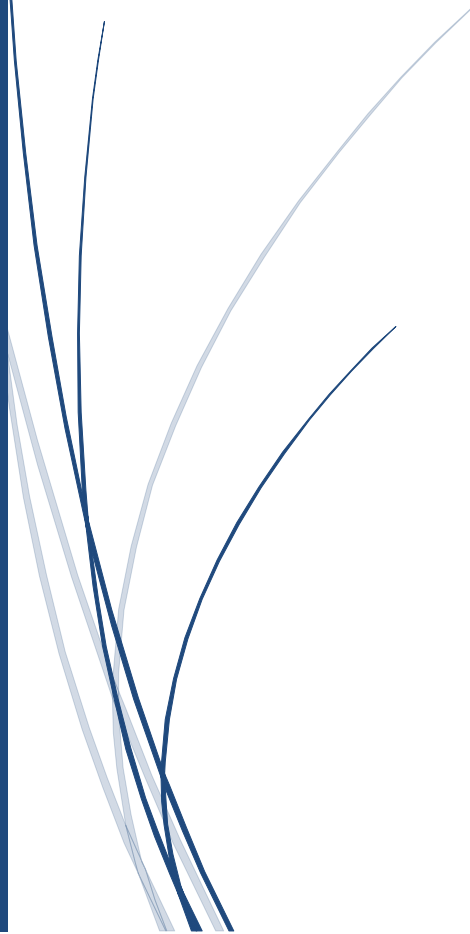




# TP Virtualisation et CloudTP5: Docker

Réalisé par:

- LARBAOUI NOUE EL IMENE
  - MAHDI ZHOR YASMINE
- 

<b>Introduction:</b>	<b>1</b>
<b>Déployer un conteneur Docker</b>	<b>1</b>
Étape 1 - Exécution d'un conteneur	1
Étape 2 - Recherche des container en cours d'exécution	2
Etape 3 - Accéder Redis	3
Étape 4 - Persistance des données	3
Étape 6 - Exécution d'un conteneur au premier plan	4
<b>Déployer un site Web HTML statique en tant que conteneur</b>	<b>4</b>
Étape 1 - Créer le Dockerfile	4
Etape 2 - Build Docker Image	5
Etape 3 - Run	5
<b>Embarquez une application Python dans un conteneur :</b>	<b>6</b>
Etape 1 - Base Image	6

## Introduction:

Docker se décrit comme "une plate-forme ouverte pour les développeurs et les administrateurs système pour créer, expédier et exécuter des applications distribuées".

Docker permet d'exécuter des conteneurs. Un conteneur est un processus en bac à sable exécutant une application et ses dépendances sur le système d'exploitation hôte. L'application à l'intérieur du conteneur se considère comme le seul processus en cours d'exécution sur la machine alors que la machine peut exécuter plusieurs conteneurs indépendamment.

Ce présent rapport englobe 3 partie:

1. Déployer un container docker en utilisant une image Redis
2. Déployer un site Web HTML statique en tant que conteneur
3. Embarquez une application Python dans un conteneur :

## Déployer un conteneur Docker

### Étape 1 - Exécution d'un conteneur

Avec Docker, tous les conteneurs sont démarrés sur la base d'une image Docker. Ces images contiennent tout le nécessaire pour lancer le processus; l'hôte ne nécessite aucune configuration ni dépendance.

Dans ce tp on utilisera une image Redis récupérer de [registry.hub.docker.com/](https://registry.hub.docker.com/)

#### A. retrouver l'image

```
$ docker search redis
```

NAME	DESCRIPTION	STARS
OFFICIAL	AUTOMATED	
redis	Redis is an open source key-value store that...	7905
[OK]		
bitnami/redis	Bitnami Redis Docker Image	138
	[OK]	
sameersbn/redis		79
	[OK]	
grokzen/redis-cluster	Redis cluster 3.0, 3.2, 4.0 & 5.0	64
rediscommander/redis-commander	Alpine image for redis-commander - Redis man...	35
	[OK]	
kubeguide/redis-master	redis-master with "Hello World!"	31
redislabs/redis	Clustered in-memory database engine compatib...	24
redislabs/redisearch	Redis With the RedisSearch module pre-loaded...	20
oliver006/redis_exporter	Prometheus Exporter for Redis Metrics. Supp...	20

```
$ docker run -d redis
bc9f6378d50c56cf138949a012854833c5782d4b4d884593c174bb8aa43057c1
$
```

Docker exécutera la dernière version disponible. Si une version particulière était requise, elle pourrait être spécifiée comme une balise, par exemple, la version 3.2 serait `docker run -d redis: 3.2`.

### Étape 2 - Recherche des container en cours d'exécution

Le conteneur lancé s'exécute en arrière-plan, la commande `docker ps` répertorie tous les conteneurs en cours d'exécution, l'image utilisée pour démarrer le conteneur et la disponibilité.

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
a49890a4aba7	redis	"docker-entrypoint.s..."	About a minute ago	Up About a min
ute 6379/tcp	cocky_hypatia			

```
$
```

La commande précédente affiche également le nom convivial et l'ID qui peuvent être utilisés pour rechercher des informations sur les conteneurs individuels.

La commande `docker inspect <friendly-name | container-id>` fournit plus de détails sur un conteneur en cours d'exécution, comme l'adresse IP.

La commande `docker logs <nom-conteneur | id-conteneur>` affichera les messages que le conteneur a écrits en erreur standard ou en sortie standard.

### Étape 3 - Accéder Redis

chaque conteneur est en bac à sable. Si un service doit être accessible par un processus qui ne s'exécute pas dans un conteneur, le port doit être exposé via l'hôte.

Une fois exposé, il est possible d'accéder au processus comme s'il s'exécutait sur le système d'exploitation hôte lui-même.

par défaut, Redis s'exécute sur le port 6379. Elle a appris que par défaut, d'autres applications et bibliothèque s'attendent à ce qu'une instance de Redis écoute sur le port.

```
$ docker run -d --name redisHostPort -p 6379:6379 redis:latest
a119592ed7bc10750d500f57f2bb82ff917f4dbc5f6cd3e0d00c89f3f08a4fdc
$
```

```
$ docker run -d --name redisDynamic -p 6379 redis:latest
060c223800c89ecae81dba06d7b03d6a26bfb35687e6efb6010ed40b33e107ca
$ docker port redisDynamic 6379
0.0.0.0:32768
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
060c223800c8	redis:latest	"docker-entrypoint.s..."	11 seconds ago	Up 9 seconds
ec3c3b609479	redis:latest	"docker-entrypoint.s..."	About a minute ago	Up About a min
ute 0.0.0.0:6379->6379/tcp	redisHostPort			
d236166ecd55	redis	"docker-entrypoint.s..."	About a minute ago	Up About a min
ute 6379/tcp	flamboyant_mcnulty			

```
$
```

### Étape 4 - Persistance des données

Toutes les données qui doivent être enregistrées sur l'hôte Docker, et non à l'intérieur des conteneurs, doivent être stockées dans `/opt/docker/data/redis/` avec la commande suivante:

```
$ docker run -d --name redisMapped -v /opt/docker/data/redis:/data redis
5c481ea44e90d5ed249d1bbad5dea1a07d2ee8d2ef781eeffb44ad2f41e6f657d
$
```

## Étape 6 - Exécution d'un conteneur au premier plan

```
$ docker run ubuntu ps
PID TTY          TIME CMD
  1 ?            00:00:00 ps
$ docker run -it ubuntu bash
root@38b78e77944f:/#
```

## Déployer un site Web HTML statique en tant que conteneur

créer une image Docker pour exécuter un site Web HTML statique à l'aide de Nginx.

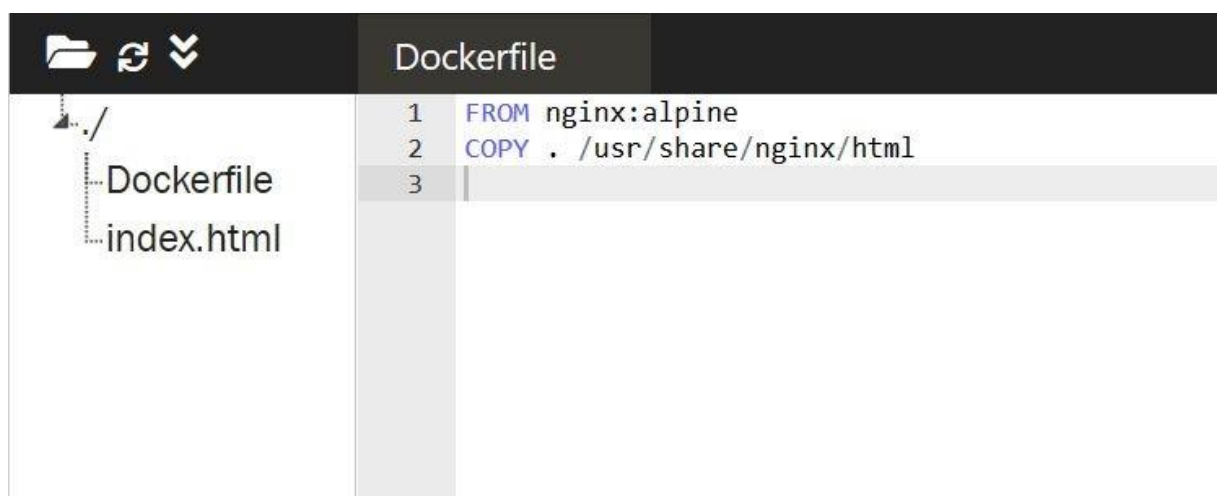
### Étape 1 - Créer le Dockerfile

Les images Docker commencent à partir d'une image de base. L'image de base doit inclure les dépendances de plate-forme requises par votre application, par exemple, lorsque la JVM ou le CLR est installé.

Cette image de base est définie comme une instruction dans le Dockerfile. Les images Docker sont construites sur la base du contenu d'un Dockerfile. Le Dockerfile est une liste d'instructions décrivant comment déployer votre application.

notre image de base ici est la version alpine de Nginx. Cela fournit le serveur Web configuré sur la distribution Linux Alpine.

Création de notre Dockerfile pour construire l'image:



La première ligne définit notre image de base. La deuxième ligne copie le contenu du répertoire actuel dans un emplacement particulier à l'intérieur du conteneur.

## Etape 2 - Build Docker Image

Le Dockerfile est utilisé par la commande de génération Docker CLI. La commande build exécute chaque instruction dans le Dockerfile. Le résultat est une image Docker intégrée qui peut être lancée et exécutée votre application configurée.

La commande build accepte certains paramètres différents. Le format est docker build -t <build-directory>. Le paramètre -t nous permet de spécifier un nom convivial pour l'image et une balise, couramment utilisé comme numéro de version. Cela nous permet de suivre les images construites et d'être sûr de la version en cours de démarrage.

Construisez notre image HTML statique en utilisant la commande build ci-dessous:

```
Terminal    docker:80  +
Your Interactive Bash Terminal. A safe place to learn and execute commands.
$
$ docker build -t webserver-image:v1 .
Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM nginx:alpine
--> 377c0837328f
Step 2/2 : COPY . /usr/share/nginx/html
--> e5e57f5e96e3
Successfully built e5e57f5e96e3
Successfully tagged webserver-image:v1
$
```

Visualisez les images docker disponible:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
webserver-image     v1                 e5e57f5e96e3       30 seconds ago     19.7MB
nginx               alpine            377c0837328f       6 days ago         19.7MB
redis               latest            4760dc956b2d       24 months ago      107MB
ubuntu              latest            f975c5035748       2 years ago        112MB
alpine              latest            3fd9065eaf02       2 years ago        4.14MB
$
```

l'image construite est webserver-image avec le tag v1.

## Etape 3 - Run

L'image intégrée peut être lancée de manière cohérente avec d'autres images Docker. Lorsqu'un conteneur se lance, il est mis en bac à sable à partir d'autres processus et réseaux sur l'hôte. Lors du démarrage d'un conteneur, il faut lui donner l'autorisation et l'accès à ce dont il a besoin.

Par exemple, pour ouvrir et se lier à un port réseau sur l'hôte, vous devez fournir le paramètre -p <host-port>: <container-port>.

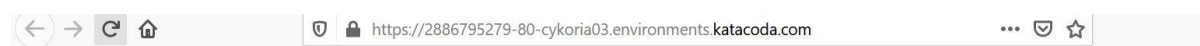


Lancez notre image nouvellement construite fournissant le nom convivial et la balise Comme il s'agit d'un serveur Web, liez le port 80 à notre hôte à l'aide du paramètre -p.

```
$ docker run -d -p 80:80 webserver-image:v1
8ad5d0ffcd137a8be7054ea545e75f1dca21f2b2caa5a660f700db5ed736076
```

Maintenant on peut accéder au résultat via le port 80 comme suit:

```
$ curl docker
<h1>Mon premier conteneur</h1>
```

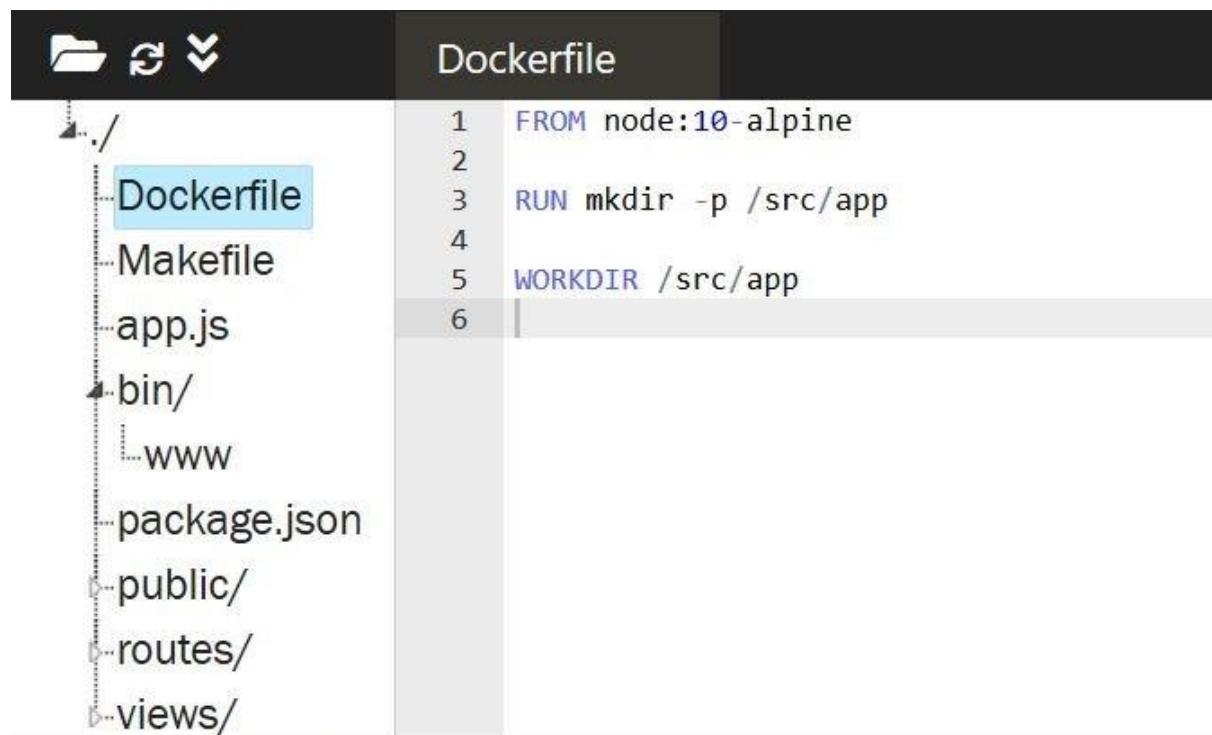


## Mon premier conteneur

maintenant nous avons un site Web HTML statique desservi par Nginx.

## Embarquez une application Node.js dans un conteneur :

### Etape 1 - Base Image



## Etape 2- NPM install

	Dockerfile
./	1 FROM node:10-alpine
Dockerfile	2
Makefile	3 RUN mkdir -p /src/app
app.js	4
bin/	5 WORKDIR /src/app
www	6
package.json	7 COPY package.json /src/app/package.json
public/	8
routes/	9 RUN npm install
	10

## Etape 3- Configuration de l'application

	Dockerfile
./	1 FROM node:10-alpine
Dockerfile	2
Makefile	3 RUN mkdir -p /src/app
app.js	4
bin/	5 WORKDIR /src/app
www	6
package.json	7 COPY package.json /src/app/package.json
public/	8
routes/	9 RUN npm install
views/	10
	11 COPY . /src/app
	12
	13 EXPOSE 3000
	14
	15 CMD [ "npm", "start" ]
	16

## Etape 4- Building et Lancement du container

Build l'image avec la commande:

```
docker build -t my-nodejs-app . ✓
```



```

$ docker build -t my-nodejs-app .
Sending build context to Docker daemon 17.92kB
Step 1/8 : FROM node:10-alpine
10-alpine: Pulling from library/node
c9b1b535fdd9: Pull complete
514d128a791d: Pull complete
ab9dddf2630f: Pull complete
acb767e231ef: Pull complete
Digest: sha256:e8d05985dd93c380a83da00d676b081dad9cce148cb4ecdf26ed684fcff1449c
Status: Downloaded newer image for node:10-alpine
---> 29fc59abc5de
Step 2/8 : RUN mkdir -p /src/app
---> Running in 948dc824fff7
Removing intermediate container 948dc824fff7
---> 30a7b142a7a7
Step 3/8 : WORKDIR /src/app

```

Terminal
+

```

Step 4/8 : COPY package.json /src/app/package.json
---> Using cache
---> 641e53fad232
Step 5/8 : RUN npm install
---> Using cache
---> 1255d6964bdc
Step 6/8 : COPY . /src/app
---> Using cache
---> 8904fa96dfde
Step 7/8 : EXPOSE 3000
---> Using cache
---> 8031856c4754
Step 8/8 : CMD [ "npm", "start" ]
---> Using cache
---> 9a651e6264de
Successfully built 9a651e6264de
Successfully tagged my-nodejs-app:latest

```

Lancer l'image construite:

```

$ docker run -d --name my-running-app -p 3000:3000 my-nodejs-app
92faa1b3dce57790b02423fe6983ed97e0f25e9243bc5d571b1041d4b99b0f8e

```

on peut tester le container avec la commande :

```
curl http://docker:3000 ✓
```

### **Etape 5- Variable d'environnement**

Avec Docker, les variables d'environnement peuvent être définies lorsque on lance le conteneur. Par exemple, avec les applications Node.js, on doit définir une variable d'environnement pour NODE\_ENV lors de l'exécution en production.

En utilisant l'option -e, nous pouvons définir le nom et la valeur comme -e NODE\_ENV = production, dans notre cas on peut faire:

```
$ docker run -d --name my-production-running-app -e NODE_ENV=production -p 3000:3000 my-nodejs-app  
d057ba2423b421bc75bcfd520b9826573fc86b2a601c53f37312b7d2c247f9d3
```