TP Réseaux : Kathará

TP Réseaux

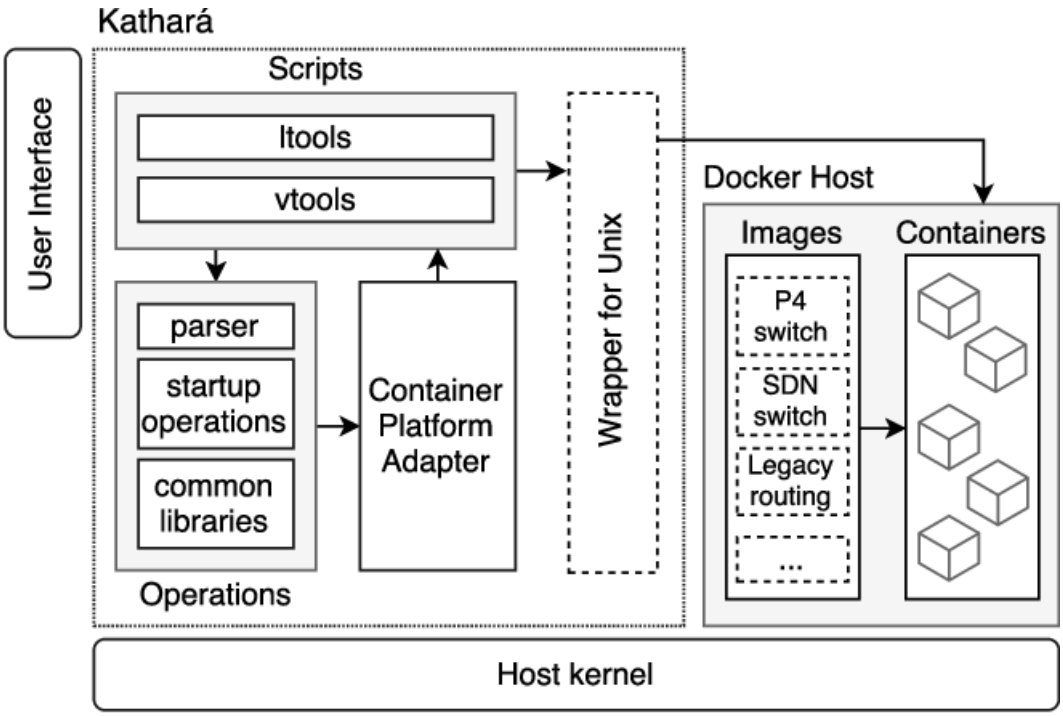
Il s'agit de présenter l'environnement Kathará utilisé dans les TP Réseaux.

Kathará : un système pour émuler des réseaux informatiques

Kathará est une infrastructure basée sur des conteneurs pour déployer des réseaux virtuels. Kathará est une implémentation de **Netkit** en utilisant **Python** et **Docker**.

Lien : www.kathara.org

Netkit est un ensemble de scripts permettant de démarrer et configurer un ensemble de machines et de créer un réseau virtuel entre ces machines. http://wiki.netkit.org/
Docker est un logiciel libre d'automatisation et de virtualisation applicative : il permet de lancer des applications dans des conteneurs logiciels. www.docker.com



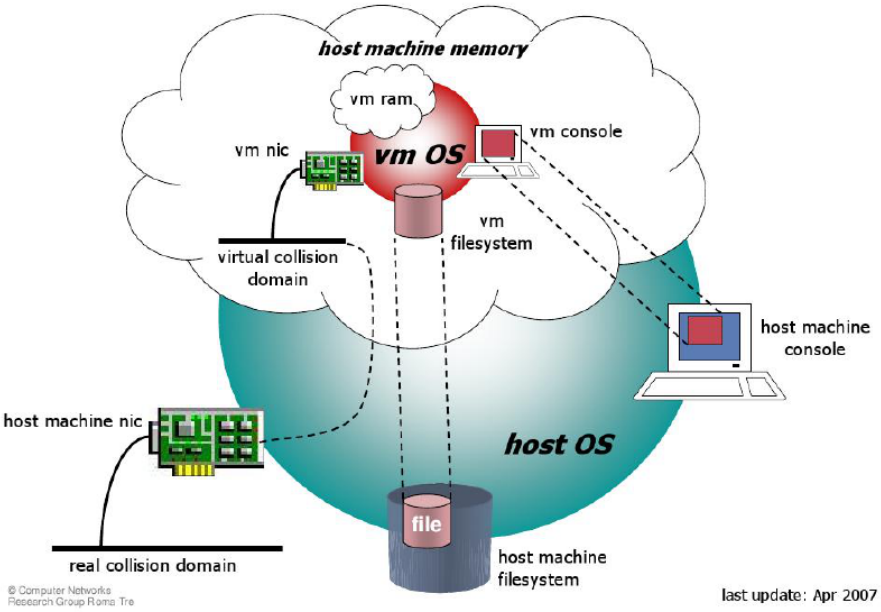
Netkit

C'est un ensemble de scripts permettant de démarrer et configurer un ensemble de machines et de créer un réseau virtuel entre ces machines.

— Émule des réseaux d'ordinateurs
— Outil de maquettage et de simulation
— Pour comprendre le fonctionnement des protocoles
— Sans avoir à investir dans des équipements
— Logiciels open source (licence GPL)

Notions de base :

Les **machines virtuelles** sont reliées à des domaines de collisions virtuels (un *hub*) et elles peuvent donc communiquer entre-elles. Chaque machine virtuelle peut jouer le rôle de PC (client et/ou serveur), routeur ou *switch*.



Les nœuds sont raccordés sur des **domaines de collision**. Un domaine de collision virtuel peut :

- être connecté à plusieurs interfaces

- chaque interface peut être connectée à un ou plusieurs domaines de collision

La maquette du réseau se nomme un ***lab***. C'est un ensemble de fichier permettant de créer puis de configurer un scénario réseau netkit automatiquement. En général, un scénario est contenu dans un répertoire du même nom.

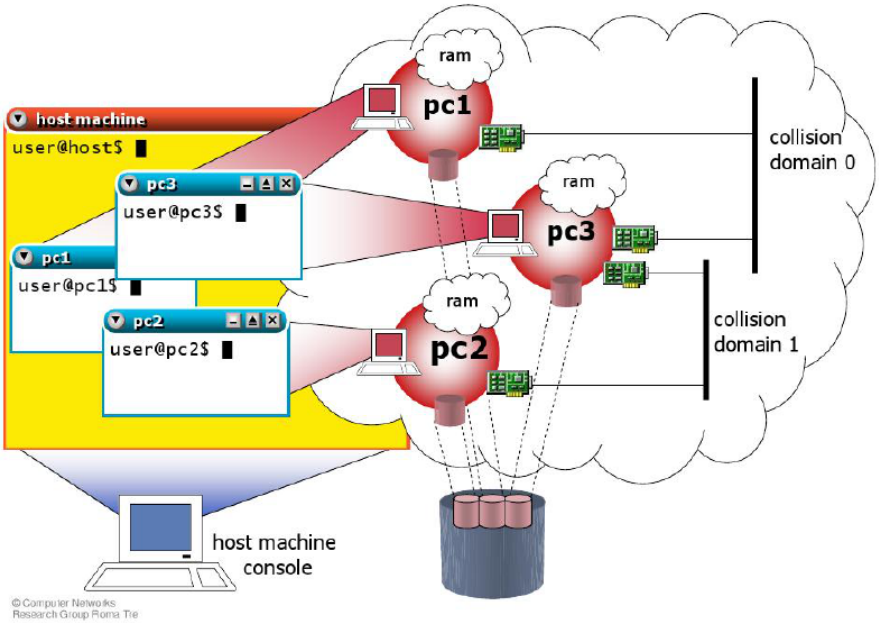
L'organisation basique d'un *lab* est la suivante :

- un fichier `lab.conf` contenant le nom des machines (pc1 et pc2 par exemple) et la configuration réseau (une domaine de collision A partagé entre l'interface 0 (ou eth0) de la machine pc1 et l'interface 0 (ou eth0) de la machine pc2.

- un fichier `pc1.startup` et un fichier `pc2.startup` contenant les commandes à exécuter respectivement sur les machines pc1 et pc2 après le démarrage de chacune.

- un répertoire `pc1` et un répertoire `pc2` contenant les fichiers à copier sur machine virtuelle après le démarrage de chacune.

```
$ cat lab.conf
pc1[0]="A" # eth0 de pc1 connecté au domaine A
pc2[0]="B"
pc3[0]="A"
pc3[1]="B" # eth1 de pc3 connecté au domaine B
```



Netkit fournit deux groupes de commandes :

- les `vcommandes`, préfixées par '**v**' qui servent pour manipuler une seule VM

- les `lcommandes`, préfixées par '**l**' qui servent à manipuler des ensembles complexes de machines virtuelles en réseau (lab)

Par exemple dans le repertoire du *lab* :

- `vstart pc1` démarre manuellement la machine `pc1`

- `vstart pc1 --eth0=A` démarre manuellement la machine `pc1` avec son interface `eth0` relié au domaine de collision `A`

- `vconfig pc1 --eth2=F` ajoute une interface `eth2` relié au domaine de collision `F`

- `lstart -s` (séquentiel) ou `lstart -p` (parallèle) démarre l'ensemble des machines du *lab*.

Et pour arrêter les machines :

- soit utiliser la commande `halt` dans le terminal de chaque machine virtuelle à arrêter.

- soit taper `vhalt pc1` dans le repertoire du *lab*, ...

- soit taper `lhalt -q` dans le repertoire du *lab*.

Il est possible de lire et écrire des fichiers entre la machine virtuelle et la machine hôte par l'intermédiaire de deux repertoires spéciaux disponibles dans chaque machine virtuelle :

- `/hosthome` sur la virtuelle représente le *home directory* sur la machine réelle.

- `/hostlab` sur la machine virtuelle représente le répertoire du *lab* sur la machine réelle.

Kathará

Installation

Sous Ubuntu 18.04 (facultatif) :

```
sudo apt -y install docker.io

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 21805
    A48E6CBBA6B991ABE76646193862B759810 add-apt-repository -y ppa:katharaframework/kathara
echo "deb http://ppa.launchpad.net/katharaframework/kathara/ubuntu bionic main" | sudo tee /
    etc/apt/sources.list.d/kathara.list
echo "deb-src http://ppa.launchpad.net/katharaframework/kathara/ubuntu bionic main" | sudo
    tee -a /etc/apt/sources.list.d/kathara.list

sudo apt update

sudo apt -y install kathara
```

Il est possible d'éditer la configuration de base de Kathará (notamment pour choisir le terminal utilisé par les machines) :

```
$ vim ~/.config/kathara.conf
{
 "image": "kathara/quagga",
 "manager_type": "docker",
 "terminal": "/usr/bin/gnome-terminal",
 "open_terminals": true,
 "hosthome_mount": false,
 "shared_mount": true,
 "device_shell": "/bin/bash",
 "net_prefix": "kathara",
 "device_prefix": "kathara",
 "debug_level": "INFO",
 "print_startup_log": true,
 "last_checked": 1590234983.2906733,
 "enable_ipv6": false
}
```

Ou en tapant la commande :

```
$ kathara settings
```

On termine par une vérification :

```
$ kathara check
```

En TP, on utilisera une machine virtuelle préconfigurée.

Vagrant est un logiciel libre et open-source pour la création et la configuration des environnements de développement virtuel. Il peut être considéré comme un wrapper autour de logiciels de virtualisation comme **VirtualBox** ou VMware. Vagrant s'utilise via une interface en ligne de commande (CLI). Dans Vagrant, une **Box** (boîte) est une machine virtuelle préconfigurée (*template*). **Vagrant Cloud** (https://app.vagrantup.com/boxes/search) sert de plateforme d'échange pour trouver des boîtes et y déposer ses propres boîtes. Lire : Vagrant

Pré-requis :

— Installer VirtualBox
— Installer Vagrant

```
$ sudo apt update
$ sudo apt install virtualbox
$ sudo apt install vagrant

# ou :
$ wget https://releases.hashicorp.com/vagrant/2.2.9/vagrant_2.2.9_x86_64.deb
$ sudo apt install ./vagrant_2.2.9_x86_64.deb

$ vagrant --version
```

Ensuite :

```
$ vagrant init tvaira/lubuntu-kathara-bts
```

Il est possible de paramétrer la machine virtuelle en éditant le fichier `Vagrantfile` (ici on va activer l'interface graphique pour l'accès à la machine virtuelle) :

```
$ vim Vagranfile
Vagrant.configure("2") do |config|
  config.vm.box = "lubuntu-kathara-bts"
  config.vm.provider "virtualbox" do |vb|
    vb.gui = true
  #   vb.cpus = 2
  #   vb.memory = 2048
  end
end
```

Et on démarre la machine virtuelle :

```
$ vagrant up
```

Pour l'arrêter, on fera :

```
$ vagrant halt
```

Utilisation

L'accès à la machine virtuelle se fait avec le compte `vagrant` et le mot de passe `vagrant`. L'environnement de la machine virtuelle préconfigurée est **Lubuntu 18.04**.

SSH

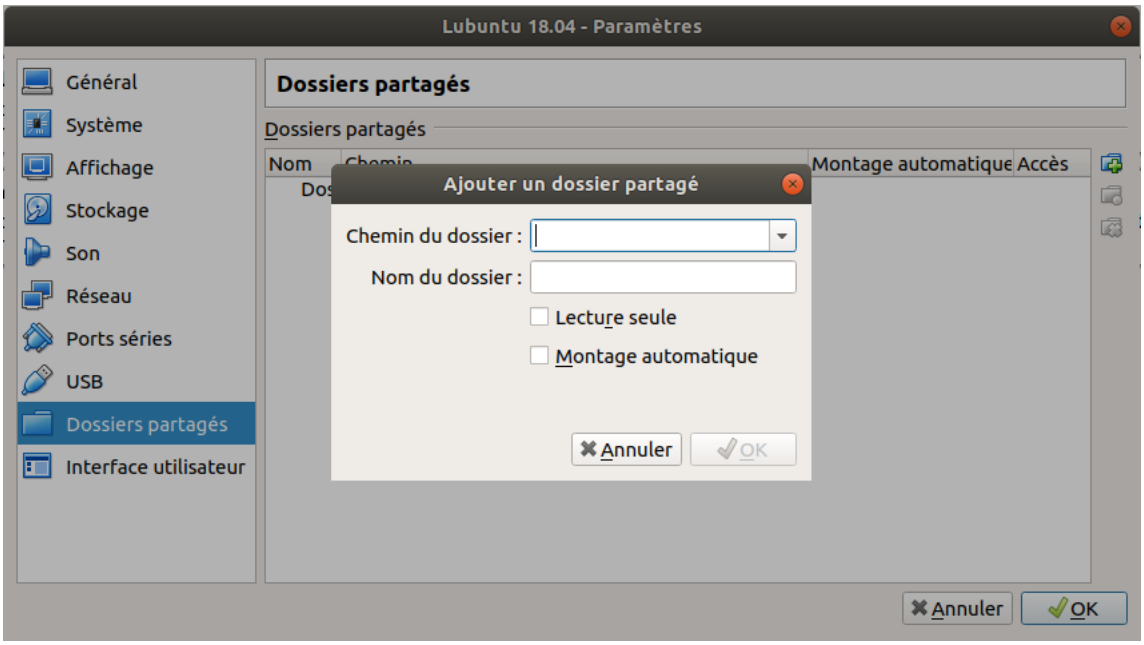
Un accès `ssh` est possible :

```
$ vagrant ssh -- -X

# ou :
$ ssh -X -p 2222 vagrant@127.0.0.1
```

Il est possible d'« échanger » des fichiers avec la machine virtuelle :

— la commande `scp` : `scp -p 2222 <fichier-source> vagrant@127.0.0.1:<destination>`
— en se connectant par l'intermédiaire du gestionnaire de fichier :

— en créant un dossier partagé avec VirtualBox :



et la commande `mount` : `sudo mount -t vboxsf <partage> <chemin>`

TP

Les maquettes

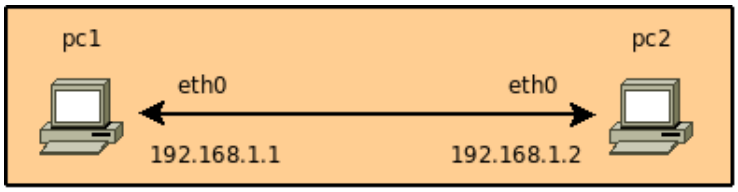
Les maquettes des TPs sont fournis sous forme d'archive `.tgz` (ou `.tar.gz`).

Les sujets des TPs ont été pré-installés dans `$HOME/Documents/tp-reseaux`.

Exemple :

```
$ tar zxvf tp1.tgz
tp1/
tp1/pc1.startup
tp1/lab.conf
tp1/pc2/
tp1/pc1/
tp1/pc2.startup

$ tree tp1
tp1
+-- lab.conf
+-- pc1
+-- pc1.startup
+-- pc2
+-- pc2.startup

$ cd tp1
```

```
$ cat lab.conf
pc1[0]="A"
pc2[0]="A"
```

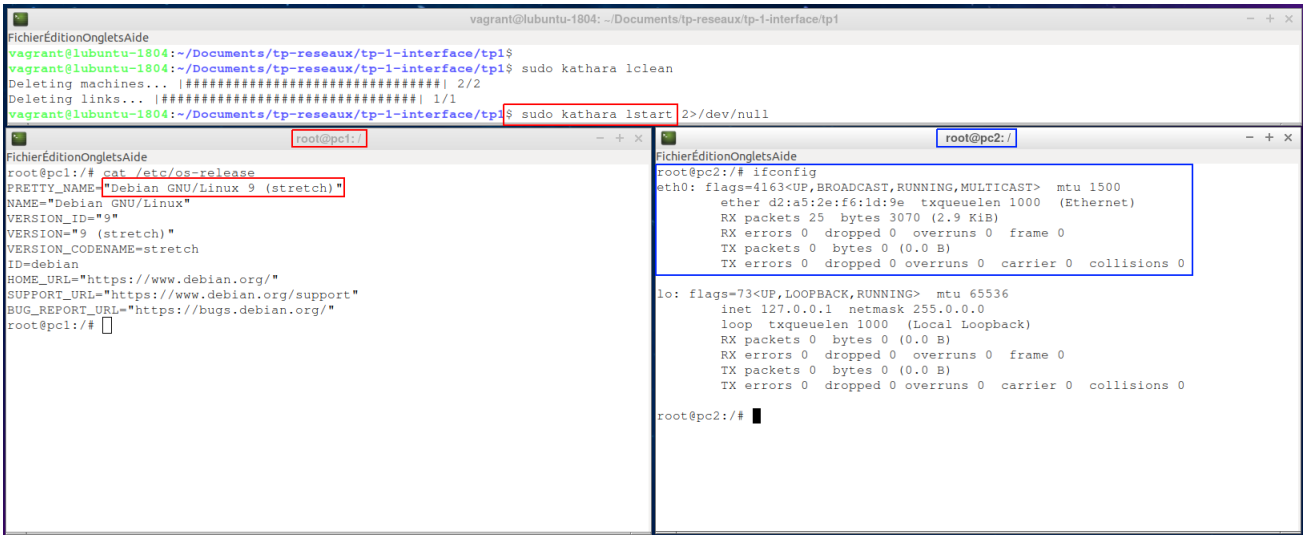


La liste des commandes kathara sont accessibles à partir du manuel : `man kathara`

Démarrer

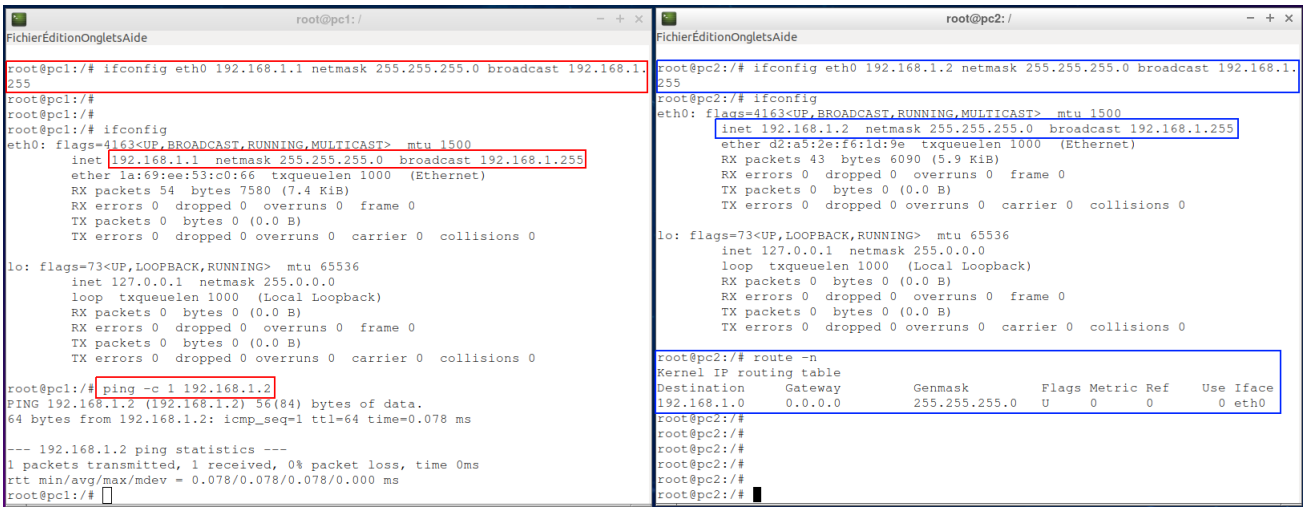
Démarrer la maquette (*lab*) :

```
$ sudo kathara lstart
```

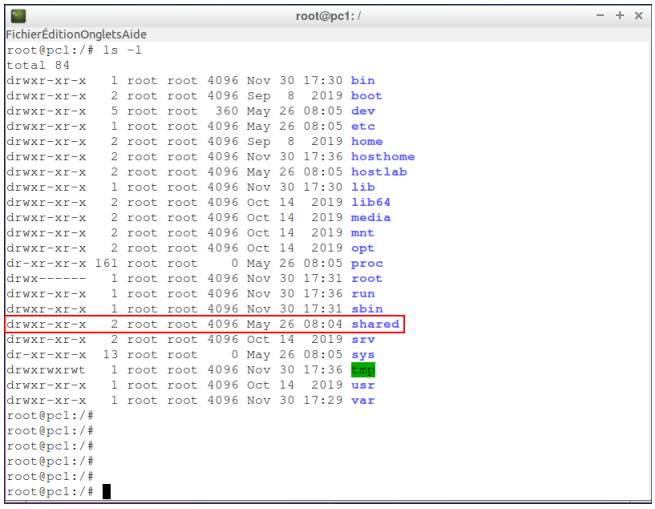


Avec l'image **Docker** utilisé ici ("kathara/quagga"), on dispose de « machines » sous **GNU/Debian 9 (stretch)**.

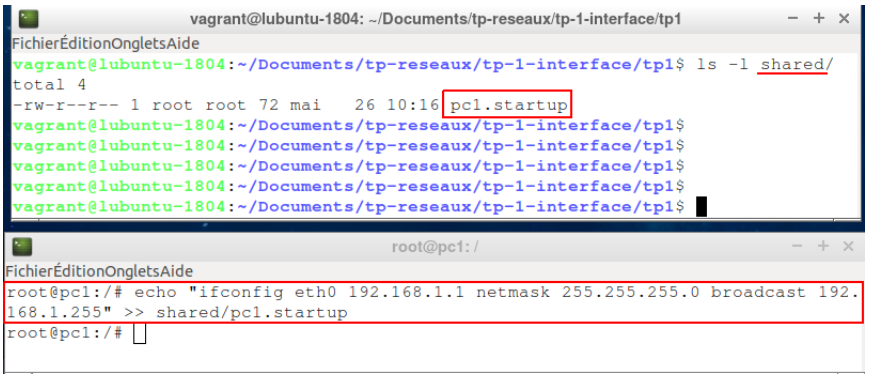
On peut utiliser des commandes GNU/Linux pour configurer et tester le réseau :



Les machines disposent d'une espace partagé avec l'hôte à partir du répertoire **/shared** :

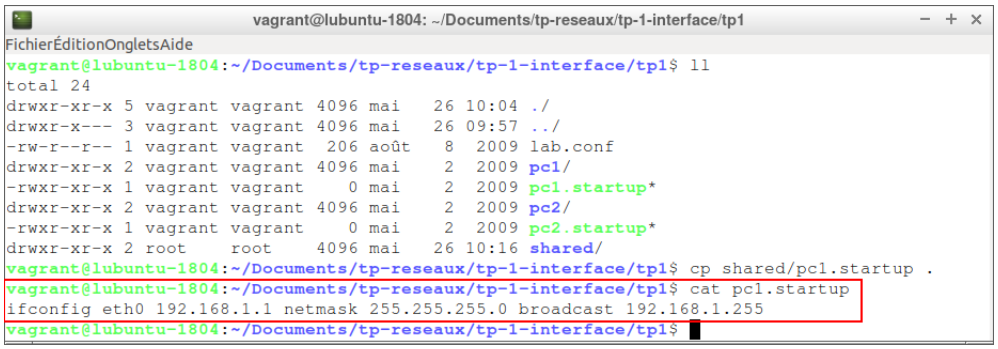


Il est donc possible d'y sauvegarder des commandes et/ou des fichiers :

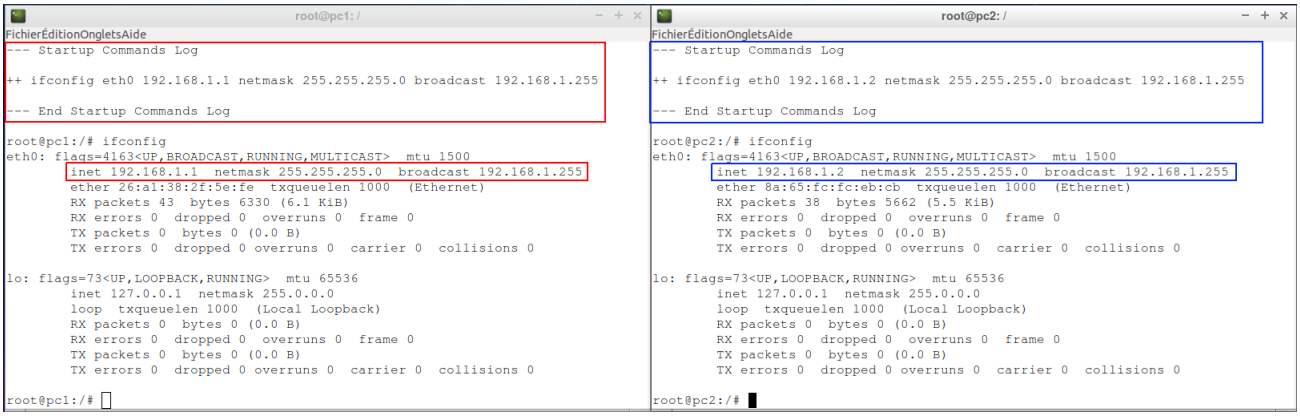


Démarrage automatique des machines

Ceci permettra de configurer les fichiers `*.startup` avec des commandes à exécuter au démarrage :



Maintenant, au démarrage de la maquette (*lab*), les machines ont exécuté les commandes de configuration d'interface :

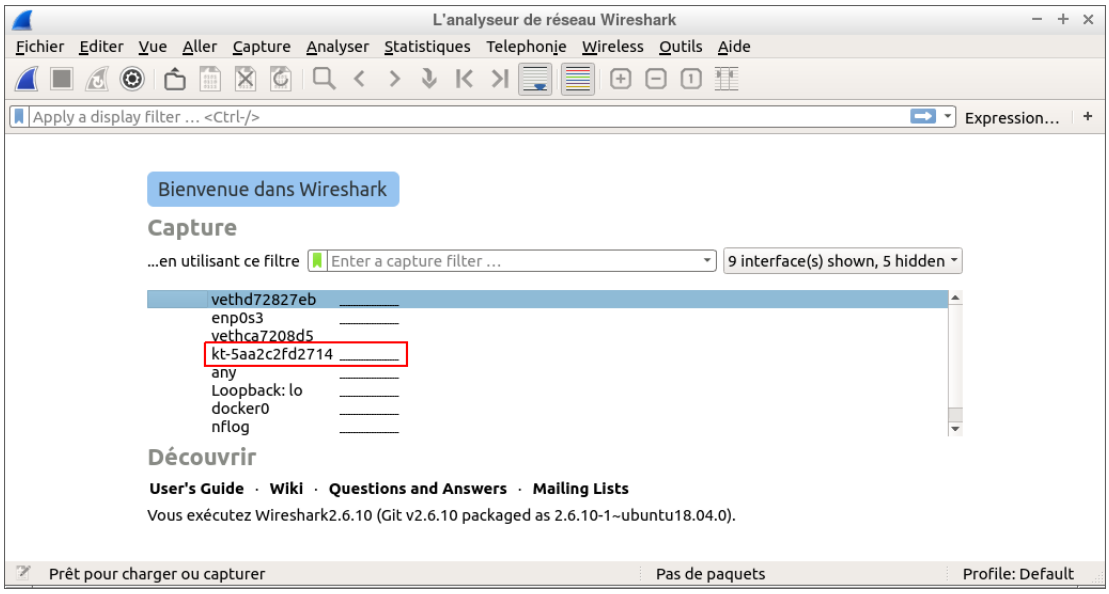


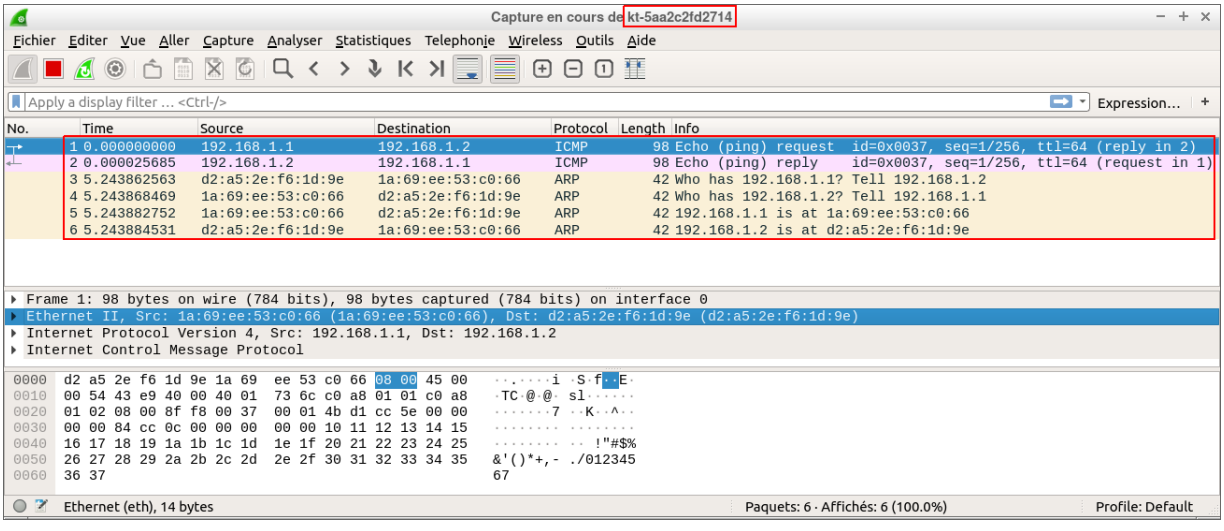
Capture du trafic réseau

Les réseaux créés dans la maquette sont « accessibles » depuis l'hôte :

```
vagrant@lubuntu-1804: ~/Documents/tp-reseaux/tp-1-interface/tp1
Fichier Édition Onglets Aide
vagrant@lubuntu-1804:~/Documents/tp-reseaux/tp-1-interface/tp1$
vagrant@lubuntu-1804:~/Documents/tp-reseaux/tp-1-interface/tp1$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
b15fdfc8ee6b        kathara/quagga      "bash"              12 minutes ago      Up 12 minutes                           kathara_vagrant_pc2_s6MAjgZlaq79AieMU5nGkw
eaa7fa5e8fbf        kathara/quagga      "bash"              12 minutes ago      Up 12 minutes                           kathara_vagrant_pc1_s6MAjgZlaq79AieMU5nGkw
vagrant@lubuntu-1804:~/Documents/tp-reseaux/tp-1-interface/tp1$ sudo kathara list
TIMESTAMP: 2020-05-26 10:18:47.046325

| LAB HASH | USER | MACHINE NAME | STATUS | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O |
|---|---|---|---|---|---|---|---|
| s6MAjgZlaq79AieMU5nGkw | vagrant | pc1 | running | 0.00% | 4.02 MB / 985.16 MB | 0.41% | 8.49 KB / 182.0 B |
| s6MAjgZlaq79AieMU5nGkw | vagrant | pc2 | running | 0.00% | 3.45 MB / 985.16 MB | 0.35% | 7.1 KB / 182.0 B |

vagrant@lubuntu-1804:~/Documents/tp-reseaux/tp-1-interface/tp1$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
2a474b697d5b        bridge              bridge              local
55f6e6d10d7f        host                host                local
5aa2c2fd2714        kathara_vagrant_A   kathara/katharanp:stretch   local
c80eb938ac38        none                null                local
vagrant@lubuntu-1804:~/Documents/tp-reseaux/tp-1-interface/tp1$
```

Ceci permet de capturer le trafic à partir de `wireshark` :



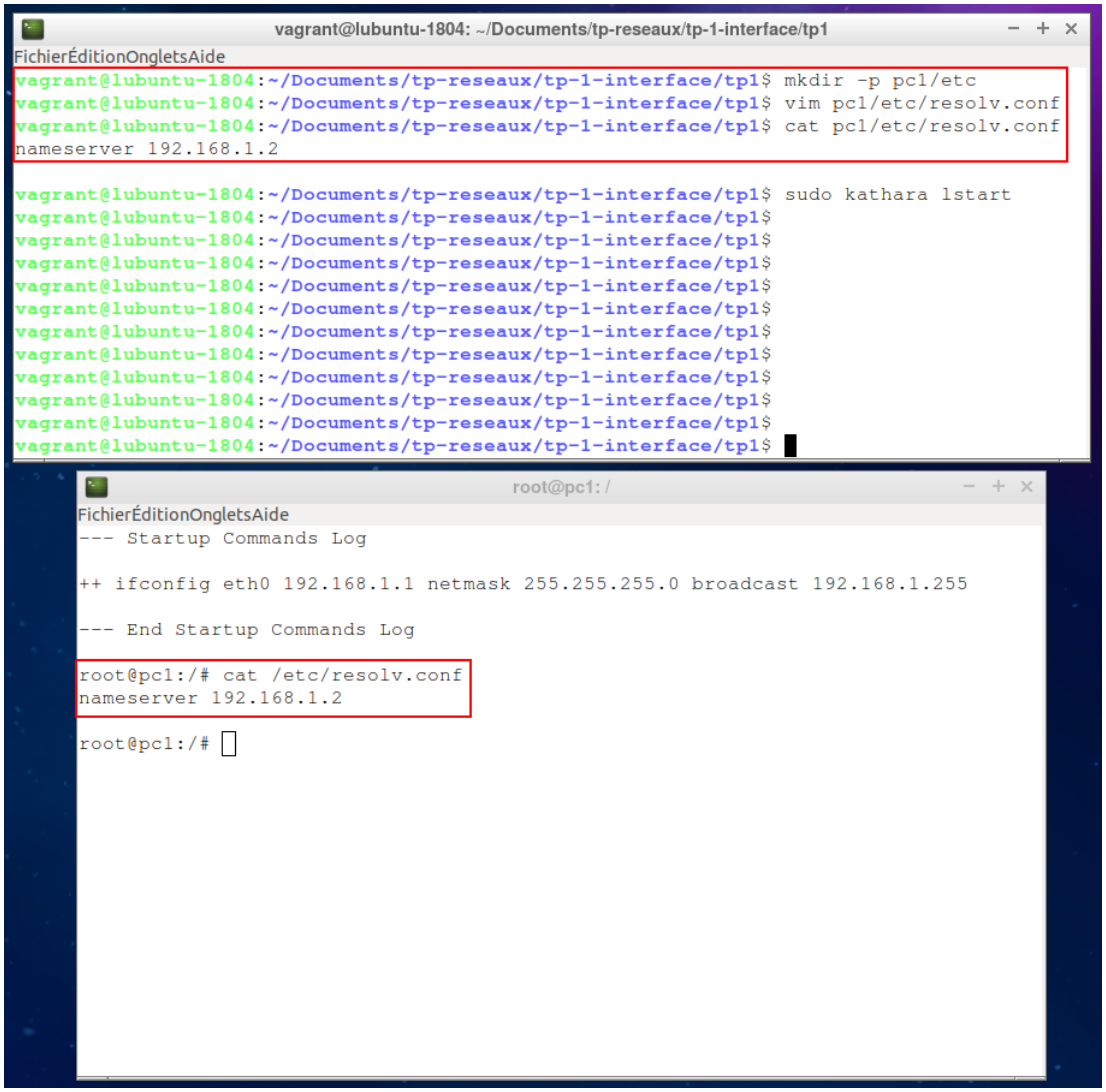


L'utilisation de **wireshark** est décrite dans la fiche http://tvaira.free.fr/bts-sn/reseaux/fiches/fiche-wireshark.pdf et dans ce TP http://tvaira.free.fr/bts-sn/reseaux/tp-autres/tp-reseaux-wi
pdf. On utilisera aussi l'outil **tcpdump** pour *sniffer* le trafic réseau directement à partir des machines. Voir : http://tvaira.free.fr/bts-sn/reseaux/fiches/fiche-tcpdump.pdf

Configuration des machines

Dans certains TP, on a besoin par exemple de configurer des services sur les machines à partir de fichier de configuration. Pour cela, on utilise l'arborescence du *lab*. En effet dans la maquette du *lab*, chaque machine possède un répertoire du même nom dont l'arborecence sera copiée au démarrage.

Exemple pour le fichier `/etc/resolv.conf` :



Ajout de paquets

Il est aussi possible qu'on ait besoin de logiciels supplémentaires (`lynx`, `dnsmasq`, etc.) non installés à l'origine. Par défaut, les machines ne sont pas reliés aux sous-réseau de l'hôte et n'ont donc pas accès à Internet. Dans ce cas, on peut soit « passer » les paquets à installer par l'arborescence partagée (dans `/opt` de la machine ou par `/shared`) soit ajouter une interface *bridge* (voir plus loin).

```
# dnsmasq :

$ dpkg -i /opt/dns-root-data_2019031302~deb9u1_all.deb
$ dpkg -i /opt/dnsmasq-base_2.76-5+deb9u2_amd64.deb
$ dpkg -i /opt/dnsmasq_2.76-5+deb9u2_all.deb

# lynx :

$ dpkg -i /opt/lynx_2.8.9dev11-1_amd64.deb
$ dpkg -i /opt/lynx-common_2.8.9dev11-1_all.deb
```

```
# netcat :

$ dpkg -i /opt/netcat-openbsd_1.130-3_amd64.deb
$ dpkg -i /opt/netcat-traditional_1.10-41+b1_amd64.deb

# ufw :

$ dpkg -i /opt/ufw_0.35-4_all.deb
```

Paquetages pour la Debian GNU/Linux 9 (stretch) : https://packages.debian.org/search?keywords=search

Arrêter

Arrêter la maquette (*lab*) :

```
$ sudo kathara lclean
```

Les machines sont détruites et leurs configurations aussi !

Mode bridge

Il est possible d'utiliser une interface de la machine pour la connecter au réseau hôte via une connexion NAT.

Le sous-réseau 172.17.0.0/16 est réservé pour l'utilisation de Kathara. Il ne faut donc pas l'utiliser pour définir des plans d'adressage des sous-réseaux.

Exemple pour une machine autonome

Sur l'hôte :

```
vagrant@lubuntu-1804:~$ sudo kathara vstart -n router --bridged --eth 0:A

vagrant@lubuntu-1804:~$ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        inet6 fe80::42:bff:feb2:6f4f prefixlen 64 scopeid 0x20<link>
        ether 02:42:0b:b2:6f:4f txqueuelen 0 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 24 bytes 3456 (3.4 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe98:97c7 prefixlen 64 scopeid 0x20<link>
        ether 08:00:27:98:97:c7 txqueuelen 1000 (Ethernet)
        RX packets 823 bytes 978737 (978.7 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 353 bytes 31911 (31.9 KB)
```

```
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
...
```

Sur la machine `router` :

```
root@router:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        ether 46:e3:d5:b8:9e:e5 txqueuelen 1000 (Ethernet)
        RX packets 55 bytes 7410 (7.2 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 87 bytes 11277 (11.0 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 11 bytes 825 (825.0 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 16 bytes 1168 (1.1 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 16 bytes 1168 (1.1 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@router:/# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0        172.17.0.1      0.0.0.0         UG    0      0        0 eth1
172.17.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth1

root@router:/# cat /etc/resolv.conf
nameserver 8.8.8.8
options edns0
search home

root@router:/# ping -c 1 www.google.fr
PING www.google.fr (172.217.18.35) 56(84) bytes of data.
64 bytes from ham02s12-in-f3.1e100.net (172.217.18.35): icmp_seq=1 ttl=61 time=4.22 ms

--- www.google.fr ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 4.223/4.223/4.223/0.000 ms
```

Exemple pour une machine d'un *lab*

Sur l'hôte :

Il faut ajouter une interface réseau supplémentaire et la connecter au réseau hôte via une connexion NAT avec l'argument `bridged`.

```
$ cat lab.conf
```

```
pc1[bridged]=true

pc1[0]="A"
pc2[0]="A"
```

```
vagrant@lubuntu-1804:~$ sudo kathara lstart
```

Sur la machine pc1 :

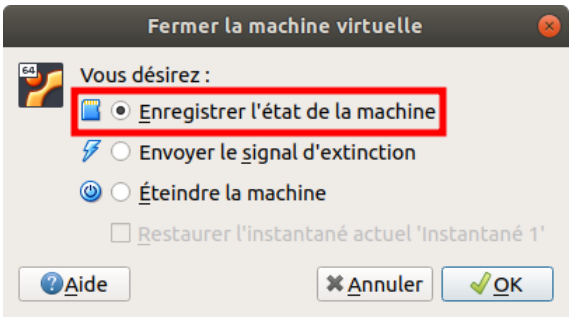
```
root@pc1:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
        ether 9e:78:9b:dc:e8:e8 txqueuelen 1000 (Ethernet)
        RX packets 26 bytes 3840 (3.7 KiB)
        RX errors 0 dropped 0 overruns 0 frame 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 43 bytes 6436 (6.2 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
```

Sauvegarde

Vous pouvez fermer les terminaux des machines kathara mais ne pas arrêter le *lab*. Il faut ensuite quitter la session ssh avec la commande `exit` sans arrêter les machines kathara :

```
$ exit
déconnexion
Connection to 127.0.0.1 closed.
```

Il est possible de sauvegarder l'état d'une machine virtuelle VirtualBox à la fermeture de celle-ci :



Ensuite, vous redémarrez la machine virtuelle VirtualBox à partir de l'état sauvegardé et relancez une session `ssh` :

```
$ vagrant ssh -- -X

# ou :
$ ssh -X -p 2222 vagrant@127.0.0.1
```

Vous pouvez lister les machines du *lab* :

Il faut tout d'abord se déplacer dans le répertoire du *lab* !

```
$ sudo kathara list

LAB HASH              USER    MACHINE NAME STATUS  CPU %  MEM USAGE / LIMIT  MEM %  NET I/O
...                   vagrant pc1          running 0.00%  9.27 MB / 985.16 MB 0.94%  3.52 KB
...                   vagrant pc2          running 0.00%  1.88 MB / 985.16 MB 0.19%  3.1 KB
```

Lorsqu'elles sont dans l'état *running*, il est possible alors de se reconnecter avec un nouveau terminal en indiquant leur nom (`pc1` et `pc2` ici) :

```
$ lxterminal --command="sudo kathara connect pc1"
$ lxterminal --command="sudo kathara connect pc2"
```