Documentation

ClearOS: Remote boot app

February 20, 2017

Contents

List of Figures

1 Requirements

Remote boote APP requirements:

- ClearOS 7
- nfs-utils >= 1.3.0
- syslinux >= 4.05
- dnsmasq >= 2.48

Note: When you install our app it will automatically resolve and download these packages.

2 Contact Support

If you find any bugs in the Remote Boot application, please take the time to send us details about the problem:

- Tells us more about what you are trying to do;
- Send us error and warning messages. Screenshots depicting the problem are very helpful;
- Step-by-step procedure to reproduce the problem;
- What is the expected result of your procedure;
- Attach relevant logs.

Please also contact us to report any problems with our documentation. You may contact us by e-mail (clearos@polilinux.com.br) or open an issue at github.com/polilinux/app-remote-boot.

3 License

ClearOS 7 and Remote boot app code are developed under GNU General Public License version 2 or later versions.

You can find the source code of the Remote boot app in the CentOS git repository or RPMs from the ClearOS package repository.

All related Linux remote boot services use GPL or specific licenses. You may find them in the ClearOS git repository.

4 About Remote Boot

The Remote Boot server enables your workstations to boot from the network and operate in *diskless* mode. This environment supports any x86 hardware with a Preboot eXecution Environment (PXE) capable ethernet interface. It allows you to manage all of your Linux operating systems centrally, making it easier to save all data in a backup server or in the cloud.

Here is a list of some advantages of running a Remote Boot environment:

- Enables efficient use of hardware and software, reducing the overall storage requirements when booting from the same network image multiple times;
- Enables centralized management of the workstations, allowing the sysadmin to:
  - Apply software updates easily – they have to be done only once for multiple workstations;
  - Manage content blocking and user permissions remotely;
  - Monitor the status of the network and workstations;
- Provides a secure and controlled environment – every workstation in the network may be accessed and audited by the administrator;

4.1 Related Technologies

The Remote Boot environment depends on at least the following applications:

- **DHCP Server:** Provides diskless stations with the initial Internet Protocol (IP) address configuration and points them to the right Syslinux server during boot;
- **Syslinux Server:** Provides the PXELINUX implementation, which allows the diskless stations to download bootable images;
- **Trivial File Transfer Protocol (TFTP) Server:** Provides kernel and initial RAM file system (initramfs) images to the TFTP clients (diskless stations);
- **Network File System (NFS) Server:** Provides a network-based filesystem which is used by all diskless stations.

When you boot a diskless station using PXE it will follow these steps:

1. The PXE-enabled ethernet card requests an IP address from the DHCP server;
2. The DHCP server offers an IP address to the client, as well as the location of `pxelinux.0`;
3. The PXE Server downloads the Syslinux boot loader (`pxelinux.0`)
4. Syslinux searches for a configuration matching the PXE client's Universally Unique Identifier (UUID) or IP network/address;
5. If there is a matching configuration file in the Syslinux server, the boot loader will receive information about the images it should download;
6. Syslinux downloads the initramfs through TFTP and boots it;
7. The initramfs loads the required network and NFS drivers, mounts the remote file systems and switches over control to the definitive Linux kernel.

Therefore, you will have to install and configure those services to get a basic remote environment working.

4.2 TFTP

TFTP *(Trivial File Transfer Protocol)* is used to transfer small files through the network. TFTP is based on UDP/IP and implements its own packet integrity check. TFTP is used in the remote boot environment to transfer the Kernel to diskless stations. After a Kernel image is downloaded a minimal system boots and tries to mount and use a Network File System (NFS) as root filesystem.

4.3 NFS

NFS is a file system stack that provide access to any file system through the network. It is used to mount the filesystems required by the *diskless* stations.

NFS is opensource and free. Licensed by General Public License version 2 (GPLv2).

4.4 syslinux

Light bootloader that implements solution for SYSLINUX, PXELINUX, ISOLINUX and EXTLINUX. It will be used to provide kernel and initrd images to diskless stations via TFTP.

Syslinux bootloader is open-source and free to use (licensed by GPL v2).

5 Remote Boot ClearOS App

ClearOS is a stable Linux server distro based on CentOS. It provides an easy to use web interface, as well as an online Marketplace with "a turn-key app installation engine", which allows the sysadmin to quickly install and configure new services in the server.

The Remote Boot app was built on top of the ClearOS framework, and it was designed to simplify the process of managing diskless workstations that boot through the network. The Remote Boot app offers a simple interface to manage:

- the NFS service and folder exports
- a TFTP server with dnsmasq
- PXE configuration files

5.1 Installing the App

To get the Remote Boot App you have to access ClearOS Marketplace from your ClearOS Dashboard and search for "remote boot". Click on "Learn more" and then on "Download and install".

5.2 Web interface

The Remote Boot app web interface depicted in Figure 5.2 offers a variety of global and per-profile options to manage computers booting through the network.

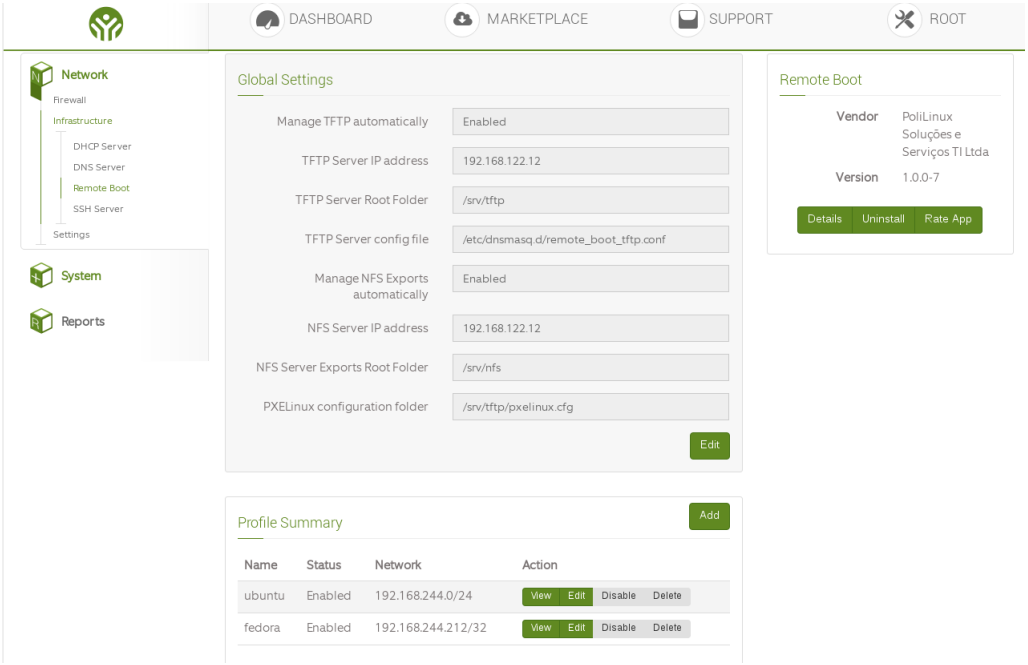

Figure 1: Main Remote Boot app web interface

Section ?? explains in more details how to create, edit and remove Remote Boot profiles using the web interface.

5.3 Uninstalling the App

To remove our app you have to follow these steps:

1. Open you ClearOS Dashboard;
2. Through menu go to Network > Infrastructure > Remote Boot;
3. Click on "Uninstall" button and then click on "Confirm Uninstall".

Note: Package dependencies downloaded during the installation (see section 1) will not be removed!

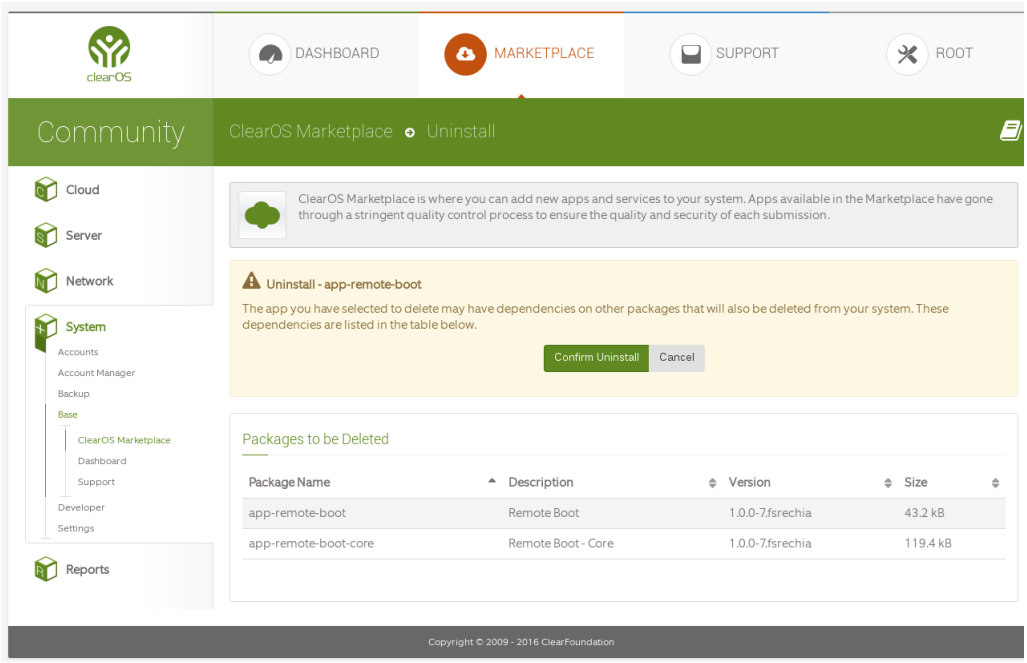

Figure 2: Removing Remote Boot App

5.4 Application Structure

Through the Remote Boot web interface you can create, edit and remove profiles to allow machines to boot using the network. Each profile is associated to a set of NFS server export entries, a profile name and a client network.

The Remote Boot app is divided into three sections:

- the Remote Boot Main page, which contains:
  - the Global Settings view,
  - and the Profile Summary View, which allows the user to manage (enable/disable, delete, and links to the profile settings form).
- the Global Settings form to allow changes to the global configuration;
- the Profile Settings form to add and edit Remote Boot profiles.

5.4.1 Global Services

In the global settings form the user configures the TFTP server and the NFS Server IP address. By default the TFTP and NFS services come disabled, but the user may enable them easily by editing the global settings:

- The TFTP service relies on the same software as the DHCP Server: `dnsmasq`. If TFTP management is enabled in the web interface, the Remote Boot app creates a TFTP configuration file under `/etc/dnsmasq.d/` and restarts the `dnsmasq` if required.
- The NFS service uses the `nfs-utils` package. If NFS service management is enabled in the web interface, the Remote Boot app enables and starts the `nfs-server` init script through `systemctl`.
- PXELinux service is provided by copying the `pxelinux.0` bootloader file into the TFTP root folder specified in the Global Settings form.

5.4.2 Profile Services

The configuration management to boot different operating systems in your network is realized through the use of profiles. Based on a few settings filled out by the system administrator, the application:

1. Automatically creates a folder structure to store the network-bootable kernel and system files.
2. Creates a PXELinux configuration file to instruct network workstations where to find the kernel.
3. Suggests Kernel parameters to be supplied to the network workstations so that they can mount and use the network-bootable system files (Linux root filesystem).

5.4.3 Application Files

Here is a list files needed by this application.

- `/etc/exports.d/remote_boot_*.exports`: NFS entries to export and offer Linux filesystems through the network;
- `/etc/dnsmasq.d/remote_boot_tftp.conf`: Configuration file to enable TFTP service using `dnsmasq`;
- `<tftp root>/pxelinux.cfg/*`: Syslinux entries to provide initial kernel images through the network;
- `/etc/clearos/remote_boot.conf`: XML configuration file that stores all global and profile settings used by the application;
- `/var/clearos/base/daemon/nfs-server.php`: PHP configlet required for ClearOS framework to properly interact with the nfs-server systemctl script;

5.5 Remote Boot Management

This section gives detailed descriptions about the configuration options for the Remote Boot application.

5.5.1 Global Settings

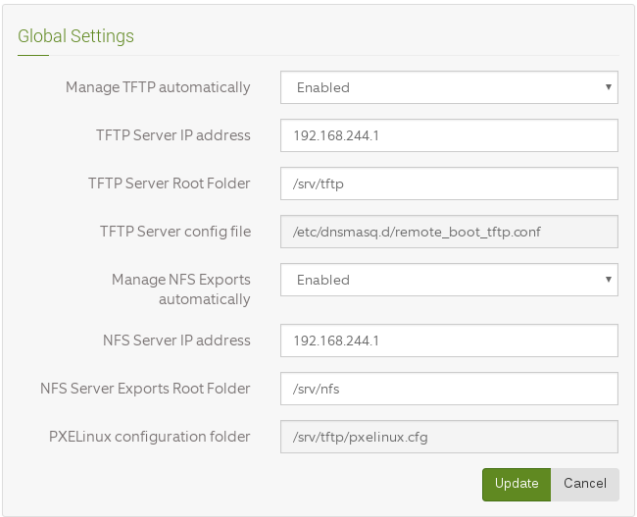

Figure 3: Edit Global Settings Form

Figure 3 shows the global settings form. The fields in this form are:

- **Manage TFTP service**: If enabled, the Remote Boot app will manage a local dnsmasq-based TFTP service.
- **TFTP Server IP address**: The IP address of the TFTP server from where the network workstations will download initrd.img and compressed and bootable Linux kernel file (vmlinuz) images.
- **TFTP Server Root Folder**: This is the root folder of the TFTP server. It is always created even when the Remote Boot app is not managing the TFTP service.
- **TFTP Server config file**: This file is only created if TFTP automatic management is enabled. This is a read-only field.
- **Manage NFS service**: If enabled, the Remote Boot app will enable and start a local NFS service.

- **NFS Server IP address**: Fill out with the IP address of the NFS server which exports the NFS folders your clients will access (e.g. rootfs, var, etc.).
- **NFS Server Exports Root Folder**: This is the root folder of the NFS server. It is always created even if the Remote Boot app is not managing a local NFS service.
- **PXELinux configuration folder**: This is the root folder of all pxelinux configuration files. This is a read-only field, and configuration files are created here on a per-profile basis.

5.5.2 Profile Settings

Each Remote Boot profile has the following main settings:

- **Profile name**: used to create a folder structure within the TFTP and NFS servers. Each profile must have a unique name. Changing this parameter affects the following fields:
  - All NFS exported folders.
  - The NFS root path in the kernel parameters.
  - The kernel folder, consequently changing the path to initrd.img and vmlinuz.
- **NFS Server IP**: used to create appropriate kernel parameters pointing to the configured NFS server. The default value is taken from the global settings. Changing this parameter affects the following field:
  - The kernel parameters field, changing the IP address of the NFS root.
- **Client Network**: used to determine the PXELinux configuration file name. Hosts booting through the network will look for this configuration file based on their acquired IP addresses. Each profile must have a unique client network, and changing this parameter automatically recalculates the PXELinux config file field.

5.6 Getting Started

Suppose that you have the network 192.168.244.0/24 and you would like to provide Remote Boot options to network clients. Assuming that all services – DHCP, NFS, TFTP, and PXELinux bootloader – must may be configured in the same ClearOS server at 192.168.244.1, the following sections explain how the system administrator should proceed.

5.6.1 Pre-requisite: DHCP Server

The Remote Boot app requires the system administrator to configure the DHCP server offered by the ClearOS Marketplace.

We assume that the DHCP server is already configured in ClearOS, as shown in Figure 4.

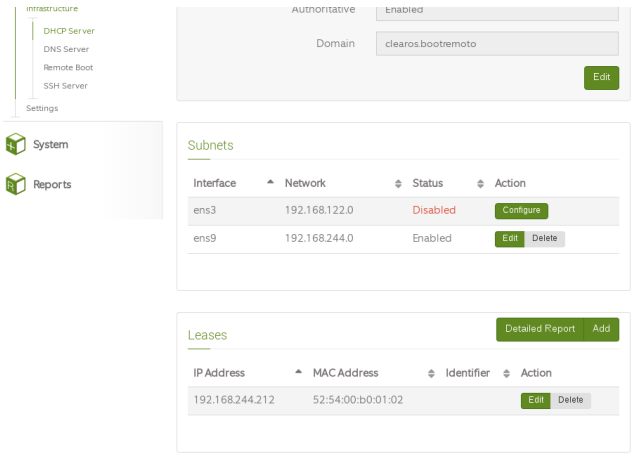

Figure 4: DHCP Server Settings Example

5.6.2 Configuring Global Settings

The system administrator configures the appropriate TFTP and NFS server addresses and enables them in the global settings, as depicted in Figure 5.

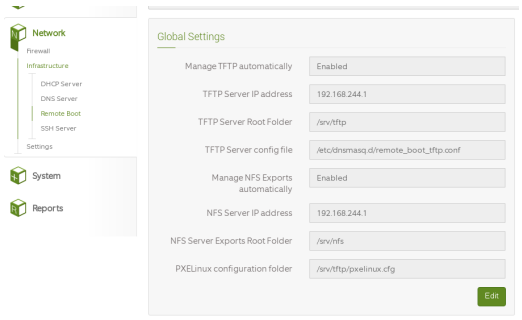

Figure 5: Remote Boot Global Settings Example

5.6.3 Adding a Profile

The next step is to create a Remote Boot profile as depicted in Figures 6 and 7. Note that here only the first Export folder is enabled to be exported – `/srv/nfs/ubuntu/rootfs`, which is created based on the profile name.

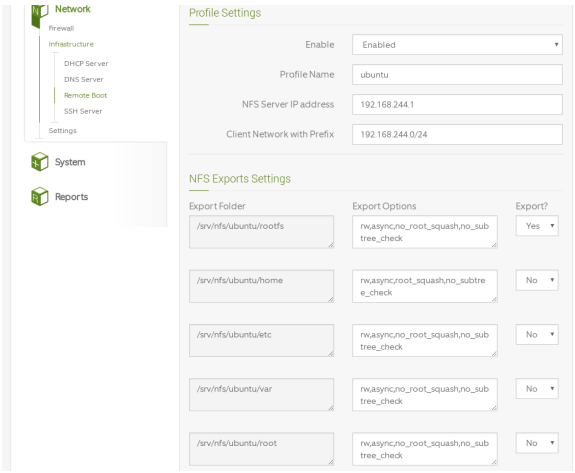

Figure 6: Example Remote Boot Profile for Ubuntu - part 1

Also note that the PXELinux configuration file `/srv/tftp/pxelinux.cfg/C0A8F4` is automatically created based on user input, where `C0A8F4` is the hexadecimal representation of the `192.168.244.0` network identifier. This means that any machines that boot and acquire an IP address via PXE in the 192.168.244.0/24 network range would match this profile we just created, and therefore attempt load the configured `initrd.img`, `vmlinuz` and `rootfs`.

5.6.4 Copy the OS files to the Server

The next steps for the system administrator would be to provide these files and filesystem for the Remote Boot to work:

1. Place a root file system under `/srv/nfs/ubuntu/rootfs`.
2. Copy or symlink a vmlinuz file (an executable Kernel image) to `/srv/tftp/ubuntu/kernel/vmlinuz`.
3. Copy or symlink an initrd.img file (an initramfs image) to `/srv/tftp/ubuntu/kernel/initrd.img`.

Please take a look at section 6 to learn more about Linux Root Filesystems.

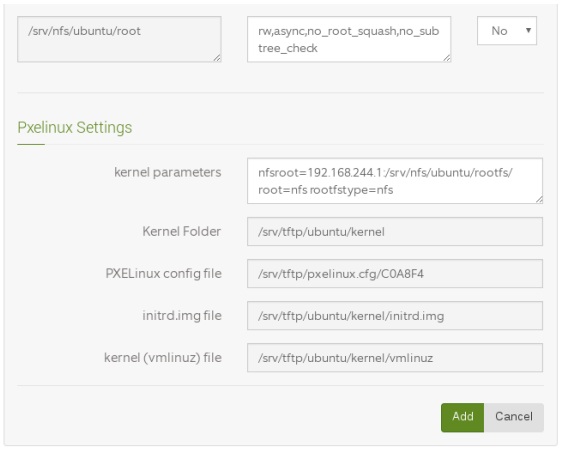

Figure 7: Example Remote Boot Profile for Ubuntu - part 2

6 The Linux Root Filesystem and Remote Boot

The Remote Boot application expects the system administrator to provide the OS to be booted across the network. The application was created with the purpose of enabling boot of remote Linux root filesystems (root-fs), containing all the required files to run a Linux system. The use of remote filesystems requires some additional considerations when compared to regular local filesystems.

A regular root-fs typically stores a vmlinuz and an initial ramdisk (initrd) under a directory called `/boot/`, whereas in a network-bootable system these files must be retrieved from the network (usually from a TFTP server) and executed before mounting the actual remote root-fs.

Another aspect to consider is that the root-fs has machine-specific configuration files in `/etc`, user files in `/home`, and a `/var` directory with files that are frequently changed during operation of the system – these directories should be mounted with read-write permissions in any machine. Other directories, such as `/bin`, `/sbin`, `/lib`, and `/usr` are not changed frequently, but may be mounted with write permissions in a local root-fs.

A remote root-fs may be shared by several computers, therefore it is safer to mount the root (`/`) root-fs as read-only, while keeping machine-specific and frequently changed directories (`/etc`, `/home`, and `/var`) exclusive to each machine.

6.1 Configuration requirements

In order to successfully boot root-fs across the network there are a few requirements that must be met:

- Both vmlinuz and initrd must include NFS and networking modules;
- The root-fs must contain all the minimum files for OS operation;
- Auth system (local or ldap) defined and configured in root-fs;
- NFS permissions (read-write for at least `/var` and `/home` directories);
- Configure initrd to isolate `/var` and `/home` directories if you plan to share the same root-fs among several machines;
- Clean or correct `/etc/fstab` (e.g. you may set `tmpfs`, `devpts`, `sysfs` and `proc` filesystems manually there);

6.2 Creating a Root Filesystem

There are two basic methods to create a root-fs:

1. Install Linux in a virtual or physical machine and copy all system files;
2. Use distro-specific tools.

The first option typically consists of booting a machine using any Linux Live USB stick and copying the root file system with `rsync` or similar tool. We recommend using the following command line options for `rsync`:

```
# sudo rsync -aHx --numeric-ids --progress origin destination
```

Please read `man rsync` for more details about these command line options.
The next sections explain a bit more about creating root filesystems with distro-specific tools.

6.2.1 Creating Debian unstable rootfs

Using Debian or Ubuntu installation execute:

```
# Run as root!
mkdir /chroot-debian-sid
debootstrap sid /chroot-debian-sid http://ftp.br.debian.org/debian
```

6.2.2 Creating Fedora last stable chroot

Using Fedora or Redhat installation run the following commands to create a root file system:

```
# Run as root!
export FEDORA_ROOTFS=/home/fedorarootfs
mkdir -pv $FEDORA_ROOTFS/var/lib/rpm
rpm --root $FEDORA_ROOTFS --initdb
dnf download --destdir=/tmp fedora-release
# rpm --root  $FEDORA_ROOTFS -ivh /tmp/fedora-release*rpm
rpm --root  $FEDORA_ROOTFS -ivh --nodeps /tmp/fedora-release*rpm
dnf --installroot=$FEDORA_ROOTFS install bash
dnf --installroot=$FEDORA_ROOTFS groupinstall "minimal install"
```

And use `dracut` to generate the initramfs image:

```
# This command creates an initramfs image based on your current Fedora kernel
dracut --kver `uname -r` --no-hostonly                    \
       --modules "bash base shutdown network ifcfg nfs"   \
       initrd.img-`uname -r`
```

6.2.3 Creating Ubuntu chroot:

Note: all steps in this section were tested in a Ubuntu 16.04 environment.
Create the rootfs Using an Ubuntu system. First, use `debootstrap` to create a minimal rootfs:

```
# Run as root!
UBUNTU_ROOTFS=/root/ubuntu
apt-get update
apt-get install debootstrap
mkdir -vp $UBUNTU_ROOTFS
debootstrap --variant=buildd --arch amd64 xenial  \
           $UBUNTU_ROOTFS                         \
           http://archive.ubuntu.com/ubuntu/
```

In Ubuntu 16.04 edit the `/etc/initramfs-tools/initramfs.conf` file and make sure that you have the following options set as below:

```
MODULES=netboot
BOOT=nfs
```

Create the initramfs image and store it in the root home directory:

```
# mkinitramfs -o /root/initrd.img-`uname -r`
```

Copy the generated initramfs and compressed kernel image (`vmlinuz`) to your Remote Boot server:

```
# scp /root/initrd.img-`uname -r`                                    \
      /boot/vmlinuz-`uname -r`                                       \
      root@<remote boot server>:/<tftp root>/<profile name>/kernel/
```

6.3 Customizing a Root Filesystem

After creating a root-fs environment to be used across the network, it is advisable to configure it before booting it: set up a password for root, create users, etc.

There are several ways to customize your root-fs. Here we briefly describe a couple of ways to do that:

1. The easiest way is to install any Linux distro in a virtual machine and do all the customization you need. Then boot a Live Linux USB stick and copy that root-fs to your ClearOS server. Using a Live Linux distro for this purpose ensures that the root-fs is not modified during the copy.

2. If you wish to use your own Linux system to customize your root filesystem, you may simply use `chroot` to change your root filesystem to the one you just created.

The script below helps with the process of changing the root filesystem for configuration/customization purposes:

```
#!/bin/bash

if [ -z ${1+x} ];
  then
    echo "Please provide a folder to chroot to..."
    exit 1
  else
    echo "Chrooting to $1"
fi

CHROOT=$1

mount -t proc proc $CHROOT/proc/
mount -t sysfs sys $CHROOT/sys/
mount -o bind /dev $CHROOT/dev/
mount -o bind /dev $CHROOT/dev/pts

chroot $CHROOT /bin/env -i  \
    HOME=/root TERM="$TERM" PS1="[\u@$CHROOT \W]\$ "  \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin:/bin  \
    /bin/bash --login

echo "Exiting $CHROOT environment"
umount $CHROOT/dev/pts
umount $CHROOT/dev/
umount $CHROOT/sys/
umount $CHROOT/proc/

echo "Cleaned up"
```

To use it simply copy the contents above to a shell script called `chroot_to.sh` an execute it:

```
# chmod +x chroot_to.sh

# ./chroot_to.sh <folder containing root filesystem>
```

6.4 Adding a Root Filesystem

When you have your root-fs ready to use, you have to copy that the `rootfs` folder configured with Remote Boot app, as depicted in Figure 8.

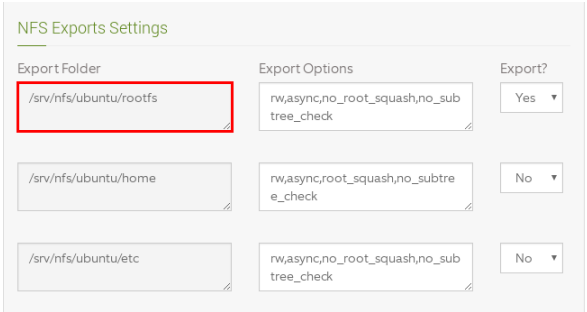

Figure 8: Root-fs Folder Location

Please be careful to keep file ownership and permissions intact when copying. We recommend using either a tarball or `rsync` to transfer the root-fs to the ClearOS server. Considering the example from Figure 8, make sure that you end up with a folder structure such as this:

```
/srv/
  nfs/
```

```
ubuntu/
  rootfs/
    bin/
    boot/
    dev/
    etc/
    home/
    lib/
    mnt/
    opt/
    proc/
    root/
    run/
    sbin/
    sys/
    tmp/
    usr/
    var/
```

Note that the directory structure above is supposed to be used by a single system, as it already contains directories such as `/var` and `/home`.

6.5 Sharing Root Filesystems

As mentioned in Section 6, you may share parts of a root-fs among several diskless clients. The benefits of this are:

1. Configure and maintain a single image for many diskless stations;
2. Saves total disk space used;
3. Saves disk and disk operation power costs in the diskless stations;
4. Each user can still have their customized home directory;
5. It is easier to enforce a single configuration profile for many diskless clients.

However, there are a few drawbacks when sharing file systems in a network. There are logs, file descriptors and lock files which have the same name across multiple diskless clients. If the same root-fs is used for more than one machine, then some directories need to be created and mounted exclusively for each machine.

Each diskless client must have exclusive access to at least the following two mount points:

1. `/var`
2. `/home/your-user`

A second possible problem is customization of diskless clients. Services like Xorg, CUPS and NetworkManager use config files from `/etc` directory. So if you need to maintain different configuration files for different diskless stations, then you'll have to keep a separate copy of `/etc` for each case.

Our app design is built to help you solve these problems. When creating a new client network profile, the Remote Boot app will also create `/home`, `/var`, `/root`, and `/etc` directories.

For example, consider the profile created in Section 5.6. The default directories were:

- /srv/nfs/ubuntu/rootfs
- /srv/nfs/ubuntu/home
- /srv/nfs/ubuntu/etc
- /srv/nfs/ubuntu/var
- /srv/nfs/ubuntu/root

If that root-fs needs to be shared, we recommend the creation of client specific directories within that directory structure. One possible way of doing this is by using IP addresses as directory names under the shared directories:

1. Diskless client `192.168.244.20`:
   `/srv/nfs/ubuntu/var/192.168.244.20/`
   `/srv/nfs/ubuntu/etc/192.168.244.20/`
   `/srv/nfs/ubuntu/home/192.168.244.20/`

2. Diskless client 192.168.244.30:
   `/srv/nfs/ubuntu/var/192.168.244.30/`
   `/srv/nfs/ubuntu/etc/192.168.244.30/`
   `/srv/nfs/ubuntu/home/192.168.244.30/`

The structure above solves the problem by providing exclusive `/home`, `/etc` and `/var` directories for each client, and still allows sharing of the same root-fs. Another advantage of using this structure is that you may now export your root-fs as read-only to protect it. This is depicted in Figure 9.

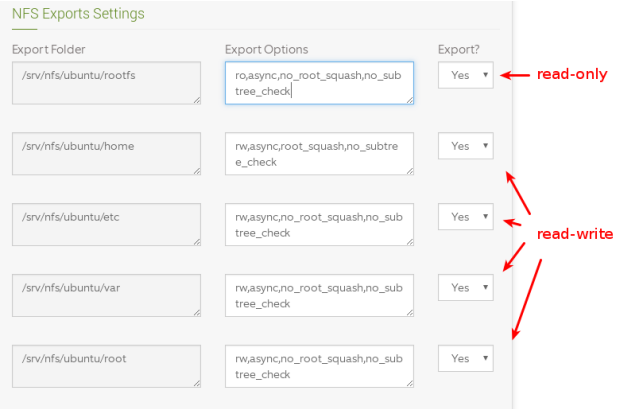

Figure 9: Exporting Shared Root-fs

One way of informing the diskless clients about their exclusive mount points is to have initrd or dracut scripts telling them what to do. You may also use custom kernel parameters to carry information to the scripts during diskless client boot time. For instance, in Figure 10 we define the `$profileroot` helper kernel parameter.

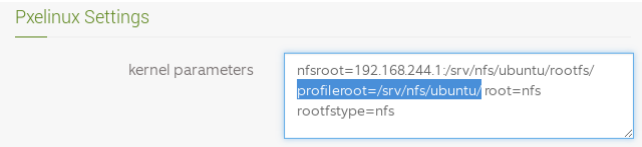

Figure 10: Profile Root Helper Kernel Parameter

And then create an initrd/dracut script that uses the helper parameter to find the profile root dir, the NFS Server IP, the NFS Client IP and mount exclusive client directories before final root-fs change:

```
#!/bin/bash

remoteboot_abort()
{
 die "Remoteboot hook: $1"
}

# get NFS server IP during boot (initramfs ubuntu format is nfsroot=<IP>:<folder>)
nfsserverip=$(echo ${nfsroot%%:/*})
if [ -z "$nfsserverip" ]
then
  # if it fails, try dracut format root=nfs:<IP>:<folder>
  nfsserverip=$(echo ${nfsroot#*:})
  nfsserverip=$(echo ${nfsserverip%%:*/})
fi
# extract diskless client IP during boot
nfsclientip=$(cat /tmp/net.*.dhcpopts | grep -m 1 -i new_ip_address | sed "s/.*=//")
```

```
# check if we may proceed:
if [ -z "$nfsserverip" ]
then
  remoteboot_abort "Could not get NFS Server IP address."
fi

if [ -z "$nfsclientip" ]
then
  remoteboot_abort "Could not get Diskless Client IP address."
fi

if [ -z "$profileroot" ]
then
  echo "Please define profileroot as a profile kernel parameter."
  echo " example: profileroot=/srv/nfs/ubuntu"
  remoteboot_abort "profileroot not defined."
fi

# finally mount all exclusive mount points and switch them to the final network root-fs
for mountpoint in /var /etc /root /home
 do
  mkdir -p /remoteboot/$mountpoint
  mount -t nfs $nfsserverip:$profileroot/$nfsclientip/$mountpoint /remoteboot/$mountpoint

  (($?)) && remoteboot_abort "Error while mounting $mountpoint"

  mount --bind /remoteboot/$mountpoint $NEWROOT/$mountpoint
  umount /remoteboot/$mountpoint
done
```

7 Known Issues

Below is a list of known problems that may happen in remote boot environments:

1. **Download of kernel images through TFTP takes too much time or fails**
   Some TFTP server versions may present problems to send files with more than 30MBytes, especially if the network is congested. Try to reduce the initrd image size as much as possible to avoid this (visit this link for more information).

2. **Failure while trying to mount NFS directiores**
   Usually this is a network problem and not NFS problem. Make sure that your network is not blocking ports required by NFS services.

3. **NFS not starting anymore**
   This may happen if the user manually edits exports in /etc/exports. Incorrect export entries in `/etc/exports`, such as entries pointing to non existing directories, may prevent the NFS server from starting.

4. **Operating System acquiring two IP addresses**
   In the past only Media Access Control (MAC) address were used for IP address leases. Nowadays some distros use a combination of MAC address and DHCP client ID. The client ID presented by PXE at boot time may be different from the DHCP client ID of the OS network manager. This could lead to each machine in the network to acquire two IP addresses and potential problems when trying to mount NFS shares that are restricted by originating IP address. The kernel parameter `rd.ip=auto` may help you solve this problem by telling dracut how to behave (see this link for more information). The use of automatic Network Managers is not recommended in the diskless stations.

5. **Initrd boot drops to emergency shell**
   This may be related to a configuration error in `/etc/fstab`. The administrator must be careful to create a proper `/etc/fstab` file, especially if the root-fs was created from a computer which originally had physical disks.

8 Acronyms

PXE Preboot eXecution Environment . . . 6
TFTP Trivial File Transfer Protocol . . . 6
DHCP Dynamic Host Configuration Protocol . . . 1
NFS Network File System . . . 6
GPLv2 General Public License version 2 . . . 7
UUID Universally Unique Identifier . . . 6
MAC Media Access Control . . . 20
IP Internet Protocol . . . 6
initramfs initial RAM file system . . . 6
root-fs Linux root filesystems . . . 14
OS Operating System . . . 1
vmlinuz compressed and bootable Linux kernel file . . . 10
initrd initial ramdisk . . . 14