RSX217 Project

Benchmark Docker contre Unikernel au regards du fonction de Firewall

Ilan Keller - ilan.keller.auditeur@lecnam.net

09/02/2023

Contents

1 Introduction

Ce projet a pour but de comparer 2 outils au regards d'une fonction de type firewall. Ces 2 outils, Docker et Unikernel permettent l'emulation d'un sous-sytème à partir d'un système existant.

D'un coté, Docker est un framework facilitant la conteneurisation. Il est populaire et largement utilisé, tant du point de vue de la production que du developpement. Docker est un système tout-en-un, clef-en-main, relativement facile d'utilisation.

De l'autre, Unikernel est une méthode de virtualisation de fonctions systemes. Issue de projet de recherche universitaire et de publication, cette méthode est peu connue. De ce fait, peu d'implémentation existe et sont disponible.Un seul projet FOSS est encore maintenu: OSv. C'est pourquoi, c'est cette implémentation, qui a été choisie.

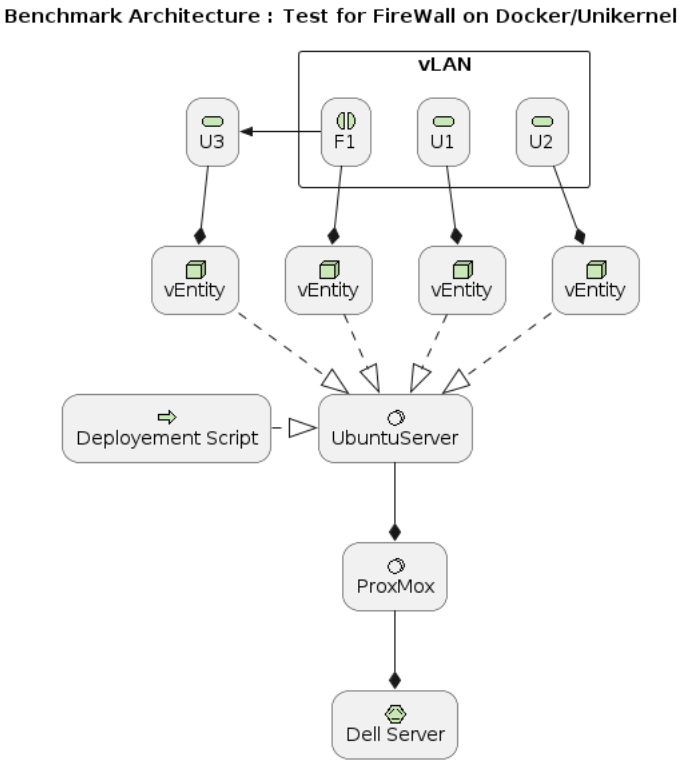
Docker et OSv peuvent être utilisé pour effectuer les mêmes taches mais diffèrent dans leur approche et dans leur nature. Afin d'évaluer deux objets, il est important de mettre en place un environnement et une cinématique de comparaison relativement identique pour la mise en place d'un service similaire.

2 Architecture Générale

Docker a fondamentalement besoin d'un OS sur lequel s'appuyer. Il peut être executé en tant que container LXC mais cette méthode est fortement déconseillée car elle peu sécurisée.

Par ailleurs, les unikernels OSv sont intégralement auto-suffisant et peuvent être construits sous la forme d'image virtuelle QEMU pouvant alors être executée avec seulement un Hyperviseur de type 1.

2.1 Architecture Système



L'achitecture choisit repose sur une machine virtuelle hôte de type Ubuntu Server. Sur cette machine sont déployées automatiquement 4 instances de containeurs docker dans l'un et 4 instances d'image QEMU-OSV dans l'autre.

2.1.1 Handicap au profit de Docker

Ce premier choix est criticable car il désavantage grandement l'implémentation d'Unikernel. Effectivement, comme dit précedemment, celle-ci est auto-suffisante et peut être excutée sur une hyperviseur baremetal. Docker quant à lui n'offre pas le même niveau de service s'il est executé sans OS hôte (problème de sécurité).

Une couche d'abstraction non nécessaire est donc imposée afin d'executer les 2 cinématiques dans un environnement similaire. Une couche supplémentaire d'abstraction induit forcément un temps de déployement et un temps d'execution plus long.

2.1.2 Une autre couche d'abstraction

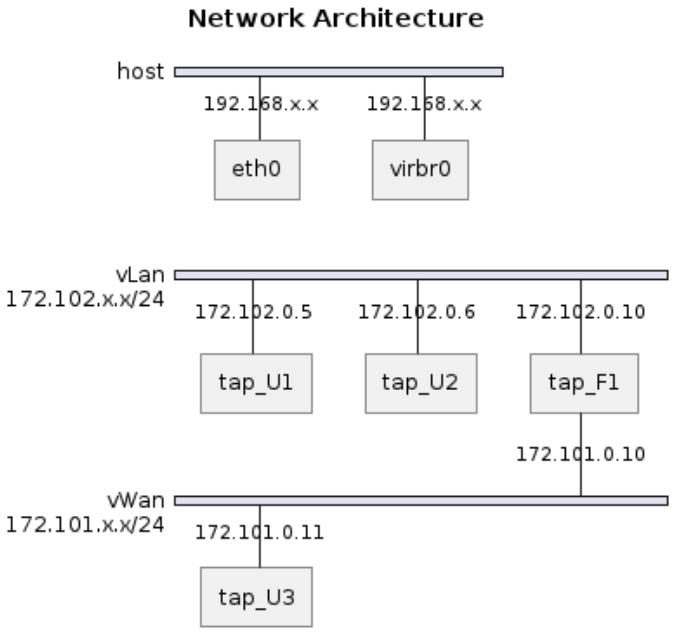
De plus, Docker nécessite une image 0S invité, afin de rester dans une configuration minimale qui puisse s'approcher d'une implémentation OSv, c'est une image de distribution Alpine (5MB) qui a été choisit. Par ailleurs, afin de pouvoir mesurer les temps de déploiements, ceux-ci ont été automatisé.

2.1.3 Déploiement automatisé

Docker propose une fonctionnalité appelé "docker-compose" permettant le deploiement de plusieurs instance configurable. OSv n'ayant pas ces fonctionnalités, le déploiement de ses instances sont effectuées à l'aide de script bash réalisé par mes soins. Afin d'executer les cinématiques de test d'une fonction réseau de type firewall, une architecture réseau a aussi été choisie.

2.2 Architecture Réseaux

L'architecture réseau choisie est identique dans les deux environnements pour les executions de la cinématique de tests. Cependant, leurs implémentations diffèrent.



2.2.1 La configuration Docker

Docker est une solution clef en main. La description sous forme d'argument d'execution, de "DockerFile" ou de fichier descriptif YAML (docker-compose) permettent une configuration suffisament fine pour ce benchmark. OSv n'a pas ce type d'implémentation, toute configuration doit se faire avec des outils systèmes relativement bas niveau.

Le framework Docker ayant un grand succès, ces fonctionnalités d'automatisation et de surcouche est particulèrement optimisé lorsqu'il s'agit de faire des choses simples. Pour des configurations relativement complexe, les experts ont tendances à critiquer cet outil.

2.2.2 La configuration OSv

La configuration réseau de l'architecture OSv a été programmé par mes soins en utilisant les utilitaires brctl, iproute2 et dnsmasq. N'ayant pas les compétences nécessaires pour rivaliser avec les implémentations du projet Docker, ma configuration est, de facto, moins optimisée.

Cependant, les outils systèmes, s'ils sont maitrisés, permettent une configuration beaucoup plus fine des intances réseaux ou de virtualisation (QEMU).

Finalement, afin de tester ces outils et l'architecture générale, il a été choisie une cinématique de test.

3 Cinématique de Test

La cinématique de test prend en compte une implémentation Python des 4 instances virtualisées décritent plus haut. Le script U3 requête un serveur U1 au travers d'un firewall, F1. En fonction du port requeté, une verification de la connexion est faite par U2.

3.1 Implémentation de fonction de FireWall

L'implémentation du firewall a été programmé par mes soins. Afin de faciliter le benchmark, mais en restant dans le thème de la VNF, le firewall manipule des packets sur la couche applicative.

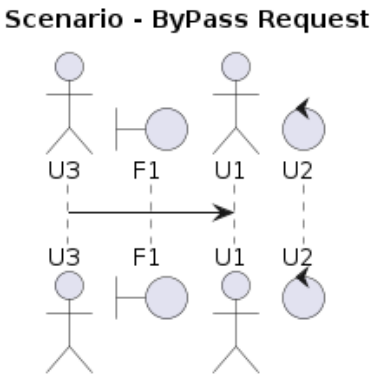
F1 ouvre donc 2 ports sur un "vWan" représentant l'accès internet afin de rediriger les paquets vers U1 ou U2 au sein d'un "vLan". U3 essaye donc de requeter le serveur U1, interne au "vLan". Cette fonction s'apparente à un proxy, cependant reste dans la définition intrinsec d'un firewall applicatif. U3 tente donc 3 requêtes:

1. Une première qui tente de contourner le firewall. Celle-ci echoue forcément car U1 est dans un sous-réseau différent de U1. Cette requête permet de tester la configuration réseau.

2. Une deuxième dite simple: U3 requête le port du firewall qui redirige la requête vers U1.

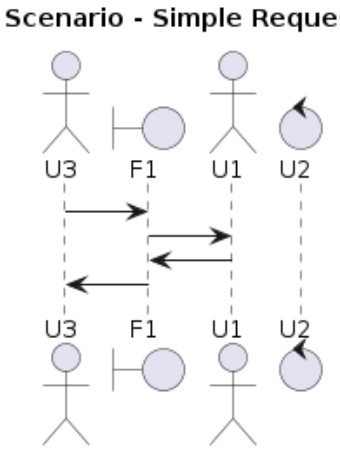
3. Une dernière dite de redirection, où F1, après avoir reçus la requête de U3, la transmet à U2 pour anaylse, avant de la remettre à U1.

3.2 Requête ByPass



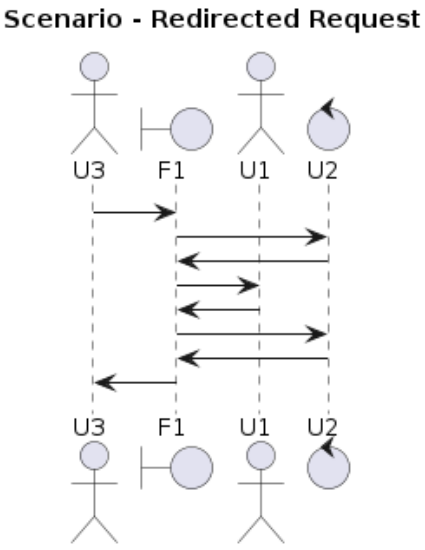
Dans cette requête, U3 tente de communiquer directement avec U1. Les deux instances n'étant pas sur le même sous-réseau, cette requête échoue.

3.3 Requête Simple



Dans cette requête, U3 communique avec U1 après être passé par le firewall F1. U1 lui repond à traver de F1.

3.4 Requête de redirection



Dans cette requête U3 tente de communiquer avec U1 sur un port différent. La requête passe par F1 qui l'a redirige vers U2. U2 réponds à F1 qui la redistribue finalement à U1. La réponse fait finalement le chemin inverse jusqu'à U3.

4 Resultats

	DOCKER	OSV
Time		
real	0m3.309s	0m9.180s
user	0m0.319s	0m0.256s
sys	0m0.046s	0m0.289s
Request		
ByPass	Operation timed out	Connection Refused
Simple	0.0218 ms	0.0662 ms
Redirection	0.0405 ms	0.0822 ms
Image Size	~70 MB	~24MB

4.1 Méthodologie de mesure

4.1.1 Difficulté de mesure cohérente

Reussir à estimer avec précision ou néamoins avec méthodologie, le temps de deploiement d'instance docker n'est pas chose aisée. Le framework n'admet pas d'options pour cela et sortir du cadre des options prédifinit du framework relève du contournement.

4.1.2 Choix pour le déploiement

Il a donc été choisie d'utiliser la commande bash 'time' pour mesurer le temps d'execution globale des 2 processus de déploiement automatique, qui inclus l'execution des 3 cinématiques. Le processus docker avec docker-compose et le processus OSv avec le script écrit pour ce projet.

Les temps, en milliseconde, présentés ensuite, correspondent aux temps écoulés entre l'envoi de chaque requete et la réception de la réponse au regards du script python U3.

4.1.3 Choix pour la commutation

Ces délais (elasped-time) sont calculé à partir d'un objet time.time() python instancié au lancement de l'execution du programme puis soustrait entre eux (voir annexe).

4.2 Explication des résultats

Le resultat, sur l'architecture système et réseau choisi, est sans équivoque. Docker est plus rapide. Cependant, ce resultat était prévisible.

4.2.1 OSv désavantagé

D'une part, l'architecture choisi, indispensable à l'utilisation de docker, n'a pas permis de tirer les avantages de l'implémentation OSv.

D'autre part, la fonction de firewall implémentée agit exclusivement dans la couche applicative et non pas dans la couche transport avec des fonctions systemes. Effectivement, OSv devient interessant quant il s'agit de manipuler des fonctions bas niveau comme netfilter, fonctions que docker manipule mais sans possibilité de configuraton pour l'utilisateur.

4.2.2 Utilisation de ressources systèmes

On constate, sur le resultat des fonctions "time" que OSv est bien plus lent dans l'absolue (real). Cela pourrait s'expliqué par un manque d'optimisation des scripts de déploiement. Mais il est clair, comme expliqué plus haut que c'est dû au fonctionnement de OSv. OSv, est construit sur la virtualisation et l'utilisation de fonction noyau définit dans l'espace système.Ainsi, afin de traiter des fonctions applicatives, il utilise des ressources système que Docker n'utilise pas (kernel - sys). Cela ce retrouve dans les temps de commutation des requêtes.

4.3 Conclusion

4.3.1 Les limites de Docker

Docker est un très bon outil, surtout parcequ'il est accessible. Malheureusement, cette accessibilité ce fait au détriment de certaines possibilités de configuration et de bonne pratique. Le premier problème de docker est la sécurité: l'accès root, la manière dont il modifie les tables IP et le manque de marge de manoeuvre disponible est décrié par la communauté.

4.3.2 Exemple de limite

Par exemple, malgrès le faite que l'on a bien spécifié à Docker que les machines se trouvent sur des plages réseaux différentes, le code d'erreur retourné par le script U3 sous Docker n'est pas le bon (voir tableau résultat - ByPass). La requête attend un timeout. L'implémentation python d'U3 étant principalement synchrone (réquete envoyée les unes après les autres), le processus U3 sous docker, attends le resultat de la requete. Alors que dans le même script U3, sous OSv, la requête échoue instantanement. J'ai encore du arrangé le fonctionnement du benchmark, en retirant cette réquete

afin d'obtenir des temps d'execution comparable (Docker +130 second avec l'attente du timeout).

4.3.3 Ouverture

Docker a donc joué à domicile avec un environnement avec de nombreux arrangements (OS Hôte, Fonction Firewall Applicative, suppression de la requête ByPass) Ce travail, pourrait avoir une suite. Un match retour, où OSv sera dans sa zone de confort. C'est à dire une fonction de firewall implémentée avec des bibliothèques de la couche système. Il serait alors interessant de mette en exèrgue une autre solution de conteneurisation comme systemd-spawn, outil de containeurisation sans surcouche applicative.

5 Annexe

5.1 GitHub link

`https://github.com/Ix0rK/docker_unikernel_firewall_benchmark`

5.2 Result OSv

```
14 U3 UP!|-->8.177757263183594e-05
15 [U3-RS] First Message to HTTP PORT|-->0.010297060012817383
16 [U3-RS] RequestSimple : Sent !|-->0.03545069694519043
17 [U3-RS] ACK from TCP Server received:|-->0.09377002716064453
18 [U3-RS] Request Simple : END !|-->0.1016855239868164
19 [U3-RWR] Second Message to SSH PORT|-->0.10966634750366211
20 [U3-RWR] RequestWithRedirection : Sent !|-->0.1171104907989502
21 [U3-RWR] ACK from TCP Server(EVE OUT) received|-->0.19220709800720215
22 [U3-RWR] RequestWithRedirection : END !|-->0.19934368133544922
23 [U3-END] END TEST|-->0.2048490047454834
24 [U3-END] END TEST : Sent !|-->0.2104175090789795
25 [U3-END] RAS received|-->0.23522138595581055
26 [U3-END] END TEST : END !|-->0.24071455001831055
27 U3 END !|-->0.246673583984375
```

Figure 1: Result for OSv Test Benchmark

5.3 Result Docker

```
360 U3_1  | U3 UP!|-->2.384185791015625e-06
361 U3_1  | [U3-RS] First Message to HTTP PORT|-->5.745887756347656e-05
362 U3_1  | [U3-RS] RequestSimple : Sent !|-->0.0003497600555419922
363 U3_1  | [U3-RS] ACK from TCP Server received:|-->0.02210235595703125
364 U3_1  | [U3-RS] Request Simple : END !|-->0.0221097469329834
365 U3_1  | [U3-RWR] Second Message to SSH PORT|-->0.02218461036682129
366 U3_1  | [U3-RWR] RequestWithRedirection : Sent !|-->0.022288799285888672
367 U3_1  | [U3-RWR] ACK from TCP Server(EVE OUT) received|-->0.06285738945007324
368 U3_1  | [U3-RWR] RequestWithRedirection : END !|-->0.06286454200744629
369 U3_1  | [U3-END] END TEST|-->0.06291532516479492
370 U3_1  | [U3-END] END TEST : Sent !|-->0.06301116943359375
371 U3_1  | [U3-END] RAS received|-->0.11439037322998047
372 U3_1  | [U3-END] END TEST : END !|-->0.11439776420593262
373 U3_1  | U3 END !|-->0.11445403099060059
```

Figure 2: Result for Docker Test Benchmark