LTSP Fat Client Setup

LTSP Fat Client Setup

Contents

The basics

```
$ apt-get install ltsp-server
```

Now setup /etc/ltsp/ltsp-server.conf to change the used directories to something under /srv (the default is /opt/ltsp for base and /var/tftpboot for tftp, which are both wrong in my interpretation of the FHS).

- /etc/ltsp/ltsp-server.conf

```
BASE="/srv/ltsp"
TFTP_DIRS="/tmp"
```

TFTP_DIRS is normally a list of directories that are exported on tftp. These directories ($TFTP_DIRS/ltsp/<architecture>) are populated with the contents of /boot from the client image chroot, which essentially contains the kernel and initrd images. In the lab we use iPXE to load the kernel and initrd and we let iPXE load everything from http instead of tftp. Using http, apart from being faster, also allows us to use symlinks from /srv/netboot/http (the http-exported directory that iPXE looks in) directly to the chroot's /boot, therefore eliminating the need for copying the contents of the chroot's /boot to TFTP_DIRS. But to avoid TFTP_DIRS taking the default value of /var/tftpboot, we set it to /tmp, so that ltsp-build-client below will copy the files to /tmp (which acts as a /dev/null-style directory in this case).

Build a client image

Thin vs Fat client

LTSP is by default configured to do thin clients. A thin client is a client that only runs X and all the GUI applications that it shows are running on the server with X forwarding. A fat client is a client that actually runs all its applications on the client's cpu. In labaki we are only interested in fat clients, therefore all the documentation in this page refers to fat clients.

Build a fat client's root filesystem

- /etc/ltsp/ltsp-build-client.conf

```
MIRROR="http://ftp.gr.debian.org/debian"
COMPONENTS="main contrib non-free"
SECURITY_MIRROR="none"
DISTRIBUTION="testing"
```

```
# Some must-have stuff
LATE_PACKAGES="
        less
        nano
        aptitude
        man
        nfs-client
"
```

Now with this config file in place, we can build the client chroot with:

```
$ ltsp-build-client --arch i386
```

Substitute i386 with the architecture that you want the client to have. The chroot will be created in $BASE/<architecture> (in our case, /srv/ltsp/i386).

Convert to fat client

This is a thin client image by default. To convert it to a fat client, you only need to install a desktop environment. The init scripts will automatically setup the image as a fat client when they detect an X session script in /usr/share/xsessions/.

See Debian Standard Desktop Setup for setting up a standard desktop.

Client configuration

- $BASE/<architecture>/etc/lts.conf (/srv/ltsp/i386/etc/lts.conf in our case)

```
[default]
   LTSP_CONFIG=True

   # Use local swap (if available) and network swap
   USE_LOCAL_SWAP=True
   NBD_SWAP=True

   # Disable swap file encryption to workaround nbd swap file bug (see below the section about NBD swap)
   ENCRYPT_SWAP=False

   # Do not use ltsp's display manager - use whatever is in /etc/X11/default-display-manager
   DEFAULT_DISPLAY_MANAGER=""

   # Do not disable getty on tty1-6
   DISABLE_GETTYS=False

   # Uncomment to start a root shell on tty12 for debugging
   #SCREEN_12=shell

   # mount /home and /storage from the server
   FSTAB_0="server:/home /home nfs defaults 0 0"
   FSTAB_1="server:/storage /storage nfs defaults 0 0"

   # Greek keyboard layout support
   XKBLAYOUT="us,gr"
   XKBVARIANT=","
   XKBOPTIONS="grp:alt_shift_toggle"
```

Other lts.conf options that might look nice but are not

- LOCALDEV: This is set to False in a fat client, even if you try to override. On thin clients, it enables sharing local devices with the server using ltspfs, so that GUI applications of the client (which are actually running on the server) can see those devices.
- LOCAL_APPS: This is set to True anyway if sshfs is installed. It doesn't affect the fat client in any way.
- NFS_HOME: The source code says it's deprecated. Use the FSTAB_0 line from the above config file instead, they are equivalent.

Configure LDAP in the client

Required to be able to use NFS properly. See Authentication.

Sound

lts.conf has some sound options too. When running in fat client mode, they do nothing. Since pulseaudio is installed by default (as a dependency of the ltsp-client package), it will start automatically with each user session. See Pulseaudio for details on how to configure it.

If pulseaudio fails to start, saying something about "Invalid argument", you have forgotten to setup /etc/idmapd.conf. See Authentication#Make_NFS_client_work_properly.

Configure the NFS server

- /etc/exports

```
/srv/ltsp       10.176.4.0/26(ro,no_root_squash,async,no_subtree_check)
/home           10.176.4.0/26(rw,sync,no_subtree_check)
/storage        10.176.4.0/26(rw,sync,no_subtree_check)
```

```
$ invoke-rc.d nfs-kernel-server restart
```

Configure the boot loader

The boot loader requires to load vmlinuz and initrd.img from $TFTP_DIRS/ltsp/<architecture> with the following kernel options:

```
ro root=/dev/nfs nfsroot=/srv/ltsp/i386 ip=dhcp boot=nfs init=/sbin/init-ltsp
```

In the lab we use iPXE, loading those files over http. The iPXE script commands look like:

```
kernel http://boot.tolabaki.her.wn/ltsp/i386/vmlinuz ro root=/dev/nfs nfsroot=/srv/ltsp/i386 ip=dhcp boot=nfs init=/sbin/init-ltsp
initrd http://boot.tolabaki.her.wn/ltsp/i386/initrd.img
boot
```

The full iPXE script is available here (https://code.tolabaki.gr/tolabaki/netboot-service/blob/master/http/ipxemenu.php).

NBD (Network Block Device) server for swap files

```
$ apt-get install nbd-server
```

and add the necessary config file: /etc/nbd-server/conf.d/swap

```
[swap]
exportname = /var/tmp/nbd-swap/%s
prerun = nbdswapd %s
postrun = rm -f %s
authfile = /etc/ltsp/nbd-server.allow
```

If the swap files remain on the server after the clients have shut down, you need to disable swap file encryption with ENCRYPT_SWAP=False in lts.conf (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=690267)

NOTE: All documentation about nbdswapd on the internet is *out of date*. Just forget about nbdswapd.

Setup schroot for better chroot management

Normally one can enter the ltsp client filesystem chroot by using the command:

```
$ ltsp-chroot --arch i386 --mount-all
```

This command works fine, but has certain disadvantages, like that you must be root to execute it or that it doesn't allow you to configure what exactly gets bind-mounted, or that it preserves the shell environment without any filtering-out of dangerous variables, and others.

Schroot on the other hand is much more flexible. To set it up we need:

```
$ aptitude install schroot
```

and then add the necessary config file:

- /etc/schroot/chroot.d/ltsp.conf

```
[ltsp-i386]
type=directory
description=LTSP Fat Client Root (i386)
directory=/srv/ltsp/i386
# use /etc/schroot/ltsp/*
profile=ltsp
# users in this group will be allowed password-less root access in the chroot
```

```
root-groups=ltspadms
# if the host is amd64, we need to set the 32-bit personality of the guest
personality=linux32
# preserve environment variables from the host
preserve-environment=true
```

- /etc/schroot/ltsp/copyfiles

```
/etc/resolv.conf
```

- /etc/schroot/ltsp/fstab

```
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/proc           /proc           none    rw,bind         0       0
/sys            /sys            none    rw,bind         0       0
/dev            /dev            none    rw,bind          0       0
/dev/pts        /dev/pts        none    rw,bind          0       0
/tmp            /tmp            none    rw,bind          0       0
none            /run            tmpfs   defaults         0       0
```

- /etc/schroot/ltsp/nssdatabases

```
# empty
```

/etc/schroot/setup.d/50chrootname also needs a minor modification to avoid confusion when the client is booted:

```
diff --git a/schroot/setup.d/50chrootname b/schroot/setup.d/50chrootname
index 4b51b5f..701a5e1 100755
--- a/schroot/setup.d/50chrootname
+++ b/schroot/setup.d/50chrootname
@@ -29,5 +29,9 @@ if [ $STAGE = "setup-start" ] || [ $STAGE = "setup-recover" ]; then

     echo "${CHROOT_NAME}" > "${CHROOT_PATH}/etc/debian_chroot"

+elif [ $STAGE = "setup-stop" ]; then
+
+   rm "${CHROOT_PATH}/etc/debian_chroot"
+
 fi
```

Finally we need a wrapper script that also sets LTSP_HANDLE_DAEMONS=false before entering the chroot. This will force the client's policy-rc.d to kick in and prevent any services from being started.

```
$ cat > /usr/local/bin/enter-ltsp-chroot <<EOF
#!/bin/sh
LTSP_HANDLE_DAEMONS=false schroot -c ltsp-i386 -u root
EOF
$ chmod +x /usr/local/bin/enter-ltsp-chroot
```

Updating/Modifying the client

On the server (idea), run:

```
$ enter-ltsp-chroot
```

You can run this command either as root or as user, provided that you are in the special ltspadms group.

Inside the chroot shell, you can install, update, remove anything you want. Changes will instantly appear on the clients (unless the client has changed some of the affected files, in which case it will have its own modified version - clients use aufs, merging a writable tmpfs on top of the read-only NFS filesystem)

Make temporary changes and roll them back later

On our setup in the lab, the $BASE directory on the server is on /srv/ltsp, which is a btrfs subvolume. You can take advantage of the fact that this is a subvolume and make changes in the chroot that you can later roll back with just 2 commands.

- Step 1: Create a snapshot of the subvolume:

```
$ cd /srv
$ btrfs subvolume snapshot ltsp ltsp_snapshot
```

- Step 2: Enter the chroot and make changes:

```
$ enter-ltsp-chroot
$ ...
$ exit
```

- To roll back:
  - Make sure all clients are offline or weird things might happen (!)

```
$ invoke-rc.d nfs-kernel-server stop
$ cd /srv
$ btrfs subvolume delete ltsp
$ mv ltsp_snapshot ltsp
$ invoke-rc.d nfs-kernel-server start
```

- To keep the changes, simply delete the snapshot:

```
$ cd /srv
$ btrfs subvolume delete ltsp_snapshot
```

Add proprietary nvidia drivers on the client

```
$ aptitude install nvidia-glx nvidia-settings nvidia-detect
$ update-alternatives --set glx /usr/lib/mesa-diverted
```

Now add this script in /usr/share/ltsp/init-ltsp.d and call it let's say 50-gpu-drivers:

```
# This file is sourced

NVIDIA_IDLISTDIR=/usr/share/nvidia

if [ -z "$ENABLE_PROPRIETARY_NVIDIA_DRIVERS" -a -f $NVIDIA_IDLISTDIR/nvidia.ids ]; then
    ENABLE_PROPRIETARY_NVIDIA_DRIVERS=1
fi

# based on nvidia-detect
if [ -z "$VGA_PCIID" ]; then
    if ! (lspci --version) > /dev/null 2>&1; then
        warn "ERROR: The 'lspci' command was not found. Please install the 'pciutils' package."
    else
        DEVICES=$(lspci -mn | awk '{ gsub("\"",""); if ($2 == "0300") { print $1 } }')
        VGA_PCIID=$(lspci -mn -s ${DEVICES%% *} | awk '{ gsub("\"",""); print $3 $4 }')
    fi
fi

if [ -n "$VGA_PCIID" -a -z "$GPU_DRIVER" ]; then
    # if the device exists in the system
    if lspci -mn | awk '{ gsub("\"",""); print $3 $4 }' | grep -q -i "$VGA_PCCID"; then
        # is it nvidia?
        if echo "$VGA_PCIID" | grep -q -E '^10de|^12d2'; then
            GPU_DRIVER="nouveau"

            if [ "$ENABLE_PROPRIETARY_NVIDIA_DRIVERS" = "1" ]; then
                # TODO: uncomment when nvidia legacy drivers become co-installable

                #if grep -q -i $VGA_PCIID $NVIDIA_IDLISTDIR/nvidia-legacy-71xx.ids
                #then
                #    GPU_DRIVER="nvidia-legacy-71xx"
                #fi

                #if grep -q -i $VGA_PCIID $NVIDIA_IDLISTDIR/nvidia-legacy-96xx.ids
                #then
                #    GPU_DRIVER="nvidia-legacy-96xx"
                #fi

                #if grep -q -i $VGA_PCIID $NVIDIA_IDLISTDIR/nvidia-legacy-173xx.ids
                #then
                #    GPU_DRIVER="nvidia-legacy-173xx"
                #fi

                if grep -q -i $VGA_PCIID $NVIDIA_IDLISTDIR/nvidia.ids
                then
                    GPU_DRIVER="nvidia"
                fi
            fi
        fi

        # TODO: maybe add radeon/fglrx support
    fi
fi

case "$GPU_DRIVER" in
    nvidia)
        modprobe nvidia
        GLX_IMPLEMENTATION="nvidia"
        ;;
```

```
    nouveau)
        modprobe nouveau
        GLX_IMPLEMENTATION="mesa"
        ;;
    *)
        GLX_IMPLEMENTATION="mesa"
        ;;
esac

case "$GLX_IMPLEMENTATION" in
    nvidia)
        update-alternatives --set glx /usr/lib/nvidia
        ln -s /etc/X11/nvidia.conf /etc/X11/xorg.conf
        ;;
    *)
        update-alternatives --set glx /usr/lib/mesa-diverted
        ;;
esac
```

This script allows you to use the proprietary nvidia drivers on hosts with supported nvidia cards, use nouveau on older nvidia cards and use intel/radeon driver on non-nvidia systems without any issues.

Give useful permissions to ltsp client users

- Add in /etc/sudoers:

```
%serveradms ALL=(ALL) ALL
ALL     ALL= /usr/bin/apt-get *, /usr/bin/aptitude *, /bin/mount *, /bin/umount *, /sbin/modprobe *, /sbin/losetup *
```

- /etc/udev/rules.d/70-ltsp-acl.rules:

```
ENV{MAJOR}=="", GOTO="ltsp_acl_end"
ACTION=="remove", GOTO="ltsp_acl_end"

# systemd replaces udev-acl entirely, skip if active
TEST=="/sys/fs/cgroup/systemd", TAG=="uaccess", GOTO="ltsp_acl_end"

# parallel
KERNEL=="lp[0-9]*", TAG+="udev-acl"
KERNEL=="parport[0-9]*", TAG+="udev-acl"

# serial
SUBSYSTEM=="tty", KERNEL!="tty|tty[0-9]*|ptmx|console|pty*", TAG+="udev-acl"

# block devices except nbd
SUBSYSTEM=="block", KERNEL!="nbd*", TAG+="udev-acl"

LABEL="ltsp_acl_end"
```

The udev rules file allows the active user (according to consolekit) to gain permissions via ACL on the listed devices (parallel, serial and all block devices).