root@master:~# cat /etc/netplan/00-installer-config.yaml.bk

# This is the network config written by 'subiquity'

network:

ethernets:

enp193s0f0np0:

match:

macaddress: 7c:c2:55:87:43:00

addresses:

- 192.168.1.1/24

nameservers: {}

mtu: 9000

enp193s0f1np1:

addresses:

- 192.168.1.199/24

- 192.168.1.200/24

- 192.168.1.201/24

- 192.168.1.202/24

version: 2

vi /etc/xdg/canonical-certification.conf

root@rare-piglet:/home/ubuntu# canonical-certification-precheck

apt install fwts-efi-runtime-dkms

network:

ethernets:

enp193s0f0:

match:

macaddress: a0:36:9f:f7:4b:e4

addresses:

- 192.168.1.1/24

nameservers: {}

mtu: 9000

dhcp4: false

enp193s0f1:

match:

macaddress: a0:36:9f:f7:4b:e5

addresses:

- 192.168.1.199/24

- 192.168.1.200/24

- 192.168.1.201/24

nameservers: {}

mtu: 9000

dhcp4: false

enp33s0f0:

dhcp4: true

enp33s0f1:

dhcp4: false

version: 2

~

Appendix D - Network Performance Tuning. If in doubt, run more iperf3 instances

than you think you’ll need; the cost of running too many instances is very low. You

can configure iperf3 to start automatically by editing a startup script, such as

/etc/rc.local. (The generic /etc/rc.local startup script is not enabled by default in

recent versions of Ubuntu. Numerous online tutorials, such as this blog post,

describe how to enable it.)

you can adjust the -n value in your

start-iperf3 command and then run the network script manually on the SUT, specifying

the number of iperf3 instances it launches via the --num-threads option, as in:

sudo /usr/lib/checkbox-provider-base/bin/network.py test -i ens1f1 \

-t iperf --iperf3 --scan-timeout 3600 --fail-threshold 80 \

--cpu-load-fail-threshold 90 --runtime 900 --num\_runs 4 --num-threads 20 \

--target 172.24.124.1

sudo optimize-network -i ens1f1 ## You may need to run this script on the iperf3 Target, on the SUT, or both.

**Running multiple iperf3 instances with parallel**

erf3.

1. Ensure that parallel is installed.
2. # sudo apt-get -y install parallel
3. Create a file that looks like this:
4. # cat commands.txt
5. iperf3 -c 172.16.21.1 -O 15 -t 30 -p 5101 -R -i 60 -T s1
6. iperf3 -c 172.16.21.1 -O 15 -t 30 -p 5102 -R -i 60 -T s2
7. iperf3 -c 172.16.21.1 -O 15 -t 30 -p 5103 -R -i 60 -T s3
8. iperf3 -c 172.16.21.1 -O 15 -t 30 -p 5104 -R -i 60 -T s4
9. Execute the commands like this:
10. # parallel -a commands.txt |tee -a 100Gb-Port0.log

command = "efibootmgr -v"

bootinfo\_bytes = (Popen(shlex.split(command), stdout=PIPE)

.communicate()[0])

bootinfo = (bootinfo\_bytes.decode(encoding="utf-8", errors="ignore")

.splitlines())

boot\_entries = {}

boot\_order = []

boot\_current = ""

if len(bootinfo) > 1:

for s in bootinfo:

if "BootOrder" in s:

try:

boot\_order = s.split(":")[1].replace(" ", "").split(",")

except IndexError:

pass

elif "BootCurrent" in s:

try:

boot\_current = s.split(":")[1].strip()

except IndexError:

pass

id: miscellanea/efi\_pxeboot

requires:

cpuinfo.platform in ("i386", "x86\_64", "aarch64")

depends: miscellanea/efi\_boot\_mode

\_summary: Test that system booted from the network

\_description:

Test to verify that the system booted from the network.

Works only on EFI-based systems.

command: efi-pxeboot.py

plugin:shell

id: miscellanea/oops

category\_id: com.canonical.plainbox::miscellanea

estimated\_duration: 10.0

requires: executable.name == 'fwts'

user: root

\_description:

Run Firmware Test Suite (fwts) oops tests.

\_summary:

Run FWTS OOPs check

environ: PLAINBOX\_SESSION\_SHARE

command:

checkbox-support-fwts\_test -l "$PLAINBOX\_SESSION\_SHARE"/fwts\_oops\_results.log -t oops

plugin: attachment

category\_id: com.canonical.plainbox::miscellanea

estimated\_duration: 0.5

id: miscellanea/oops\_results.log

command:

[ -e "${PLAINBOX\_SESSION\_SHARE}"/fwts\_oops\_results.log ] && xz -c "${PLAINBOX\_SESSION\_SHARE}"/fwts\_oops\_results.log

\_description: Attaches the FWTS oops results log to the submission

plugin: shell

category\_id: com.canonical.plainbox::miscellanea

estimated\_duration: 0.5

id: miscellanea/efi\_boot\_mode

requires:

cpuinfo.platform in ("i386", "x86\_64", "aarch64")

\_summary: Test that system booted in EFI mode

\_description:

Test to verify that the system booted in EFI mode with Secure Boot active.

command: boot\_mode\_test.py efi

plugin: shell

id: udev\_resource\_attachment

plugin: attachment

category\_id: com.canonical.plainbox::info

command: udev\_resource.py

estimated\_duration: 0.432

\_description: Attaches the output of udev\_resource.py, for debugging purposes

plugin: attachment

category\_id: com.canonical.plainbox::info

id: info/buildstamp

template-engine: jinja2

estimated\_duration: 0.1

\_description: Attaches the buildstamp identifier for the OS

\_summary: Attaches the buildstamp identifier for the OS

command:

{%- if \_\_on\_ubuntucore\_\_ %}

if [ -s /run/mnt/ubuntu-seed/.disk/info ]; then

cat /run/mnt/ubuntu-seed/.disk/info

elif [ -s /writable/system-data/etc/buildstamp ]; then

cat /writable/system-data/etc/buildstamp

elif [ -e /var/lib/snapd/seed/seed.yaml ]; then

echo && date -r /var/lib/snapd/seed/seed.yaml -R

else

exit 1

fi

{% else -%}

if [ -s /var/lib/ubuntu\_dist\_channel ]; then # PC projects

cat /var/lib/ubuntu\_dist\_channel

elif [ -s /var/log/installer/media-info ]; then # Stock installer info

cat /var/log/installer/media-info

elif [ -s /.disk/info ]; then

cat /.disk/info

elif [ -s /etc/media-info ]; then

cat /etc/media-info

else

exit 1

fi

{% endif -%}

plugin: attachment

category\_id: com.canonical.plainbox::info

id: installer\_debug.gz

user: root

command: [ -f /var/log/installer/debug ] && gzip -9 -c /var/log/installer/debug

estimated\_duration: 0.1

\_description: Attaches the installer debug log if it exists.

plugin: shell

category\_id: com.canonical.plainbox::networking

id: networking/ntp

flags: also-after-suspend

requires: package.name == 'ntpdate'

user: root

command: network\_ntp\_test.py

\_description: Test to see if we can sync local clock to an NTP server

plugin: attachment

category\_id: com.canonical.plainbox::info

id: info/kvm\_output

estimated\_duration: 0.1

\_summary:

Attaches console log from the kvm\_check\_vm test

\_description:

Attaches the debug log from the virtualization/kvm\_check\_vm test

to the results submission.

command: [ -f "$PLAINBOX\_SESSION\_SHARE"/virt\_debug ] && cat "$PLAINBOX\_SESSION\_SHARE"/virt\_debug

plugin: attachment

category\_id: com.canonical.plainbox::info

id: info/kvm\_output

estimated\_duration: 0.1

\_summary:

Attaches console log from the kvm\_check\_vm test

\_description:

Attaches the debug log from the virtualization/kvm\_check\_vm test

to the results submission.

command: [ -f "$PLAINBOX\_SESSION\_SHARE"/virt\_debug ] && cat "$PLAINBOX\_SESSION\_SHARE"/virt\_debug

**Debug symbol packages**

**Import the signing key**

Import the debug symbol archive [signing key](https://help.ubuntu.com/community/Repositories/Ubuntu#Authentication_Tab) from the Ubuntu server. On Ubuntu 18.04 LTS and newer, run the following command:

sudo apt install ubuntu-dbgsym-keyring

**Create a ddebs.list file**

Create an /etc/apt/sources.list.d/ddebs.list by running the following line at a terminal:

echo "deb http://ddebs.ubuntu.com $(lsb\_release -cs) main restricted universe multiverse

deb http://ddebs.ubuntu.com $(lsb\_release -cs)-updates main restricted universe multiverse

deb http://ddebs.ubuntu.com $(lsb\_release -cs)-proposed main restricted universe multiverse" | \

sudo tee -a /etc/apt/sources.list.d/ddebs.list

You can also add these repositories in your software sources from the Ubuntu software center or from Synaptic (refer to [this article](https://help.ubuntu.com/community/Repositories/Ubuntu), especially the section on [adding other repositories](https://help.ubuntu.com/community/Repositories/Ubuntu#Adding_Other_Repositories)). You will need to add lines like:

deb http://ddebs.ubuntu.com focal main restricted universe multiverse

**Note**:  
Make sure you replace “focal” with the Ubuntu release name you’re using.

**Update package list**

Run the following to update your package list or click the Reload button if you used the Synaptic Package Manager:

sudo apt-get update

**Manual install of debug packages**

To install the debug symbol package (\*-dbgsym.ddeb) for a specific package, you can now invoke:

sudo apt-get install PACKAGE-dbgsym

For example, to install the debug symbols for xserver-xorg-core:

sudo apt-get install xserver-xorg-core-dbgsym

As mentioned in the section above, some packages will ship their debug symbols via \*-dbg.deb packages instead. Using glibc as an example, you can install its debug symbols using:

sudo apt-get install libc6-dbg

This procedure will install the debug symbol package for a single package only. It is likely that the binary uses shared libraries in other packages, and their debug symbols may be needed in order to obtain a readable stack trace or perform other debugging tasks.

**The debian-goodies tool**

You can use the find-dbgsym-packages command from the debian-goodies package to find debug symbols for a core file, running PID or binary path.

For a binary path it only finds debug symbols for the actual binary itself, and not any dynamically linked library dependencies or other libraries loaded at runtime. For that functionality to work you need to use either a core file or a running PID (which is the preferred method).

This tool will find both -dbg and -dbgsym style packages. However it only finds debug symbols for APT repositories that are currently enabled and updated, so you need to ensure that you enable at least the ddebs.ubuntu.com archive as described above. For a Launchpad PPA or the Ubuntu Cloud Archive you need to add another source line with the component changed from main to main/debug:

sudo apt install debian-goodies

find-dbgsym-packages [core\_path|running\_pid|binary\_path]

udevadm control --reload && udevadm trigger

network:

ethernets:

enp193s0f0:

addresses:

- 192.168.1.254/24

match:

macaddress: 7c:c2:55:79:81:1e

mtu: 9000

nameservers:

addresses:

- 192.168.1.1

search:

- maas

set-name: enp193s0f0

enp193s0f1:

match:

macaddress: 7c:c2:55:79:81:1f

addresses:

- 192.168.1.199/24

- 192.168.1.200/24

- 192.168.1.201/24

mtu: 9000

nameservers:

addresses:

- 192.168.1.1

search:

- maas

set-name: enp193s0f1

usb0:

match:

macaddress: 00:00:00:00:00:00

mtu: 1500

set-name: usb0

version: 2

**if** option arch = 00:07 **and** exists user-**class** **and** option user-**class** = "iPXE" {

*# iPXE uefi\_amd64*

filename "grubx64.efi";

} **elsif** option arch = 00:09 **and** exists user-**class** **and** option user-**class** = "iPXE" {

*# iPXE uefi\_amd64*

filename "grubx64.efi";

}



