Cours : infrastructures réseaux de base avec Kathara

Présentation de Kathará

Kathará est un Framework de l'Université de Rome qui succède à **Netkit** en utilisant **Docker** et **Python**. Cette solution permet de créer de concevoir et de tester des architectures de réseaux locaux en implémentant ma-chines virtuelles légères sous la forme de conteneur Docker reliés ensemble par un réseau virtuel indépendant du réseau de la machine d'accueil. Il est ainsi possible de tester une configuration réseau complexe sans la nécessité de droits spéciaux sur **l'ordinateur hôte**.

La réalisation d'infrastructure réseau est de nos jour de plus en plus complexe car cela met en œuvre :

- Des ordinateurs différents de type serveur et client,
- Des services et protocoles réseaux divers (DHCP, DNS, Web, Supervision, RIP, etc.)
- Des équipements actifs comme les switchs, les routeurs, les pare feux,
- La gestion de plusieurs interfaces réseaux au niveau d'un même équipement,

Cela se traduit par des topologies réseaux variées en plus d'être complexes.

Dans un cadre d'enseignement, où l'on ne dispose pas des matériels nécessaires pour implémenter les topolo-gies réseaux afin de les étudier et les tester, il est possible d'utiliser des logiciels de simulation comme Cisco Paquet Tracer. Mais ce logiciel qui permet l'étude d'infrastructure réseau est moins adapté à la mise en place de services réseaux. Il permet de reproduire un certain nombre de fonctionnalités sans en reproduire le comporte-ment réel en termes de performance.

Kathará permet de mettre en œuvre des protocoles récents permettant la virtualisation des fonctions de réseau (NFV) et la mise en réseau définie par logiciel (SDN). Cela va modifier la manière de mettre en réseau des ser-vices, en permettant la programmation des infrastructures réseaux dans le but de séparer la logique de l'infrastructure à réaliser du matériel qui va la mettre en œuvre. Ensemble, ils présentent plusieurs avantages, principalement en termes d'évolutivité et de flexibilité, pour déployer des fonctions de réseau virtuel (VNF)

Kathará permet d'émuler des réseaux d'ordinateurs de type Linux en utilisant Docker. Chaque équipement ré-seau (serveur, client, routeur, switch) est un conteneur Docker. La souplesse de Docker permet d'avoir des conteneurs basés sur des images personnalisée permettant la mise en œuvre de :

- **Quagga**, une suite de logiciels de routage implémentant les protocoles OSPF, RIP, BGP et IS-IS pour les routeurs,
- **Open vSwitch**, une solution de gestion de commutateur virtuel,
- **P4** un langage de programmation des équipements réseaux.

Dans le cours ne sera pas abordé que ces fonctionnalités de l'utilisation de Kathara permettant de maquetter des infrastructures réseaux offrant des services de base.

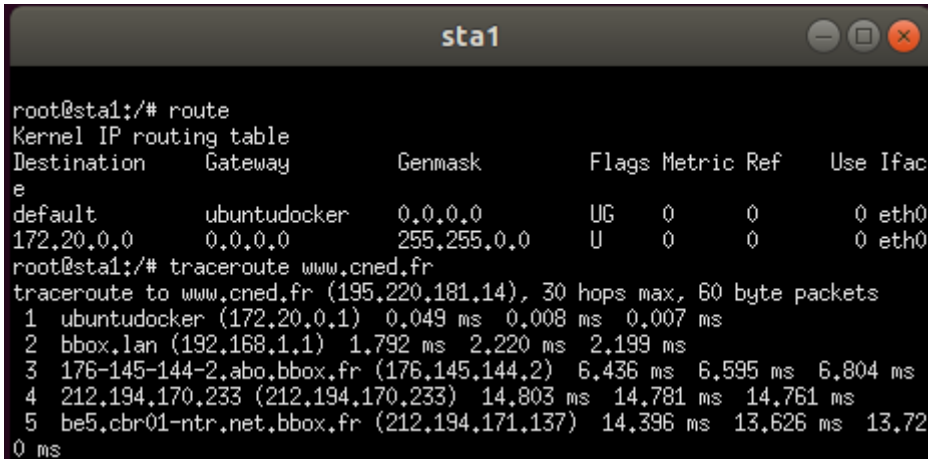
Lien vers le site officiel de Kathara de l'Université de Rome : http://www.kathara.org/

L’émulation de périphériques réseau

Avec Kathara, chaque équipement réseau qui sera créé dans un conteneur Docker possède :

- Une console en mode texte,
- De la mémoire,
- Un système de fichiers,
- Une ou plusieurs interfaces réseaux selon vos besoins.

Voici un exemple d’équipement réseau qui correspond à un client :



En utilisant Docker, Kathara permet :

- De créer et de gérer plusieurs **machines virtuelles (VMs)** sous forme de **conteneurs Docker**,
- De relier ces VMs à des **domaines de collision** qui sont des hubs virtuels pour permettre aux VMs de communique entre elles,
- De définir le rôle de chaque VM soit comme simple ordinateur **client linux**, comme **serveur** en installant si nécessaire des paquets logiciels supplémentaires, comme **routeur** ou comme **switch**.

Le jeu de commandes de Kathara

Netkit fournit deux groupes de commandes :

- les **vcommandes**, préfixées par **’v’**, qui permettent de manipuler une seule VM à la fois ;
- les **lcommandes**, préfixées par **’l’** qui servent à manipuler des ensembles complexes de machines virtuelles en réseau. Dans le langage de Kathara, il s’agit des **Labs** (laboratoires).

Si vous souhaitez travailler avec une seule VM, utilisez les vcommandes. Sinon, pour travailler avec plusieurs VMs, il est préférable et bien plus pratique de créer un laboratoire (Lab) et d’utiliser alors les lcommandes.

Utilisation de machines autonomes

les v-commandes

Commande	Action
vstart	
vlist	→ donner la liste des VMs actives
vconfig	→ configurer à la volée une VM comme par exemple affecter une interface à la volée.
vclean	→ arrêter une VM et nettoyer les processus, configurations et fichiers temporaires créés

Gérer une VM

Pour s’assurer du bon fonctionnement de Kathara, vous pouvez créer depuis un terminal une première machine virtuelle avec le nom **sta1**. **Création d’une VM avec vstart**

```
btssio@ubuntudocker:~$ vstart --eth=0:HubDCA sta1
```

Explications :

- La commande vstart permet de lancer en interactif une VM ;
- **–eth** permet de définir le numéro de l’interface réseau **eth0** associée à au domaine de collision **HubDCA** (hub virtuel) ; Un domaine de collision correspond à un **concentrateur** pour Kathara.
- Le nom de la VM **sta1** est le dernier paramètre

Voici votre première VM Netkit avec la session root automatiquement ouverte :

```
btssio@ubuntudocker: ~
File  Edit  View  Search  Terminal  Help
btssio@ubuntudocker:~$ kathara vstart -n sta1 --eth 0:HubDCA --bridged
===================== Starting Machine =====================
Deploying links... |################                | 1/2
Deploying machines... |################################| 1/1
btssio@ubuntudocker:~$
```

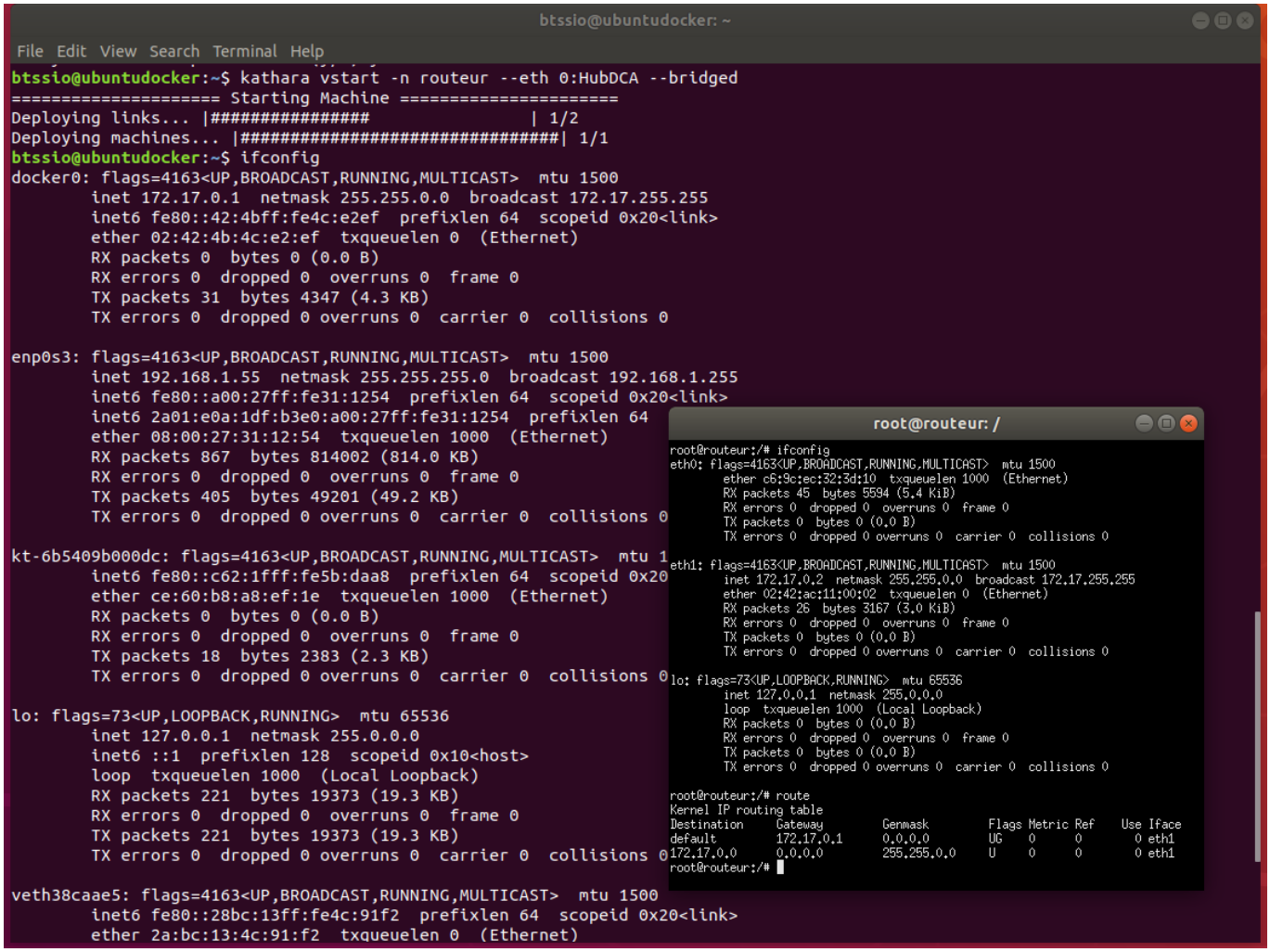
```
root@sta1: /
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 22:33:2b:e5:81:ed  txqueuelen 1000  (Ethernet)
        RX packets 36  bytes 4836 (4.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 21  bytes 2706 (2.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@sta1:/#
```

Pour cette VM sta1, aucune adresse IP n’a été définie. C’est le bridge Docker créé par Kathara qui a fourni la configuration IP 172.20.0.2/16 dans le réseau 172.20.0.0/16. La passerelle est 172.20.0.1/16 et la VM accède à Internet



Pour visualiser le bridge créé par Kathara, tapez la commande suivante dans le terminal de votre serveur Debian/Ubuntu :

```
btssio@ubuntudocker:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UP group default qlen 1000
    link/ether 08:00:27:af:88:bf brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.199/24 brd 192.168.1.255 scope global dynamic enp0s3
    valid_lft 82586sec preferred_lft 82586sec
    inet6 fe80::a00:27ff:feaf:88bf/64 scope link
    valid_lft forever preferred_lft forever
3: br-8edf20a49895: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc no-
queue state DOWN group default
    link/ether 02:42:64:3d:be:51 brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.1/16 brd 172.20.255.255 scope global br-8edf20a49895
    valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
```

```
DOWN group default
    link/ether 02:42:1d:98:0e:31 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
    valid_lft forever preferred_lft forever
btssio@ubuntudocker:~$
```

INFORMATION
Le sous-réseau 172.0.0.0/8 est réservé pour l'utilisation de Kathara. Il ne faut donc pas l'utiliser pour défi-nir des plans d'adressage de vos sous-réseaux. Lors de la création de VMs autonomes ou dans des labs comme vous le verrez ensuite, Kathara va créer autant de bridges que vous définissez de domaine de colli-sion en leur associant un sous-réseau différent basé sur ce sous-réseau 172.0.0.0/8. Le premier de ces sous-réseaux est 172.19.0.0/16, le dernier est 172.255.0.0/16.

Visualisation de la VM créée avec vlist

```
btssio@ubuntudocker:~$ vlist
CONTAINER ID NAME             CPU % MEM USAGE / LIMIT MEM %  NET I/O BLOCK
I/O PIDS
8ceba73bfb3f netkit_1000_sta1   0.00%  2.16MiB/1.419GiB  0.15%  6.58kB/0B
12.2MB/0B 2
NETWORK ID          NAME                DRIVER              SCOPE
f332c80a9dbd        bridge              bridge              local
c28efeb7848e        host                host                local
2877cae4fa72        netkit_1000_H       bridge              local
708458e85954        none                null                local
btssio@ubuntudocker:~$
```

Vous pouvez visualiser :

- Les caractéristiques de la VM **sta1** : son **ID Docker 8ceba73bfb3f** ainsi que les ressources consommées ;
- La liste des interface réseaux du serveur Debian/Ubuntu qui montre le bridge **netkit_1000_H** créé par Kathara et qui est associé au domaine de collision **HubDCA**.

Le commande Docker montre le conteneur **8ceba73bfb3f** qui correspond à **sta1** et l'image **kathara/netkit_base** qui a été utilisée.

```
btssio@ubuntudocker:~$ docker ps -a
CONTAINER ID  IMAGE                COMMAND     CREATED  STATUS PORTS NAMES
8ceba73bfb3f kathara/netkit_base  "/bin/bash" 3 m…  Up 3 m…
netkit_1000_sta1
…
```

Cette autre commande permet également de visualiser le container Docker créé et vous pouvez le visualiser avec la commande suivante :

```
btssio@ubuntudocker:~$ docker container ls
```

```
CONTAINER ID IMAGE                COMMAND     CREATED STATUS  PORTS  NAMES
8ceba73bfb3f kathara/netkit_base "/bin/bash" 4 m…    Up 4 m…
netkit_1000_sta1
btssio@ubuntudocker:~$
```

Pour arrêter la VM avec une commande Kathara ; le container est alors supprimé :

INFORMATION
La commande **vconfig** semble actuellement ne pas fonctionner convenablement

Arrêter une VM avec vclean Le container est alors supprimé :

```
btssio@ubuntudocker:~$ vclean sta1
Any network still in use by another machine will not be deleted (and will
raise an error instead)
Containers will be deleted
netkit_1000_sta1
netkit_1000_H
btssio@ubuntudocker:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
btssio@ubuntudocker:~$
```

Utilisation des labs

Pourquoi utiliser les labs ?

Quand vous avez à créer et gérer une seule VM, vous utilisez les v-commandes. Pour des infrastructures utilisant plusieurs VMs, il est préférable de créer un laboratoire ou **lab** et de manipuler ce lab avec les **l-commandes**. Ces labs permettent de concevoir mais aussi de conserver une architecture réseau complexe ou que l'on souhaite pouvoir réutiliser et cela d'autant plus facilement qu'un lab se traduit par une arborescence de dossiers contenant des fichiers de configuration. Un lab occupe très peu de place et est facile à sauvegarder et à échanger. Vous trouverez sur Internet des ressources et des exemples de labs.

Voici des liens vers les labs proposés par l'équipe de Kathara :

- https://github.com/KatharaFramework/Kathara-Labs/wiki

Sur ce site, vous trouverez de la documentation ainsi que des exemples de simulations qui peuvent être très complexes.

Un Lab **minimaliste** consiste en une arborescence comprenant :

- Obligatoirement un fichier de configuration (**lab.conf**) qui décrit les machines virtuelles qui seront lancées, leurs interfaces et les domaines de collision.
- un **répertoire** par machine virtuelle qui sera lancée, dans lequel on peut stocker des fichiers.

Pour l'instant, laissez ce répertoire vide.
- Eventuellement pour les VM des fichiers **VMx.startup** et/ou un fichier **VMx.shutdown** qui indiquent les actions à réaliser lors du lancement ou de l'arrêt de la VMx (VMx correspond au nom de la VM Kathara). Cela permet de configurées automatiquement la VMx lors du lancement du lab à l'aide de scripts shell. Les scripts doivent être exécutables.

Les l-commandes

Les l-commandes sont utilisables dans le répertoire du lab qui doit contenir au minimum le fichier lab.conf, ex-ception faire de la commande **lwipe**.

Commande	Action
lstart	→ pour lancer un lab.
lclean	→ arrêter les VMs et nettoyer les processus, configurations et fi-chiers temporaires créés
linfo	→ information sur le laboratoire.
ltest	→ vérification du bon fonctionnement du laboratoire.
lwipe	→ arrêter les VMs des labs de Kathara et arrêter une VM et nettoyer les processus, configurations et fichiers temporaires créés.

Dans l'activité sur les infrastructures réseaux avec Kathara sera abordé la création des labs :

- la création de l'arborescence des dossiers du lab,
- le contenu du fichier lab.conf,
- la création des fichiers qui permettent de personnaliser le fonctionnement des VMs,
- l'utilisation des l-commandes,
- l'ipmasquerade pour configurer une VM comme routeur NAT.

Etendre les fonctionnalités de l'image Kathara

Lorsque Kathara crée une VM, avec une v-commande ou une l-commande, c'est un conteneur qui est créé par Docker à partir de l'image **kathara/netkit_base**.

L'équipe de développement de Kathara a intégré à cette image un certain nombre de paquets logiciels de telle sorte que les VMs créées puissent les utiliser. Cependant, si dans la réalisation de votre maquette vous avez besoin d'un paquet logiciel qui n'est pas présent dans les VMs, vous pouvez **l'ajouter** à l'image **kathara/netkit_base** afin que toutes les VMs qui seront ensuite créées puissent l'utiliser.

La démarche qu'il faut suivre est la suivante :

- **Créer** un conteneur avec image kathara/netkit_base,

```
btssio@ubuntudocker:~$ docker run -itd --name kathara_cned
kathara/netkit_base
```

INFORMATION
Le conteneur est créé en lui associant un nom à votre convenance, ce qui sera plus facile pour l'identifier.

- **Se connecter** à une console du conteneur :

```
btssio@ubuntudocker:~$ docker exec -it kathara_cned bash
root@93e38cf2c65:/#
```

- **Mettre à jour** le conteneur :

```
root@93e38cf2c65:/#apt-get update && apt-get upgrade
```

- **Installer** les paquets logiciels voulus ; pour cet exemple la bibliothèque scapy pour python :

```
root@93e38cf2c65:/#apt-get install scapy
```

- **Quitter** le conteneur :

```
root@93e38cf2c65:/#apt-get clean && exit
```

- **Enregistrer** le conteneur modifié dans l'image **actuelle** ou comme une **nouvelle image**. La deuxième solution sera utilisée car elle permet de garder l'image de base et dans ce cas il faut faudra indiquer le nom de cette nouvelle image, soit pour toutes les VMs, soit uniquement pour celles qui en ont besoin.
- **Pour information : enregistrer** le conteneur dans l'image actuelle :

```
btssio@ubuntudocker:~$ docker commit kathara_btssio kathara/netkit_base
```

- **A faire : enregistrer** le conteneur dans une nouvelle image:

```
btssio@ubuntudocker:~$ docker commit kathara_cned kathara/netkit_btssio
```

- **Utiliser** la nouvelle image pour tous les conteneurs Kathara:

Il est nécessaire de modifier la directive IMAGE_NAME de Kathara en modifiant à ligne 15 du fichier **/opt/Kathara/bin/python/netkit_commons.py** :

```
btssio@ubuntudocker:~$ nano /opt/Kathara/bin/python/netkit_commons.py
```

- **Utiliser** la nouvelle image pour une seule VM d'un lab :

Il suffit d'indiquer dans le fichier lab.conf un paramètre supplémentaire précisant l'image à utiliser :

```
Routeur[image]=netkit_btssio
```

Vous disposez maintenant de deux images pour les VMs de Kathara :

- L'image de base **kathara/netkit_base**,
- Et une image personnalisée **kathara/netkit_btssio**.