HEPL
Haute Ecole de la Province de Liège

Rapport
Architecture des systèmes d'exploitation

NOUPOUE Cameron

Groupe 402, fait le 17/12/2024

Versions utilisées:

Kernel: 6.6.15

Busybox: 1.34.0

NGINX: 1.19.5

Identifiants :

Kernel: root, NOUPOUE2024

OAUTH2: cameron, password

Création d'un OS linux entièrement à partir des sources sur /dev/sda5 (*étapes 1 et 2*)

Ces étapes m'ont permis de mettre en pratique un grand nombre de connaissances vues au cours théorique en créant une VM vide et la configurant comme nécessaire pour pouvoir s'y connecter en SSH. J'ai ensuite pu partitionner cette machine et y créer des systèmes de fichiers. Ces étapes m'ont également permis de me familiariser avec le fait de monter et démonter des partitions. J'ai pu utiliser un gestionnaire de paquets, télécharger des utilitaires à partir des sources, changer le root de l'OS et me familiariser avec la notion de GRUB, créer un FHS, ...

Bref, un tas de compétences pratiques qui permettent de mieux comprendre le fonctionnement plus bas niveau des noyaux Linux. Travailler avec Busybox et Glibc, par exemple, me fait comprendre à quel point les OS avec lesquels on a l'habitude d'utiliser ont des couches d'abstractions énormes dont on ne se rend pas compte.

J'ai rencontré quelques difficultés sur lesquels j'ai passé pas mal de temps mais qui, au final, m'ont permis d'encore mieux comprendre certains points. Par exemple : j'ai eu pas mal de problèmes concernant la configuration de mon kernel. En effet, j'ai dû revenir dans le make menuconfig pour changer plusieurs points et lorsque je sauvegardais, je n'avais pas compris qu'il fallait recopier le fichier « bzImage » dans le dossier « boot » et donc je ne voyais pas mes changements lorsque je relançais mon kernel. J'ai également eu des problèmes pour me connecter en SSH, j'ai donc du activer le SSH pour le root (via /etc/ssh/sshd_config), malgré que le user auquel j'essayais de me connecter en SSH était bien dans le groupe « wheel ». Ma partition /dev/sda5 n'était pas bien montée en ext4, j'ai donc eu des problèmes lorsque je voulais monter la partition.

Création d'une clé USB bootable contenant votre propre système GNU/Linux (*étape 3*)

J'ai pu formatter une clé USB dans laquelle j'ai mis un bootloader plus léger que GRUB afin d'y mettre mon kernel compilé. L'objectif étant de rendre la clé fonctionnelle.

Cette étape était très concrète et m'a permis de voir de manière pratique comment booter un OS sur une machine.

Évidemment, cette étape a apporté son lot de difficultés : ma clé USB étant en 3.1, VMware avait des difficultés à la reconnaitre et se coupait instantanément lorsque je la branchais dans la VM, donc j'ai dû configurer les devices USB sur du 2.0. Ensuite, une fois que ma clé était bootable, je n'avais pas de connexion dessus lorsque je la démarrais sur différents appareils, et c'était car je n'avais pas compris qu'il fallait activer l'interface réseau sur mon OS et puis attribuer l'adresse ip donnée par l'utilitaire « udhcpc » ne se mettait pas automatiquement mais qu'il fallait l'attribuer manuellement sur l'interface.

Installation de Docker sur votre VM Gentoo (*étape 4*)

Cette étape consistait à installer Docker sur la VM Gentoo afin de l'utiliser sur une nouvelle instance de notre kernel personnel.

Pour activer la conteneurisation, il fallait d'abord l'autoriser via les namespaces et les Cgroups. Il fallait ensuite recompiler le noyau et mettre à jour le GRUB.

Ensuite, j'ai installé Docker en utilisant un gestionnaire de paquets. Il ne restait plus qu'à relancer la VM en lançant notre nouveau noyau et Docker était utilisable.

J'ai cependant rencontré un problème lorsque j'ai voulu télécharger Docker via le gestionnaire de paquets car l'horloge de ma VM était restée bloquée dans le passé, le téléchargement n'était donc pas possible puisque les certificats étaient biaisés. J'ai changé manuellement l'heure de ma VM pour y remédier.

Working with containers on your Gentoo WM with your personal kernel (*étape 5*)

Cette étape m'a permis d'héberger une application conteneurisée regroupant plusieurs services sur ma machine et de la lancer au démarrage via des fichiers « docker compose ».

Grâce aux différents fichiers « Dockerfile », j'ai compris comment personnaliser mon application selon mes besoins.

Puisque mon application était multi-containers, j'ai également appris à créer un réseau docker interne entre mes différents services afin qu'ils puissent communiquer entre eux et des volumes afin de faire persister les données de mon application.

Pour automatiser le déploiement de cette application, j'ai mis en place un fichier « docker compose » qui est un langage déclaratif afin d'y décrire le résultat final souhaité. J'ai également testé la fonctionnalité « scale » qui déploie plusieurs conteneurs pour un même service afin d'alléger les tâches et les ressources.

Cette étape était également compliquée, surtout au niveau de la communication entre les différents services via le réseau docker que j'ai créé. J'ai également pris du temps à comprendre comment faire matcher les références aux services Docker dans les fichiers du frontend (à savoir « ***index.html*** » et « ***nginx.conf*** ».

De plus, en utilisant le nom du service dans la référence au websocket (« ***index.html*** »), cela ne fonctionnait pas, j'ai donc temporairement mis l'adresse ip statique du déploiement, avant de trouver une solution plus viable et automatique.

Lorsque j'ai testé la fonction « scale », j'ai modifié mon « docker compose » car le nom de mon conteneur pour le backend ainsi que le numéro de port étaient fixes, et donc le Docker engine ne pouvait pas créer deux instances avec ces deux mêmes paramètres. C'était également une des difficultés auxquelles j'ai fait face

Création et utilisation d'un cluster Kubernetes

Grâce à cette étape, j'ai pu apprendre Kubernetes dans un environnement réel. Au travers de l'architecture mise en place dans le Data Center qui m'a été

attribué (place 32), j'ai pu configurer un master node et lui associer deux worker nodes. J'ai pu ensuite configurer mon application multi-conteneurisée afin d'avoir encore plus de contrôle via la gestion des pods, l'auto-healing, le scale, l'option de créer des replicas, le load balancing, … fournis par Kubernetes.

J'ai ensuite déployé mon application à l'extérieur en exposant le point d'entrée (à savoir « oauth2 »). Tous les autres services ont uniquement un port interne afin de pouvoir communiquer entre eux tout en restant inaccessible depuis l'extérieur (à savoir « ClusterIP »).

Concernant les images et le stockage de notre application, j'ai créé un serveur NFS externe sur lequel j'ai copié les images et les ai rendus accessible via une « private registry » et où j'ai monté une partition concernant les fichiers relatifs à la base de données.

Pour rentre accessible les fichiers de base de données sur le NFS depuis le master node et ses workers, il faut configurer l'accès à ceux-ci via un déploiement sur le master node avec un PV (persistant volume) qui établit l'accès au NFS indépendamment des pods afin de survivre au-delà de leur durée de vie ainsi que centraliser son management et un PVC (persistent volume claim) qui fait une demande d'accès au PV.

Cette étape fût très concrète pour un environnement de production. C'était donc très intéressant à réaliser pour ma connaissance personnelle.

C'est certainement l'étape qui m'a causé le plus de difficicultés : dans un premier temps, j'ai ajouté à la VM une interface réseau en « bridge » pour la connecter à la carte réseau de mon ordinateur physique, mais le mode « bridge » de base de VMware ne fonctionnant pas, je suis passé par le Virtual Network Editor.

Lors de la création de mes fichiers de déploiements ainsi que le déploiement, j'ai fait beaucoup d'aller-retour entre le Data Center, le NFS server et ma VM Docker car j'ai été contraint de modifier plusieurs fichiers et images et donc je redevais construire des images (avec des nouveaux tags, …). Par exemple, le fichier, certains de mes services étaient écrits en majuscule, ou avec des tirets et cela n'était pas compatible avec la syntaxe définie par Kubernetes, j'ai dû changer la version de MongoDB vers la v4.4, j'ai dû créer une image spécifique pour Oauth contenant directement les fichiers « .htpasswd » car j'utilisais auparavant l'image provenant de Dockerhub mais et j'ajoutais le fichier de crédits en local, ce qui n'était plus possible sur Kubernetes. Tous ces allers-retours m'ont fait perdre beaucoup de temps mais étaient essentiels et m'ont permis d'être à l'aise avec les concepts clés et les commandes.

J'ai également eu du mal à gérer l'espace disponible (fort restreint) sur le serveur NFS, par exemple : supprimer certaines collections ou index sur les fichiers de ma base de données sur le serveur NFS

Configuration matérielle

Concernant ma machine physique, il s'agissait d'un Asus tournant sur Ubuntu 24.04 avec un Intel core i5 et 16gb de RAM.

Les temps de compilation :

- *Noyau Gentoo :* ***00:28:45***
- *Glibc :* ***00:20:38***
- *Kernel personnel :* ***00:25:21***
- *Installation Docker :* ***00:39:12***

L'espace occupé :

- Le disque de la VM (Gentoo + perso) occupe environ 11 GB d'espace.

```
NOUPOUE2024KERNEL ~/MultiContainerWebapp # uname -a
Linux NOUPOUE2024KERNEL 6.6.15 #2 SMP PREEMPT_DYNAMIC Thu Oct 24 18:13:45 CEST 2
024 x86_64 Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz GenuineIntel GNU/Linux
NOUPOUE2024KERNEL ~/MultiContainerWebapp # df -h
Sys. de fichiers Taille Utilisé Dispo Uti% Monté sur
/dev/root           19G     11G  7,2G  60% /
```

- Docker et les conteneurs prennent environ 2,5 Go.

```
NOUPOUE2024KERNEL ~ # docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          7         4         2.498GB   2.498GB (100%)
Containers      4         0         1.114kB   1.114kB (100%)
Local Volumes   12        2         215.1MB   0B (0%)
Build Cache     0         0         0B        0B
```

- La clé USB bootable, qui contient l'OS personnel, utilise environ 200 MB.

```
NOUPOUE2024KERNEL /mnt/usb-mylinux # df -h /dev/sdd*
Sys. de fichiers Taille Utilisé Dispo Uti% Monté sur
devtmpfs           2,0G       0  2,0G   0% /dev
/dev/sdd1          197M    142M   56M  72% /mnt/usb-boot
/dev/sdd2          182M     55M  114M  33% /mnt/usb-mylinux
devtmpfs           2,0G       0  2,0G   0% /dev
```

Lors de ce cours, j'ai pu apprendre beaucoup de compétences à propos de Linux. Avant ce cours, j'avais les connaissances nécessaires en tant qu'utilisateur « *intermédiaire* » dans Linux et je me débrouillais plutôt bien pour utiliser cet OS. Après ce cours, je me sens plus à l'aise et fluide d'un point de vue d'un créateur, je suis capable de monter des partitions, comprendre le boot, qui était quelque chose de très flou pour moi, gérer les systèmes de fichiers, … Je comprends également bien mieux ce qui se cache derrière chaque exécutable (ls, …) grâce aux outils que l'on a utilisé tel que busybox, glibc. Et encore pleins d'autres choses.

A propos de Docker, avant le cours, je comprenais le fonctionnement de base, c'est-à-dire, le principe et j'étais capable d'utiliser des images et les faire tourner chez moi, en local. Après ce cours, je suis également capable de créer mes propres images, de les personnaliser, de les déployer de manière automatique

avec un « docker compose », je comprends les principes de « layers » pour optimiser mes images, etc. J'ai également pu comprendre le principe de volume qui est un principe fondamental en tant qu'informaticien.

Pour ce qui est de Kubernetes, c'était une totale découverte pour ma part étant donné que je n'en avais jamais entendu parler auparavant. Après ce cours, je comprends maintenant le but et l'objectif derrière Kubernetes. J'ai pu manipuler des pods, des clusters, et automatiser une application conteneurisée dans un environnement se rapprochant d'une production, tout en pouvant optimiser et orchestrer mes conteneurs.