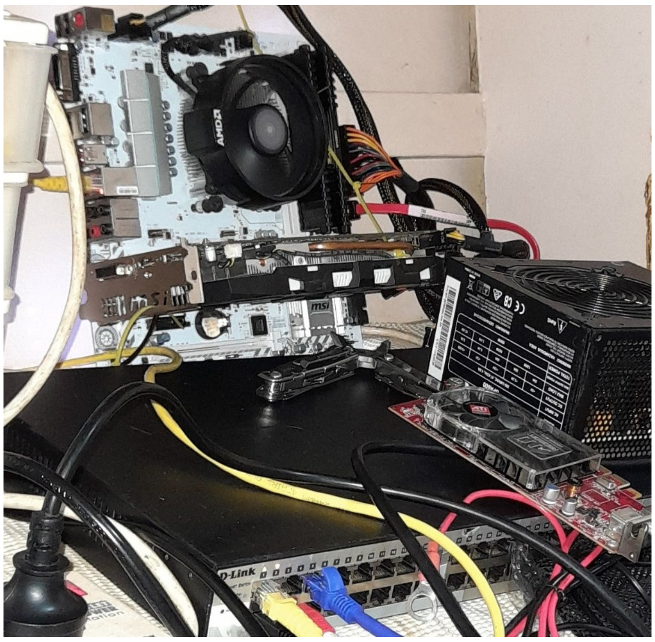
Project Delta

by Maurice Blake



Project Delta is a case-less linux server that is used for general computational power and availability and features in a few of my projects due to its nature.

Specs:

- B350m mortar arctic motherboard
- AMD stealth cpu cooler with thermal-take thermal paste
- 16gb DDR4 4200mhz vengeance ram
- GTX 1060 6gb msi
-

Software:

- Ubuntu 20.04 (OS) server
- Docker
- Samba file server
- T-rex ethereum mining script with daemon-ised process

Background information:

Delta used to be a desktop of mine which I used to play games on with but after the demand for computational power grew and complexity of my projects which required a non-stop computer I decided to install Ubuntu 18 and use it as my main system initially creating VMs to be used in my network such as a dedicated windows 7 device to test the safety of certain programs before I release to my computers.

Another use which became apparent was that I needed a file server after server Raymond became too unreliable to justify using it without the risk of loosing terabytes of data due to drive errors.

Origin:

Delta was one of 4 computers initially used in my network setup, Delta originated from a dell Optiplex of which I upgraded the hardware and fitted a ATI graphics card to allow the option for basic graphical programs and applications. After a setting up my rack with dl380 g5 serversI then had no proper use for Delta so I used it as a dedicated CD-ROM burner for installing Proxmox and Rompaq for the dl380`s. Although after moving houses I needed a compact PC to be used to help fill the place of my old servers until I could get them back, so I moved the HDD from the Optiplex to the motherboard and fitted it with a GTX 1060 and 16gb ram and upgraded the OS to Ubuntu 20.04 and made it into my main system which I ran headless and accessed with ssh.

Tasks done:

Daemon-ised T-rex ethereum miner:

After using T-rex and being limited by how it outputs to the shell and it runs parented by the ssh session meaning exiting the connection would cause the process to end, a solution to this I found was to redirect stdout and stderr and make it a background process. How I did this was directing the output and errors to **‘/dev/null’** and run as a background task, usually to do that is to hit ‘CTRL-Z’, and ‘bg’ to then put the task in the background as a job. But I can just append the full command with ‘&’ and that will do the same thing without having to emulate keystrokes. So **‘[command] &’** Unix devices have a null device which is a ‘blackhole’ of sorts which whatever data is written to it is deleted. In this case I wanted to delete ‘standard output (stdout)’ and ‘standard error (stderr)’ because I didnt want random text and errors being spawned from the commandwhen I want it to run silent and in the background, to direct to the unwanted information I used ‘>’ followed but **‘dev/null’** which is the location of the null device, so ***‘[command] > dev/null’***. But this alone wont do errors because it is only doing stdout, so I needed to append this command by specifying that i want to direct both stderr and stdout to null so the command with ***‘[command] 1> /dev/null 2>&1’***. The ‘1’ is a file descriptor for stdout and ‘2’ is the file descriptor for stderr, so what the full command means is:

1. stdout is directed to null.
2. stderr is directed to file descriptor ‘1’ and ‘&’ tells that ‘1’ is a file descriptor not a file name.
3. Since stderr is directed to the stdout stream, it ends up being directed to null aswell.

So the full command in the case of T-rex being used would look like:

‘./T-rex.run 1> /*dev/null 2>&1’*