Raspberry Pi PXE Boot – Démarrage réseau d'un Pi 4 sans carte SD

Table of Contents

:

Ce tutoriel Raspberry Pi PXE Boot guide à travers le démarrage réseau d'un Raspberry Pi 4 sans carte SD. On utilisera un autre Raspberry Pi 4 avec une carte SD comme serveur netboot.

Les objectifs de ce tutoriel sont :

- Simplifier le provisionnement et la maintenance de Pi autant que possible.
- Automatiser autant que possible les mises à jour/mises à niveau.

Le démarrage en réseau est un bon moyen d'y parvenir. Par exemple, lorsqu'on démarre un Pi en réseau, il ne nécessite pas de carte SD pour démarrer. Le système d'exploitation et le système de fichiers vivent sur un serveur central. Parce que la plupart des approvisionnements se font sur un serveur central, on peut éventuellement l'automatiser via des scripts.

Présentation de PXE

Qu'est-ce que le PXE, comment ça marche ?

PXE signifie Preboot Execution Environment. À un niveau élevé, PXE est une norme pour le démarrage réseau d'un ordinateur. Il utilise des protocoles réseau standard pour réaliser le démarrage réseau. Plus précisément IP, UDP, DHCP et TFTP. PXE est généralement utilisé de l'une des deux manières suivantes :

- Amorçage initial ou approvisionnement d'un serveur compatible réseau. Dans ce cas d'utilisation, le processus de démarrage PXE initialise le système en installant un système d'exploitation sur le stockage local. Par exemple, utiliser dd pour écrire une image disque ou utiliser un programme d'installation de préconfiguration Debian.
- Systèmes sans disque qui démarrent toujours à partir du réseau. Par exemple, le processus que nous suivons dans ce tutoriel.

Le flux de démarrage PXE. Les implémentations peuvent différer. De plus, les composants du serveur peuvent être répartis sur plusieurs hôtes.

Description du flux PXE

- Le client s'allume, le micrologiciel de la carte d'interface réseau du client envoie une requête DHCP sur le réseau.
- Un serveur DHCP répond avec une offre de bail DHCP. Cette offre de location aura une option pour le « prochain-serveur ». La valeur de l'option “next-server” est l'adresse IP ou le nom du serveur à partir duquel le client téléchargera ses fichiers de démarrage initiaux. Le champ next-server est connu sous le nom d'option 66 dans le protocole DHCP.
- Le client télécharge les fichiers via TFTP à partir de l'hôte spécifié dans le champ next-server du bail. Généralement, les fichiers sont un noyau et une image initrd. Cependant, il pourrait s'agir d'autre chose. Par exemple, il pourrait charger en chaîne un chargeur de démarrage réseau ou un autre client PXE comme IPXE.
- Le client démarre les fichiers téléchargés et startes son processus de sangle de démarrage.
- À ce stade, le client peut démarrer une installation du système d'exploitation ou démarrer en tant que système sans disque.

Phase 1 - Configuration du client PXE

Configuration du client RPi3

piPXE disponible ici est une version du micrologiciel de démarrage réseau **iPXE** pour le Raspberry Pi.

L'image de la carte SD contient le Micrologiciel **TianoCore EDK2 UEFI** conçu pour la plate-forme RPi3

Télécharger `sdcard.img` et le graver sur n'importe quelle carte micro SD vierge à l'aide d'un outil tel que dd ou Etcher.

Pour compiler à partir des sources, il faut divers outils de construction, y compris une version de compilation croisée de gcc pour construire des binaires AArch64.

Outils de compilation Fedora :

```
sudo dnf install -y binutils gcc gcc-aarch64-linux-gnu \
```

```
                    git-core iasl libuuid-devel make \
                    mtools perl python subversion xz-devel
```

Outils de compilation Ubuntu :

```
sudo apt install -y build-essential gcc-aarch64-linux-gnu \
                    git iasl lzma-dev mtools perl python \
                    subversion uuid-dev
```

Cloner ce référentiel et exécuter make. Cela construira tous les composants requis et générera éventuellement l'image de la carte SD `sdcard.img`.

Configuration du client RPi4

Mise à jour de l'EEPROM

Le Raspberry Pi 4 possède une **EEPROM** capable de démarrer en réseau. Pour configurer le démarrage réseau il faut démarrer le système au moins une fois avec un système linux sur une carte SD.

Commencer par configurer le système client pour netboot. C'est le Raspberry Pi qui finira par démarrer sans carte micro SD installée.

- Télécharger Raspbian Lite `wget https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2021-05-28/2021-05-07-raspios-buster-armhf-lite.zip`
- Copier l'image Buster sur une carte SD en utilisant la commande dd suivante, (remplacer sdX par le périphérique de carte SD). Avertissement! Cela écrasera les données sur le périphérique spécifié: `sudo dd if=2019-09-26-raspbian-buster-lite.img of=/dev/sdX bs=4M` (Si le carte SD contient déjà une table de partition, le système peut la monter automatiquement lors de l'insertion. Démonter ou éjecter tous les volumes montés à partir de la carte micro SD. Ensuite, utiliser la commande dd pour copier l'image sur le carte micro SD. La commande dd prend quelques minutes sur mon ordinateur portable).
- Insérer la carte SD dans le client Raspberry Pi 4 et démarrer. L'utilisation de la version allégée de raspbian ne donne qu'une console texte, pour une console graphique, utiliser la version complète et cela devrait fonctionner.
- Se connecter via la console en utilisant le login par défaut : pi/raspberry
- Connecter le Raspberry Pi à Internet via un câble Ethernet.
- Mettre à jour le système d'exploitation Raspbian via apt-get et installer le programme rpi-config :

```
sudo apt-get update
sudo apt-get full-upgrade
sudo apt-get install rpi-eeprom
```

Examiner ensuite la configuration de le chargeur de démarrage à l'aide de cette commande :

```
```

```
vcgencmd bootloader_config
```

Voici la sortie sur un Raspberry Pi 4 tout juste sorti de la boîte :

```
pi@raspberrypi:~ $ vcgencmd bootloader_config
 BOOT_UART=0
 WAKE_ON_GPIO=1
 POWER_OFF_ON_HALT=0
 FREEZE_VERSION=0
```

On doit modifier la configuration du chargeur de démarrage pour démarrer à partir du réseau à l'aide du paramètre **BOOT_ORDER**. Pour ce faire, il faut l'extraire de l'image **EEPROM**. Une fois extrait, apporter les modifications pour activer le démarrage PXE. Enfin, le réinstaller dans le chargeur de démarrage.

On le fait avec ces étapes :

- Accéder au répertoire où sont stockées les images du chargeur de démarrage :

```
cd /lib/firmware/raspberrypi/bootloader/beta/
```

- Faire une copie du dernier fichier image du micrologiciel. Dans le cas, c'était pieeprom-2019-11-18.bin :

```
cp pieeprom-2019-11-18.bin nouveau-pieeprom.bin
```

- Extraire la config de l'image eeprom

```
rpi-eeprom-config new-pieeprom.bin > bootconf.txt
```

- Dans `bootconf.txt`, remplacer la variable **BOOT_ORDER=0x1** par **BOOT_ORDER=0x21**:
  - **0X1** signifie uniquement démarrer à partir de la carte SD.
  - **0x21** signifie essayer d'abord le démarrage de la carte SD, puis le démarrage du réseau.
- Enregistrer maintenant le nouveau fichier bootconf.txt dans l'image du micrologiciel qu'on a copié précédemment :

```
rpi-eeprom-config --out netboot-pieeprom.bin --config bootconf.txt new-
pieeprom.bin
```

- Installer maintenant le nouveau chargeur de démarrage :

```
sudo rpi-eeprom-update -d -f ./netboot-pieeprom.bin
```

- Si on a une erreur avec la commande ci-dessus se produit, vérifier que le mise à niveau complète apt-get s'est terminée avec succès.

 **Désactivation de la mise à jour automatique rpi-eeprom**: `rpi-update` se mettra à

jour par défaut. **rpi-eeprom-update** fait cela. Une mise à jour du micrologiciel pourrait désactiver le démarrage PXE dans l'eeprom. On peut désactiver les mises à jour automatiques en masquant la mise à jour rpi-eeprom via systemctl. On peut mettre à jour manuellement l'eeprom en exécutant rpi-eeprom-update lorsqu'on le souhaite.

```
sudo systemctl mask rpi-eeprom-update
```

On est à mi-chemin du premier démarrage net. Le client de démarrage réseau Raspberry Pi est configuré pour le démarrage PXE. Avant d'arrêter le Pi 4, noter l'adresse MAC de l'interface Ethernet. On peut le faire en exécutant `ip addr show eth0` et en copiant la valeur du champ link/ether.

Configuration du démarrage UEFI

Une autre méthode pour démarrer en réseau le rPI avec d'autres systèmes d'exploitation prenant en charge **PXE/iPXE** consiste à utiliser **UEFI** sur le rPI lui-même via la carte SD, ce qui permet de démarrer tout ce dont on a besoin via **UEFI**.

Dans un premier temps il faut compiler le firmware **EDK2 Raspberry Pi 4 UEFI** (RPI_EFI.fd).

Créer un nouveau dossier (répertoire) sur l'ordinateur de développement local à utiliser comme espace de travail, par exemple `/work/git/tianocore`:

```
export WORKSPACE=/work/git/tianocore
mkdir -p $WORKSPACE
cd $WORKSPACE
```

Dans ce dossier, cloner les sources de:

1. edk2
2. edk2-platforms
3. edk2-non-osi (pour construire des plates-formes qui en ont besoin)

Le commit du 3 février 2023 ajoute la prise en charge d'EFI*MP*SERVICES_PROTOCOL pour la Plate-forme/RPi4 mais l'application de test n'est pas ajoutée à l'image, cela provoque un echec de la construction il est donc préférable d'utiliser une version précédente.

La version des commits validés au 15 décembre 2022 a abouti:
- edk2 commit du 15/12/2022 0adc35fccd59c8c5171273319ec899aa48fc2c35
- edk2-platforms commit du 15/12/2022 20e07099d8f11889d101dd710ca85001be20e179
-edk2-non-osi commit du 03/06/2022 61662e8596dd9a64e3372f9a3ba6622d2628607c

```
git clone https://github.com/tianocore/edk2.git
cd edk2
```

```
git checkout 0adc35fccd59c8c5171273319ec899aa48fc2c35
git submodule update --init
cd ../
git clone https://github.com/tianocore/edk2-platforms.git
cd edk2-platforms
git checkout 20e07099d8f11889d101dd710ca85001be20e179
git submodule update --init
cd ../
git clone https://github.com/tianocore/edk2-non-osi.git
cd edk2-non-osi/
git checkout 61662e8596dd9a64e3372f9a3ba6622d2628607c
git submodule update --init
cd ../
```

Configurer **PACKAGES_PATH** pour pointer vers les emplacements de ces trois dépôts :

```
export PACKAGES_PATH=$PWD/edk2:$PWD/edk2-platforms:$PWD/edk2-non-osi
```

Configurer l'environnement de construction (cela modifiera les variables d'environnement)

```
edk2/edksetup.sh
```

Construire **BaseTools** (ne peut actuellement pas être construit en parallèle, donc on ne peut spécifier aucune option -j, que ce soit sur la ligne de commande ou dans une variable d'environnement MAKEFLAGS.)

```
make -C edk2/BaseTools
```

Options de construction: Il existe un certain nombre d'options qui peuvent (ou doivent) être spécifiées au moment de la construction. Leurs valeurs par défaut sont définies dans edk2/Conf/target.txt. Si on ne travaille que sur une seule plate-forme, il est logique de simplement mettre à jour ce fichier.

target.txt	option de la ligne de commande	Description
ACTIVE_PLATFORM	-p	Fichier de description (.dsc) de la plateforme.
TARGET	-b	Au choix DEBUG, RELEASE ou NOOPT.
TARGET_ARCH	-a	architecture pour laquelle construire.
TOOL_CHAIN_TAG	-t	Profil de chaîne d'outils à utiliser pour la construction.
MAX_CONCURRENT_THREAD_NUMBER	-n	à peu près équivalent à make -j.

Lorsqu'il est spécifié sur la ligne de commande, -b peut être répété plusieurs fois afin de créer plusieurs cibles de manière séquentielle.

Après une compilation réussie, les images résultantes peuvent être trouvées dans `Build/{Platform Name}/{TARGET}_{TOOL_CHAIN_TAG}/FV`.

Le processus de construction principal peut s'exécuter en parallèle - alors avant de le lancer, déterminer le nombre de threads dont on dispose avec la commande `getconf _NPROCESSORS_ONLN`

puis indiquer à la construction d'en utiliser un peu plus :`NUM_CPUS=$((getconf _NPROCESSORSONLN + 2))`

Pour lancer le processus de construction principal il faut utiliser la balise de chaîne d'outils, utiliser **GCC5** pour gcc version 5 ou ultérieure, **GCC4x** pour les versions antérieures, ou **CLANG35/CLANG38** selon le cas lors de la construction avec clang.

```
build -n $NUM_CPUS -a AARCH64 -t GCC5 -p Platform/RaspberryPi/RPi4/RPi4.dsc
```

Repérer dans les sorties la ligne commençant par **Fd File**, elle donne l'emplacement et la nom du fichier compilé Fd (dans le cas de Raspberry `RPI_EFI.fd` comme ci-dessous:

```
Name Fd File Name:RPI_EFI (/media/ubuntu/9fbabb5b-c3fa-4417-9472-73c786a19de6/Data/SynologyDrive/repos/os/tianocore/Build/RPi4/DEBUG_GCC5/FV/RPI_EFI.fd)
```

Le fichier de description est résolu par la commande build en recherchant dans tous les emplacements spécifiés dans PACKAGES_PATH.

Pour une compilation croisée ou de la construction avec une version du compilateur différente de la version par défaut de gcc ou clang(/binutils), il faut en outre informer la commande build de la chaîne d'outils à utiliser. Pour ce faire, définir la variable d'environnement **{TOOL_CHAIN_TAG}_{TARGET_ARCH}_PREFIX** - dans le cas ci-dessus, `GCC5_AARCH64_PREFIX=aarch64-linux-gnu-`.

Utilisation des scripts d'aide uefi-tools:

uefi-tools est un ensemble totalement non officiel de scripts d'assistance développés par Linaro. Ils automatisent la détermination de toutes les options manuelles ci-dessus et stockent les chemins d'accès aux fichiers de description de la plate-forme dans un fichier

de configuration séparé. De plus, ils simplifient la création en masse d'un grand nombre de plates-formes.

L'intention (dans la mesure du possible) est de maintenir cette configuration à jour avec toutes les plates-formes qui existent dans la branche principale edk2-platforms.

- Récupérer le dépôt **uefi-tools**: `git clone https://git.linaro.org/uefi/uefi-tools.git`

- Exécuter le script de **edk2-build.sh**: `./uefi-tools/edk2-build.sh RPi4`

La construction se termine par un résumé des plates-formes/cibles qui ont été construites, qui ont réussi et qui ont échoué (et le nombre total de chacune).

Comme la commande build elle-même, edk2-build.sh prend en charge la spécification de plusieurs cibles sur une seule ligne de commande, mais il permet également de spécifier plusieurs plates-formes (ou toutes pour créer toutes les plates-formes connues). Ainsi, afin de construire toutes les plates-formes décrites par le fichier de configuration, pour les cibles DEBUG et RELEASE : `./uefi-tools/edk2-build.sh -b DEBUG -b RELEASE`

Formater une carte uSD en FAT16 ou FAT32

```
mkfs.vfat /dev/sde1
```

Monter la partition

```
mount /dev/sde1 /mnt
```

Créer les sous répertoires nécessaires:

```
mkdir -pv /mnt/{overlays,firmware/brcm}
```

Copier le firmware `RPI_EFI.fd` généré sur la partition

```
cp $WORKSPACE/Build/RPi4/DEBUG_GCC5/FV/RPI_EFI.fd /mnt
```

Télécharger et copier les fichiers suivants à partir de https://github.com/raspberrypi/firmware/tree/master/boot:

- bcm2711-rpi-4-b.dtb
- bcm2711-rpi-400.dtb
- bcm2711-rpi-cm4.dtb
- fixup4.dat
- start4.elf
- overlays/miniuart-bt.dbto ou overlays/disable-bt.dtbo (facultatif)

- overlays/upstream-pi4.dtbo

```
for _unit in bcm2711-rpi-4-b.dtb bcm2711-rpi-400.dtb bcm2711-rpi-cm4.dtb
fixup4.dat start4.elf overlays/miniuart-bt.dtbo overlays/upstream-pi4.dtbo;
do sudo wget -O /mnt/$_unit
https://github.com/raspberrypi/firmware/raw/master/boot/$_unit; done
```

Créer un fichier `config.txt` avec le contenu suivant :

```
cat > /mnt/config.txt<<'EOF'
arm_64bit=1
enable_uart=1
uart_2ndstage=1
enable_gic=1
armstub=RPI_EFI.fd
disable_commandline_tags=1
disable_overscan=1
device_tree_address=0x1f0000
device_tree_end=0x200000
dtoverlay=miniuart-bt
dtoverlay=upstream-pi4
EOF
```

Si on souhaite utiliser **PL011** au lieu du **miniUART**, on peut ajouter les lignes :

`dtoverlay=miniuart-bt`

cela nécessite que `miniuart-bt.dbto` ait été copié dans un répertoire overlays/ sur la carte uSD. Alternativement, on peut utiliser **disable-bt** au lieu de **miniuart-bt** si on n'a pas besoin de Bluetooth.

L'arbre du système de fichier doit ressembler à ceci:

```
cd /mnt
tree
.
├── RPI_EFI.fd
├── bcm2711-rpi-4-b.dtb
├── bcm2711-rpi-400.dtb
├── bcm2711-rpi-cm4.dtb
├── config.txt
├── fixup4.dat
├── overlays
│   ├── miniuart-bt.dtbo
│   └── upstream-pi4.dtbo
```

```
└── start4.elf
```

L'image ainsi constituée fait approximativement 4Mo:

```
cd /mnt
du -alh
2.0M ./RPI_EFI.fd
4.0K ./overlays/miniuart-bt.dtbo
4.0K ./overlays/upstream-pi4.dtbo
12K ./overlays
4.0K ./config.txt
52K ./bcm2711-rpi-4-b.dtb
52K ./bcm2711-rpi-400.dtb
52K ./bcm2711-rpi-cm4.dtb
8.0K ./fixup4.dat
2.2M ./start4.elf
4.3M .
```

Une version précompilé du firmware **EDK2 Raspberry Pi 4 UEFI** est disponible sur le site https://github.com/pftf/RPi4:

```
wget -qO-
https://github.com/pftf/RPi4/releases/download/v1.34/RPi4_UEFI_Firm
ware_v1.34.zip| bsdtar -xvf- -C /tmp
```

Démonter le carte SD

```
cd $WORKSPACE
umount /mnt
```

Débrancher et mettre de côté le client de démarrage Raspberry Pi PXE pour le moment. On va effectuer la configuration du serveur. C'est aussi le bon moment pour retirer la carte SD. Il n'est plus nécessaire maintenant que le Pi démarre en réseau.

Pour démarrer dans le shell UEFI, il faut appuyer sur F1 pendant le processus de démarrage.

Une fois dans le shell on peut modifier les options de démarrage avec **setvar**, par exemple pour désactiver la limitation de la RAM à 3 GB:

```
setvar RamLimitTo3GB -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs
-rt -nv =0x00000000
```

Phase 2 - Configuration du serveur PXE

Lorsqu'on a terminé la configuration du client, on peut utiliser la même carte SD pour le serveur ou en utiliser une seconde. Par exemple, utiliser deux cartes micro SD différentes au cas où on aurait besoin de démarrer le client à partir de la micro SD à des fins de débogage.

Installer Raspbian et les outils nécessaires

Installer Raspbian sur la deuxième carte en répétant les instructions plus tôt dans le didacticiel. Ensuite, démarrer le serveur à partir de la carte SD. Certaines des étapes de configuration initiale du serveur seront familières. Démarrer le serveur connecté à une connexion Internet. On a besoin de la connexion Internet pour mettre à jour et installer les packages. Plus tard dans cette phase, on le retirera d'Internet et le branchera directement sur l'autre Raspberry Pi.

- Mettre à jour Raspbian et installer **rpi-eeprom**, **rsync** et **dnsmasq**
- Mettre à jour le système d'exploitation Raspbian via apt-get et installer le programme **rpi-config**. Cette étape peut prendre un certain temps. Le temps variera en fonction de la vitesse de le connexion Internet.

```
sudo apt-get update
sudo apt-get full-upgrade
sudo apt-get install rpi-eeprom
```

Installer **rsync** et **dnsmasq**. On utilisera rsync pour faire une copie du système d'exploitation de base et dnsmasq comme serveur **DHCP** et **TFTP**. **NFS** sera utilisé pour exposer le système de fichiers racine au client.

```
sudo apt-get install rsync dnsmasq nfs-kernel-server
```

Afin de pouvoir démarrer en UEFI sur le rPI, mettre à jour le fichier `/etc/dnsmasq.conf` pour inclure les éléments suivants (parexemple pour démarrer l'image netboot de Ubuntu 20.04 AARCH64):

```
dhcp-match=set:aarch64,60,PXEClient:Arch:00011:UNDI:003000
dhcp-boot=tag:aarch64,ubuntu20.04/boot/bootnetaa64.efi
```

Configurer NFS et tftp

Créer les répertoires de démarrage NFS, tftp et créer le système de fichiers de démarrage réseau de base

Créer les répertoires NFS et tftpboot. Le répertoire `/nfs/client1` sera la racine du système de fichiers de le client Raspberry Pi. Si on ajoute plus de Pis, il faudra ajouter plus de répertoires clients.

Le répertoire /tftpboot sera utilisé par tous vos Pis de démarrage réseau. Il contient le chargeur de démarrage et les fichiers nécessaires pour démarrer le système.

```
sudo mkdir -p /nfs/client1
sudo mkdir -p /tftpboot
sudo chmod 777 /tftpboot
```

Copier le système de fichiers du système d'exploitation du Pi dans le répertoire `/nfs/client1`. On va exclure certains fichiers du rsync. Il s'agit d'une mesure préventive au cas où on exécuterait à nouveau cette commande après avoir configuré le réseau et dnsmasq. Cette commande prend un certain temps en raison des caractéristiques d'E/S des cartes SD. Ils sont lents

```
sudo rsync -xa --progress --exclude /nfs/client1 \
    --exclude /etc/systemd/network/10-eth0.netdev \
    --exclude /etc/systemd/network/11-eth0.network \
    --exclude /etc/dnsmasq.conf \
    / /nfs/client1
```

Maintenant, utiliser chroot pour changer la racine dans ce répertoire. Mais avant de chrooter, il faut lier le montage des systèmes de fichiers virtuels requis dans le répertoire client de base.

Une fois dans le chroot, supprimer les clés SSH du serveur. Ensuite, reconfigurer le package du serveur openssh qui régénérera les clés. De plus, activer le serveur ssh afin qu'on puisse se connecter à distance lorsque le client se connecte.

```
cd /nfs/client1
sudo mount --bind /dev dev
sudo mount --bind /sys sys
sudo mount --bind /proc proc
sudo chroot . rm /etc/ssh/ssh_host_*
sudo chroot . dpkg-reconfigure openssh-server
sudo chroot . systemctl enable ssh
sudo umount dev sys proc
```

Configurer le serveur PXE pour utiliser une IP statique

Le serveur PXE est un serveur DHCP. Cela signifie qu'il attribue des adresses IP et une configuration réseau aux clients qui les demandent. Dans ce cas, le client de démarrage Raspberry Pi PXE. Si on ne veut pas que le serveur de démarrage PXE lui-même exécute le client DHCP. Par conséquent, il faut désactiver le client DHCP. Créer un nouveau fichier systemd pour désactiver le client DHCP sur eth0. Le chemin du fichier qu'on souhaite créer est `/etc/systemd/network/10-eth0.netdev`. Son contenu doit être :

```
[Match]
Name=eth0
[Network]
```

```
DHCP=no
```

Créer le fichier `/etc/systemd/network/11-eth0.network` avec le contenu suivant. 192.168.2.1 est le serveur DNS et adresse de passerelle. Dans ce tutoriel, on n'a pas de passerelle ou de serveur DNS à cette adresse. De plus, aucun n'est nécessaire pour ce tutoriel. On les a là comme espace réservé, donc si on veut connecter ce système, je peut déposer un routeur sur le réseau à cette adresse. On peut probablement laisser DNS et Gateway de côté.

```
[Match]
Name=eth0

[Network]
Address=192.168.2.100/24
DNS=192.168.2.1
Gateway=192.168.2.1
```

Désactiver le service client DHCP dhcpcd qui est activé par défaut sur raspbian (il s'agit de « dhcpcd » et non de « dhcpd », le premier est un client DHCP, le second un serveur).

```
sudo systemctl stop dhcpcd
sudo systemctl disable dhcpcd
```

Configurer dnsmasq pour le démarrage PXE

Cette étape configure dnsmasq pour prendre en charge le démarrage PXE.

La première chose qu'on doit faire est de désactiver le service DNS embarqué dans dnsmasq : on n'a besoin que des fonctionnalités DHCP et tftp proposées par l'application. Pour atteindre cela, il faut iniquer l'option port : elle est utilisée pour déterminer quel port doit être utilisé pour DNS ; définir sa valeur sur 0 désactive le service. On peut ajouter l'instruction à la fin du fichier de configuration.

```
port=0
```

La deuxième chose qu'on doit faire est de spécifier l'interface réseau qui sera utilisée pour écouter les requêtes DHCP. Dans le cas, ladite interface est eth0, on écrit donc :

```
interface=eth0
```

Si on ne veut pas utiliser une interface spécifique, on peut spécifier une adresse IP, en utilisant l'option listen-address à la place.

Spécification de la plage IP/du mode proxy, cette étape de configuration est très importante et change en fonction de la configuration réseau.

Si le service DHCP fourni par dnsmasq est le seul du réseau, dans cette étape il faut simplement configurer la plage d'adresses IP qui sera attribuée aux clients, et éventuellement une durée de bail

par exemple :

```
dhcp-range=192.168.0.100,192.168.0.200,12h
```

Dans la ligne ci-dessus, la plage d'adresses IP disponibles est définie en séparant les limites inférieure et supérieure par une virgule. Dans ce cas, on a défini une plage qui va de 192.168.0.100 à 192.168.200 ; on a également fixé une durée de location de 12h.

Le deuxième cas est probablement le plus courant dans une configuration standard/domestique, où le service DHCP est généralement fourni par un routeur. Si tel est le cas, dnsmasq doit être configuré pour s'exécuter en mode proxy afin d'éviter les conflits. Dans ces cas, on peut écrire :

```
dhcp-range=192.168.0.0, proxy
```

On a saisi deux éléments séparés par une virgule : le premier est l'adresse du sous-réseau (192.168.0.0), le second est le mot clé « proxy ».

Le chargeur de démarrage **pxelinux** est capable de fonctionner à la fois en mode EFI et en mode BIOS, on doit donc trouver un moyen de servir le fichier approprié en fonction du mode utilisé par le client.

DHCP utilise une série d'options pour l'échange d'informations : l'option 93 (client-arch) est utilisée pour transmettre des informations sur l'architecture du client. Le tableau ci-dessous affiche les valeurs numériques et de chaîne des options et les architectures auxquelles elles font référence :

Valeur d'option	**Valeur de chaîne**	**Architecture**
0	x86PC	Intel x86PC
1	PC98	NCA/PC98
2	IA64_EFI	EFI Itanium
3	Alpha	DEC Alpha
4	Arc_x86	Arc x86
5	Intel_Lean_Client	Client Intel Lean
6	IA32_EFI	EFI IA32
7	BC_EFI	EFI BC
8	Xscale_EFI	EFI Xscale
9	X86-64_EFI	EFI x86-64

Pour spécifier quel fichier doit être fourni pour le mode approprié utilisé par le client, on peut utiliser l'option pxe-service.

Par exemple pour x86PC, entrer la ligne suivante :

```
pxe-service=0,"PXELINUX (BIOS)",bios/pxelinux
```

On a fourni trois valeurs séparées par une virgule à l'option : la première est le type de système client (x86PC), la seconde est le texte du menu et la troisième est le fichier qui sera téléchargé par le client pour effectuer le démarrage. Le chemin du fichier est relatif à la racine tftp. Dans ce cas, il se trouve dans le répertoire bios qu'on doit créer auparavant et s'appelle pxelinux.0 : le nom doit être signalé

sans l'extension .0, comme on peut le voir ci-dessus.

Pour le mode EFI x86-64, à la place, indiquer :

```
pxe-service=x86-64_EFI,"PXELINUX (EFI)",efi64/syslinux.efi
```

Le contenu de `/etc/dnsmasq.conf` doit ressembler à ceci:

```
interface=eth0
no-hosts
dhcp-range=192.168.0.100,192.168.0.200,12h
log-dhcp
enable-tftp
tftp-root=/tftpboot
pxe-service=0,"Raspberry Pi Boot"
```

Ensuite, copier les fichiers de démarrage du répertoire `/boot` dans le répertoire `tftpboot`.

```
sudo cp -r /boot/* /tftpboot/
```

Activer **systemd-networkd** et **dnsmasq**. Redémarrer **dnsmasq** pour confirmer que la configuration est valide. Enfin, redémarrer et vérifier que le Pi propose un réseau correctement configuré.

```
sudo systemctl enable systemd-networkd
sudo systemctl enable dnsmasq.service
sudo systemctl restart dnsmasq.service
sudo reboot
```

Il faut maintenant mettre à jour le fichier cmdline.txt dans `/tftpboot`. Ce fichier contient les paramètres du noyau qui sont transmis au client Raspberry Pi au moment du démarrage. Editer `/tftpboot/cmdline.txt` le remplacer par :

```
console=serial0,115200 console=tty1 root=/dev/nfs
nfsroot=192.168.2.100:/nfs/client1,vers=3 rw ip=dhcp rootwait
elevator=deadline
```

Configurer les exports NFS sur le serveur de démarrage PXE

Cette étape configure les exports. Les exports sont des systèmes de fichiers qui sont partagés ou exportés via NFS. Pour ce faire, il faut configurer le service `/etc/exports` et redémarrer les services liés à NFS.

Le contenu de `/etc/exports` doit être le suivant.

```
/nfs/client1 *(rw,sync,no_subtree_check,no_root_squash)
```

```
/tftpboot *(rw,sync,no_subtree_check,no_root_squash)
```

Configurer le fichier /etc/fstab pour monter via NFS

On a presque terminé! Une dernière étape pour modifier le fichier `/etc/fstab` dans le système de fichiers du client. Cela indiquera au client de monter son volume racine sur le serveur NFS. Mettre ce qui suit dans `/nfs/client1/etc/fstab`.

```
proc       /proc        proc     defaults    0    0
192.168.2.100:/tftpboot /boot nfs defaults,vers=3 0 0
```

Enfin, activer et redémarrer les services liés à NFS.

```
sudo systemctl enable rpcbind
sudo systemctl restart rpcbind
sudo systemctl enable nfs-kernel-server
sudo systemctl restart nfs-kernel-server
```

Faire maintenant un dernier redémarrage sur le serveur pour faire bonne mesure. Jeter un œil aux journaux système et aux statuts systemctl pour voir si tout a démarré correctement.