

Rapport de projet

« HumansBestFriend »

Groupe TD-49 – Virtualisation & Conteneurisation

Table des matières

Introduction	2
Equipe	2
Sujet	2
Réalisation	3
Docker Hub.....	9
Conclusion.....	9

Introduction

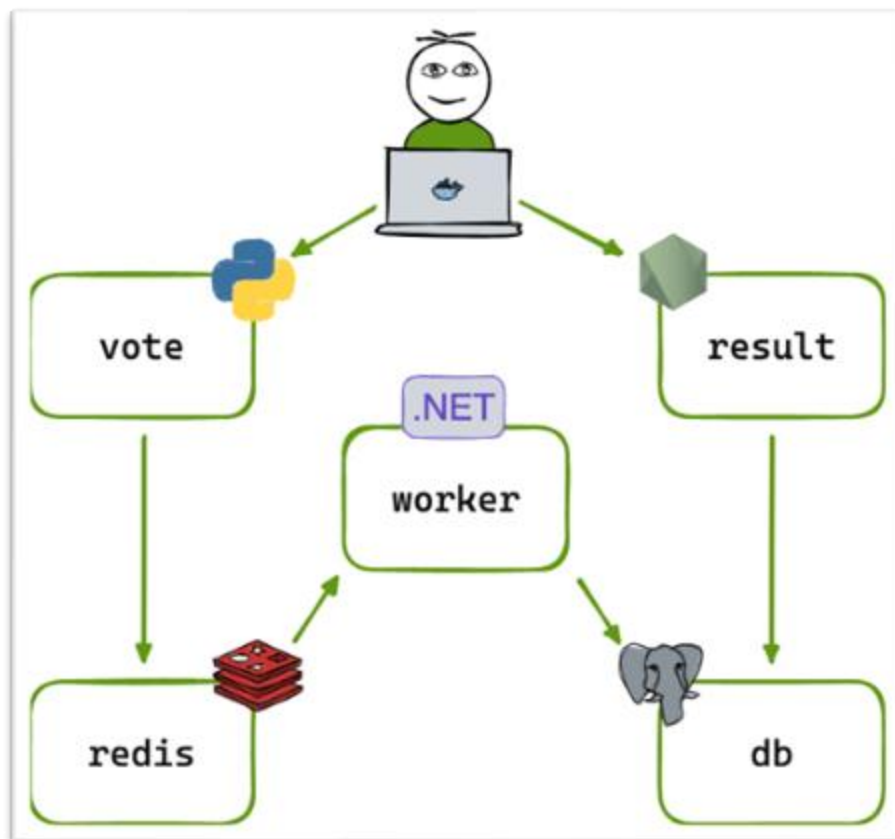
Equipe

- DA CRUZ Benjamin
- DE CASTRO Lucas
- DRAMÉ Arouna
- LATEB Samy

Sujet

Le projet consiste en la création d'une application distribuée simple s'exécutant sur plusieurs conteneurs Docker. L'ensemble du développement doit être réalisé à l'intérieur d'une machine virtuelle exécutant Docker et Docker Compose, conformément à la documentation Docker fournie au cours. Les technologies utilisées incluent Python, Node.js, .NET, Redis pour la messagerie, et Postgres pour le stockage.

Pour la réalisation de ce projet, nous avons créé un fork du projet initial (voir [ici](#)) vers notre dépôt Github (voir [ici](#)).



Réalisation

L'intégralité du code source de notre projet se trouve dans notre [dépôt publique Github](#) et c'est à partir de celui-ci que toute l'installation se fait dans notre procédure d'installation (cf. [SUBMISSION.md](#)), comme spécifié dans le sujet.

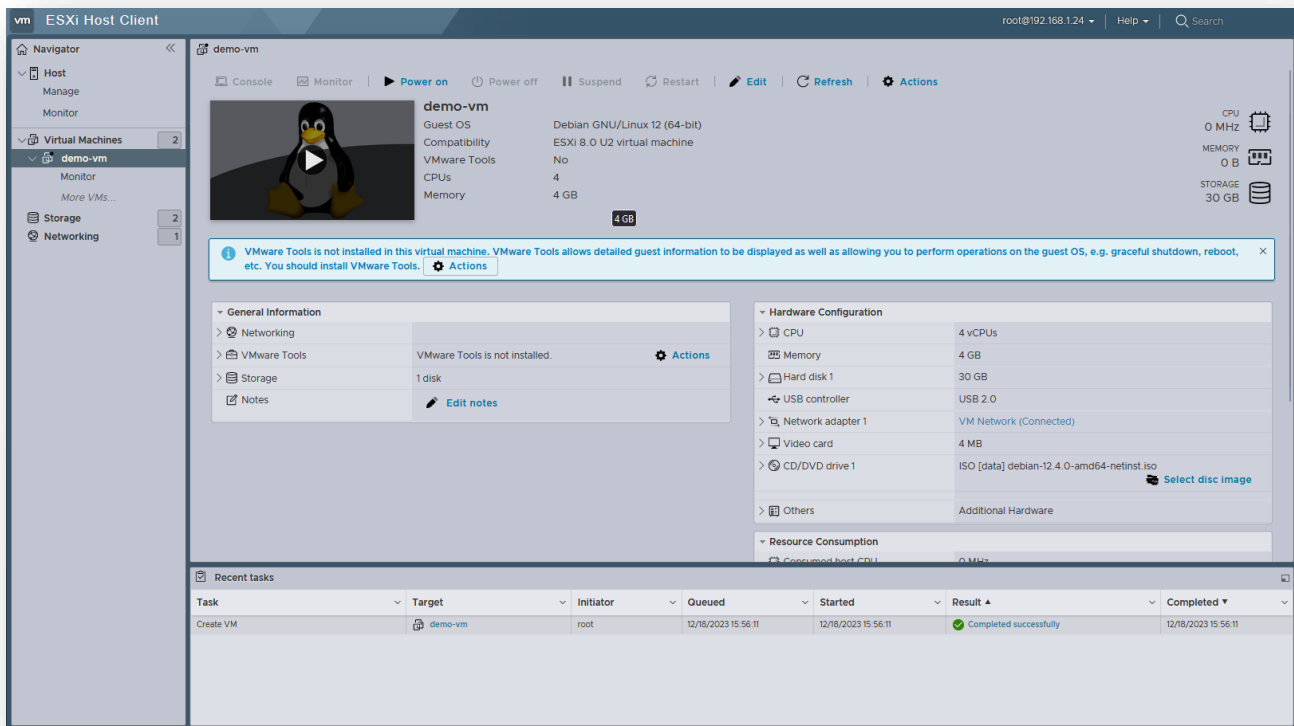
Afin de démontrer le bon fonctionnement de notre projet, nous avons réalisé une démo d'installation et de fonctionnement de notre projet sous ses deux formes : la forme « manuelle » qui n'utilise que des commandes docker et sans utiliser docker compose, et une forme dite « automatisée » qui utilise docker compose.

Tout cela a été réalisé dans une machine virtuelle nommée « demo-vm » installée sur notre instance ESXI.

The screenshot shows the 'New virtual machine' wizard in VMware vSphere, specifically the 'Ready to complete' step. The wizard is titled 'New virtual machine - demo-vm (ESXi 8.0 U2 virtual machine)'. On the left, a sidebar lists five steps: 1. Select creation type, 2. Select a name and guest OS, 3. Select storage, 4. Customize settings, and 5. Ready to complete. The main area displays a summary of the configuration settings for the virtual machine.

Ready to complete	
Review your settings selection before finishing the wizard	
Name	demo-vm
Datastore	data
Guest OS name	Debian GNU/Linux 12 (64-bit)
Compatibility	ESXi 8.0 U2 virtual machine
vCPUs	4
Memory	4096 MB
Network adapters	1
Network adapter 1 network	VM Network
Network adapter 1 type	VMXNET 3
IDE controller 0	IDE 0
IDE controller 1	IDE 1
SCSI controller 0	VMware Paravirtual
SATA controller 0	New SATA controller
Hard disk 1	
Capacity	30 GB
Datastore	[data] demo-vm/
Mode	Dependent
Provisioning	Thick provisioned, lazily zeroed
Controller	SCSI controller 0 : 0
CD/DVD drive 1	
Backing	[data] debian-12.4.0-amd64-netinst.iso
Connected	Yes
USB controller 1	USB 2.0

At the bottom right, there are four buttons: CANCEL, BACK, NEXT, and FINISH. The FINISH button is highlighted in blue.



Nous avons ensuite installé un serveur SSH sur notre VM afin de pouvoir nous y connecter depuis notre hôte, et ainsi rendre les étapes suivantes de lancement de l'application beaucoup plus aisées.

```

root@192.168.1.115
> ssh root@192.168.1.115
The authenticity of host '192.168.1.115 (192.168.1.115)' can't be established.
ED25519 key fingerprint is SHA256:OglcZF5NuEu44HVQXT2UETUeDrCUyuqcjQsqIAeZNE4.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.115' (ED25519) to the list of known hosts.
root@192.168.1.115's password:
Linux demo-vm 6.1.0-16-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.67-1 (2023-12-12) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Dec 18 16:36:37 2023
root@demo-vm:~#

```

Passons maintenant à l'installation de notre application de la première manière, l'installation « manuelle » (uniquement avec des commandes docker et sans docker compose).

On clone d'abord le dépôt et on entre dedans.

```
root@192.168.1.115
root@demo-vm:~# git clone https://github.com/LcsH0s/virtu-project-td49.git
cd virtu-project-td49/src/cmd/
Cloning into 'virtu-project-td49'...
remote: Enumerating objects: 145, done.
remote: Counting objects: 100% (145/145), done.
remote: Compressing objects: 100% (107/107), done.
remote: Total 145 (delta 51), reused 117 (delta 26), pack-reused 0
Receiving objects: 100% (145/145), 241.22 KiB | 4.39 MiB/s, done.
Resolving deltas: 100% (51/51), done.
root@demo-vm:~/virtu-project-td49/src/cmd# |
```

On lance ensuite successivement toutes les commandes docker spécifiées dans la [procédure d'installation](#) afin de créer les conteneurs, réseaux et volumes nécessaires.

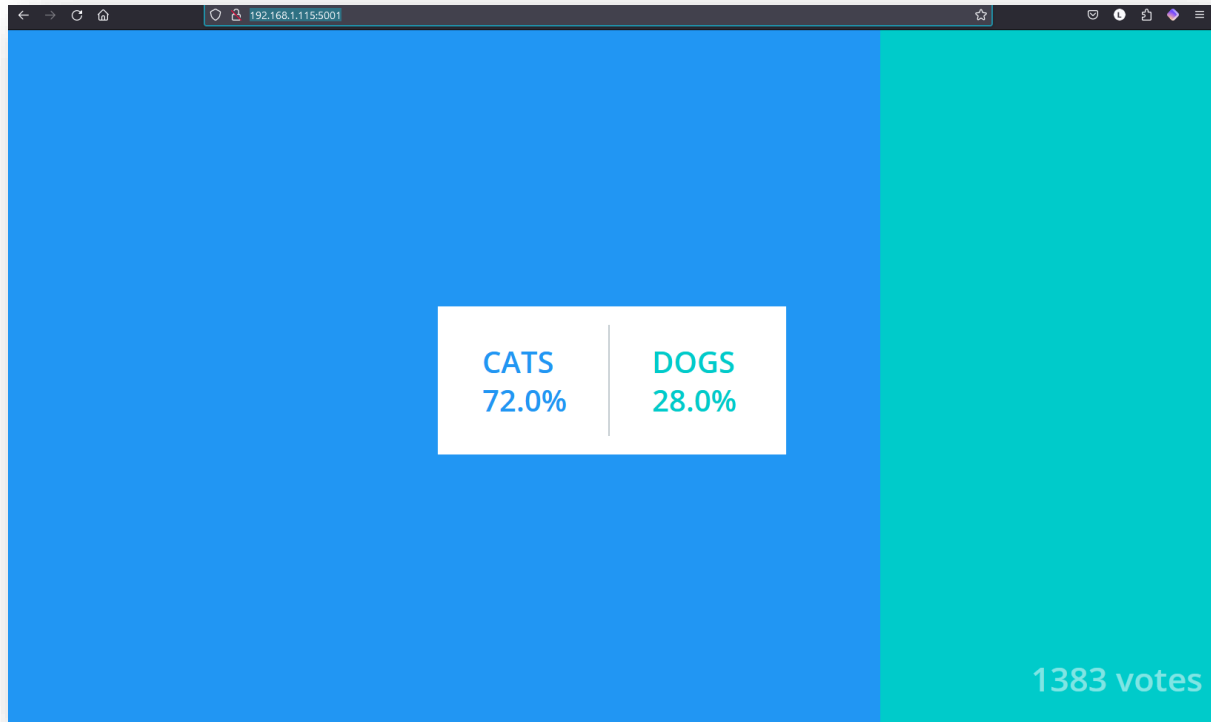
Une version automatisée de ces commandes existe également dans le [Makefile](#) de ce même dossier courant. Il a pour simple rôle d'accélérer le déploiement mais il exécute sensiblement les mêmes commandes.

Après cette procédure, nous pouvons vérifier que tous nos conteneurs sont bien lancés grâce à la commande « docker ps ».

```
Every 2.0s: docker ps
demo-vm: Mon Dec 18 16:47:19 2023
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
155df0a9a564	samlotus/virtu-worker:cmd	"dotnet Worker.dll"	44 seconds ago	Up 43 seconds		worker
1a3b41489b66	samlotus/virtu-result:cmd	"/usr/bin/tini -- no..."	44 seconds ago	Up 43 seconds (healthy)	127.0.0.1:9229->9229/tcp, 0.0.0.0:5001->80/tcp, :::5001->80/tcp	result
2bd3d0e0eb03	redis	"docker-entrypoint.s..."	45 seconds ago	Up 44 seconds (healthy)	0.0.0.0:6379->6379/tcp, :::6379->6379/tcp	redis
be5f9f140073	postgres:15-alpine	"docker-entrypoint.s..."	50 seconds ago	Up 48 seconds (healthy)	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	db
5cbe552f80c4	samlotus/virtu-vote:cmd	"unicorn app:app -b..."	59 seconds ago	Up 57 seconds	0.0.0.0:5002->80/tcp, :::5002->80/tcp	vote

Enfin, nous pouvons nous rendre sur l'URL de notre application, ici <http://192.168.1.115:5002> pour voter et <http://192.168.1.115:5001> pour les résultats du vote. **Cette IP dépend de l'IP de votre hôte docker.**



Cats vs Dogs!

CATS ☒

DOGS ☐

(Tip: you can change your vote)

Processed by container ID
5cbe552f80c4

On constate bien que les deux pages fonctionnent correctement et que notre application est fonctionnelle.

Maintenant reproduisons les mêmes étapes avec la version « automatisée » de notre application, c'est-à-dire utilisant docker compose. Cette fois-ci, nous rentrons dans le dossier src/comp qui contient les sources de la version compose.

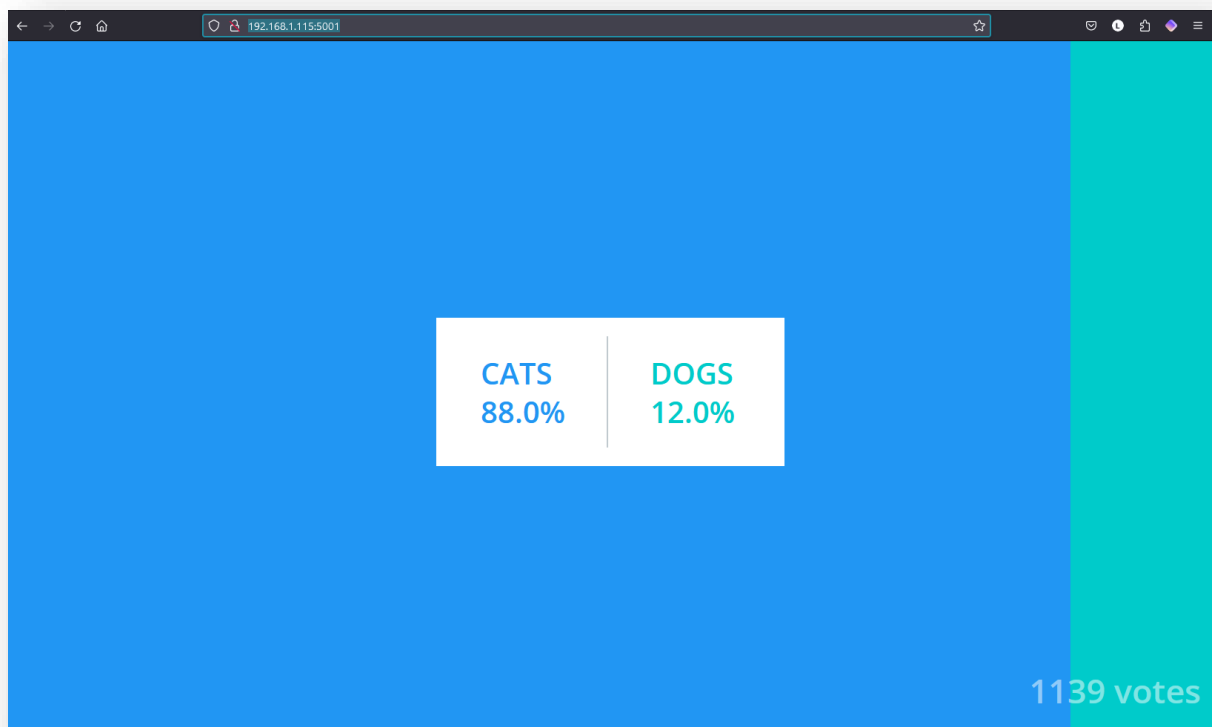
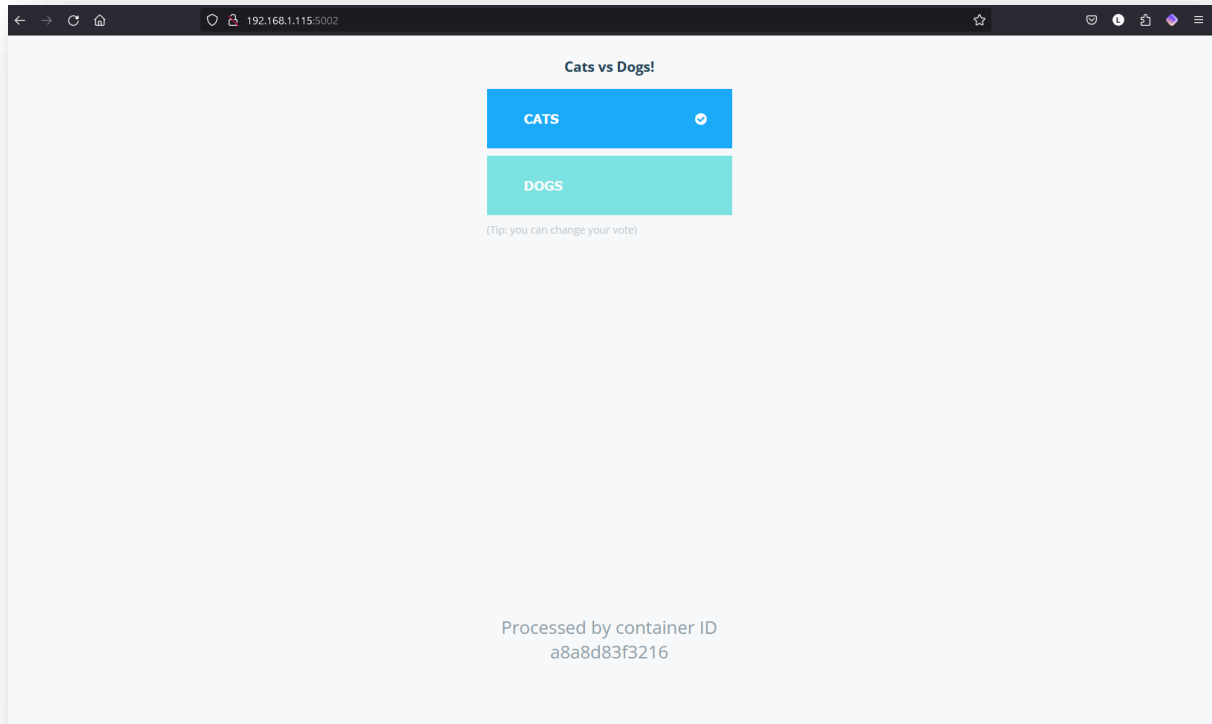
Il nous suffit simplement d'exécuter la commande « docker compose up --build -d » spécifiée dans la procédure de lancement et tout le reste se déroule de manière automatisée.

```
root@demo-vm:~/virtu-project-td49/src# cd comp
root@demo-vm:~/virtu-project-td49/src/comp# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
root@demo-vm:~/virtu-project-td49/src/comp# docker compose up --build -d
[+] Running 23/24
  db 14 layers [#####]      0B/0B      Pulling      7.6s
    ✓ 619014d83c02 Pull complete      0.7s
    ✓ 7ec0fe6664f6 Pull complete      0.4s
    ✓ 9ca7ba8f7764 Pull complete      0.3s
    ✓ 9e1155d037e2 Pull complete      0.7s
    ✓ febcfb7f8870 Pull complete      0.9s
    ✓ 8c78c79412b5 Pull complete      4.1s
    ✓ 5a35744405c5 Pull complete      1.1s
    ✓ 27717922e067 Pull complete      1.2s
    ✓ 36f0c5255550 Download complete    2.0s
    ✓ dbf0a396f422 Download complete    1.4s
    ✓ ec4c06ea33e5 Download complete    1.6s
    ✓ e8dd33eba6d1 Download complete    1.8s
    ✓ 51c81b3b2c20 Download complete    1.9s
    ✓ 2a03dd76f5d7 Download complete    2.1s
  redis 8 layers [#####]    0B/0B      Pulled      4.8s
    ✓ 661ff4d9561e Already exists      0.0s
    ✓ 963a98d2b6c2 Pull complete      2.2s
    ✓ dddcc6acb2ed Pull complete      2.8s
    ✓ ff0ac71727e7 Pull complete      2.4s
    ✓ 7959e5ad61f4 Pull complete      2.8s
    ✓ d767cfa2fc09 Pull complete      2.7s
    ✓ 4f4fb700ef54 Pull complete      2.8s
    ✓ 64281ae03186 Pull complete      3.0s
```

Notre « docker ps » nous affiche bien que tous les conteneurs sont lancés.

```
root@192.168.1.115
=> => exporting layers      0.0s
=> => writing image sha256:59053057ae560a236af2fa0e588e3c411fc79b1077a26f5b86e7cc2f064238f4 0.0s
=> => naming to docker.io/samlotus/virtu-seed-data:comp 0.0s
=> CACHED [worker stage-1 2/3] WORKDIR /app 0.0s
=> [worker stage-1 3/3] COPY --from=build /app . 0.0s
=> [worker] exporting to image 0.0s
=> => exporting layers      0.0s
=> => writing image sha256:69bb4903c1af575298b811afc1b03cdf88db49fe36c9279a3d27fc9056be513 0.0s
=> => naming to docker.io/samlotus/virtu-worker:comp 0.0s
[+] Running 9/9
  ✓ Network comp_front-tier      Created      0.1s
  ✓ Network comp_back-tier      Created      0.1s
  ✓ Volume "comp_db-data"      Created      0.0s
  ✓ Container comp-db-1      Healthy      0.0s
  ✓ Container comp-redis-1      Healthy      0.0s
  ✓ Container comp-worker-1      Started      0.0s
  ✓ Container comp-vote-1      Healthy      0.0s
  ✓ Container comp-result-1      Started      0.0s
  ✓ Container comp-seed-data-1      Started      0.0s
root@demo-vm:~/virtu-project-td49/src/comp# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED    STATUS    PORTS    NAMES
a8a8d83f3216   samlotus/virtu-vote:comp  "gunicorn app:app -b..." 51 seconds ago  Up 49 seconds (healthy)  0.0.0.0:5002->80/tcp, :::5002->80/tcp  comp-vote-1
c4198dbd3712   samlotus/virtu-result:comp  "nodemon --inspect=0..." 51 seconds ago  Up 44 seconds            127.0.0.1:9229->9229/tcp, 0.0.0.0:5001->80/tcp, :::5001->80/tcp  comp-result-1
b79ed3ce1b17   samlotus/virtu-worker:comp  "dotnet Worker.dll"        51 seconds ago  Up 44 seconds            comp-worker-1
ec144d686413   postgres:9.4  "docker-entrypoint.s..." 51 seconds ago  Up 50 seconds (healthy)  5432/tcp  comp-db-1
root@demo-vm:~/virtu-project-td49/src/comp#
root@demo-vm:~/virtu-project-td49/src/comp#
root@demo-vm:~/virtu-project-td49/src/comp#
```

Nous pouvons alors vérifier que notre application fonctionne correctement en nous rendant sur les mêmes URLs que précédemment et nous constatons que tout fonctionne correctement.



Docker Hub

Toutes les images customisées que nous avons utilisées dans ce projet peuvent être retrouvées sur Docker Hub (voir [ici la liste des images](#) dans notre dépôt Github).

Conclusion

En conclusion, ce projet a été une plongée enrichissante dans le monde complexe et interconnecté des applications distribuées. En intégrant diverses technologies telles que Python, Node.js, .NET, Redis et Postgres, nous avons conçu une architecture complète orchestrée par Docker Compose. L'utilisation de conteneurs a démontré son efficacité dans la gestion des dépendances, la cohérence du déploiement et la facilité de mise à l'échelle.

Le déploiement sur une infrastructure VMware EXSI a mis en lumière l'importance de l'orchestration dans des environnements réels, tout en soulignant la nécessité d'une collaboration d'équipe bien organisée. La documentation claire, les fichiers Dockerfile et docker-compose.yml ont joué un rôle crucial dans la cohérence du processus de développement.

Le recours à Docker Compose a particulièrement marqué les esprits, soulignant son rôle central dans la simplification de la gestion et du déploiement de l'application. Cette expérience renforce notre compréhension des avantages pratiques de l'orchestration de conteneurs dans le développement d'applications distribuées. En somme, ce projet a été une exploration approfondie, combinant théorie et pratique, et nous a permis d'acquérir des compétences précieuses dans la conception et le déploiement d'applications distribuées à grande échelle.