# Docker

# CHEAT SHEET

Aurélien GARNIER & Luc FASQUELLE

Juin 2018

Cabinet de conseil IT et Agilité | SOAT

# GESTION DES CONTENEURS (1/2)

CHEAT SHEET

docker

**exécuter** en mode détaché un conteneur nommé webserver, à partir de l'image nginx version 1.10.1, mappé sur le port interne 80 en lui partageant un répertoire local

```
docker run -d --name webserver -p 80:80 -v
$(pwd)/myConf.d:/etc/nginx/conf.d nginx:1.10.1
```

**récupérer** le fichier myConf à l'intérieur d'un conteneur nommé webserver pour le **copier** en local

```
docker cp webserver:/etc/nginx/conf.d/myConf ~
```

**exécuter** un conteneur nommé webserver en lui passant des variables d'environnement

```
docker run --env VERSION=1.0 --env WEBHOST=myhost
--name webserver nginx:1.10.1
```

**exécuter** une commande au sein du conteneur webserver en mode interactif

```
docker exec -ti webserver $CMD
```

**afficher les processus** de mon conteneur webserver

```
docker top webserver
```

**visualiser** en continu les logs de mon conteneur webserver

```
docker logs --follow webserver
```

**envoyer** les logs de mon conteneur nginx vers le server greylog en lui spécifiant le driver gelf

```
docker run --log-driver gelf --log-opt gelf-
address=udp://1.2.3.4:12201 nginx:1.10.1
```

NOTES

# GESTION DES CONTENEURS (2/2)

CHEAT SHEET

docker

**supprimer les conteneurs** arrêtés

```
docker rm $(docker ps -a -q --filter
"status=created" --filter "status=exited" --filter
"status=dead" --filter "status=paused")
```

**inspecter** certains champs du JSON qui décrit mon conteneur webserver

```
docker container inspect --format="{{json
.State.Status}}" webserver
```

**afficher** tous les conteneurs en état Running dont le nom contient webserver

```
docker ps --filter name=webserver
```

**visualiser les changements** opérés dans mon conteneur webserver

```
docker diff webserver
```

**détruire** tous les conteneurs tournant actuellement

```
docker kill $(docker ps -q)
```

NOTES

# GESTION DES IMAGES

CHEAT SHEET

docker

**supprimer l'image** redis même si elle est utilisée

```
docker rmi -f redis
```

créer un **tag** stable pour l'image my-app

```
docker tag my-app:0.1.1 my-app:stable
```

me **connecter à une registry** existante, avec mon login

```
docker login -u me@corp.com registry.corp:5000
```

**récupérer une image** depuis ma registry

```
docker pull registry.corp/my-app:stable
```

**publier une image** dans ma registry

```
docker push registry.corp/my-app:0.1.2
```

**nettoyer** les images de plus de 24h, qu'elles soient utilisées ou non par des conteneurs

```
docker image prune --all --filter "until=24h"
```

**nettoyer** les conteneurs arrêtés / images obsolètes / volumes inutilisés

```
docker container prune
docker image prune
docker volume prune
```

**sauvegarder l'image** webserver sous forme de tarball, pour la transmettre sans passer par une registry puis **l'importer** sur la machine d'un partenaire

```
docker save -o webserver.tar webserver
docker load -i webserver.tar
```

NOTES

# DOCKER COMPOSE (1/2)

CHEAT SHEET

docker

**exécuter** en mode détaché une stack de conteneurs depuis une configuration docker-compose en m'assurant que les conteneurs seront recréés même s'ils sont inchangés

```
docker-compose up --force-recreate -d
```

**valider la configuration** du fichier docker-compose-dev.yml

```
docker-compose --file ./docker-compose-dev.yml config
```

**détruire** les conteneurs d'une stack et nettoyer tous les volumes associés

```
docker-compose down --volumes
```

**construire** les images de mon docker-compose tout en supprimant les conteneurs intermédiaires construits lors du build

```
docker-compose build --force-rm
```

**stopper** les conteneurs d'une stack sans les supprimer

```
docker-compose stop
```

NOTES

# DOCKER COMPOSE (2/2)



docker

- exemple de fichier docker-compose.yml

```yaml
version: '3.3'

services:
 webserver:
  image: nginx:1.10.1
  environment:
   - APP_VERSION=1.0-SNAPSHOT
   - SITE_NAME=mySite
  volumes:
   - ./mySite/conf:/etc/nginx/conf.d
   - data
  ports:
   - 8081:80
   - 8443:443
  labels:
   - 'com.example.version=1.0'
   - 'com.example.appname=myApp'
 volumes:
  data:
```

NOTES

**Aurélien GARNIER**
**Lead dev Java & Devops - SOAT**

*Développeur dans l'écosystème Java depuis 2005, j'interviens aujourd'hui sur diverses missions d'expertise en tant que leader technique sur des problématiques variées allant de l'architecture au développement d'applications, en accordant une importance particulière aux problématiques DevOps.*
*Aujourd'hui, la solution Docker intègre mon panel de compétences, et constitue mon fil conducteur à SOAT à travers des masterclass, formations et autres livrables autour de ce thème.*

**Luc FASQUELLE**
**Développeur .NET & Cloud - SOAT**

*Fort de mon expérience dans les environnements Cloud ( Azure, AWS ... ), j'interviens chez différents clients aux domaines fonctionnels et techniques variés ( .Net, React, Angular ... ) afin de les aider à réaliser leurs projets avec succès.*
*Ayant une volonté de rester en adéquation avec les besoins client, je suis également de très près l'écosystème de la containérisation avec entre autres, Docker et Kubernetes.*

SOAT
89 quai Panhard et Levassor 75013 Paris
tél. : + (33) 1.44.75.42.55
contact@soat.fr - www.soat.fr

**Retrouvez-nous sur**

Cabinet de conseil IT et Agilité | SOAT