qui reprend la base de tout ce qu'on a pu étudier** depuis le début de ce chapitre.

Le but de cet exercice est de créer un script qui affiche la configuration réseau basique et les ports mappés de tous les conteneurs de votre machine locale (peu importe leur état).

Voici un aperçu du résultat final :

```
ubuntuc :
    IP : 172.17.0.3/16
    MacAddress : 02:42:ac:11:00:03
    Gateway : 172.17.0.1
    Ports :
      - 8000:9000
      - 8001:9001
      - 8002:9002

httpdc :
    IP : 172.17.0.2/16
    MacAddress : 02:42:ac:11:00:02
    Gateway : 172.17.0.1
    Ports :
      - 80:80
```

Vous pouvez utiliser n'importe quel langage de programmation. En ce qui me concerne, j'ai utilisé un script bash.

Solution

J'espère que vous avez réussi à réaliser ce tp ! Je vous présente ici ma solution. Bien sûr, je ne détiens pas la meilleure solution donc n'hésitez à partager votre code dans les commentaires ou sur le serveur discord.

```
#!/bin/bash
```

```
# Récupération des noms des conteneurs
containers=$(docker ps -a --format={{.Names}})

for container in $containers
do
    # Récupération des informations du conteneur
    IP=$(docker inspect --format='{{.NetworkSettings.IPAddress}}' $container)
    IPp=$(docker inspect --format='{{.NetworkSettings.IPPrefixLen}}' $container)
    MACADDR=$(docker inspect --format='{{.NetworkSettings.MacAddress}}' $container)
    GATEWAY=$(docker inspect --format='{{.NetworkSettings.Gateway}}' $container)
    PORTS=$(docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}}{{


    # Affichage des informations du conteneur
    echo -e "$container :"
    echo -e "\tIP : $IP/$IPp"
    echo -e "\tMacAddress : $MACADDR"
    echo -e "\tGateway : $GATEWAY"
    echo -e "\tPorts : \n${PORTS//\/tcp/}\n"
done
```

Conclusion

Il est temps de conclure ce chapitre. Dans les chapitres précédents, nous avions vu comment déboguer quelques objets docker comme les volumes et les réseaux Docker, mais cette fois-ci nous nous sommes plus concentré sur le débogage des conteneurs et images Docker.

Comme toujours, je vous partage un pense-bête des commandes que nous avons pu voir dans ce chapitre :

```
## Récupérer des informations de bas niveau d'un conteneur ou d'une image
docker inspect <CONTAINER_ID ou CONTAINER_NAME ou IMAGE_NAME ou IMAGE_ID>
    -f ou --format : formater le résultat



## Afficher en temps réels les statistiques des différentes
## ressources consommées par votre conteneur en mode streaming
docker stats <CONTAINER_ID ou CONTAINER_NAME>
    -f ou --format : formater le résultat
    --no-stream : désactiver le mode streaming
```

```
## Visualiser des informations sur les différentes couche de votre image
docker history <IMAGE_NAME ou IMAGE_ID>
    -f ou --format : formater le résultat


## Examiner les logs d'un conteneur
docker logs <CONTAINER_ID ou CONTAINER_NAME>
    -f : suivre en permanence les logs du conteneur
    -t : afficher la date et l'heure de la réception de la ligne de log
    --tail <NOMBRE DE LIGNE> = nombre de lignes à afficher à partir de la fin (par de
```