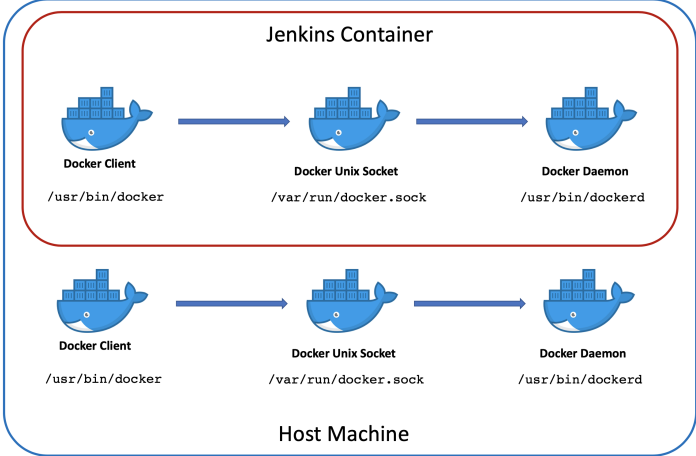
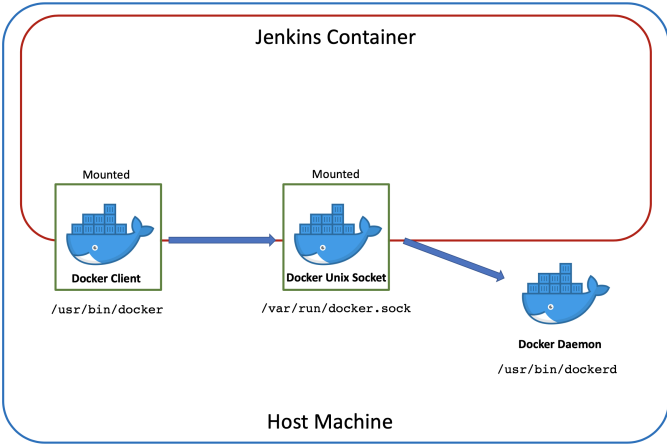


Discussion à propos de la solution privilégiée par l'opinion de Jenkins Officiel

Parmis les alternatives possibles pour accéder à un engine Docker à partir d'un Jenkins Dockerisé

Solutions Possibles	Avantage	Inconvénients
<p>Solution 1:</p> <p>Consiste à installer les trois composants directement à l'intérieur du conteneur Jenkins</p> 	<p>processus de docker sont généralement isolés</p>	<p>Cette solution nécessite de donner un accès privilégié au conteneur du docker.</p> <p>Une nouvelle image doit être construite.</p> <p>Le point d'entrée de l'image Jenkins de base est écrasé, car il n'y a pas de moyen (non super-ultra hacky) d'étendre le point d'entrée de l'image de base.</p> <p>L'installation d'un docker dans un docker n'est généralement pas recommandée.</p>
<p>Solution 2:</p> <p>Consiste à monter le client docker et un socket unix dans un conteneur Jenkins</p> 	<p>Peut travailler directement avec l'image de base Jenkins</p> <p>Le conteneur Jenkins ne nécessite pas d'accès privilégié</p> <p>Il semble que ce soit la solution la plus courante sur Internet</p>	<p>Pas d'isolement entre l'hôte et le conteneur Jenkins</p> <p>Le socket docker peut ne pas avoir les bonnes autorisations dans le conteneur Jenkins</p>

<p>Solution 3:</p> <p>Création de l'image officielle du docker-in-docker (dind) -</p> <p>Consiste à monter le client docker dans le conteneur Jenkins et de l'utiliser pour se connecter à la socket TCP</p>	<p>Peut travailler directement avec l'image de base Jenkins</p> <p>Isolement entre l'hôte et le conteneur Jenkins</p>	<p>Nécessité de faire fonctionner un conteneur supplémentaire</p> <p>Nécessité d'accorder un accès privilégié à la dind. * Toutefois, cela vaut mieux que de l'accorder directement au conteneur Jenkins.</p>
<p>Solution officielle</p> <p>J'ai testé la méthode du Dockerfile sans la méthode Dind qui ne la nécessite pas juste pour l'utilisation de Jenkins</p>	<p>La solution officielle sera toujours la meilleure d'après l'auteur tiuweehan.</p> <p>Travaille avec 2 images et 2 container : l'image de base de jenkins contenant le plugin Blueocean – 1 docker DinD</p> <p>plusieurs manières de lancer le tout: docker run, dockerfile, docker-compose</p>	<p>Nécessité de faire fonctionner un conteneur supplémentaire (inconvenient idem solution3)</p> <p>L'exécution de Docker dans Docker nécessite actuellement un accès privilégié pour fonctionner correctement (inconvenient idem solution3)</p>

Mise en place et remarque:

Pour la solution 1: s1-Build-a-new-Jenkins-image-with-Docker-installed

Build de l'image jenkins-docker à **1.18 GB**

Build image

`docker build /path/to/Dockerfile --tag jenkins-docker`

```
# Run container
Docker run --rm -d -v ./jenkins_home:/var/jenkins_home -p 8080:8080 -p
50000:50000 --privileged jenkins-docker
1bf80297aab2040bd9ffb9b7fe6764cd7105614033ba103e982794c72c24772b

docker exec -it 1bf /bin/bash
# récupération du mdp:
cd /var/jenkins_home/secrets/
cat initialadminpassword
ThePasswordIsDisplayed
```

Pour la solution 2: s2-Mount-the-host-docker-unix-socket-onto-the-Jenkins-container

docker-compose up fait le build et créer 1 image et 1 container
 avantage qu'il crée le dossier Jenkins

Il faut quand même sauter dans le container pour obtenir le mdp pour débloquent Jenkins

```
docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS            PORTS              NAMES
71fa6fcaf474       jenkins/jenkins:lts  "/sbin/tini -- /usr/..."  2 minutes ago
Up 2 minutes      0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000/tcp  jenkins

docker exec -it 71f /bin/bash
# récupération du mdp:
root@71fa6fcaf474:/# cd /var/jenkins_home/secrets
root@71fa6fcaf474:/var/jenkins_home/secrets# cat initialAdminPassword
b20cedc6312e4d3bb8fd33525f43e139
```

Pour la solution 3: s3-Run-the-official docker-in-docker-image

docker-compose up fait le build et créer 1 image jenkins/jenkins: latest (566MB) et 1 container

Les avantages sont qu'il:

- Crée le dossier Jenkins
- Génère le mot de passe dans le build de manière à ce que l'on ne doit pas se connecter sur le container en ligne de commande
- utilise l'image Dind qui ne pèse quasi rien (258.02 MB)
- ne faut utiliser qu'une seule commande pour le lancer: docker-compose up
- il utilise les volumes pour la persistance de données, on garde donc nos données

Pour la solution officielle: s4-how-to-install-jenkins-on-docker-official

<https://www.jenkins.io/doc/book/installing/docker/#on-windows>

J'ai juste executer la commande pour créer le network : `docker network create jenkins`
 ensuite:

J'ai passé l'étape du docker 3. le RUN pour l'installation du DinD car elle ne m'a pas parue nécessaire à Jenkins. Je l'ai testée et je me retrouve avec 2 images et 2 containers a la place d'une image et d'un container. Au vue de cela, nous avons +- le même résultat que dans la solution3 mais avec des images plus lourdes.

Voici le code: # Run a docker:dind Docker image

```
docker run --name jenkins-docker --rm --detach ^
--privileged --network jenkins --network-alias docker ^
--env DOCKER_TLS_CERTDIR=/certs ^
```

```
--volume jenkins-docker-certs:/certs/client ^
--volume jenkins-data:/var/jenkins_home ^
docker:dind
```

le Dockerfile:

<https://www.jenkins.io/doc/book/installing/docker/#downloading-and-running-jenkins-in-docker>

Du point 4. Customise official Jenkins Docker image, by executing below two steps: ...

```
FROM jenkins/jenkins:2.263.4-lts-jdk11
USER root
RUN apt-get update && apt-get install -y apt-transport-https \
    ca-certificates curl gnupg2 \
    software-properties-common
RUN curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -
RUN apt-key fingerprint 0EBFCD88
RUN add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/debian \
    $(lsb_release -cs) stable"
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
RUN jenkins-plugin-cli --plugins blueocean:1.24.4
```

docker build -t myjenkins-blueocean:1.1 .

```
docker run --name jenkins-blueocean --rm --detach ^
--network jenkins --env DOCKER_HOST=tcp://docker:2376 ^
--env DOCKER_CERT_PATH=/certs/client --env DOCKER_TLS_VERIFY=1 ^
--volume jenkins-data:/var/jenkins_home ^
--volume jenkins-docker-certs:/certs/client:ro ^
--publish 8080:8080 --publish 50000:50000 myjenkins-blueocean:1.1
db7157675bcc9f1adfad4bbfeb0e9167262569bb303225168b33ea5559e8a7a0
```

```
docker exec -it jenkins-blueocean bash
```

récupération du mdp:

```
jenkins@db7157675bcc:/$ cd /var/jenkins_home/secrets/
jenkins@db7157675bcc:~/secrets$ cat initialAdminPassword
533f0a50813149c080b8eafc2b300ed0
```

Dans ce cas:

- il ne crée pas de dossier Jenkins à l'endroit courant
- Il crée 1 seul container à partir de l'image jenkins/jenkins:2.263.4-lts-jdk11 * contrairement aux autres solutions
- installe les OS Linux et
- utilise 2 images: le client Docker ainsi que le plugin blue-ocean qui fait **1.1GB**
- Il faut sauter dans le container pour aller chercher le mot de passe pour déverrouiller Jenkins

* de cette image, le projet Jenkins met à jour le système d'exploitation de base et la version Java utilisés dans les images jenkins/jenkins:latest et jenkins/jenkins:lts. La mise à jour remplace l'OpenJDK 8u242 par l'AdoptOpenJDK 8u282 et remplace la Debian 9 ("Stretch") par la Debian 10 ("Buster").

Si l'on stop les containers on perd le contenu ? Non ils restent stockés sur les volumes.

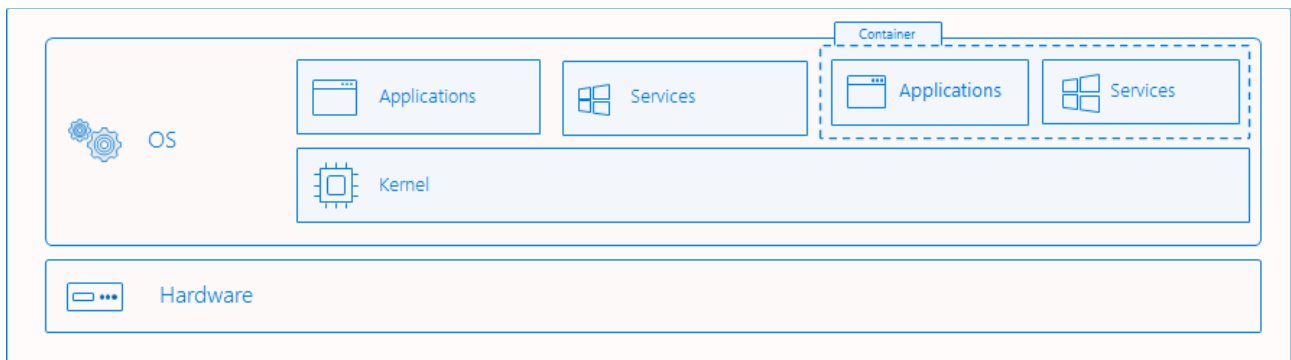
Discussion en ce qui concerne les inconvénients:

En ce qui concerne **l'isolation**:

Sur le site officielle il est écrit: " Docker est une plate-forme permettant d'exécuter des applications dans un environnement isolé appelé "conteneur" ..."

Isolement des processus

Avec l'isolation de processus, plusieurs instances de conteneurs s'exécutent simultanément sur un hôte donné. Lorsqu'ils fonctionnent dans ce mode, les conteneurs partagent le même noyau avec l'hôte ainsi qu'entre eux.



En ce qui concerne: **la nécessité d'accorder un accès privilégié à la dind**:

Dans la solution officielle, l'exécution de Docker dans Docker **nécessite actuellement un accès privilégié** pour fonctionner correctement (inconvenient idem de la solution3).

voir doc: <https://www.jenkins.io/doc/book/installing/docker/> point 4

```
docker run \  
  --name jenkins-docker \  
  --rm \  
  --detach \  
  --privileged \  
  --
```

Cette exigence peut être assouplie avec les nouvelles versions du noyau Linux.

https://www.trendmicro.com/en_us/research/19/1/why-running-a-privileged-container-in-docker-is-a-bad-idea.html

Un cas d'utilisation d'un conteneur privilégié est l'exécution d'un démon Docker à l'intérieur d'un conteneur Docker ; un autre est celui où le conteneur nécessite un accès matériel direct.

Aujourd'hui, il existe différents cas d'utilisation pour l'exécution de conteneurs privilégiés, comme l'automatisation des tâches d'intégration et de livraison continues (CI/CD) dans le serveur d'automatisation open-source Jenkins.

Cependant, leurs utilisation n'est pas nécessairement sécurisée et leurs utilisation, sans les sécuriser, a de graves conséquences. L'utilisation d'un conteneur avec un drapeau privilégié permet aux

équipes internes d'avoir un accès critique aux ressources de l'hôte - mais en abusant d'un conteneur privilégié, les cybercriminels peuvent également y accéder.

Comme le conteneur privilégié est créé en raison de la nécessité d'améliorer les autorisations, il y a de fortes chances qu'un attaquant puisse exécuter du code en tant que root. Cela implique qu'un attaquant sera en mesure d'exécuter l'hôte racine complet avec toutes les capacités disponibles.

Conclusion: Il va de soi que la meilleure solution est d'utiliser l'image DinD

En ce qui concerne: **les sockets:**

Par défaut, vous êtes censé utiliser des sockets TCP pour communiquer avec vos applications fonctionnant dans Docker. - **solution 3 et solution 4-**

Mais que faire si vous voulez utiliser des sockets Unix à la place ? - **solution 1,2, 4-**

La réponse est : vous faites en sorte que l'application crée le fichier socket dans un volume et lui attribue les autorisations appropriées.

Vous pouvez l'utiliser pour votre serveur web pour parler à votre application ou pour des communications entre conteneurs. **La partie la plus délicate est de définir les autorisations appropriées sur la socket.**

<https://www.jujens.eu/posts/en/2017/Feb/15/docker-unix-socket/>

Conclusion

A titre personnel, la solution 1 ne mérite pas que l'on s'y attarde car on l'utilise une image beaucoup trop lourde. De même que la solution 4 qui est de la doc officielle: l'image de jenkins augmenté du plugin Blue-ocean à une taille supérieure à 1G. La solution officielle m'a paru très semblable à la solution 3, et je pense que peut-être elle prévaut cette solution 3 comparée à d'autres .

Mais je trouve qu'avec un container et 1 image en +, de + l'image du plugin est très/trop lourde.

Ma préférence va clairement la **solution 3**

parcequ'elle offre beaucoup + d'avantage que les autres , il y a moins de configurations a faire
Elle utilise les bonnes pratique aussi, image DinD, pas trop lourde

Je pense que Jenkins s'appuie lui-même sur cette solution

Et un dernier tableau de comparaison:

	isolation	Socket TCP	Socket Unix	Dind	Volumes	Accès pivilègié
solution 1	oui		oui		non	oui
solution 2	non		oui		oui	non
solution 3	oui	oui		oui	oui	oui
Solution 4 off dockerfile		oui		non	oui	oui
Solution off avec le RUN			oui	oui	oui	oui