SERVIDOR
(ipv4: 192.168.1.1, interface: enp1s0)

DNSMASQ: DHCP e TFTP server

> sudo apt install dnsmasq

– Editar arquivo de configuração:

> “sudo nano /etc/dnsmasq.conf “ :

```
interface=enp1s0
bind-interfaces
domain=lib.local

dhcp=enp1s0,192.168.1.0,192.168.1.100,8h

enable-tftp
tftp-root=/srv/tftp/pxeboot
dhcp-root=syslinux.efi,<nome de usuário>,192.168.1.1
pxe-prompt=”Press F8 for PXE Network Boot.”, 2
pxe-service=x86PC, “Install OS via PXE”,pxelinux
```

– Editar Inicialização do dnsmasq
(o dnsmasq incia antes da interface ser carregada, então resultava em erro, o que requeria reiniciar o serviço dele. Com essa alteração, o dnsmasq só inicia após as interfaces)

> sudo nano */lib/systemd/system/dnsmasq.service*
> Em [Unit], definir:

```
After = network-online.target
Wants = network-online.target
```

> sudo apt install syslinux-efi nfs-kernel-server initramfs-tools -y

– Estrutura para o TFTP :

> sudo mkdir -p /srv/tftp/pxeboot/pxelinux.cfg
> sudo mkdir -p /srv/tftp/pxeboot/casper
> sudo cp /<*PATH-TO*>/syslinux.efi /srv/tftp/pxeboot/.
> sudo cp /<*PATH-TO*>/ldlinux.e64 /srv/tftp/pxeboot/.

– Editar arquivo padrão de configuração do boot PXE:

> sudo nano /srv/tftp/pxeboot/pxelinux.cfg/default (criando)

```
DEFAULT linux
LABEL linux
KERNEL casper/linux
APPEND initrd=casper/initrd.gz boot=casper netboot=nfs nfsroot=192.168.1.1:/nfsroot/pxefiles/ ip=dhcp rw
```

– Para o NFS

> sudo mkdir /nfsroot
> sudo mkdir /nfsroot/pxefiles
> sudo mkdir /nfsroot/experiments

– Configurar arquivo de exports do servidor :

> sudo nano /etc/exports

```
/nfsroot/pxefiles 192.168.1.0/24(rw,no_root_squash,async,insecure,no_subtree_check)
/nfsroot/experiments 192.168.1.0/24(rw,no_root_squash,async,insecure,no_subtree_check)
```

> sudo exportfs -rv
> sudo chmod -R 777 /nfsroot/pxefiles
> sudo chmod -R 777 /nfsroot/experiments
> sudo chmod -R 777 /srv/tftp/pxeboot (acho que não precisa)

– Gerar chave pública para SSH

> cd ~/.ssh (criar se não existir)
> ssh-keygen -t rsa -f id_rsa
    > salvo em ~/.ssh
> chmod 600 ~/.ssh/id_rsa*
> cd ~/.ssh && ssh-copy-id -i id_rsa.pub client_lib@192.168.1.xxx
> eval 'ssh-agent'
> ssh-add id_rsa

~~(copia id_rsa.pub para /nfsroot/experiments)~~

CLIENTE

UBUNTU 20.04 : fazer boot com pendrive com ubuntu 20.04.4

> Device name = Client <ou qualquer coisa>
> Username = ClienteLIB <ou qualquer coisa>
> Password = ******* <ou qualquer coisa>

– Opções de instalação :

> Instalação normal
> Logar automaticamente
> Não ativar opção de instalar software de terceiros

– Opções do sistema :

> Settings
    > Appearance
        > Auto Hide the Dock **ON**
    > Notifications
        > Popups & Lock Screen **OFF**
        > Do Not Disturb **ON**
    > Privacy
        > Delete cookies and site data when Firefox is closed
        > Screen Lock **OFF**
        > Connectivity
            > Connectivity Checking **OFF**
    > Power
        > Blank Screen **Never**

> Firefox settings
    > Settings
        > Search : DuckDuckGo
    > Privacy & Security

> Firefox Data Collection and Use **OFF**

– Sudo sem pedir senha

> sudo visudo

```
cliente_lib ALL=(ALL) NOPASSWORD: ALL
```

– Instalar pacotes

> sudo apt-get update
> sudo apt-get upgrade
> sudo apt-get install ~~mysql-client~~ ssh ~~xterm-geany~~ nfs-common autofs ~~git~~ net-tools ntp ~~xdotool~~ ~~chromium-browser~~ (talvez não precise de alguns)
> Instalações do Eduardo:
> > VSCode, QJoyStick

– Instalar Psychtoolbox + Octave

http://psychtoolbox.org/download.html

– Instalar Systemback para ubuntu 20.04

> Tente :

```
sudo add-apt-repository --remove ppa:nemh/systemback

{sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 382003C2C8B7B4AB813E915B14E4942973C62A1B}

ou

{sudo apt-key adv --keyserver pgp.mit.edu --recv-keys 382003C2C8B7B4AB813E915B14E4942973C62A1B}

sudo add-apt-repository "deb http://ppa.launchpad.net/nemh/systemback/ubuntu xenial main"

sudo apt update
sudo apt install systemback
```

– Usar autofs para montar pasta compartilhada pelo server

> sudo mkdir /nfsroot
> sudo nano /etc/auto.master

```
/nfsroot /etc/auto.nfs --ghost
```

> sudo nano /etc/auto.nfs

```
experiments -fstype=nfs 192.168.1.1:/nfsroot/experiments
```

> sudo service autofs restart

– Configurar SSH só com chave

~~> mkdir /home/neurodebian/.ssh~~
~~> touch /home/neurodebian/.ssh/authorized_keys~~
~~> cat /nfsroot/experiments/id_rsa.pub >> /home/neurodebian/.ssh/authorized_keys~~
~~> rm /nfsroot/experiments/id_rsa.pub~~

> sudo nano /etc/ssh/sshd_config, descomentar e alterar para **no**:
> > PasswordAuthentication **no**

– Criar Live Sytem com Systemback

> Abrir systemback

> Criar Live System
    > Incluir arquivos de dados do usuário (Include User data files)
> Transformar em ISO
> Enviar para o servidor
    > cp /home/systemback_*.iso /nfsroot/experiments/.

– Desligar máquina e entrar na BIOS (UEFI)

> Geral
    > Deixar boot sequency somente com opção PXE Ethernet IPv4
> Power
    > Desligar Deep Sleep Control
    > Ativar Wake on Lan (Lan Only, IPv4)
> System configuration
    > USB configuration
        > “Enable USB 3.0 Ports” **OFF** (ausente talvez)

DE VOLTA AO SERVIDOR

– Montar ISO do cliente
(*copiar arquivos da ISO montada para as pastas que serão usadas pelos clientes, para que deem boot)*

> sudo mkdir /nfsroot/systemback
> sudo mount /nfsroot/experiments/systemback_*.iso /nfsroot/systemback
> sudo rm -Rf /nfsroot/pxefiles/*
> sudo rm -Rf /srv/tftp/pxeboot/casper/*
> sudo cp -Rf /nfsroot/systemback/* /nfsroot/pxefiles/.
> sudo cp /nfsroot/pxefiles/casper/vmlinuz /srv/tftp/pxeboot/casper/vmlinuz
> sudo cp /nfsroot/pxefiles/casper/initrd.gz /srv/tftp/pxeboot/casper/initrd.gz
> sudo chmod -R 777 /nfsroot/pxefiles
> sudo chmod -R 777 /srv/tftp/pxeboot

– Limpeza

> sudo umount /nfsroot/systemback
> sudo rmdir /nfsroot/systemback
> rm /nfsroot/experiments/id_rsa.pub

– Reiniciar TFTP

> sudo systemctl restart dnsmasq.service

– Reiniciar máquina

> sudo reboot

Wake on Lan

No servidor, crie o seguinte código em python:

> touch /home/servidor_lib/WoL.py

```
import socket
import struck

# broadcast da subrede 192.168.1.0/24
broadcast_ip = 192.168.1.255

for mac_address in (“FC:34:97:7A:DB:4E”, “FC:34:97:7A:DB:4E”):
        print(f”Ligando máquina com MAC: {mac_address}“)
        # mac da máquina alvo
        mac_address = mac_address.replace(mac_address[2], '')

        data = ''.join(['FFFFFFFFFFFF', mac_address * 20])
        send_data = b''

        # Split up the hex values and pack.
        for j in range(0, len(data), 2):
                send_data = b''.join([send_data, struct.pack('B', int( data[j: j + 2], 16) ) ])

        #Broadcast it to the LAN.
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        sock.sendto(send_data, (broadcast_ip, 7))
```

Shutdown clients : SSH

> touch /home/servidor_lib/shut_clients.py

Converter chave privada para formato suportadado pelo paramiko:

> sudo cp ~/.ssh/id_rsa ~/.ssh/id_rsa_cp

> ssh-keygen -p -f ~/.ssh/id_rsa -m pem -P “” -N “”

> pip install paramiko

```
#!/usr/bin/env python3

# -*- coding: utf-8

from paramiko import SSHClient
import paramiko, time

ips = ["192.168.1.80","192.168.1.3"]

ssh = SSHClient()

ssh.set_missing_host_key_policy(paramiko. AutoAddPolicy())
k = paramiko.RSAKey.from_private_key_file("/home/servidor_lib/.ssh/id_rsa")

for ip in ips:

        ssh.connect(ip, username="client_lib", pkey=k)

        ssh_stdin, ssh_stdout, ssh_stderr = ssh.exec_command("sudo reboot", get_pty=True)
        ssh_stdin.write(input("Senha da máquina: ") + "\n")
        ssh_stdin.flush()

        ssh.close()
```

Modificar arquivos (e.g. adicionar scrips) aos clientes : filesystem.squashfs

> Copiar o arquivo filesystem.squashfs, que contém todos diretórios e arquivos dos clientes:

```
sudo cp /nfsroot/pxefiles/casper/filesystem.squashfs /media/iso
```

> Descompactar o arquivos (unsquashfs)

```
sudo unsquashfs -f /media/iso/filesystem.squashfs
```

Uma nova pasta com a descompactação será criada em ~/ (/home/servidor_lib/), com nome squashfs-root. Lá, estarão todos os arquivos, que podem ser modificados.

Após as alterações, crie um novo filesystem.squashfs apartir do squashfs-root e mova-o para o local original

```
> sudo mksquashfs /home/servidor_lib/squahfs-root /home/servidor_lib/filesystem.squashfs

> mv /home/servidor_lib/filesystem.squashfs /nfsroot/pxefiles/casper/filesystem.squashfs
```

Instalar/remover pacotes, etc. na imagem dos clientes: chroot, cubic, Ubuntu Builder

Mais simples: chroot

> Criar o squashfs-root do sistema:

```
sudo cp /nfsroot/pxefiles/casper/filesystem.squashfs /media/iso
sudo unsquashfs -f /media/iso/filesystem.squashfs
```

> alterar raiz para /squahsfs-root

```
sudo chroot ~/squashfs-root
```

> mountar /run do host em ~/squashfs-root/run, para obter configurações de rede necessárias para que conecte-se à internet

```
sudo mount -o bind run ~/squashfs-root/run
```

> Fazer instalações e alterações necessárias, através do terminal. Não é possível rodar aplicações em GUI.

> limpeza

```
apt clean
rm -rf /tmp/* ~/.bash_history
rm /etc/resolv.conf
rm /var/lib/dbus/machine-id (talvez não precise)
```

> Desmontar

```
sudo umount ~/squashfs-root /run
```

> compacte squashfs-root

```
sudo mksquashfs /home/servidor_lib/squahfs-root /home/servidor_lib/filesystem.squashfs
mv /home/servidor_lib/filesystem.squashfs /nfsroot/pxefiles/casper/filesystem.squashfs
```

Startup script

1a opção:

Criar arquivo */etc/X11/Xsession.d/* :

Nome deve começar com dois numeros, seguido de “custom_”, por ser feito um script pelo usuário, depois um nome, usando “–“ para separar palavras. Ex: 99custom_octave-deploy.

O script executa antes da área de trabalho aparecer, então só a janela do programa chamado no script aparece. Depois, quando fechado, carrega o desktop.

2a opção:

Abrir configurações e ir em Applications, procurar Startup Applications. Entre e adicione uma nova entrada, informando o nome e o comando a ser executado (ex: octave –force-gui). Irá executar depois da área de trabalho carregar.

Também é possível editando manualmente arquivo ~/.config/autostart