Midnight Code Documentation

Saturn Installation and Operations Manual

An Instructional Guide and Handbook for v2 (Daphnis)

Classified: PUBLIC

Project Code: “Saturn”

Status: RELEASE
Version: 2.0
Issue Date: June 30, 2020
Document Type: CONTROLLED

Table of Contents

1 How to use this manual

This manual is a guide to assist people with the assembly, configuration and operation of a generation two, Midnight Code Saturn storage appliance, known as *Daphnis*.

The manual has been designed so that it can be read from start to finish to build a capability from infrastructure to service, or it can be used as a ready reference to seek out targeted information, by concept.

Where to find ..

This manual consists of five sections:

Chapter 1 – How to use this manual

This chapter (the one you're reading now) provides detail on the structure of the manual, how each section should be used and what standard mnemonics have been applied.

Chapter 2 – About Saturn

Understand Saturn: Learn about what Saturn is, what it is intended to be, how it came about and where it is going. Chapter 2 is a good chapter to read if you just want to know if Saturn is going to fit your storage needs.

Chapter 3 – Installation

From hardware to software installation, chapter 3 takes you through a build process from the ground up (including pictures), and calls out things that you should consider before building your own. If you already have hardware then treat chapter 3 like a ready-reference for finding the software solution best suited to your build.

Chapter 4 – Operation

Once you have your Saturn device ready, you need to know how to configure it. Learn how to configure the Saturn device, establish a storage resource and serve it to the network. Then read on to understand what maintenance and diagnostic options are available to you and what common problems can be quickly remediated.

Chapter 5 – Licensing

All of the licensing information about Saturn has been published in chapter 5. This includes the licensing of the distribution itself, the copyright of the Midnight Code software, and the licensing of the constituent software packages that make up the distribution.

You will find concepts in each section are also grouped into subsections that further reflect the relationships of that material.

Conventions

This manual uses a set of common notations to improve its readability and reference-ability.

- **Box**

  Information found in a grey box like this one means that the included text is a literal command or configuration item;

  ```
  type in this command or configuration item
  ```

  Be careful with line wrapping in the command boxes when cutting and pasting from the document.

- **List**

  Numbered lists should be evaluated in the documented order. Bullet-point lists are lists of items that belong to a concept without any particular order. Both types of list can contain nested lists which reflect an ordered or unordered list of derivative concepts (respectively).

- **Link**

  Cross referencing (linking from one part of this document to another) will be used to indicate related concept or dependent process without repeating the same text twice in the manual. External links (linking from this document to other documents or Internet resources) are provided for further reading, or to indicate the source of a concept, data or file.

This manual has been written in English with Australian/British spelling.

Completeness

This manual is structured to allow you to test its completeness and relevance;

1. Physically, this manual is 105 pages long. If you have less than 105 pages (including the cover page) then you are missing a page and should obtain a full copy of the document. All pages, other than the cover, are numbered.
2. Logically this manual is at version number 2.0, which means that it was written against version 2.0 of the Saturn software. If you have a version of the software that is greater (more recent) than the version of this manual, then check to see if there is a newer manual.

This manual is stored electronically at the Midnight Code web site. It should be accessible via both the Project site[1] and the Papers site[2].

1 http://midnightcode.org/projects/saturn/
2 http://midnightcode.org/papers/

2 About Saturn

Project Saturn is intended to deliver a Network Attached Storage (NAS) solution from commodity, off-the-shelf hardware and open source software.

Each of the Saturn versions was named after the moons of Saturn[3] (prior to the discovery of S/2009 S 1[4], and setting aside the moonlets[5]). This document describes the second generation of Saturn, known as *Daphnis.*

Saturn Generation 1 – Pan

Saturn was an embedded platform that created a Network Attached Storage (NAS) device from any modern x86 computer and without specialist hardware.

Saturn was built on GlusterFS which catered for most standard RAID-like configurations (distributed, redundant and striped storage models), as well as supporting network-based asynchronous file replication to enable more common enterprise storage network (multi-box, multi-rack, multi-site) configurations. Beneath GlusterFS, Saturn is a portable Linux distribution produced by Midnight Code that would boot from a 32MB USB key ensuring no storage was wasted on firmware.

Saturn Generation 2 – Daphnis

Taking advantage of more than a decade's worth of hardware improvements, and asserting the principle of horizontal scalability, Saturn is now a NAS comprised of individual storage nodes – multi-core ARM processors and gigabytes of memory per connected drive; current off-the-shelf technology that is reflective of industrial evolution.

To better suit the objectives of the project, Gen2 Saturn has been built on the distributed file-system MooseFS. In this generation, no effort has been made to embed the distribution, instead leveraging an Ubuntu distro that has been modified by the hardware OEM to support the tin. Unfortunately this means the firmware is a whale at 2GB, but is deployed via Micro SD card to ensure that even at this scale, no storage is wasted on firmware.

2.1 Objectives of Project Saturn

Saturn aims to be a high quality Network Attached Storage (NAS) appliance that can be assembled by home and/or office users without the expense and complexity of enterprise solutions.

2.1.1 Drivers

Storage is fundamental to any information system, and not just via its capacity[6]. The richer the storage capability – the more robust and reliable it is, the more integral and the more secure it is – the greater the capability that it enables in downstream architectures. Storage moves from being "a place for files", to a highly available, high speed delivery platform capable of being the one repository for all digital assets (tools and information) and a modern example of that is S3.

Project Saturn was kicked off in another era – in 2006 we needed about 2TB of storage and had to construct it from 300MB drives, while in 2020 we can use 16TB drives to construct a PB – but in all that time the problem has not changed. We thirst for storage from the same needs.

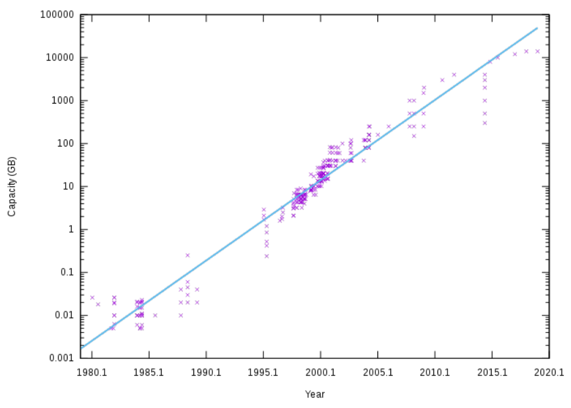


Figure 1: HDD capacity in GB, plotted against time

3 https://en.wikipedia.org/wiki/Moons_of_Saturn#Confirmed_moons
4 https://en.wikipedia.org/wiki/S/2009_S_1
5 https://en.wikipedia.org/wiki/Moons_of_Saturn#Ring_moonlets
6 https://en.wikipedia.org/wiki/File:Hard_drive_capacity_over_time.svg

Firstly there were a number of market drivers that lead to the creation of Project Saturn:

- Commercial NAS (Network Attached Storage) and SAN (Storage Area Network) solutions tended to be unnecessarily expensive. Hence, despite their impressive features and maturity, they aren't practical.
- Alternatively, the consumer grade storage devices were under powered (at the time there was 10Mbps performance on 1Gbps NICs), limited enclosure capacity (hard upper bound limits on scalability), as well as proprietary on-disk data formats that couldn't be recovered in case of hardware failures and other solutions are not extensible at all (adding a drive adding another logical volume, not more capacity to the existing logical volume).
- Modern market drivers also include putting a cap on cost and managing data sovereignty – keeping data on premises lowers latency, cost and risk.

And then there were a number of operational and investment drivers:

- Mirroring as a means to achieve high availability was unfavourable due its cost (mirroring 5 drives costs 10 drives, while RAID5 of 5 drives costs 6 drives for example).
- RAID solutions were limited to a single host – so there was no way to scale beyond one chassis' worth of capacity, nor was there any way to do off-site replication with RAID alone.
- Replacing hardware makes no sense – i.e. replacing a 1TB drive with a 2TB drive gives you one new terabyte of capacity at the cost of two terabytes – implying that new capacity should only ever be added, and old capacity should only be removed after it has come to the end of its service life (i.e. failed in situ).
- Adding a drive should add capacity to an existing logical capacity (i.e. concatenation of drives should be achievable without the rebuilding the logical volume and redeploying the data).
- Replication and Concatenation functions should not be limited to a single chassis/enclosure or even a single site – there should be no practical upper limit to the amount of storage that can be created from the acquisition of COTS (Common Off The Shelf) hardware, and the Saturn software.
- Ideally the storage fabric is secure – encrypted at rest, in transit and authenticated as consumed. This driver was not included in the first generation as the COTS CPU capacity was insufficient to be practical.
- The storage service should also be easy to access from many clients, easy to operate (with no/low maintenance) and hard to lose data.

Modern drivers could be said to be things like:

- Flexible interfaces – the ability to add/expose an object storage interface for example, or;
- Flexible storage classes – choose for IOPS or Capacity at the app layer, or;
- Power consumption – minimise power overhead beyond the drive loading requirements.

2.1.2 Target Infrastructure Features

The above drivers results in the following infrastructure *features*, whereby Project Saturn design decisions are considered against the following feature objectives:

- **Distributed**
  - Must be capable of concatenating multiple drives on either a single node or multiple nodes into a single contiguous storage capacity
  - Must be expansible such that adding another drive to an existing storage capacity must not require recreation of that storage capacity (destroying, creating, formatting)
- **Highly Available**
  - Must be capable of replicating data between drives on either a single node or multiple nodes into a

single highly available storage capacity

- Must be capable of multiple replications to support the “two copies in the same rack, one copy in another” principle, and to address the Bathtub curve penalty seen with bulk drive purchases[7] i.e. multi-drive/node failures
- Should support high-latency asynchronous replication to enable multi-site deployments

- **No dependence on strict RAID implementations**
  - Must not employ proprietary on-disk storage - Hardware RAID tends to employ proprietary on-disk storage and thus the volume or, indeed, any individual disk becomes unrecoverable when the RAID set is broken
  - Must enable good throughput on commodity equipment - Software RAID tends to bring about poor performance without accelerated hardware features
- **Recovery of replicated data should not be location specific**
  - Should be able to recover a failed node at one site either at its home site or at any other site with that data (i.e. being able to transport an empty or out-of-sync node to another location for high speed data recovery is highly desirable)
- **No need for manual maintenance**
  - All maintenance tasks should be exceptional and ideally limited to Moves, Adds and Changes
  - All sub-components that require routine maintenance should be automated where-ever practicable
- **No lost data**
  - Data degradation (aka Bit Rot[8]) will be automatically detected and corrected
  - When configuring the NAS it should not be possible to lose or destroy data without asserting a conscious administrative decision to do so
- **Highly accessible**
  - Locally (LAN) presented storage capacity should be accessible via native client interfaces (like SMB or NFS), though client-side agents are acceptable
  - Remotely (WAN) presented storage capacity should be accessible via API-friendly interfaces (like REST based object storage)
- **Confidential**
  - All stored data (including metadata) will be encrypted at rest, to mitigate physical compromise (theft)
  - All data in transit should be encrypted, to mitigate local network compromise (MitM)
  - Private mount points from the common storage pool must be authenticated

2.1.3 Content Use and Test Cases

The following use cases have been defined to help ensure that Project Saturn's development achieves its functional targets:

- **Non-Shared Content Model**
  - Data that is globally distributed but optionally not mounted (not presented locally)
  - The presented file-system is stackable with other modules, such as the Crypto loop

7 http://en.wikipedia.org/wiki/Bathtub_curve
8 https://en.wikipedia.org/wiki/Data_degradation

- **Shared Content Model (per above, plus)**
  - Multiple mounts (many simultaneous users)
  - Global name space
  - One user's content should not be delete-able by another's
- **Common Content Model**
  - Local access of remote content should create a local copy (a “transfer once” policy)
  - A user should be able to know if he is the person who is about to delete the last copy of something

In addition to the above, the following test cases have been defined to help validate the technical architecture in the context of the functional targets:

- Saturn will pass the logical architecture tests if it can facilitate;
  - One device with one disk
  - One device with many disks
  - One device locally with one device elsewhere
  - One device locally and two devices elsewhere
  - Two devices locally and one device elsewhere

2.1.4 Abandoned Objectives

Saturn was originally intended to support common LAN file sharing protocols (including SMB/CIFS, NFS and AoE/ATA over Ethernet) with a scalable storage capacity (minimum, one terabyte of storage), and at a LAN access rate of no less than one gigabit per second (1Gbps). There is currently no intention to provide AoE (ATA over Ethernet) support in any future version of Saturn.

We have remained open to the prospect that if no one DFS (Distributed File System) solution would meet our requirements that we would be prepared to consider integration our own replication solution (for example, an rsync or torrent solution) into the back-end to create an automated replication overlay. To-date the distribution aspect of the architecture has not been completely resolved.

2.1.5 Key Limitations

Saturn is undergoing constant development. The following key limitations should be noted:

- Mirroring has been accepted for this generation as there was no alternative.
- A Filer Head will be used to provide above-based features, including:
  - Encryption in transit, which will need to be provided by the Filer Head to minimise device load, and;
  - Both native (NFS) and API (Object) interfaces will need to be provided by the Filer Head to add these capabilities over the top of a client-side agent on the Filer Head
- There remains no “multi-master” solution so the above, 2.1.3 Content Use and Test Cases, are largely moot as these were intended to force out the multi-master capability (note the shared and common content references).

2.2 History of the Project

Project Saturn was always intended to be a multi-terabyte NAS device that was compatible with both Linux and Windows desktop and server platforms, but it has been a long journey.

2.2.1 Pan (2006 – 2019)

The first generation of Saturn was ambitious – it was seeking to provide a consumer grade NAS with enterprise features, built from open source software. This was at a time when SATA was new, 200 - 300GB drives were “big drives” and started before GlusterFS was acquired by RedHat.

The project succeeded, but not to the level desired.

Development time was hampered by full-time roles in Enterprise IT, house moves, kids, divorce, etc. By the end of 2019 there were multiple Generation 1 Saturn deployments and they were in good service, providing useful functionality. The fact that this was achieved to any degree, let alone to provide a decade of reliable service, will always be celebrated as a success story.

Figure 2: Saturn Generation 1 Deployment – Looking Sharp

The problem was that, in practice, Gen1 Saturn deployments were singular monolithic instances that were constrained by chassis drive bays, SATA ports and power supplies. In many ways this was the antithesis of the project’s design goals.

The root cause was a combination of hardware cost and the "heavy touch" approach (a software architecture that was driving a view of static deployment) within the GlusterFS software which made the dynamism of a light-weight distribution and joining/parting nodes very difficult to manage. The result was to avoid networked builds, which then drove monolithic instances and ultimately killed off any form of distributed file-system.

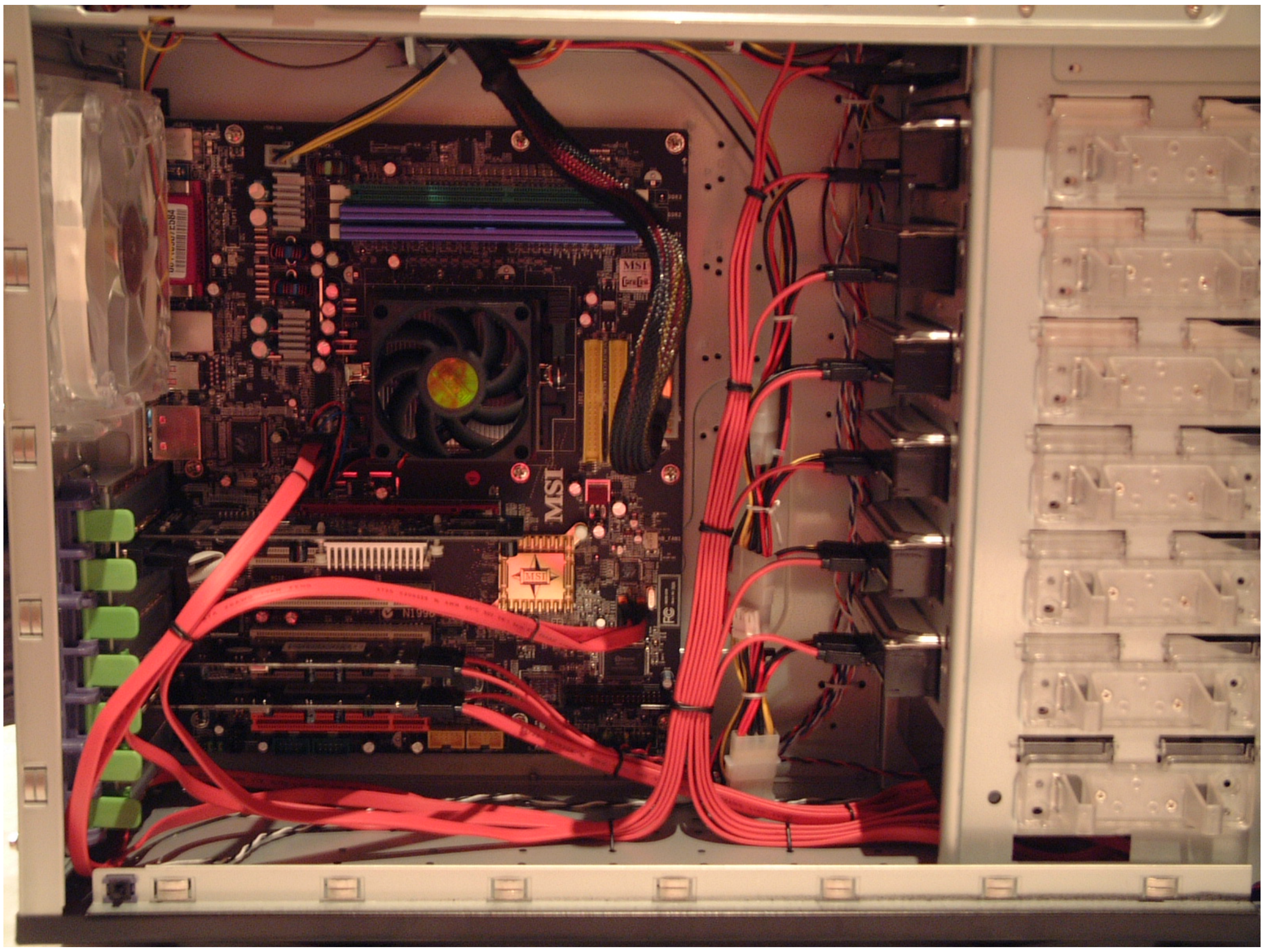


Figure 3: Saturn Generation 1 Deployment – A Monolithic Deployment, Almost Full

GlusterFS had served the project well, but not without its pain-points. Key to the first generation:

- **Drives were balanced by Gigabyte and not by Percentage**. This meant that a deployment of drives of different sizes would have issues as the smaller drives filled.
- **Re-balancing was a manual task**. So adding a new drive, or replacing an old one, would mean a series of manual steps required to shift data.
- **Loss of the first drive would lose all metadata**. The presented file-system would *disappear* if the first disk failed, as it seemed to be primary for the metadata (file-system structure).
- **Moving directories would occasionally fail**. When moving a directory full of files at the presented file-system level, there was an occasional glitch that would result in files being lost/stranded.
- **Files could end up in a state where they were unmanageable**. At the presented file-system layer a file that entered this type of *untracked* state could be seen but neither be accessed nor deleted.

The GlusterFS project has evolved past these pain-points, but the embedded version didn't change, so they remained with Saturn to the end of Gen1. The huge advantage that GlusterFS gave Saturn was that in all cases other than hardware failure, the data was recoverable by accessing the raw EXT4 partition under the concatenated GlusterFS volume.

2006

Saturn had been on the cards for quite a while, though a lack of available time had held-back initial in-roads into the project. To motivate myself I ordered hardware for the project on January 11, 2006, which meant that coding had to start! At that time, utilising the best price-to-megabyte ratio for IDE/SATA hard disk drives (320GB SATA I disks), Saturn would provide roughly two terabytes of storage for about US$1,000 (valued as at Q1, 2006).

It was obvious that a number of projects being developed at Midnight Code (i.e. the "Planet Series") would benefit from a single integrated architecture that was able to provide a common, extensible and lightweight embedded Linux distribution to all projects thereby removing re-work that would

have been performed for each. And with the success of the CHAOS Linux distribution[9], a number of lessons had been learnt regarding the internal ad-hoc management architecture that was used to deploy and regulate the Operating System internals.

Thus a new architecture was conceived to retain the benefits and build on the learnings of the CHAOS experience. This architecture was encapsulated in a framework called libMidnightCode[10] which slowly evolved from fundamentals such as text manipulation and socket management (in 2005) to holistic functions such as message brokering, interface (NIC) management and configuration management. Both Project Saturn (http://midnightcode.org/projects/saturn/) and Project ATS (http://midnightcode.org/projects/ats/) were the first Midnight Code projects to employ the incomplete libMidnightCode software, which in turn drove requirements back down into libMidnightCode.

By the end of January that year;

- libMidnightCode had been extended to support automatic detection of the first 16 IDE devices and the first 16 SCSI devices available under a Linux 2.6 kernel (16 being an arbitrary limit). Discovery information included the Vendor, Product, Revision, Serial Number and Capacity for each drive.
- libMidnightCode was also extended to support configuration file writing (as well as configuration file reading, which it already supported). This should allow the device to manipulate its own configuration file, leaving the user free to manage it via HTTP
- A proof-of-concept build (the "reference platform") was made from the Linux 2.4 kernel combined with a libMidnightCode capable of low level disk and interface management along with the corresponding configuration file controls, and bundled into a USB key booting image.
- Development effort was then focused on getting libMidnightCode to interface with a HTTP user environment so that management could be enabled from user to system.

By using libMidnightCode, the reference platform immediately took on the lightweight nature of CHAOS. Even in its experimental form was still a tiny distribution (about 12Mbytes in size) and it was able to utilise USB mass-storage hardware for its embedded OS (i.e. it could boot from a USB key) guaranteeing that no disk storage capacity would be wasted to the host OS – every disk in the resultant NAS would be shared storage – for a Saturn appliance built from the reference platform.

Unfortunately, there was limited hardware support for SATA controllers in the 2.4 kernel series, so the reference platform failed to boot the then Project Saturn hardware.

By the end of October, version 1.2 of libMidnightCode had been integrated with a proof of concept Linux 2.6 kernel based reference platform and been used to USB boot the Project Saturn hardware to multi-user state. Testing had shown that the proof of concept platform with its integrated libMidnightCode software (particularly with improvements to the init process) was able to detect all of the drives in the system, be user configured via static config file, and also fall back to a default known-good state in case of a missing config file; something CHAOS has previously been able to do.

Project Saturn was well on its way.

9 http://midnightcode.org/projects/chaos/
10 http://midnightcode.org/projects/libmidnightcode/

2007 – 2008

By March 2007 the high level web interface (webint) library framework was in place, but was read-only (not usable as an interactive administration interface).

Unfortunately for Saturn, 2007 saw all of my project time consumed by Project ATS. The advances in libMidnightCode therefore favoured physical control and interface systems;

- In March, libMidnightCode saw improvements in the socket library, and the addition of Video4Linux UVC camera support.
- By June, the Casting protocol had been implemented, allowing for the future creation of a very simple, yet significantly more robust, Tyd replacement for CHAOS - dreamt of since the socket library code of August 2006. libMidnightCode version 1.4 also includes a new Servo motor control library and GPS library (for gpsd interaction), amongst a number of minor tweaks such as a child socket finder, safe list iteration, etc ad nausea.
- By July 2007, the Bootp/DHCP protocol had been implemented, facilitating integrated interface address discovery. A Syslog client library had also been added to allow for integrated system logging. A lot of debugging went into finding an issue with some Project ATS equipment (to do with global variables in the library that were assigned static values which were not being shared by the application that was using the library - thus there was a pseudo-random dependency on system memory being zeroed).

The creation of a Bootp/DHCP client removed the last dependency for the boot prompt options in the original CHAOS image and meant that the reference platform no longer needed boot parameters to configure the booted operating environment. The reference platform was running kernel 2.6.22.

In 2008 I started a new role and my night time project work was replaced with late nights in the office. People were becoming interested in Saturn, they were all starting to see the same personal storage problems (scale and availability issues) and realised that the off-the-shelf solutions were very poor fits for their problem. But there was nothing of Saturn that could even be played with – apart from standalone concept demonstrators and a boot-able reference platform that essentially provided you with a web server and a shell prompt from bare metal.

Although it had a firm base and broad interest, Project Saturn was well and truly stalled.

2009 – 2010

The year 2009 was another great year for Saturn;

- In February I was mobbed – people who had heard about Saturn wanted it – they also wanted it to solve problems that hadn't been anticipated because they started comparing Saturn's original objectives to enterprise solutions and thought that the gaps could be closed. With the additional view-points a hefty debate ran to evolve Saturn's requirements from the various commodity and enterprise experiences the group had.
- One of the most persistent views was that of the recently publicised Google File System (GFS) used internally to Google[11] which seemed to feature most of the benefits that were being sought, and certainly drove the availability view of “2 copies in the same rack, 1 copy in another”.
- At the same time, a comprehensive review was done of all of the available distributed file systems. We went to prototype on a couple of these;
    - We prototyped the MySQLFS but found that it was disappointingly slow on our 50GB test volume
    - We found that Hadoop had been modelled off the Google File System principles by Yahoo[12], however when we went to prototype Hadoop we learned that it was written in JAVA and that there was no open way to distribute the JAVA runtime commercially. It was possible to compile Hadoop using “gcj” but only up to version 0.4.0 (we had tested Hadoop versions 0.4.0, 0.5.0, 0.6.2 and 0.19.1 – the latest at that time).

11 http://web.archive.org/web/20090220151747/http://labs.google.com/papers/gfs.html
12 http://en.wikipedia.org/wiki/Hadoop#Architecture

- By June we had decided on GlusterFS as the Open Source technology that would provide the storage feature-set that most closely matched our requirements, developing configurations for Distributed and Replicated storage in GlusterFS version 2.02.
- And, in July, GlusterFS was introduced to the reference platform to create a distinct “Project Saturn” image.
- In August I ran a thought experiment with Chris to see if we could get a “de-dupe” layer into GlusterFS for Saturn and spent two weeks trying to implement the straw-man before I fell ill.
- At some stage in the 2009 development season the Bootp/DHCP client code had been broken.

Unfortunately in September 2009 I was assigned to an urgent piece of work that ran for a month and and then a large Program of work that ate into my night-times again, for most of the next two years.

By March 2010, GlusterFS 3.0.3 had been released changing the architecture of the GlusterFS software and disabling one of the core features we had successfully deployed – the replication of an irregularly sized distributed volume. Instead GlusterFS required a distributed/replicated volume to be consistent of replicated bricks (matched disks on alternate hosts) that were then added to distribute volumes.

Project Saturn had seen a great run but was stalled again.

2011 – 2012

After I moved house, in October 2011, the BIOS battery had to be replaced in the Project Saturn hardware – a clear indication of both the age of the project and the lack of recent time investment.

Project Saturn had its third wave in 2012 when I took some time off which ended with both Michael and I sharing a week of holiday hours on Saturn;

- We initially persisted with GlusterFS 3.2.6 because we were so entrenched in the storage architecture that it enabled.
- A number of patches were created and submitted to the Gluster project identifying and resolving 32bit issues in GlusterFS 3.2.6, amongst other features sought for Saturn
- The aged Project Saturn hardware was refreshed - Four 2TB drives were added (replacing 4 x 320GB drives) along with 32GB if RAM to create a 7.2TB volume under a GlusterFS 3.2.5 and with Linux Kernel 3.2.14.
- The move to GlusterFS 3.2.5 also created a number of embedding issues due to Gluster managing its state via extended attributes in the root directory of the managed file system
- libMidnightCode 1.6 was extended to manage GlusterFS and remove the need for custom shell scripts
    - A month later the Project Saturn hardware was increased to five drives on GlusterFS 3.3.0 (replacing another 320GB drive)
    - A portmapper was added to the Saturn image to enable Gluster's native NFS daemon
- Two months later another a sixth drive was added to the first volume and a second volume of two 3TB nodes was added – driving out a number of 32bit issues in libMidnightCode.
- The user interface (HTTP Administrative interface that was still read-only) was expanded to show all storage configuration information

By Q3 2012 there were two production Saturn nodes (including the refreshed Project Saturn hardware) with combined total of around 30TB of distributed NAS-presented storage and a number of development nodes in existence.

Although it wasn't complete, Project Saturn was finally good enough to run in production.

2013

In early 2013 the two production Saturn nodes had hit their first ceiling – there was insufficient chassis space to add further physical drives, and the capacity projections showed that there would be insufficient logical space to store additional content in about 3 months. This lead to a piece of work on the embedding of GlusterFS's clustering capability, including circumvention of the additional state controls in Gluster software.

Due to the increasing interest in a (finally) usable Saturn, Michael and I were beginning to see the problems that was going to come with trying to explain Project Saturn, the Saturn software and the tips and tricks that we'd developed from our experience in building production Saturn environments. So I developed the first Saturn Manual in May.

Thus, at the time of writing, Saturn is capable of presenting a volume of capacity that is created from both the local and remote storage capacity of multiple Saturn nodes. The first firmware version was published 26 May 2013 (midnightcode-saturn-0.1-20130526230106-usb.burn.gz).

Later this year both production sites will add a second node each and cluster to additional capacity and we are anticipating another production site and a couple of new experimental sites.

2015

A number of improvements were bundled, including the ability to run local startup scripts, and published in an updated firmware on 14 September 2015 (midnightcode-saturn-0.1-20150914222409-usb.burn.gz). The bootp/dhcpc code that had been written for libMidnightCode was never recovered – so at its end Saturn still lacked dynamic IP support, despite having it in 2007.

There were no further updates released for the first generation Saturn platform.

2019

At the end of its service life, the largest Saturn instances were running 8TB SATA III drives and presenting 40TB+ to their respective networks.

2.2.2 Daphnis (2019 –)

In November 2019 the parts were acquired to develop a prototype of the next generation of Saturn. The goal of Gen2 was to embed the distributed file-system into the heart of the solution (ideally one disk per processor enabled storage node, and no more than two), and to make the solution fan-less (passively cooled) if possible.

The remainder of this manual describes the construction and operation of the second generation Saturn.

2019, December – Prototype 1

Over the Christmas break in 2019, the first prototype was assembled.

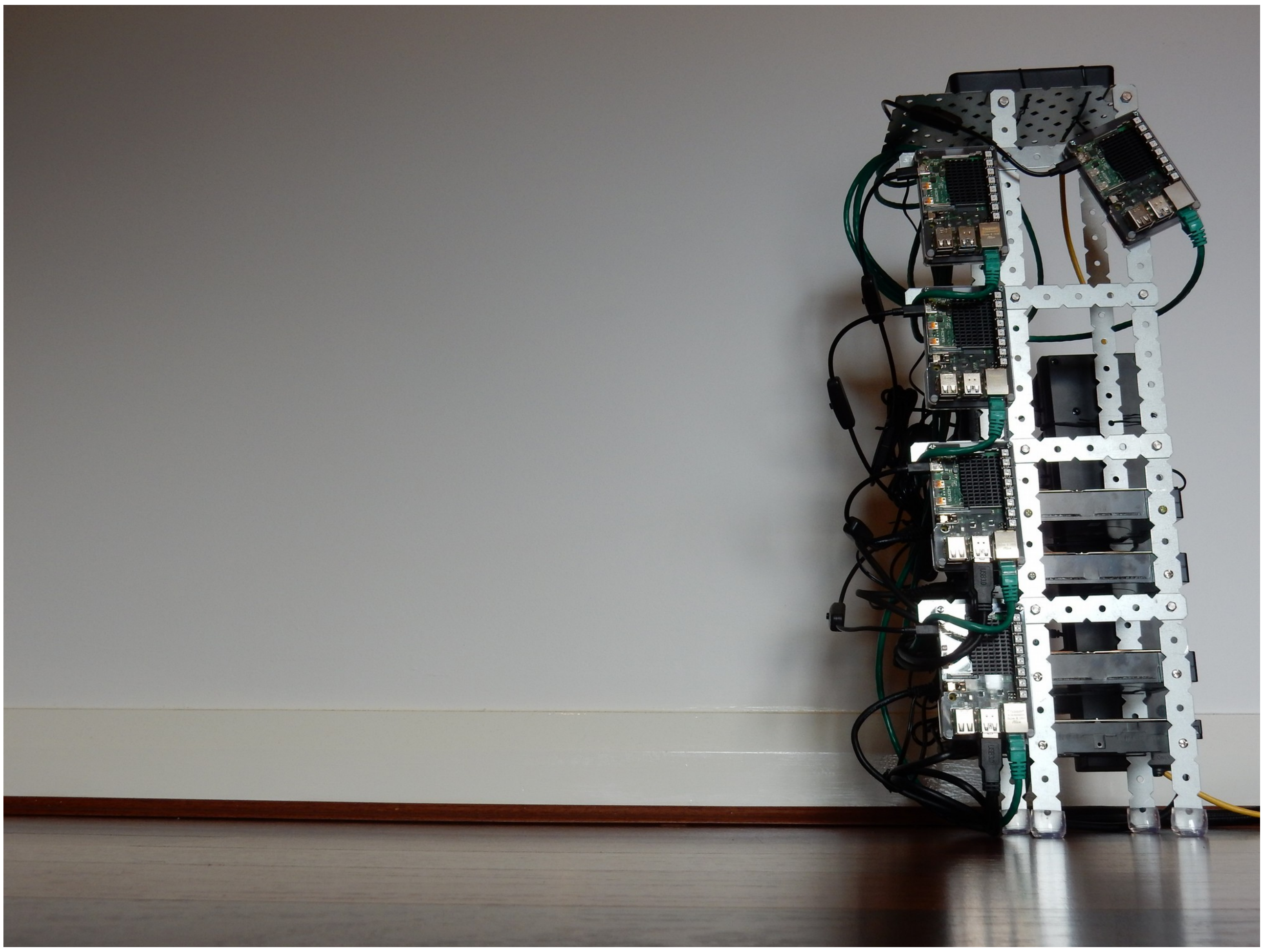


Figure 4: Saturn Generation 2 Prototype 1 – with four (4) Storage Nodes and four (4) of eight (8) Drives

From a hardware perspective the principle idea was to leverage the recently released Raspberry Pi version 4, with two drives per storage node. The Pi4 has 4 CPU cores, and each storage node had 1GB of RAM.

As the Pi4 doesn't have an off-the-shelf 2 x 3.5" HDD based chassis, a frame was created that leveraged the disks as the structural components. It was designed to be a repeating pattern that could be built as small as a singe (1) node of two (2) drives or as large as depicted, with capacity for four (4) nodes and eight (8) drives.

The frame was built to ensure maximal airflow in the hope that the resultant deployment, even fully populated, would cool passively in an enclosed room without air conditioning.

This prototype was fitted out with 8TB drives, for testing.

The Pi4 also lacks SATA ports, so the prototype was constructed with externally powered USB3 SATA adaptors.

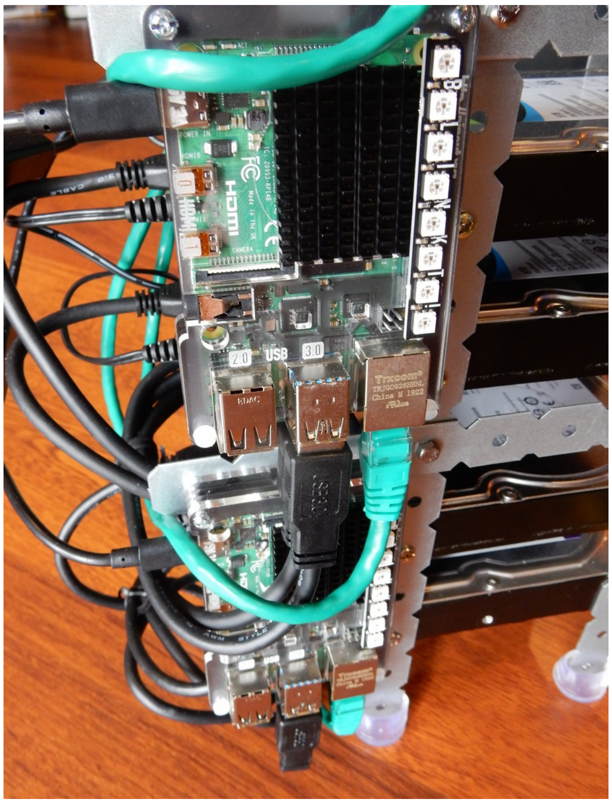


Figure 5: Saturn Gen2 Prototype 1 – two (2) Storage Nodes

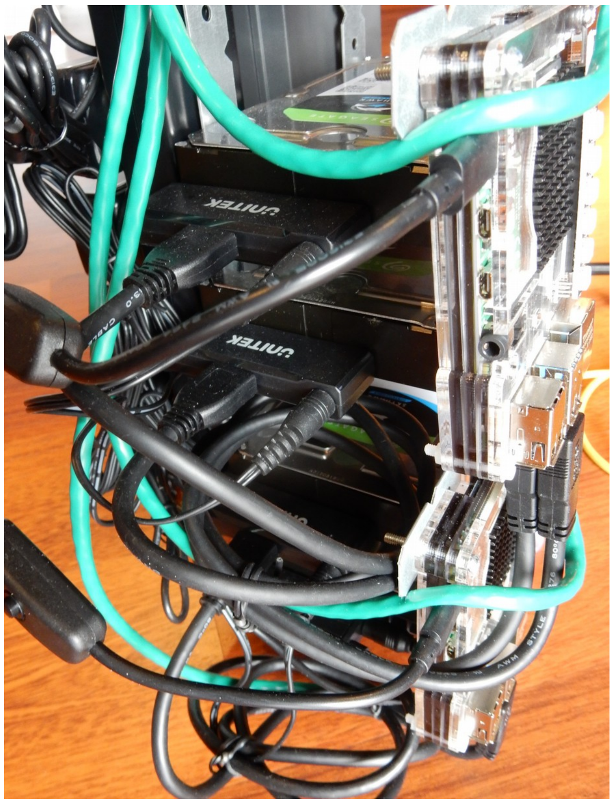


Figure 6: Saturn Gen2 Prototype 1 – USB3 SATA III

From a software perspective, this prototype was built upon the open source distributed file-system LizardFS. In addition to supporting the ARM CPU architecture, LizardFS was chosen over its predecessor (MooseFS) as it had support for Erasure Coding (i.e. RAIDZ).

LizardFS leverages a Master node for the file-system metadata, hence the fifth Pi4 which was the Master node – with 4GB of RAM.

Prototype 1 was a failure. There were a number of concerns that arose out of this prototype:

- The presented LizardFS share was only able to transfer files (write) at about 15MBps (120Mbps), well under the rates experienced in Gen1 Saturn builds.
- While it was largely assumed that the poor performance was related to the USB adaptors, there was concern with the CPU limits of the Pi4 when working with whole disk encryption.
- The performance issue didn’t matter as the need for three “wall wart” power plugs per storage node wasn’t practical, so a better hardware solution was needed anyway. Even the use of a single powered USB3 Hub for the two 3.5” hard drives wasn’t a viable alternative (the power for the drive needed to come from the non-USB source).

What was needed was a more powerful single board computer, one with a native SATA III interface, and which was capable of powering large capacity 3.5” drives.

2020, April – Prototype 2

The Odroid HC2 units[13] were ordered in January, just before nCov-19 hit. So when work contracts were terminated and Government Health Orders forbid outside activity, a three month window opened up to finish the next generation of Saturn.

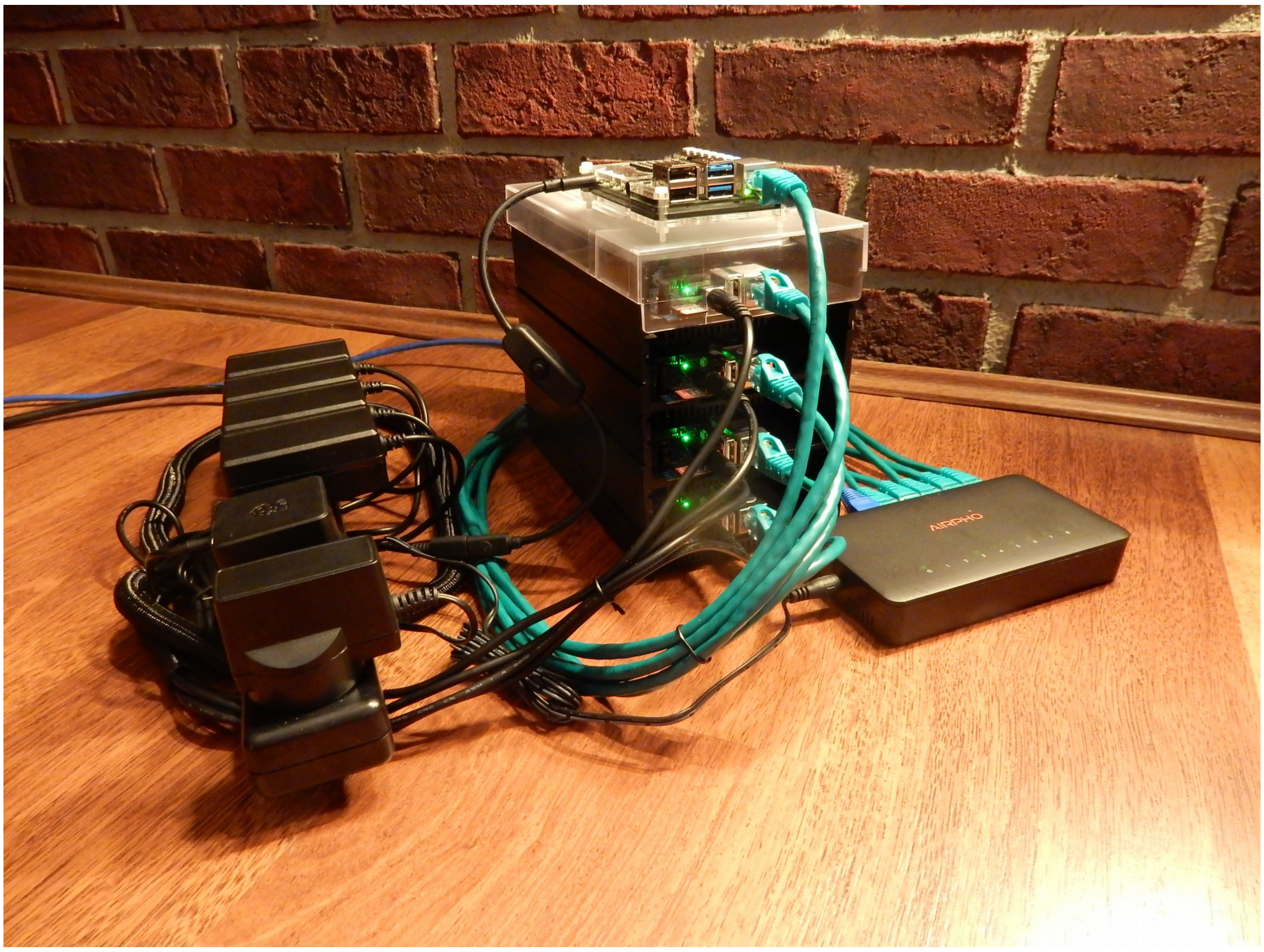


Figure 7: Saturn Generation 2 Prototype 2 – with four (4) Storage Nodes and four (4) Drives

From a hardware perspective, the HC2 has 8 CPU cores and 2GB of RAM. It has a native SATA III interface, and is designed to power the attached hard drive from its own 12v 2A plug. The same four (4) 8TB drives that were used in Prototype 1 were redeployed to Prototype 2. A 4GB Pi4 was retained as the Master node.

Weeks were spent in cycles of building, performance testing, tuning and re-testing, before re-building.

There were a number of hard lessons learnt from this process:

1. **Performance** – better hardware enabled focused diagnostics
   a) While the 8TB drives supported r/w performance of about 230MBps, and the HC2 supported line rate gigabit ethernet at 125MBps, with LizardFS this prototype could write at a maximum of 55MBps (440Mbps), and read at a maximum of 46MBps (368Mbps). Even noting that a distributed storage fabric writing synchronously can't exceed the performance of any one node, these results weren't acceptable compared to the Gen1 Saturn experience.
   b) Replacing LizardFS with MooseFS immediately provided write speeds of 62MBps (496Mbps) and read speeds of 98MBps (784Mbps), before performing any Moose-specific tuning – which then brought write speeds up to 80MBps (640Mbps).

13 https://www.hardkernel.com/shop/odroid-hc2-home-cloud-two/

2. **Heat** – increased write speeds, on full-disk encryption, introduced the heat problem
    a) With the HC2's physically deployed in the depicted "stack", with a cover on the top storage node, the top node eventually failed (power-cycled itself, with no syslog entries indicating why). It was assumed that the clear plastic cover on the top node was acting as a blanket so it was removed.
    b) Still in the depicted "stack", but now without a cover on the top storage node, performance was pushed and again the top node eventually failed (power-cycling without a log entry). It was assumed that the heat was accruing through the metal of the stacked devices, and the hottest HC2 was therefore the top device.
    c) The HC2s were spread out across the floor such that each storage node was independent of the others. The clear plastic cover was re-installed on the first node. As performance was pushed the covered node failed. At this point scripts were used to track the five thermal zones on each HC2, as well as the hard drive temperature.
    d) Based on the kernel source for the HC2 and the Seagate drive specifications, alarms were set at 93ºC for the five HC2 thermal zones (on the assumption that the power cycling occurred in the 99-109ºC ball-park), and 63ºC for the hard drive (with a 70ºC operating limit). Both alarms were triggered frequently while doing large file transfers to the MooseFS enabled storage array – when in a stack. When not stacked the HC2 thermal zone alarms were triggered frequently, but the hard drive alarm wasn't – confirming that the heat was CPU driven, with the high performance encryption.
3. **Vibration** – with test cycles running longer, the prototype was moved from the floor
    a) Having confirmed that the heat problem was with the CPU, an attempt was made to improve airflow under the CPU (particularly given the HC2 chassis is a heatsink with fins on the under side). The HC2s were put on a bookshelf with the CPU area over-hanging the edge of the shelf, to allow air circulation, using the mass of the hard drive to keep the HC2 comfortably stable on the shelf. This seemed to reduce thermal zone temperatures by between 5 and 10ºC.
    b) After one of the HC2s fell off the metre-high shelf days into a week-long test cycle, it was confirmed that the mass of the "hanging" power and ethernet cables combined with the drive vibration resulted in a "guided walk" that drove the HC2 off the shelf, and destroyed the hard drive that it contained (though notably, not the HC2 itself).
    c) Keeping the HC2 fully shelved (and anchored with a 4mm lip) meant having to find an alternative passive cooling solution. The solution was to set the two CPU scaling max frequency values to 1Ghz for all eight cores (down from 1.5Ghz for the first four cores and 2Ghz for the second four cores). This took approximately 15ºC off the thermal zone peak temperatures and only reduced the write performance by an average of 5MBps (40Mbps) – now at about 75MBps (600Mbps).

With the write performance now consistent with the Gen1 Saturn instances, it was ready to scale.

2020, June – Daphnis

The first Daphnis build is comprised of ten (10) storage nodes – six (6) x 16TB drives plus four (4) x 8TB drives, deployed in an A/B configuration to take advantage of local high availability (two copies of all files on the local site) as well as predictable grouping (meaning that the "A" devices could be plumbed to one UPS and the "B" devices plumbed to another). There is no Pi4 in this generation, making the infrastructure layer homogeneous and highly scalable.

The remainder of this document describes the build and operational steps for the final version.

2.3 With Many Thanks

Midnight Code thanks Michael Welton for his massive time investment in the theory, specification and testing of Saturn (both generations).

Michael first embraced the intent of the project in 2009 when he saw its potential, and its shortcomings. He then helped drive much of the thinking that shaped the Saturn we have today including some of the key enterprise characteristics that we have come to depend on. I could only guess at the hundreds of hours that Michael has contributed in nights, weekends, and even through holiday stretches – across cafes, dining rooms, lounge room floors and the occasional cramped office – working through requirements, proposals, architectures and then tirelessly building and testing Saturn implementations (How many USB keys did we buy in the end?). Through the heated debates about theoretical use cases to the vast data losses of failed early builds, Michael has persisted with the project and has driven it with a genuine desire to achieve its ambitious goals. Champion.

Midnight Code also sends its thanks to Chris Kelada for his review of the revitalised Saturn proposal in 2009, as well as his contribution to one of my most favourite coffee-table thought experiments – Why can't we have an "enterprise" style open source de-dupe? How hard could it possibly be!? The resultant straw-man became the Midnight Code GlusterFS De-duplication Xlator – still incomplete (bogged down in the Gluster/FUSE internals due to my coding skills and availability) at the time of writing – but very achievable thanks to those two hours of coffee table time one winter's eve in 2009.

2.4 About Midnight Code

Midnight Code is a singular resource created by Ian Latter to house and share the most useful open source software that he has developed.

These programs are the publicly publishable, cumulative and structured outputs of the seemingly ceaseless need to create that stems from the author himself. Some of the projects have a long meandering history that is due to their unique evolution, while others have been created simply to fit a niche need. The works that have been developed by the author under contract for commercial organisations are not public, and hence have not been published here. Though public works by the author, as published here, have been used to develop private (commercial) software and appliances.

The main project grouping is "The Planet Series" project set. This series of projects is designed to bring Linux to life, in the Home or Office, to fulfil the complete spectrum of communications and life-style technologies for all non-enterprise consumers. These projects start at Mercury with the development environment required to get you started, and end at Pluto with connectivity from your LAN to the rest of the universe.

Each project is clearly defined, and contains screen shots, documentation, source code, links and activity information, as identified.

3 Installation

In this chapter we look at the hardware and software installation of a Saturn deployment. If you already have spare/available hardware then you can skip straight down to the software installation.

3.1 Considerations, before you begin

Before commencing a build/deployment, consider the following to ensure that your Saturn deployment is right for your needs.

Service Qualities

Ideally you should consider the service qualities[14] for the storage facility that you are designing:

- **Availability** (the degree to which something is available for use), including:
    - **Manageability**, the ability to gather information about the state of something and to control it
    - **Serviceability**, the ability to identify problems and take corrective action, such as to repair or upgrade a component in a running system
    - **Performance**, the ability of a component to perform its tasks in an appropriate time
    - **Reliability**, or resistance to failure
    - **Recoverability**, or the ability to restore a system to a working state after an interruption
    - **Locatability**, the ability of a system to be found when needed
- **Assurance**, including:
    - **Security**, or the protection of information from unauthorized access
    - **Integrity**, or the assurance that data has not been corrupted
    - **Credibility**, or the level of trust in the integrity of the system and its data
- **Usability**, or ease-of-operation by users, including:
    - **International Operation**, including multi-lingual and multi-cultural abilities
- **Adaptability**, including:
    - **Interoperability**, whether within or outside the organisation (for instance, interoperability of calendaring or scheduling functions may be key to the usefulness of a system)
    - **Scalability**, the ability of a component to grow or shrink its performance or capacity appropriately to the demands of the environment in which it operates
    - **Portability**, of data, people, applications, and components
    - **Extensibility**, or the ability to accept new functionality
    - The ability to offer access to services in new paradigms such as object-orientation

You must be aware of the existence of these qualities and the extent of their influence on the choice of building blocks used in implementing the solution. Before commencing your build, it is recommended to create a quality matrix, describing the relationships between each functional service and the qualities that influence it.

Whatever your priorities are, be true to your requirements and build accordingly. This isn't wasted work – it will be the case whether you choose to use Saturn or another storage technology.

14 http://pubs.opengroup.org/architecture/togaf8-doc/arch/chap19.html

3.2 Hardware

For Gen2, the Saturn hardware architecture is a single node type that is horizontally scaled. It is exceedingly simple, and individual nodes are shipped off the shelf and straight to your door ready to run.

3.2.1 Single Board Computer (SBC)

The Odroid HC2[15] is a fantastic little board that has been designed to attach a SATA3 drive to a gigabit network.

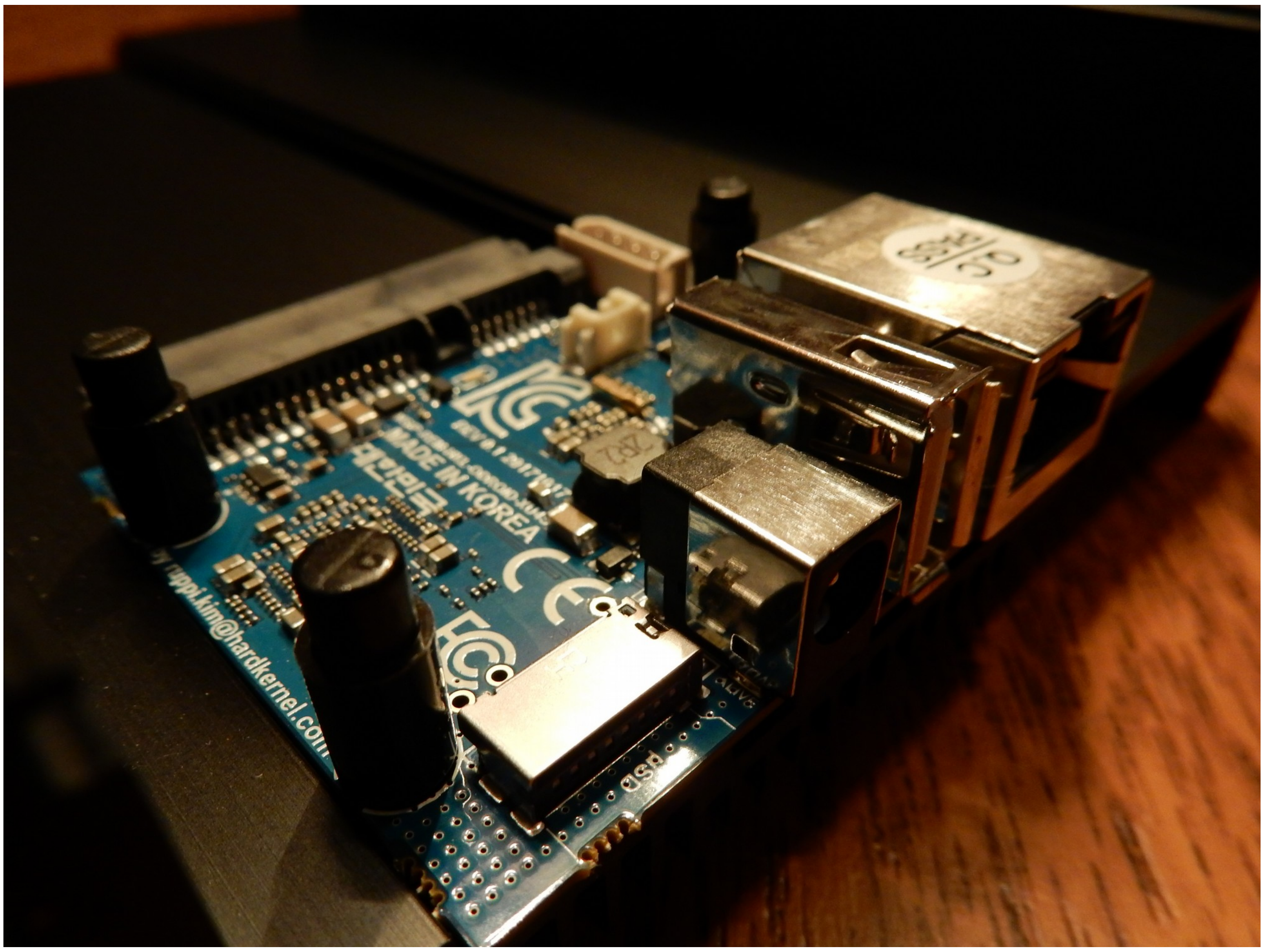


Figure 8: Odroid HC2 – 8 CPU Cores, 2GB RAM, SATA3, 1GB NIC

At half the size of a credit card this little SBC is packed with just the right tech:

- 8 x CPU Cores (Exynos 5422, ARMv7 rev 3)
  - 4 x Cortex-A7 at 1.5Ghz (cpu0)
  - 4 x Cortex-A15 at 2Ghz (cpu4)
- 2GB RAM (DDR3)
- 1 x SATA III (for 2.5/3.5inch HDD/SSD)
- 1 x GB NIC
- 1 x USB 2.0 Host Port
- 1 x Micro SD card slot (UHS-1 capable)

15 https://www.hardkernel.com/shop/odroid-hc2-home-cloud-two/

3.2.2 Enclosure and Thermal Considerations

The Odroid HC2 is shipped ready-mounted to an aluminium heat sink that doubles as a 3.5 inch drive bay.

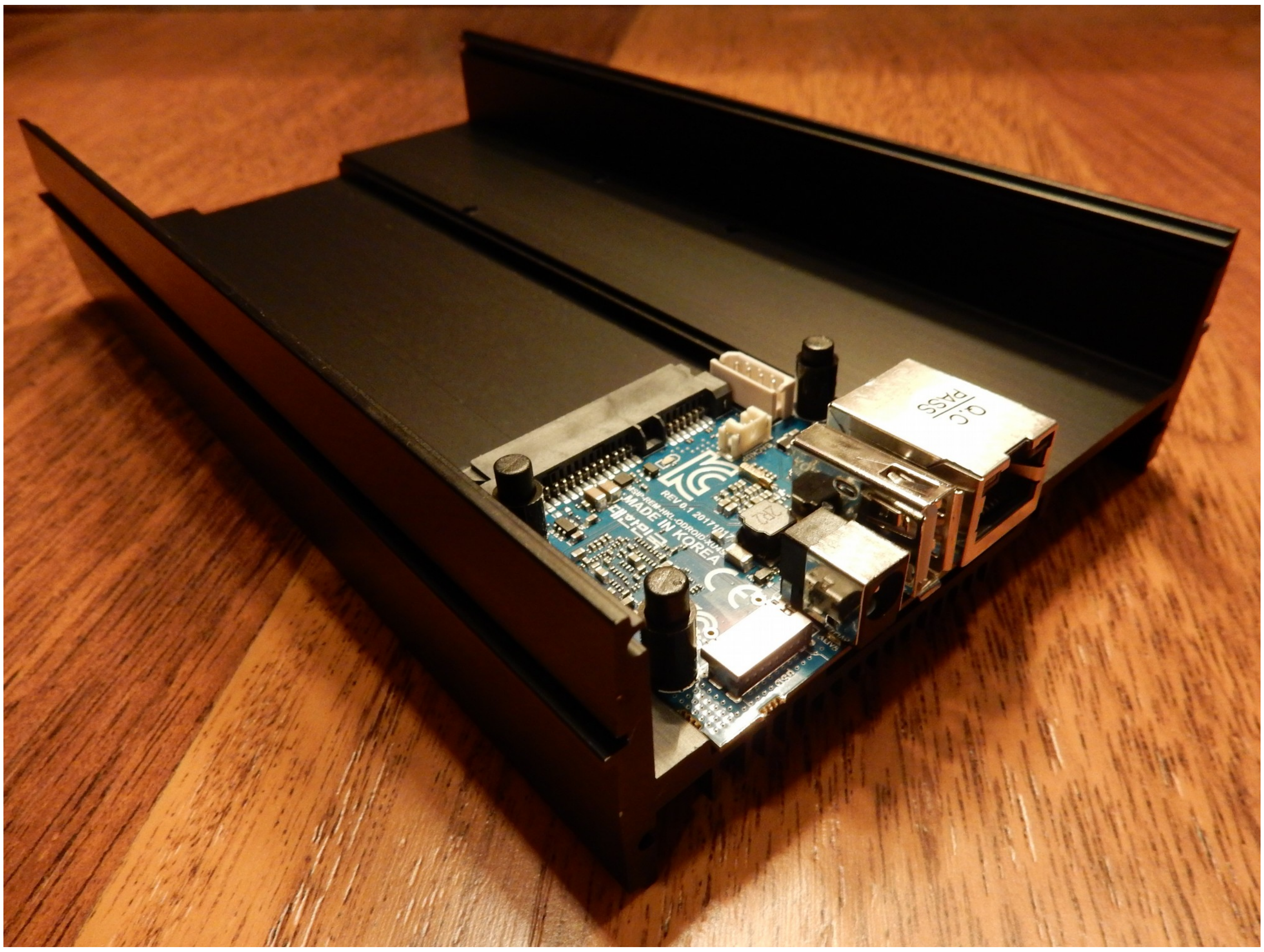


Figure 9: Odroid HC2 – As shipped, mounted in an open-faced aluminium enclosure

The total enclosure size is approximately 197mm x 115mm x 42mm.

While the manufacturer does sell a plastic cover to complete the enclosure, it is not recommended for passively cooled installations. Given the use of encryption, at high loading (gigabit file transfers in the order of terabytes) the HC2 accrues heat and then power cycles when over-heated.

Also, despite the ability to stack the enclosures, it is not recommended to do so, for the same reason.

To provide static thermal management the software layer build includes a boot-time configuration change that will rate limit the CPU (see section 4.2.3 Thermal Management, below, for details).

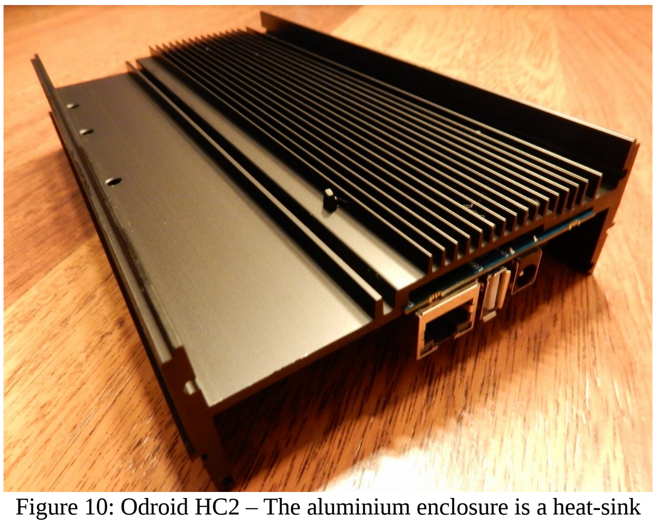


Figure 10: Odroid HC2 – The aluminium enclosure is a heat-sink

3.2.3 Power

The Odroid HC2 requires a 12v DC 2A power source. The inner diameter is 2.1mm, while the outer diameter is 5.5mm, and it is centre positive. The HC2 can consume as little as 1A but will go up to 2A with high CPU load and fully spinning 3.5inch HDD. The manufacturer sells the same AC adaptor with the pins localised for each region (Asia/Korea[16], AU[17], UK[18], US[19]). Note that using this power adaptor will require one AC outlet per node deployed.

3.2.4 Boot and Root Storage (Micro SD Card)

No operating system software is stored on the hard drive that will be attached to the node – that hard drive is dedicated entirely to the communal file-system data. That means the OS “boot and root” drive is a Micro SD card, which is used to boot the node and to store the Linux kernel and the Linux root file-system.

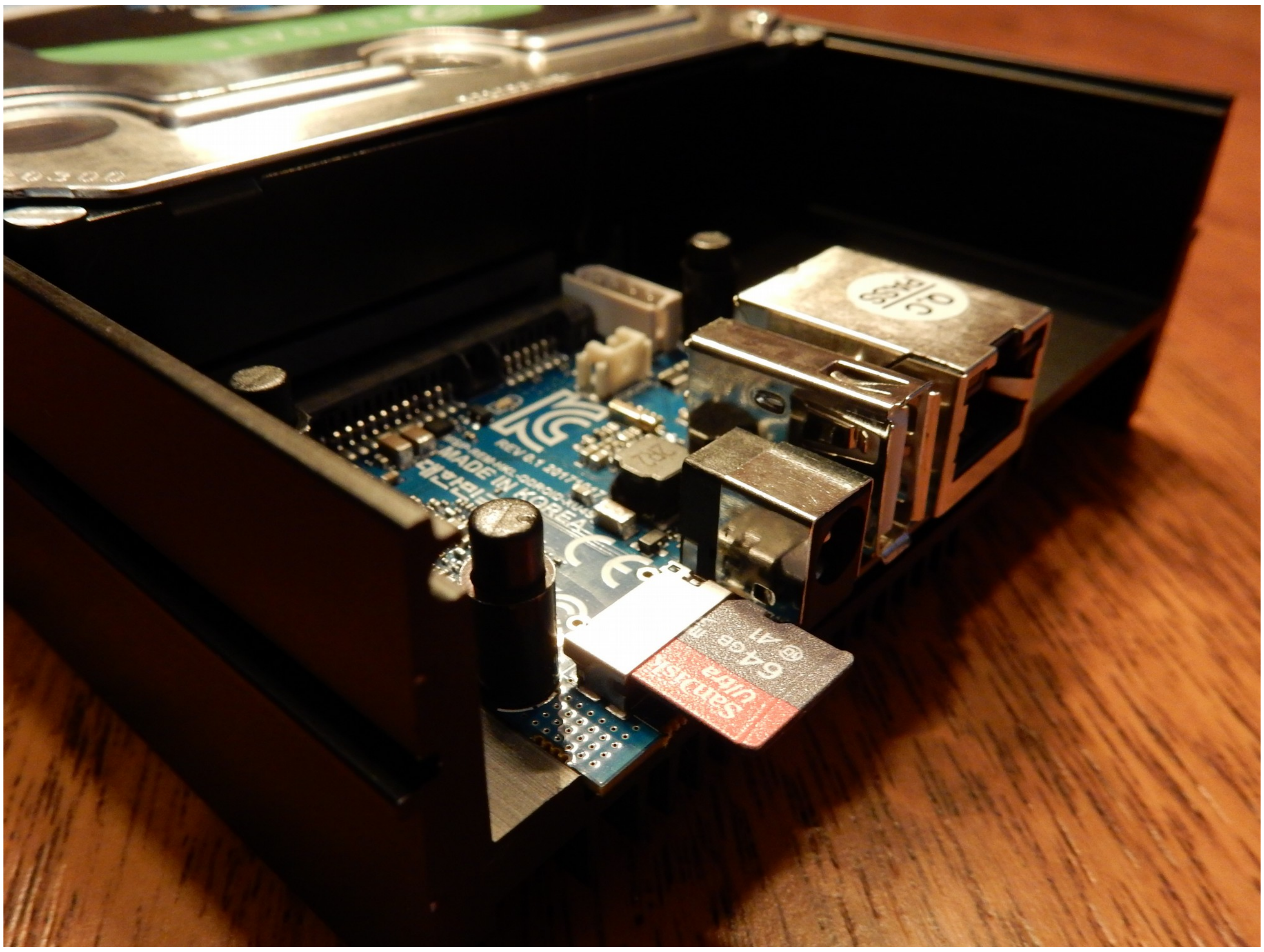


Figure 11: Odroid HC2 – Depicted with SanDisk Ultra 64GB microSDXC UHS-1

The software architecture for the Gen2 Saturn changed from the first version. Originally, metadata was going to be stored on the root file-system, and there is always concern when an embedded system uses a standard Unix syslog daemon. In both cases it’s fear of filling the file-system under certain conditions which would prevent the node from operating.

16 https://www.hardkernel.com/shop/12v-2a-power-supply-asia-korea-round-plug/
17 https://www.hardkernel.com/shop/12v-2a-power-supply-australia-plug/
18 https://www.hardkernel.com/shop/12v-2a-power-supply-uk-plug/
19 https://www.hardkernel.com/shop/12v-2a-power-supply-us-plug/

So while the larger SanDisk Ultra 64GB microSDXC UHS-1 card is depicted for (and used in) this build, it isn't required. The recommended Micro SD card is an 8GB card, so long as all data (including metadata) is stored on the data drive, and the logging is either limited in volume or, preferably, shipped to a centralised logging service (which will also help to maximise the life-span of the SD card).

3.2.5 Data Storage (Hard Disk Drive)

The Odroid HC2 will take either 2.5inch or 3.5inch SATA3 drives, however, the drives deployed will depend on the type of storage solution that you require.

For large capacity, the most cost effective storage is magnetic media. For high IOPS with some capacity, you could deploy more of the smaller capacity magnetic media with high RPMs (multiple 2-4TB drives), while for extremely high IOPS you may need to deploy flash (SATA SSD). It is also possible to deploy a blend of drive types and manage their use by policy at the software layer.

Saturn has always been about larger capacities rather than IOPS (the most common deployment type), so in this build the left over 8TB Seagate Surveillance (3.5inch, SATA3) drives from a Gen1 Saturn will be recycled, in conjunction with new 16TB Seagate Surveillance drives (ST16000VE000, 3.5inch SATA3[20]). The 16TB drives should consume less power and make less noise than its predecessors.

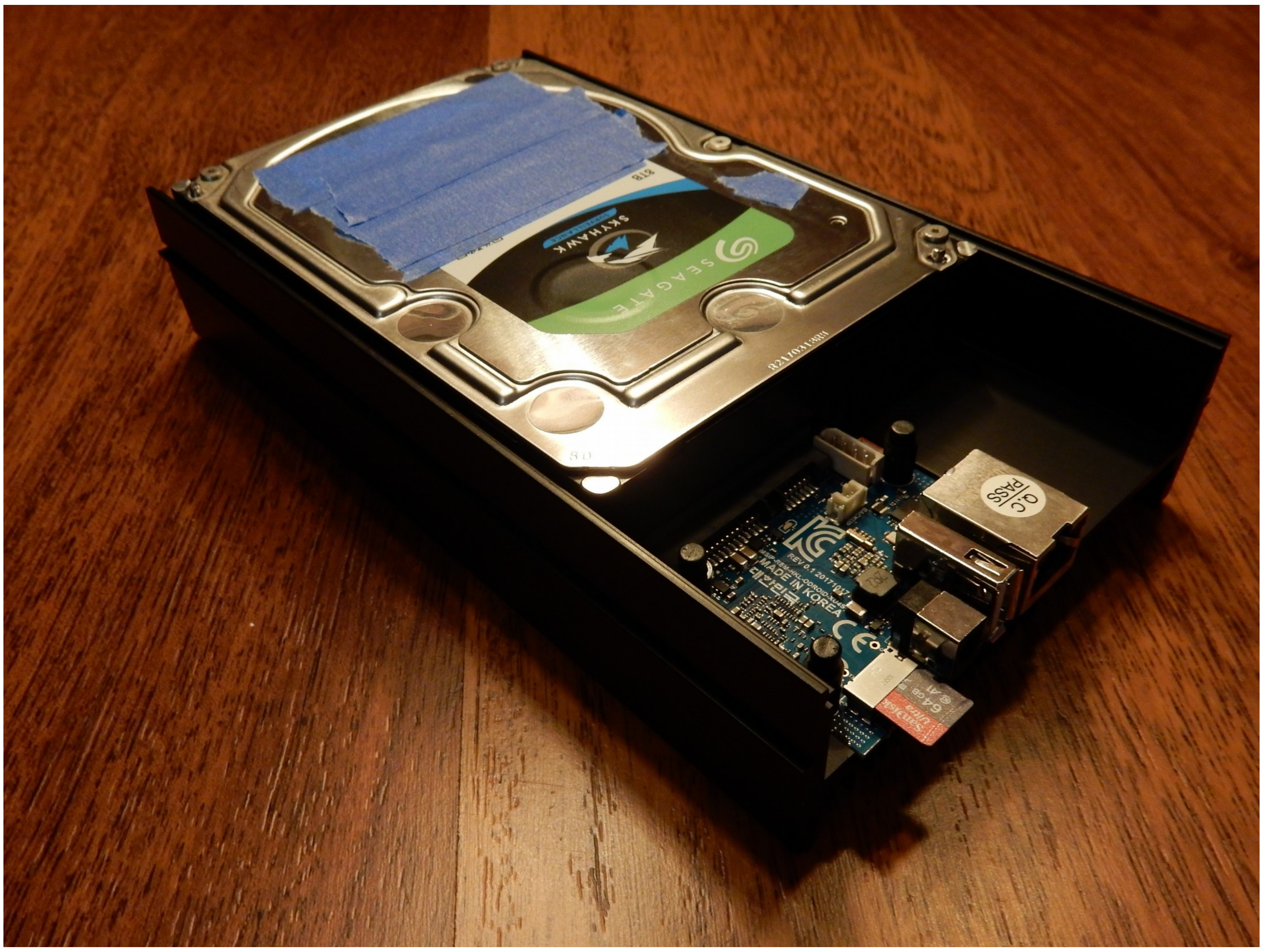


Figure 12: Odroid HC2 – Depicted with 3.5inch Seagate Surveillance Drive (8TB)

See the manufacturer's notes regarding the mounting holes for the HC2. Note that both the 8 and 16TB Seagate drives tested have been successfully and stably mounted with two screws per drive, without any customisation[21].

20 https://www.pclan.com.au/seagate-skyhawk-ai-st16000ve000
21 https://wiki.odroid.com/odroid-xu4/troubleshooting/adding_mount_holes_at_the_odroid-hc2_heatsink

3.2.6 Rack Mounting

For those deploying to Data Centres with active cooling, racking via shelves and stacking drives would be straight forward and cost effective. However, key to enabling passive cooling is ensuring there is sufficient airflow around each storage node. A lot of emphasis was put on finding a common standardised shelf that would enable air to pass through the shelf that the storage nodes were sitting on, in order to maximise the effectiveness of the heat-sink built in to the HC2 enclosure.

Although there were much cheaper alternatives, in order to provide the reader with a globally consistent build option, the Ikea TJUSIG Shoe Rack[22] was selected.

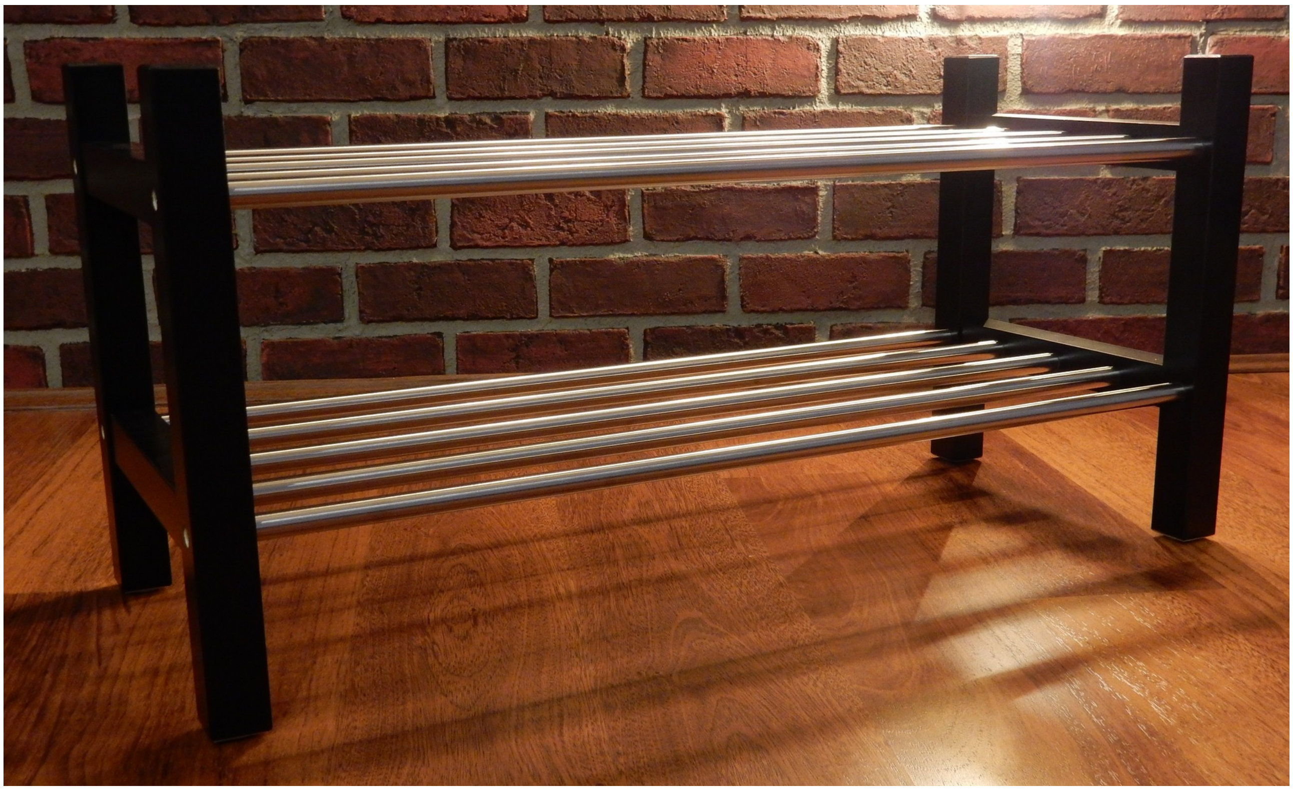


Figure 13: Ikea TJUSIG – Standardised Saturn Generation 2 Racking Unit

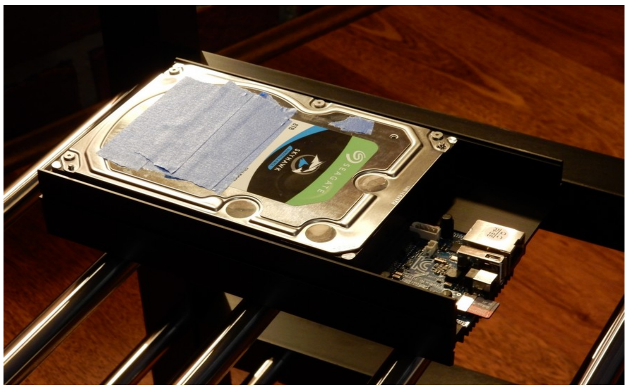


Figure 14: Ikea TJUSIG – Meet the Odroid HC2

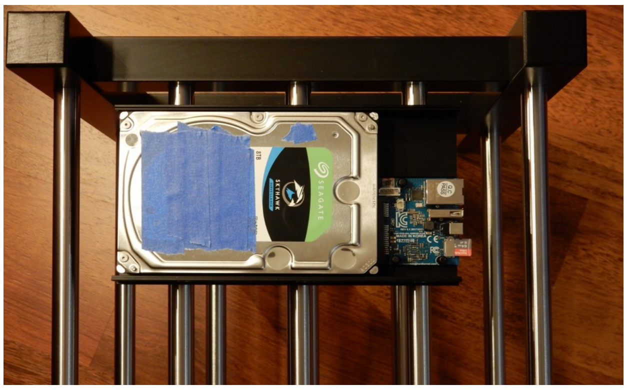


Figure 15: Ikea TJUSIG – Provides ample airflow

Assembled, the rack's outside measurements are 322mm deep, 788mm wide and 374mm high. Each drive sits on three of the five metal rails. This unit will comfortably house five (5) nodes per shelf for ten (10) nodes per rack. It has been designed to stack, making it space efficient while maintaining good airflow. It also means that a petabyte can be assembled within a square metre of floor space.

22 https://www.ikea.com/au/en/p/tjusig-shoe-rack-black-90160956/

3.2.7 Final Assembly

Assembly for a complete rack of ten (10) nodes is straight forward.

Plug-in the Nodes

Plug each node in to its DC power adaptor and it's 1m CAT5e/CAT6 network cable[23]. Using the cable ties (aka zip ties)[24] ensure that power and network cables are tightly attached to the front metal rail, as depicted.

The reason the cable tie is applied is that it effectively anchors the entire node to a fixed distance from the front rail, ensuring that there is no risk of a *guided walk*, in case of unit vibration meeting uneven flooring.

If you are concerned that the node can still *wag* itself left and right, then you could add a smaller braid or rubber tubing to rails two (2) and four (4) which would prevent any sideways movement and also further dampen vibration. As the rails don't accrue any significant heat, this modification would be unlikely to negatively affect cooling.

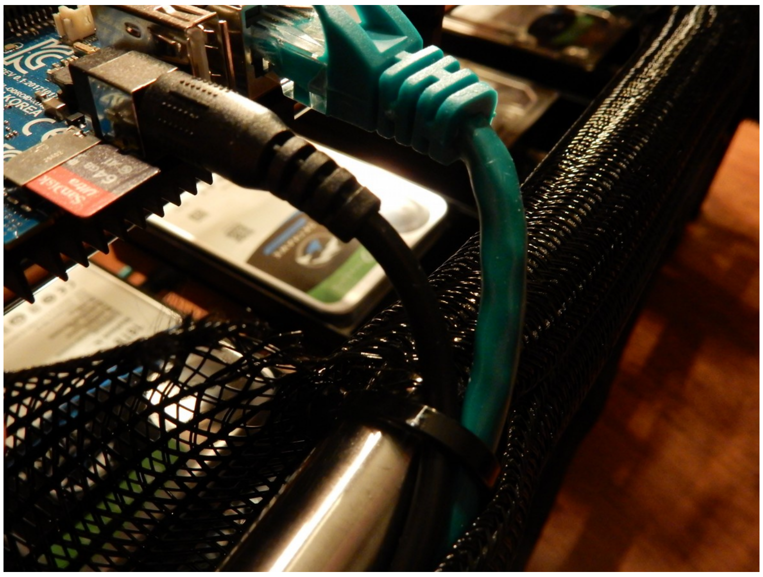


Figure 16: Node cables tightly cable (zip) tied to the front rail

Trunk the Cables

Cut two lengths of the self-closing braided wire wrap[25] using the front rail as a guide, but being sure to leave a few centimetres of extra braid at the end.

Starting from the left for the bottom shelf carefully trunk the cables underneath the front rail using one length of braided wire wrap.

At each node, cut half-way across the braided wrap to provide a *slot* that allows the power and network cables to pass from the trunk to the node.

Repeat this process for the top shelf, starting from the right.

Trunking the cables keeps the build tidy and prevents mistakes (accidentally pulling or cutting cables) in production, and removes any unnecessary impediment to airflow.

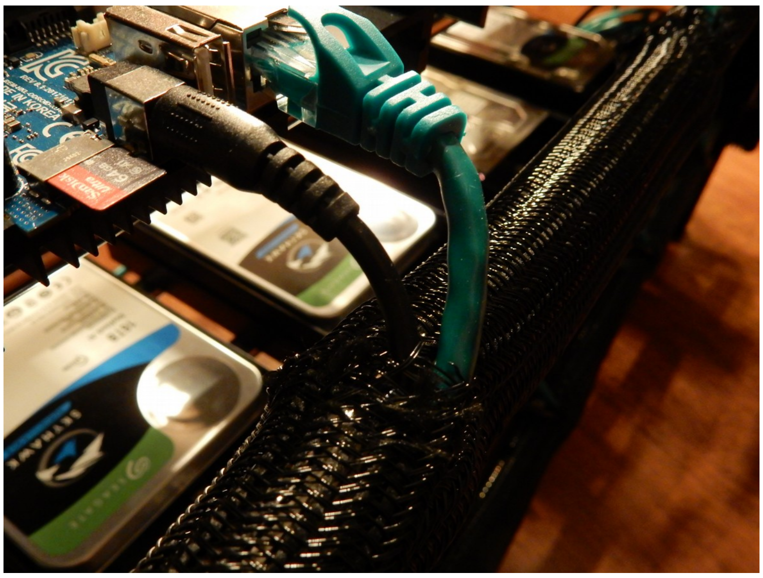


Figure 17: Cable trunk on the top shelf, with closed braided wire wrap

23 https://www.jaycar.com.au/cat-5e-patch-cable-1m-green/p/YN8231
24 https://www.jaycar.com.au/150mm-black-cable-ties-pk-100/p/HP1204
25 https://www.jaycar.com.au/self-closing-braided-wire-wrap-19mm-x-2m/p/WH5636

Bottom Shelf, Right-Hand Wiring

All of the cabling on the bottom shelf now trunks to the right-hand end of the rack.

Power Rail

Cable tie the 6-way power board to the top wooden structural support of the rack – two ties for length, and two anchor points for support. Be careful to avoid the conductive pins – ensure that the cable ties run between the power outlets, not through them.

Plugin the five (5) node DC power supplies and the one (1) access switch DC power supply.

Access Switch

Using three cable ties to provide length, mount the 8-port gigabit access switch[26] to the front wooden structural support of the rack, above the bottom support.

Connect the five (5) node network cables and the one (1) up-link network cable[27] for the distribution switch.

Tidy Cabling

Cable tie the end of the braided wrap so that the cable trunk is kept off the floor – attach it to the bottom wooden structural support of the rack.

Pickup any slack in the cables by coiling them, and use cable ties where necessary to ensure that the spare cable lengths are kept within the rack space.

The only cables that should protrude are the up-link network cable and the lead in for the power board.

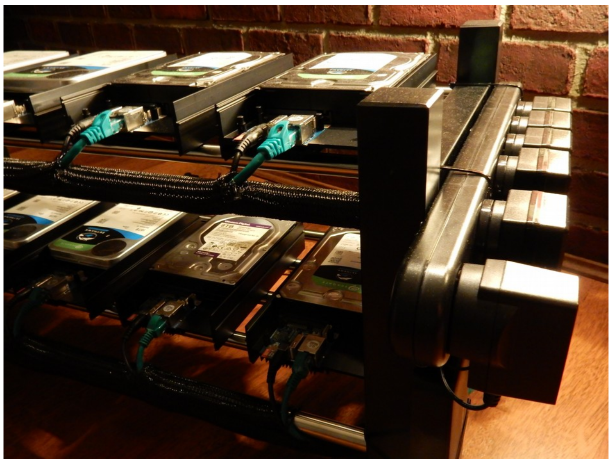


Figure 18: Right-hand power-rail (6-way power board)

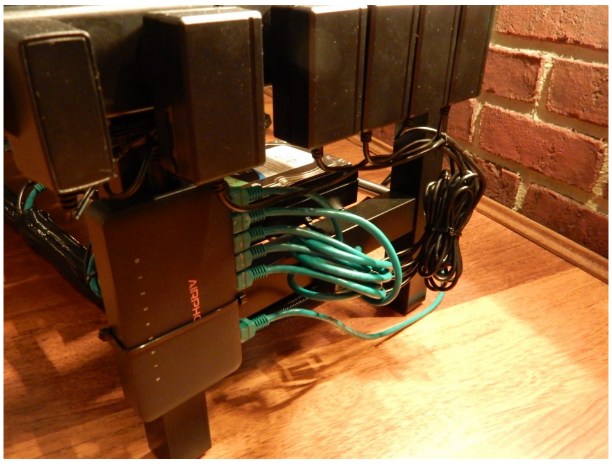


Figure 19: Right-hand access switch and cabling

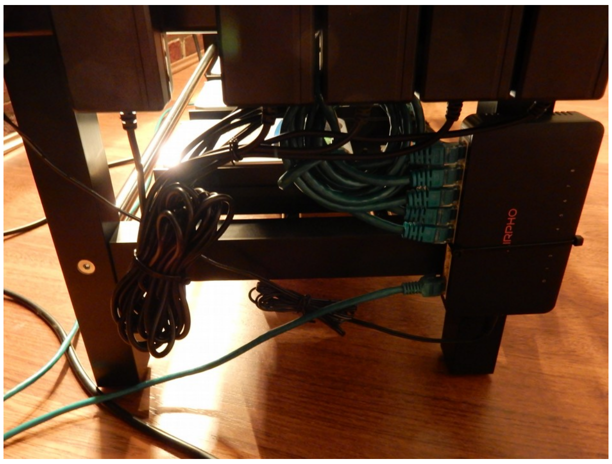


Figure 20: Left-hand power, access switch and cabling

Top Shelf, Left-Hand Wiring

Repeat the previous steps with the left-hand end of the rack, for all of the top shelf cabling. The key difference being the trunk cable tied to the top wooden support.

Note that if you've bought the same components, the positions will switch. For example:

- The lead-in cable for the power board will protrude from the front of the rack rather than the back, and;
- The gigabit switch will have its power connector underneath rather than on top, and;
- If you connect the power adaptors in the same positions then their node numbering will run backwards.

At completion you will have four cables leading away from the rack – two (2) power cables (one per shelf) and two (2) network cables (one per shelf). This means that, by assigning nodes on each shelf a different label, you have physical redundancy (power and networking) aligned with the logical redundancy of the "A"and "B" nodes.

26 https://www.jaycar.com.au/airpho-8-port-gigabit-ethernet-switch/p/YN8386
27 https://www.jaycar.com.au/5m-cat5e-patch-cable-green/p/YN8234

Welcome to Saturn – the Second Generation – *Daphnis*

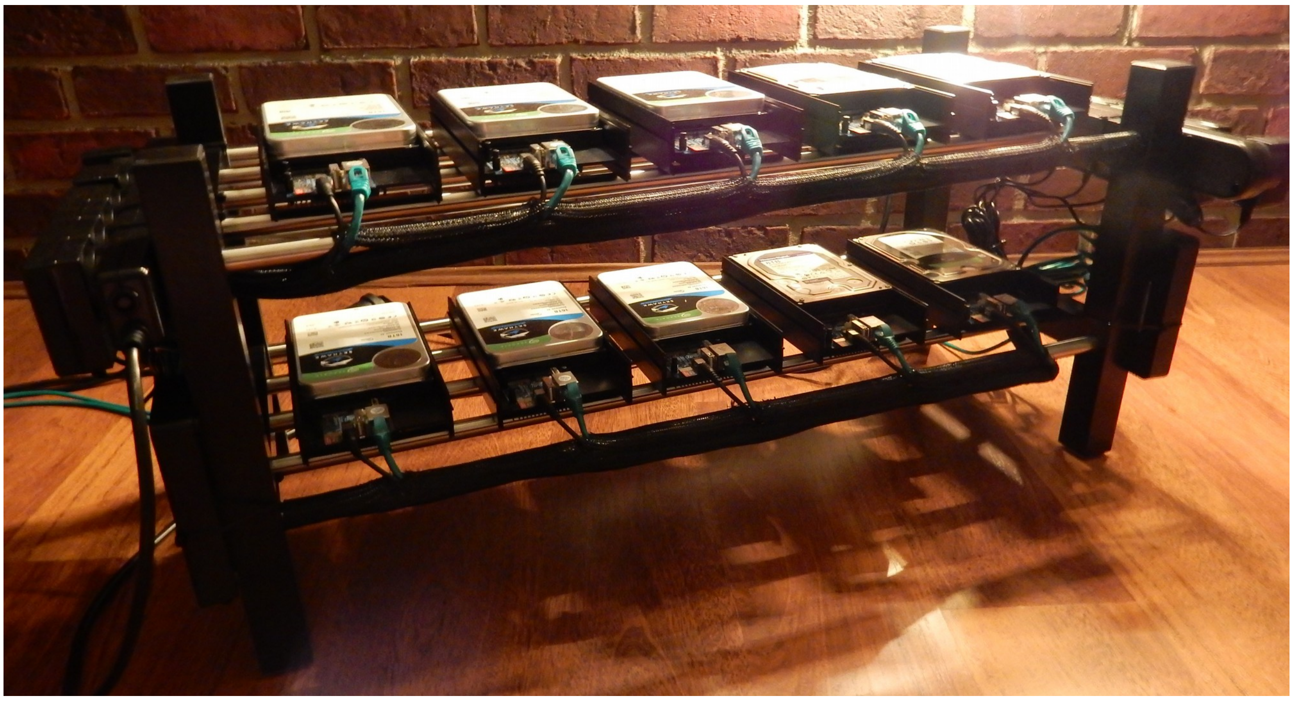


Figure 21: A ten (10) node rack of the Second Generation Saturn – *Daphnis*

Congratulations, at this point you have completed the physical build. Once you've deployed the software you can power-up the cluster, you are ready to go!

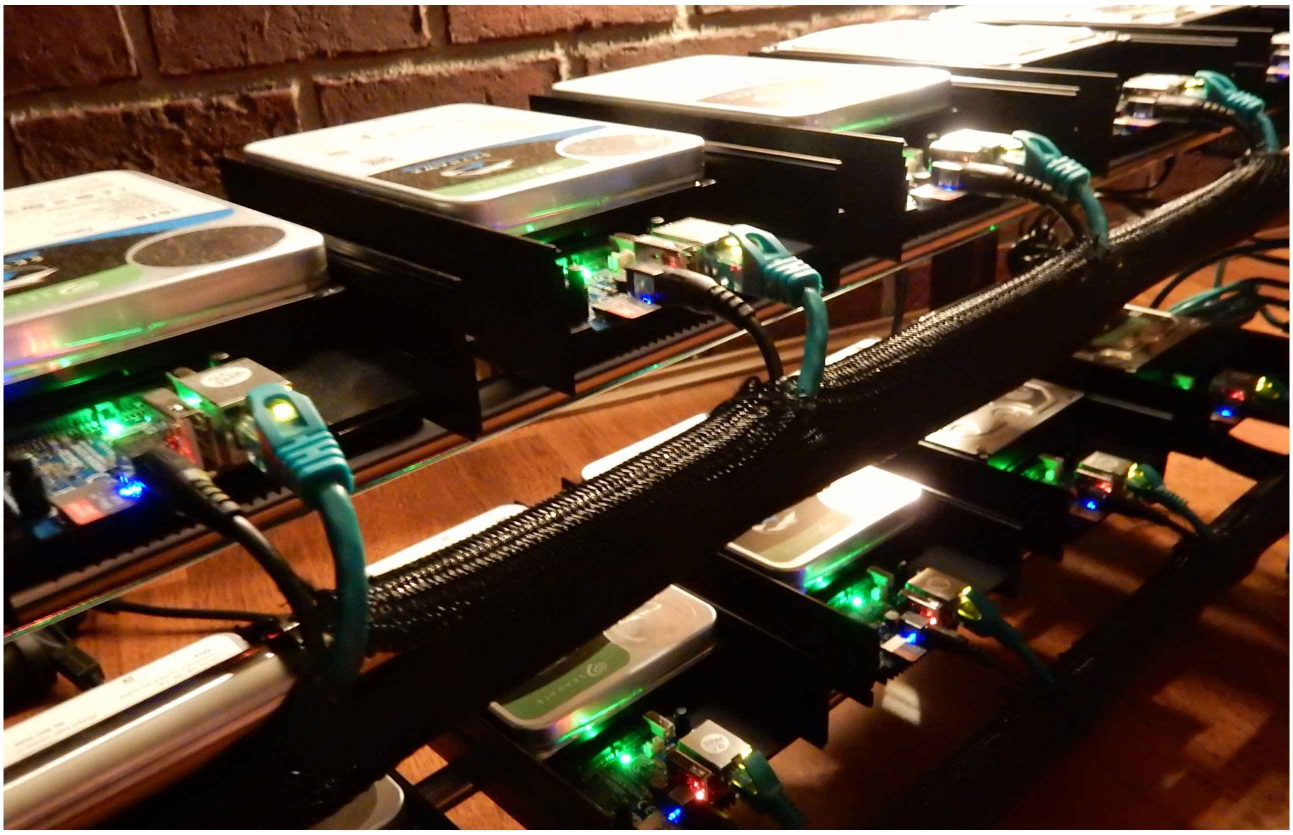


Figure 22: A ten (10) node rack of Saturn, up and running

3.2.8 Hardware Security Token (Optional)

If you value your data at all, then it is recommended that you deploy disk encryption as part of this build.

Whole- (or full-) disk encryption provides physical security for stored data. Should a drive be improperly disposed of, or even if it's simply stolen, then the data on that drive is not immediately compromised – mitigating the all too common *data spill* risk[28].

For the authentication of that encryption layer, this build leverages a hardware security token to provide a predictably strong key.

The Yubico YubiKey Neo[29] has been tested with this build.

The HMAC-SHA1 Challenge-Response and programmable second slot features are used to ensure that a human presented key (pass phrase) is turned into a hash via the secret key that is stored internally to the YubiKey.

While this physical security token is not being used in a two-factor authentication scheme, as a single-factor cryptographic device[30] its value is the ability to assure a large key space for search / brute-force attacks, without onerous requirements on user pass- word/phrase formation.

Please see section 4.2.1 Encryption and Key Management, below, for more detail.

Figure 23: A physical security token – the YubiKey Neo

28 https://nvd.nist.gov/800-53/Rev4/control/IR-9
29 https://support.yubico.com/support/solutions/articles/15000006494-yubikey-neo
30 https://pages.nist.gov/800-63-3/sp800-63b.html#sfcd

3.2.9 Bill of Materials and Cost Benchmarks

The following table lists the components required to build a Gen2 Saturn of modest capacity and performance:

Component	Described	# per Rack	AUD Each	AUD per Rack	% Cost	AUD per R-GB [31]	AUD per U-GB[32]
Odroid HC2 (incl Power)	Single Board Computer (SBC)	10	$95	$950	10.6%	$0.0059	$0.0064
SanDisk 8GB	Boot and Root Storage (Micro SD Card)	10	$11	$110	1.2%	$0.0007	$0.0007
Seagate ST16000VE000	Data Storage (Hard Disk Drive)	10	$760	$7,600	85.1%	$0.0475	$0.0514
Ikea TJUSIG Shoe Rack	Rack Mounting	1	$49	$49	0.6%	$0.0003	$0.0003
Power Board 6-way (Surge and Overload)	Final Assembly	2	$14	$28	0.3%	$0.0002	$0.0002
AIRPHO Gigabit Switch (8 port)	Final Assembly	2	$55	$110	1.2%	$0.0007	$0.0007
Patch Cable, Green (1m)	Final Assembly	10	$4	$40	0.5%	$0.0002	$0.0003
Patch Cable, Green (5m)	Final Assembly	2	$9	$18	0.2%	$0.0001	$0.0001
Self Closing Braided Wrap (19mm x 2m)	Final Assembly	1	$17	$17	0.2%	$0.0001	$0.0001
Cable Tie 100 pack (150 x 3.6mm)	Final Assembly	1	$8	$8	0.1%	$0.0001	$0.0001
			Total Capital Expenditure	$8,930	100%	$0.0558	$0.0603

At the time of writing, 1AUD is 0.69USD; making the build US$6,132.29 or **US$0.0383 per GB (raw) or US$0.04143 per GB (usable), CapEx**. To put that into perspective, let's benchmark Saturn against the market.

CapEx

Of the Cloud storage providers there is only one known to openly share a CapEx view. Before continuing, I want to acknowledge that we have been decade-long fans of the Backblaze open source Storage Pod[33] – legends – so it is with pride that we are benchmarking against the published Backblaze specs.

The Backblaze open source Storage Pod Version 6.0 is **US$0.036 per GB (raw)[34] CapEx**, In comparison Saturn is 6.4% more expensive, noting:

- Saturn's costing includes racks, power rails and access switches (a little over 2% of Saturn's cost), and;
- Saturn has a smoother cost curve – i.e. adding one drive at a time, the Storage Pod has a step-cost at each new chassis, where Saturn has step costs at each shoe rack and network switch with a significantly smaller investment (total and relative).

Adding OpEx

There are three ongoing costs that should be factored within the service life of the infrastructure.

31 Based on raw disk capacity of 16 TB x 10 nodes = 160,000 GB
32 Based on user presented capacity as reported by "df -BG" = 148,000GB
33 https://www.backblaze.com/blog/petabytes-on-a-budget-how-to-build-cheap-cloud-storage/
34 https://www.backblaze.com/blog/open-source-data-storage-server/

Energy

The only significant operational cost relative to this installation is the energy cost:

- The local retail electrical supply is charged at an average rate of AU$0.23155 per kWh (Peak for 6h at $0.4510/h, Shoulder for 9h at $0.1914/h, and Off-peak for 9h at $0.1254/h – including taxes).
- There is no cooling cost as the infrastructure is passively cooled.
- The HC2 uses a 12v 2A power supply – or 24W at peak loading. Assuming the worst case of peak loading all for all ten nodes (240W), running all day (24h) means the cluster will consume approximately 5.76kWh per day.
- So the total energy cost (absolute worst case) would be **AU$1.33 (US$0.92) for the cluster, per day** which equates to:
    - **US$0.0001725 per GB per month (raw)**, or;
    - **US$0.0001865 per GB per month (usable)** [35].

Facilities

Note that at substantial scale – perhaps beyond the five petabyte range – the facilities costs (floor space) should be factored against the cooling costs. As Saturn has been designed to avoid active cooling it requires greater physical space – an interplay that will be reflected in the costs at scale (depending on your facilities costs, active cooling might be cheaper than cubic meters).

As this implementation will not exceed a petabyte, there are no facilities costs being factored (it consuming otherwise unused floor space).

FTE

The Gen1 Saturn has proven to be a *low-touch* solution, and the Gen2 Saturn promises to be even more self-sufficient.

No human effort (full time equivalent) has been costed as this implementation will not exceed a petabyte.

OpEx and Cloud Services

Before we begin there is one other modelling assumption that need to be defined. Cloud Computing proponents would argue that there is high availability offered with the Cloud storage products, as set out in their service level agreements (SLAs). There are two ways that this could be addressed:

1. *Assume that a given SLA translates to a technical implementation*.

   This can be approximated by taking the same Saturn build cost and dividing over less usable capacity. Two copies of the data will take twice the infrastructure; three copies will take thrice. Effectively, this is just a multiplier on the *per GB per month* cost, which is a multiplier on ROI time. The objective would be to use the entire life of the infrastructure (use it until it fails) but without customer impact as components expire – to the extent that you hope the cost of redundant infrastructure pales against the post payback period – which is all margin.

2. *Assume that a given SLA translates to a risk calculation*.

   An SLA is a financial model on a risk model, regardless of the technical implementation. This can be approximated by reducing the life of the infrastructure, for example, managing the mean-time between failure (MTBF) by decommissioning Saturn racks after two years of service while storing only a single copy of the data. So long as the ROI is less than the service life, then the delta between the ROI period and the service life becomes margin. Any delta between the service life imposed and the service life that may have survived to the MTBF or beyond, will be lost margin, wagered against the SLA penalty payouts.

In this model these two outcomes can be equal – using twice as much infrastructure to keep two copies of the data

35 All month calculations are based on 30 day months.

for four (4) years, versus halving the service life of the infrastructure to keep one copy of the data for two years, would be equally valid for example.

For the sake of the modelling, we will give the market the benefit of the doubt and assume both: three (3) copies of all data and a hard service life of three (3) years.

Now we can evaluate the return on investment (ROI) period for Saturn against the market

Backblaze B2

Backblaze rents storage to the market at **US$0.005 per GB per month (usable)** but doesn't talk to an SLA[36].

Market	Assume	Less Operational Expenditure		Amortise Capital	Return on Investment	
USD per GB/mth	**Data Copies**	**Total USD per U-GB/mth**	**Remaining**	**Total USD per U-GB**	**Payback Months**	**Total USD per U-GB, 3yr Saving**
$0.005	1	$0.0001865	$0.0048135	$0.04143	8.6	$0.13188990
	2	$0.0003730	$0.0046270	$0.08286	17.9	$0.08374870
	3	$0.0005595	$0.0044405	$0.12429	28.0	$0.03552400

The Saturn usable storage payback period, scaled to carry 3 copies of data and recovered against the Backblaze B2 rate of US$0.005 per GB, is about **28 months**. Against a hard service life of 36 months, the Saturn solution would **save US$0.04 per GB**.

Microsoft Azure Blob

The cheapest Azure Blob (Standard GPv2, LRS Hot) rate in the local region, is rented to the market at **US$0.0184 per GB per month**[37]. Note that this is only the storage cost – Microsoft also charges for Write, List, Create, Read, and all other operation types (except Delete, which is free) – none of which have been included in this model.

Although advertising locally redundant storage (LRS) as "11 9s", with the *durability of objects over a given year by keeping multiple copies of your data in one datacenter*[38], the SLA offered is monthly and comes in two tiers, less than 99% and less than 99.9%[39].

Market	Assume	Less Operational Expenditure		Amortise Capital	Return on Investment	
USD per GB/mth	**Data Copies**	**Total USD per U-GB/mth**	**Remaining**	**Total USD per U-GB**	**Payback Months**	**Total USD per U-GB, 3yr Saving**
$0.0184	1	$0.0001865	$0.0182135	$0.04143	2.3	$0.61379495
	2	$0.0003730	$0.0180270	$0.08286	4.6	$0.56604780
	3	$0.0005595	$0.0178405	$0.12429	7.0	$0.51737450

The Saturn usable storage payback period, scaled to carry 3 copies of data and recovered against the Microsoft Azure Blob rate of US$0.0184 per GB, is about **7 months**. Against a hard service life of 36 months, the Saturn solution would **save US$0.52 per GB**.

Amazon AWS S3

The cheapest Amazon S3 Standard rate in the local region, is rented to the market at **US$0.023 per GB per month**[40]. Note that this is only the storage cost – Amazon also charges for PUT, COPY, POST, LIST, GET, SELECT, and all other request types – none of which have been included in this model.

36 https://www.backblaze.com/b2/cloud-storage-pricing.html
37 https://azure.microsoft.com/en-us/pricing/details/storage/blobs/
38 https://azure.microsoft.com/en-us/pricing/details/storage/
39 https://azure.microsoft.com/en-us/support/legal/sla/storage/v1_5/
40 https://aws.amazon.com/s3/pricing/

The SLA offered is monthly and comes in three tiers, less than 95.0%, less than 99.0% but greater than 95.0% and less than 99.9% and greater than 99.0%[41].

Market	Assume	Less Operational Expenditure		Amortise Capital	Return on Investment	
USD per GB/mth	**Data Copies**	**Total USD per U-GB/mth**	**Remaining**	**Total USD per U-GB**	**Payback Months**	**Total USD per U-GB, 3yr Saving**
$0.023	1	$0.0001865	$0.0228135	$0.04143	1.8	$0.78022170
	2	$0.0003730	$0.0226270	$0.08286	3.7	$0.73085210
	3	$0.0005595	$0.0224405	$0.12429	5.5	$0.68443525

The Saturn usable storage payback period, scaled to carry 3 copies of data and recovered against the Amazon AWS S3 rate of US$0.023 per GB, is about **5½ months**. Against a hard service life of 36 months, the Saturn solution would **save US$0.68 per GB**.

41 https://aws.amazon.com/s3/sla/

3.3 Software

In this section you will find the steps required to image, install and configure the software to build the Saturn cluster, and to begin using your new storage fabric.

3.3.1 Key Concepts

This generation of Saturn is built upon MooseFS. The application architecture differs to what Saturn has been comprised of before. In order to understand the Saturn architecture you need to understand how MooseFS is intended to be structured/deployed.

So, let's look at the four core components of the MooseFS application architecture:

Metadata Server –

The Metdata Server manages the location (layout) of files, file access and name-space hierarchy. In the open source version of MooseFS, only one Metadata Server instance can run per cluster. A client only talks to the Metadata Server to retrieve / update a file's layout and attributes. The contents of a file (the data itself) is transferred directly between the client and the Chunk Servers. The Metadata Server is a user-space daemon. For performance reasons, the metadata is kept in memory and is only periodically written to disk.

Metalogger Server –

The Metalogger Server periodically retrieves the metadata from the Metadata Server to store it as a backup. There is no known limit to the number of Metalogger Servers that can be run per cluster. In the event of failure of the one Metadata Server instance, the files written by a Metalogger Server can used to launch a replacement Metadata Server. The Metalogger Server is a user-space daemon.

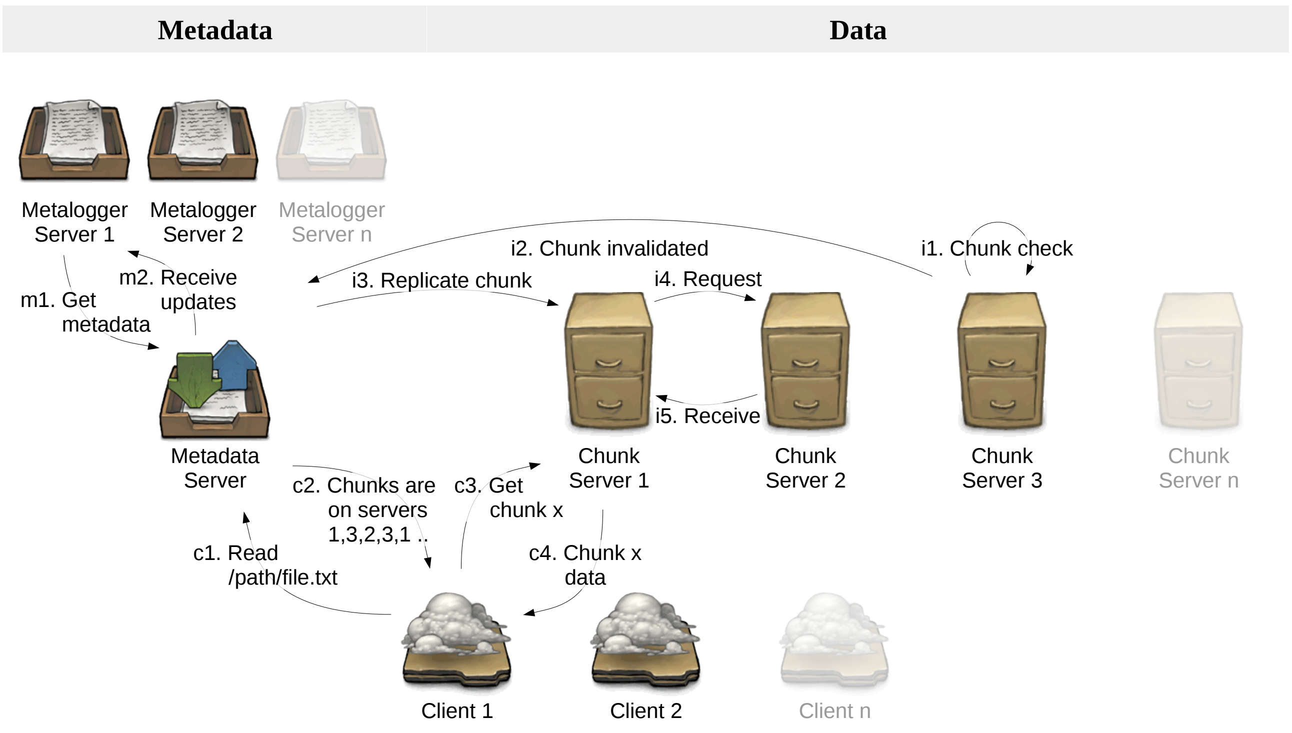
Chunk Server –

The Chunk Server stores the data (file contents) in 64MB *chunks*. Communicating with the Metadata Server, the Chunk Server will periodically validate chunk integrity using stored checksums, it will optionally replicate chunks amongst the other chunk servers in the cluster and, if corrupt, a chunk will automatically be invalidated and another copy replicated (if replication is configured, by policy). To store the chunks on disk it relies on the underlying local file system (XFS is recommended). There is no known limit to the number of Chunk Servers that can be run per cluster (the marketing material refers to "thousands"). The Chunk Server is also a user-space daemon.

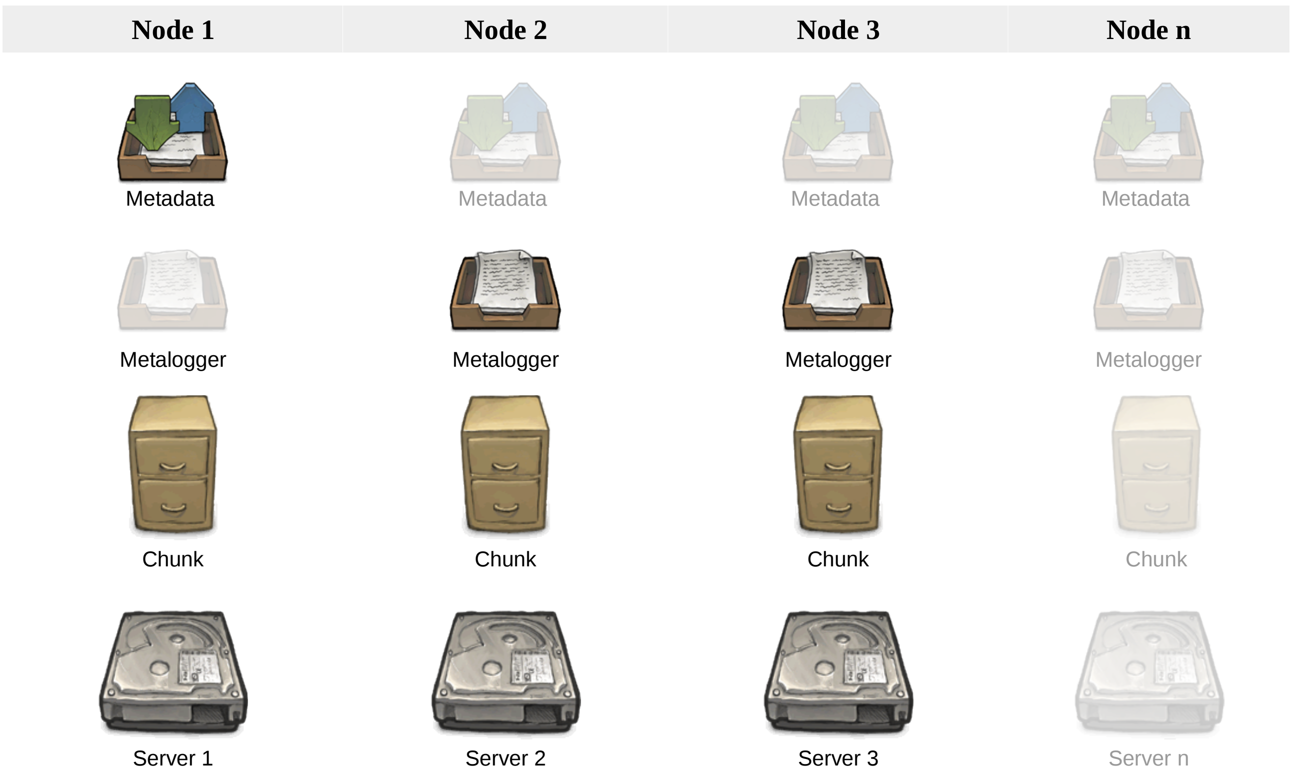
Client –

Each Client communicates directly to both the Metadata Server and the Chunk Servers. MooseFS clients mount the file system into user-space via FUSE. The resultant presented storage from the cluster appears to be a locally mounted POSIX file-system, in the same way NFS or SMB might look. There is no known limit to the number of Clients that can connect to the cluster. It is important to note that the Client reports the raw capacity of the cluster, while the actual consumed space will be dependent upon the number of replicas each chunk makes, by policy.

At a high level, the logical architecture therefore looks like the following.



Implementing this logical architecture physically would be performant but would have a high infrastructure overhead (the above would require six (6) computers, for example). The better solution is to make the physical node a repeatable object, and to activate the features as the logical architecture permits, with one data drive per node (per HC2), i.e. here the same number of instances are deployed with only three (3) computers –



Interestingly this approach to implementation highlights the lack of a truly distributed (active-active) metadata system within MooseFS. It needs to be noted that this system does exist, but only in the licensed / paid “Pro” edition.

But is this physical implementation practical? Yes – in terms of memory, the HC2 has 2GB RAM:

- The Metadata Server consumes approximately 500MB of RAM per million files stored
- The Metalogger Server consumes approximately 5MB of RAM
- The Chunk Server typically consumes less than 200MB of RAM
- The remainder of memory is available to TCP and disk buffers (the build mfsmaster node has 570MB of RAM consumed and has made 1.3GB available to buffers)

In terms of CPU, the HC2 has 8 CPU cores which are more than sufficient, however there are two cases where contention will occur:

- This build ensures that whole-disk encryption is enabled. At high speeds, there is CPU contention between the Chunk Server and the kernel’s encryption software, but the result is a reasonable (and very usable) file-system.
- There has been only one case that has been observed where one workload (a client application) was attempting to extract metadata from hundreds of data files, simultaneously. This was metadata intensive, to the extent that the Metadata Server impeded data throughput to / from Chunk Servers. Unless you expect to frequently see this type of workload, this implementation approach won’t affect you.

As a result of the above, all MooseFS components will be installed on each node, and enabled / disabled accordingly. There will be no distinct Metadata or Metalogger device in this build – instead there will the notion of a **master node** (where the Metadata Server runs) and **slave nodes** (where the Metalogger Server runs). It would be reasonable to expand to a dedicated Metadata node if the cluster was hosting more than two (2) million files, or if the typical workloads are metadata intensive.

In addition to the infrastructure savings, the advantage of this physical architecture is that every non-Metadata Server node is ready to become the Metadata Server in case of failure. It also means there are as n-1 backups of the metadata, distributed through-out the cluster.

Thanks to Mattahan (Paul Davey) for the use of his awesome icons[42] and the memorable axiom “Ignorance is the root of boredom”; I hope your apocalypse has been productive.

42 http://www.iconarchive.com/show/buuf-icons-by-mattahan.1.html

3.3.2 Activity 1: Image the Boot Media

Before you can boot Saturn you must prepare your boot media. There are two methods for preparing the boot media provided in this document, manual – if you wish to perform each step yourself – and scripted – to build a reliable solution, quickly. Please choose one of these and follow those steps to complete this activity.

Please note that these instructions assume that you're using Linux to prepare your boot media.

3.3.2.1 Manual Imaging Process

In case you are building Saturn on different hardware, or just want understand the steps in detail, then please find the manual imaging steps below.

Step 1: Create a working directory

On your workstation, create project directory for Saturn off your home directory, along with a directory that you can use for all of the imaging activity:

```
mkdir -p ~/Saturn/image
```

There is no need to repeat this process in the future.

Step 2: Download the OS Image

The most desirable OS image for the embedded storage node is the smallest Debian Linux distribution that is supported on the hardware.

Odroid refer to the HC2 as their *XU4* platform. For the XU4, Odroid offer[43] a modified version of Ubuntu LTS version 18.04. The *minimal* version is available from the EU[44], KR[45], and US[46].

Change into the project's image directory:

```
cd ~/Saturn/image
```

Download the compressed image (note that it consumes 400MB of disk space):

```
wget "https://dn.odroid.com/5422/ODROID-XU3/Ubuntu/ubuntu-18.04.3-4.14-minimal-odroid-xu4-
20190910.img.xz"
```

And then decompress it (note that it consumes 2.6GB of disk space):

```
xz -d ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz
```

Step 3: Customise the OS Image

In order to assuredly build storage nodes of the same quality every time, the same software versions must be used. However, the Ubuntu image will automatically update itself once it boots. So the auto-update feature (unattended-upgrades) need to be disabled in the image prior to first boot.

Change into the project's image directory:

```
cd ~/Saturn/image
```

Create a temporary mount-point that will be used to mount the raw root partition:

43 https://wiki.odroid.com/odroid-xu4/getting_started/os_installation_guide
44 http://de.eu.odroid.in/ubuntu_18.04lts/XU3_XU4_MC1_HC1_HC2/ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz
45 https://dn.odroid.com/5422/ODROID-XU3/Ubuntu/ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz
46 https://odroid.in/ubuntu_18.04lts/XU3_XU4_MC1_HC1_HC2/ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz

```
mkdir ./mp
```

Find the offset within the Ubuntu image for the root file system:

```
fdisk -lu ./ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img | grep img2
```

Take the *Start* figure (the first number), which should be 264192, and use that to mount the root partition, by offset:

```
sudo mount -o loop,offset=264192 "./ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img"
"./mp"
```

Now use the *rm* command to delete the scripts and binaries needed for the auto-update feature, in the image:

```
sudo rm "./mp/usr/bin/unattended-upgrades"
sudo rm "./mp/usr/bin/unattended-upgrade"
sudo rm "./mp/etc/rc0.d/K01unattended-upgrades"
sudo rm "./mp/etc/rc2.d/S02unattended-upgrades"
sudo rm "./mp/etc/rc3.d/S02unattended-upgrades"
sudo rm "./mp/etc/rc4.d/S02unattended-upgrades"
sudo rm "./mp/etc/rc5.d/S02unattended-upgrades"
sudo rm "./mp/etc/rc6.d/K01unattended-upgrades"
sudo rm "./mp/etc/init.d/unattended-upgrades"
sudo rm "./mp/etc/pm/sleep.d/10_unattended-upgrades-hibernate"
sudo rm "./mp/etc/kernel/postinst.d/unattended-upgrades"
sudo rm "./mp/etc/systemd/system/multi-user.target.wants/unattended-upgrades.service"
```

Un-mount the raw image root file system:

```
sudo umount ./mp
```

Remove the temporary mount-point that is no longer needed:

```
rmdir ./mp
```

The on-disk Ubuntu image has now been customised. There is no need to repeat this process in the future.

Step 4: Burn the Image to the Micro SD card

Plug a new Micro SD card into your card reader (USB Micro SD card adaptors are common[47]), and then connect the card reader to your build system.

To find the USB storage device in Linux, note:

```
dmesg | grep sd | tail
```

Assuming that you found the USB device at “/dev/sdz”, make sure that the following reference to “/dev/sdx” is changed to “/dev/sdz”.

Change into the project’s image directory:

```
cd ~/Saturn/image
```

Then, use the Burn HC2 script to disable the auto-update feature in the image:

```
sudo dd if=ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img of=/dev/sdx bs=512
```

To ensure the image is written to the Micro SD card and not just cache, flush the disk buffers:

```
sudo sync
```

47 https://www.adafruit.com/product/939

Once the sync process has completely finished, you are ready to boot from the Micro SD card.

Unplug the USB Micro SD card adaptor from the computer. Then, extract the Micro SD card from the USB adaptor.

You will need to repeat this step (Step 4), once per HC2 hardware instance.

3.3.2.2 Scripted Imaging Process

The following scripts have been provided to make imaging the boot media a simple task, as well as to include some safe-guards for those late night builds. These scripts may not be appropriate for your environment so please check them before you use them, and adjust them where necessary.

Step 1: Create a working directory

On your workstation, create project directory for Saturn off your home directory, along with a directory that you can use for all of the imaging activity:

```
mkdir -p ~/Saturn/image
```

There is no need to repeat this process in the future.

Step 2: Download the OS Image

The most desirable OS image for the embedded storage node is the smallest Debian Linux distribution that is supported on the hardware.

Odroid refer to the HC2 as their *XU4* platform. For the XU4, Odroid offer[48] a modified version of Ubuntu LTS version 18.04. The *minimal* version is available from the EU[49], KR[50], and US[51].

Change into the project's image directory:

```
cd ~/Saturn/image
```

Download the compressed image (note that it consumes 400MB of disk space):

```
wget “https://dn.odroid.com/5422/ODROID-XU3/Ubuntu/ubuntu-18.04.3-4.14-minimal-odroid-xu4-
20190910.img.xz”
```

And then decompress it (note that it consumes 2.6GB of disk space):

```
xz -d ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz
```

Step 3: Customise the OS Image

In order to assuredly build storage nodes of the same quality every time, the same software versions must be used. However, the Ubuntu image will automatically update itself once it boots. So the auto-update feature (unattended-upgrades) need to be disabled in the image prior to first boot.

Change into the project's image directory:

```
cd ~/Saturn/image
```

Copy the below scripts (mount-part.sh, prefix-ubuntu.sh, and burn-hc2.sh) into this project image directory, where the Ubuntu image file is. Note that a tar-ball is available on the project web site.

48 https://wiki.odroid.com/odroid-xu4/getting_started/os_installation_guide
49 http://de.eu.odroid.in/ubuntu_18.04lts/XU3_XU4_MC1_HC1_HC2/ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz
50 https://dn.odroid.com/5422/ODROID-XU3/Ubuntu/ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz
51 https://odroid.in/ubuntu_18.04lts/XU3_XU4_MC1_HC1_HC2/ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img.xz

Make sure the scripts are executable:

```
chmod 755 *.sh
```

Now use the Pre-Fix Ubuntu script to disable the auto-update feature in the image:

```
sudo ./prefix-ubuntu.sh
```

The on-disk Ubuntu image has now been customised. There is no need to repeat this process in the future.

Step 4: Burn the Image to the Micro SD card

Plug a new Micro SD card into your card reader (USB Micro SD card adaptors are common[52]), and then connect the card reader to your build system.

To find the USB storage device in Linux, note:

```
dmesg | grep sd | tail
```

Assuming that you found the USB device at "/dev/sdz", make sure that the burn-hc2.sh script references to "/dev/sdx" are all changed to "/dev/sdz".

Change into the project's image directory:

```
cd ~/Saturn/image
```

Then, use the Burn HC2 script to disable the auto-update feature in the image:

```
sudo ./burn-hc2.sh
```

Once the Burn HC2 script has completely finished, you are ready to boot from the Micro SD card.

Unplug the USB Micro SD card adaptor from the computer. Then, extract the Micro SD card from the USB adaptor.

You will need to repeat this step (Step 4), once per HC2 hardware instance.

Script: Mount Partition (mount-part.sh)

This script will mount a given partition of a raw image file, so that it can be accessed natively on the local system.

```
#!/bin/bash

usage() {
  echo
  echo "Usage: ${0} {image} {partition} {mountpoint}"
  echo
  echo "Example: ${0} ./ubuntu.img 2 ./mp"
  echo
  exit 1
}

if [ ! ${#} -eq 3 ]
then
  echo "Error: Incorrect number of parameters!"
  usage
fi
if [ ! -f "${1}" ]
then
  echo "Error: File ${1} not found!"
  usage
fi
if [ -z "${2}" ]
then
  echo "Error: Invalid partition ${2}!"
```

52 https://www.adafruit.com/product/939

```
  usage
fi
if [ ! -d "${3}" ]
then
  echo "Error: Mount point ${3} not found!"
  usage
fi

image="${1}"
partition="${2}"
mountpoint="${3}"

partline=`/sbin/fdisk -lu "${image}" | grep "${image}${partition}"`
offset=`echo ${partline} | cut -f2 -d' '`
offsetn=$(( ${offset} + 0 ))
if [ "${offset}" != "${offsetn}" ]
then
  echo "Error: Unable to find offset (found ${offset} in string)"
  echo "       (${partline})"
  exit 1
fi

off=$(( ${offsetn} * 512 ))
mount -o loop,offset=${off} "${image}" "${mountpoint}"
if [ ${?} -ne 0 ]
then
  echo "Error: Failed to mount partition ${partition} at offset ${off}"
  exit 1
fi
exit 0
```

Script: Pre-Fix Ubuntu (prefix-ubuntu.sh)

This script disables the unattended upgrade functionality of Ubuntu in the raw image. It has been written to ensure that this document describes a repeatable *known-good* build (i.e. doesn't change the build result).

```
#!/bin/bash

image="./ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img"
partition=2
mountpoint="/tmp/prefix-$$"
mountpart="./mount-part.sh"

if [ -d "${mountpoint}" ]
then
  echo "${0}: Error mount-point ${mountpoint} exists"
  exit 1
fi
if [ ! -x "${mountpart}" ]
then
  echo "${0}: Error mount partition script ${mountpart} is not executable"
  exit 1
fi

mkdir -p "${mountpoint}"
"${mountpart}" "${image}" ${partition} "${mountpoint}"
if [ ${?} -ne 0 ]
then
  echo "${0}: Error mount partition failed"
  rmdir "${mountpoint}"
  exit 1
fi

rm "${mountpoint}/usr/bin/unattended-upgrades"
rm "${mountpoint}/usr/bin/unattended-upgrade"
rm "${mountpoint}/etc/rc0.d/K01unattended-upgrades"
rm "${mountpoint}/etc/rc2.d/S02unattended-upgrades"
rm "${mountpoint}/etc/rc3.d/S02unattended-upgrades"
rm "${mountpoint}/etc/rc4.d/S02unattended-upgrades"
rm "${mountpoint}/etc/rc5.d/S02unattended-upgrades"
rm "${mountpoint}/etc/rc6.d/K01unattended-upgrades"
rm "${mountpoint}/etc/init.d/unattended-upgrades"
rm "${mountpoint}/etc/pm/sleep.d/10_unattended-upgrades-hibernate"
rm "${mountpoint}/etc/kernel/postinst.d/unattended-upgrades"
rm "${mountpoint}/etc/systemd/system/multi-user.target.wants/unattended-upgrades.service"

umount "${mountpoint}"
rmdir "${mountpoint}"
if [ -d "${mountpoint}" ]
then
  echo "${0}: Error unable to remove mount-point ${mountpoint}"
  exit 1
fi

exit 0
```

Script: Burn HC2 (burn-hc2.sh)

This script will write the Ubuntu image to the Micro SD card via a card reader. Note that /dev/sdx is hard coded into this script to ensure that the operator isn’t able to erase the boot/root disk on the development host – this will not be the right value for your system. Please modify and use this script carefully, as it can irretrievably destroy all data on your system.

```
#!/bin/bash


whoiam=`id | cut -c1-5`
if [ "${whoiam}x" != "uid=0x" ]
then
  echo "Error: you need to be root."
  exit 1
fi

echo
echo "Searching for NTFS partition with exFAT .."
dev=`sfdisk -l 2>/dev/null | grep "NTFS" | grep "exFAT" | cut -c1-8`
if [ "${dev}" != "/dev/sdx" ]
then
  echo "Warning: NTFS volume is not at /dev/sdx!"
  echo
  echo "Searching for W95 partition with FAT32 .."
  dev=`sfdisk -l 2>/dev/null | grep "W95" | grep "FAT32" | cut -c1-8`
  if [ "${dev}" != "/dev/sdx" ]
  then
    echo "Error: W95 FAT32 volume is not at /dev/sdx!"
    exit 1
  else
    echo "Found."
  fi
else
  echo "Found."
fi
if [ "${dev}" != "/dev/sdx" ]
then
  echo
  echo "I'm out."
  exit 2
fi
echo
echo "About to overwrite ${dev} <-- you sure? [enter twice for ok]"
read line
read line
echo "Writing ./ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img to ${dev} .."
dd if=./ubuntu-18.04.3-4.14-minimal-odroid-xu4-20190910.img of=${dev} bs=512
echo "Done."
echo "sync .."
sync
echo "Done."

exit 0
```

3.3.3 Activity 2: Boot and Login

The HC2 has no physical console (i.e. screen and keyboard). To access the Ubuntu Linux shell, you will need to SSH in to the device.

Repeat the following process for each HC2 storage node that you bring online.

Step 1: Ensure DHCP (or BOOTP) is Available

For this process to work, your network will need to have access to a running BOOTP or DHCP server. Please ensure:

1. DHCP (or BOOTP) is being served to your target LAN segment, and;
2. You have access to the *leases* data so that you can see which devices have been assigned what IP addresses, and;
3. You have the ability to statically assign an IP address to a given MAC address.

Typically a LAN (or VLAN) segment will have DHCP provided from the local router, which may be forwarding for an enterprise IP address management system. If you don't administer the DHCP server, then work with your network management team to ensure you have what you need to continue.

Step 2: Boot and Obtain the IP Address

With the imaged Micro SD card inserted, apply power to the HC2. If the device boots successfully the blue LED will flash, two pulses per second, like a heart beat.

At this point the device should have acquired an IP address from the DHCP server.

From the DHCP *leases* data, obtain the IP and MAC address (OEM 00:1E:06) for the host with the agent reporting itself as ***odroid***.

Step 3: SSH to the Node at its temporary IP

Make sure you can SSH in to the device at its temporary IP address – login as ***root*** with password ***odroid***:

```
ssh root@10.146.24.153
```

Step 4: Assign a Static IP Address, and Reboot

In the DHCP server, configure the target static IP address for the MAC address that acquired in Step 2.

Once the configuration change has been made in the DHCP server, reboot the HC2 device within the SSH shell:

```
reboot
```

Step 5: SSH to the Node at its permanent IP

Make sure you can SSH in to the device at its permanent IP address – login as ***root*** with password ***odroid***:

```
ssh root@192.168.1.1
```

Logout again when done:

```
exit
```

See the next section for assumptions regarding IP addressing.

3.3.4 Activity 3: Install and Configure the Software

With your storage node booted and configured with its permanent IP address, you are now ready to install and configure the software.

There are two methods for installation provided in this document, manual – if you wish to perform each step yourself – and scripted – to build a reliable solution, quickly. Please choose one of these and follow those steps to complete this activity.

Note that this process assumes:

- Internet access is available to download OS packages
- That the Saturn LAN is configured as –
    - Subnet: 192.168.1.0
    - Netmask: 255.255.255.0 (CIDR/24)
    - Gateway: 192.168.1.250 (and default route)
        - DNS Server is on the Gateway
        - NTP Server is on the Gateway
- That the Admin LAN is configured as –
    - Subnet: 192.168.2.0
    - Netmask: 255.255.255.0 (CIDR/24)
- Each storage node will be named and numbered sequentially, beginning with *c1* at 192.168.1.1
    - The *mfsmaster* virtual IP will be at: 192.168.1.111
- Storage nodes will be grouped into two classes –
    - c1, c2, c3, c4 and c5 will have the label “A” – the “A” shelf
    - c6, c7, c8, c9 and c10 will have the label “B” – the “B” shelf

Please adjust details in the build process to suit your environment.

3.3.4.1 Manual Installation Process

In case you are installing Saturn on different platform (i.e. VM guests, Docker, or other Linux distributions), or just want understand the steps in detail, then please find the manual installation and configuration steps below.

Step 1: Create a working directory

On your workstation, create project directory for Saturn off your home directory, along with a directory that you can use for all of the installation activity:

```
mkdir -p ~/Saturn/install
```

There is no need to repeat this process in the future.

Step 2: Upload the scripts

Change into the project’s install directory:

```
cd ~/Saturn/install
```

Copy the below scripts (init-disk, mount-all, and umount-all) into this project image directory, where the Ubuntu image file is. Note that a tar-ball is available on the project web site.

Upload all of the scripts to the new storage node – login as ***root*** with password ***odroid***:

```
scp * root@192.168.1.1:.
```

Step 3: Login to the Node

Login to the new storage node – login as ***root*** with password ***odroid***:

```
ssh root@192.168.1.1
```

Step 4: Install base Packages

As the root user on the new storage node, first ensure that the apt data is up to date:

```
apt-get update
```

Install the persistent packet filtering packages:

```
apt-get install -q -y iptables-persistent netfilter-persistent
```

Install the familiar network tools (such as *ifconfig*):

```
apt-get install -q -y net-tools
```

Install the whole-disk encryption tooling and the hardware security token support:

```
apt-get install -q -y cryptsetup yubikey-luks
```

Note that there's a bug in the Yubikey LUKS shell script, as it refers to the wrong path. Correct it with the following:

```
sed -i "s#scripts\/#lib\/cryptsetup\/cryptdisks\.#g" /usr/share/yubikey-luks/ykluks-
keyscript
```

Finally, install the partitioning, file-system, and supporting diagnostic tools:

```
apt-get install -q -y gdisk xfsprogs hdparm hddtemp iperf3 rsync
```

Step 5: Disable unused Services

As the root user on the new storage node, disable the unwanted ModemManager service:

```
systemctl disable ModemManager
systemctl stop ModemManager
```

Disable the unwanted unattended-upgrades service:

```
systemctl disable unattended-upgrades
systemctl stop unattended-upgrades
```

Disable the unwanted WPA Supplicant service:

```
systemctl disable wpa_supplicant
systemctl stop wpa_supplicant
```

Disable unwanted pre-scheduled tasks:

```
rm "/etc/cron.daily/apt-compat"
rm "/etc/cron.daily/bsdmainutils"
rm "/etc/cron.daily/ubuntu-advantage-tools"
```

Disable un-needed TTY interface:

```
systemctl disable "getty@tty1"
rm -f "/etc/systemd/system/getty.target.wants/getty@tty1.service"
```

Step 6: Kernel Tuning

As the root user on the new storage node, remove the existing */etc/sysctl.d* directory and sysctl config file:

```
rm “/etc/sysctl.conf”
rm -r "/etc/sysctl.d"
```

Create the */etc/sysctl.d* directory:

```
mkdir “/etc/sysctl.d”
chmod 755 “/etc/sysctl.d”
```

Use a text editor to save the following configuration text to */etc/sysctl.conf*:

```
## ARM Tuning
kernel.printk = 3 4 1 3
vm.min_free_kbytes = 16384
## Out-of-the-box Raspbian Hardening
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
## Out-of-the-box Debian Hardening
vm.mmap_min_addr = 32768
kernel.kptr_restrict = 1
kernel.yama.ptrace_scope = 1
net.ipv4.conf.default.rp_filter=1
net.ipv4.conf.all.rp_filter=1
net.ipv6.conf.all.use_tempaddr = 2
net.ipv6.conf.default.use_tempaddr = 2
## Security / Hardening
kernel.sysrq = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
net.ipv6.conf.eth0.disable_ipv6 = 1
net.netfilter.nf_conntrack_tcp_loose = 0
## Performance over security
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_timestamps = 0
## Network Performance
net.ipv4.tcp_window_scaling = 1
net.core.rmem_max = 268435456
net.core.rmem_default = 67108864
net.core.wmem_max = 268435456
net.core.wmem_default = 67108864
net.core.optmem_max = 2097152
net.ipv4.tcp_rmem = 1048576 67108864 268435456
net.ipv4.tcp_wmem = 1048576 67108864 268435456
net.ipv4.tcp_mem = 1048576 67108864 268435456
net.ipv4.udp_mem = 1048576 67108864 268435456
net.core.netdev_max_backlog = 10000
net.ipv4.tcp_no_metrics_save = 1
## Memory / Cache Management
vm.overcommit_memory = 1
# vm.dirty_background_ratio = 3
vm.dirty_ratio = 10
vm.vfs_cache_pressure = 120
```

Fix the permissions on the */etc/sysctl.conf* file:

```
chmod 644 “/etc/sysctl.conf”
```

And then symlink that file:

```
ln -s "/etc/sysctl.cnf" "/etc/sysctl.d/99-sysctl.conf"
```

Step 7: Time Synchronisation

As the root user on the new storage node, use a text editor to add the following to the end of the */etc/systemd/timesyncd.conf* file:

```
NTP=192.168.1.250
```

Then enable time synchronisation:

```
timedatectl set-ntp true
```

Step 8: Configure Firewall Rules

As the root user on the new storage node, use a text editor to make the following the entire contents of the */etc/iptables/rules.v6* file:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
COMMIT
```

Then use a text editor to make the following the entire contents of the */etc/iptables/rules.v4* file:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m tcp -p tcp --dport 9400:9450 -m state --state NEW -j ACCEPT
-A INPUT -m tcp -p tcp --dport 22 -s 192.168.2.0/24 -m state --state NEW -j ACCEPT
-A INPUT -m udp -p udp --sport 67:68 --dport 67:68 -d 255.255.255.255 -m state --state NEW
-j ACCEPT
-A INPUT -d 224.0.0.1 -j DROP
-A INPUT -m udp -p udp --dport 137:138 -j DROP
-A INPUT -j LOG --log-prefix "INPUT:DROP:" --log-level 6
-A INPUT -j DROP
-A FORWARD -j LOG --log-prefix "FORWARD:DROP:" --log-level 6
-A FORWARD -j DROP
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -m tcp -p tcp --dport 9400:9450 -m state --state NEW -j ACCEPT
-A OUTPUT -m udp -p udp --dport 123 -m state --state NEW -j ACCEPT
-A OUTPUT -m udp -p udp --sport 67:68 --dport 67:68 -m state --state NEW -j ACCEPT
-A OUTPUT -m udp -p udp --dport 53 -m state --state NEW -j ACCEPT
-A OUTPUT -m tcp -p tcp --dport 53 -m state --state NEW -j ACCEPT
-A OUTPUT -j LOG --log-prefix "OUTPUT:DROP:" --log-level 6
-A OUTPUT -j DROP
COMMIT
```

Step 9: Install MooseFS Packages

As the root user on the new storage node, add the MooseFS apt repository key to the local repo:

```
wget -O - "https://ppa.moosefs.com/moosefs.key" | apt-key add -
```

Then, leverage the Raspbian repo for MooseFS packages as MooseFS has no armf platform architecture under its Ubuntu repo. Use a text editor to create */etc/apt/sources.list.d/moosefs.list* with the following content:

```
deb http://ppa.moosefs.com/moosefs-3/apt/raspbian/stretch stretch main
```

Ensure that the apt data is up to date:

```
apt-get update
```

Install the MooseFS master packages (Metadata Server, CGI and CLI components):

```
apt-get install -q -y moosefs-master moosefs-cgi moosefs-cgiserv moosefs-cli
```

Install the remaining MooseFS packages (Metalogger Server and Chunk Server):

```
apt-get install -q -y moosefs-chunkserver moosefs-metalogger
```

If you wish to disable this repo again until you’re ready to do patching then, with the MooseFS packages installed, remove the unneeded repo and update the apt data:

```
rm /etc/apt/sources.list.d/moosefs.list
apt-get update
```

Step 10: Configure MooseFS Services

As the root user on the new storage node, ensure that the */etc/mfs* directory is in place with the correct permissions:

```
mkdir /etc/mfs
chown root:root /etc/mfs
chmod 755 /etc/mfs
```

Create the */chunks* directory, with the correct permissions, to mount the encrypted volumes:

```
mkdir /chunks
chown root:root /chunks
chmod 755 /chunks
```

Using a text editor, add the *mfsmaster* entry to */etc/hosts* for the virtual IP address:

```
192.168.1.111 mfsmaster
```

Metadata Server

Copy the example *mfsexports.cfg* into the */etc/mfs* directory:

```
cp "/etc/mfs/mfsexports.cfg.sample" "/etc/mfs/mfsexports.cfg"
chmod 644 "/etc/mfs/mfsexports.cfg"
```

Copy the example *mfsmaster.cfg* into the */etc/mfs* directory:

```
cp "/etc/mfs/mfsmaster.cfg.sample" "/etc/mfs/mfsmaster.cfg"
chmod 644 "/etc/mfs/mfsmaster.cfg"
```

Use a text editor to add the following to the end of the */etc/mfs/mfsmaster.cfg* file:

```
### SATURN ###
DATA_PATH = /metadata/master
NICE_LEVEL = -15
# LOCK_MEMORY = 1
ATIME_MODE = 4
```

Prepare the */etc/default/moosefs-master* file:

```
touch "/etc/default/moosefs-master"
chmod 644 "/etc/default/moosefs-master"
```

Metalogger Server

Copy the example *mfsmetalogger.cfg* into the */etc/mfs* directory:

```
cp "/etc/mfs/mfsmetalogger.cfg.sample" "/etc/mfs/mfsmetalogger.cfg"
chmod 644 "/etc/mfs/mfsmetalogger.cfg"
```

Use a text editor to add the following to the end of the */etc/mfs/mfsmetalogger.cfg* file:

```
### SATURN ###
DATA_PATH = /metadata/logger
NICE_LEVEL = -15
META_DOWNLOAD_FREQ = 2
```

Prepare the */etc/default/moosefs-metalogger* file:

```
touch "/etc/default/moosefs-metalogger"
chmod 644 "/etc/default/moosefs-metalogger"
```

CGI Server

Prepare the */etc/default/moosefs-cgiserv* file:

```
touch "/etc/default/moosefs-cgiserv"
chmod 644 "/etc/default/moosefs-cgiserv"
```

Prepare the */etc/default/moosefs-cgiserv* file:

```
touch "/etc/default/moosefs-cgiserv"
chmod 644 "/etc/default/moosefs-cgiserv"
```

Enable the CGI Server:

```
echo "MFSCGISERV_ENABLE=true" > /etc/default/moosefs-cgiserv
systemctl enable moosefs-cgiserv
```

Chunk Server

Copy the example *mfshdd.cfg* into the */etc/mfs* directory:

```
cp "/etc/mfs/mfshdd.cfg.sample" "/etc/mfs/mfshdd.cfg"
chmod 644 "/etc/mfs/mfshdd.cfg"
```

Copy the example *mfschunkserver.cfg* into the */etc/mfs* directory:

```
cp "/etc/mfs/mfschunkserver.cfg.sample" "/etc/mfs/mfschunkserver.cfg"
chmod 644 "/etc/mfs/mfschunkserver.cfg"
```

Use a text editor to add the following to the end of the */etc/mfs/mfschunkserver.cfg* file – noting whether this node is an “A” node or a “B” node:

```
### SATURN ###
NICE_LEVEL = -10
LOCK_MEMORY = 1
LABELS = A
```

Prepare the */etc/default/moosefs-chunkserver* file:

```
touch "/etc/default/moosefs-chunkserver"
chmod 644 "/etc/default/moosefs-chunkserver"
```

Enable the Chunk Server:

```
echo "MFSCHUNKSERVER_ENABLE=true" > /etc/default/moosefs-chunkserver
systemctl enable moosefs-chunkserver
```

Master Node

If this node is to be the one master node in the cluster, then perform the following.

Enable the Metadata Server:

```
echo "MFSMASTER_ENABLE=true" > /etc/default/moosefs-master
systemctl enable moosefs-master
```

Disable the Metalogger Server:

```
echo "MFSMETALOGGER_ENABLE=false" > /etc/default/moosefs-metalogger
systemctl disable moosefs-metalogger
```

Slave Node

If this node is not the one master node in the cluster, then perform the following.

Disable the Metadata Server:

```
echo "MFSMASTER_ENABLE=false" > /etc/default/moosefs-master
systemctl disnable moosefs-master
```

Enable the Metalogger Server:

```
echo "MFSMETALOGGER_ENABLE=true" > /etc/default/moosefs-metalogger
systemctl enable moosefs-metalogger
```

Step 11: Create the Script: iptdown

As the root user on the new storage node, ensure that the */usr/sbin/iptdown* script is in place with the correct permissions:

```
touch "/usr/sbin/iptdown"
chmod 700 "/usr/sbin/iptdown"
```

Then use a text editor to make the following the entire contents of the */usr/sbin/iptdown* file:

```
#!/bin/bash

iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
```

Step 12: Create the Script: sched-disk

As the root user on the new storage node, ensure that the */usr/sbin/sched-disk* script is in place with the correct permissions:

```
touch "/usr/sbin/sched-disk"
chmod 700 "/usr/sbin/sched-disk"
```

Then use a text editor to make the following the entire contents of the */usr/sbin/sched-disk* file:

```
#!/bin/bash

for DISK in sda sdb sdc sdd sde sdf sdg sdh
do
  if [ -b "/dev/${DISK}" ]
  then
    echo "+--- Configuring scheduler for /dev/${DISK}"
    echo "deadline"   > /sys/block/${DISK}/queue/scheduler
    echo "24"         > /sys/block/${DISK}/queue/iosched/fifo_batch
    echo "750"        > /sys/block/${DISK}/queue/iosched/read_expire
    echo "4"          > /sys/block/${DISK}/queue/iosched/writes_starved
    echo "128"        > /sys/block/${DISK}/queue/read_ahead_kb
    echo "128"        > /sys/block/${DISK}/queue/nr_requests
  fi
done
```

Step 13: Create the Script: tempck

As the root user on the new storage node, ensure that the */usr/sbin/tempck* script is in place with the correct permissions:

```
touch "/usr/sbin/tempck"
chmod 700 "/usr/sbin/tempck"
```

Then use a text editor to make the following the entire contents of the */usr/sbin/tempck* file:

```
#!/bin/bash

for ZONE in 0 1 2 3 4 5 6 7
do
  if [ -f "/sys/devices/virtual/thermal/thermal_zone${ZONE}/temp" ]
  then
    temp=`cat "/sys/devices/virtual/thermal/thermal_zone${ZONE}/temp"`
    temp=$(( ${temp} / 1000 ))
    alarm=""
    if [ ${temp} -gt 93 ]
    then
      alarm=" <-- warning"
    fi
    echo "tz${ZONE}: ${temp} ${alarm}"
  fi
done
for DISK in sda sdb sdc sdd sde sdf sdg sdh
do
  if [ -b "/dev/${DISK}" ]
  then
    temp=`hddtemp -n -u c SATA:/dev/${DISK}`
    if [ ${temp} -gt 63 ]
    then
      alarm=" <-- warning"
    fi
    echo "${DISK}: ${temp} ${alarm}"
  fi
done
```

Step 14: Create the Script: tempwatch

As the root user on the new storage node, ensure that the */usr/sbin/tempwatch* script is in place with the correct permissions:

```
touch "/usr/sbin/tempwatch"
chmod 700 "/usr/sbin/tempwatch"
```

Then use a text editor to make the following the entire contents of the */usr/sbin/tempwatch* file:

```
#!/bin/bash

watch -n2 tempck
```

Step 15: Create the Script: cpucap

As the root user on the new storage node, ensure that the */usr/sbin/cpucap* script is in place with the correct permissions:

```
touch "/usr/sbin/cpucap"
chmod 700 "/usr/sbin/cpucap"
```

Then use a text editor to make the following the entire contents of the */usr/sbin/cpucap* file:

```
#!/bin/bash

if [ "${1}x" == "resetx" -o "${1}x" == "restorex" ]
then
  echo "1500000" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
  echo "2000000" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq
  echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  echo "performance" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor
else
  echo "1000000" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
  echo "1000000" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq
  echo "ondemand" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  echo "ondemand" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor
fi

max0f=`cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq`
max4f=`cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq`
gov0f=`cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor`
gov4f=`cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor`
echo "First CPU, governor and max frequency:  ${gov0f} ${max0f}"
echo "Second CPU, governor and max frequency: ${gov4f} ${max4f}"
```

Step 16: Create the Script: masternic

As the root user on the new storage node, ensure that the */usr/sbin/masternic* script is in place with the correct permissions:

```
touch "/usr/sbin/masternic"
chmod 700 "/usr/sbin/masternic"
```

Then use a text editor to make the following the entire contents of the */usr/sbin/masternic* file:

```
#!/bin/bash

am_master=0
am_slave=0

if [ -f "/etc/default/moosefs-master" ]
then
  grep "MFSMASTER_ENABLE=true" "/etc/default/moosefs-master" >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    am_master=1
  fi
fi
if [ -f "/etc/default/moosefs-metalogger" ]
then
  grep "MFSMETALOGGER_ENABLE=true" "/etc/default/moosefs-metalogger" >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    am_slave=1
  fi
fi

if [ ${am_master} -eq 1 -a ${am_slave} -eq 0 ]
then
  echo "Assessment: This node is a master."
  logger "masternic assessment: This node is a master."
  arping -q -c 3 -f -I eth0 mfsmaster >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    echo "Error: There is another node responding as master, aborting."
    logger "masternic error: There is another node responding as master, aborting."
    exit 1
  else
    echo "Enabling master interface"
    logger "masternic enabling master interface"
    ifconfig eth0:1 mfsmaster netmask 255.255.255.255
    exit 0
  fi
elif [ ${am_master} -eq 0 -a ${am_slave} -eq 1 ]
then
  echo "Assessment: This node is a slave."
  logger "masternic assessment: This node is a slave."
  echo "Disabling master interface"
  logger "masternic disabling master interface"
  ifconfig eth0:1 down >/dev/null 2>&1
  exit 0
elif [ ${am_master} -eq 1 -a ${am_slave} -eq 1 ]
then
  echo "Error: This node is both master and slave, aborting"
  logger "masternic error: This node is both master and slave, aborting"
  exit 1
elif [ ${am_master} -eq 0 -a ${am_slave} -eq 0 ]
then
  echo "Error: This node is neither master nor slave, aborting"
  logger "masternic error: This node is neither master nor slave, aborting"
  exit 1
fi
```

Step 17: Edit the Script: rc.local

As the root user on the new storage node, use a text editor to make the following the entire contents of the */etc/rc.local* file:

```
#!/bin/sh -e

for scr in cpucap masternic sched-disk
do
  if [ -x "/usr/sbin/${scr}" ]
  then
    nohup "/usr/sbin/${scr}" < /dev/null > /dev/null 2>&1 &
  fi
done

exit 0
```

Step 18: Copy the Script: init-disk

As the root user on the new storage node, copy the uploaded (in Step 2) *init-disk* script to */usr/sbin* with the correct permissions:

```
cp /root/init-disk /usr/sbin/
chmod 700 /usr/sbin/init-disk
```

Step 19: Copy the Script: mount-all

As the root user on the new storage node, copy the uploaded (in Step 2) *mount-all* script to */usr/sbin* with the correct permissions:

```
cp /root/mount-all /usr/sbin/
chmod 700 /usr/sbin/mount-all
```

Step 20: Copy the Script: umount-all

As the root user on the new storage node, copy the uploaded (in Step 2) u*mount-all* script to */usr/sbin* with the correct permissions:

```
cp /root/mount-all /usr/sbin/
chmod 700 /usr/sbin/umount-all
```

Step 21: Set the host name and remove the banners

As the root user on the new storage node, set the hostname:

```
echo “c1” > /etc/hostname
```

Update the hostname in */etc/hosts*:

```
sed -i "s/odroid/c1/g" /etc/hosts
```

Remove the banners to reduce unnecessary information leakage:

```
cp /etc/issue /etc/issue-orig
echo > /etc/issue
cp /etc/issue.net /etc/issue.net-orig
echo > /etc/issue.net
```

Step 22: Add the admin user:

As the root user on the new storage node, set the admin user *guru*:

```
adduser "guru" --gecos ",,," --disabled-password
```

Ensure that the admin user has *sudo* rights:

```
usermod -a -G sudo "guru"
```

Set the admin user’s password:

```
passwd "guru"
```

Step 23: Harden the SSHD

As the root user on the new storage node, use a text editor to make the following the entire contents of the */etc/ssh/sshd_config* file, in order to disable remote root logon:

```
ChallengeResponseAuthentication no
UsePAM yes
X11Forwarding no
PrintMotd no
cceptEnv LANG LC_*
Subsystem      sftp    /usr/lib/openssh/sftp-server
```

Step 24: Validate remote access

Do not logout of the storage node until you are satisfied that you can login as the guru user with the password you set – i.e.:

```
ssh guru@192.168.1.1
```

As remote root access will be disabled as a security feature, you will need to *sudo* to root, from guru:

```
sudo bash
```

You can now move on to initialising the disk for this node.

3.3.4.2 Scripted Installation Process

The following scripts have been provided to make the software installation and setup a simple task, as well as to include some safe-guards for those late night builds. These scripts may not be appropriate for your environment so please check them before you use them, and adjust them where necessary.

Step 1: Create a working directory

On your workstation, create project directory for Saturn off your home directory, along with a directory that you can use for all of the installation activity:

```
mkdir -p ~/Saturn/install
```

There is no need to repeat this process in the future.

Step 2: Upload the installer

Change into the project's install directory:

```
cd ~/Saturn/install
```

Copy the below scripts (saturn-install.sh, init-disk, mount-all, and umount-all) into this project image directory, where the Ubuntu image file is. Note that a tar-ball is available on the project web site.

Upload all of the scripts to the new storage node – login as ***root*** with password ***odroid***:

```
scp * root@192.168.1.1:.
```

Step 3: Install and Configure the Node

Login to the new storage node – login as ***root*** with password ***odroid***:

```
ssh root@192.168.1.1
```

Make sure the installation script is executable on the node:

```
chmod 700 saturn-install.sh
```

<u>Master Node</u>

If this node is to be the one master node in the cluster (and in this case, the first node is *c1* and it is a *master*), then perform the following:

```
./saturn-install.sh c1 master
```

<u>Slave Node</u>

There can only be one master in the cluster, so the next node would be *c2* as a *slave*, then *c3* as a *slave*, etc. If this node is not the one master node in the cluster, then perform the following:

```
./saturn-install.sh c2 slave
```

<u>Validate</u>

At the end of the script a new account for the administrative user *guru* will be created. Do not logout of the storage node until you are satisfied that you can login as the guru user with the password you set – i.e.:

```
ssh guru@192.168.1.1
```

As remote root access will be disabled as a security feature, you will need to *sudo* to root, from guru:

```
sudo bash
```

You can now move on to initialising the disk for this node.

Script: Saturn Install (saturn-install.sh)

This script will perform all of the post-boot steps required to install and configure the open source software needed for Saturn.

```
#!/bin/bash

declare -A node_ips
node_ips=(
  ['c1']="192.168.1.1"
  ['c2']="192.168.1.2"
  ['c3']="192.168.1.3"
  ['c4']="192.168.1.4"
  ['c5']="192.168.1.5"
  ['c6']="192.168.1.6"
  ['c7']="192.168.1.7"
  ['c8']="192.168.1.8"
  ['c9']="192.168.1.9"
  ['c10']="192.168.1.10"
)
declare -A node_labels
node_labels=(
  ['c1']="A"
  ['c2']="A"
  ['c3']="A"
  ['c4']="A"
  ['c5']="A"
  ['c6']="B"
  ['c7']="B"
  ['c8']="B"
  ['c9']="B"
  ['c10']="B"
)

mfsmaster="192.168.1.111"
lan_gateway="192.168.1.250"
lan_cidr="22"
is_static="false"
ntp_server="192.168.1.250"
name_server="192.168.1.250"
admin_network="192.168.2.0/24"


usage() {
  echo
  echo "Usage: ${0} node-name [master|slave]"
  echo
  echo -n "       Where valid node-name values are:"
  for key in "${!node_ips[@]}"
  do
    echo -n " ${key}"
  done
  echo
  echo
  exit 1
}


failure() {
  echo
  echo "Failure detected, exiting."
  echo
  exit 2
}


do_keyboard() {
  if [ ! -f "/etc/default/keyboard.orig" ]
  then
    echo
    echo "+---- Fixing keyboard layout"
    cp /etc/default/keyboard /etc/default/keyboard.orig
    sed -i "s/gb/us/g" /etc/default/keyboard
  fi
}


do_base_packages() {
  dpkg -l iptables-persistent >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    echo
    echo "+---- Adding base packages"
    export DEBIAN_FRONTEND="noninteractive"
    apt-get update
    [ ${?} -ne 0 ] && failure
    apt-get install -q -y iptables-persistent netfilter-persistent
    [ ${?} -ne 0 ] && failure
    apt-get install -q -y net-tools
    [ ${?} -ne 0 ] && failure
    apt-get install -q -y cryptsetup yubikey-luks
    [ ${?} -ne 0 ] && failure
    sed -i "s#scripts\/#lib\/cryptsetup\/cryptdisks\.#g" /usr/share/yubikey-luks/ykluks-
keyscript
    apt-get install -q -y gdisk xfsprogs hdparm hddtemp iperf3 rsync
    [ ${?} -ne 0 ] && failure
  fi
}


do_moosefs_packages() {
  dpkg -l moosefs-* >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    echo
    echo "+---- Adding MooseFS packages"
    # Ubuntu has no armf, so leveraging Raspbian, carefully
    wget -O - "https://ppa.moosefs.com/moosefs.key" | apt-key add -
    # echo "deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/bionic bionic main" >
/etc/apt/sources.list.d/moosefs.list
    echo "deb http://ppa.moosefs.com/moosefs-3/apt/raspbian/stretch stretch main" >
/etc/apt/sources.list.d/moosefs.list
    apt-get update
    [ ${?} -ne 0 ] && failure
    apt-get install -q -y moosefs-master moosefs-cgi moosefs-cgiserv moosefs-cli
    [ ${?} -ne 0 ] && failure
    apt-get install -q -y moosefs-chunkserver moosefs-metalogger
    [ ${?} -ne 0 ] && failure
    rm /etc/apt/sources.list.d/moosefs.list
    apt-get update
    [ ${?} -ne 0 ] && failure
  fi
}


do_modules() {
  if [ ! -f "/etc/modprobe.d/xt_recent.conf" ]
  then
    echo
    echo "+---- Tuning/blacklisting kernel modules"
    echo "options xt_recent ip_list_tot=2000 ip_pkt_list_tot=255" >
/etc/modprobe.d/xt_recent.conf
    chmod 644 "/etc/modprobe.d/xt_recent.conf"
  fi
}


do_base_services() {
  echo
  echo "+---- Disabling base services"
  systemctl disable ModemManager
  systemctl stop ModemManager
  systemctl disable unattended-upgrades
  systemctl stop unattended-upgrades
  systemctl disable wpa_supplicant
  systemctl stop wpa_supplicant
  if [ -f "/etc/cron.daily/apt-compat" ]
  then
    rm "/etc/cron.daily/apt-compat"
  fi
  if [ -f "/etc/cron.daily/bsdmainutils" ]
  then
    rm "/etc/cron.daily/bsdmainutils"
  fi
  if [ -f "/etc/cron.daily/ubuntu-advantage-tools" ]
  then
    rm "/etc/cron.daily/ubuntu-advantage-tools"
  fi
}


do_moosefs_services() {
  [ ${#} -ne 2 ] && failure
  iam="${1}"
  lbl="${2}"
  echo
  echo "+---- Configuring MooseFS services"
  if [ ! -d "/etc/mfs" ]
  then
    mkdir "/etc/mfs"
  fi
  chown root:root "/etc/mfs"
  chmod 755 "/etc/mfs"
  if [ ! -d /chunks ]
  then
    mkdir /chunks
  fi
  chown root:root /chunks
  chmod 755 /chunks

  grep "mfsmaster" /etc/hosts >/dev/null 2>&1
  if [ ${?} != 0 ]
  then
    echo "${mfsmaster} mfsmaster" >> /etc/hosts
  fi

  # MASTER
  #~~~~~~~~~~
  if [ -f "/etc/mfs/mfsexports.cfg.sample" ]
  then
    cp "/etc/mfs/mfsexports.cfg.sample" "/etc/mfs/mfsexports.cfg"
    chmod 644 "/etc/mfs/mfsexports.cfg"
  fi
  if [ -f "/etc/mfs/mfsmaster.cfg.sample" ]
  then
    (
      cat "/etc/mfs/mfsmaster.cfg.sample"
      echo
      echo "### SATURN ###"
      echo "DATA_PATH = /metadata/master"
      echo "NICE_LEVEL = -15"
      echo "# LOCK_MEMORY = 1"
      echo "ATIME_MODE = 4"
    ) > "/etc/mfs/mfsmaster.cfg"
    chmod 644 "/etc/mfs/mfsmaster.cfg"
  fi
  if [ ! -f "/etc/default/moosefs-master" ]
  then
    touch "/etc/default/moosefs-master"
    chmod 644 "/etc/default/moosefs-master"
  fi
```

```
  # METALOGGER
  #~~~~~~~~~~~~
  if [ -f "/etc/mfs/mfsmetalogger.cfg.sample" ]
  then
    (
      cat "/etc/mfs/mfsmetalogger.cfg.sample"
      echo
      echo "### SATURN ###"
      echo "DATA_PATH = /metadata/logger"
      echo "NICE_LEVEL = -15"
      echo "META_DOWNLOAD_FREQ = 2"
    ) > "/etc/mfs/mfsmetalogger.cfg"
    chmod 644 "/etc/mfs/mfsmetalogger.cfg"
  fi

  # CGI SERVER
  #~~~~~~~~~~~~
  if [ ! -f "/etc/default/moosefs-cgiserv" ]
  then
    touch "/etc/default/moosefs-cgiserv"
    chmod 644 "/etc/default/moosefs-cgiserv"
  fi

  # CHUNK SERVER
  #~~~~~~~~~~~~~~
  if [ -f "/etc/mfs/mfschunkserver.cfg.sample" ]
  then
    (
      cat "/etc/mfs/mfschunkserver.cfg.sample"
      echo
      echo "### SATURN ###"
      echo "NICE_LEVEL = -10"
      echo "LOCK_MEMORY = 1"
      echo "LABELS = ${lbl}"
    ) > "/etc/mfs/mfschunkserver.cfg"
    chmod 644 "/etc/mfs/mfschunkserver.cfg"
  fi
  if [ -f "/etc/mfs/mfshdd.cfg.sample" ]
  then
    cp "/etc/mfs/mfshdd.cfg.sample" "/etc/mfs/mfshdd.cfg"
    chmod 644 "/etc/mfs/mfshdd.cfg"
  fi

  # Master / Slave
  #~~~~~~~~~~~~~~~~
  if [ "${iam}" == "master" ]
  then
    echo "MFSMASTER_ENABLE=true"      > /etc/default/moosefs-master
    echo "MFSMETALOGGER_ENABLE=false" > /etc/default/moosefs-metalogger
    systemctl disable moosefs-metalogger
    systemctl enable moosefs-master
  else
    echo "MFSMASTER_ENABLE=false"     > /etc/default/moosefs-master
    echo "MFSMETALOGGER_ENABLE=true"  > /etc/default/moosefs-metalogger
    systemctl disable moosefs-master
    systemctl enable moosefs-metalogger
  fi
  echo "MFSCGISERV_ENABLE=true"        > /etc/default/moosefs-cgiserv
  echo "MFSCHUNKSERVER_ENABLE=true"    > /etc/default/moosefs-chunkserver
  systemctl enable moosefs-cgiserv
  systemctl enable moosefs-chunkserver
}


do_script_copies() {
  echo
  echo "+---- Copying other scripts"
  for scr in init-disk mount-all umount-all wireck
  do
    if [ -f "${scr}" ]
    then
```

```
      mv "${scr}" "/usr/sbin/${scr}"
      chown root:root "/usr/sbin/${scr}"
      chmod 700 "/usr/sbin/${scr}"
    fi
  done
}


do_kernel() {
  echo
  echo "+---- Kernel Tuning"
  sysctld="/etc/sysctl.d"
  sysctlf="/etc/sysctl.conf"
  if [ -f "${sysctlf}" ]
  then
    rm -f "${sysctlf}" ]
    if [ -f "${sysctlf}" ]
    then
      echo "Warning: the sysctl file could not be removed; ${sysctlf}"
      # failure
    fi
  fi
  touch "${sysctlf}"
  if [ ! -f "${sysctlf}" ]
  then
    echo "Warning: the sysctl file could not be created; ${sysctlf}"
    # failure
  fi
  chmod 644 "${sysctlf}"
  if [ -d "${sysctld}" ]
  then
    rm -rf "${sysctld}" ]
    if [ -d "${sysctld}" ]
    then
      echo "Warning: the sysctl directory could not be removed; ${sysctld}"
      # failure
    fi
  fi
  mkdir "${sysctld}"
  if [ ! -d "${sysctld}" ]
  then
    echo "Warning: the sysctl directory could not be created; ${sysctld}"
    failure
  fi
  chmod 755 "${sysctld}"
  ln -s "${sysctlf}" "${sysctld}/99-sysctl.conf"
  (
    echo "## ARM Tuning"
    echo "kernel.printk = 3 4 1 3"
    echo "vm.min_free_kbytes = 16384"
    echo "## Out-of-the-box Raspbian Hardening"
    echo "fs.protected_hardlinks = 1"
    echo "fs.protected_symlinks = 1"
    echo "## Out-of-the-box Debian Hardening"
    echo "vm.mmap_min_addr = 32768 "
    echo "kernel.kptr_restrict = 1"
    echo "kernel.yama.ptrace_scope = 1"
    echo "net.ipv4.conf.default.rp_filter=1"
    echo "net.ipv4.conf.all.rp_filter=1"
    echo "net.ipv6.conf.all.use_tempaddr = 2"
    echo "net.ipv6.conf.default.use_tempaddr = 2"
    echo "## Security / Hardening"
    echo "kernel.sysrq = 0"
    echo "net.ipv4.conf.all.accept_redirects = 0"
    echo "net.ipv4.conf.all.send_redirects = 0"
    echo "net.ipv6.conf.all.disable_ipv6 = 1"
    echo "net.ipv6.conf.default.disable_ipv6 = 1"
    echo "net.ipv6.conf.lo.disable_ipv6 = 1"
    echo "net.ipv6.conf.eth0.disable_ipv6 = 1"
    echo "net.netfilter.nf_conntrack_tcp_loose = 0"
    echo "## Performance over security"
    echo "net.ipv4.tcp_syncookies = 0"
    echo "net.ipv4.tcp_timestamps = 0"
    echo "## Network Performance"
    echo "net.ipv4.tcp_window_scaling = 1"
    echo "net.core.rmem_max = 268435456"
    echo "net.core.rmem_default = 67108864"
    echo "net.core.wmem_max = 268435456"
    echo "net.core.wmem_default = 67108864"
    echo "net.core.optmem_max = 2097152"
    echo "net.ipv4.tcp_rmem = 1048576 67108864 268435456"
    echo "net.ipv4.tcp_wmem = 1048576 67108864 268435456"
    echo "net.ipv4.tcp_mem = 1048576 67108864 268435456"
    echo "net.ipv4.udp_mem = 1048576 67108864 268435456"
    echo "net.core.netdev_max_backlog = 10000"
    echo "net.ipv4.tcp_no_metrics_save = 1"
    echo "## Memory / Cache Management"
    echo "vm.overcommit_memory = 1"
    echo "# vm.dirty_background_ratio = 3"
    echo "vm.dirty_ratio = 10"
    echo "vm.vfs_cache_pressure = 120"
  ) > "${sysctlf}"
}


do_timesync() {
  if [ ${#ntp_server} -ne 0 ]
  then
    grep "^NTP=" /etc/systemd/timesyncd.conf >/dev/null 2>&1
    if [ ${?} -eq 1 ]
    then
      echo
      echo "+---- Setting NTP server to IP address: ${ntp_server}"
      echo >> /etc/systemd/timesyncd.conf
      echo "NTP=${ntp_server}" >> /etc/systemd/timesyncd.conf
    fi
  fi
  echo
  echo "+---- Enabling network time synchronisation"
  timedatectl set-ntp true
}


do_nic() {
  [ ${#} -ne 1 ] && failure
  iam="${1}"
  if [ "${is_static}x" == "truex" ]
  then
    echo
    echo "+---- Setting static IP address: ${iam}"
    (
      echo "hostname"
      echo "clientid"
      echo "persistent"
      echo "option rapid_commit"
      echo "option domain_name_servers, domain_name, domain_search, host_name"
      echo "option classless_static_routes"
      echo "option interface_mtu"
      echo "# option ntp_servers"
      echo "require dhcp_server_identifier"
      echo "slaac private"
      echo "interface eth0"
      echo "noipv6"
      echo "nodhcp"
      echo "static ip_address=${iam}/${lan_cidr}"
      echo "static routers=${lan_gateway}"
      echo "static domain_name_servers=${name_server}"
    ) > "/etc/dhcpcd.conf"
  fi
}


do_firewall() {
  echo
  echo "+---- Setting iptables rules"
  (
    echo "*filter"
    echo ":INPUT DROP [0:0]"
    echo ":FORWARD DROP [0:0]"
    echo ":OUTPUT DROP [0:0]"
    echo "COMMIT"
  ) > "/etc/iptables/rules.v6"
  (
    echo "*filter"
    echo ":INPUT DROP [0:0]"
    echo ":FORWARD DROP [0:0]"
    echo ":OUTPUT DROP [0:0]"
    echo "-A INPUT -i lo -j ACCEPT"
    echo "-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT"
    echo "-A INPUT -m tcp -p tcp --dport 9400:9450 -m state --state NEW -j ACCEPT"
    echo "-A INPUT -m tcp -p tcp --dport 22 -s ${admin_network} -m state --state NEW -j
ACCEPT"
    echo "-A INPUT -m udp -p udp --sport 67:68 --dport 67:68 -d 255.255.255.255 -m state
--state NEW -j ACCEPT"
    echo "-A INPUT -d 224.0.0.1 -j DROP"
    echo "-A INPUT -m udp -p udp --dport 137:138 -j DROP"
    echo "-A INPUT -j LOG --log-prefix \"INPUT:DROP:\" --log-level 6"
    echo "-A INPUT -j DROP"
    echo "-A FORWARD -j LOG --log-prefix \"FORWARD:DROP:\" --log-level 6"
    echo "-A FORWARD -j DROP"
    echo "-A OUTPUT -o lo -j ACCEPT"
    echo "-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT"
    echo "-A OUTPUT -m tcp -p tcp --dport 9400:9450 -m state --state NEW -j ACCEPT"
    echo "-A OUTPUT -m udp -p udp --dport 123 -m state --state NEW -j ACCEPT"
    echo "-A OUTPUT -m udp -p udp --sport 67:68 --dport 67:68 -m state --state NEW -j
ACCEPT"
    echo "-A OUTPUT -m udp -p udp --dport 53 -m state --state NEW -j ACCEPT"
    echo "-A OUTPUT -m tcp -p tcp --dport 53 -m state --state NEW -j ACCEPT"
    echo "-A OUTPUT -j LOG --log-prefix \"OUTPUT:DROP:\" --log-level 6"
    echo "-A OUTPUT -j DROP"
    echo "COMMIT"
  ) > "/etc/iptables/rules.v4"
}


do_sshd() {
  echo
  echo "+---- Hardening sshd"
  # fix to prevent root login, aligns to Raspbian
  (
    echo "ChallengeResponseAuthentication no"
    echo "UsePAM yes"
    echo "X11Forwarding no"
    echo "PrintMotd no"
    echo "AcceptEnv LANG LC_*"
    echo "Subsystem   sftp   /usr/lib/openssh/sftp-server"
  ) > "/etc/ssh/sshd_config"
  # systemctl enable ssh
}


do_tty() {
  if [ -f "/etc/systemd/system/getty.target.wants/getty@tty1.service" ]
  then
    echo
    echo "+---- Disabling tty1"
    systemctl disable "getty@tty1"
    rm -f "/etc/systemd/system/getty.target.wants/getty@tty1.service"
  fi
}


do_unbrand() {
  [ ${#} -ne 1 ] && failure
  iam="${1}"
  if [ ! -f "/etc/issue-org" ]
  then
    echo
    echo "+---- Unbranding logon screens"
    cp /etc/issue /etc/issue-orig
    echo > /etc/issue
    cp /etc/issue.net /etc/issue.net-orig
    echo > /etc/issue.net
  fi
  echo "${iam}" > /etc/hostname
  sed -i "s/odroid/${iam}/g" /etc/hosts
  [ ${?} -ne 0 ] && failure
}


do_admin_user() {
  adm="guru"
  echo
  echo "+---- Adding admin account: ${adm}"
  # addgroup "${adm}"
  adduser "${adm}" --gecos ",,," --disabled-password
  # [ ${?} -ne 0 ] && failure
  usermod -a -G sudo "${adm}"
  [ ${?} -ne 0 ] && failure
  echo "passwd(${adm})"
  passwd "${adm}"
  [ ${?} -ne 0 ] && failure
}


do_script_iptdown() {
  scr="/usr/sbin/iptdown"
  if [ ! -x "${scr}" ]
  then
    echo
    echo "+---- Creating script iptdown: ${scr}"
    (
      echo "#!/bin/bash"
      echo
      echo "iptables -P INPUT ACCEPT"
      echo "iptables -P OUTPUT ACCEPT"
      echo "iptables -F"
    ) > "${scr}"
    chown root:root "${scr}"
    chmod 700 "${scr}"
  fi
}


do_script_scheddsk() {
  scr="/usr/sbin/sched-disk"
  if [ ! -x "${scr}" ]
  then
    echo
    echo "+---- Creating script sched-disk: ${scr}"
    (
      echo "#!/bin/bash"
      echo
      echo "for DISK in sda sdb sdc sdd sde sdf sdg sdh"
      echo "do"
      echo "  if [ -b \"/dev/\${DISK}\" ]"
      echo "  then"
      echo "    echo \"+--- Configuring scheduler for /dev/\${DISK}\""
      echo "    echo \"deadline\"   > /sys/block/\${DISK}/queue/scheduler"
      echo "    echo \"24\"         > /sys/block/\${DISK}/queue/iosched/fifo_batch"
      echo "    echo \"750\"        > /sys/block/\${DISK}/queue/iosched/read_expire"
      echo "    echo \"4\"          > /sys/block/\${DISK}/queue/iosched/writes_starved"
      echo "    echo \"128\"        > /sys/block/\${DISK}/queue/read_ahead_kb"
      echo "    echo \"128\"        > /sys/block/\${DISK}/queue/nr_requests"
      echo "  fi"
      echo "done"
    ) > "${scr}"
    chown root:root "${scr}"
    chmod 700 "${scr}"
  fi
}


do_script_tempck() {
  scr="/usr/sbin/tempck"
  if [ ! -x "${scr}" ]
  then
    echo
    echo "+---- Creating script tempck: ${scr}"
    (
      echo "#!/bin/bash"
      echo
      echo "for ZONE in 0 1 2 3 4 5 6 7"
      echo "do"
      echo "  if [ -f \"/sys/devices/virtual/thermal/thermal_zone\${ZONE}/temp\" ]"
      echo "  then"
      echo "    temp=\`cat \"/sys/devices/virtual/thermal/thermal_zone\${ZONE}/temp\"\`"
      echo "    temp=\$(( \${temp} / 1000 ))"
      echo "    alarm=\"\""
      echo "    if [ \${temp} -gt 93 ]"
      echo "    then"
      echo "      alarm=\" <-- warning\""
      echo "    fi"
      echo "    echo \"tz\${ZONE}: \${temp} \${alarm}\""
      echo "  fi"
      echo "done"
      echo "for DISK in sda sdb sdc sdd sde sdf sdg sdh"
      echo "do"
      echo "  if [ -b \"/dev/\${DISK}\" ]"
      echo "  then"
      echo "    temp=\`hddtemp -n -u c SATA:/dev/\${DISK}\`"
      echo "    if [ \${temp} -gt 63 ]"
      echo "    then"
      echo "      alarm=\" <-- warning\""
      echo "    fi"
      echo "    echo \"\${DISK}: \${temp} \${alarm}\""
      echo "  fi"
      echo "done"
    ) > "${scr}"
    chown root:root "${scr}"
    chmod 700 "${scr}"
  fi
}


do_script_tempwatch() {
  scr="/usr/sbin/tempwatch"
  if [ ! -x "${scr}" ]
  then
    echo
    echo "+---- Creating script tempwatch: ${scr}"
    (
      echo "#!/bin/bash"
      echo
      echo "watch -n2 tempck"
    ) > "${scr}"
    chown root:root "${scr}"
    chmod 700 "${scr}"
  fi
}


do_script_cpucap() {
  scr="/usr/sbin/cpucap"
  if [ ! -x "${scr}" ]
  then
    echo
    echo "+---- Creating script cpucap: ${scr}"
    (
      echo "#!/bin/bash"
      echo
      echo "if [ \"\${1}x\" == \"resetx\" -o \"\${1}x\" == \"restorex\" ]"
      echo "then"
      echo "  echo \"1500000\" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq"
      echo "  echo \"2000000\" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq"
      echo "  echo \"performance\" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor"
      echo "  echo \"performance\" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor"
      echo "else"
      echo "  echo \"1000000\" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq"
      echo "  echo \"1000000\" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq"
      echo "  echo \"ondemand\" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor"
      echo "  echo \"ondemand\" > /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor"
      echo "fi"
      echo
      echo "max0f=\`cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq\`"
      echo "max4f=\`cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq\`"
      echo "gov0f=\`cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor\`"
      echo "gov4f=\`cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor\`"
      echo "echo \"First CPU, governor and max frequency:  \${gov0f} \${max0f}\""
      echo "echo \"Second CPU, governor and max frequency: \${gov4f} \${max4f}\""
    ) > "${scr}"
    chown root:root "${scr}"
    chmod 700 "${scr}"
  fi
}


do_script_masternic() {
  scr="/usr/sbin/masternic"
  if [ ! -x "${scr}" ]
  then
    echo
    echo "+---- Creating script masternic: ${scr}"
    (
      echo "#!/bin/bash"
      echo
      echo "am_master=0"
      echo "am_slave=0"
      echo
      echo "if [ -f \"/etc/default/moosefs-master\" ]"
      echo "then"
      echo "  grep \"MFSMASTER_ENABLE=true\" \"/etc/default/moosefs-master\" >/dev/null
2>&1"
      echo "  if [ \${?} -eq 0 ]"
      echo "  then"
      echo "    am_master=1"
      echo "  fi"
      echo "fi"
      echo "if [ -f \"/etc/default/moosefs-metalogger\" ]"
      echo "then"
      echo "  grep \"MFSMETALOGGER_ENABLE=true\" \"/etc/default/moosefs-metalogger\"
>/dev/null 2>&1"
      echo "  if [ \${?} -eq 0 ]"
      echo "  then"
      echo "    am_slave=1"
      echo "  fi"
      echo "fi"
      echo
      echo "if [ \${am_master} -eq 1 -a \${am_slave} -eq 0 ]"
      echo "then"
      echo "  echo \"Assessment: This node is a master.\""
      echo "  logger \"masternic assessment: This node is a master.\""
      echo "  arping -q -c 3 -f -I eth0 mfsmaster >/dev/null 2>&1"
      echo "  if [ \${?} -eq 0 ]"
      echo "  then"
      echo "    echo \"Error: There is another node responding as master, aborting.\""
      echo "    logger \"masternic error: There is another node responding as master,
aborting.\""
      echo "    exit 1"
      echo "  else"
      echo "    echo \"Enabling master interface\""
      echo "    logger \"masternic enabling master interface\""
      echo "    ifconfig eth0:1 mfsmaster netmask 255.255.255.255"
      echo "    exit 0"
      echo "  fi"
      echo "elif [ \${am_master} -eq 0 -a \${am_slave} -eq 1 ]"
      echo "then"
      echo "  echo \"Assessment: This node is a slave.\""
      echo "  logger \"masternic assessment: This node is a slave.\""
      echo "  echo \"Disabling master interface\""
      echo "  logger \"masternic disabling master interface\""
      echo "  ifconfig eth0:1 down >/dev/null 2>&1"
      echo "  exit 0"
      echo "elif [ \${am_master} -eq 1 -a \${am_slave} -eq 1 ]"
      echo "then"
      echo "  echo \"Error: This node is both master and slave, aborting\""
      echo "  logger \"masternic error: This node is both master and slave, aborting\""
      echo "  exit 1"
      echo "elif [ \${am_master} -eq 0 -a \${am_slave} -eq 0 ]"
      echo "then"
      echo "  echo \"Error: This node is neither master nor slave, aborting\""
      echo "  logger \"masternic error: This node is neither master nor slave, aborting\""
      echo "  exit 1"
      echo "fi"
    ) > "${scr}"
    chown root:root "${scr}"
    chmod 700 "${scr}"
  fi
}


do_script_rclocal() {
  scr="/etc/rc.local"
  echo
  echo "+---- Creating script rc.local: ${scr}"
  (
    echo "#!/bin/sh -e"
    echo
    echo "for scr in cpucap masternic sched-disk wireck"
    echo "do"
    echo "  if [ -x \"/usr/sbin/\${scr}\" ]"
    echo "  then"
    echo "    nohup \"/usr/sbin/\${scr}\" < /dev/null > /dev/null 2>&1 &"
    echo "  fi"
    echo "done"
    echo
    echo "exit 0"
  ) > "${scr}"
  chown root:root "${scr}"
  chmod 755 "${scr}"
}



dev=""
if [ ${#} -ne 1 -a ${#} -ne 2 ]
then
  echo "Error: This program requires one or two paramaters."
  usage
elif [ ! ${node_ips[$1]+_} ]
then
  echo "Error: The parameter '${1}' is not recognised."
  usage
elif [ ${#} -eq 2 -a "${2}x" != "masterx" -a "${2}x" != "slavex" ]
then
  echo "Error: The parameter '${2}' is not recognised."
  usage
fi
if [ ${#} -eq 2 ]
then
  thistype="${2}"
else
  thistype="slave"
fi
```

```
thishost="${1}"
thisip="${node_ips[$1]}"
thislabel="${node_labels[$1]}"

echo
echo "Building node: ${thishost}"
echo "Building as:   ${thistype}"

do_keyboard
do_base_packages
do_base_services
do_kernel
do_modules
do_timesync
do_nic "${thisip}"
do_firewall
do_moosefs_packages
do_moosefs_services "${thistype}" "${thislabel}"
do_script_iptdown
do_script_scheddsk
do_script_cpucap
do_script_tempck
do_script_tempwatch
do_script_masternic
do_script_rclocal
do_script_copies
do_tty
do_unbrand "${thishost}"
do_admin_user
do_sshd

echo
echo "Done."
echo
echo "Run init-disk to setup the first disk."
echo
```

Script: Initialise Disk (init-disk)

This script will prepare a disk for use in Saturn – establishing a partition table, encryption, file-system and managing physical tokens for security token-based access. Although the HC2 only has one drive bay, this script has been written to support multiple drives so that it can be used on different hardware/platforms.

```
#!/bin/bash

usage() {
  bn=`basename "${0}"`
  echo
  echo "Usage: ${0} {disk device} [--raw|--gpt] [--ext|--xfs]"
  echo "    [--crypto [--blank] | --info | --tokinit | --tokadd [--slot {n}] ]"
  echo
  echo "Vanilla Examples"
  echo "  Init disk, EXT4 on raw disk:  ${bn} /dev/sdx --raw --ext"
  echo "  Init disk, XFS on part 1:     ${bn} /dev/sdx --gpt --xfs"
  echo
  echo "Crypto Examples"
  echo "  Init disk (break-glass key):  ${bn} /dev/sdx --crypto --gpt --xfs"
  echo "  Show info from existing disk: ${bn} /dev/sdx1 --crypto --info"
  echo "  Initialise token (once only): ${bn} /dev/sdx1 --crypto --tokinit"
  echo "  Enroll primary Yubikey:       ${bn} /dev/sdx1 --crypto --tokadd --slot 1"
  echo "  Enroll recovery Yubikey:      ${bn} /dev/sdx1 --crypto --tokadd --slot 2"
  echo "  Delete key material, slot 2:  ${bn} /dev/sdx1 --crypto --tokdel --slot 2"
  echo
  echo "Recommended"
  echo "  Fromat, break-glass pw: ${bn} /dev/sda --crypto --gpt --xfs"
  echo "  Enroll the Yubikey Neo: ${bn} /dev/sda1 --crypto --tokadd --slot 1"
  echo
  exit 1
}

do_gpt() {
  echo
  echo "+---- Initialising device '${1}' with GPT .."
  dd if=/dev/zero of=${1} bs=512 count=4096
  sed -e 's/\s*\([\+0-9a-zA-Z]*\).*/\1/' << EOF | gdisk "${1}"
    2  # create new GPT
    Y  # Yes, delete all partitions
    o  # clear the in memory partition table
    Y  # Yes, delete all partitions
    n  # create new partition
    1  # parition 1
       # default, first block
       # default, last block
       # default, 8300 Linux Filesytem
    p  # print the partition table
    w  # write the partition to disk
    Y  # Yes, delete all partitions
    q  # quit the program
EOF
  echo
}

do_xfs() {
  echo
  echo "+---- Initialising device '${1}' with XFS .."
  mkfs.xfs -s size=4k -f "${1}"
  echo
}

do_ext() {
  echo
  echo "+---- Initialising device '${1}' with EXT4 .."
  mke2fs -t ext4 -F -J size=256 "${1}"
  echo
}

do_metadata() {
  echo
  echo "+---- Writing metadata structure to '${1}' .."
  mkdir "${1}/metadata"
  mkdir "${1}/metadata/master"
  mkdir "${1}/metadata/logger"
  chmod -R 755 "${1}/metadata"
  chown -R mfs:mfs "${1}/metadata"
  if [ -L "/metadata" ]
  then
    echo "Warning: symlink exists, skipping metadata linkage and seeding."
  else
    ln -s "${1}/metadata" "/metadata"
    grep "MFSMASTER_ENABLE=true" "/etc/default/moosefs-master" >/dev/null 2>&1
    if [ ${?} -eq 0 ]
    then
      mv /var/lib/mfs/metadata.* "${1}/metadata/master/"
      chown mfs:mfs "${1}/metadata/master"/*
    fi
    sed -i "s#var\/lib\/mfs#metadata\/master#g" "/lib/systemd/system/moosefs-master.service"
    sed -i "s#var\/lib\/mfs#metadata\/logger#g" "/lib/systemd/system/moosefs-
metalogger.service"
  fi
}

info() {
  echo
  echo "+---- Dumping info for device '${1}' .."
  cryptsetup luksDump "${1}"
  echo
  exit 0
}

tokinit() {
  echo
  echo "+---- Initialising token position two for challenge-response .."
  ykpersonalize -2 -ochal-resp -ochal-hmac -ohmac-lt64 -ochal-btn-trig -oserial-api-visible
  echo
  exit 0
}

tokadd() {
  echo
  echo "+---- Enrolling token for device '${1}' in slot '${2}' .."
  yubikey-luks-enroll -d "${1}" -s "${2}"
  echo
  exit 0
}

tokdel() {
  echo
  echo "+---- Deleting auth material for device '${1}' in slot '${2}' .."
  cryptsetup luksKillSlot "${1}" "${2}"
  echo
  exit 0
}

if [ ! ${#} -gt 0 ]
then
  echo "Error: This program requires at least one paramater."
  usage
fi

dev=""
tin="false"
tad="false"
tde="false"
inf="false"
raw="false"
ext="false"
xfs="false"
gpt="false"
rnd="false"
cry="false"
slot="1"
while :; do
  case "${1}" in
    -b|--blank)
      rnd="true"
      ;;
    -c|--crypto)
      cry="true"
      ;;
    -e|--ext)
      ext="true"
      ;;
    -g|--gpt)
      gpt="true"
      ;;
    -h|--help)
      usage
      ;;
    -i|--info)
      inf="true"
      ;;
    -r|--raw)
      raw="true"
      ;;
    -s|--slot)
      shift
      if [ ${#2} -eq 1 -a -n "${2}" -a ${2} -lt 8 ] 2>/dev/null
      then
        slot="${2}"
      fi
      ;;
    -ta|--tokadd)
      tad="true"
      ;;
    -td|--tokdel)
```

```
      tde="true"
      ;;
    -ti|--tokinit)
      tin="true"
      ;;
    -x|--xfs)
      xfs="true"
      ;;
    *)
      if [ ${#1} -eq 0 ]
      then
        break
      fi
      dev="${1}"
      ;;
  esac
  shift
done

if [ "${dev}x" == "x" ]
then
  echo "Error: This program requires at least the device paramater."
  usage
fi

if [ "${raw}x" == "truex" -a "${gpt}x" == "truex" ]
then
  echo "Error: Raw and GPT have been selected, please select one or the other."
  usage
fi

if [ "${ext}x" == "falsex" -a "${xfs}x" == "falsex" ]
then
  if [ "${cry}x" == "falsex" ]
  then
    echo "Error: Neither EXT nor XFS has been selected, please choose one."
    usage
  else
    if [ "${raw}x" == "truex" -o "${gpt}x" == "truex" ]
    then
      echo "Error: Neither EXT nor XFS has been selected, please choose one."
      usage
    fi
  fi
fi
if [ "${ext}x" == "truex" -a "${xfs}x" == "truex" ]
then
  echo "Error: EXT and XFS have been selected, please select one or the other."
  usage
fi

if [ ! -b "${dev}" ]
then
  echo "Error: Block device '${dev}' does not exist or is invalid."
  usage
fi
dsk=`basename "${dev}"`

if [ "${inf}x" == "truex" ]
then
  if [ "${cry}x" == "falsex" ]
  then
    echo "Error: Info is a crypto function but crypto was not specified."
    usage
  fi
  info "${dev}"
fi

if [ "${tad}x" == "truex" ]
then
  if [ "${cry}x" == "falsex" ]
  then
```

```
    echo "Error: Token adding is a crypto function but crypto was not specified."
    usage
  fi
  tokadd "${dev}" "${slot}"
fi

if [ "${tde}x" == "truex" ]
then
  if [ "${cry}x" == "falsex" ]
  then
    echo "Error: Token deleting is a crypto function but crypto was not specified."
    usage
  fi
  tokdel "${dev}" "${slot}"
fi

if [ "${tin}x" == "truex" ]
then
  if [ "${cry}x" == "falsex" ]
  then
    echo "Error: Token initialisation is a crypto function but crypto was not specified."
    usage
  fi
  tokinit "${dev}"
fi

# echo $dev
# echo $raw
# echo $rnd

if [ -n "${dev: -1}" -a "${dev: -1}" -eq "${dev: -1}" ] 2>/dev/null
then
  if [ "${raw}x" == "truex" ]
  then
    echo "Error: Raw disk device selected but '${dev}' has numeric index (partition ${dev:
-1}?)"
    usage
  fi
  if [ "${gpt}x" == "truex" ]
  then
    echo "Error: GPT selected but '${dev}' has numeric index (partition ${dev: -1}?)"
    usage
  fi
else
  if [ "${gpt}x" == "falsex" -a "${raw}x" == "falsex" ]
  then
    echo "Error: partitioned device selected but '${dev}' has no numeric index (whole
disk?}"
    usage
  fi
fi

part="${dev}"
if [ "${gpt}x" == "truex" ]
then
  part="${dev}1"
fi

echo "About to erase/overwrite '${dev}' <-- are you sure? [enter twice for ok]"
read line
read line

if [ "${gpt}x" == "truex" ]
then
  echo
  echo "+---- Partitioning device '${dev}' with GPT .."
  do_gpt "${dev}"
fi

if [ "${cry}x" == "falsex" ]
then
  if [ "${ext}x" == "truex" ]
  then
    echo
    echo "+---- Writing EXT4 filesystem to '${part}' .."
    do_ext "${part}"
  elif [ "${xfs}x" == "truex" ]
  then
    echo
    echo "+---- Writing XFS filesystem to '${part}' .."
    do_xfs "${part}"
  fi
  uid=$(blkid -o value -s UUID "${part}")
  if [ ${?} -ne 0 ]
  then
    echo "Error: unable to get UUID for '${part}' (see blkid)"
    exit 2
  fi
  mnt="/chunks/${uid}"

  echo
  echo "+---- Mounting/Unmounting new filesystem '${part}' -> '${mnt}'"
  if [ ! -d "${mnt}" ]
  then
    mkdir -p "${mnt}"
    chown root:root "${mnt}"
    chmod 755 "${mnt}"
  fi
  mount "${part}" "${mnt}"
  if [ ${?} -eq 0 ]
  then
    chown mfs:mfs "${mnt}"
    chmod 755 "${mnt}"
    do_metadata "${mnt}"
    umount "${mnt}"
  else
    echo "Error: failed to mount '${part}' on '${mnt}' ..?"
    exit 1
  fi

  grep "${mnt}" /etc/fstab >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    if [ "${ext}x" == "truex" ]
    then
      echo "UUID=${uid} ${mnt} ext4 rw,noatime,nodiratime,nodev,nosuid,noexec,barrier=1 0 2"
>> /etc/fstab
    fi
    if [ "${xfs}x" == "truex" ]
    then
      echo "UUID=${uid} ${mnt} xfs
rw,noatime,nodiratime,nodev,nosuid,noexec,attr2,largeio,inode64,noquota 0 2" >> /etc/fstab
    fi
  else
    echo "Warning: ${mnt} is already in /etc/fstab, skipping"
  fi
  grep "${mnt}" /etc/mfs/mfshdd.cfg >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    echo "${mnt}" >> /etc/mfs/mfshdd.cfg
  else
    echo "Warning: ${mnt} is already in /etc/mfs/mfshdd.cfg, skipping"
  fi

else
  echo
  echo "+---- Formatting device '${part}' as LUKS device .."
  cryptsetup -v -y --cipher aes-cbc-essiv:sha256 --key-size 256 luksFormat "${part}"
  if [ ${?} -ne 0 ]
  then
    echo "Error: cryptsetup did not execute successfully (${?}), exiting."
    exit 2
  fi
  uid=$(blkid -o value -s UUID "${part}")
  if [ ${?} -ne 0 ]
  then
    echo "Error: unable to get UUID for '${part}' (see blkid)"
    exit 2
  fi
  map="${uid}"
  mnt="/chunks/${map}"

  echo
  echo "+---- Opening device '${part}' as LUKS device '/dev/mapper/${map}' .."
  cryptsetup -v luksOpen "${part}" "${map}"
  if [ ${?} -ne 0 ]
  then
    echo "Error: cryptsetup did not execute successfully (${?}), exiting."
    exit 2
  fi
  if [ ! -b "/dev/mapper/${map}" ]
  then
    echo "Error: LUKS device '/dev/mapper/${map}' does not exist or is not valid."
    exit 2
  fi

  if [ "${rnd}x" == "truex" ]
  then
    echo
    echo "+---- Zeroing new LUKS device '/dev/mapper/${map}' will take hours (8hrs/TB on
USB3) .."
    dd if=/dev/zero of="/dev/mapper/${map}" bs=128M status=progress
  fi

  if [ "${ext}x" == "truex" ]
  then
    echo
    echo "+---- Writing EXT4 filesystem to '/dev/mapper/${map}' .."
    do_ext "/dev/mapper/${map}"
  elif [ "${xfs}x" == "truex" ]
  then
    echo
    echo "+---- Writing XFS filesystem to '/dev/mapper/${map}' .."
    do_xfs "/dev/mapper/${map}"
  fi

  echo
  echo "+---- Mounting/Unmounting new filesystem '/dev/mapper/${map}' -> '${mnt}'"
  if [ ! -d "${mnt}" ]
  then
    mkdir -p "${mnt}"
    chown root:root "${mnt}"
    chmod 755 "${mnt}"
  fi
  mount "/dev/mapper/${map}" "${mnt}"
  if [ ${?} -eq 0 ]
  then
    chown mfs:mfs "${mnt}"
    chmod 755 "${mnt}"
    do_metadata "${mnt}"
    umount "${mnt}"
  else
    echo "Error: failed to mount '/dev/mapper/${map}' on '${mnt}' ..?"
    exit 1
  fi

  echo
  echo "+---- Closing LUKS device '/dev/mapper/${map}' .."
  cryptsetup luksClose "/dev/mapper/${map}"
  sync

  echo
  echo "+---- Adding LUKS device '/dev/mapper/${map}' to LizardFS as '${mnt}' .."
  if [ ! -f "/etc/crypttab" ]
  then
    touch "/etc/crypttab"
    chmod 600 "/etc/crypttab"
  fi
  grep "^${map}" /etc/crypttab >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    echo "${map} UUID=${uid} none noauto,luks,keyscript=/usr/share/yubikey-luks/ykluks-
keyscript" >> /etc/crypttab
    #### for the non-yubikey
    ## echo "${map} UUID=${uid} none luks" >> /etc/crypttab
  else
    echo "Warning: ^${map} is already in /etc/crypttab, skipping"
  fi
  grep "${mnt}" /etc/fstab >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    if [ "${ext}x" == "truex" ]
    then
      echo "/dev/mapper/${map} ${mnt} ext4
rw,noauto,noatime,nodiratime,nodev,nosuid,noexec,barrier=1 0 2" >> /etc/fstab
    fi
    if [ "${xfs}x" == "truex" ]
    then
      echo "/dev/mapper/${map} ${mnt} xfs
rw,noauto,noatime,nodiratime,nodev,nosuid,noexec,attr2,largeio,inode64,noquota 0 2" >>
/etc/fstab
    fi
  else
    echo "Warning: ${mnt} is already in /etc/fstab, skipping"
  fi
  grep "${mnt}" /etc/mfs/mfshdd.cfg >/dev/null 2>&1
  if [ ${?} -eq 1 ]
  then
    echo "${mnt}" >> /etc/mfs/mfshdd.cfg
  else
    echo "Warning: ${mnt} is already in /etc/mfs/mfshdd.cfg, skipping"
  fi
fi

echo
echo "+---- Complete"
echo "  Next steps:"
echo "      reboot"
echo "    then;"
echo "      mount-all"
# echo "        cryptdisks_start ${uid}"
# echo "        mount -a"
# echo "        systemctl restart moosefs-chunkserver"
echo

exit 0
```

Script: Mount All (mount-all)

This script simplifies the mounting of disks after boot, allowing for the presentation of physical security tokens to unlock drives and safely start the dependent MooseFS processes. Although the HC2 only has one drive bay, this script has been written to support multiple drives so that it can be used on different hardware/platforms.

```
#!/bin/bash

exec 3</dev/tty || exec 3<&0

grep -v ^# /etc/crypttab | while read line
do
  id=`echo $line | cut -f1 -d' '`
  if [ ${#id} -eq 36 ]
  then
    echo
    echo "+---- Mounting encrypted device '${id}' .."
    if [ -b "/dev/mapper/${id}" ]
    then
      echo "Warning: encrypted device already mounted, skipping."
    else
      cryptdisks_start "${args[@]}" <&3 "${id}"
      if [ ${?} -ne 0 -o ! -b "/dev/mapper/${id}" ]
      then
        echo "Error: cryptdisks_start did not execute successfully (${?}), exiting."
        exec 3<&-
        exit 2
      fi
    fi
    echo "+---- Mounting file system for '${id}' .."
    mount | grep "^/dev/mapper/${id}" >/dev/null
    if [ ${?} -eq 0 ]
    then
      echo "Warning: file system already mounted, skipping."
    else
      mount "/chunks/${id}"
      if [ ${?} -ne 0 ]
      then
        echo "Error: mount did not execute successfully (${?}), exiting."
        exec 3<&-
        exit 2
      fi
    fi
  fi
done

grep -v ^# /etc/fstab | grep -w chunks | while read line
do
  id=`echo $line | cut -f1 -d' '`
  if [ ${#id} -eq 41 ]
  then
    echo "+---- Mounting file system for '${id}' .."
    if [ -d "/chunks/${id}/lost+found" ]
    then
      echo "Warning: file system already mounted, skipping."
    else
      mount "/chunks/${id}"
      if [ ${?} -ne 0 -o ! -d "/chunks/${id}/lost+found" ]
      then
        echo "Error: mount did not execute successfully (${?}), exiting."
        exec 3<&-
        exit 2
      fi
    fi
  fi
done

echo
echo "+---- Restarting services .."
if [ -f "/etc/default/moosefs-master" ]
then
  grep "MFSMASTER_ENABLE=true" "/etc/default/moosefs-master" >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    systemctl restart moosefs-master
  fi
fi
if [ -f "/etc/default/moosefs-metalogger" ]
then
  grep "MFSMETALOGGER_ENABLE=true" "/etc/default/moosefs-metalogger" >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    systemctl restart moosefs-metalogger
  fi
fi
systemctl restart moosefs-chunkserver

echo
echo "+---- Done."

exec 3<&-
```

Script: Unmount All (umount-all)

This script simplifies the unmounting of disks before shutdown, safely shutting down the dependent MooseFS processes before unmounting drives. Although the HC2 only has one drive bay, this script has been written to support multiple drives so that it can be used on different hardware/platforms.

```
#!/bin/bash

exec 3</dev/tty || exec 3<&0

echo
echo "+---- Stopping services .."
if [ -f "/etc/default/moosefs-master" ]
then
  grep "MFSMASTER_ENABLE=true" "/etc/default/moosefs-master" >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    systemctl stop moosefs-master
  fi
fi
if [ -f "/etc/default/moosefs-metalogger" ]
then
  grep "MFSMETALOGGER_ENABLE=true" "/etc/default/moosefs-metalogger" >/dev/null 2>&1
  if [ ${?} -eq 0 ]
  then
    systemctl stop moosefs-metalogger
  fi
fi
systemctl stop moosefs-chunkserver

grep -v ^# /etc/crypttab | while read line
do
  id=`echo $line | cut -f1 -d' '`
  if [ ${#id} -eq 36 ]
  then
    echo
    echo "+---- Unmounting file system for '${id}' .."
    mount | grep "^/dev/mapper/${id}" >/dev/null
    if [ ${?} -eq 0 ]
    then
      umount "/chunks/${id}"
      if [ ${?} -ne 0 ]
      then
        echo "Error: umount did not execute successfully (${?}), exiting."
        exec 3<&-
        exit 2
      fi
    else
      echo "Warning: file system already unmounted, skipping."
    fi
    echo "+---- Unmounting encrypted device '${id}' .."
    if [ -b "/dev/mapper/${id}" ]
    then
      cryptdisks_stop "${args[@]}" <&3 "${id}"
      if [ ${?} -ne 0 -o -b "/dev/mapper/${id}" ]
      then
        echo "Error: cryptdisks_stop did not execute successfully (${?}), exiting."
        exec 3<&-
        exit 2
      fi
    else
      echo "Warning: encrypted device already unmounted, skipping."
    fi
  fi
done

grep -v ^# /etc/fstab | grep -w chunks | while read line
do
  id=`echo $line | cut -f1 -d' '`
  if [ ${#id} -eq 41 ]
  then
```

```
    echo "+---- Unmounting file system for '${id}' .."
    mount | grep "^/dev/mapper/${id}" >/dev/null
    if [ ${?} -eq 0 ]
    then
      umount "/chunks/${id}"
      if [ ${?} -ne 0 ]
      then
        echo "Error: umount did not execute successfully (${?}), exiting."
        exec 3<&-
        exit 2
      fi
    else
      echo "Warning: file system already unmounted, skipping."
    fi
  fi
done

echo
echo "+---- Done."

exec 3<&-
```

3.3.5 Activity 4: Initialise the Drive

With your storage node booted, assigned a permanent IP address, software installed and configured, you are now ready to initialise the drive.

There are two methods for drive initialisation provided in this document, manual – if you wish to perform each step yourself – and scripted – to perform a reliable process, quickly. Please choose one of these and follow those steps to complete this activity.

Note that this activity assumes you are building the HC2 node, and therefore have only one drive. However, this process is repeatable on systems with more than one drive.

3.3.5.1 Manual Initialisation Process

In case you are installing Saturn on different platform (i.e. VM guests, Docker, or other Linux distributions), or just want understand the steps in detail, then please find the manual disk initialisation steps below.

Step 1: Login to the new node

Login to the new storage node – login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

Step 2: Initialise the disk

As the *guru* user on the new storage node, use the *dd* command to blank the disk – warning that this command is **destructive**:

```
sudo dd if=/dev/zero of=/dev/sda bs=512 count=4096
```

Apply the new layout on the disk, creating the new partition table:

```
sudo gdisk /dev/sda
```

And then enter the following when prompted:

```
    2  # create new GPT
    Y  # Yes, delete all partitions
    o  # clear the in memory partition table
    Y  # Yes, delete all partitions
    n  # create new partition
    1  # parition 1
       # enter/default, first block
       # enter/default, last block
       # enter/default, 8300 Linux Filesytem
    p  # print the partition table
    w  # write the partition to disk
    Y  # Yes, delete all partitions
    q  # quit the program
```

<u>Without Whole-disk Encryption</u>

If you wish to run without whole-disk encryption, then write the new XFS file-system, onto the disk partition that you created above:

```
sudo mkfs.xfs -s size=4k -f /dev/sda1
```

Retrieve the Unique Identifier for this partition:

```
blkid -o value -s UUID "/dev/sda1"
```

Let’s assume the returned UUID value was xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx and noting this will be unique per implementation.

Now, create the permanent mount-point in the */chunks* directory, substituting this example with your UUID from the previous command:

```
sudo mkdir -p "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
sudo chown root:root "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
sudo chmod 755 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Mount the new file-system on its permanent mount-point:

```
sudo mount /dev/sda1 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

With the new file-system mounted, set its permissions:

```
sudo chown mfs:mfs "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
sudo chmod 755 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

The above two command sequences ensure that the permanent mount point exists and is unwritable for the MooseFS daemons. While the on-disk file system is able to be written to by MooseFS. This will prevent the MooseFS daemons – the Chunk Server in particular – from filling the root file-system before the data drive is mounted.

Redirect the stored metadata so that it is on the data drive:

```
sudo mkdir "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata"
sudo mkdir "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata/master"
sudo mkdir "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata/logger"
sudo chmod -R 755 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata"
sudo chown -R mfs:mfs "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata"
sudo ln -s "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata" "/metadata"
```

If this node is the master node, then move the master metadata into the new location (being careful with the line wrapping):

```
sudo mv /var/lib/mfs/metadata.* "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/metadata/master/"
sudo chown mfs:mfs "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata/master"/*
```

Now, correct the systemd configuration for the Metadata and Metalogger Servers):

```
sudo sed -i "s#var\/lib\/mfs#metadata\/master#g" "/lib/systemd/system/moosefs-
master.service"
sudo sed -i "s#var\/lib\/mfs#metadata\/logger#g" "/lib/systemd/system/moosefs-
metalogger.service"
```

Unmount the new file-system from its permanent mount-point:

```
sudo umount "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Use a text editor to add the following line to */etc/fstab* in order to automate the mount – making sure to substitute the UUID per the previous steps:

```
UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx /chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx xfs
rw,noatime,nodiratime,nodev,nosuid,noexec,attr2,largeio,inode64,noquota 0 2
```

Finally, use a text editor to add the following line to */etc/mfs/mfshdd.cfg* to add this drive to the Chunk Server:

```
/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

<u>With the Recommended Whole-disk Encryption</u>

If you wish to run with whole-disk encryption, then write the new encryption layer, onto the disk partition that you created above:

```
sudo cryptsetup -v -y --cipher aes-cbc-essiv:sha256 --key-size 256 luksFormat /dev/sda1
```

Retrieve the Unique Identifier for this partition:

```
blkid -o value -s UUID "/dev/sda1"
```

Let’s assume the returned UUID value was xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx and noting this will be unique per implementation.

Unlock and open the new encryption layer:

```
sudo cryptsetup -v luksOpen /dev/sda1 "/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Then write the new XFS file-system, onto the encrypted partition that you created above:

```
sudo mkfs.xfs -s size=4k -f "/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Now, create the permanent mount-point in the */chunks* directory, substituting this example with your UUID from the previous command:

```
sudo mkdir -p "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
sudo chown root:root "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
sudo chmod 755 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Mount the new file-system on its permanent mount-point:

```
sudo mount "/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" "/chunks/xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxxx"
```

With the new file-system mounted, set its permissions:

```
sudo chown mfs:mfs "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
sudo chmod 755 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

The above two command sequences ensure that the permanent mount point exists and is unwritable for the MooseFS daemons. While the on-disk file system is able to be written to by MooseFS. This will prevent the MooseFS daemons – the Chunk Server in particular – from filling the root file-system before the data drive is mounted. This is particularly important for encrypted volumes, as the mount process will require your input before it can proceed.

Redirect the stored metadata so that it is on the data drive:

```
sudo mkdir "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata"
sudo mkdir "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata/master"
sudo mkdir "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata/logger"
sudo chmod -R 755 "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata"
sudo chown -R mfs:mfs "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata"
sudo ln -s "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata" "/metadata"
```

If this node is the master node, then move the master metadata into the new location (being careful with the line wrapping):

```
sudo mv /var/lib/mfs/metadata.* "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/metadata/master/"
sudo chown mfs:mfs "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/metadata/master"/*
```

Now, correct the systemd configuration for the Metadata and Metalogger Servers):

```
sudo sed -i "s#var\/lib\/mfs#metadata\/master#g" "/lib/systemd/system/moosefs-
master.service"
sudo sed -i "s#var\/lib\/mfs#metadata\/logger#g" "/lib/systemd/system/moosefs-
metalogger.service"
```

Unmount the new file-system from its permanent mount-point:

```
sudo umount "/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Close and lock the new encryption layer:

```
sudo cryptsetup luksClose "/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

Prepare the special file */etc/crypttab* in case it does not exist:

```
sudo touch "/etc/crypttab"
sudo chmod 600 "/etc/crypttab"
```

Use a text editor to add the following line to */etc/crypttab* in order to automate the mount – making sure to substitute the UUID per the previous steps:

```
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx none
noauto,luks,keyscript=/usr/share/yubikey-luks/ykluks-keyscript
```

Then, use a text editor to add the following line to */etc/fstab* in order to automate the mount – making sure to substitute the UUID per the previous steps:

```
/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx /chunks/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx xfs
rw,noauto,noatime,nodiratime,nodev,nosuid,noexec,attr2,largeio,inode64,noquota 0 2
```

Finally, use a text editor to add the following line to */etc/mfs/mfshdd.cfg* to add this drive to the Chunk Server:

```
/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

The encryption process will require what is known as a “Break Glass” (or Firecall) key – a passphrase that can be used to unlock the encrypted volume. It is recommended that this passphrase only be used in case of emergencies.

For routine operations, a primary physical security token should be used, via LUKS key slot 1 (i.e. in addition to the pass phrase in slot 0):

```
yubikey-luks-enroll -d /dev/sda1 -s 1
```

A secondary (or backup) physical security token should also be enrolled, via LUKS key slot 2:

```
yubikey-luks-enroll -d /dev/sda1 -s 2
```

Please see section 4.2.1 Encryption and Key Management, below, for further detail.

Step 3: Complete

At this point, if you have successfully followed all of the above build activities, then your storage node will be complete and ready to use. As the root user on the new storage node, reboot the storage node:

```
sudo reboot
```

If you have chosen the recommended whole-disk encryption then on restart the node will require your physical security token to unlock the drive. Please see the Operations chapter on unlocking a drive.

3.3.5.2 Scripted Initialisation Process

Through Activity 3, the *init-disk* script has already been installed on the storage node. It will help to make the disk initialisation a simple task, as well as to include some safe-guards for those late night deployments. These scripts may not be appropriate for your environment so please check them before you use them, and adjust them where necessary.

Step 1: Login to the new node

Login to the new storage node – login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

Step 2: Initialise the disk

As the *guru* user on the new storage node, use the *init-disk* script to blank the disk and layout the new file-system.

If you wish to run without whole-disk encryption, then initialise the disk as follows:

```
sudo init-disk /dev/sda --gpt --xfs
```

If you wish to run with the recommended whole-disk encryption, then initialise the disk as follows:

```
sudo init-disk /dev/sda --crypto --gpt --xfs
```

The encryption process will require what is known as a “Break Glass” (or Firecall) key – a passphrase that can be used to unlock the encrypted volume. It is recommended that this passphrase only be used in case of emergencies.

For routine operations, a primary physical security token should be used, via LUKS key slot 1 (i.e. in addition to the pass phrase in slot 0):

```
sudo init-disk /dev/sda1 --crypto --tokadd --slot 1
```

A secondary (or backup) physical security token should also be enrolled, via LUKS key slot 2:

```
sudo init-disk /dev/sda1 --crypto --tokadd --slot 2
```

Please see section 4.2.1 Encryption and Key Management, below, for further detail.

Step 3: Complete

At this point, if you have successfully followed all of the above build activities, then your storage node will be complete and ready to use.

As the root user on the new storage node, reboot the storage node:

```
sudo reboot
```

If you have chosen the recommended whole-disk encryption then on restart the node will require your physical security token to unlock the drive. Please see the Operations chapter on unlocking a drive.

4 Operation

With your storage node booted, assigned a permanent IP address, software installed and configured, and drive initialised, you are ready to operate the node and the storage cluster.

Before reading this section, please make sure you understand the key concepts per section 3.3.1 Key Concepts, above.

4.1 Administratively Accessing Saturn

There are three ways to administratively access the Gen2 Saturn environment. Select the admin interface, depending on what you are attempting to achieve.

Note: Do not expose any of these services to the Internet, or any other untrusted environments. If you require remote access to these services then ensure they are behind additional filtering, authentication and encryption layers (i.e. firewalls and VPN).

From a Remote Shell (SSH)

To login to any storage node – use *ssh* to login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

If privileged access is required then use *sudo* to perform that action:

```
sudo reboot
```

Accessing each storage node this way provides access to the kernel, filesystem and raw network layers – useful for configuration and diagnostics. This is also the first layer of the storage software configuration (most configuration items are managed via files in */etc/mfs* on each storage node, for example).

From the Web (Browser)

To login to the cluster – use *http* to the *mfsmaster* node on port 9425/TCP, to see the state of the entire environment:

```
http://192.168.1.111:9425
```

This interface provides the best monitoring view of the MooseFS software, including the state of replication, node availability, disk IO, network IO, CPU and memory consumption.

From the MooseFS Command Line Interface (CLI)

The CLI is deployed to the administrative workstation. The most commonly deployed Saturn environments, for the author, is currently Debian based, so this section describes a Debian administrative workstation deployment. Note, however, that there are other packages available – for FreeBSD, OSX, and RedHat (see also the MooseFS install guide[53], and the MooseFS repo[54]).

To add the MooseFS CLI to a Debian-based workstation / server, add the MooseFS apt repository key to the local repo, as root:

```
wget -O - "https://ppa.moosefs.com/moosefs.key" | apt-key add -
```

To find your Debian version:

```
cat /etc/debian_version
```

53 https://moosefs.com/Content/Downloads/moosefs-installation.pdf
54 http://ppa.moosefs.com/moosefs-3/

Use that information to find and select the appropriate MooseFS repo:

- Ubuntu
  - 20.04 Focal: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/focal focal main
  - 18.04 Bionic: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/bionic bionic main
  - 16.04 Xenial: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/xenial xenial main
  - 14.04 Trusty: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/trusty trusty main
  - 12.10 Quantal: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/quantal quantal main
  - 12.04 Precise: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/precise precise main
  - 10.10 Maverick: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/maverick maverick main
  - 10.04 Lucid: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/lucid lucid main
- Debian
  - 10.0 Buster: deb http://ppa.moosefs.com/moosefs-3/apt/debian/buster buster main
  - 9.0 Stretch: deb http://ppa.moosefs.com/moosefs-3/apt/debian/stretch stretch main
  - 8.0 Jessie: deb http://ppa.moosefs.com/moosefs-3/apt/debian/jessie jessie main
  - 7.0 Wheezy: deb http://ppa.moosefs.com/moosefs-3/apt/debian/wheezy wheezy main
  - 6.0 Squeeze: deb http://ppa.moosefs.com/moosefs-3/apt/debian/squeeze squeeze main
  - 5.0 Lenny: deb http://ppa.moosefs.com/moosefs-3/apt/debian/lenny lenny main
- Raspbian
  - 9.0 Stretch: deb http://ppa.moosefs.com/moosefs-3/apt/raspbian/stretch stretch main
  - 8.0 Jessie: deb http://ppa.moosefs.com/moosefs-3/apt/raspbian/jessie jessie main

Use a text editor to create */etc/apt/sources.list.d/moosefs.list* with the appropriate repo:

```
deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/focal focal main
```

Ensure that the apt data is up to date:

```
apt-get update
```

Install the MooseFS CLI package:

```
apt-get install -q -y moosefs-cli
```

If you wish to disable this repo again until you're ready to do patching then, with the MooseFS package installed, remove the unneeded repo and update the apt data:

```
rm /etc/apt/sources.list.d/moosefs.list
apt-get update
```

The MooseFS CLI tools are there to manage and monitor the MooseFS file-system. Please see the section 4.3 Storage Operations & Maintenance, below, for further detail.

4.2 Device Operations & Maintenance

This section addresses common administrative functions of the storage node infrastructure. Note that this section only covers basic use, and that additional resources are available for more complex deployments – see also the MooseFS User's Manual[55] for more detail.

4.2.1 Encryption and Key Management

Saturn leverages LUKS – the Linux Unified Key Setup[56] – for its whole-disk encryption implementation. It is beyond the scope of this document to provide a primer on all of the features of LUKS, so please find additional reading online. The most important concept to understand is *key slots*.

Understanding LUKS Key Slots

LUKS allows up up to eight (8) keys to unlock the vault to the key that decrypts the volume. It is significant to note that the user-entered keys do not themselves decrypt the volume; they provide access to the master key that decrypts the volume.

The important process/configuration touch-points are:

- Each key is stored in a *key slot* numbered zero (0) to seven (7).
- When you first create an encrypted volume, you're asked to enter a key phrase – this is stored in *slot 0*.
- To add or remove a key to/fro one of the key slots, requires the entry of an existing key.

So think of the the eight key slots in LUKS as eight different encrypted versions of the same master key, under eight different passwords. Because of this architecture LUKS has incredibly flexible key management, allowing Saturn to build-in both pass-phrase and physical security token support.

Understanding the YubiKey, HMAC-SHA1 Challenge-Response and Programmable Slots

The company Yubico[57] makes and sells a broad portfolio of physical security token devices – known as *YubiKeys*[58]. Each YubiKey device has been implemented with a multitude of protocols to try and maximise their versatility.

The Saturn build relies upon the Debian packages of *yubikey-personalization* and *yubikey-luks* to provide hardware security token services to LUKS. Combined, these tool-sets requires specific features of the YubiKey token product, and will therefore affect product selection.

<u>YubiKey Personalization</u> [59] [60]

The YubiKey Personalization package contains a library and command line tool used to configure YubiKeys. *Personalization* in this case means:

1. Setting a new AES secret key on the YubiKey, and;
2. Configuring the second slot on the YubiKey (not to be confused with the LUKS key slots) to perform Challenge-Response[61].

This is not all that the *ykpersonalize* program is capable of – a full set of parameters and their setting values can be found in the YubiKey Manual[62].

55 https://moosefs.com/Content/Downloads/moosefs-3-0-users-manual.pdf
56 https://en.wikipedia.org/wiki/Linux_Unified_Key_Setup
57 https://www.yubico.com/
58 https://www.yubico.com/products/
59 https://developers.yubico.com/yubikey-personalization/
60 https://github.com/Yubico/yubikey-personalization
61 https://www.yubico.com/products/services-software/personalization-tools/challenge-response/
62 https://www.yubico.com/wp-content/uploads/2015/03/YubiKeyManual_v3.4.pdf

HMAC-SHA1 Challenge-Response

An HMAC is a Hash-based Message Authentication Code[63] – it takes two inputs, a key and data, to calculate a hash. In the case of the Yubikey, and the *ykchalresp* program, it takes user input as the data and sends it to the YubiKey such that the AES key (set above) doesn't leave the physical token.

By way of example, the Challenge-Response process works as follows:

1. User enters their passphrase.
2. The *ykchalresp* program sends that passphrase to the YubiKey.
3. The YubiKey performs the HMAC using the passphrase as input and the internal AES secret as the key.
4. The resulting hashed value is sent back to the application.

It is this hashed value that is provided to the LUKS key slot as a credential.

Selecting a YubiKey physical security token

Not all YubiKey products have the second programmable slot, and not all of the YubiKey products work with *ykpersonalize*. Between the two support pages published by Yubico at the time of writing (i.e. the *Personalization Tool Guide*[64] and the *Linux Challenge Response Guide*[65]), the following YubiKeys have been extracted as likely viable physical tokens for this build:

Edition \ Version	Standard	NEO	Edge	4	5	FIPS
Base	YubiKey Standard	YubiKey NEO	YubiKey Edge	YubiKey 4		YubiKey FIPS
Nano	YubiKey Nano	YubiKey NEO-n	YubiKey Edge-n	YubiKey 4 Nano	YubiKey 5 Nano	YubiKey Nano FIPS
USB C				YubiKey 4C	YubiKey 5C	YubiKey C FIPS
USB C Nano				YubiKey 4C Nano	YubiKey 5C Nano	YubiKey C Nano FIPS
USB C & Lightning					YubiKey 5Ci	
NFC					YubiKey 5 NFC	

Please note that this build has only been tested and validated with the *YubiKey NEO*.

Native Implementation

Initialising the YubiKey token with programmable slot two (2) for challenge-response, and with flashing LED for challenge notification and button press to trigger the response:

```
ykpersonalize -2 -ochal-resp -ochal-hmac -ohmac-lt64 -ochal-btn-trig -oserial-api-visible
```

Enrolling the Yubikey token for device /dev/sda1 in LUKS key slot 1 (i.e. in addition to the pass phrase in slot 0):

```
yubikey-luks-enroll -d /dev/sda1 -s 1
```

Deleting the authentication material (removing the YubiKey hash) for device /dev/sda1 in LUKS key slot 1:

```
cryptsetup luksKillSlot /dev/sda1 1
```

63 https://en.wikipedia.org/wiki/Hash-based_message_authentication_code
64 https://support.yubico.com/support/solutions/articles/15000006424-yubikey-personalization-tool-user-guide
65 https://support.yubico.com/support/solutions/articles/15000011355-ubuntu-linux-login-guide-challenge-response

Saturn Implementation

The script */usr/sbin/init-disk* has been created to simplify disk initialisation and key management.

To run the script, login to any storage node – use *ssh* to login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

Then use *sudo* to run *init-disk.*

Initialising the YubiKey token with programmable slot two (2) for challenge-response, and with flashing LED for challenge notification and button press to trigger the response:

```
sudo init-disk /dev/sda1 --crypto --tokinit
```

Enrolling the Yubikey token for device /dev/sda1 in LUKS key slot 1 (i.e. in addition to the pass phrase in slot 0):

```
sudo init-disk /dev/sda1 --crypto --tokadd --slot 1
```

Deleting the authentication material (removing the YubiKey hash) for device /dev/sda1 in LUKS key slot 1:

```
sudo init-disk /dev/sda1 --crypto --tokdel --slot 1
```

Security and Availability Considerations

There are some additional considerations that should be factored as part of your risk assessment and process development activities:

1. **LUKS version 1** – This build guide includes steps that deploy a LUKS version 1 container. Ensure that LUKS version 1 suits your needs before deploying this solution for production data. The included scripts should work with LUKS version 2, but have not been tested/validated.
2. **Losing keys costs data** – Be aware that it is entirely possible to to lose all of your data simply by losing/forgetting the keys that open LUKS.

   As it is recommended that disk encryption be deployed, it is therefore strongly recommended that you deploy:

   - A long and complex pass-phrase as the break-glass (fire-call) key, in slot 0, and;
   - A primary physical security token (YubiKey) with pass-phrase hash key, in slot 1, and;
   - A secondary physical security token (Yubikey) with pass-phrase hash key, in slot 2.
3. **Losing integrity costs a node** – Also, be aware that the whole-disk encryption underpins MooseFS on each Chunk Server. This means that if there is disk corruption / bit-rot that prevents the encrypted disk from being accessed, then the entire Chunk Server will be lost.

   This build document produces storage capability with two copies of each chunk – meaning that the loss of any one node won’t lose the data. And the loss of one node will cause the cluster to automatically redistribute chunks (where capacity is available) in order to mitigate a second failed node, in time.

   Be sure to include this in your planning before deploying this solution for production data.
4. **Not 2FA/MFA** – Note that this guide does not refer to the YubiKey authentication mechanism as either two-factor authentication (2FA) or multi-factor authentication (MFA). If an adversary were to obtain the hash that is emitted from the *ykchalresp* program, then that hash alone could be used as a password to unlock the given LUKS key slot. Neither of the two factors the user sees (the Yubikey pass phrase and the Yubikey physical security token) need to be presented to successfully authenticate.

Note, too, that a USB extension cable (Male-to-Female, of any length) is required to use the YubiKey, as the placement of the USB port on the HC2 obscures the YubiKey button with the DC power plug.

4.2.2 Performance Management

There are two areas where performance tuning has been focused, network and disk IO.

Network Performance Tuning

Per the installation process there is set of configuration items written to */etc/sysctl.conf* to tune the kernel. The following are the key network performance configuration items.

```
## Performance over security
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_timestamps = 0
## Network Performance
net.ipv4.tcp_window_scaling = 1
net.core.rmem_max = 268435456
net.core.rmem_default = 67108864
net.core.wmem_max = 268435456
net.core.wmem_default = 67108864
net.core.optmem_max = 2097152
net.ipv4.tcp_rmem = 1048576 67108864 268435456
net.ipv4.tcp_wmem = 1048576 67108864 268435456
net.ipv4.tcp_mem = 1048576 67108864 268435456
net.ipv4.udp_mem = 1048576 67108864 268435456
net.core.netdev_max_backlog = 10000
net.ipv4.tcp_no_metrics_save = 1
```

The goal was to maximise throughput, though this was originally developed for LizardFS – see the Gen2 prototype discussion in section 2.2.2 Daphnis (2019 –), above. As such these settings were initially based on, and scaled down from, the LizardFS tuning guide[66]. From there the settings were carried forward as part of the baseline performance evaluation between MosseFS and LizardFS, but were never validated as *necessary* to maximise throughput on a 1Gbps network for MooseFS.

By configuration item:

- **net.ipv4.tcp_syncookies**

  SYN Cookies are a defence against SYN floods. However, this is disabled to allow selective acknowledgements to be used to reduce latency and improve throughput. This could probably be enabled as it only takes affect once *net.ipv4.tcp_max_syn_backlog* has been reached.

- **net.ipv4.tcp_timestamps**

  TCP Time Stamps are required for SYN Cookies but are otherwise not recommended for security reasons (they leak the node time state).

- **net.ipv4.tcp_window_scaling**

  TCP Window Scaling allows the TCP receive window to exceed 64KB, enabling higher throughput[67].

- **net.core.rmem_max**
  - Sets the maximum receive buffer size for all types of network connections (limit setsockopt to 268,435,456 bytes or 65,536 pages).
- **net.core.rmem_default**
  - Sets the default receive buffer size for all types of network connections (default setsockopt to 67,108,864 bytes or 16,384 pages).
- **net.core.wmem_max**
  - Sets the maximum write buffer size for all types of network connections (limit setsockopt to

66 https://docs.lizardfs.com/cookbook/linux.html#basic-network-adjustments-for-linux
67 https://en.wikipedia.org/wiki/TCP_window_scale_option

268,435,456 bytes or 65,536 pages).

- **net.core.wmem_default**

  Sets the default write buffer size for all types of network connections (default setsockopt to 67,108,864 bytes or 16,384 pages).

- **net.core.optmem_max**

  Proportionally scale out the per socket ancillary buffer (to 2,097,152 bytes)[68].

- **net.ipv4.tcp_mem**

  Defines the TCP stack memory usage behaviour in three values. The first value specifies the low threshold, below this point, there is no memory usage pressure. The second value is the threshold for pressuring memory usage down. The final value defines the maximum number of pages at which point TCP streams / packets start getting dropped until reaching the lesser threshold. These values have been set to 1,048,576 bytes, 67,108,864 bytes and 268,435,456 bytes respectively, or 256 pages, 16,384 pages and 65,536 pages respectively.

- **net.ipv4.udp_mem**

  Defined the UDP stack memory usage, per above (1,048,576 bytes, 67,108,864 bytes and 268,435,456 bytes respectively, or 256 pages, 16,384 pages and 65,536 pages respectively).

- **net.ipv4.tcp_rmem**

  Defined the TCP read stack memory usage, per above (1,048,576 bytes, 67,108,864 bytes and 268,435,456 bytes respectively, or 256 pages, 16,384 pages and 65,536 pages respectively).

- **net.ipv4.tcp_wmem**

  Defined the TCP write stack memory usage, per above (1,048,576 bytes, 67,108,864 bytes and 268,435,456 bytes respectively, or 256 pages, 16,384 pages and 65,536 pages respectively).

- **net.core.netdev_max_backlog**

  The number of new inbound packets that will be managed (10,000)

- **net.ipv4.tcp_no_metrics_save**

  By default, TCP saves various connection metrics in the route cache when the connection closes, so that connections established in the near future can use these to set initial conditions. Usually, this increases overall performance, but may sometimes cause performance degradation. When set, TCP will not cache metrics on closing connections.

Network Performance Testing

These settings were validated for raw TCP performance with iperf3, which was included in the build process. To validate any network performance changes (in TCP), use the following on the *server* end of the test:

```
iperf3 -s -p 9401
```

And use the following on the *client* end of the test:

```
iperf3 -c <server ip> -p 9401
```

Note that the port number is not significant to the test, but if you have applied the firewall rules then this configuration will leverage an available TCP port.

You should see line rate transfers between the HC2s as well as clients and the HC2s (averages of 900Mbps or higher).

68 https://man7.org/linux/man-pages/man3/cmsg.3.html

This type of performance testing can be used to isolate node/LAN performance issues from other performance issues (such as disk IO and software issues).

Disk Performance Tuning

Per the installation process there is set of configuration items written to */etc/sysctl.conf* to tune the kernel. The following are the key disk performance configuration items.

```
## Memory / Cache Management
vm.overcommit_memory = 1
# vm.dirty_background_ratio = 3
vm.dirty_ratio = 10
vm.vfs_cache_pressure = 120
```

The objective of these tuning parameters was to put appropriate pressure on the cache memory to provide a balance between caching and available memory. They have been constructed on the basis that the disk IO scheduler is the *deadline* scheduler.

- **vm.dirty_ratio**

  This value represents the percentage of total system memory at which dirty pages created by application disk writes will be flushed out to disk. Until this percentage is reached no pages are flushed to disk. A value of 10 mean that data will be written into system memory until the file system cache has a size of 10% of the server's RAM.

- **vm.vfs_cache_pressure**
  - At the default value of vfs_cache_pressure=100 the kernel will attempt to reclaim dentries and inodes at a "fair" rate with respect to pagecache and swapcache reclaim. Decreasing vfs_cache_pressure causes the kernel to prefer to retain dentry and inode caches. When vfs_cache_pressure=0, the kernel will never reclaim dentries and inodes due to memory pressure and this can easily lead to out-of-memory conditions. Increasing vfs_cache_pressure beyond 100 causes the kernel to prefer to reclaim dentries and inodes. If you have a lot of small files, then consider increasing this value beyond 100. When working with small numbers of larger files, the lower value should see higher performance.

- **vm.overcommit_memory**

  The MooseFS Metadata Server forks to write a copy of its memory database to disk. That process fork implies, to the kernel, that the memory uses will be duplicated (effectively doubles the amount of memory to be used by maetadata processes). In practice, the fork command does not copy the entire memory, only the modified fragments are copied as needed. As the child process in the metadata fork only reads the memory and dumps it to disk, not much memory content will change. Therefore setting this will prevent the error *fork error (store data in foreground - it will block master for a while)* in low memory situations (see the support[69] page – and note that this is a stability setting, not a performance setting).

The disk scheduler can be identified by probing the kernel, per drive:

```
cat /sys/block/sda/queue/scheduler
```

The default scheduler in the Ubuntu distribution is *cfq*, which is a fair broad-purpose scheduler. For the traditional non-flash / magnetic drives in a dedicated storage system, the better scheduler is *deadline* as it is time based and will reduce latency.

69 https://moosefs.com/support/

The script */usr/sbin/sched-disk* has been written to set the scheduler and key configuration items, per drive:

```
echo "deadline"   > /sys/block/sda/queue/scheduler
echo "24"         > /sys/block/sda/queue/iosched/fifo_batch
echo "750"        > /sys/block/sda/queue/iosched/read_expire
echo "4"          > /sys/block/sda/queue/iosched/writes_starved
echo "128"        > /sys/block/sda/queue/read_ahead_kb
echo "128"        > /sys/block/sda/queue/nr_requests
```

Note that the installation procedure adds execution of *sched-disk* to the startup script */etc/rc.local* to ensure that the attached drives are configured consistently across reboots. The script was not designed to be used interactively so it does produce output.

Note also that SSDs should use the *noop* scheduler – none have been tested in this build, but more detail can be found online[70].

Disk Performance Testing

There is an art to disk performance testing that won't be explained in this manual. However some basic measures can be taken for linear operations that will provide rough diagnostics for disk performance.

Part of the difficulty of performance testing on the HC2 is that it doesn't have much in the way of external connectivity. The network has capacity for 125MBps (1Gbps), but the hard drive should have capacity of 250MBps (2Gbps). The USB interface is only USB2, and adding USB drives to store test files will add latency. The use of */dev/zero* to create files faster, at 1.7GBps (13.6Gbps) than the 250MBps (2Gbps) is fine, except that Linux can identify sparse files and store them more efficiently than data files. And then the use of /dev/urandom for noisy data is slower at 40MBps (320Mbps) than the test device.

In each of the following cases, the dd command is used, and it returns the average speed in MBps.

Raw Disk

To test raw write disk IO, on a new storage node (with an unformatted drive), write 10GB to disk (noting that this is **destructive**):

```
dd if=/dev/zero of=/dev/sda bs=64M count=160
```

To test raw read disk IO, on a new storage node (with an unformatted drive), read 10GB from disk:

```
dd if=/dev/sda of=/dev/null bs=64M count=160
```

These performance figures should closely match your drive specifications (manufacturer's specs – read/write at approximately 250MBps for the Seagate ST16000VE000, even with CPU performance limited).

Disk Encrypted

To test encrypted write disk IO, on a new storage node (with an initialised drive), write 10GB to disk (noting that this is **destructive**):

```
dd if=/dev/zero of=/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx bs=64M count=160
```

To test encrypted read disk IO, on a new storage node (with an initialised drive), read 10GB from disk:

```
dd if=/dev/mapper/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx of=/dev/null bs=64M count=160
```

These performance figures will reflect the ability of the HC2 to perform encryption (write at approximately 110MBps and read at approximately 50MBps with CPU performance limited; or write at approximately 150MBps and read at approximately 70MBps with CPU performance unbounded).

70 https://wiki.debian.org/SSDOptimization#Low-Latency_IO-Scheduler

Encrypted File-system

To test encrypted write file-system IO, on a new storage node (with an initialised drive), write 10GB to a file on the mounted partition:

```
dd if=/dev/zero of=/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/10GB.bin bs=64M count=160
```

To test encrypted read file-system IO, on a new storage node (with an initialised drive), read the 10GB from a file on the mounted partition:

```
dd if=/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/10GB.bin of=/dev/null bs=64M count=160
```

These performance figures will reflect the ability of the HC2 to perform encryption (write at approximately 105MBps and read at approximately 40MBps with CPU performance limited; or write at approximately 140MBps and read at approximately 55MBps with CPU performance unbounded).

While your figures may vary, this will provide some indication of the node performance before adding the MooseFS storage layer.

4.2.3 Thermal Management

As noted in the Gen2 prototype discussion in section 2.2.2 Daphnis (2019 –), above, with whole-disk encryption and long duration high-throughput transfers the HC2 had temperature (overheating) issues in a passively cooled environment.

The two scripts written to help with the issue were a temperature monitoring script, and a CPU performance limiting script.

Checking Thermal Limits (tempwatch)

There are five (5) thermal zones, numbered zero (0) to four (4) on the HC2, and their temperature can be probed by using *cat* to read the value from of the kernel:

```
cat /sys/devices/virtual/thermal/thermal_zone0/temp
```

Dividing that number by 1000 give the current temperature in degrees Celsius (ºC).

For the disk drive, the *hddtemp* program will return the current drive temperature in ºC:

```
hddtemp -n -u c SATA:/dev/sda
```

Both of these probe types were codified within the script */usr/sbin/tempck*, and wrapped in the refreshing script */usr/sbin/tempwatch* which runs tempck every 2 seconds to provide the administrator with a current visual assessment of the thermal condition of the node.

Based on the kernel source for the HC2 and the Seagate drive specifications, alarms were set at 93ºC for the five (5) HC2 thermal zones (on the assumption that the power cycling occurred in the 99-109ºC ball-park), and 63ºC for the hard drive (with a 70ºC operating limit).

To run the script, login to any storage node – use *ssh* to login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

Then *sudo* to run *tempwatch*:

```
sudo tempwatch
```

That script will output the current temperature values in ºC:

```
tz0: 46
tz1: 48
tz2: 51
tz3: 47
tz4: 43
sda: 39
```

If the temperature is excessive it will notify you as follows:

```
tz0: 84
tz1: 88
tz2: 94  <-- warning
tz3: 89
tz4: 85
sda: 67  <-- warning
```

Managing Temperature (cpucap)

To manage the temperature without adding active cooling, in addition to the physical deployment (which ensures greater circulation) the CPU performance profile was changed to limit how much heat it could generate.

CPU scaling figures can be probed and set to/fro the kernel. Out of the box, the Ubuntu distribution maximises performance. i.e. the four (4) cores of the Cortex-A7 (cpu0) are set to the *performance* scaling governor and a *1.5Ghz* scaling max frequency:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

While the four (4) cores of the Cortex-A15 (cpu4) are also set to the *performance* scaling governor and a *2.0Ghz* scaling max frequency:

```
cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_governor
cat /sys/devices/system/cpu/cpu4/cpufreq/scaling_max_freq
```

Essentially all eight (8) cores are pinned to their top speed.

The CPU can be managed by setting all eight (8) cores to the *ondemand* scaling governor, and a scaling max frequency of *1.0Ghz*, allowing the CPU to burst to higher speeds as needed but not exceeding 1Ghz per core.

Both of these states have been codified in the */usr/sbin/cpucap* script.

To run the script, login to any storage node – use *ssh* to login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

Then *sudo* to run *cpucap*:

```
sudo cpucap
```

The script will change the current CPU capacity to the limited settings and show you the kernel state:

```
First CPU, governor and max frequency:  ondemand 1000000
Second CPU, governor and max frequency: ondemand 1000000
```

If you run *sudo* with *cpucap reset*, the default high-performance settings will be restored:

```
sudo cpucap reset
```

With the output reflecting the change to the kernel state:

```
First CPU, governor and max frequency:  performance 1500000
Second CPU, governor and max frequency: performance 2000000
```

Note that the installation procedure adds execution of *cpucap* to the startup script */etc/rc.local* to ensure that the performance is limited consistently across reboots.

CPU Performance Testing

The testing for this was the transfer of a 10GB test file to the presented storage, using *rsync* (which was slightly slower than *cp*). Repeated testing confirmed that the difference in performance was not as significant as one would expect – rsync showed an average speed of 80MBps for uploads with full processor capacity, and an average speed of 75MBps for uploads with processor capacity limited to 1Ghz.

Observation on the MooseFS *http* based monitoring console showed, interestingly, that raw disk IO was much more significantly impacted – with disk write speed being approximately 250Mbps with full processor capacity, and approximately 180MBps with processor capacity limited to 1Ghz.

Comparatively, there was an average of 15ºC taken off the thermal zone peak temperatures. So for the small reduction in the user presented write performance, by an average of 5MBps (40Mbps) – now at about 75MBps (600Mbps), this was considered a reasonable outcome.

4.3 Storage Operations & Maintenance

This section addresses common administrative functions of the storage pool that is presented to the clients. Note that this section only covers basic use, and that additional resources are available for more complex deployments – such as those that use quotas and snapshots – see the MooseFS User's Manual[71] for more detail.

4.3.1 Storage Tiers and Storage Policy

MooseFS has a labelling and policy scheme that it refers too as *storage classes*. There is a native MooseFS manual (the MooseFS Storage Classes Manual[72]) dedicated entirely to storage, so the topic will only be covered lightly here. For reference, this section is going to deliver the MooseFS manual example of *Scenario 1: Two server rooms (A and B).*

There are two prerequisites before progressing with this section:

1. On your administrative workstation, make sure you have installed the Client (see section 4.5.1 Installing the Client, below)
2. Also on your administrative workstation, make sure you have installed the CLI (see section 4.1 Administratively Accessing Saturn, above)

This manual set out to produce highly available storage – two shelves of storage where one shelf is mirrored to/fro the other. Out of the box, MooseFS will write two copies of each chunk, and distribute chunks between nodes based on available storage capacity. This is an excellent model for high availability but it lacks predictability. In order to align physical availability features, such as switch ports and uninteruptable power supplies (UPS), we need to know which group of physical devices will always have "the other copy".

As a part of the node installation process (see section 3.3.4 Activity 3: Install and Configure the Software, above; Step 10, Chunk Server sub-section) the nodes were grouped into two classes –

- Nodes c1, c2, c3, c4 and c5 were give the label "A", and;
- Nodes c6, c7, c8, c9 and c10 were given the label "B".

All of the "A" nodes are to be deployed to one location (with its own power and network infrastructure), while all of the "B" nodes are to be deployed to a separate independent location.

Defining Policy

To ensure that one complete copy of the data set is available when either site goes offline, we need to create a storage class that defines that policy intent. Let's call this storage class *Gold* - as we're going to give this data Gold Class treatment.

First, as *root* on the administrative workstation, create the mount point:

```
mkdir /mnt/daphnis
```

Then mount the file-system:

```
mount -t moosefs 192.168.1.111:/ /mnt/daphnis
```

Now, define the *Gold* storage class as requiring one chunk on an A node and another chunk on a B node:

```
mfsscadmin /mnt/daphnis create A,B Gold
```

71 https://moosefs.com/Content/Downloads/moosefs-3-0-users-manual.pdf
72 https://moosefs.com/Content/Downloads/moosefs-storage-classes-manual.pdf

Finally, apply the *Gold* storage class to the root of that volume:

```
mfssetsclass Gold /mnt/daphnis
```

Testing Policy

At this point, if you visit the Web based administration interface you will find the Gold class listed as id 10 on the Resources page:

```
http://192.168.1.111:9425/mfs.cgi?sections=RS
```

Further, you can test this configuration by copying a file on to the presented file-system:

```
dd if=/dev/urandom of=/mnt/daphnis/1GB.bin bs=64M count=16 iflag=fullblock
```

You can see where the chunks for this file are stored, by using *mfsfileinfo*:

```
mfsfileinfo /mnt/daphnis/1GB.bin
```

Or you could calculate the SHA hash of that file:

```
sha1sum -b /mnt/daphnis/1GB.bin
```

Then powering down all of the B nodes and then perform the sha1sum test again to confirm the file is complete and fully accessible despite a simulated *site failure*.

Further Reading

It is important to note that a storage class can be applied at any point in the directory hierarchy, and that there is a rich logic to the class definition that allows for some advanced storage tier structures to be established. Please refer to the MooseFS Storage Classes Manual for additional detail.

4.3.2 Adding / Removing a Disk (Scaling)

This section looks at the steps requires to add or remove a hard disk drive from the storage pool.

Adding (scale-out)

One of the most common operational activities that you will perform is adding drives to increase the presented storage capacity.

In this manual, the HC2 has been used as the hardware that underpins each storage node. The HC2 provides one new drive to the cluster per node added. Following the steps to add a new HC2 (see the Activities in section 3.3 Software, above).

If, however, you're adding drives within a multi-drive chassis, then the final step that adds the drive to the storage pool is the modification to *mfshdd.cfg* on the Chunk Server. i.e. Use a text editor to add the following line to */etc/mfs/mfshdd.cfg* which adds the drive's mount point as available storage:

```
/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Once the this entry has been added the Chunk Server will automatically report this capacity to the cluster. Once the cluster is aware, it will automatically redistribute existing chunks from other cluster nodes until all nodes have the same percentage of available capacity.

Removing (scale-back)

Removing a drive from the storage pool only requires a modification to *mfshdd.cfg* on the Chunk Server.

There are two approaches to this. The most drastic is to comment out the entry for the mount point, and to unmount the drive. i.e. Use a text editor to comment out (insert the hash character) the following line in

/etc/mfs/mfshdd.cfg:

```
#/chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Within 30 minutes the cluster will decide that this chunks are not returning and will (if replicas are available) attempt to restore them by creating new replicas in the remaining nodes.

The *nicer* way to remove a drive, particularly if there is no chunk replication configured, is to signal the impending removal of the drive to the cluster. Inserting the *less than* character (<) on the entry for the mount point will leave the drive usable, but ensure that all chunks are transferred to other drives / nodes. i.e. Use a text editor to insert the '<' per the following line in */etc/mfs/mfshdd.cfg*:

```
</chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Once the this entry has been added the Chunk Server will automatically report the change to the cluster. Once the cluster is aware, it will automatically redistribute existing chunks from this node until the drive is empty. At hat point the *mfshdd.cfg* entry can be commented out and the drive can be removed without the loss of any chunks.

4.4 Booting and Rebooting

The following section describes concepts and processes related to starting up and shutting down the Saturn cluster.

Booting the Cluster

There are only three considerations when booting the entire cluster:

- Physical: The largest power draw on the node will occur when spinning up the drive. If you don't have sufficient power on the circuit then consider staggering the node boot times.
- Logical:
  - Although it shouldn't matter, the *mfsmaster* node should be started first
  - The mfsmaster will observe that data is missing while nodes are still being started. It will begin replicating blocks in accordance with the storage policy. Once the cluster has been fully started the mfsmaster will remove the excessive blocks. If this is concerning, then please read into MooseFS "mainenance mode".

Booting a Node

With the imaged Micro SD card inserted, apply power to the HC2. Within a few minutes:

- If the device boots successfully the blue LED will flash, two pulses per second, like a heart beat.
- So long as the DHCP server is responding as needed, the node will acquire its permanent IP address.

Once the node is up, with its IP address assigned, it will be administratively accessible via SSH. Once the *mfsmaster* node is up, the administrative MooseFS status information will be available via HTTP. Please see section 4.1 Administratively Accessing Saturn, above, for access details.

For next steps:

- Check to make sure the *mfsmaster* node has the *mfsmaster* interface configured (see *masternic*, below)
- If you've performed the recommended deployment then the drive will be encrypted, and will need authentication to unlock and mount (see *mount-all*, below).

Master Designation (masternic)

As all nodes are capable of being the master node (the one running the Metadata Server), the *mfsmaster* IP address is able to "float" - it can be enabled where it is needed.

Each node must determine if it is the master node or if it is a slave node, at boot time. To do this, a script checks the */etc/default/moosefs-master* and */etc/default/moosefs-metalogger* configuration files. The node states are:

- Master – if MFSMASTER_ENABLE=true and MFSMETALOGGER_ENABLE=false
- Slave – if MFSMASTER_ENABLE=false and MFSMETALOGGER_ENABLE=true
- Any other state is ignored (which effectively defaults the node to the slave state).

The IP address for *mfsmaster* is in the */etc/hosts* file for each node, per the installation process.

If the node believes that it is a master, it will check the LAN segment for the presence of the *mfsmaster*:

```
sudo arping -q -c 3 -f -I eth0 mfsmaster
```

If the *mfsmaster* is responding then the node aborts to avoid conflict. Otherwise, the node will go on to configure the floating *mfsmaster* interface as a virtual interface:

```
sudo ifconfig eth0:1 mfsmaster netmask 255.255.255.255
```

If the node believes that it is a slave it will disable the floating *mfsmaster* interface:

```
sudo ifconfig eth0:1 down
```

The script that implements this logic in Saturn is */usr/sbin/masternic* and it is automatically run from */etc/rc.local* on each boot. It can also be run manually at any time:

```
sudo masternic
```

The *masternic* script logs its decision-making and actions to */var/log/syslog*.

Mounting Encrypted Volumes (mount-all)

Where a node has an encrypted volume, credentials will need to be supplied to unlock and mount that volume.

Unlock the Disk

Using a text editor you can see the encrypted devices in */etc/crypttab*. Find the UUID and start the disk:

```
sudo cryptdisks_start xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Physical Security Token

If you’ve enrolled a YubiKey, then plug the it in to the HC2 (via the USB extension cable):

- With the YubiKey connected to the node, enter the YubiKey pass phrase in to the *cryptdisks_start* challenge.
- Then the LED on the YubiKey button should pulse quickly.
- Press the button.

Break-Glass Key

If you have not enrolled a YubiKey, or you have lost or damaged it, then use the break-glass (fire-call) key:

- Simply type the long LUKS pass-phrase in.

Mount the Volume

With the disk unlocked the unencrypted volume can now be mounted:

```
sudo mount /chunks/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

At this point you should see that file-system show up in the *mount* or *df* commands, and you should be able see a directory listing via the *ls* command, etc.

Restart the Daemons

If this node is the *mfsmaster* then restart the Metadata Server:

```
sudo systemctl restart moosefs-master
```

If this node is not the *mfsmaster* then restart the Metalogger Server:

```
sudo systemctl restart moosefs-metalogger
```

Then restart the Chunk Server:

```
sudo systemctl restart moosefs-chunkserver
```

Scripted

To simplify the process, the script that implements this logic in Saturn is */usr/sbin/mount-all* and can be run as follows:

```
sudo mount-all
```

Rebooting or Shutting-down a Node

Login to storage node – use *ssh* to login as ***guru*** with the password that you set:

```
ssh guru@192.168.1.1
```

Note that Linux will stop all processes and unmount all volumes as part of its standard shutdown process. However, if you wish to manage this directly, the */usr/sbin/umount-all* script has been provided via the build process:

```
sudo umount-all
```

Then use *sudo* to reboot the node:

```
sudo reboot
```

Or to shutdown and power-off the node:

```
sudo shutdown -P now
```

Shutting Down the Cluster

The one key consideration for shutting down the cluster is to shut down the *mfsmaster* last. This gives each node a chance to provide its final state as it shuts down.

4.5 Clients

The following section describes concepts and configuration items that are required for users to be able to access the presented storage.

4.5.1 Installing the Client

The most commonly deployed Saturn environments, for the author, is currently Debian based, so this section describes a Debian client deployment. Note, however, that there are other clients available – for FreeBSD, OSX, and RedHat (see also the MooseFS install guide[73], and the MooseFS repo[74]).

To add the MooseFS client to a Debian-based workstation / server, add the MooseFS apt repository key to the local repo, as root:

```
wget -O - "https://ppa.moosefs.com/moosefs.key" | apt-key add -
```

To find your Debian version:

```
cat /etc/debian_version
```

Use that information to find and select the appropriate MooseFS repo:

- Ubuntu
  - 20.04 Focal: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/focal focal main
  - 18.04 Bionic: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/bionic bionic main
  - 16.04 Xenial: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/xenial xenial main
  - 14.04 Trusty: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/trusty trusty main
  - 12.10 Quantal: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/quantal quantal main
  - 12.04 Precise: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/precise precise main
  - 10.10 Maverick: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/maverick maverick main
  - 10.04 Lucid: deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/lucid lucid main
- Debian
  - 10.0 Buster: deb http://ppa.moosefs.com/moosefs-3/apt/debian/buster buster main
  - 9.0 Stretch: deb http://ppa.moosefs.com/moosefs-3/apt/debian/stretch stretch main
  - 8.0 Jessie: deb http://ppa.moosefs.com/moosefs-3/apt/debian/jessie jessie main
  - 7.0 Wheezy: deb http://ppa.moosefs.com/moosefs-3/apt/debian/wheezy wheezy main
  - 6.0 Squeeze: deb http://ppa.moosefs.com/moosefs-3/apt/debian/squeeze squeeze main
  - 5.0 Lenny: deb http://ppa.moosefs.com/moosefs-3/apt/debian/lenny lenny main
- Raspbian
  - 9.0 Stretch: deb http://ppa.moosefs.com/moosefs-3/apt/raspbian/stretch stretch main
  - 8.0 Jessie: deb http://ppa.moosefs.com/moosefs-3/apt/raspbian/jessie jessie main

73 https://moosefs.com/Content/Downloads/moosefs-installation.pdf
74 http://ppa.moosefs.com/moosefs-3/

Use a text editor to create */etc/apt/sources.list.d/moosefs.list* with the appropriate repo:

```
deb http://ppa.moosefs.com/moosefs-3/apt/ubuntu/focal focal main
```

Ensure that the apt data is up to date:

```
apt-get update
```

Install the MooseFS client package:

```
apt-get install -q -y moosefs-client
```

If you wish to disable this repo again until you're ready to do patching then, with the MooseFS package installed, remove the unneeded repo and update the apt data:

```
rm /etc/apt/sources.list.d/moosefs.list
apt-get update
```

4.5.2 Mounting the File-system

There are a couple of different ways to mount the presented file-system, depending on the permanence of the mount.

Manual

With the client installed, whether on a workstation or a server, the presented storage can be mounted by a privileged user.

First, as *root* on the workstation, create the mount point:

```
mkdir /mnt/daphnis
```

Then mount the file-system:

```
mount -t moosefs 192.168.1.111:/ /mnt/daphnis
```

Note that any sub-directory can be directly mounted also – for example the payroll sub-directory within HR:

```
mount -t moosefs 192.168.1.111:/hr/payroll /mnt/payroll
```

Automatic

With the client installed, whether on a workstation or a server, the presented storage can be mounted automatically at boot time, once configured by a privileged user.

As *root* on the server, create the mount point:

```
mkdir /mnt/daphnis
```

Then use a text editor to add the following line to */etc/fstab*:

```
192.168.1.111:/ /mnt/daphnis moosefs defaults,mfsdelayedinit,rw,noatime,nodev,nosuid 0 0
```

Note the use of the option *mfsdelayedinit* which prevents the client from erroring out during the boot process. Instead it forks, and waits for the network to become available to mount the volume in the background.

Furthermore, a privileged user is able to mount / unmount the volume using just the mount point – i.e.:

```
mount /mnt/daphnis
```

4.5.3 Content Management Scripts

The following scripts may be useful in managing content on the presented file system. These scripts would be run on the client-side – on the mounted Saturn file-system.

None of these scripts have been added to the standard build process as they're not likely to be relevant to most installations. However, as they were thrown together to suit particular use cases that arose in production, they have been included for completeness.

For external file-system synchronisation *rsync*, on a Filer Head to the Saturn deployment, is recommended.

Resetting Permissions (fix-rights.sh)

One script that has not been included in the installation scripts, but which has been useful following large changes, is *fix-rights.sh*.

From time to time projects can introduce poorly permissioned files and directories. As the production storage use is for a single user class, there is no permission hierarchy leveraged on the content directly. Hence, the following was created as a simple mechanism to reset the permissions on all directories and files, recursively.

```
#!/bin/bash

if [ ${#} -ne 1 ]
then
  echo
  echo "Usage: ${0} /path/to/saturn"
  echo
  exit 1
fi

if [ ! -d "${1}" ]
then
  echo "Error: \"${1}\" is not a valid directory."
  exit 1
fi

pwd="${1}"

echo "Structure to correct: ${pwd}"
chown -R root:root "${pwd}"/*
echo "Fixing directories .."
find "${pwd}" -type d -print0 | xargs -0 chmod 755
echo "Fixing files .."
find "${pwd}" -type f -print0 | xargs -0 chmod 644
```

This script may not be appropriate for your environment but is provided here for reference.

A variation of this script can be useful where Saturn is being used to host personal files (i.e. users' home directories), but there is no common directory (i.e. users are locally provisioned). The following script will recursively set the files and directories in Jonathan Archer's home directory, to User ID 1000 and Group ID 1000, to match the local account ownership on his workstation:

```
find "/mnt/daphnis/personal/Archer, Jonathan" -type f -print0 | xargs -0 chown -h 1000:1000
```

Note the "-h" on the *chown* which prevents symbolic links from being followed.

Copying Delta Files (undiff.sh)

Also not included in the installation process is a script that was used at one time to copy the difference of files onto Saturn based on the output of a diff that had been run against two directory listings.

```
#!/bin/bash

if [ ${#} -ne 1 ]
then
  echo
  echo "Usage: undiff diff-file.txt"
  echo
  echo "  Note: will take \"> file\" files to be copied here."
  echo
  exit 1
fi

cat "${1}" | grep ^\< | while read line
do
  srcfile=`echo "{$line}" | cut -c3-`
  # echo "cp \"${srcfile}\" .”
  cp "$(srcfile)" .
done
```

This external synchronisation process was later replaced with rsync.

Copying Delta Files (cpdiff.sh)

The *diff* script was another script that was briefly used to synchronise two directories, the current working directory with the one specified.

```
#!/bin/bash

if [ ${#} -ne 1 ]
then
  echo
  echo "Usage: ${0} /path/to/saturn/dir/subdir"
  echo
  echo "  Copy files from the current directory to the specified directory."
  echo
  exit 1
fi

if [ ! -d "${1}" ]
then
  echo "Error: \"${1}\" is not a valid directory."
  exit 1
fi

dest="${1}"

ls -1 | while read line
do
  if [ -f "${dest}/${line}" ]
  then
    echo "skip."
  else
    echo "${line}"
    echo " -- copy --"
    cp "${line}" "${dest}/"
  fi
done
```

As with the above *undiff* script, this external synchronisation process was later replaced with rsync.

5 Licensing

The following sections detail the licensing of the Saturn and the components that comprise Saturn.

5.1 Midnight Code Trademark

The term Midnight Code and the two half-moon mnemonic are registered trademarks of Ian Latter.

5.2 The Midnight Code Applications and libMidnightCode Library

All Midnight Code source code, including the libMidnightCode library and the Midnight Code applications are released with the following copyright, trademark and licensing terms.

5.3 This Document