wce-triage-v2 Documentation

Release latest

Apr 08, 2023

Contents

CHAPTER 1

WCE Triage App

1.1 About World Computer Exchange

World Computer Exchange (WCE) is a volunteer organization, located in Hull, Massachusetts, USA. The primary mission of WCE is to refurbish and reuse computers in developing countries. See https://www.worldcomputerexchange.org for more details.

What is WCE Triage? When a donated computer arrives, the volunteers first physically clean and assess the computer. We call this process triaging. For hardware, volunteers replace or install components as needed based on the triage. In order to streamline and have consistent decisions about the state of computer, we use software to gather information.

Since WCE ships the computers with Ubuntu as OS, it makes sense to run Ubuntu to triage.

This package is designed around running a minimal Ubuntu/Linux, and gather computer states, and displays the triage information. Let's call it **Triage App**. Triage app also restores disk image to computer disk, and creates disk image from the computer's disk as well.

1.2 Overview of WCE operations

For the operation of WCE, there are 4 categories of computers involved.

1. **Desktop client** - this is the product we produce
2. **Triaging computer** - this is a computer being triaged by the WCE voluteers using Triage app. The computer will be a desktop client when it passes the test, repaired or salvaged.
3. **Network Server** - this is a Ubuntu/Linux server that is capable of running Triage app on PXE booted client over network, and thus allows PXE booted Desktop clients to install the disk image over network. Typically, this is a head-less server that is used during the game day.
4. **Workstation** - this is a Ubuntu/Linux desktop that is used for authoring WCE contents and disk images. The workstation is also a Network Server as well. This allows us to check out the disk image created immediately by

connecting a desktop client to the workstation. Typically, this is a Ubuntu desktop computer used for imaging disks and creating contents off-game day.

Triage app contains a setup script for each category. For example, setup script for Workstation installs all necessary Ubuntu and Python packages, prepares the disk image directories, installs dnsmasq for PXE boot, inetd/tftp server for network boot, kernel NFS server for client's NFS booting, etc.

1.2.1 Disk image installation operation

Once Triage app runs, go to "Load Disk" tab, select the source disk image, and a disk to load. Click "Load". Operation is the same on both USB based Triage app or network based. When you choose the source, the restore type is automatically chosen, and generally you should not change the restore type. (See *Disk Image Directories* for the details.)

If USB flash drive has enough storage space, it's possible to restore disk of the desktop client.

1.2.2 Creating disk image operation

Once Triage app runs on Workstation, or Network server, go to "Create Disk Image" tab, chose the source disk, and choose the disk image type. For Ubuntu 18.04LTS based computer, you must choose "WCE Ubuntu 18.04LTS". For older versions of Ubuntu, choose "WCE Ubuntu 16.04LTS".

"Triage USB flash drive" is for the disk image of USB sticks and not for desktop clients.

1.3 Setting up Workstation/Network server

As the triage app contains the setup script for workstation, it requires to install Python3 pip, and install "wce_triage" Python package on the computer. The setup script is exptected to run on freshly installed Ubuntu 18.04LTS desktop (or WCE's desktop client.)

```
$ sudo -H apt install -y python3-pip
$ sudo -H pip3 install --no-cache-dir -i https://test.pypi.org/simple/ --no-deps wce_
↪triage
```

From the terminal, once this is done, run following command.

```
$ python3 -m wce_triage.setup.setup_workstation
```

Similary for Network server, run *python3 -m wce_triage.setup.setup_network_server*. Once again, this is expected to run on freshly installed Ubuntu 18.04LTS server with only OpenSSH server isntalled during installation. If you already have installed dnsmasq with your own settings, or lighttpd server, etc., you should avoid running the set up script as it overwrites the config files. There is no config back up or any of precautions included. You are warned.

1.4 Creating Bootable Triage App on Disk/USB stick

This is the insturctions of creating USB stick that runs Triage app. Since the Triage app can load the triage app disk image to USB stick, this is not often practiced. Bootstrapping is hard, and knowledge must be kept somewhere. In the future (very likely year 2020 for Ubuntu 20.04LTS), I have to do this again.

1.4.1 Step 1: Acquire Ubuntu 18.04LTS mini.iso installer

Making a bootable USB stick from mini.iso is tricky. 'Create Installer' of Ubuntu does not work for mini.iso.

- For Mac:

  Use balenaEtcher. This macOS app works and probably the simplest.

- For Linux:

  Most likely, "dd" works. Find out the USB stick device and

  dd if=mini.iso of=/dev/<USB_STICK_DEVICE> bs=1M

1.4.2 Step 2: Install mini.iso to a disk

Disk can be an external disk, USB stick, etc. I recommend using a normal disk (or SSD) to make it faster rather than USB stick. Boot from mini.iso bootable and install minimal. Machine name is "wcetriage". User name/password is "triage/triage".

1.4.3 Step 3: Bootstrap

Once installation is done, boot into the installed system. One way or the other, you need to get network going. mini.iso is bare-bone (on purpose.)

Here is what you can do:

- if you have an ethernet, use it. First, find out the ethernet device name.

  $ ip addr

Usually, "lo" is the loopback device and first. 2nd and on is the network device.

```
2: <YOUR-DEVICE-HERE>: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500...
```

create netplan file

```
$ sudo mkdir /run/netplan
```

Using text editor, create a netplan file as follow. Indentation is critical to netplan so this should look exactly as follow

```
#/run/netplan/bootstrap.yaml file example
#
network:
  version: 2
  renderer: networkd
  ethernets:
    <YOUR-DEVICE-HERE>:
      dhcp4: yes
      optional: yes
```

start network

```
$ sudo netplan generate
$ sudo netplan apply
```

1.4.4 Step 4: Download wce_triage software

```
$ sudo -H apt install -y python3-pip
$ sudo -H pip3 install --no-cache-dir -i https://test.pypi.org/simple/ --no-deps wce_
↪triage
```

At this point, if you want to switch over to use WIFI instead of ethernet, you can do so by

```
$ sudo -H python3 -m wce_triage.bin.start_network
```

This module scans the network devices and runs netplan. If you want to use WIFI, set up a guest network as follow

```
SSID: wcetriage
Wifi password: thepasswordiswcetriage
```

You can use your existing network.:

```
$ export TRIAGE_SSID=<YOUR-SSID>
$ export TRIAGE_PASSWORD=<YOUR-WIFI-PASSWORD>
$ sudo -H python3 -m wce_traige.bin.start_network
```

"wcetriage" - is used for testing WIFI device during WCE's triage. In other word, if you have a wifi router with wcetriage/thepasswordiswcetriage, running triage software automatically connects to the wifi router thus it tests the WIFI device.

1.4.5 Step 5: Install the rest of WCE triage assets and set up the installer

```
$ python3 -m wce_triage.setup.setup_triage_system
```

You should run this from terminal. It probably asks you some questions. Answer appropriately. For grub installation, install to the disk device you booted. Once the set up script has done it's job, the disk is bootable and ready for the triage.

Since the setup script is still weak - meaning that, it may fail for many and unknown reasons. Please let me know by filing bug at the project bug report.

CHAPTER 2

Triage App archtecture

Now, how-to part is done. Let's get into the technical part of Triage app.

2.1 Grand Overview

Triae app is made out of two pieces - the backend "WCE Triage" which is the engine part of operations, and Triage UI which is Web based user interface. This exercises major parts of desktop client. It runs same Xorg X-server, Pulseaudio server, so if any major component is missing such as incompatible video card or missing sound driver on Ubuntu, we will catch it.

It also allows us to run the same Triage app on workstation for disk imaging and loading disk image from the web browser already on the workstation.

2.2 wce-triage overview

The core of WCE triage is written in Python3. The reason is that, the mini.iso of Ubuntu 18.04LTS includes Python3 already so to not increase the footprint, Python3 is a natural choice. The source code is available at https://github.com/ntai/wce-triage-v2. (This readme is part of it.) The details are in the latter part of this document.

2.3 wce-kiosk overview

The front-end UI uses React.js, and the source is available at https://github.com/ntai/wce-triage-ui. For the details, please see the project document. it's developed on Mac by me at the moment, and quite crude. The release build does not require anything extra from internet, and HTTP server in wce-triage handles the requests.

2.4 WCE Triage backend (wce-triage-v2)

The package provides following features:

- Triage information gathering and decision making
- Initialize/partition triage USB stick
- Initialize/partition disk for WCE's computers
- Create disk image from partition (aka image disk)
- Load disk image to partition (aka load/restore disk)
- Wipe disk by zero fill (no other methods provided as of now)
- Make usb stick/disk bootable
- HTTP server for WCE Kiosk web browser

In the source tree, there are following directories, "bin", "components", "http", "lib", "ops", "setup".

2.4.1 "components" directory

Each file here represents the major component of computer. During triage, each component gathers info on the machine. "computer" component works as the clearing house/storage of components.

2.4.2 "bin", "lib", and "ops" directories

The files here are the back end of disk operations. The real details of design will have to wait for documenting the source code. For now, each "task" represents each step of disk operation, and "task runner" or "runner" runs these tasks in sequence to do the disk operations. For example, to partition a disk, "partition runner" creates all necessary tasks and runs it. A task in it runs "parted" to partition the disk, "fetch" to read the parition map, "refresh" to get the partition information, and "mkfs" task runs mkfs command for the partitions. Some of more "difficult" operation such as reading compressed disk image and restoring it to disk is written as a standalone command in "bin" directory, and a task runs the "bin" to complete the task.

The design of task and task runner can be discussed and critiqued to no end but braking down small operations into task so far was a real winner as I can assemble the tasks in different ways for different application and yet I don't need to write same operations twice.

2.4.3 "http" directory

There is only one file in this. httpserver.py. The server is based on aiohttp package that uses Python's asyncio.

Once the backend's functionalities are implemented and tested, wiring up the functionality such as create disk image is pretty straightforward. However, as aiohttp being coroutine, you need to care what operation is blocking. For example, Python's standard "time.sleep()" halts entire process, or looping on reading file blocks other http request. To make this to work, you need to dive into many different Python libraries. If the code looks simple, I've done a good job.

2.5 WCE Triage details

- It boots a minimalistic Ubuntu Linux.
- When it boots, it starts two services "wce-triage" and "wce-kiosk" as described above.

2.5.1 Triage information gathering and decision making

Information gathering of individual component is in each python module in wce_triage/components, except computer/Computer. Currently, following components are implemented. - cpu.py - disk.py - memory.py - network.py - optical_drive.py - pci.py - sensor.py - sound.py - video.py

The module name says pretty much what it is. Disk and network are somewhat special as the rest of wce-triage uses the instances of disk and network during not just triaging but imaging/restoring partclone image as well as starting network during triage.

Computer module collects the components' information and makes the triage decision. The criteria of triage is decided by WCE.

2.6 WCE Disk Image File and Directories

In order to make things "simple" and consistent, I designed a simple structure for the disk image. The disk images are stored in */usr/local/share/wce/wce-disk-images*. Under the directory, there are subdirectories. For now, conventions are "triage", "wce-16" and "wce-18". "triage" is for Triage USB image, "wce-16" for Ubuntu 16.04LTS and older, and "wce-18" for Ubuntu 18.04LTS and newer.

The reason Ubuntu16.04 and 18.04 have to be separted is based on the EXT4 file system is not backward compatible. When you mkfs EXT4 partition for Ubuntu 16.04 on 18.04 machine, you need to pass down an option to not use "metadat_csum". If not, Ubuntu 16.04LTS disk loaded on EXT4+metadata_csum cannot boot.

You can have arbitary subdirectory under "wce-disk-images". So, we start producing Ubuntu 20.04LTS, we'd create "wce-20" (or any other name).

In the subdirectory, each subdirectory must contain a disk image metadata. For this, you need to create a file named ".disk_image_type.json".

Here is the actual example of it in "wce-16".

```
{ "id": "wce-16",
  "filestem": "wce-mate16",
  "name": "WCE Ubuntu 16.04LTS",
  "timestamp": true,
  "ext4_version": "1.42",
  "partition_map": "gpt"}
```

The "id" must match with the subdirectory name. (It probalby works even if it doesn't but that's the convention.) This is a tag that the web browser uses for disk image type ID. "filestem" is used when you create a disk image. So, if you create a disk image in this directory, the file name starts with "wce-mate16". "timestamp" should be always true to ID when the disk image is created. The disk image creation app always adds the file system in its name as well. "ext4_version" is the one mentioned above. By declaring the ext4_version (which is actually the version number of libext4, I think), the partition task adds necessary mkfs option for Ubuntu 16.04 even if it's running on 18.04.

For wce-18, ext4_version is 1.44.

With the locations well known, httpsever easily finds all of disk images with it's metadata, and sends it up to web browser. Also, when you create a disk image, the image name is always consistent, and stored in well known location.

It's not difficult to have different "wce-disk-images" directory, and as a matter of fact, if you mount a different disk and there is a directory right below the mount point, httpserver will find it as well for loading. However, for creating image, it's always stored in "/usr/local/share/wce/wce-disk-images/FOO".

2.7 Network Server for PXE boot and triage/disk imaging

The setup script does the servers set up but there are two important ingredients missing. One is the kernel/initrd for initial boot, and the NFS root directory for the desktop client. For the former, you need "/var/lib/netboot" directory sufficiently stuffed. "setup/install_pxeboot.py" should take care of this part.

2nd ingredients is the root file system. "/var/lib/netclient/wcetriage" needs to be filled by the "triage disk". With working triage USB stick (or disk) in hand, you need to mount the disk, and rsync everything from the triage disk to the "wcetriage" directory. NFS server serves this directory as NFS root for client to use.

2.7.1 ONE VERY IMPORTANT INGREDIENTS FOR TRIAGE AND NETWORK BOOT - CUSTOM INITRD

For triage app to run on USB stick or NFS mounted root which is read-only, it needs to run using "unionfs" - aka aufs. What this does is to layer a file system over other file system. The base layer (read-only) is accesed if upper layer (writable and memory based tempfs) doesn't have the file, and if a file is modified or created, it stays on the upper layer.

To this to work, initrd file contains a script to set up the aufs by creating tempfs, moving read-only file system to "/ro", and mount the aufs as root "/" file system. If you do not recreate (aka update) the initrd without this script, this does not work. triage.setup.setup_FOO installs the script and updates initrd file. If you use a stock initrd, this brakes down. If you are curious, you can take a look at the script for initrd. *wce-triage-v2/wce_triage/setup/patches/server/etc/initramfs-tools/scripts/init-bottom/__rootaufs* is the shell script for this. Same copy is included for triage and workstation, but not in the desktop client for obvious reason.

2.7.2 Network Server Post Installation Configuration

In order for network server to work properly, you have to manually configure the network interface (for now). This is because the network server (and workstation as well) need to prohibit offering DHCP on the NIC that is connected to your network. For PXE to work, it needs to have it's own subnet/separate network from your LAN, or else your LAN would be totally confused by more than one DHCP server running, and one of them is this destructive Triage app server. In some near future, I am thinking about the network setting to be done on the Triage web as well, but until I get there, you need to manually edit /etc/dnsmasq.conf and /etc/netplan/foo.yaml for your network hardware.