POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

# Rapport de projet

## Nuage pour site web


Tuteur : Xavier Redon
Thomas Vantroys

Guillaume VILLEMONT

5ème année cycle ingénieurs
Informatique, Microélectronique et Automatique

Année universitaire 2016/2017

# Sommaire

# Introduction

## Contexte

La facilité d'accès à internet et la démocratisation des services en ligne et du cloud font que de plus en plus de gens choisissent de confier leurs données et leurs logiciels à des compagnies hébergeant ces services. Ces fournisseurs utilisent souvent des solutions techniques opensource mais rognent de plus en plus sur la vie privée de leurs utilisateurs. Il est tout à fait envisageable aujourd'hui d'auto héberger son site web et son cloud en utilisant les mêmes solutions que celles en production.

Dans le cadre de mon projet de fin d'étude en IMA, j'ai choisi d'étudier les solutions pour héberger des applications sur un serveur sécurisé avec des ressources limitées. La principale utilisation sera ici l'hébergement de site web pour l'école et les étudiants.

## Cahier des charges

Ce projet est destiné à héberger de nombreux sites web sur un serveur pour des utilisateurs n'ayant pas nécessairement de connaissances en informatique.

- Accessibilité :
  - depuis internet, via une adresse ipv4 ou ipv6, pour ne pas consommer trop d'adresse sur le réseau
  - via une interface graphique facile d'utilisation, doit pouvoir être utilisé sans connaissance approfondie :

    création d'un nom de site web

    modification des pages du site

    suppression du site

- Production :
  - utilisation optimale des ressources du système, doit pouvoir être mise en production sur tout type de machine, donc peu gourmand en ressource.
  - robustesse, fiabilité des logiciels, logiciel stable.
  - extensibilité, ajout et suppression de sites facilement, gestionnaire de conteneurs approprié.

- Sécurité :
  - réduire le nombre de ports exposés, idéalement deux ports, 80 et 443.
  - intégration avec le système hôte, Linux
  - exécution avec le minimum de privilège, minimiser les dégâts en cas d'attaques.

# Travaux effectués

## Choix technologiques

Il existe plusieurs solutions pour faire tourner notre serveur web et nos applications. La plus naïve car la plus simple est d'installer chaque logiciel directement sur la machine hôte. Cela garanti le bon fonctionnement de notre serveur web mais nous devons alors nous restreindre dans notre choix de logiciel. En effet chaque application va utiliser les mêmes ressources du système et partager les mêmes fichiers et mémoires.

La deuxième, la virtualisation, est une solution technique qui permet de faire tourner plusieurs OS comme si il y avait plusieurs machines physiques, mais le tout sur une seule et unique machine. Les deux grands types sont la virtualisation hardware et software. Le premier est réalisé grâce à des composants matériels auxquels nous n'avons pas accès ici. Le deuxième, plus envisageable, utilise un hyperviseur logiciel installé sur l'os hôte et permet alors de faire tourner plusieurs OS par-dessus l'hôte. Cependant la virtualisation impacte énormément l'utilisation des ressources. Chaque OS virtuel demande plusieurs secondes pour démarrer. Le nombre d'OS virtualisables peut s'estimer à moins d'une dizaine par cœur de processeur, soit moins d'une quarantaine d'OS sur les machines actuelles 4 cores.

Une autre solution de plus en plus utilisée est la 'conteneurisation' (« containerization » en anglais). C'est cette méthode qui sera utilisée pour le serveur.

## La conteneurisation

La conteneurisation est une technologie des années 2000 introduite avec LXC, mais fortement popularisée depuis 2013 par Docker, un gestionnaire de conteneur, rendant beaucoup plus simple la gestion des applications. Un conteneur se présente sous forme d'une image pouvant regrouper des données, des bibliothèques ou des applications entières. Le conteneur se lance comme un processus et utilise le système hôte sans hyperviseur contrairement aux machines virtuelles. Chaque conteneur est indépendant. L'application qu'il héberge peut donc être démarrée, mise à jour, ou arrêtée au besoin. Cela rend la virtualisation d'application plus légère que la virtualisation d'OS et permet de faire tourner des centaines de conteneurs. Cette technologie tend à se démocratiser car il est ainsi aisé de fournir le même environnement pour le développement que pour la production.

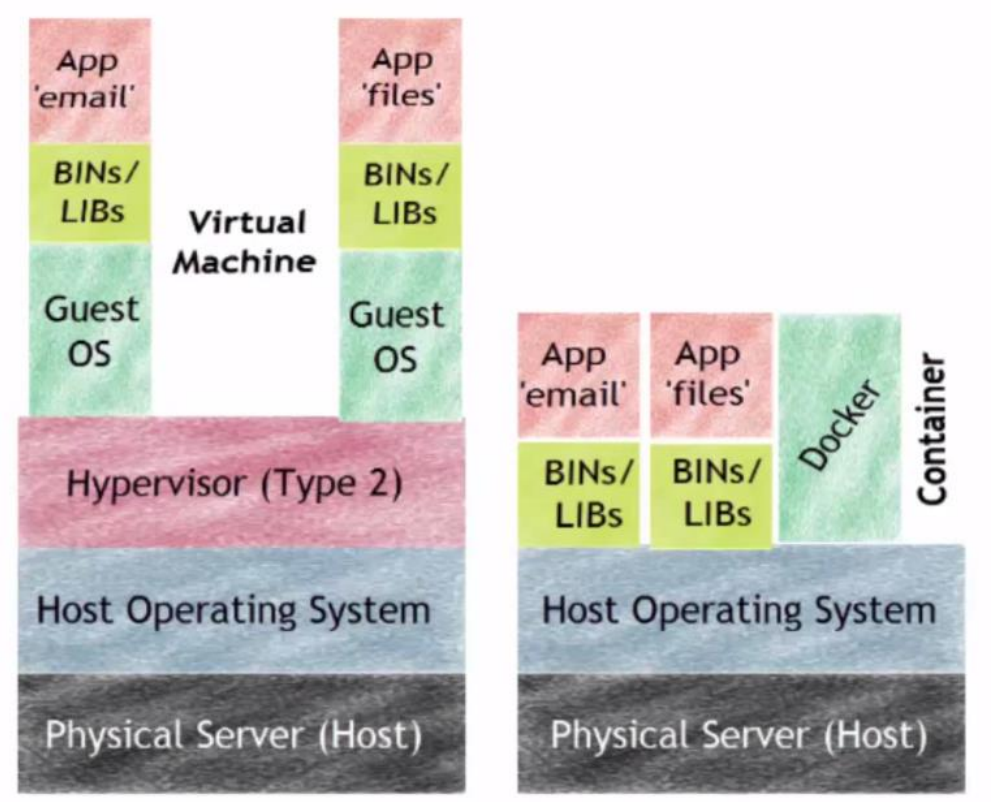



*Illustration comparatif d'architecture de machine virtuelle et conteneur*

### Docker vs Rocket

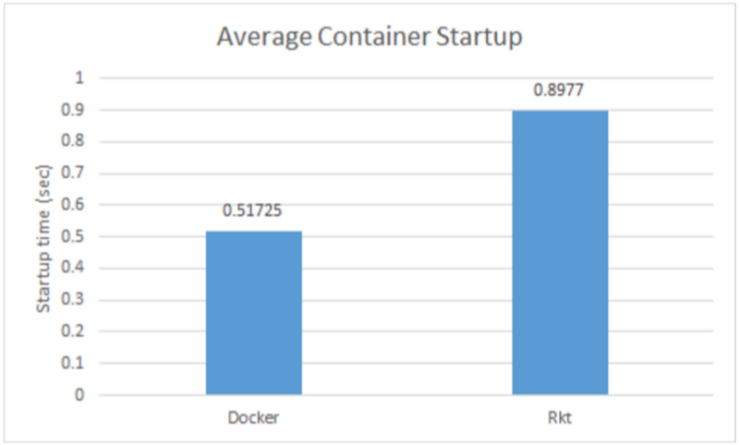
Le choix par défaut est souvent Docker[1]. Commencé en 2013, le projet a pris de l'ampleur grâce à sa facilité d'utilisation. Cependant de nombreuses autres méthodes de virtualisation existaient déjà et d'autres se sont développées par la suite. C'est le cas de Rocket, projet soutenu par CoreOS, qui se pose en concurrent direct de Docker. Rocket se veut plus 'Linux-friendly' avec une meilleure sécurité que Docker[2].

Voici un aperçu des principaux logiciels de virtualisation de container et leurs caractéristiques :

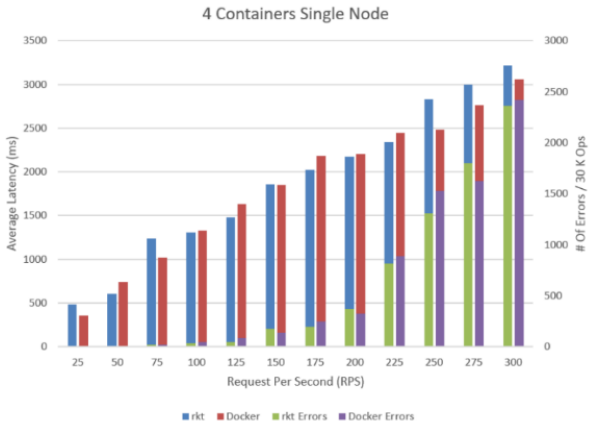
| | Utilisation d'un daemon centralisé | Comptabilité des images | Installation facile sur le serveur école (Debian) | Utilisation principale | Sécurité |
|---|---|---|---|---|---|
| **Docker**  | Oui | docker images | Dépôt officiel | applications, commandes simples, configuration facile | Téléchargement et vérification automatiques, usage en tant que root |
| **Rocket**  | non | docker images, OCI images | Dépôt unstable ou installation .deb | applications, commandes simples, configuration facile | Téléchargement et vérification automatiques |
| **runC**  | non | OCI images | Dépôt officiel (installé avec Docker v1.11) | applications, commandes complexes, compréhension détaillé du système | |
| **LXC/LXD**  | oui | OCI images, full system operating images | Dépôt officiel | OS complets, commandes simples, configuration complexes | Téléchargement et vérification automatiques (uniquement pour OS complet) |
| **systemd-nspawn** | non | OCI images, full system operating images | Inclus avec systemd | applications et OS complets | |

[1] http://www.coscale.com/blog/docker-usage-statistics-increased-adoption-by-enterprises-and-for-production-use
[2] https://coreos.com/blog/rocket.html

Le premier critère de choix est le type de programme. En cherchant la simplicité, un programme tel que Docker ou LXC, avec un daemon et un programme client, est plus compliqué à gérer. Le deuxième critère est la compatibilité avec les standards. Mon choix s'est donc orienté vers Rocket qui présente tous les avantages sur le papier. En pratique il s'intègre mieux avec les systèmes Linux : il est notamment possible d'écrire des fichiers de lancement de conteneurs en bash, là où Docker utilise sa propre syntaxe. Le paramétrage est plus fin et permet une plus grand liberté mais est aussi plus complexe. Rocket cherche à respecter l'idéologie Unix en ne faisant qu'une tâche mais en la faisant bien. Mais Rocket, en tant que jeune projet, ne bénéficie pas d'une grand communauté (une centaine de contributeur contre un millier pour Docker[3]). Cependant, en termes de performance, Rocket fait déjà aussi bien de Docker. Il est capable de gérer autant de conteneur avec les mêmes ressources, sans produire plus d'erreurs. Seul point négatif, le démarrage des conteneurs est presque deux fois plus long :




*Temps de démarrage d'un conteneur Docker / Rocket*




*Comparaison des erreurs en fonction du nombre de conteneur Plus de détails [4]*

[3]https://shivammaharshi.wordpress.com/2016/08/16/docker-vs-rkt-benchmarking-community-documentation-support/
[4] https://shivammaharshi.wordpress.com/2016/08/16/docker-vs-rkt-benchmarking-performance-benchmarks/

# Developpement

## Installation

Rocket s'installe rapidement car disponible dans les dépôts officiels de Debian sid (unstable). La commande se résume donc à :

```
sudo apt-get install rkt
```

On importe ensuite différents conteneurs pré-existant sur les dépôts de Docker et CoreOS, à savoir hub.docker et quay.io :

```
rkt fetch –-insecure-options=image docker://apache
rkt fetch quay.io/httpd
```

Sur le dépôt Docker les images ne sont pas signées avec des clés pgp. Or rkt cherche à vérifier les images par défaut, on précise donc de ne pas vérifier avec l'option insecure. Pour vérifier les images de CoreOS, il a fallu passé la configuration du proxy en variable globale comme pour apt :

```
export https_proxy=proxy.polytech-lille.fr:3128
```

Il est ensuite possible de lancer le conteneur :

```
rkt run docker://apache
```

Mais il faut savoir que toutes les données d'un conteneur sont éphémères et disparaissent à son arrêt. Il est donc indispensable de monter des volumes de l'hôte vers le conteneur. Les volumes sont des dossiers partagés entre les deux systèmes, hébergés sur le système hôte, ils conservent les configurations intactes.

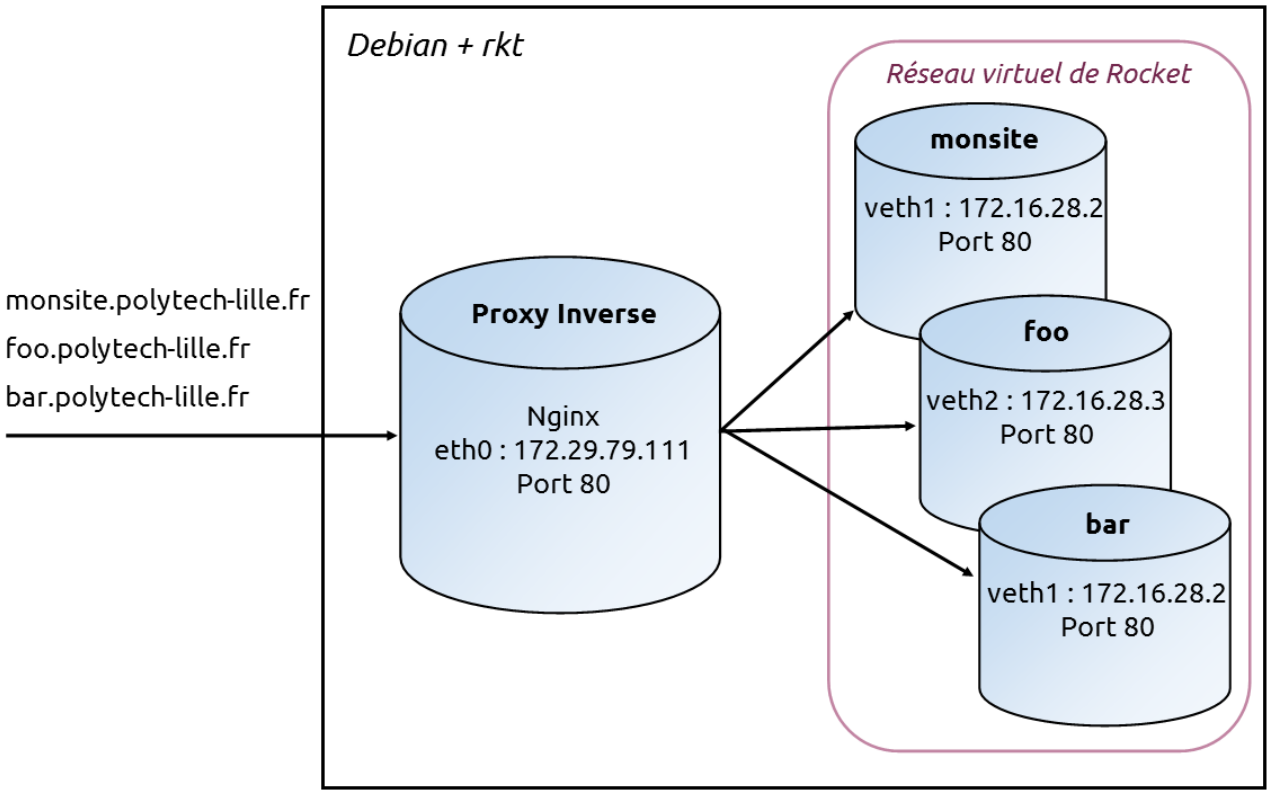
La standardisation de systemd comme système d'initialisation de nombreuses distributions Linux m'a poussé à utiliser rkt avec systemd. Cela permet d'avoir accès aux logs de chaque container très simplement. La commande pour lancer un conteneur devient :

```
systemd-run --slice=machine rkt --debug=true run –volume config,kind=host,source=$MON_DOSSIER
docker://apache --mount volume=config,target=/var/www/html --net=default:IP=$ADDR_IP
```

On peut ainsi créer un grand nombre de conteneurs dans un même réseau virtuel, ici le réseau default. Il est aussi possible de créer des réseaux pour séparer les applications. Spécifier le réseau nous permet surtout de préciser l'adresse IP à donner au conteneur. C'est la grande différence avec Docker qui assigne ses IP de manières aléatoires dans le réseau.

## Proxy inverse

Le proxy inverse sert ici à rediriger le trafic entrant vers le conteneur virtualisant l'application demandée. On utilise Nginx, pour sa légèreté et sa facilité à modifier sa configuration par simple ajout en fin de fichier. Nginx tourne lui-même dans un conteneur puisqu'il n'y a aucun argument pour ne pas le faire et le conteneur isole l'application ce qui réduit la surface de potentielles attaques. Tout le serveur est géré par rkt, ainsi on ne craint pas de mise à jour régressive qui pourrait endommager notre proxy.




*Architecture des conteneurs et du réseau*

## Automatisation

L'intérêt premier de ce serveur est de fournir une interface graphique, idéalement via une page web, pour créer simplement un conteneur avec un serveur web et ses données. Pour donner accès simplement aux commandes assez longues de Rocket, des scripts sont disponibles. Au nombre de deux, un pour la création un pour la suppression (cf. Annexes), ils permettent de gérer les utilisateurs, les données html, les conteneurs et la configuration réseau. Un utilisateur a la possibilité de créer plusieurs containers et donc plusieurs sites.

Les scripts utilisent une base de données pour garder une trace des conteneurs en exécution. AU simple format texte, elle présente les paramètres suivant :

```
#<numéro> <nom_de_sous_domaine> <adresse_IP> <ID_container> <type> <volume>
```

Le nom de sous domaine est choisi par l'utilisateur. L'adresse IP est incrémentée suivant la valeur de la précédente. L'ID du conteneur est unique et aléatoire, généré par rkt. Le type de conteneur spécifie quel image est utilisé (Apache, Nginx, Apache+Perl). Le volume indique où les données html sont montés sur le conteneur.

Un sous-domaine est créé automatiquement dans la configuration du proxy inverse (/etc/nginx/config.d/default.conf) en ajoutant une section 'server' :

```
#<ID_container>
server {
   listen      80;
   server_name <nom_de_sous_domaine>.p1nk.unic0rn.pw;

   location / {
      proxy_pass http://<adresse_IP>;
      index   index.html index.htm;
   }
}
```

On peut alors accéder directement à son site web qui se crée en moins de 5 secondes.

Un avantage de Rocket est que les volumes montés dans le conteneur sont dynamiques contrairement à Docker où les données sont simplement copiées. Ainsi on peut modifier son site web sur la machine hôte et les changements seront pris en compte par le conteneur, sans besoin de le redémarrer.

## Création de conteneurs

De nombreux conteneurs sont disponibles en ligne sur les bibliothèques du hub Docker et Quay.io. Créés par la communauté ou par les distributeurs officiels, ils sont nombreux et offrent beaucoup de possibilité. Néanmoins, un utilitaire de création de conteneur est présent sur le serveur. Cela permet, entre autre, de définir la base de son image sur laquelle tournera l'application, de choisir l'application, de modifier le port ouvert, de définir des points de montage par défaut, de combiner des logiciels et spécifier leur version. L'utilitaire utilisé ici est ACbuild (pour 'Another Container Build tool'). Un script regroupe déjà les commandes essentielles pour créer son conteneur personnalisé mais toutes les options ne sont pas prises en compte pour la création du site web. Notamment le port à ouvrir est défini par défaut sur le port 80 lors de la création d'un conteneur, forcer une autre valeur avec ACbuild sur un nouveau conteneur entrainera son non fonctionnement lors de sa mise en ligne.

La création de conteneur personnalisé reste anecdotique puisque la majorité des configurations classiques de serveur sont déjà disponible en ligne. Mais il reste important de pouvoir fournir ce service.

# Difficultés & évolutions possibles

## Difficultés

Au début de ce projet en janvier 2017, je n'avais aucune connaissance et expérience sur la technologie des conteneurs. Une grosse partie du travail a d'abord été de comprendre le fonctionnement. Puis j'ai réalisé quelques tests avec Docker sur une machine virtuelle. J'ai choisi d'utiliser Docker pour commencer car il reste simple d'utilisation et la documentation est très fournie, qu'elle soit officielle ou de la communauté. Cela m'a permis de me familiariser avec le vocabulaire et les commandes utilisées. Rocket bénéficie d'une bonne documentation officielle mais peu d'exemple concret.

Quelques noms de commandes ont changées mais l'idée restait la même. Cependant je me suis heurté à certaines erreurs qui n'étaient pas documentées. Je me suis aperçu que Rocket ne permet pas de monter plusieurs volumes hôte sur un conteneur. Ce qui, pour notre usage, nous empêche d'offrir une configuration personnalisée pour chaque serveur web. En effet nous devons fournir les données html du site web via un conteneur, mais il n'est alors plus possible de modifier les fichiers de configuration du conteneur hébergeant Apache ou Nginx.

Rocket ne permet pas non plus de redémarrer un conteneur. Un conteneur arrêté ne peut qu'être supprimé. Il a fallu prendre ça en compte dans les scripts.

## Interface graphique

Ce projet se destine à fournir une interface simple pour créer son site web sans connaissance en système Linux, web ou réseau. La solution envisagée est une page web proposant dans un onglet 'Création' les champs suivant : nom du site web, type du serveur, langage dynamique. Un sous menu 'Avancé' donnerait accès aux options du script de création de conteneur personnalisé via ACBuild. Dans un second onglet 'Suppression', un menu déroulant propose tous les sites de l'utilisateur. Un onglet d'aide pourrait aussi expliquer à l'utilisateur comment uploader et modifier les pages de son site. Idéalement, le dossier contenant les données html serait créé dans la session de l'élève. Modifier son site reviendrait à se connecter à sa session. Ce système implique qu'il soit possible de monter les dossiers utilisateurs sur le

serveur via ftp par exemple. Héberger les données sur le serveur directement est aussi tout à fait envisageable. Malheureusement, par manque de temps et de compétence, aucune interface graphique n'est disponible pour le moment. Les scripts fournissent néanmoins une ébauche des possibilités, via des questions/réponses en terminal.

## Améliorations

Le back-end peut aussi être améliorée. La gestion des utilisateurs et de l'adressage IP des conteneurs sont fonctionnelles mais pourraient être améliorées. Aucune vérification de la disponibilité du nom de domaine n'est réalisée. La suppression d'un conteneur entraine la libération de son adresse IP virtuelle dans le réseau Rocket. Ces adresses restent vacantes car l'assignement des IP est incrémentale et ne vérifie pas les disponibilités des adresses précédentes lors de la création d'un nouveau conteneur.

# Conclusion

Durant ce projet, j'ai mené un débat sur les solutions techniques disponibles pour monter un serveur hébergeant de nombreux sites web en limitant les ressources utilisées et offrant un système sécurisé. La solution retenue fût Rocket, un gestionnaire de conteneurs en fort développement soutenu par CoreOS. Le choix de la conteneurisation offre à la fois les avantages d'isolation et de sécurité des machines virtuelles, et tire aussi parti des ressources du système hôte.

Ce projet fut enrichissant sur de nombreux points. En effet la thématique aborde un enjeu d'actualité dans un monde de plus en plus connecté. De plus la diversité des solutions possibles m'a forcé à poser clairement les limites et la faisabilité du projet. L'utilisation de la conteneurisation, une technologie en plein développement, m'a de suite motivé et m'a finalement apporté de réelle compétence professionnelle.

Bien que le serveur ne soit pas encore prêt pour la production, le travail de synthèse mené ici permettra, je l'espère, de terminer ce projet et propose une alternative à ceux des années passées en offrant ainsi une réflexion sur les solutions disponibles.

# Bibliographie

[1] http://www.coscale.com/blog/docker-usage-statistics-increased-adoption-by-enterprises-and-for-production-use

[2] https://coreos.com/blog/rocket.html

[3] https://shivammaharshi.wordpress.com/2016/08/16/docker-vs-rkt-benchmarking-community-documentation-support/

[4] https://shivammaharshi.wordpress.com/2016/08/16/docker-vs-rkt-benchmarking-performance-benchmarks/

[5]https://jvns.ca/blog/2016/11/03/what-happens-when-you-run-a-rkt-container/

[6] https://pettigrew.rocks/2016/05/30/a-beginners-guide-to-rkt-containers/

[7] https://www.guillaume-leduc.fr/docker-comme-solution-de-virtualisation-theorie.html

[8] https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html

[9] https://bobcares.com/blog/docker-vs-rkt-rocket/

# Annexes

Script de création d'un nouveau conteneur :

```
#!/bin/bash

DB=/root/rkt/scripts/db.txt
CONFIG_PROXY=/root/rkt/nginx/config/conf.d/default.conf
DOSSIER_UTILISATEUR=/root/rkt/users

read -p "Quel est votre nom d'utilisateur :" USER

read -p "Selectionnez votre nom de domaine (.p1nk.unic0rn.pw) :" NOM

# Choisir son type de serveur
OPTIONS=("Nginx" "Apache" "Apache+PHP" "Apache+Perl")

echo "Selectionnez votre serveur :"
select SVR in "${OPTIONS[@]}"; do
   case $SVR in
      "Nginx" ) SERVER=nginx; TARGET='/usr/share/nginx/html'; break;;
      "Apache" ) SERVER=httpd; TARGET='/usr/local/apache2/htdocs'; break;;
      "Apache+PHP" ) SERVER=eboraas/apache-php; TARGET='/var/www/html'; break;;
      "Apache+Perl" ) SERVER=cloudposse/apache-perl; TARGET='/var/www/html'; break;;
   esac
done

# Assigner une adresse IP
nbr_site=`cat $DB | wc -l`
if [ $nbr_site -gt 1 ]; then
      nbr_site=$((nbr_site+1))
      ADDR_IP=172.16.28.$nbr_site
else
      nbr_site=2
      ADDR_IP=172.16.28.$nbr_site
fi

# Creation du dossier utilisateur servant les données (index.html, css...)
if [ -d "$DOSSIER_UTILISATEUR/$USER" ]; then
      if [ -d "DOSSIER_UTILISATEUR/$USER/$NOM" ]; then
            echo "Votre site existe déjà."
      else
            mkdir $DOSSIER_UTILISATEUR/$USER/$NOM
```

```
            fi
        else
            mkdir $DOSSIER_UTILISATEUR/$USER
            mkdir $DOSSIER_UTILISATEUR/$USER/$NOM
        fi

        # Lancement du serveur conteneur
        systemd-run --slice=machine rkt run --volume
html,kind=host,source=$DOSSIER_UTILISATEUR/$USER/$NOM docker://$SERVER --mount
volume=html,target=$TARGET --net=default:IP=$ADDR_IP
        sleep 4
        # Recupération de l'ID du conteneur
        ID=`rkt list | grep $ADDR_IP | cut -f1`
        # Mise à jour de la table de données
        echo "$nbr_site $NOM    $ADDR_IP        $ID     $SERVER $TARGET" >> $DB

        # Mise à jour de la config du reverse proxy Nginx
        echo "#$ID
        server {
           listen      80;
           server_name $NOM.p1nk.unic0rn.pw;

           location / {
              proxy_pass http://$ADDR_IP;
              index   index.html index.htm;
           }
        }" >> $CONFIG_PROXY
        # Actualisation du reverse proxy = redemerrage du conteneur
        #arrêt
        PROXY=`rkt list | grep nginx | sort -k6 | cut -f 1 | sed -n 1p`
        echo "Arrêt du proxy..."
        rkt stop $PROXY
        sleep 1
        #suppression
        echo "Suppression du proxy..."
        rkt rm $PROXY
        sleep 1
        #demarrage
        echo "Démarrage du proxy..."
        systemd-run --slice=machine rkt --debug=true run --net=host --volume
config,kind=host,source=/root/rkt/nginx/config docker://nginx --mount
volume=config,target=/etc/nginx/

        echo "Fait."
```

Script de suppression d'un conteneur et de ses données :

```
#!/bin/bash
DB=/root/rkt/scripts/db.txt
CONFIG_PROXY=/root/rkt/nginx/config/conf.d/default.conf
DOSSIER_UTILISATEUR=/root/rkt/users

read -p "Quel est votre nom d'utilisateur :" USER

read -p "Selectionnez votre nom de domaine (.p1nk.unic0rn.pw) :" NOM

echo "Attention! Toutes les données de votre site vont être supprimées."

# Choisir son type de serveur
echo "Voulez-vous continuer :"
select YN in "Oui" "Non"; do
    case $YN in
        Oui ) break;;
        Non ) exit;;
    esac
done

# Récupérer l'ID et l'IPdu conteneur
ADDR_IP=`cat $DB | grep $NOM | cut -f3`
ID=`cat $DB | grep $NOM | cut -f4`

rkt stop $ID
sleep 2
rkt rm $ID
sleep 1
rm -r $DOSSIER_UTILISATEUR/$USER/$NOM

# supprimer config proxy
i=0
while read -r line
do
        i=$((i+1))
        if [ $line = "#$ID" ]; then
                break;
        fi
done < "$CONFIG_PROXY"
j=$((i+9))
echo "Suppression des lignes $i à $j..."
sed "$i,$j"d $CONFIG_PROXY >> $CONFIG_PROXY.tmp
mv $CONFIG_PROXY.tmp $CONFIG_PROXY
```

```
        # supprimer de la DB
        LINE=`cat $DB | grep $NOM | cut -f1`
        sed "$LINE"d $DB >> $DB.tmp
        mv $DB.tmp $DB

        # Actualisation du reverse proxy = redemerrage du conteneur
        #arrêt
        PROXY=`rkt list | grep nginx | sort -k6 | cut -f 1 | sed -n 1p`
        rkt stop $PROXY
        sleep 1
        ##suppression
        rkt rm $PROXY
        ##demarrage
        systemd-run --slice=machine rkt run --net=host --volume
config,kind=host,source=/root/rkt/nginx/config docker://nginx --mount
volume=config,target=/etc/nginx/
```

Exemple possible de la base de données :

| | | | | | |
|---|---|---|---|---|---|
| 1 | nom | ip | id | type | volume |
| 2 | monsite | 172.16.28.2 | 36d4e73c | cloudposse/apache-perl | /var/www/html |
| 3 | foo | 172.16.28.3 | a7436eb0 | eboraas/apache-php | /var/www/html |
| 4 | bar | 172.16.28.4 | 9a854952 | httpd | /usr/local/apache2/htdocs |
| 5 | test | 172.16.28.5 | 5a379afd | nginx | /usr/share/nginx/html |