

# Komplex Server Manual for Administrator 2<sup>nd</sup>*ed*

Complex Network Research Center

최광종<sup>1</sup> 2016.01.01 ~ 2016.10.31

조민재<sup>2</sup> 2018.11.01 ~ 2019.10.31

최호연<sup>3</sup> 2021.07.19 ~

<sup>1</sup>ckj0721@gmail.com

<sup>2</sup>mj\_03@naver.com

<sup>3</sup>hoyun1009@snu.ac.kr

# Contents

<b>1 들어가며</b>	<b>5</b>
1.1 GNU/Linux . . . . .	7
1.1.1 역사 . . . . .	7
1.1.2 개념과 용어들 . . . . .	7
1.2 Bash . . . . .	10
1.2.1 Command Line . . . . .	10
1.2.2 Bash Commands . . . . .	10
1.2.3 Bash Environment . . . . .	11
1.2.4 Bash Scripting . . . . .	12
1.3 Arch Linux . . . . .	12
1.3.1 Pacman . . . . .	13
1.3.2 AUR . . . . .	14
1.3.3 Systemd . . . . .	14
<b>2 Komplex 구조</b>	<b>16</b>
2.1 Nodes . . . . .	16
2.2 Network . . . . .	17
<b>3 Komplex 설치</b>	<b>19</b>
3.1 Arch-linux Installation: Master, Disk, GPU . . . . .	19
3.2 Package Installation: Master, Disk, GPU . . . . .	23
3.3 Network Configuration: Master, Disk, GPU . . . . .	24
3.3.1 Network Interface Controller . . . . .	24
3.3.2 DHCP . . . . .	25
3.3.3 Static IP . . . . .	26
3.3.4 Internet Sharing . . . . .	28
3.4 RAID: Disk . . . . .	30

3.5	DNS: Master, Disk, GPU . . . . .	31
3.6	NFS: Master, Disk, GPU . . . . .	32
3.7	PXE: Master, Node . . . . .	34
3.7.1	Preparation: Node . . . . .	35
3.7.2	Installation: Master . . . . .	36
3.8	NTP: Master, Disk, Node . . . . .	40
3.9	Mailing: Master . . . . .	42
<b>4</b>	<b>Komplex 운영</b>	<b>43</b>
4.1	Turn-On/Off . . . . .	43
4.2	Update . . . . .	44
4.3	계정 관리 . . . . .	46
4.3.1	계정 생성 . . . . .	46
4.3.2	계정 삭제 . . . . .	47
4.3.3	계정 정보 수정 . . . . .	47
4.4	노드 관리 . . . . .	48
4.4.1	노드 추가 . . . . .	48
4.4.2	노드 제거 . . . . .	50
4.5	디스크 관리 . . . . .	51
4.5.1	mdmonitor . . . . .	51
4.5.2	S.M.A.R.T . . . . .	51
4.5.3	Scrubbing . . . . .	52
4.5.4	Remove and Recover RAID array . . . . .	52
4.6	Security . . . . .	53
4.6.1	OpenSSH . . . . .	53
4.6.2	PAM . . . . .	55
4.6.3	Firewall . . . . .	56
4.7	Server Room Temperature . . . . .	57
4.8	Bash Scripts . . . . .	58
4.8.1	CopyToNodes . . . . .	58
4.8.2	NewNode . . . . .	59
4.8.3	DeleteNode . . . . .	60
4.8.4	NewUser & DeleteUser . . . . .	60
4.8.5	UpdateUserInfoOn... . . . . .	61
4.8.6	UpdateSystemFilesOnNodes . . . . .	61
4.8.7	net-share-host & net-share-client . . . . .	62
4.9	Services . . . . .	62

4.9.1	ExternalNetwork-terminal . . . . .	63
4.9.2	Firewall-Blacklist-update . . . . .	63
4.9.3	Firewall . . . . .	64
4.9.4	Firewall-terminal . . . . .	64
4.9.5	Admin Report . . . . .	64
4.9.6	Temperature Monitor . . . . .	65
4.9.7	Disk Alert . . . . .	65
4.10	SPG: Job Scheduler . . . . .	66
4.11	Machine Repair . . . . .	68
<b>5</b>	<b>Problems and Solutions</b>	<b>69</b>
5.1	General Solutions . . . . .	69
5.1.1	systemd-journald . . . . .	69
5.1.2	ARM: Archlinux Rollback Machine . . . . .	70
5.2	Komplex Issues . . . . .	70
5.2.1	Komplex Freeze . . . . .	70
5.2.2	Program Shut-down . . . . .	71
5.2.3	Unexpected Unmount . . . . .	72
5.3	Server Room Issues . . . . .	72
5.3.1	정전 및 누전기 차단 . . . . .	72
5.4	Archlinux Issues . . . . .	73
5.4.1	GDM: Wayland (16.01.01) . . . . .	73
5.4.2	shadow (21.08.08) . . . . .	73
5.4.3	PAM (21.08.15) . . . . .	73
5.4.4	Package update (21.09.17) . . . . .	74
5.5	Hardware Issues . . . . .	74
5.5.1	Intel: TSX bug (16.01.01) . . . . .	74
5.5.2	Intel: Pentium FDIV bug (16.01.13) . . . . .	75
5.5.3	Intel: TSX bug (19.01.01) . . . . .	75
5.5.4	Intel NIC chipset: I219-V (21.09.15) . . . . .	75
<b>6</b>	<b>Appendix</b>	<b>77</b>
6.1	Hardware Tips . . . . .	77
6.1.1	komplex(tenet) . . . . .	77
6.1.2	kreat(xenet) . . . . .	79
6.1.3	kuda . . . . .	80
6.1.4	DISK . . . . .	81

6.1.5	Network device . . . . .	82
6.1.6	Rack . . . . .	83
6.2	Custom Hooks . . . . .	84
6.3	Custom Bash Scripts . . . . .	87
6.4	Custom Services . . . . .	103

# Chapter 1

## 들어가며

이 문서는 서울대학교 통계물리그룹의 서버 관리자를 위해 쓰여졌다. 본 글은 연구실에 내려오던 “CNRC 서버 관리 매뉴얼”과 이를 박진하가 수정한 “komplex 따라잡기 튜토리얼”, 그리고 이덕재 박사님의 “Understanding Linux and Komplex”를 참고해서 작성되었다.

이 문서는 연구실에서 운영하고 있는 “komplex Server”(이하 komplex) 관리자를 대상으로 komplex에 대한 종체적인 이해를 제공하기 위해 쓰여졌다. 첫 장에서는 komplex를 포함한 리눅스 전반에 대한 내용을 다룬다. 두 번째 장에서는 구체적으로 komplex가 어떻게 구성되어 있는지를 살펴볼 것이다. 세 번째 장에서는 komplex를 구축하는 과정을 순서대로 보일 것이고 이를 통해서 자연스럽게 komplex가 어떻게 작동하는지 익히게 될 것이다. 만약 연습용 서버(mini-komplex)를 만든다면 주로 ‘komplex 설치’를 참고하면 좋을 것이다. 네 번째 장은 komplex 관리자를 위한 내용들로 komplex를 운영하면서 주로 하게 될 업무들에 대한 매뉴얼들, komplex를 위한 특별한 유틸리티 도구들의 사용법 등을 다룬다. 네 번째 장은 굳이 관리자가 아니라면 건너뛰어도 좋다. 마지막으로 komplex를 운영하면서 갑작스럽게 맞닥뜨렸던 문제들과 그에 대한 해결방안들을 제공한다. 마지막 장은 리눅스 사용자 특히 아치리눅스 사용자라면 도움이 될 만한 내용들이 있으니 참고하기를 바란다.

본 글을 읽기 전에 리눅스(Linux) 혹은 쉘(shell)에 대한 약간의 지식과 경험을 갖추는 것이 좋다. 적어도 3장의 komplex 설치를 진행하기 전에 먼저 PC에 아치리눅스(Archlinux)를 설치해보는 것이 좋을 것이다. 또한 하드웨어의 설치를 제외한 모든 작업들이 쉘의 한 종류인 배쉬(Bash, Bourne Again Shell)에서 이루어질 것이므로 배쉬를 미리 경험해보는 것이 좋을 것이다.

리눅스에 입문하는 초보자라면 이 문서에 등장하는 단어들이 너무나 생소해서 혼란스러울 것이다. 가능한 많은 해설을 통해서 이해를 돋겠지만 이 글이 서버 관리자를 위한 매뉴얼로 쓰여졌기에 초보자가 읽기에는 다소 불친절할 수 있다. 특히 실습을 병행하면서 읽어야지만 이해가 되는 부분이 많아서 글만 읽어서는 잘 이해가 되지 않을 수도 있다. 서버 설치 이전에 이 글을 선행적으로 읽는다면 나열되는 내용들이나 구체적인 명령어들은 어차피 나중에 다시 봐야 될 것이기에 ‘이런 것들이 있구나’ 정도로 가볍게 보고 넘어가는 것이 좋다. 다만 komplex의 구조를 잘 보고 어떤 요소들로 구성되어 있는지, 각 요소들이 어떠한 기능을 하는지, 요소들끼리는 어떤 관계를 가지는지, 그리고

각 요소들이 왜 도입되었는지를 주목하면서 읽도록 하자.

본격적으로 들어가기 전에 읽는 이에게 하나의 팁을 드리자면, 샵질과 구글링(Googling)과 위키는 실력 향상의 지름길이니 이를 염두해두면 두려울 것이 없다는 것이다. 그럼 시작해보자.

2016년 최광종

최광종 박사님에 의해 처음 작성된 이 매뉴얼은 2021년 새로이 개편되어 사용자를 위한 매뉴얼과 관리자를 위한 매뉴얼로 나뉘게 되었다. 본 글에서는 5년의 기간동안 실제 komplex를 운영하며 몇가지 문제들을 해결하는 과정이 추가되었으며, 조민재 박사님에 의해 몇가지 서버 운영에 대한 팁들이 추가되었다. 그리고 2021년 연구실의 변화로 새로운 komplex 서버를 만들며 수정 및 추가되어야 할 내용들이 반영되었다.

이와 동시에, 사용자를 위한 komplex 매뉴얼을 만들게 되며 본 매뉴얼에서는 komplex를 구성 및 운영하기 위해 관리자가 알아야 하는 내용만을 담게 되었다. 관리자로서 일하는 동안 알아야 하는 대부분을 담았기에 관리자는 모든 내용을 숙지하지 못하더라도 한번쯤 읽어보아 문제 상황이 있을 때 찾아볼 수 있기를 바란다. 차차 읽어보며 느끼겠지만, 다만 그 내용이 극히 방대하기 때문에 주석으로 참고 링크를 달아놓은 경우가 많다.

본 매뉴얼의 독자가 되는 관리자는 어느정도 리눅스에 익숙하다는 가정 하에 집필되었다. 이런 경우 우선 komplex의 기본 구조등을 이해하기 위해 사용자용 매뉴얼을 숙지하고 다시 도전해 보자. 다시 한 번 강조하지만 두려워 하지 말고 아치 위키와 구글링을 병행하여 문제를 해결해 보도록 하자. 리눅스에서의 오류 메세지는 매우 유용하기 때문에 문제가 생긴다면 오류 메세지 전체를 구글링하는 것도 좋은 해결 방식이다.

komplex의 운영체제인 아치리눅스는 매우 변화가 많은 운영체제중 하나이다. 이때문에, 현재 작성된 매뉴얼 역시 시간이 지남에 따라 맞지 않을 수 있다. 이후 (의지가 있는) 관리자가 지속적으로 내용을 수정하며 매뉴얼을 업데이트 해 나가주길 당부한다.

2021년 최호연

## 1.1 GNU/Linux

일반적으로 서버는 GNU/Linux의 운영체제를 사용한다. 일반적으로 pc를 다룰 때 이 둘을 구분하지 않지만, 엄밀히 말하면 GNU는 운영체제, linux는 GNU가 사용하는 커널이라고 이해할 수 있겠다. 하지만 거의 대부분의 경우 GNU는 커널로서 linux를 사용하기 때문에 운영체제를 linux 혹은 GNU/Linux라고 부른다. 본 매뉴얼에서도 이후 이를 구분하지 않고 사용할 것이다.

komplex는 위의 GNU/Linux 운영체제 중 아치 리눅스(Arch linux)를 사용한다. 리눅스에 대해 관심이 있었다면 Ubuntu, CentOS, redhat, openSUSE 등의 리눅스 배포판에 대해 들어보았을 것이다. 아치 리눅스도 그러한 배포판들 중 하나라고 생각하면 된다. 다만 일반적인 배포판들과 차이가 있다면 업데이트 주기가 매우 짧고 프로그램 패키지가 매우 최신 버전으로 제공된다는 점이다.

이를 제외하고는 대체로 리눅스라는 운영체제의 공통 분모가 있기 때문에, 이와 관련된 전반적인 개념들을 가볍게 짚고 넘어가도록 하겠다. 이 절은 아치리눅스로 가기 위한 도입부로서 리눅스를 소개하는 수준으로 그 내용이 빈약하고 또 잘못된 내용이 있을 수도 있다. 부족한 내용이 있거나 의심가는 부분이 있다면 스스로 찾아서 채워나가도록 하자.

### 1.1.1 역사

리눅스는 유닉스(Unix)와 유사하게 C와 어셈블리어로 만들어진 커널 및 운영체제이다. 상술했듯 현재 대부분은 자유 소프트웨어를 표방하는 GNU 운영체제를 뜻한다.<sup>1</sup> 대부분 특정 라이센스를 가지는 오픈소스 프로젝트들이기 때문에 매우 다양한 배포판이 존재하며, 지속적으로 개발되고 있는 운영체제이며 그 종류가 매우 다양하다.<sup>2</sup>

현재 많이 쓰이는 리눅스 배포판들은 크게 debian 계열, redhad 계열, suse 계열, 그리고 arch linux 계열로 나뉜다. 이들은 모두 리눅스라는 공통점이 있지만 동시에 차이점들이 분명하다. 따라서 komplex에 대한 내용을 구글링 하다 보면 리눅스에 대한 많은 정보가 나오는데, 다른 계열(예를 들자면 debian 계열에 속하는 ubuntu등)에서의 해결책은 매우 주의하여 읽어보고 적용해야 한다.

### 1.1.2 개념과 용어들

#### Kernel

리눅스의 핵심이 되는 것은 바로 커널(Kernel)이다. 컴퓨터는 CPU, 메모리, 그 외의 각종 하드웨어로 구성되는데 커널은 어플리케이션(Application, 윈도우 운영체제에서는 응용프로그램으로 불리기도 함)이 비록 하드웨어에 대한 구체적인 사양을 모르더라도 사용할 수 있게 해준다. 만약 커널이 없으면 우리는 개별 컴퓨터마다 코딩을 각각해주어야 할지도 모른다. 참고로 어플리케이션이 커널에게 하드웨어를 사용하기 위한 요청을 system call이라고 부른다.

---

<sup>1</sup>GNU는 GNU is Not Unix의 줄임말로, Unix의 독점 체계가 들어오면서 이에 반대하는 사람들이 유닉스와는 비슷하지만, 고유한 코드를 쓰는 GNU를 개발하게 되었다.

<sup>2</sup>[https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux\\_Distribution\\_Timeline.svg](https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg)

커널은 각각의 하드웨어들을 모듈(Module)로 나누어서 관리한다. 예컨데 그래픽 카드에 대한 커널 모듈은 그래픽 드라이버이고, 네트워크 인터페이스 컨트롤러(NIC) 카드에 대한 커널 모듈은 네트워크 드라이버가 된다. 이 말은 새로운 하드웨어를 사용하고자 한다면 그에 따른 커널 모듈 역시 필요로 한다는 것을 말한다. **lsmod**를 통해서 현재 작동하고 있는 모듈을 확인해볼 수 있다.<sup>3</sup>

## Process

어플리케이션의 요청에 실행된 작업을 프로세스(process)라고 부른다. 각각 프로세스는 PID<sup>4</sup>를 통해서 식별된다. 프로세스를 실행시킨 유저는 RUID로 부모 프로세스는 PPID로 나타낸다. 모든 프로세스는 부모 프로세스를 가지는데 최상위의 부모로 init(혹은 systemd)를 가진다. init의 PID가 1이다.

프로세스들이 점유하고 있는 리소스(메모리, CPU)와 상태들은 **ps**, **top**, **htop** 등으로 확인 가능하다. 프로세스의 상태는 대개 R(running)/S(sleeping)/D(disk sleeping)/Z(zombi)/T(suspended)으로 나타낸다. 그러나 프로그램마다 프로세스의 상태를 나타내는 방법이 조금씩 다를 수도 있는데 이는 해당 프로그램의 메뉴얼에서 확인할 수 있다.

프로세스에도 명령을 내려 제어할 수 있는데, 이를 신호(signal)을 준다고 한다. 프로세스에게 신호를 줘서 프로세스를 종료하거나(SIGTERM, SIGKILL), 해당 프로세스의 상위 부모로 버리거나(disown), 잠시 멈추거나, 백그라운드로 돌리거나 할 수도 있다. 프로세스를 다루는 방법에 대해서는 **htop**과 **jobs**의 메뉴얼을 확인하자.

## Shell & Terminal

쉘(shell)과 터미널(terminal)을 혼용하여 사용하는 사람이 많지만 이는 분명히 구분되는 두개의 프로그램이다. 터미널은 쉘의 내용을 사용자에게 보여주는 프로그램으로, 스크롤바를 사용할 것인지, 바탕색을 어떤 것으로 할 것인지, 글자체를 어떻게 할 것인지 등을 결정하는 프로그램이다. 쉘은 실제로 명령어가 수행되는 프로그램으로, 대부분의 경우 터미널이라는 용어를 쉘의 뜻으로 사용하는 경우가 많다.

우선 komplex에 접속하면 커서가 깜빡이는 쉘을 가장 먼저 마주하게 된다. 쉘은 기본적으로 명령어를 한 줄씩 받아서 처리한다. 쉘을 통해서 작업하는 모습이 리눅스나 도스 환경에 대한 대중적인 이미지가 아닐까 싶다. 쉘에도 zsh, csh, ksh, fish shell 등의 다양한 종류가 있지만 배쉬(Bourne-Again Shell, bash)가 가장 잘 알려진 쉘이다. 아치리눅스에서도 기본적으로 배쉬를 사용한다. 배쉬를 사용하는 방법은 1.2절에서 곧 다룰 것이다.

## File

리눅스에서의 모든 작업은 파일을 통해 이루어진다. 사용자가 데이터를 저장하기 위한 파일과 프로세스를 작동시키기 위한 프로그램뿐만 아니라 현재 입출력되고 있는 모든 내용들 역시 파일을

---

<sup>3</sup>마젠타로 표현된 단어들은 배쉬 명령어이다. 자세한 내용은 곧 뒤은 1.2절에서 다룰 것이다.

<sup>4</sup>Process ID

통해서 이루어진다.

### File descriptor

프로세스가 파일을 읽고 쓰기 위해서 file descriptor를 거치게 된다. file descriptor로 0(stdin), 1(stdout), 2(stderr)가 기본적으로 설정되어 있고 추가로 사용자가 지정할 수도 있다. File descriptor를 다루기 위해 redirection( > , >> ), pipe( | )의 사용법을 익혀야 한다.

### File system

File system은 두 가지 의미로 불린다. 하나는 디렉토리(directory)와 파일들이 이루는 구조를 말한다. 윈도우의 탐색기가 바로 이런 의미의 파일 시스템을 다루는 프로그램이다.

또 다른 의미로 파일 시스템은 파일이 저장되는 형태를 의미한다. 예컨데 윈도우에서는 FAT 혹은 NTFS와 같은 포맷 형식이 후자의 파일 시스템이다. 물론 파일 시스템의 두 의미가 전혀 무관한 것은 아니고 연관되어 있다. 리눅스에서는 파일 시스템으로 보통 ext4를 이용한다. 또한 UEFI 부트로더 등이 존재하는 부팅 파티션의 경우 efi file system을 사용하기도 한다.

### Directory structure convention

디렉토리는 폴더(folder)로 부르기도 하며 본 문서에서도 둘을 혼용해서 사용한다. 리눅스에서 루트 아래의 디렉토리들은 다음과 같이 분류되어 있다.

**/boot** : 커널(kernel) 이미지 `vmlinuz-linux`를 비롯해서 부팅에 필요한 파일들이 들어있다. 부트 (`/boot`) 디렉토리를 위한 물리적인 파티션을 따로 두기도 한다.

**/home** : 사용자를 위한 디렉토리로 사용자의 홈디렉토리가 기본적으로 이곳에 생성된다. 루트의 홈 디렉토리는 `/root`에 따로 존재한다. komplex의 경우 디스크 서버를 따로 두어 사용자의 홈 디렉토리를 `/pds/pdsXX` 등으로 정하기 때문에 주의하자.

**/etc** : 온갖 프로그램들의 설정파일들이 담겨져 있는 디렉토리로 폴더명과 다르게 중요하고 또 자주 다루게 될 디렉토리이다. 서버 관리자라면 이 폴더와 친숙해져야 한다. 윈도우 운영체제의 ‘Program Files’에 대응되는 디렉토리라고 생각하면 이해가 쉽다.

**/usr** : `/usr/bin`에는 실행파일이, `/usr/lib`에는 라이브러리가, `/usr/include`에는 헤더파일들이 들어 있다. `/usr/share`에는 font와 icon과 같이 사용자들이 같이 사용하는 파일들이 들어 있고, `/usr/local`에는 패키지 관리자가 다루지 않는 프로그램들과 관련된 파일들이 들어 있다.

**/var** : 로그 데이터, 특정 프로그램이 사용하는 임시파일 등과 같은 가변 데이터 파일들이 들어있다. 프로그램에 따라<sup>5</sup> `/etc` 뿐만 아니라 `/var` 디렉토리의 설정을 요구하는 경우가 있다.

기타 : `/dev`는 하드웨어 장치, `/proc`은 프로세스, `/sys` 시스템과 관련된 파일들이 들어 있다.

<sup>5</sup>대표적으로 LATEX를 사용할 수 있게 해주는 `texlive-most` 등이 있다.

앞선 내용 정도만 숙지해도 이후의 리눅스와 관련한 내용을 이해하는데 무리가 없을 것 같다. 또한 중요한 몇 가지 키워드들을 알게되었으니 어떤 것부터 시작해야하는지에 대한 막막함을 조금 덜어주었을 것이라 기대한다. 다음으로 리눅스 운영체제로 운영되는 서버에서 주로 사용되는 인터페이스인 배쉬에 대해서 다루도록 하자.

## 1.2 Bash

인터페이스(interface)는 사용자가 컴퓨터와 상호작용하기 위해서 운영체제가 제공하는 작업환경을 말한다. 윈도우에서 프로그램을 실행시키기 위해서 사람이 마우스를 이용해 커서를 움직이고 실행파일을 더블클릭으로 실행시키는데 이러한 것들이 바로 윈도우의 인터페이스이다. 근래에는 리눅스 배포판들이 Gnome, KDE 등 윈도우에 못지 않은 유려한 GUI(Graph user interface)을 제공한다. PC의 경우에는 GUI를 구성하는 것이 여려모로 편리하지만, 서버에서는 동시에 사용하는 사람이 많기에 리소스를 많이 사용하는 GUI보다는 CUI(Character user interface)로 구성하는게 일반적이다. CUI가 대중적이지는 않지만 역사적으로는 먼저 사용되었고 여전히 프로그래밍에서는 널리 사용되는 작업 환경이다. CUI에서 컴퓨터는 사용자로부터 명령어를 한줄씩 받아서 순서대로 처리하게 된다. komplex 역시 CUI로 쉘의 일종인 배쉬로 인터페이스가 구성되어 있다.

### 1.2.1 Command Line

어떠한 설정도 하지 않은 기본 bash 쉘에서 관리자 계정인 루트(root)로 ‘terminal2’에 접속하면 다음과 같다.

```
[root@terminal2 ~]#_
```

배쉬의 커멘드 라인을 통해서 현 계정(root)과 접속한 곳(terminal2) 그리고 현재 작업 중인 디렉토리 위치(~)를 알 수 있다. ‘~’는 현 계정의 홈디렉토리를 나타낸다. 밑줄(\_)은 커서를 나타낸 것이다. 그리고 ‘#’으로 명령어줄이 시작하는 것을 통해서 현 계정이 관리자 계정임을 알 수 있다. 만약 일반 계정으로 예컨데 newbi로 접속하면 다음과 같이 ‘#’가 아닌 ‘\$’로 명령어줄이 시작한다.

```
[newbi@terminal2 ~]$_
```

구글링을 통한 웹문서와 아치위키 그리고 이 글까지 포함해서 ‘\$’나 ‘#’로 시작하는 문장을 보면 대체로 배쉬에서 명령문을 나타내는 것으로 이해하면 된다. 특히 ‘#’으로 시작하면 관리자 권한으로 명령을 실행하라는 것으로 파악하면 된다.

### 1.2.2 Bash Commands

만약 배쉬가 처음인 사용자라면 서버에 접속했을 때 덩그러니 홀로 깜빡이는 커서 앞에서 앞으로 무엇을 해야하는지 당황스러울 것이다. 마치 외국어를 배울 때 단어와 사전을 찾는 법을 배우듯이,

가장 먼저 해야하는 것은 배쉬의 명령어들(Command)과 그 사용법을 보는 법을 익히는 것이다.

본 문서에서 명령어들은 **마젠타**로 나타내져 있다. 그리고 각 명령어들은 대체로 메뉴얼을 갖고 있으니, 모르는 명령어가 있거나 명령어에 옵션이 어떤 의미인지를 알고 싶다면 메뉴얼을 통해서 먼저 확인하자. **man**의 메뉴얼은 `$ man man` 혹은 구글에서의 ‘man man’ 검색으로 확인 가능하다. 다음의 명령어들은 알아두는 것이 좋다.

- Manual page: **man**
- Process management: **ps**, **pstree**, **bg**, **fg**
- System monitoring: **iostat**, **iotop**, **iftop**, **top**, **htop**
- Kernel module management: **lsmod**, **modprobe**
- Network management: **ip**
- Text utilities: **echo**, **cat**, **head**, **tail**, **more**, **less**, **grep**, **tr**, **sort**, **uniq**, **cut**
- Advanced text utilities: **awk**, **sed**
- Simple web access: **ping**, **wget**, **curl**
- File management: **ls**, **cd**, **cp**, **mv**, **mkdir**, **rm**, **pwd**, **find**, **ln**, **lsof**
- File system management : **df**, **du**, **mount**, **umount**, **mkfs**, **cfdisk**, **cfdisk**, **which**
- User management: **useradd**, **userdel**, **passwd**, **groupadd**, **gpasswd**, **last**, **who**, **chmod**, **chown**, **su**
- Archlinux: pacman, **systemctl**, **journalctl**, **coredumpctl**
- Others: **watch**, **time**, **source**

만약 위의 명령어들이 어색하더라도 당장 외울 필요는 없다. 배쉬를 사용한다면 자주 사용하게 될 명령어들이라 저절로 익숙해질 것이기 때문이다.

### 1.2.3 Bash Environment

서버에서 작업을 하다보면 반복적인 일을 하게 될 경우가 생긴다. 이런 경우를 도움이 될 만한 몇 가지는 방법이 있다.

- bash completion: 자동완성기능(auto-completion) 사용하는 것이다. 주로 ‘Tab’ 키를 이용해서 자동완성기능을 사용한다. completion을 제공하는 패키지를 설치할 수도 있고 수동으로 설정하는 방법도 있다.

- bash alias: bash alias는 단축키 지정과 같은 효과를 준다. `~/.bashrc`에서 현 계정에서 사용하는 bash alias들을 확인할 수 있다. bash alias를 이용해서 `ls`나 `grep`의 coloring option을 항상 켜지도록 할 수 있다. `source`와 같이 사용하면 alias를 불러올 수도 있는데 여러모로 유용하니 기억해두자.
- bash history: 배쉬는 기존에 받은 명령어들은 기록해두는데 이를 bash alias를 이용하면 기존의 사용했던 명령어들을 쉽게 불러올 수 있게 할 수 있다. 물론 history를 불러오는 다른 방법도 있지만 이 방법이 가장 간단한 것 같다.

대체로 각각의 기능들을 설정하는 방법과 사용하는 방법은 아치리눅스 위키를 참고하도록 하자.

#### 1.2.4 Bash Scripting

배쉬에서 작업하다보면 명령어가 길어지게 되어 있고 명령어줄에서 작업하는게 불편해지는 시점이 온다. 특히 반복적인 일을 하거나 더 추상화된 일을 하기 위해서는 배쉬 스크립팅은 필수적인 작업이 될 것이다. 배쉬 스크립팅과 관련해서 다음 사이트를 참고하자.

- Greg's BashGuide (<https://guide.bash.academy/>)

참고로 보기 좋게 배쉬 스크립팅하는 건 정말 어렵다. 그럼에도 불구하고 awk와 gnuplot 그리고 배쉬를 이용한 스크립트만으로 시뮬레이션을 제외한 연구에 대부분의 일이 가능하고 배쉬 스크립팅을 배워두면 서버에서 작업도 더욱 용이하고 효율적으로 할 수 있으므로 배워서 나쁠게 없다.

#### Shebang and POSIX

대체로 배쉬 스크립트의 첫머리가 `#!/bin/bash`로 시작하는 것을 볼 수 있다. 이것을 shebang line 혹은 bang line이라고 부른다. shebang line은 쉘이 배쉬 스크립트임을 알게된다. sawk나 gnuplot의 스크립트 역시 shebang line을 이용해서 스크립트임을 알려준다. 간혹 shebang line이 없거나 `#!/bin/sh`으로 시작하는 경우에 다른 운영체제들과 호환성을 높인 포지스(POSIX)를 이용하는 스크립트들이다.

### 1.3 Arch Linux

앞서 말했던 것처럼 리눅스에는 여러 종류들이 있다. 커널이 리눅스의 핵심이기는 하지만 그것으로 OS의 모든 요소들이 결정되지 않는다. 파일 관리자, 패키지 관리자, GUI 기반 등의 차이로 인해서 배포판이 차이가 나게 된다. 예컨데 어플리케이션을 관리하는 패키지 관리자를 보면, debian 계열인 우분투와 데비안은 .deb으로 끝나는 데비안 패키지를 이용하고, redhat 계열인 페도라와 센트OS 등은 .rpm으로 끝나는 rpm 패키지를 이용한다.

komplex에서는 이러한 배포판들 가운데서 아치리눅스(Arch linux)를 이용한다. 아치리눅스의 장점은 일단 무료로 배포되고<sup>6</sup> 위키(Wiki)와 QnA가 방대하고 포럼이 잘 활성화되어 있어 문제가 생겼을 때 해결을 위한 구글링이 용이하다. 또한 패키지(package)와 리눅스 버전을 항상 최신으로 유지할 수 있다. 단점이라면 다른 리눅스에 비해 초보자의 진입 장벽이 조금(?) 존재한다. ‘다음, 다음, …’으로만으로 쉽게 설치되는 요즘의 대중적인 리눅스 배포판에서 못 느끼겠지만, 아치리눅스를 이용하게 되면 윈도우를 왜 돈을 주고 구입하는지 이해하게 된다. 이러한 진입장벽은 오히려 장점이 되기도 하는데 일단 진입 장벽을 넘기한 한다면 그 과정에서 저절로 리눅스에 대해서 많은 것들을 배울 수 있기 때문이다. 아치리눅스에 대한 더 자세한 내용들은 아치리눅스 홈페이지 (<http://www.archlinux.org>)나 위키피디아의 아치리눅스 페이지를 참고하도록 하자.

### 1.3.1 Pacman

아치리눅스에 모든 패키지들을 pacman이라는 패키지 관리자(package manager)에 의해서 관리된다.<sup>7</sup> 아치리눅스에서 패키지가 아우르는 범위가 굉장히 넓다. gnuplot, awk, inkscape 등과 같은 어플리케이션 뿐만 아니라 clang, gcc와 같은 컴파일러(compiler)와 nvidia graphic driver와 같은 장치 드라이버, 그리고 boost와 같은 라이브러리까지 모두 각각의 패키지의 대상이 된다. 패키지 관리자인 pacman은 이러한 패키지들을 설치, 제거, 관리 해주는 역할을 한다.

물론 아치리눅스에서도 웹을 통해서 배포되는 패키지들을 수동으로 설치하는 것도 가능한다. 그러나 아치리눅스 측에서는 굳이 권장하지는 않는다. pacman을 통해서 저장소에 등록된 최신 패키지를 항상 유지할 수 있을 뿐만 아니라 pacman이 패키지들 간의 의존성을 매우 철저하게 검사 해주는데 굳이 따로 설치할 때 따르는 위험을 감수할 필요는 없기 때문이다. 아치리눅스의 핵심인 바로 견고함이 잘 드러나는 지점이다.

pacman 그 자체가 명령어이므로 그 사용법이 간단하다. 예로 각 프로세스의 네트워크 사용량을 확인할 수 있는 **nethogs**라는 어플리케이션을 검색/설치/제거하는 방법은 다음과 같다. 참고로 본 글에서 어플리케이션 또는 패키지들은 **파랑**으로 나타냈다.

- 패키지 검색: `# pacman -Ss nethogs`
- 패키지 설치: `# pacman -S nethogs`
- 패키지 제거: `# pacman -R nethogs`
- 패키지 정보: `# pacman -Qi nethogs`

이 외에도 pacman를 이용하는 방법은 pacman의 메뉴얼을 참고하도록 하자. 본 글에서 ‘특정 패키지를 설치하라’는 말은 대개 pacman을 이용해서 해당 패키지를 설치하라는 말이다.

---

<sup>6</sup>흔히 오픈 소스와 무료를 혼동하는데 이 둘은 엄연히 다른 개념이다. 물론 대부분의 리눅스들이 오픈 소스이며 무료이지만 오픈 소스이지만 유료로 제공되는 리눅스 배포판들도 있다.

<sup>7</sup>이는 다른 종류의 리눅스 배포판에도 일반적으로 존재한다. 예를 들어 debian 계열의 리눅스의 경우 apt라는 패키지 관리자를 사용한다.

### 1.3.2 AUR

대체로 많은 어플리케이션이 저장소에 이미 올라와 있지만 간혹가다 필요한 어플리케이션이 저장소에는 올라와 있지않고 사설 저장소(AUR, Arch User Repository)에만 올라와 있는 경우가 있다. 이 경우에는 **pacman**으로 설치할 수 없고 수동으로 설치하거나 **yay**등의 AUR helper 프로그램을 이용하여 **pacman**과 유사하게 설치해야 한다.

하지만 아치리눅스는 수동으로 설치하는 방법을 강력히 추천한다. 이는, 모든 종류의 AUR helper 들 역시 **pacman**이 설치할 수 있는 공식 저장소에 있지 않고 AUR에 존재하기 때문에, 그 안정성을 보장하기 어렵기 때문이다. 게다가, AUR에 있는 패키지들은 **pacman**으로 업데이트 할 수 없고<sup>8</sup> 따로 업데이트를 해주어야 한다.

komplex에서는 AUR 프로그램의 설치를 기본적으로 금지한다. 하지만 정말 필요한 프로그램을 공식 저장소에서 지원하지 않는다면, 전 그룹원들과 상의 하에 komplex에 설치하고, 반드시 매뉴얼에 기록하여 이후 업데이트를 할 때에 같이 업데이트 할 수 있도록 한다.

먼저 AUR 패키지를 수동을 설치하는 방법은 다음과 같다. 우선적으로 pacman에서 제공해 주는 패키지 중 **base-devel**을 설치해 주어야 한다.

```
# pacman -S base-devel
```

예컨데 *foo*라는 패키지를 설치하는 경우를 생각해보자. 위키 홈페이지의 AUR에서 패키지의 git cloen URL와 package base를 확인할 수 있는데, *foo*의 git clone URL이 *fooURL.git*이고 package base가 *fooBASE*라고 하자. 그러면 패키지 *foo*을 설치하는 방법은 다음과 같다.

```
# git clone fooURL.git
# cd fooBASE
# makepkg -sri
```

다시 한 번 말하지만, 이렇게 설치한 패키지들의 경우 pacman이 업데이트 해주지 않는다.<sup>9</sup> 따라서, 업데이트를 해주기 위해서는 위의 작업을 다시 반복해주어야 하는 번거로움이 존재한다.

### 1.3.3 Systemd

시스템 데몬(systemd)는 background에서 돌아가는 서비스 프로그램들을 실행시켜주고 관리해주는 프로그램이다. 이렇게 시스템에서 항상 켜져 있는 프로그램들을 데몬(daemon)이라고 말한다. 데몬들은 대체로 이름에 접미사로 ‘d’가 붙는다. 앞으로 데몬들을 **올리브**로 나타낼 것이다. 시스템 데몬(**systemd**) 자체도 이름을 통해서 알 수 있듯이 데몬의 일종이다.

시스템 데몬으로 데몬을 실행, 정지, 활성화 등의 제어하기 위해 **systemctl**라는 명령어를 사용한다. **systemctl**에서 ‘ctl’은 control의 약자로 시스템 컨트롤로 읽어주면 된다. 시스템 컨트롤로 데몬을 관리하는 방법을 알아보자. 예로 **ntpd**라는 데몬을 들어보자.<sup>10</sup>

<sup>8</sup>pacman의 설정을 수정하여 해당 패키지가 존재하는 사설 저장소까지 scan하도록 한다면 가능하지만, 권장하지 않는다.

<sup>9</sup>대신 설치된 패키지 검색 및 삭제는 가능하다.

<sup>10</sup>**ntpd**가 무엇을 하는 데몬인지는 3.5절에서 확인할 수 있다.

- 데몬 상태 확인: `# systemctl status ntpd`
- 데몬 실행: `# systemctl start ntpd`
- 데몬 정지: `# systemctl stop ntpd`
- 데몬 재실행: `# systemctl restart ntpd`
- 데몬 활성화: `# systemctl enable ntpd`
- 데몬 비활성화: `# systemctl disable ntpd`
- 데모 초기화 및 활성화: `# systemctl reenable ntpd`

‘어떤 데몬을 실행하라’라고 하면 위와 같이 시스템 컨트롤을 이용해서 데몬을 실행하라는 의미이다. 참고로 데몬들의 목록은 `/etc/systemd/system/multi-user.target.wants/`에서 확인할 수 있다.

보통은 데몬들은 설정파일을 바꿔준 후에 재실행시켜주면 된다. 하지만 몇몇 데몬들은 설정파일이 변경되거나 서비스 유닛이 수정될 경우 `systemctl daemon-reload`를 요구하거나 아예 재부팅을 요구할 수도 있어 서버에서 데몬을 재실행할 때는 주의를 요한다.

시스템 데몬의 로그를 남겨주는 시스템 저널(**journald**)이 존재한다. 시스템에 문제가 생겼을 때 이를 저널을 통해서 확인할 수 있기 때문에 꼭 알아두어야 한다. 시스템 저널에 관한 더 자세한 내용은 5.1.1절에서 확인하도록 하자.

# Chapter 2

## Komplex 구조

komplex는 계산을 수행하는 컴퓨터들이 따로 저장 장치를 두지 않는 disk-less cluster로 구성되어 있다. 이 장에서는 disk-less cluster의 전형적인 예인 komplex가 어떤 물리적인 구조와 논리적인 구조를 가지는지에 대해서 다룬다.

### 2.1 Nodes

komplex를 비롯한 서버는 일반적으로 여러 대의 컴퓨터들과 컴퓨터들을 서로 이어주는 네트워크 장비들로 구성되어 있다. komplex를 구성하는 컴퓨터의 물리적인 형태를 보면 흔히 보는 PC 형태도 있고 주로 서버용으로 사용되는 랙 마운트 형태도 있다. 각각의 컴퓨터 구성도 다양한데 CPU의 버전, GPU의 유무, HDD를 비롯한 저장장치의 유무, NIC(Network interface controller) 카드의 개수 등에서 차이를 보인다. 그럼에도 불구하고 논리적으로 보았을 때 komplex를 구성하는 컴퓨터는 다음과 같이 다섯 가지로 분류할 수 있다.

- Master node: 마스터 노드(이하 마스터)는 komplex에서 1대만 존재하는 컴퓨터로 `master`라는 hostname을 가지고 있으며, ip는 10.0.0.1로 지정되어 있다. 마스터는 계산을 수행하는 terminal과 cpu worker 노드에게 OS를 제공하며, NTP를 이용하여 서버 전체의 시간을 동기화시켜주고(TO BE DONE), DNS 서버로 서버 전체의 네트워크 주소를 관리한다. 마스터는 본인의 OS와 내부망의 노드들의 OS를 제공해주어야 하므로 별도의 디스크가 필요하다.
- Terminal Node: 터미널 노드(이하 터미널)는 사용자들이 외부망(인터넷)을 이용해서 서버에 접속하기 위해 필요한 컴퓨터이다. `terminalX`라는 hostname을 가지고 있으며, ip는 10.0.0.X로 배정된다. 서버를 사용하는 사용자 수에 따라 터미널 수를 추가할 수 있을 것이다. 터미널을 통해서 사용자들이 서버에 계산을 수행하기 위한 명령을 내리게 된다. 또한 터미널에서 간단한 데이터 분석과 파일 작업 그리고 그림 정도는 그릴 수 있다. 터미널은 마스터로부터 OS를 제공받고 작업할 저장 공간은 디스크 서버로부터 제공받기 때문에 별도의 저장 공간은 따로

필요없으나 외부망과 내부망에 동시에 연결될 수 있어야 하므로 NIC가 적어도 2개 필요하다.

- Disk Node: 디스크 노드(이하 디스크)는 사용자들이 사용할 저장 공간을 제공한다. `diskX`라는 hostname을 가지고 있으며, ip는 10.0.1.X로 배정된다. 디스크 서버는 마스터로부터 OS를 받지 않고 별도로 OS를 설치하며 또한 서버에 저장공간을 제공해야 하므로 적어도 두 개 이상의 디스크로 이루어져 있다.
- CPU worker Node: 계산 노드 혹은 간단히 노드는 서버의 실질적인 계산을 진행하는 컴퓨터로 서버에 CPU와 메모리를 제공한다. 10.0.2.X 등의 ip 주소가 배정된다. OS는 마스터에서 제공 받으며, 기본적인 시스템 데몬들이 실행된다.
- GPU worker Node: GPU 계산을 위한 노드이다. CPU가 당연히 장착되어 있기 때문에 cpu 계산도 가능하지만, 일반적으로 gpu 계산의 보조의 개념으로서만 작동할 것이다. 또한 GPU를 동작시키기 위한 그래픽 드라이버가 필요하며 이는 pc에 장착된 gpu의 모델에 따라 구분되어야 하기 때문에, 각각의 GPU 노드는 os를 가져야 한다.

## 2.2 Network

서버의 네트워크 구조는 크게 내부 컴퓨터끼리 통신이 가능한 네트워크(인트라넷)과 외부에서의 접속이 가능하도록 하는 네트워크(인터넷)으로 구성되어 있다.

외부망에 연결하기 위해 서울대학교 중앙 전산원에 외부 ip를 신청하고 관련 정보를 알아야 한다. 이 정보는 이후 3.3절에서 고정 ip를 설정하기 위해 사용될 것이다.

- 서브넷 마스크(subnet mask): 255.255.255.0
- 게이트웨이: 발급받은 IP주소의 마지막 번호(octet)을 1로 바꾸면 된다.
- 서울대학교 중앙전산원 DNS 주소: 147.46.80.1
- 서울대학교 중앙전산원 보조 DNS 주소: 147.46.37.10

2021년 현재 외부 네트워크는 100Mb/s로 연결되어 있으며<sup>1</sup> 내부 네트워크는 1Gb/s(기가비트)로 연결되어 있다.

---

<sup>1</sup>서울대학교 전산원측에 특정 학외 인터넷 속도 제한 해제 요청을 신청하지 않는 한, 약 80Mb/s의 속도로 제한된다.

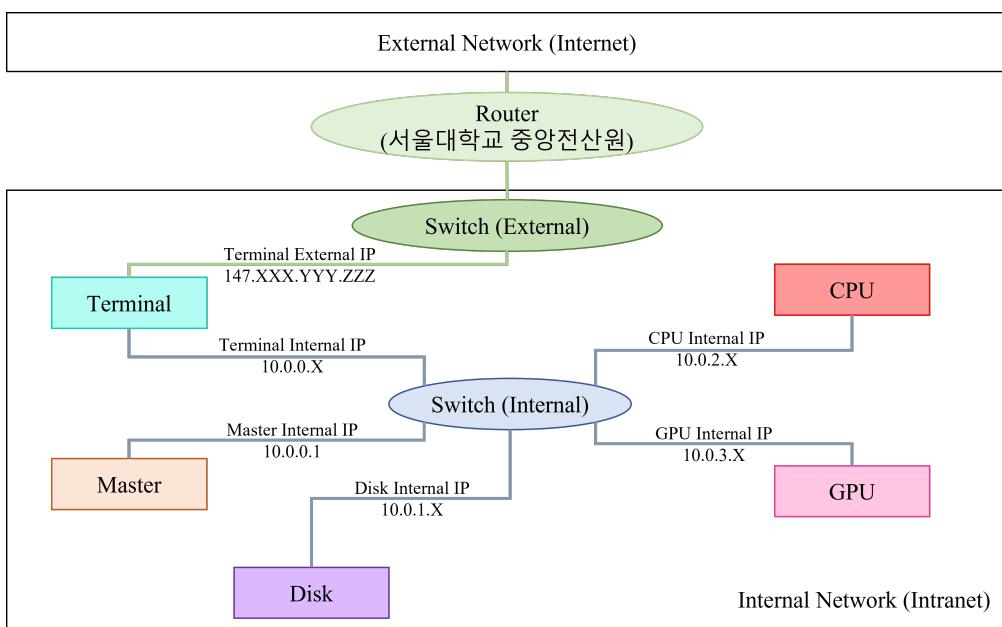


Figure 2.1: Network structure of komplex server. 실제 서버실에서는 모든 worker가 하나의 switch에 연결될 수 없으므로, 각 노드들은 자신이 위치한 랙(rack)에 장착된 스위치로 연결되고, 각 스위치들이 다시 메인 스위치로 연결되는 형식을 취한다.

# Chapter 3

## Komplex 설치

komplex를 설치하기 위해서 필요한 최소한 준비물은 다음과 같다.

- 아치리눅스 설치디스크 (USB 준비하자)
- PC 4대: 마스터, 디스크, 터미널, 노드 (Mini komplex를 만들 때)
- 스위치 2대: 외부망, 내부망

마스터 → 디스크 → 터미널 → 계산노드 순서를 권장한다. 2.2절을 참고하여 PC들과 스위치를 랜선으로 연결해 주자.

### 3.1 Arch-linux Installation: Master, Disk, GPU

개별적으로 OS를 설치해주어야 하는 컴퓨터는 마스터와 디스크이다. GPU 노드의 경우에 마스터와 다른 GPU 드라이브가 필요할 경우에는 그 노드 역시 OS를 따로 설치해주어야 한다.

아치리눅스를 설치하는 방법은 공식 홈페이지의 Installation guide<sup>1</sup>를 따라가는 것이 좋다. 특히 Installation guide가 계속 갱신되어서 때문에 따로 정리해두기가 어렵다. 따라서 반드시 본 매뉴얼과 installation guide를 동시에 보며 진행해야 할 것이다. 설치할 때 몇 가지 선택지가 있는데, komplex에 사용되고 있는 방식과 주의사항들이 무엇인지만 설명하고자 한다.

#### Installation Medium

자세한 내용은 아치 위키를 찾아보자.<sup>2</sup> 아치 리눅스를 설치하기 위해 우선 iso파일이 설치된 설치 디스크가 필요하며, 일반적으로 USB 메모리를 주로 사용한다. 이는 LINUX 혹은 UNIX 시스템에서 만들기를 권장한다. 우선 USB를 개인 pc에 연결하고, mount를 하지 않아야 한다. 이때 해당 USB

---

<sup>1</sup>[https://wiki.archlinux.org/title/Installation\\_guide](https://wiki.archlinux.org/title/Installation_guide)

<sup>2</sup>[https://wiki.archlinux.org/title/USB\\_flash\\_installation\\_medium](https://wiki.archlinux.org/title/USB_flash_installation_medium)

의 경로를 `/dev/sdX`라 하자. 해당 경로를 알고 싶다면, `lsblk` 명령어를 통해 확인해 볼 수 있다. 동시에 `mount`되지 않았다는 점도 다시 한 번 확인하자.

USB의 연결을 완료했다면, 개인 pc에서 <https://archlinux.org/download/>의 주소에서 South Korea 부분을 찾는다. 일반적으로 premi 혹은 lanet 서버를 많이 사용하며, harukasan 서버 역시 종종 사용된다. 서버에 올라가 있는 "archlinux-version-x86\_64.iso" 파일을 받고 다음과 같이 USB에 설치하자. 설치하면서 USB의 내용이 모두 사라지기 때문에, 비어있는 메모리인지 주의하여야 한다.

```
# cp /path/to/iso/file /dev/sdX
```

## BIOS and UEFI

BIOS와 UEFI는 pc의 하드웨어와 운영체제 사이를 이어주는 메인보드의 펌웨어로 BIOS(혹은 경우에 따라 LEGACY라고도 부른다)에서 UEFI로 세대 전환이 거의 완료되었다. 메인보드에 따라 지원하는 종류가 다르며, 2021년 기준 대부분의 메인보드들은 두가지 종류를 모두 지원한다. 하지만 기본 설정으로 UEFI만 지원하도록 되어있는 경우가 있을수도 있으니, 문제가 있을경우 이를 확인해보도록 한다.

부팅 파티션의 마운트 포인트는 BIOS의 경우 `/mnt/boot`로, UEFI의 경우 `/mnt/efi`를 기본으로 한다.<sup>3</sup> 이후 아치 리눅스를 설치하면서 `pacstrap /mnt base linux linux-firmware`를 실행할 것이다. 이를 실행하면, BIOS/UEFI의 선택과 무관하게 `/mnt/boot`에 `vmlinuz-linux`와 `initramfs-linux` 등이 만들어 질 텐데, 그 이유는 `/etc/mkinitcpio.d/linux.preset`에 해당 경로로 설정되었기 때문이다. 혹시 Syslinux를 사용할때 `esp`를 `/mnt/efi`로 설정하고 싶다면, 이를 변경하도록 하자. 자세한 내용은 3.7.2절을 참고하자.

마스터의 경우, 후술될 부트로더를 `syslinux`를 사용하여야 하는데, 해당 부트로더가 UEFI 부팅을 안전하게 지원하지 않는다. 또한 마스터에 연결된 오래된 계산노드들 같은 경우, UEFI를 지원하지 않을 가능성이 있기 때문에, 해당 노드들도 부팅할 수 있도록 가능하면 BIOS 형식을 쓰도록 하자. 마스터 이외의 DISK와 GPU 머신의 경우, UEFI 형식을 사용해도 무방하다.

## Partition

- 파티션(partition)을 시작할 때 MBR과 GPT 가운데서 고를 수 있다. 개인용으로 사용한다면 MBR과 GRUB를 사용한 방법이 설치가 쉽다. 그러나 2TB 이상의 HDD를 사용하는 디스크 서버의 경우에는 꼭 GPT를 이용해야 한다. GPT 설정을 위해 파티션 어플리케이션으로 `cfdisk`를 이용하면 된다.
- 스왑(Swap)은 메모리가 부족할 경우에 저장 공간에서 메모리를 잡기 위한 용도로 메모리 부족으로 인해 컴퓨터가 다운되는 것을 막아준다. 스왑의 크기는 메모리 크기의 10% 내외로

---

<sup>3</sup>UEFI의 경우 해당 경로는 변경 가능하기에 아치위키에서는 `esp`(EFI system partition)이라 명명한다. Syslinux의 경우, `esp` 외부에 있는 파일에 접근하지 못하기 때문에, `/mnt/efi` 대신, `/mnt/boot`에 마운트 할것을 권장한다.

잡아준다.

- 디스크의 HDD를 포맷(format)할 때는 inode의 크기를 충분히 잡아주어야 한다. inode의 용량에 따라 저장장치에 만들 수 있는 파일과 폴더의 개수가 정해지는데 데이터를 많이 작성하는 디스크의 경우 PC 수준으로 inode의 크기를 지정하면 부족할 수 있다. 참고로 inode의 용량은 `df -hi`로 확인 가능하다.

## Internet

아치리눅스는 설치 디스크만 넣어다고 설치를 진행할 수 있는게 아니라 인터넷이 꼭 연결되어 있어야 한다. 설치 디스크는 다만 아치리눅스 설치를 위한 기본적인 아치리눅스(?)가 깔려 있을 뿐이다. 먼저 설치 디스크로 아치리눅스를 부팅하고 구글의 DNS 주소인 8.8.8.8로 ping을 보내어 인터넷 연결되어 있는지 확인해보도록 하자.

```
# ping 8.8.8.8
```

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=38 time=51.7 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=38 time=51.8 ms  
...
```

위와 같이 나오다면 인터넷이 연결되어 있으니 ‘ctrl + c’를 눌러 중지하자. 한참을 기다려도 위와 같이 나오지 않는다면 인터넷이 연결이 안되어 있는 것이다.

설치 디스크의 경우 자동으로 DHCP를 이용하여 유동 IP로 접속하도록 되어 있기 때문에, 만약 외부망에 연결된 공유기로부터 IP를 제공받을 수 있는 상황이라면 인터넷이 연결될 것이다. 설치 과정에서는 문제 없지만, 실제로 서버를 가동할 때에는 반드시 서울대학교에서 고정 IP를 발급받아 3.3절을 참고하여 고정 IP로 설정해 놓고 쓰도록 하자.

## Mirrorlist

미러 리스트는 아치 리눅스의 공식 패키지들을 받을 수 있는 곳으로, 현재 komplex에서는 아래와 같은 한국의 미러 리스트를 사용하고 있다. 하지만 이는 자주 바뀔 수 있기 때문에 문제가 생긴다면 `/etc/pacman.d/mirrorlist`를 수정해주자. 부팅 디스크를 통해 아치 리눅스를 설치할 때 설정한 값이 그대로 설치된 아치 리눅스에도 남아있기 때문에 미리 설정해 두자.

```
/etc/pacman.d/mirrorlist
```

```
Server = https://ftp.lanet.kr/pub/archlinux/$repo/os/$arch  
Server = https://mirror.premi.st/archlinux/$repo/os/$arch  
Server = https://ftp.harukasan.org/archlinux/$repo/os/$arch  
Server = https://mirror.anigil.com/archlinux/$repo/os/$arch
```

```
Server = http://ftp.lanet.kr/pub/archlinux/$repo/os/$arch
Server = http://mirror.premi.st/archlinux/$repo/os/$arch
Server = http://ftp.harukasan.org/archlinux/$repo/os/$arch
Server = http://mirror.anigi.com/archlinux/$repo/os/$arch
```

## Boot loader

부트로더는 개인 PC의 경우에 **grub** 혹은 **systemd-boot**를 주로 사용한다. 하지만 komplex에서 노드들을 PXELINUX로 부팅시키기 때문에 마스터는 **syslinux**를 부트로더로 사용해야 한다. PXE부팅에 대한 자세한 내용은 3.7를 참고하자. 디스크와 GPU 머신의 경우는 부트로더로 **grub**를 사용해도 무방하다. 특히 디스크의 경우에는 RAID를 해야 하는 경우가 있는데, **syslinux**의 경우 **mdadm**을 통한 RAID의 최신 버전을 지원하지 않으므로, **grub**이 조금 더 선호된다.

## Microcode

컴퓨터의 cpu에 따라 **intel-ucode** 혹은 **amd-ucode**의 설치를 권장한다. 이를 설치하지 않더라도 사용에는 큰 지장이 없으나, 혹시 모를 버그의 예방을 위함이다. 설치 이후, `/boot` 디렉토리에 `cpu_manufacturer-ucode.img` 가 있는지 확인해 보자. 어떤 부트로더를 사용했는지에 따라 부트로더의 configuration을 해 주는 방법이 달라진다.

GRUB의 경우, `grub-mkconfig -o /boot/grub/grub.cfg`를 통해 .cfg파일을 생성해 줄 때 microcode를 자동으로 확인하여 추가해 준다.

```
/boot/grub/grub.cfg
...
echo 'Loading initial ramdisk'
initrd /boot/cpu_manufacturer-ucode.img /boot/initramfs-linux.img
...
```

Syslinux의 경우, 존재하는 .cfg파일에 직접 microcode를 추가해 주어야 한다.

```
/boot/syslinux/syslinux.cfg
...
LABEL arch
MENU LABEL Arch Linux
LINUX ..../vmlinuz-linux
INITRD ..../cpu_manufacturer-ucode.img,..../initramfs-linux.img
...
```

## 3.2 Package Installation: Master, Disk, GPU

마스터는 각각의 노드들에게 OS를 비롯한 여러 유ти리티까지 제공해주어야 한다. 이에 먼저 마스터에 필요한 유ти리티 패키지들을 설치해주고 다시 각각의 노드들에 전달해주는 방식을 사용한다. 이후 새로운 패키지들을 깔거나 업데이트 할 때도, 우선 마스터를 업데이트 하고 이후 각 노드들을 재부팅 해주어야 한다. 서버를 업데이트하는 자세한 내용은 4.2를 참고하자.

### List of Packages

komplex에 설치되어온 패키지들은 다음과 같다.

- Basic: **awk**, **bc**, **man**, **rsync**
- Text Editor: **vim**, **nano**, **emacs**
- System Monitor: **htop**, **iostop**, **iftop**, **nethogs**, **net-tools**, **sysstat**, **lm\_sensors**
- Network: **dhcpd**, **netctl**, **openssh**, **dnsmasq**, **ethtool**
- Internet: **wget**, **git**, **msmtp**
- Security: **libpwquality**
- Disk: **mdadm**, **smartmontools**
- Power: **cpupower**
- Shell: **zsh**, **fish**, **bash-completion**, **tmux**, **powerline**
- Compiler: **clang**, **gcc-fortran**, **go**, **cmake**
- Python: **python**, **python-pip**
- Debug: **valgrind**
- Graphic: **gnuplot**, **cups-pdf**, **inkscape**
- Others: **gptfdisk**, **cpio**, **r**
- L<sup>A</sup>T<sub>E</sub>X: **texlive-most**, **texlive-lang**, **ghostscript**

위에 나열한 패키지 말고도 더 많은 패키지가 설치되어 있는데 설치된 패키지들은 `# pacman -Qq`로 확인할 수 있다. 만약 이후에 패키지가 추가로 설치된다면 관리자가 위의 리스트를 갱신해주면 좋을 것 같다.

I/O가 많이 이루어지는 작업을 해야하는 경우 네트워크를 통해서 작업하기 보다 바로 디스크에서 작업해야 할 수도 있다. 이 경우에 디스크에도 위와 같은 유ти리티들을 설치해주어야 한다.

## Graphic Driver: GPU

우리 그룹은 AMD보다 NVIDIA의 GPU를 선호에 왔다. 그에 맞춰 그래픽 드라이버 역시 nvidia 계열의 그래픽 드라이버를 설치해주면 된다.<sup>4</sup> `pacman -S nvidia`를 통해 그래픽 드라이버를 설치 할 경우에 재부팅 해야한다는 사실을 기억하자. 기본적으로 마스터에는 외장 그래픽카드를 사용하지 않기 때문에 위의 드라이버를 설치할 필요는 없으며 gpu 머신에 한정하여 설치하면 된다. 아치리눅스에서는 NVIDIA의 홈페이지에서 드라이버를 받는 것보다 `pacman`을 통해 설치하는 방식을 권장한다.

## 3.3 Network Configuration: Master, Disk, GPU

인터넷에 연결하는 방법은 제공되는 네트워크 환경에 따라 다양할 수 있는데, 외부망에 직접 연결된 PC의 network 설정을 유동 IP 또는 고정 IP으로 바꾸어서 연결할 수도 있고(ethernet), 인터넷에 이미 연결된 PC를 거쳐서 인터넷을 연결할 수도 있다(route).

직접적으로 외부망에 연결된 PC를 인터넷이 되도록 하는 방법을 먼저 알아보도록 하자. 여기서 실질적으로 네트워크 설정을 다루는 법을 배우기 때문에 일차적으로 인터넷 연결하는데 사용되지만 이후에 서버 내부망에 연결하기 위한 네트워크 설정도 동일하게 이루어지므로 잘 숙지하도록 하자. 아치리눅스 위키의 Network configuration 페이지<sup>5</sup>를 참고하자.

### 3.3.1 Network Interface Controller

PC에 설치된 NIC(Network interface Controller)를 확인하는 방법부터 알아보자.

`ip addr`<sup>6</sup> 또는 `ip link` 또는 `ls /sys/class/net/`를 치면 네트워크 인터페이스 목록들을 확인할 수 있다. `lo`로 시작하는 인터페이스는 loop device로 인터넷에 연결하기 위해서 사용할 수는 없고, 그 외의 인터페이스들이 실제로 설치된 NIC의 인터페이스이다. 대체로 NIC가 리얼텍(Realtek)인 경우에 'enp'으로 시작하고 인텔(intel)인 경우에는 'eth'로 시작하는데 인터페이스 이름은 사용자가 바꿔줄 수도 있고 제조사 또는 제품마다 다를 수 있으니까 참고만 하자. 아래에서는 설명을 돋기 위해서 예제로 인터페이스를 *interface*라고 할 것이다.

인터페이스의 이름을 바꾸고 싶다면 아래와 같이 파일을 작성해주고 재부팅을 해주면 된다. `mac:addr`에 NIC의 mac 주소를 넣어주고 `name`에 인터페이스의 이름을 넣어주면 된다. 마스터와 터미널과 같이 NIC가 2개인 경우 외부 인터넷과 연결된 NIC의 이름을 "external", 내부망과 연결된 NIC의 이름을 "internal"로 한다.

```
/etc/udev/rules.d/10-network.rules
```

```
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="mac:addr", NAME=="name"
```

<sup>4</sup><https://wiki.archlinux.org/title/NVIDIA>

<sup>5</sup>[https://wiki.archlinux.org/title/Network\\_configuration](https://wiki.archlinux.org/title/Network_configuration)

<sup>6</sup> `ip address` 와 같다.

내부망만 연결되어 있는 다른 노드들은 이 작업을 해주지 않아도 괜찮지만, 이후 network sharing 등의 작업을 위하여 "internal"이라 정하도록 하자. 노드들의 경우 4.8.2에서 다룰 `NewNode.sh` 가 `/root/admin/70-persistent-if-name-komplex.rules`에 추가해 주는 작업으로 대신 해주며, 디스크의 경우 직접 적어주어야 한다.

### 3.3.2 DHCP

앞서 언급했다시피 공유기로부터 유동 IP를 할당 받을 수 있는 경우에는 DHCP를 이용할 수 있다. 설치 디스크의 경우에는 자동으로 인터페이스를 검색해서 DHCP로 연결을 시도하지만 방금 설치가 된 리눅스이거나 본래 고정 IP를 사용하다가 유동 IP를 사용하려는 경우에는 따로 설정을 해주어야 한다.

#### dhcpcd

`dhcpcd`를 이용하는 방법을 살펴보자. 먼저 NIC가 켜져 있어야 한다. `ip addr` 를 쳐주었을 때 인터페이스의 상태가 `UP` 이어야 한다. 만약 그렇지 않다면 인터페이스를 켜주도록 하자.

```
# ip link set interface up
```

다음으로 인터페이스를 `dhcpcd`로 작동시켜주기만 하면 된다.

```
# systemctl start dhcpcd@interface
```

부팅 시에 매번 `dhcpcd`를 작동시키려면 `enable` 해주면 된다.

```
# systemctl enable dhcpcd@interface
```

`dhcpcd@interface`에서 뒤의 `@interface`를 적어주지 않는다면, `ip addr`에서 나타나는 모든 인터페이스들에 `dhcpcd` 서비스가 시작된다.

#### netctl

다른 방법으로 `netctl`<sup>7</sup>을 이용할 수도 있다. 해당 방법을 선택한다면, 충돌이 나지 않도록 우선 `dhcpcd`를 종료하고, 부팅시 자동으로 실행되는 것을 방지하기 위해 `disable` 도 같이 해주도록 하자.

```
# systemctl stop dhcpcd@interface
# systemctl disable dhcpcd@interface
```

인터페이스 역시 꺼주어야 한다.

<sup>7</sup><https://wiki.archlinux.org/title/Netctl>

```
# ip link set interface down
```

**netctl**이 DHCP를 이용할 수 있도록 프로필(profile)를 만들어주어야 한다. 프로필 예제들은 `/etc/netctl/example/`에 있는데 그 가운데서 `ethernet-dhcp`를 `/etc/netctl/`에 복사해준다. 이때 기존에 **netctl** 프로필이 있을 수도 있으니 중복되지 않게 이름을 바꿔주도록 하자. 예컨데 `new-dhcp`로 하겠다. 이제 다음과 같이 프로필을 작성하자.

```
/etc/netctl/new-dhcp
```

Description='A example of dhcp ethernet connection with netctl'

Interface=*interface*

Connection=ethernet

IP=dhcp

이제 **netctl**으로 프로필 켜주자. **systemd**와 유사하게 부팅시에 매번 프로필을 이용할 때는 `enable` 시켜주면 된다. **netctl** 서비스를 시작하면, 다시 인터페이스가 켜진다.

```
# netctl start new-dhcp  
# netctl enable new-dhcp
```

이후 `$ ip addr` 와 `$ ping 8.8.8.8` 등으로 인터넷 연결을 확인하자.

### 3.3.3 Static IP

학내에서 건물 외벽으로 제공되는 랜포트와 연결해서 인터넷을 연결하는 경우에는 고정 IP를 받아서 사용해야 한다. 2.2절에서 마스터와 터미널에 제공되는 고정 IP를 보면 알겠지만, 학내에서 사용하는 고정 IP는 147.47.xxx.xxx 또는 147.46.xxx.xxx 형태이다. 뿐만 아니라 komplex 내부망 역시 사설망으로 고정 IP를 할당하여 운영하고 있다. 즉, 고정 IP 설정은 외부망으로의 연결 뿐만 아니라 내부망에서 연결에도 사용되므로 잘 숙지하도록 하자. 현재 komplex에서는 **netctl**을 사용하는 방법을 기본으로 사용하고 있다.

#### SubNet mask, Gateway, DNS

외부망에 연결된 NIC의 경우 서브넷 마스크가 255.255.255.0이기 때문에 IP 주소의 *prefix*가 24가 된다.<sup>8</sup> 내부망에 연결된 NIC의 경우 할당하는 서브넷이 10.0.0.1 ~10.0.255.255이기 때문에, 서브넷 마스크가 255.255.0.0이 되며 이에 따라 *prefix*는 16이 된다.

게이트웨이의 경우, 외부망과 연결된 NIC의 설정에 필요한 것으로 고정 IP에서 마지막 숫자만 1로 바꾸어 주면 된다.<sup>9</sup> 따라서, 내부망과 연결된 NIC의 경우, 게이트웨이 설정은 건너뛰어도 된다.

<sup>8</sup>이는 255.255.255.0을 2진수로 쓰면 1111 1111.1111 1111.1111 1111.0000 0000이 되어 24개의 연속된 1이 나오기 때문에 그러하다.

<sup>9</sup>이는 서브넷마스크가 255.255.255.0이기 때문에 그러하다

DNS 주소는 서울대학교의 시스템 147.46.80.1를 사용한다. 역시 외부망과 연결된 NIC의 설정에만 필요하며, 내부망과 연결된 NIC 설정시에는 해당 과정을 comment out하거나 건너뛰면 된다.

### netctl

DHCP에서와 마찬가지로 **netctl**를 이용하여 고정 ip를 설정할 수도 있다. 고정 ip를 사용하기 위한 프로필을 먼저 작성해주어야 한다. /etc/netctl/example에서 **ethernet-static**를 /etc/netctl/로 복사해온다. 마찬가지로 기존에 netctl 프로필과 겹치지 않게 프로필의 이름을 잘 정하자. 예컨데 **new-static**으로 하겠다. 이제 다음과 같이 프로필을 작성해주자.

```
/etc/netctl/new-static
```

```
Description='A example of static ethernet connection with netctl'  
Interface=interface  
Connection=ethernet  
IP=static  
Address='static-ip/prefix'  
Gateway='gateway-addr'  
DNS=('dns-addr')
```

인터페이스가 이미 작동하고 있다면 꺼준 후에 netctl로 작성한 프로필을 켜주도록 하자. 부팅시에 매번 프로필을 사용하도록 **enable** 역시 시켜주면 된다.

```
# ip link set interface down  
# netctl start new-static  
# netctl enable new-static
```

### ip addr & ip route

**ip addr** 와 **ip route** 를 이용하여 고정 IP를 설정해보도록 하자. DHCP의 경우와 마찬가지로 인터페이스가 켜져 있는지 확인하고 켜져있다면 켜주도록 하자. 다음으로 인터페이스에 고정 IP 주소(예를 들어 *static-ip*)를 등록하자.

```
# ip link set interface up  
# ip addr add static-ip/prefix dev interface
```

다음으로 인터페이스의 게이트웨이 주소(예로 *gateway-addr*)를 설정해주자.

```
# ip route add default via gateway-addr dev interface
```

하지만 komplex2의 경우에 위의 명령어 대신 아래의 명령어를 쳐야 작동하였다. (210522 MJ)

```
# ip route add default via gateway-addr onlink dev interface
```

마지막으로 DNS 주소(예로 *dns-addr*)를 설정해주자. DNS 주소는 `/etc/resolv.conf`에 추가해주면 된다.

```
/etc/resolv.conf
```

```
nameserver dns-addr
```

## Terminal

터미널 노드의 경우, 마스터의 경우와 같이 외부 접속을 위한 static ip설정을 해야 한다. 하지만, PXE부팅을 통해 운영체제를 마스터로부터 받기 때문에, 위와 같이 `netctl`을 통해 static ip를 설정해 주기 어렵다. 이때문에 `systemctl`을 통해 서비스의 형태로 ip를 설정해 주어야 한다. 구체적인 내용은 4.9.1절에서 다룬다.

## Initialization

`dhcpcd` 또는 `netctl`을 이용하는데 어떻게 해도 DHCP 혹은 Static IP가 안 된다면 인터페이스의 설정을 모두 초기화해보자.

```
# ip addr flush dev interface
# ip route flush dev interface
```

### 3.3.4 Internet Sharing

이번에는 인터넷에 연결된 PC가 라우팅을 통해 다른 PC가 인터넷을 사용할 수 있게 해주는 방법을 살펴보자. 대체로 디스크의 리눅스 업데이트를 할 때 이 방법은 사용한다. 왜냐하면 디스크들은 서버 내부망으로만 이어져 있는데 리눅스 업데이트를 하기 위해서는 외부망을 필요로 하기 때문이다. 앞에서 설명한 방법을 사용하려면 디스크를 내부망에서 분리하여 외부망에 연결하고 네트워크를 설정을 바꿔주는 작업을 해주어야 하는데 이 작업이 매우 매우 번거롭기 때문이다. 아치리눅스 위키의 Internet sharing 페이지<sup>10</sup>를 참고하면서 진행하자.

인터넷이 연결된 PC를 호스트(host)라고 부르고 그 ip주소를 *host-ip*라 하자 (ex.10.0.0.2). 아직 인터넷이 연결되지 않은 PC를 클라이언트(client)라고 부르며 그 ip 주소를 *client-ip*라 한다 (ex.10.0.1.1). 먼저 클라이언트의 네트워크 설정을 해 준뒤, 호스트가 라우팅을 할 수 있도록 네트워크를 설정해줄 것이다.

---

<sup>10</sup>[https://wiki.archlinux.org/title/Internet\\_sharing](https://wiki.archlinux.org/title/Internet_sharing)

## Client

우선 클라이언트에서 host를 통해 라우팅 되어 들어오는 패킷을 받기 위한 설정을 해주어야 한다. 일반적인 경우 해 줄 필요 없지만, 24의 subnet prefix를 가지는 ip주소를 설정해 주자<sup>11</sup>.

```
# ip addr add ip/24 dev client
```

그리고, # ip route show 를 통해 현재 default route를 확인하자. 만약 default route가 host-ip 가 아닌, ip라면,<sup>12</sup> 다음과 같이 삭제해 주어야 한다.

```
# ip route del default via ip dev client
```

그리고 다음과 같이 host-ip를 추가해 주자.

```
# ip route add default via host-ip dev client
```

이러한 작업은 주로 터미널이 host의 역할을, 디스크 서버가 client의 역할을 하게 될텐데, 이를 위해 각 디스크 서버에서는 /root/admin/net-share-client.sh 가 이 작업을 해준다. 해당 스크립트에 대한 자세한 사항은 4.8.7절을 참고하자.

또한, DNS의 nameserver를 마스터의 ip 주소로 해야 한다. 이 작업은 컴퓨터를 설치한 이후 한번만 진행하면 될 것이다.

```
/etc/resolv.conf
```

```
nameserver=10.0.0.1
```

## Host

호스트가 인터넷을 공유해 주기 위해서는, 외부 인터넷과 연결된 NIC *external*이 내부 인트라넷과 연결된 NIC *internal*로 패킷을 전송해 줄 수 있어야 한다. 이를 위해 우선 nftables를 사용한 방화벽의 설정을 다음과 같이 해주자.

```
# nft add rule ip nat POSTROUTING oifname "external" masquerade  
# nft add rule ip filter FORWARD ct state related,established accept  
# nft add rule ip filter FORWARD iifname "internal" oifname "external" accept
```

이는 이후 4.9.4절에서 다루는 **Firewall-terminal**에서 작업을 해주기 때문에, 직접 해줄 필요는 없을 것이다.

다음으로, *external*에서 *internal*로 포워딩을 해주어야 한다. Client의 경우와 마찬가지로 subnet prefix 24를 설정해 주어야 한다. 그리고 sysctl을 사용하며, net.ipv4.ip\_forward 항목의 값을 1로 설정함으로서 켜주고, 0으로 설정하여 꺼줄 수 있다. 이는 /root/admin/net-share-host.sh 로 구현되어 있으며, 자세한 사항은 4.8.7절에서 확인할 수 있다. 해당 스크립트를 사용하여 포워딩을 켜주고, 사용한 후에는 다시 동일한 스크립트를 실행하여 꺼주는 습관을 들이자.

<sup>11</sup>Sharing 해주는 호스트인 터미널의 내부 IP주소와 client의 내부 IP 주소 앞 세자리(3 byte)가 다르면 이 작업을 해주지 않아도 된다.

<sup>12</sup>대부분의 경우에 마스터의 내부 ip인 10.0.0.1로 설정되어 있을 것이다.

### 3.4 RAID: Disk

레이드(RAID; Redundant Array of Independent Disks)는 기본적으로 복수의 디스크 드라이브를 마치 하나의 디스크 드라이브처럼 사용하는 것을 말한다. 하나의 디스크가 물리적으로 고장나서 복원이 어려울 때 미러링을 통해서 백업된 디스크를 통해서 복원이 가능하다. 레이드는 그 레벨(level)에 따라 구현되는 방법이 다른데 기본적으로 백업을 위해서는 RAID 1을 사용한다. RAID 0은 striping을 사용하여 디스크 I/O의 속도를 높일 수 있는 방법이지만 백업을 위한 용도는 아니다. 추가로 RAID 0과 RAID 1을 조합한 RAID 10도 있다. 현재(21.07.27) 디스크에 RAID5를 도입하였다. 이후 서버를 증축하여 디스크를 6개이상 사용한 RAID를 구성하고자 한다면 RAID6를 고려해볼만 하다. RAID5, RAID6는 parity 드라이브를 사용하여 redundancy를 보장하는데, 구체적인 내용은 직접 찾아보자.

RAID의 설치는 `mdadm`을 사용한 위키페이지를 따라간다.<sup>13</sup> 아래에 설치 시에 참고할 몇 가지 사항들을 남겨둔다.

- 디스크 서버의 RAID 어레이의 디바이스 이름은 `/dev/md1`, `/dev/md2` 등으로, 마운트 포인트는 `/pds/pds11`, `/pds/pds12` 등으로 이름을 맞춰준다. pds의 첫번째 숫자는 디스크 서버의 번호, 두번째 숫자는 해당 디스크 서버에서의 어레이 번호로 맞춰준다.
- 처음 RAID를 진행할 경우, 각 하드 디스크에서 마지막 100MiB<sup>14</sup> 정도는 unallocated로 남겨둔 채로 파티션을 나누자. 이는, 서로 다른 브랜드의 하드 디스크를 사용할 경우 같은 용량으로 표시되었다 하더라도 조금씩 용량의 차이가 있을 수 있기 때문이다. 서버를 운영하다 하나의 하드 디스크에 문제가 생겨 새로운 디스크로 교체할 때, 본래보다 조금 작은 용량의 하드 디스크를 사용한다면 생길 수 있는 오류를 방지할 수 있다.
- RAID할 때 array의 chunk size가 얼마인지를 정해주어야 한다. RAID array에서 다룰 데이터가 작은 파일에 여러개 나뉘어 있는지, 하나의 커다란 파일에 있는지에 따라 최적화된 값이 달라진다. komplex에서는 주로 네트워크단에서 I/O의 bottleneck이 있기 때문에, 하나의 커다란 파일에 데이터를 넣는 것을 권장한다. 사용되는 chunk size는 후술될 항목의 stripe값이 32KiB 혹은 64KiB가 되도록 정해주자.
- RAID array를 만드는 시간은 사용된 하드 디스크 용량과 개수에 따라 다르지만, 최소 1시간 이상 소요된다. Array가 만들어 진 이후 포맷을 해야 한다는 점에 주의하자. 만들어지는 진행상황은 `$ watch -n 1 'cat /proc/mdstat'` 을 통해 실시간으로 확인할 수 있다.
- 디스크 서버의 경우, RAID의 관리 외에 다른 작업을 하지 않으므로, 사용 가능한 최대한의 cpu 연산량을 RAID의 sync 속도를 높이기 위해 쓸 수 있다.

<sup>13</sup><https://wiki.archlinux.org/title/RAID#Installation>

<sup>14</sup>Mebibyte. 1MiB =  $2^{10}$ KiB =  $2^{20}$ Byte. 참고로 1MB =  $10^3$ KB =  $10^6$ Byte.

```
# sysctl -w dev.raid.speed_limit_min=600000
# sysctl -w dev.raid.speed_limit_max=600000
# echo 4 > /sys/block/md1/md/group_thread_cnt
```

- RAID array를 포맷할 때, 적절한 stride와 stripe 값을 사용해 주어야 한다. 아치 위키의 해당 항목<sup>15</sup>을 보면 잘 계산해주자. 예를 들어, 총 3개의 하드 디스크를 RAID5로 묶어 data 드라이브 2개, parity 드라이브 1개로 구성하며 chunk size는 64KiB, ext4 포맷으로 진행할 경우를 생각해 보자. stride 값은 (chunk size)/(block size)=64/4=16<sup>16</sup>이며, stripe값은 (data drive num)\*(stride)=2\*16=32 으로 계산된다.
- 포맷까지 완료해준 RAID array를 `/etc/fstab`<sup>17</sup>에 등록해 주어야 한다. 아치 리눅스 설치 때와 같이 `/mnt`에 순서대로 `mount`해준 뒤에서 `genfstab`을 이용한다. `genfstab`은 아치 리눅스에서 사용할 수 없으며, 오직 시동디스크에서만 사용할 수 있기 때문에 다시 한번 시동 디스크로 부팅하여야 한다. 부팅 후 순서대로 마운트를 해 주고 `genfstab`을 하자.

```
# genfstab -U /mnt > /mnt/etc/fstab
```

- RAID를 하는 주된 이유는, 디스크 I/O의 속도 증가에도 있지만 무엇보다 하드 디스크가 고장 났을 경우, 내부의 데이터의 손실을 막기 위해서이다. 이를 위해 `mdadm`에서는 `mdmonitor`이라는 시스템 데몬을 통해 RAID된 디스크들의 상태를 지속적으로 체크한다. 이에 문제가 있을 경우 관리자에 메일을 보내도록 되어 있는데, 이에 대한 자세한 내용은 4.5를 참고하자. 해당 메일을 받았을 때 관리자는 이를 반드시 확인하여 혹시 하드 디스크의 고장이 있을 경우 신속하게 새로운 하드 디스크로 교체하여야 한다.

## 3.5 DNS: Master, Disk, GPU

### dnsmasq: Master

2장에서 언급한 것처럼 komplex에서 내부망을 이루는 컴퓨터들은 고정 IP를 가지고 있고 또 이를 DNS<sup>18</sup>를 통해 사용하기 쉽게 만들어 놓았다. 물론 DNS를 사용하기 위해서 마스터가 DNS Server를 제공해도록 해야 한다. 설치는 어렵지 않은데 아래와 같이 `dnsmasq`을 설정해주고 데몬을 켜주면 된다.<sup>19</sup> `dnsmasq`는 연결되어있는 기기의 mac 주소에 내부 ip를 뿌려주는 역할을 한다. 즉, 스스로 os를 가지면서 자신의 ip를 `netctl` 등으로 설정할 수 있는 disk, kuda의 경우 `dnsmasq.conf`의 정보를 사용하지 않는다. 하지만 4.8절의 다양한 script들이 이를 기반으로 서버에 등록되어 있는 컴퓨터들을 인식하기 때문에, 써주어야 한다.

<sup>15</sup>[https://wiki.archlinux.org/title/RAID#Calculating\\_the\\_stride\\_and\\_stripe\\_width](https://wiki.archlinux.org/title/RAID#Calculating_the_stride_and_stripe_width)

<sup>16</sup>ext4 포맷의 block size는 4KiB로 정해져 있다.

<sup>17</sup>컴퓨터가 부팅되면서 진행될 mount들을 저장해둔다.

<sup>18</sup>Domain Name System. 숫자로 이루어진 ip 주소 대신에 영문으로 이루어진 주소를 사용할 수 있게 해준다.

<sup>19</sup><https://wiki.archlinux.org/title/Dnsmasq>

```
/etc/dnsmasq.conf  
...  
#DHCP server setup  
interface=internal  
dhcp-range=10.0.0.1,10.0.255.254,255.255.0.0,1h  
...
```

실제 가동중인 마스터의 `/etc/dnsmasq.conf`를 확인한다면 위의 DHCP server setup 이외에 TFTP와 PXELINUX관련 설정, 그리고 서버의 다른 노드들에 대한 내용도 추가가 되어 있는 것을 볼 수 있다. 이는 각각 3.7절과 4.8.2절의 `NewNode.sh` 스크립트를 참고하자.

## Hosts: Master, Disk, GPU

다음으로, `/etc/hosts` 파일에 서버에 사용될 노드들의 ip 주소와 hostname을 적어주자. 해당 파일에 등록된 정보들을 기반으로 ip 주소 대신, 설정된 영문 이름(hostname)으로 접근할 수 있게 된다. 작성 후, `/root/admin/hosts.template` 역시 업데이트하여 `NewNode.sh.sh` 가 서버의 새로운 노드의 호스트 파일을 추가할 수 있도록 하자. 자세한 내용은 4.8.2절 참조.

```
/etc/hosts  
...  
# MASTER and TERMINAL  
10.0.0.1      master.snu.ac.kr      master  
10.0.0.2      terminal2.snu.ac.kr    terminal2  
10.0.0.3      terminal3.snu.ac.kr    terminal3  
...
```

## 3.6 NFS: Master, Disk, GPU

NFS(Network File System)는 디스크 서버가 자신이 가지고 있는 저장 장치(디스크 드라이브)를 네트워크의 각 노드들이 사용할 수 있도록 해준다. 네트워크의 노드들이 동시에 디스크 드라이브를 사용하더라도 NFS가 지속적으로 동기화를 해주어 마치 하나의 PC에서 사용하는 것과 같은 효과를 준다. 물론 네트워크 통신을 이용하는 만큼 빠르지 않고 서버가 커질 수록 NFS에 걸리는 부하도 커진다.

NFS 구축은 크게 두 가지 작업으로 이루어지는데, 먼저 디스크 서버를 구축하는 것으로 디스크 서버가 자신이 가지고 있는 디스크 드라이브에 네트워크의 각 노드들이 접속할 수 있도록 설정해주는 것이고 다음으로 네트워크의 각 노드들이 디스크 서버가 제공하는 디스크 드라이브를 사용할 수 있도록 설정해주는 작업이다.

아치리눅스 위키를 참고하면<sup>20</sup> 디스크는 server section을 따라 설치해주면 되고, 마스터와 계산 노드들은 client section을 따라 설치해준다. 디스크와 마스크 모두 패키지 **nfs-utils**를 설치해주어야 한다.

## Server: Disk

먼저 디스크에서 공유할 저장 공간(디렉토리)를 `/etc(exports`에서 해준다. 아치리눅스 위키에서는 nfs로 외부에 제공할 폴더를 만들고 거기에 정말 공유할 디렉토리를 바인딩한 후에 그것을 `fstab`으로 자동으로 마운팅한 다음에 export 하는데 komplex에서는 굳이 그럴 필요 없다. 3.4 절에서 `/pds/pds11`, `/pds/pds12` 등으로 레이드 어레이를 만들었을 것이다. 그렇게 레이드된 어레이를 바로 export 해주자. 예컨데 다음과 같이 설정한다.

```
/etc(exports
/pds/pds11 *(fsid=1,rw,no_subtree_check,no_root_squash)
/pds/pds12 *(fsid=2,rw,no_subtree_check,no_root_squash)
```

설정 파일을 만든 후에 반드시 re-export해준다.

```
# exportfs -arv
```

마지막으로, 디스크는 NFS 서버를 제공해 주는 작업만을 해주기 때문에<sup>21</sup> 사용 가능한 컴퓨팅 자원을 모두 이 작업에 몰아주는 편이 좋다. `/etc/nfs.conf`에 다음과 같이 써주자.

```
/etc/nfs.conf
...
[nfsd]
threads=128
...
```

설정이 끝난 후에는 **nfs-server**와 **rpcbind**를 start/enable해주자.

## Client: Master, GPU

NFS에서는 여러 대의 디스크들이 모두 각각 서버가 되고 마스터와 노드들이 클라이언트가 된다. 물론 디스크들 간에도 서로의 저장 공간을 공유하기 위해서는 디스크가 서버이자 클라이언트가 되도록 해야겠지만 komplex에서는 디스크 간에 서로 저장공간을 공유할 이유가 없기 때문에 각각의 디스크 간에 저장공간을 공유하지 않고 오로지 서버 역할만 하도록 되어 있다.

<sup>20</sup><https://wiki.archlinux.org/title/NFS>

<sup>21</sup>이는 대부분의 경우 마스터도 마찬가지이다. 이후 3.7절에서 마스터 역시 NFS를 사용하여 `/tftpboot`, `/root` 등의 디렉토리를 export할 것인데, 이때도 같은 작업을 해준다.

우선 클라이언트에서 **rpcbind**, **nfs-client.target**, **remote-fs.target**를 모두 start/enable해 주자. 다음을 통해 디스크 서버 *disk*가 NFS로 네트워크를 통해서 저장 공간을 제공하고 있는지 확인할 수 있다.

```
# showmount -e disk
Export list for disk:
/pds/pds11 *
/pds/pds12 *
```

디스크 서버에서의 경로와 마찬가지로, 클라이언트에서도 **/pds/pds11**, **/pds/pds12**의 경로의 빈 디렉토리를 만들자. 그런 다음 **/etc/fstab**에 아래와 같이 마운트 설정을 추가해준다.

```
/etc/fstab
...
disk1:/pds/pds11 /pds/pds11 nfs rw,exec,hard,vers=4 0 0
disk1:/pds/pds12 /pds/pds12 nfs rw,exec,hard,vers=4 0 0
...
```

마운트 할 때의 옵션은 두가지 종류가 있다. 첫번째는 mount 관련 옵션으로, rw 및 exec가 이에 해당한다. rw의 경우 해당 디스크에서 read/write 작업을 할 수 있다는 것이며 exec의 경우 해당 디스크의 binary 파일을 실행할 수 있다는 것을 뜻한다. 두번째로 NFS 관련 옵션으로 hard가 있는데, 이는 인터넷 연결 등의 문제로 NFS request가 time out 되었을 때, 계속해서 다시 시도하라는 옵션이다.

이외의 다양한 NFS의 옵션을 사용할 수 있으며, 이들은 모두 아치위키에 설명되어 있으니 참고하자. 이제 마스터가 부팅될 때 자동으로 disk1의 **/pds/pds11**이 클라이언트의 **/pds/pds11**에 마운트 될 것이다. 재부팅하지 않고 **mount disk1:/pds/pds11 /pds/pds11**로 바로 마운트 할 수도 있지만, 위의 옵션을 사용하지 않는다면 추가적인 문제가 발생할 수 있다. 이 방법은 때로 마운트가 예기치 않게 풀어졌을 때도 사용할 수 있다. 물론 NFS 마운트가 예기치 못하게 풀렸다면 좋지 않은 상황들이 연이어 발생하기에 마운트를 다시 하는 것으로 충분하지 않을 수 있다.

계산 노드 역시 마스터에서 했던 것과 동일한 원리로 NFS의 클라이언트로 설정해주어야 한다. 이는 4.8.2절의 **NewNode.sh**으로 구현되어 있다. 이때 주의할 점은 **/root/admin/fstab.template**를 기반으로 설정되는데, 해당 파일은 마스터의 **/etc/fstab**과는 내용이 다르다. **절대로 마스터의 fstab 을 계산 노드의 fstab 으로 사용하면 안된다.**

### 3.7 PXE: Master, Node

PXE 부팅은 diskless cluster의 핵심이 되는 기술로 계산 노드들이 마스터를 통해서 OS를 받아 부팅되도록 한다. 이 과정이 서버 구축에서 가장 어려운 부분일 것이다. 먼저 PXE 부팅을 다루기 전에 일반적인 리눅스 PC의 부팅 과정을 살펴보자.

- 전원을 키면 BIOS 또는 UEFI가 부트로더(boot loader)를 실행한다. 리눅스에서 주로 부트로더는 **grub** 혹은 **systemd-boot**를 사용한다.
- 부트로더가 커널(kernel)과 램디스크(initial ramdisk)의 이미지를 로드하고 루트 프로세트(**systemd**, init)가 실행된다. (low-level boot process)
- systemd**가 예약된 서비스와 데몬들을 실행한다. 모든 서비스들이 실행되면 부팅이 완료된다. (high-level boot process)

이와 비교해서 노드에서 PXE 부팅이 이루어지는 과정은 다음과 같다.

- 노드에 전원을 키면 BIOS가 NIC로 네트워크에 부트로더를 요청하는 broadcasting을 한다.
- 마스터의 DHCP가 계산 노드의 broadcasting을 확인하고 등록된 NIC인 경우 해당 노드에 IP를 제공한다.
- 마스터의 TFTP가 노드에게 부트로더에 대응되는 NBP(network bootstrap program)을 제공한다. komplex의 경우에는 syslinux의 pxelinux.0를 쓴다.
- 노드의 NBP는 마스터에 저장되어 있는 커널과 램디스크 이미지를 불러온다. 이 과정에서 마스터에 저장된 노드의 시스템 파일이 노드에 마운트한다.
- systemd**가 예약된 서비스와 데몬들을 실행한다. 모든 서비스들이 실행되면 부팅이 완료된다.

PXE 부팅을 구현 방법은 여러 가지가 있을 수 있는데, komplex에서는 DHCP와 TFTP를 **dns-masq** 데몬으로 구현하고 세부적으로 NFS와 syslinux의 PXELINUX를 이용하여 PXE 부팅한다. 아치리눅스 위키에서 Diskless system<sup>22</sup> 페이지, PXE 페이지<sup>23</sup>, syslinux 페이지의 PXELINUX<sup>24</sup>를 참고하도록 하자.

### 3.7.1 Preparation: Node

노드에서 먼저 PXE 부팅이 될 수 있도록 BIOS를 설정 해주도록 하자. 마더보드마다 다르지만 전원이 들어올 때 F12키 또는 Delete키 등을 눌러주면 BIOS 설정으로 진입할 수 있다. 마더보드마다 구체적인 항목은 다를 수 있는데, 대체로 Networking Booting 혹은 PXE Booting과 관련된 항목을 찾아서 활성화해주면 된다. 참고로 노드에 HDD가 있다면 쓰지 않기 때문에 제거해주고 BIOS 설정에서 SATA 설정도 비활성화해주자.

<sup>22</sup>[https://wiki.archlinux.org/title/Diskless\\_system](https://wiki.archlinux.org/title/Diskless_system)

<sup>23</sup><https://wiki.archlinux.org/title/PXE>

<sup>24</sup><https://wiki.archlinux.org/title/syslinux#PXELINUX>

### 3.7.2 Installation: Master

마스터에서 다루어야 할 설정이 매우 많지만 앞서 말한 PXE 부팅 과정을 염두해두고 필요한 것들을 하나씩 채워간다고 생각하면 이해가 쉬울 것이다.

#### DHCP + TFTP

앞에서 Node가 BIOS를 통해서 네트워크에 IP를 얻기 위해서 broadcasting를 하고 있으므로 마스터에서는 이를 받고 IP를 할당해주어야 할 것이다. 이 작업은 3.5절에서 다룬 `/etc/dnsmasq.conf`에 각 노드의 NIC의 mac 주소에 대하여 IP를 적어놓아야 한다. 4.8.2의 `NewNode.sh` 스크립트에서 대신 진행해 주기 때문에, 지금 단계에서는 해주지 않아도 무방하다.

노드가 마스터로부터 IP를 받았으니 NBP를 실행시켜야 한다. NBP를 비롯한 커널과 램디스크 이미지를 보내주기 위해서 TFTP 서버를 구축해야 한다. `dnsmasq` 데몬이 TFTP 서버 역할도 해줄 수 있다. komplex에서는 ‘특별히’ PXE 부팅을 하기 위한 모든 파일들을 마스터의 `/tftpboot/`에 모아두고 있다. `dnsmasq`의 설정에 다음과 같이 PXE 부팅을 위한 설정을 추가해주도록 하자.

```
/etc/dnsmasq.conf

#PXELINUX setup
dhcp-boot=pxelinux.0
dhcp-option-force=208,f1:00:74:7e
dhcp-option-force=210,/tftpboot/

#TFTP server setup
enable-tftp
tftp-root=/tftpboot
```

만약 Mini komplex를 구축 중이라면 마스터에 `/tftpboot`라는 폴더가 없을텐데 지금 만들어 주고 그 안에 부팅 방식 `boot`에 따라 `NBP`와 `core-module`을 복사해 주자.<sup>25</sup> (TO BE DONE FOR UEFI). 더불어서 각 노드들의 `NBP`의 설정 파일들이 담길 `/tftpboot/pxelinux.cfg` 폴더를 만들어주자.

```
# cp /usr/lib/syslinux/boot/NBP /tftpboot/
# cp /usr/lib/syslinux/boot/core-module /tftpboot/
# mkdir /tftpboot/pxelinux.cfg
```

다음으로, NBP는 커널(`vmlinuz-linux`)과 램디스크(`initramfs-komplex-node.img`)의 위치를 `/tftpboot/pxelinux.cfg/01-node-mac-addr`를 보고 찾는다. 예컨데 terminal2의 mac address가 `terminal-mac-addr`이라고 하면 terminal2의 pxelinux 설정은 다음과 같다.

<sup>25</sup> BIOS의 경우 `boot`는 `bios`, `pxelinux.0` 와 `ldlinux.c32`. UEFI의 경우 `boot`는 `efi64`, `syslinux.efi` 와 `ldlinux.e64`이다.

```
/tftpboot/pxelinux.cfg/01-terminal-mac-addr

default terminal2
label terminal2
kernel vmlinuz-linux
append initrd=initramfs-komplex-node.img rootfstype=nfs root=/dev/nfs
nfsroot=10.0.0.1:/tftpboot/terminal2,rsize=16384,wszie=16384
ip=dhcp BOOTIF=01-\{it terminal-mac} vga=773
```

위에서 마지막 세 줄은 본래 붙여서 한 줄로 쓴다. 설정을 보면 커널과 램디스크 이미지의 파일명과 NFS로 파일 시스템을 마운트하는 것을 확인할 수 있다.

### Kenel image

커널 이미지는 마스터의 커널 이미지를 그대로 사용해도 무방하다.

```
# cp /boot/vmlinuz-linux /tftpboot/
```

마스터의 리눅스가 업데이트되면 커널 이미지가 새롭게 만들어지는데 이를 노드들에게도 적용하기 위해서 다시 마스터의 커널 이미지를 복사하고 계산 노드의 램디스크를 새로이 만들어 주어야 한다. 자세한 내용은 4.2절을 참고하자.

### Initial ramdisk

램디스크 이미지를 만드는 과정이 좀 까다롭다. 왜냐하면 마스터와 노드들의 하드웨어의 구성이 다르고 각 노드마다 설정을 다르게 해줄 부분이 있기 때문이다. 특히 마스터와 터미널의 경우에는 NIC가 두 개이고 그 중에는 외부로 연결되어 있는데 계산 노드의 경우에는 NIC를 내부로만 연결되어 있는 것을 고려해줘야 한다. 그러므로 각 노드 별로 작동할 수 있도록 램디스크를 만들어주어야 한다. 일반적으로 arch linux의 설치 과정에는 다음과 같은 작업이 **pacstrap**을 사용할 때 숨어있다.

```
# mkinitcpio -p linux
```

이는 기본적으로 만들어진 `/etc/mkinitcpio.d/linux.preset`를 이용해서 램디스크 만든다는 것을 의미한다. `linux.preset`를 보면 `/etc/mkinitcpio.conf`를 따라 램디스크가 만들어지는 것을 볼 수 있다. 이제 각 노드들을 위한 램디스크를 만들어 보도록 하자. 먼저 다음과 같이 노드를 위한 preset를 만들어 준다.

```
/etc/mkinitcpio.d/komplex.preset

ALL_config="/etc/mkinitcpio-komplex-node.conf"
ALL_kver="/boot/vmlinuz-linux"
PRESETS=( 'default' 'fallback' )
default_image="/tftpboot/initramfs-komplex-node.img"
```

```
fallback_image="/tftpboot/initramfs-komplex-node-fallback.img"
fallback_options="-S autodetect"
```

preset을 보면 이번에는 `/etc/mkinitcpio-komplex-node.conf` 을 참고해서 램디스크를 만들게 되어 있고 만들어진 이미지는 `/tftpboot/initramfs-komplex-node.img` 으로 저장되도록 되어 있다. `/etc/mkinitcpio-komplex-node.conf` 는 다음과 같이 작성한다.

```
/etc/mkinitcpio-komplex-node.conf
MODULES="e1000 e1000e r8169 8139cp 8139too bnx2 bnx2x tg3"
BINARIES="/sbin/mount.nfs"
FILES=""
HOOKS="base udev net-komplex autodetect modconf block shutdown filesystems
      keyboard mount-komplex1 persistent-if-name-komplex"
```

`/etc/mkinitcpio.conf` 와 비교하면 MODULES과 BINARIES에 몇 가지가 추가되고 HOOKS에 `net-komplex`, `mount-komplex1`, `persistent-if-name-komplex`이 추가된 것을 볼 수 있다. MODULES에 추가된 것은 노드마다 다른 NIC 모듈들을 모두 써준 것이다. HOOKS에 추가된 세 개의 custum hook은 이덕재 박사님이 작성하신 kernel hook script이다. custum hook은 `/etc/initcpio/install` 과 `/etc/initcpio/hooks` 에서 그 내용을 확인할 수 있다.

이 중, `persistent-if-name-komplex` 만 이해하면 될 것이다. 이것은, NIC가 2개로 구성된 터미널을 위한 hook이다. 이외에 다른 두개의 custom hook에 대한 구체적인 내용이 필요하다면 6.2 절을 참고하자.

```
/etc/initcpio/install/persistent-if-name-komplex
#!/bin/bash
build() {
    local rules tool
    add_file "/root/admin/70-persistent-if-name-komplex.rules"
}
help() {
    cat <<HELPEOF
This hook will set network interface names of komplex2 and komplex3
HELPEOF
}
```

`persistent-if-name-komplex` 은 따로 runtime hook이 없고 대신에 NIC 인터페이스의 이름을 설정하도록 되어 있는데 다음과 같다.

```
/root/admin/70-persistent-if-name-komplex.rules
```

```
#TERMINAL2
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="84:2b:2b:b6:01:a7", NAME="internal"
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="00:08:9f:d8:aa:45", NAME="external"
```

4.8.2절의 `NewNode.sh` 스크립트가 해당 파일에 각 계산 노드들의 정보 역시 추가해 주는데, 이를 통해 해당 파일이 수정되었다면, 램 디스크 이미지를 아래와 같이 새로 만들어 주어야 한다.

```
# mkinicpio -p komplex
```

이제 komplex 노드의 램디스크 이미지와 fallback 이미지 `initramfs-komplex-node.img` 와 `initramfs-komplex-node-fallback.img` 가 `/tftpboot/`에 생성된 것을 확인할 수 있다.

### File system mount: NFS

램디스크의 설정에서 마스터의 `/tftpboot/`에 있는 노드의 파일 시스템을 노드에 NFS를 이용해서 마운트하는 것을 확인했을 것이다. 3.6절에서는 NFS를 이용해서 디스크가 마스터에게 자신의 저장공간을 제공하게 했다면 이번에는 마스터가 노드에게 파일 시스템을 담은 저장공간을 NFS로 제공하도록 할 것이다. 이미 램디스크를 만들면서 노드가 마스터의 `/tftpboot/node`를 자신의 파일 시스템으로 마운트하도록 설정했다는 것을 염두해두자.

일단 마스터에서 파일 시스템을 제공할 수 있도록 마스터의 `/etc/exports`를 다음과 같이 작성해주자.

```
/etc/exports
```

```
/tftpboot *(rw,no_root_squash,no_subtree_check)
/boot      *(rw,no_root_squash,no_subtree_check)
/home     *(rw,no_root_squash,no_subtree_check)
/opt       *(rw,no_root_squash,no_subtree_check)
/root     *(rw,no_root_squash,no_subtree_check)
/usr      *(rw,no_root_squash,no_subtree_check)
```

그리고 3.6에서 디스크 서버에서 해준것과 마찬가지로 `# exportfs -arv` 으로 설정을 re-export 해주어야 한다. 또한 `nfs-server`와 `rpcbind`를 start/endable 해준다. 이것으로 마스터는 NFS 서버와 클라이언트 역할을 모두 하게 된다.

마지막으로 남은 작업은 마운트할 파일 시스템을 노드별로 만들어주는 일이다. 예컨대 terminal2를 위해서 `/tftpboot/terminal2` 폴더를 만들고 그 아래에 `boot`, `etc`, `home`, `sys` 등의 폴더와 파일을 마스터에서 복사해오되, 노드 개별의 설정에 맞게 수정을 해주고 `usr`와 같이 노드들과 마스터가 공동으로 사용할 있는 파일은 링크를 걸어준다. 또한 디스크의 저장장치를 마운트 할 `/pds/pds11` 등과 같은 폴더도 만들어주어야 한다. 이러한 작업을 모든 계산 노드에 대해서 해주어야 하기 때문에 수작업으로 하기 벅찰 것이다. komplex에서는 4.8.2절에서 살펴볼 `NewNode.sh`

가 이 작업을 수행한다. `NewNode.sh`를 보면 이 작업이 구체적으로 어떻게 진행되는지 살펴볼 수 있을 것이다. 여기까지 진행되면 PXE 부팅을 위한 과정은 끝난 것이다.

### 3.8 NTP: Master, Disk, Node

NTP(Network Time Protocol)는 서버의 노드들의 시간을 동기화 해주는 역할을 해준다. 만약 시간을 동기화해주지 않는다면 NFS를 비롯해 네트워크를 통해서 이루어지는 작업에 문제가 발생할 수 있다.

komplex는 외부망에 접근할 수 있는 노드가 제한되어 있기 때문에 모든 노드가 외부 NTP 서버로 동기화할 수 없다. 따라서 터미널의 internet sharing을 통해 외부 인터넷과 연결된 마스터만 외부 NTP 서버에 시간을 받아오고 마스터가 다시 내부 NTP 서버가 되어서 디스크와 계산 노드들에게 시간을 제공하도록 되어 있다.

이제 아치리눅스 위키의 NTP 페이지<sup>26</sup>를 따라 NTP를 설치하자. 일단 마스터와 디스크 모두에 `ntp`를 설치해주도록 하자. 앞에서 설명해준 것처럼 마스터를 외부 NTP 서버에서 시간을 받아오기 위해 다음 옵션을 넣어준다.

```
master: /etc/ntp.conf

# For additional information see:
# - https://wiki.archlinux.org/index.php/Network_Time_Protocol_daemon
# - http://support.ntp.org/bin/view/Support/GettingStarted
# - the ntp.conf man page

# Sync always panic threshold
tinker panic 0

# Associate to Arch's NTP pool
server 0.asia.pool.ntp.org
server 1.asia.pool.ntp.org
server 2.asia.pool.ntp.org
server 3.asia.pool.ntp.org

# By default, the server allows:
# - all queries from the local host
# - only time queries from remote hosts, protected by rate limiting and kod
server 127.127.1.0
```

---

<sup>26</sup>[https://wiki.archlinux.org/title/Network\\_Time\\_Protocol\\_daemon](https://wiki.archlinux.org/title/Network_Time_Protocol_daemon)

```

restrict default ignore
restrict 10.0.0.0 mask 255.255.0.0 noquery nopeer nomodify
restrict 127.0.0.1
restrict ::1
interface ignore ipv6
interface listen ipv4

# Location of drift file
driftfile /var/lib/ntp/ntp.drift
logfile /var/log/ntp.log

```

이제 설정은 완료되었고 `ntpd -u ntp:ntp`를 통해 ntp를 키도록 하자.<sup>27</sup> 그리고 부팅할 때 ntp가 자동으로 자동되도록 하기 위해서 `systemctl enable ntpd`를 해준다.

디스크에서는 NTP 서버를 아치에서 제공해 주는 서버가 아닌, 마스터로 지정해 주면 된다.

```

disk: /etc/ntp.conf

tinker panic 0

# Sync with the master server
server 10.0.0.1

# Only allow read-only access from localhost
restrict default nopeer noquery
restrict 127.0.0.1
restrict ::1

# Location of drift file
driftfile /var/lib/ntp/ntp.drift
logfile /var/log/ntp.log

```

계산 노드 역시 디스크와 마찬가지 이지만, 네트워크의 부담을 줄이기 위해 logging을 하지 않는다. 터미널을 비롯한 계산 노드들은 `/root/admin/ntp.conf.template`을 기반으로 4.8.2절의 `NewNode.sh`를 통해 만들어진다.

```

/root/admin/ntp.conf.template

# Sync always ignoring panic threshold.
tinker panic 0

```

<sup>27</sup>이는 `ntp`라는 이름의 user와 group으로서 서비스되도록 한다.

```

# Sync with the master server
server 10.0.0.1

# Only allow read-only access from localhost
restrict default noquery nopeer
restrict 127.0.0.1
restrict ::1

```

그 다음으로 디스크와 계산 노드에서도 마스터와 마찬가지로 ntp를 켜주고 **ntpd**를 enable해주면 된다.

### 3.9 Mailing: Master

komplex는 서버의 현황을 그룹원들에게 메일로 알려주고 있다. 예를들면 4.9.5절의 admin report와 같이 현재 서버의 가동 현황을 보고하는 등의 메일이 있겠다. 이러한 메일링 서비스는 **msmtp**를 사용한다.<sup>28</sup> 설치를 완료하였다면, 메일을 보내기 위한 계정 정보를 설정해 주어야 한다. 일반적으로 계정 정보는 `/root/.msmtprc`에 저장된다. 아치 위키에서도 나와있듯 반드시 권한설정을 600으로 해주어야 한다. 현재 연구실에서 사용하는 계정 정보는 다음과 같다.

```

/root/.msmtprc

account      cnrc
auth         plain
tls          off
host         mulli.snu.ac.kr
from         cnrc@phya.snu.ac.kr
user         cnrc
password     dummy_passwd_for_manual

```

위의 계정 정보에서 보면 알겠지만, 계정의 비밀번호가 평문으로 되어있다. 이에 따라 해당 파일이 외부에 나가지 않도록 매우 조심하여야 할 것이다.

계정의 설정이 끝났다면, **msmtp**를 이용해서 다음과 같이 메일을 보낼 수 있다.

```
# echo "hello from komplex" | msmt -a account to_user@domain.server
```

실제로 komplex에서 보내는 정기적인 메일들은 모두 특별한 조건이 되었을 경우 보내지도록 서비스의 형태로 관리되고 있다. 자세한 내용은 4.9절을 참고하자.

---

<sup>28</sup><https://wiki.archlinux.org/title/msmtp>

## Chapter 4

# Komplex 운영

앞 장에서는 komplex를 예로 들어서 disk-less server를 구축하는 방법을 다루었다. 이번 장은 komplex 서버 자체에 좀 더 초점을 두어서 komplex에서 특별히 사용하는 service나 bash script 그리고 job manager인 spg 등을 다루도록 하겠다. 더불어서 정기 전기 점검 시에 일어날 정전에 대비해서 서버를 재가동하거나 계산 노드의 고장을 수리하거나 새로운 사용자를 추가하거나 등의 서버 관리자가 komplex를 운영하면서 정기적으로 마주칠 업무들을 다루게 될 것이다.

### 4.1 Turn-On/Off

많지는 않지만 때때로 komplex 서버 전체를 물리적으로 전원 끄고 다시 켜야할 상황이 생긴다. 특히, 1년에 한번씩 정기적으로 있는 정전의 경우 약 한달 전부터 공지가 내려오며, 관리자는 해당 공지를 확인하고 최소한 일주일 전까지 전체 그룹원들에게 서버 다운의 공지를 해야 한다. 이외에도 예상치 못한 문제로 부득이하게 서버를 꺼야 할 때도 있을 것이다.

서버를 끄고 키는 일은 매우 조심해서 해야 하는 작업으로 잘못된다면 매우 오랜 시간동안 고생해야 할수도 있기에 반드시 주의를 기울여 조심스럽게 해야 한다.

#### Power Off

우선 `# spg job -a`, `# spg user` 등의 명령어로 현재 돌아가고 있는 작업들을 확인하자. 꺼지지 않았다면 root 계정으로 `# spg KILL -u user` 등으로 작업을 꺼 주어야 한다. **spg**에 대한 자세한 내용은 4.10를 참고하자. 또한, 이 기회에 서버를 업데이트 하는 것도 고려해 주어야 한다. 일반적으로 재부팅을 하지 않아도 업데이트 사항이 잘 적용되지만 업데이트 된 패키지들에 따라 컴퓨터의 재부팅을 요구하는 것도 있다. 4.2를 참고하여 업데이트를 진행하기를 권장한다.

모든 작업이 완료된 것을 확인하면, 4.5.3절에서 구체적으로 서술하겠지만, 디스크 서버에 scrubbing 작업을 해주어야 한다. 이는 정기적인 정전에 대해서만 적용되는 사항으로 1년에 한번꼴로 진행한다고 생각하면 될 것이다. 이 작업은 최소 4시간 이상 걸리며, 약 12시간 정도의 여유시간을 가지고

진행하는 것이 좋다. 예를 들어 예고된 정전 시간이 토요일 오전 9시 부터라면, 금요일 오전 (혹은 목요일 저녁)까지 그룹원들의 작업의 마무리를 부탁하고, 디스크의 scrubbing 작업에 들어가자.

Scrubbing 작업까지 마무리 되었다면, 실제로 서버를 이루는 컴퓨터들을 꺼야 한다.<sup>1</sup> 반드시 각 컴퓨터들을 아래의 순서에 맞추어 꺼야 한다.

1. 계산 노드
2. GPU 머신
3. 터미널
4. 마스터
5. 디스크

계산 노드와 터미널들의 경우 `# ssh node-name poweroff`, `# poweroff` 등으로 끌 수 있지만, 마스터와 디스크의 경우는, 직접 서버실에서 꺼주어야 한다.

## Power On

서버를 켜 때에는, 끄는 순서의 역순으로 하여야 한다. 또한 끌때와는 반대로 각 컴퓨터들을 물리적으로 일일이 전원 버튼을 눌러줄 수밖에 없다.<sup>2</sup> 디스크와 GPU 서버와 같이 os를 가지고 있는 서버들의 경우 바로바로 전원을 켜주어도 괜찮다.

계산노드의 경우 네트워크로부터 마스터에 있는 os를 받아와야 하는데, 다수의 계산노드들을 한번에 켜다면 메인 스위치 혹은 마스터의 네트워크에 큰 부하를 줄 수 있다. 이럴 경우 os를 제대로 받아오지 못하거나 최악의 경우 스위치가 고장나는 등의 문제가 생길 수 있다. 따라서 마스터의 `journald`를 통해 노드들이 정상적으로 os를 받아가는지 확인하며 천천히 켜주어야 한다.

## 4.2 Update

패키지를 새로 설치하는 경우에 대체로 바로 설치가 가능하지만 때때로 core library나 pacman library를 업데이트 해줄 필요가 생기는 경우가 있다. 또한 사용자들이 주로 사용하는 패키지의 최신 버전을 사용하거나 리눅스 시스템에 발생한 버그를 잡기 위해서도 업데이트를 해주어야 되는 경우도 있다. 더불어 OpenSSH와 같은 패키지를 너무 오랜된 버전으로 사용하고 있다면 보안상에 문제가 생길 수도 있어 업데이트 해주어야 하기도 한다. 또한 업데이트를 매우 오랫동안 하지 않을 경우에 막상 업데이트가 필요한 경우 업데이트가 실패될 수도 있으므로, 정기 점검으로 인한 정전이나 불시의 문제로 서버를 재가동해야 하는 상황이 생긴다면 틈틈히 서버를 업데이트 해주도록 하자.

---

<sup>1</sup> 2번의 GPU머신 까지는 scrubbing이 완료되기 전에 종료해도 무방하다. 이론적으로 터미널까지도 종료해도 괜찮지만, 그럴경우 원격접속이 어려워질 수 있다.

<sup>2</sup> WOL(Wake-On Lan)기능을 사용할 수도 있지만, 이를 위해서는 서버실에 있는 모든 네트워크 스위치를 해당 기능을 지원하는 것으로 교체해야 한다.

**pacman**를 포함한 아치리눅스에서 설치된 패키지들은 `# pacman -Syyu`를 통해서 모두 업데이트 할 수 있다. `# pacman -Syy`로 저장소를 먼저 업데이트하고 `# pacman -Syu`로 **pacman**과 패키지들을 업그레이드 해줘도 된다. 미러리스트가 오래된 경우에 업데이트가 잘 안되거나 느릴 수 있는데, `# pacman -Sy pacman-mirrorlist`로 미러리스트를 먼저 다운받고 패키지 업데이트를 하는 것도 좋다.

## Update for komplex

마스터와 리눅스가 별도로 설치된 디스크와 GPU 서버는 **pacman**을 이용해서 먼저 리눅스 업데이트 해준다. 외부망에 연결하기 위해서는 3.3.4절을 참고하자. 마스터가 업데이트 한 후에 계산 노드를 업데이트를 해주기 위해서는 PXE 부팅에서 komplex용 램디스크를 만들었던 과정을 다시 반복해주어야 한다. 이 작업은 4.8.6절의 `UpdateSystemFilesOnNodes.sh`가 대신 해 줄 것이다. 해당 스크립트를 통해 터미널과 계산 노드들을 업데이트 해 주자.

## Update records

업데이트를 하며 주요 변경점이 있던 기록은 다음과 같다.

- (16.08.19) 디스크 및 마스터 업그레이드: 마스터에서 아치리눅스 패키지 업데이트 시에 pacman-keyring 문제가 조금 있던 것 외에 큰 문제가 없었다. 몇몇 노드에서 SSH접속이 느리던 문제가 업데이트 이후에 사라졌고 덕분에 **spg**의 속도가 향상되었다.
- (21.08.12) 마스터 업그레이드: 방화벽을 담당하는 **iptables**를 업데이트 하니 MASQUERADE가 안되는 문제가 있었다. 이는 마스터를 재부팅 함으로 해결하였다.
- (21.08.13) 방화벽 서비스를 **iptables**에서부터 **nftables**로 이전하였다. 이는 주로 **iptables-translate** 유틸리티를 사용하여 이전에 있던 iptables의 규칙들을 nftables의 규칙으로 바꾸는 작업이었다. 추가로, iptables에 default로 존재하지만 nftables에는 존재하지 않는 기본 table에 대한 설정을 바꾸어 주었다.
- (21.12.21) 파이썬의 메이저 버전이 3.9에서 3.10으로 업그레이드 됨에따라, **spg** 등을 위해 설치하였던 패키지들을 다시 한번 **pip**으로 설치해야 하였다.

## pacman Keyring Update

마스터 혹은 디스크의 경우에 업데이트 주기가 길기 때문에 업데이트 하는 도중에 keyring에 문제가 생길 때가 있다. 이 경우에는 `# pacman -Sy archlinux-keyring`으로 key들을 재구성해 주기를 바란다. 그럼에도 문제가 생긴다면 `# pacman-key --refresh-keys`로 모든 키를 refresh 해주고 다시 시도해 보자.

## 4.3 계정 관리

계정과 그룹 관리에 관해서는 아치리눅스 위키의 users and group 페이지를 참고하도록 하자.<sup>3</sup>

### 계정 정보

komplex를 이용하는 계정들의 목록과 설정은 `/etc/passwd`에서 확인할 수 있다.

```
/etc/passwd
```

```
ntp:x:87:87:Network Time Protocol:/var/lib/ntp:/bin/false  
newbi:x:1594:1594:name(e-mail):/pds/pds21/newbi:/bin/bash
```

‘계정이름:x:계정번호:그룹번호:계정 설명:홈디렉토리:로그인 쉘’의 순으로 되어 있다. 참고로 시작 디렉토리는 로그인 했을 때의 워킹 디렉토리(working directory)를 말한다. 참고로 계정번호가 1000보다 작은 것들은 시스템을 위한 계정들이고 1000이상이 사용자 위한 계정들이다. 따라서 시스템 계정들을 만드리는 일이 없도록 하자.<sup>4</sup>

시스템 계정이 아닌 실제 사용자들은 모두 `users` 그룹에 속해있다. 따라서 등록된 사용자는 `$ getent group users`를 통해 확인할 수 있다.

#### 4.3.1 계정 생성

계정의 이름은 15 글자를 넘어가지 않도록 하자<sup>5</sup>. 미래의 서버 관리자가 해당 계정의 주인을 헷갈리지 않기 위해 가능하면 사용자의 이름으로 하는것이 관리하기 편할 것이다.

새로운 계정은 일반적인 pc에서 계정을 만드는것과 다를 바 없다. 계정의 이름과 홈 디렉토리를 설정해 주고, `users` 그룹에 해당 계정을 넣어준다. 이러한 작업은 4.8.4절의 `NewUser.sh`를 사용하여 만들 수 있다. 예컨데 `newbi`라는 계정을 `/pds/pds21/newbi`를 홈 디렉토리로 하도록 만들어보자.

```
# cd /root/admin  
# ./NewUser.sh newbi /pds/pds21 comment
```

위의 내용을 실행하면, `newbi`의 비밀번호를 물어보게 된다. 비밀번호는 영문과 숫자, 그리고 특수 문자를 조합하여 12자리 이상으로 해주자. 해당 기준은 komplex에서 계정을 관리하는 PAM에서의 설정으로 정해진다. 자세한 내용은 4.6.2을 참고하자.

마스터에서 계정을 만들었으면 서버 전체에서 계정이 사용될 수 있도록 디스크와 계산 노드에서 모두 계정을 추가해주어야 한다. 각각 노드와 디스크에 접속해서 위의 작업을 반복할 수도 있겠지만 계정 및 그룹의 파일을 복사하는 편이 쉽다. 계산 노드의 경우에는 마스터의 `/etc/passwd`,

<sup>3</sup>[https://wiki.archlinux.org/title/users\\_and\\_groups](https://wiki.archlinux.org/title/users_and_groups)

<sup>4</sup>시스템 계정으로 사용되는 `nobody` 계정은 65534를 사용하니 주의하자.

<sup>5</sup>리눅스 자체에서는 사용자 이름 길이에 대한 제한이 있지만, `spg`에서 15자 이상이 넘어가면 약간의 문제가 생길 수 있다.

`/etc/group`, `/etc/shadow`, `/etc/gshadow` 파일들을 그대로 각 계산 노드의 파일 시스템 아래에 복사해주면 된다. OS가 따로 깔려있는 경우 시스템 계정들의 UID가 다를 수 있기 때문에 계정 파일과 그룹 파일을 그대로 복사하면 안된다. 따라서 계산 노드를 위한 `UpdateUserInfoOnNodes.sh` 와 OS가 설치된 노드를 위한 `UpdateUserInfoOnOSServers.sh` 를 각각 실행해 주어야 한다. 예를 들어 터미널 2,3번과 디스크 1,2번에 업데이트 하기 위해서는 다음과 같다.

```
# cd /root/admin  
# ./UpdateUserInfoOnNodes.sh terminal 2 3  
# ./UpdateUserInfoOnOSServers.sh disk 1 2
```

### 4.3.2 계정 삭제

계정을 삭제하는 방법은 계정을 추가하면서 넣어준 정보들을 지워주면 된다. 우선 마스터에서 users 그룹에서 계정을 지우고, 계정 삭제를 진행한다.

```
# gpasswd -d newbi users  
# userdel newbi
```

위 작업은 4.8.4절의 `DeleteUser.sh` 가 해줄 것이다. 이제 계정 생성에서 했던 것처럼 서버 전체에서 계정이 삭제될 수 있도록 디스크와 계산 노드에 적용을 해주어야 한다. 계정을 생성했을 때와 마찬가지로 `UpdateUserInfoOnNodes.sh` 와 `UpdateUserInfoOnOSServers.sh` 를 사용하자.

마지막으로 계정이 사용했던 홈디렉토리를 제거한다. 데이터를 보관할 경우에는 굳이 제거하지 않아도 무방하다.

### 4.3.3 계정 정보 수정

위의 계정 추가 및 삭제를 비롯하여 계정의 여러가지 정보들을 변경해 주기 위해서는 우선 마스터에서 아래의 작업을 해 주어야 한다. 그리고 나서 4.8.5절의 `UpdateUserInfoOnNodes.sh` 와 `UpdateUserInfoOnOSServers.sh` 를 사용하여 모든 노드들에 계정 정보를 갱신해 주도록 하자.

- 계정의 비밀번호는 `# passwd newbi` 라고 쳐준 후에 현재 비밀번호와 바꿀 비밀번호 입력해 주면 된다.
- 계정의 홈 디렉토리를 변경하기 위해 우선 새로운 홈 디렉토리 `newhome`을 만들어 주자. 그리고 다음을 통해 해당 디렉토리의 소유권을 바꾸어 주어야 한다.

```
# chown -R newbi:newbi newhome
```

그리고 `# usermod -d newhome newbie` 으로 바꾸어 줄 수 있다. 기존의 홈디렉토리에 있는 계정 관련 파일이나 디렉토리 예컨데 `~/.bashrc`, `~/.ssh/` 와 같이 숨겨진 파일과 디렉토리들은 새로운 홈 디렉토리로 옮겨준다.

- 계정의 로그인 쉘은 `# usermod -s shell newbie`로 바꾸어 줄 수 있다. 예를 들어 zsh로 로그인 쉘을 바꿀 경우, `shell`은 `/bin/zsh`가 된다.
- 계정 소유자의 이메일 주소등이 변경되었을 경우, 계정의 comment를 바꾸어 주어야 한다. 이는 `# usermod -c "comment" newbie`로 바꾸어 줄 수 있다. 현재 komplex에서 사용하는 `comment`은 영문 실명(이메일 주소)의 형식을 따른다.

## 4.4 노드 관리

### 4.4.1 노드 추가

새로운 노드를 클러스터에 추가하기 위해서는 우선 master에서 `/etc/hosts`에 추가하고자 하는 노드가 있는지 확인하여야 한다. 클러스터에 속해있는 모든 노드들에 적용되어야 하기 때문에 한번 수정하면 전체 적용하기 번거롭다. 따라서 일반적으로 현재 상황보다 여유있게 적어놓지만, 점점 많은 노드를 추가할수록 반드시 확인해 주고, 만약 부족하다면 업데이트 해주자.

만약 `/etc/hosts`를 수정하였다면, 반드시 `/root/admin/hosts.template` 역시 수정하여야 한다. 또한, 4.8.1절의 `CopyToNodes.sh`를 통해 `/etc/hosts`를 업데이트 해주고, OS가 있는 디스크 및 GPU서버의 경우 `scp`를 사용해 마스터의 내용을 각 노드로 복사해 해주도록 하자.

#### CPU Worker

보통의 CPU 계산 노드는 다음과 같이 설치할 수 있다.

- 노드 BIOS에서 PXE 부팅 설정, Mac 주소 확인<sup>6)</sup>
- 4.8.2절의 `NewNode.sh` 스크립트를 통해 마스터에서 노드를 위한 PXE, NFS, NTP, SSH, DHCP 등의 설정과 파일 시스템 구성
- 서버실에 노드 배치 (전원 및 랜선 연결)
- 노드 부팅
- 4.10절을 참고하여 SPG에 노드 정보 추가

#### TERMINAL

위의 CPU와 거의 같은 작업을 한다. 하지만, 3번 항목 이후 다음과 같은 추가적인 작업을 진행해 주어야 한다.

---

<sup>6)</sup>랜선을 연결해야 바이오스 상에서 맥 주소를 확인할 수 있는 경우도 있다.

- 외부 인터넷에 연결하기 위해 중앙전산원에 고정 ip를 신청하자. ssh를 위한 포트는 4.6.1절에서 언급하겠지만 56번 포트를 사용할 것이기 때문에 추가적인 포트 신청은 필요하지 않다. 터미널에는 두개의 NIC를 장착하기 때문에, external로 사용할 mac 주소로 신청해야 한다.<sup>7</sup>
- 3.7.2절을 참고하여 /root/admin/70-persistent-if-name-komplex.rules에 외부와 연결될 NIC 관련 설정을 추가하자. 해당 파일을 수정하면서 램디스크 새로 만들어주어야 하기 때문에 # mkinitcpio -p komplex 까지 해주어야 한다.
- 4.9.1절과 4.9.4절을 참고하여 고정 ip 설정과 방화벽 설정을 해준다. 마스터의 Firewall-Blacklist-update와 터미널의 ExternalNetwork-terminal Firewall-terminal에 관련된 서비스 유닛을 적절한 경로에 위치시켜주고 그에 대응하는 실행 파일까지 업데이트 해야 한다.
- NewNode.sh에서 사용하는 /root/admin/sshd\_config.template는 일반 계산 노드에 적용되는 설정이다. 4.6.1을 참고하여 터미널의 /etc/sshd\_config의 ListenAddress 항목을 수정해주어야 한다.

## DISK

- 3절에서 디스크 서버에 해당하는 과정을 모두 수행해 주어야 한다. 특히, RAID 어레이를 잘 만들도록 하자. 디스크 서버는 직접 os를 가지고 있기 때문에, CPU항목의 2번과 3번 과정은 건너뛰어도 된다.
- 클러스터의 root와 디스크에서의 root가 내부망을 통해 서로 ssh접속을 할 수 있도록 각자의 id\_rsa.pub 키를 서로의 authorized\_keys에 추가해 주어야 한다. 자세한 내용은 4.6.1절을 참고하자.
- NewNode.sh를 사용하지 않기 때문에, 마스터의 /etc/dnsmasq.conf에 디스크의 mac주소와 해당되는 ip주소를 수작업으로 적어주자. 3.5절 참고.
- network sharing을 쉽게 할 수 있도록 3.3.4을 참고하여 /root/admin/ 디렉토리를 만들고, net-share-client.sh를 만들어주자.
- 4.5.1절에서 다룰 mdmonitor에서 warning을 줄 수 있도록 /root/service/ 디렉토리를 만들고, 다른 디스크 서버를 참고하여 DiskAlert.sh를 만들어주자.
- 클러스터가 해당 디스크에 접근할 수 있도록, 모든 노드들의 fstab에 새로운 디스크 노드를 추가해주어야 한다. 우선 마스터에서 /etc/fstab, /root/admin/fstab.template를 업데이트하고 각 CopyToNodes.sh를 사용해 각 노드들에 전달해 주자. 3.6절에서도 언급하였듯, hosts와는 달리 마스터의 fstab과 계산 노드의 fstab은 다르다. 절대로 마스터의

---

<sup>7</sup>내부망과의 인터넷 연결 속도가 중요하기 때문에, 가능하다면 업그레이드 할 수 있는 pcie 카드형태의 NIC를 internal으로 할당하자.

`fstab` 을 계산 노드의 `fstab` 으로 사용하면 안된다. OS가 존재하는 GPU노드에게는 **scp** 를 사용하여 복사해 주자.

- 앞으로 새로 들어올 노드들도 해당 디스크를 인식할 수 있도록 `NewNode.sh` 스크립트의 Disk server 섹션에도 추가해주자.
- NIC의 이름을 internal로 바꾸자. 자세한 내용은 3.3.1참고.

## GPU

GPU 서버 역시 디스크와 마찬가지로 os를 가져야 하기 때문에 추가과정 1,2,3을 해주어야 한다. 추가적으로 **pacman**을 통해 그래픽 드라이버 설치등을 진행해 주자.

### 4.4.2 노드 제거

기존의 노드가 고장이 날 경우에 수리하기 전까지는 **spg**의 machine 정보에서 노드를 꺼주고 수리한 후에 다시 켜주면 된다. 하지만 고장의 상태가 심각하여 수리하는 것이 어렵거나 노후되어 경제적인 이유로 수리하지 않을 경우에 komplex에서 노드를 제거해준다. 노드를 제거하는 방법은 추가했던 방법의 역순으로 진행하면 된다.

1. 4.10절을 참고하여 SPG에서 노드 제거
2. 노드 전원 종료 (전원 및 랜선 분리)
3. `DeleteNode.sh` 를 통해 마스터에서의 파일 시스템 및 설정 삭제

## 노드 설정 변경

흔하지는 않지만 NIC가 고장나서 교체할 수도 있을 것이다. 이 경우에 기존의 노드를 제거하고 새롭게 노드를 추가해도 되고 NIC와 관련된 정보들을 수정하는 방법도 있다. 기존의 노드는 제거하거나 추가하는 방법은 위를 참고하면 될 것 같고 NIC와 관련된 정보들을 다음과 같이 바꿔면 된다.

1. 마스터의 `/tftpboot/pxlinux.cfg` 디렉토리에 있는 `01-node-mac-addr` 의 파일의 이름을 새로 얻은 Mac 주소로 바꿔준다.
2. `/var/lib/misc/dnsmasq.leases` 에서 해당 노드의 이전 Mac 주소 또는 새로운 MAC 주소 가 적힌 줄은 모두 지워준다.
3. `/etc/dnsmasq.conf` 에서 해당 노드의 Mac 주소를 변경하고 **dnsmasq**를 재실행해준다.
4. `/root/admin/70-persistent-if-name-komplex.rule` 의 내용을 변경해 주자.

(Optional). 노드의 성능이 변경이 있을 경우에 SPG에서 노드의 정보를 갱신 (4.10절 참고)

## 4.5 디스크 관리

데이터를 저장하는데 사용하는 하드 디스크는 소모품으로서, 일반적으로 AFT<sup>8</sup> 혹은 MTBF<sup>9</sup>등으로 하드 디스크 제조사에서 그 수명을 안내한다. 특히 komplex 서버와 같이 여러명의 사용자가 24시간 디스크에 접근할 경우, 일반 pc의 경우보다 더 고장날 확률이 높을 것이 자명하다.

### 4.5.1 mdmonitor

3.4에서 사용한 **mdadm** 패키지는 만들어진 RAID 어레이의 상태를 확인하는 서비스 **mdmonitor**를 기본적으로 제공한다. 해당 서비스에서는 RAID 어레이의 상태를 확인하고, 이를 구성하는 하드 디스크의 고장등으로 인하여 어레이가 손상되었을 경우, `/etc/mdadm.conf`에 적혀져 있는 MAILADDR로 alert 메일을 보내거나 PROGRAM을 실행한다.

komplex에서는 다음과 같이 PROGRAM 변수를 사용한다.

```
DISK: /etc/mdadm.conf
```

```
...
```

```
PROGRAM /root/service/DiskAlert.sh
```

```
...
```

이를 통해 관리자에게 alert 메일을 주도록 되어 있다. 해당 메일을 받았다면, 현재 디스크 서버의 상태가 매우 위험한 상황이므로 반드시 4.5.4를 참고하여 디스크를 교체하여야 한다. 구체적인 내용은 4.9를 참고하자.

### 4.5.2 S.M.A.R.T

S.M.A.R.T(Self-Monitoring, Analysis, and Reporting Technology)는 4.5.1절에서 문제가 생겼을 경우 어떤 하드 디스크가 문제가 있는지 검사해 주는데 사용된다. 우선 **smartmontools**를 설치하자. 일반적으로 `mdmonitor --detail` 혹은 `cat /proc/mdstat`을 통해 어떤 디스크가 잘못되었는지 확인할 수 있지만, 해당 방법이 여의치 않다면 써볼 수 있다.

만약 4.5.1에서 문제가 된 어레이가 `/dev/sdb2`, `/dev/sdc2`, `/dev/sdd2`로 이루어 졌다면 다음과 같은 명령어로 각 하드 디스크를 검사할 수 있다.

```
# smartctl -t short /dev/sdb
# smartctl -t short /dev/sdc
# smartctl -t short /dev/sdd
```

이때 주의할 점은 파티션이 포함된 device가 아닌 하드 디스크 전체를 확인해야 한다는 점이다. 위의 명령어는 short 테스트로 약 5분 안에 검사가 끝난다. 검사 결과는 다음과 같이 볼 수 있다.

<sup>8</sup>Annualized Failure Rate. 1년동안 작동했을 때 고장이 날 확률

<sup>9</sup>Mean Time Between FAilure. 평균 무고장 시간

```
# smartctl -l selftest /dev/sdb
```

만약 모든 하드 디스크들이 테스트를 통과하였다면, `-t short` 대신 `-t long` 옵션을 주어 검사를 해 볼수도 있을 것이다. 하지만 이는 수시간이 넘게 걸릴 수 있다.

#### 4.5.3 Scrubbing

Scrubbing은 **mdadm**에서 제공해 주는 기능으로, RAID 어레이의 상태를 확인하고 가능할 경우 복구까지 진행해 준다.<sup>10</sup>

RAID 어레이의 이름이 `/dev/mdX` 라면, 다음과 같은 명령어로 scrubbing을 진행할 수 있다.

```
echo check > /sys/block/mdX/md/sync_action
```

명령어를 확인하면 단순히 `/sys/block/mdX/md/sync_action` 파일에 `check`라는 글을 써놓는 것과 같다. Scrubbing되는 상황은 `/proc/mdstat`에 기록되며, 디스크를 정상적으로 사용할 수 있으나.

해당 작업은 수시간 이상 걸리기 때문에, 중간에 검사를 중단하여야 할 경우 다음과 같이 할 수 있다.

```
echo idle > /sys/block/mdX/md/sync_action
```

즉, `check` 명령어 대신, `idle` 을 redirect하면 된다. Scrubbing이 성공적으로 끝났을 경우 `/sys/block/mdX/md/sync_action`의 파일 내용을 확인해 보면 역시 `idle`이라고 써있는 것을 확인할 수 있을 것이다.

Scrubbing이 진행될 때 디스크의 사용량이 매우 높아진다. 이로 인해 데이터의 입출력이 상당히 늦어질 수 있는데, komplex의 특성상 24시간 디스크가 사용되며 데이터를 저장하는 경우가 매우 많기 때문에 가능하면 작업이 없을 때 scrubbing을 진행해야 할 것이다. 이때문에 서버를 내리기 약 12시간 전에 작업들을 모두 종료하고 scrubbing을 진행한 다음 서버를 끄는 식으로 진행한다. 서버 종료에 대한 자세한 방법은 4.1를 확인하자.

Scrubbing을 하면서 문제가 되는 섹터를 발견하면, `/sys/block/mdX/md/mismatch_cnt` 파일에 그 숫자를 적어둔다. RAID5 혹은 RAID6에서는 mismatch count가 존재한다면 이는 대부분 하드웨어적인 문제인 것을 판단한다. 이때문에 mismatch count가 증가한다면, 이를 하드 디스크를 바꿔주어야 할 시기가 다가오고 있다고 이해하면 된다.

#### 4.5.4 Remove and Recover RAID array

**mdmonitor**에서 문제가 있는 하드 디스크 `/dev/sdAY` 가 발견되었다면, 이를 RAID에서 제외하고 새로운 디스크를 추가해야 할 것이다. 우선 다음과 같은 명령어로 레이드에서 디스크를 제외하자.

<sup>10</sup>[https://wiki.archlinux.org/title/RAID#RAID\\_Maintenance](https://wiki.archlinux.org/title/RAID#RAID_Maintenance)

```
mdadm --fail /dev/mdX /dev/sdAY  
mdadm --remove /dev/mdX /dev/sdAY
```

위의 명령어 이후 `lsblk`를 통해 마운트 된 장치를 확인해보면, `/dev/sdAY`는 `/dev/mdX`를 구성하고 있지 않을 것이다. 만약 예기치 못한 문제 (SATA선의 문제등)로 위의 작업을 하지 못한 채 하드 디스크의 연결이 끊어졌다면 위의 작업을 건너뛰게 될 것이다.

이후 새로운 하드 디스크를 추가하자. 이미 RAID를 구성하고 있는 하드 디스크들과 같은 용량을 가지도록 partition을 `/dev/sdBZ`로 나누어야 한다. 만약 새로 추가하는 디스크가 이전에 다른 RAID를 구성한 적이 있다면, super block을 삭제해야 한다.

```
mdadm --zero-superblock /dev/sdBZ
```

만약 새로운 하드 디스크를 사용한다면 위의 작업을 해주지 않아도 될 것이다. 그리고, 해당 디스크를 복구해야 하는 array에 추가하자.

```
mdadm --add /dev/mdX /dev/sdBZ
```

위의 작업 역시 scrubbing과 같이 수시간이 소요될 수 있다.

## 4.6 Security

외부로 연결이 되어 있는 마스터와 터미널과 마스터의 보안을 신경써주어야 한다. 인터넷을 돌 아다니면서 서버를 공격하는 봇(bot)들이 수시로 접속을 시도하기 때문이다. 서버 내부망은 USB와 같이 직접 연결이 되지 않는 이상 외부자가 접속할 방법이 전혀 없으므로 걱정이 없다. (물론 물리적으로 접속이 허용되면 바로 보안이 풀렸다고 보면 된다.) 아치리눅스 위키의 Security 페이지<sup>11</sup>를 참고하면 도움이 될 내용들이 정리되어 있다.

### 4.6.1 OpenSSH

komplex에서는 일반적으로 직접 컴퓨터에 접속하는 것이 아니라, 연구실 혹은 개인 pc로 `ssh`를 통해 원격 접속하고 있다. 이러한 접속은 `openssh` 패키지를 사용하기 때문에 해당 패키지의 버전을 최신으로 유지하는것이 매우 중요하다. `openssh`는 `ssh`접속을 받는 host와 `ssh`접속을 하는 client로서의 기능을 모두 제공한다.

#### Host

외부로부터의 접속을 관리하는데는 `sshd`가 사용된다. 해당 데몬은 `/etc/ssh/sshd_config`로 설정할 수 있다. 다양한 설정들이 있는데, 현재 komplex에서 설정된 내용만 살펴보자.

- ListenAddress: 해당 항목에 쓰여진 ip로 들어오는 경우만 접속을 허용한다.

<sup>11</sup><https://wiki.archlinux.org/title/Security>

- master: 외부망에서의 접근을 차단하는 것을 원칙으로 하기 때문에, 내부 ip 주소 10.0.0.1 이 설정되었다.
- terminal: 외부망에서는 보안을 위해 기본 포트 22에서 변경된 56번 포트로 접근할 수 있게 한다. 따라서 *external-ip*:56 으로 설정되어 있다. 내부망은 22번을 그대로 사용하기 때문에 *internal-ip*:22로 되어 있다.
- node: 외부에서의 설정이 되지 않은 default값으로, 어떤 ip로 22번 포트 접속이 가능하도록 설정되어 있다. 실제로는 내부망과만 연결되어 있기 때문에 보안에 무관하다.
- PermitRootLogin: root 계정으로의 원격접속의 허용여부.
  - master: ListenAddress 항목에서 이미 내부망으로만 접속을 허용했기 때문에, root 계정으로의 로그인을 허용하도록 되어 있다.
  - terminal/node: 기본적으로 원격 접속을 차단한다. 하지만 후술될 Match 항목에서 내부 ip주소를 통해 들어오는 경우 특별히 접속을 허용하는 정책을 사용한다.
- Match (condition): condition을 만족하는 경우 아래의 설정을 적용한다.
  - terminal/node: 접속해오는 곳의 ip 주소가 내부망(10.0.\*.\* ) 이라면, root 계정으로의 접속을 허용한다.

만약 `/etc/ssh/sshd_config` 의 내용을 수정하였을 경우, 이를 반영하기 위해 `sshd`를 재시작 해주어야 한다.

```
# systemctl restart sshd
```

또한, 계산 노드에 적용되는 설정일 경우, `/root/admin/sshd_config.template` 을 수정해 주고 4.4의 경우와 같이 `CopyToNodes.sh` 및 `scp`을 사용해 전체 서버를 업데이트 및 `sshd`를 재시작 해주어야 한다.

## Client

`ssh`를 통한 접속에는 두 가지 종류의 인증 방식을 사용할 수 있다. 첫번째는 password를 통한 접속으로, 별다른 설정을 해주지 않고 이미 계정에 설정된 password를 쳐서 들어가는 방식이다.

두번째로는 SSH key<sup>12</sup>로 인증을 하는 방식이다. 한번 key를 만들고 이를 host쪽에 등록하여 비밀 번호 없이 접근할 수 있다. 이는 komplex 내부망에 이미 구현되어 있는데 터미널에서 다른 노드들로 접속할 때 굳이 비밀번호를 묻지 않는 것을 확인할 수 있다. 구현 방법 역시 6.3의 `NewNode.sh` 와 `NewUser.sh` 스크립트에서 확인할 수 있다.

---

<sup>12</sup>[https://wiki.archlinux.org/title/SSH\\_keys](https://wiki.archlinux.org/title/SSH_keys)

## 4.6.2 PAM

komplex에서는 사용자의 로그인 및 비밀번호 정책들을 PAM(Pluggable Authentication Modules)<sup>13</sup>을 통해 설정한다. PAM은 모듈 형식으로 구성되어 있으며, 모듈이라 불리는 최소 기능 단위는 `/usr/lib/security/` 디렉토리에 `.so`의 형식으로 존재한다. 각 모듈의 기본 configuration은 `/etc/security`에 `.conf`의 형식으로 존재한다. 예를 들어, `pam_faillock.so`의 기본 설정파일은 `faillock.conf`이고 `pam_pwquality.so`의 기본 설정파일은 `pwquality.conf`이다.

PAM의 작동은 특정 상황이 되었을 때 `/etc/pam.d`에 있는 정책 지침서대로 따라간다. 예를 들어 원격에서 `ssh`로 로그인을 시도할 경우, 우선 `/etc/pam.d/sshd`에 있는 지침대로 PAM이 작동한다.

```
/etc/pam.d/sshd
```

```
auth include system-remote-login
account include system-remote-login
password include system-remote-login
session include system-remote-login
```

첫번째 열의 type은 인증 형식, 두번째 열의 control은 PAM의 행동을 지칭한다. 마지막으로 세번째 열은 행동의 대상을 말해주고, 경우에 따라 옵션을 설정해 준다. 이때 설정된 옵션들은 기본 설정 파일의 옵션보다 우선시된다. 이를 기반으로 `/etc/pam.d/sshd`를 해석해 보면, 어떤 형식으로 인증 방식이 들어오던지 `system-remote-login`의 정책대로 수행하라 라는 것이다. 이와 같이 각 type과 정책들을 stacking 해 나갈 수 있으며, 로그인과 관련된 가장 상위 단계 정책으로 `system-auth`가 존재한다. 따라서 해당 파일을 수정한다면 대부분의 로그인 관련 정책들이 수정된다.

### Login restriction

로그인과 관련된 PAM은 `ssh` 혹은 `su`로 로그인을 하는 경우에 관련되어 stack된 정책들이 실행되며 상술했듯 `/etc/pam.d/system-auth`가 최상위 정책이다. 알 수 없는 이유로 특정 계정의 비밀번호가 단시간내에 과도하게 틀릴 경우, 그에 대한 조치가 필요할 것이다. komplex에서는 `pam_faillock.so` 모듈을 사용한다. 설정파일인 `faillock.conf`에서 구체적으로 그 행동을 정의하는데, 1시간 이내에 6번 틀릴 경우, 24시간동안 접속을 제한하는 것을 원칙으로 한다.

실 사용자의 실수로 위와 같은 문제가 발생하여 접속이 제한될 경우, 관리자는 다음과 같이 접속 제한을 해제할 수 있다.

```
# faillock --reset --user username
```

### Password

일반 사용자의 계정이라도 komplex에서 기본적으로 권한이 많이 열려있기 때문에 비밀번호를 간단하게 만들지 않아야 한다. 사용자들의 비밀번호들은 `/etc/shadow`에 암호화되어 저장되어 있

<sup>13</sup><https://wiki.archlinux.org/title/PAM>

다. 특히 루트 계정의 비밀번호도 이 파일에 있으므로 보안에서 매우 중요한 파일이다. 암호화되어서 육안으로 바로 비밀번호를 알 수 없지만 ‘John the ripper’와 같은 프로그램을 통해서 암호를 해독할 수 있으므로 주의를 요한다. 이 때문에 만약 외부인에게 서버 접속을 허용했다면 보안이 뚫렸다고 봄야 한다.<sup>14</sup> 이에 관련하여 공용계정이나 임시계정을 만들지 않도록 한다.

비밀번호화 관련된 PAM은 **passwd**로 비밀번호를 수정하는 경우에 **/etc/pam.d/passwd** 정책이 실행된다. komplex에서는 사용자 계정의 비밀번호 복잡도를 유지하기 위해 **pam\_pwquality.so** 모듈을 사용한다. 로그인 제한과 마찬가지로, 기본 설정 파일인 **pwquality.conf**에 그 설정이 있으며 구체적으로는 12자 이상의 영문과 숫자, 특수문자를 섞게 되어 있다.

```
/etc/pam.d/passwd  
#%PAM-1.0  
password required pam_pwquality.so  
password required pam_unix.so sha512 shadow use_authtok
```

#### 4.6.3 Firewall

마스터나 터미널에서 **journalctl**<sup>15</sup>를 켜보면 아래와 같이 알 수 없는 곳에서 수시로 **ssh** 접속을 시도하는 것을 볼 수 있다.

```
master sshd[13967]: Failed password for root from 221.203.142.133 port 21777 ssh2  
...  
master sshd[29749]: Failed password for invalid user admin from 183.52.118.252 port 48540 ssh2
```

komplex에서는 **nftables** 패키지를 이용한 방화벽을 통해 이를 차단하는데 **nft**에 규칙들을 추가해서 특정한 IP만 접속을 허용해주거나 막는 방식으로 작동한다.<sup>16</sup><sup>17</sup> white list나 black list를 통해서 방화벽이 작동한다고 보면 이해가 쉬울 것이다. white list를 이용하는게 보완의 측면에서 더 안정적일 수 있지만 여려모로 불편하기 때문에 komplex에서는 black list를 이용한다. 예컨데 white list를 이용하는 경우에 해외 학회 중에 komplex에 접속하기 위해서는 국내에 허용된 PC를 우회해서 서버에 접속해야하는 불편함이 있다.

**nft**를 사용하는 방법은 아치리눅스의 nftables 페이지<sup>18</sup>를 참고하자. 방화벽은 family, table, chain, rule의 순서로 계층이 있으며 komplex에서는 ip family(ipv4)만을 사용한다.

예를들어 특정 *ip-addr*의 접근을 막고 싶다면, ip family에 있는 filter table에 FORWARD chain에 새로운 rule을 추가해 주면 된다.

<sup>14</sup>이 경우 서버를 다시 구축해야 한다.

<sup>15</sup>systemd로 작동되는 서비스의 log를 기록해준다. <https://wiki.archlinux.org/title/Systemd/Journal>

<sup>16</sup>인터넷에는 **/etc/hosts.allow** 와 **/etc/hosts.deny**를 사용한 TCP wrappers 방법이 많이 소개되고 있지만, 해당 서비스는 1997년 이후 업데이트 되지 않아 결국 2011년 arch linux에서 deprecate된 방법으로 사용하지 말아야 한다.

[https://archlinux.org/news/dropping-tcp\\_wrappers-support/](https://archlinux.org/news/dropping-tcp_wrappers-support/)

<sup>17</sup>이전에는 **iptables**를 사용했지만今は legacy framework가 되어 **nftables**를 사용하게 되었다.

<sup>18</sup><https://wiki.archlinux.org/title/Nftables>

```
# nft add rule ip filter INPUT ip saddr ip-addr drop
```

nftables의 규칙을 잘못 건드리면 네트워크 전체가 막통이 될 수도 있으니 사용할 때 신중을 기해야 한다. komplex에서는 방화벽을 체계적으로 관리하기 위해 마스터와 터미널을 위해 4.9.3, 4.9.4에서와 같이 서비스를 만들어서 사용하고 있다.

각 방화벽들은 `/root/service/Firewall-Blacklist.txt`를 기반으로 차단할 IP들을 설정한다. 이를 위해 해당 파일을 일주일에 한번씩 업데이트 해주는 데몬 **Firewall-Blacklist-update**를 운영중이다. 자세한 내용은 4.9.2 참고.

각각의 서비스가 잘 작동되고 있는지 확인하기 위해 마스터에서 다음을 확인해 주자.

```
# systemctl status Firewall  
# systemctl status Firewall-Blacklist-update
```

터미널에서는 blacklist를 마스터와 공유하기 때문에, 방화벽 서비스만 확인해 주면 된다.

```
# systemctl status Firewall-terminal
```

## 4.7 Server Room Temperature

컴퓨터는 발열이 심한 장비인데 밀폐된 서버실에 수 백대의 PC가 가동되므로 온도가 쉽게 높아진다. 서버실 온도가 높아지면 장비의 고장뿐만 아니라 화재의 위험도 있으므로 서버실 온도를 유지하기 위한 냉방장치가 필요로 한다. 현재 서버실에 4대의 에어컨이 가동 중이며 10분에 한번씩 서버실 온도를 측정하게 된다. 특정 온도(30도)가 넘어간다면 관리자에게 경고 메일을 보내주는 서비스가 가동중이다. 여기서는 온도계 사용법을 다루고 온도 모니터링 및 메일링과 관련된 내용은 4.9절에서 확인하도록 하자.

### Thermometer

komplex의 온도계는 두종류가 있으며, 한가지는 Temperature@lert사의 모델로, 10.0.0.254로 내부망과 연결되어 있다. 온도계를 다음과 같이 이용할 수 있다.

- 현재온도 확인:

```
$ curl http://10.0.0.254/xmlfeed.rb 라고 실행하면, 예를 들어,  
<currentReading>25.5</currentReading> 와 같이 뜨는데 그 사이의 '25.5'가 현재 온도  
(섭씨)이다.
```

- 시간 별로 기록된 온도 확인: `$ curl http://10.0.0.254/logfile.rb`

두번째로는 아두이노를 사용한 온도센서가 있다. 현재 tenet98, tenet170, tenet182, tenet211에 각각 usb를 통해 연결되어 있다. 온도값은 `/root/service/arduino-temperature.py`을 실행하여 얻을 수 있다. 이를 실행하기 위해서는 `pyserial` 패키지가 필요하다. 이후 4.10절에서 자세하게 다룰 것 이지만, 다음과 같이 해당 패키지를 설치하여야 한다.

```
pip install pyserial
```

각각의 온도계의 위치는 ??절의 서버실 지도를 참고하자.

## 4.8 Bash Scripts

komplex의 `/root/admin` 디렉토리에는 komplex 운영의 편의를 도모하기 위해 연구실에서 손수 만든 bash script들이 모여 있다. 이 절에서는 해당 스크립트들의 사용법을 다룰 것이며, 구체적인 스크립트들이 필요할 경우, 6.3절에서 확인할 수 있다.

이 절에서 다루는 모든 script들은 반드시 마스터에서 root 계정으로 접속하여 `/root/admin` 디렉토리에서 실행하여야 한다.

### Common

`Common.sh` 는 직접적으로 실행하는 script가 아니다. 다른 script들이 실행될 때, 현재 마스터에서, root 계정으로 실행하는지를 확인한다. 또한, tftpboot에 대한 기본적인 변수들을 설정해 준다.

#### 4.8.1 CopyToNodes

`CopyToNodes.sh` 는 마스터의 파일을 여러개의 노드들에 한번에 복사하기 위해서 만든 스크립트이다. 후술될 `NewNode.sh`에서 PXE, NFS, NTP 등의 설정 파일들을 마스터에서 노드의 파일 시스템으로 복사하는 용도 등으로 사용된다. 이외에도 4.4절에서 언급했듯이, 수정된 `/etc/hosts` 등의 설정 파일을 노드들에게 업데이트 해줄 수 있다.

해당 script는 총 5개의 argument를 가진다.

1. 복사할 파일의 마스터에서의 절대 경로. 반드시 절대경로이어야 한다.
2. 복사할 파일이 위치할 노드에서의 절대 경로. 반드시 절대경로이어야 한다.
3. 대상이 되는 노드의 prefix. 예를들면 terminal, disk 등이 있을 것이다.
4. 대상이 되는 노드들의 시작 번호. 시작 번호까지 포함된다.
5. 대상이 되는 노드들이 끝 번호. 끝 번호까지 포함된다.

예를들어, 마스터에서 `/root/admin/hosts.template`에 위치한 파일을 `terminal2`에서부터 `terminal3` 까지 터미널 기준 `/etc/hosts`에 복사하고 싶다면, 다음과 같이 사용하면 된다. 시작 번호와 끝 번호 사이에 비어있는 노드가 있을 경우, 이는 무시된다.

```
[root@master]# ./CopyToNodes.sh /root/admin/hosts.template /etc/hosts terminal 2 3
```

### 4.8.2 NewNode

`NewNode.sh` 는 komplex에 새 노드를 추가하기 위한 배수 스크립트이다. 이는 os가 없는 terminal 혹은 CPU 계산노드에 사용된다. 이때, `/root/admin`에 있는 6개의 template들을 사용해 설정 파일들을 복사하게 되어있다.

- `hosts.template` : 3.5절
- `fstab.template` : 3.6절
- `nfs-server.conf.template` : 3.6절
- `pxelinuxcfg.template` : 3.7절
- `ntp.conf.template` : 3.8절
- `sshd_config.template` : 4.6.1절

각 항목들의 설명 위치를 참고하여 위의 template들을 반드시 up to date로 유지하여야 한다.

또한, 마스터에서 설정되어 있는 프로그램 및 서비스들 대부분이 그대로 옮겨지기 때문에, 마스터에서 작동하는 서비스들이 완전히 설치되지 않은 상태에서 `NewNode.sh`를 통해 새로운 노드를 만든다면, 해당 노드를 삭제하고 다시 만들어야 하는 불상사가 있을수도 있으니 주의하여야 한다.

해당 script는 총 3개의 argument를 가진다.

1. 새롭게 만들어질 노드의 호스트 이름
2. 새롭게 만들어질 노드의 mac 주소
3. 새롭게 만들어질 노드의 내부 ip 주소

예를 들어, 00:11:22:33:44:55의 맥 주소를 komplex100이라는 이름으로 새로이 생성하고자 한다. 이때 내부 ip주소는 convention에 맞춰 10.0.2.100이 될 것이다.

```
[root@master]# ./NewNode.sh komplex100 00:11:22:33:44:55 10.0.2.100
```

위와 같이 실행한다면, 입력을 확인하는 문구가 나오고, 새로운 노드를 만들기 시작한다. 마지막 출력중에 `You should restart the dnsmasq daemon`에 따라 다음의 작업을 추가적으로 해주어야 한다.

```
[root@master]# systemctl restart dnsmasq
```

`You should make initrd`에 따라 램 디스크를 만들어야 한다.

```
[root@master]# mkinitcpio -p komplex
```

`You should update machine info on /root/spg`에 따라, 4.10절을 참고하여 계산 노드의 정보를 SPG에 등록하자.

#### 4.8.3 DeleteNode

`DeleteNode.sh` 는 komplex에서 노드를 제거해 준다. 노드의 고장 등으로 서버에서 제외하여야 할 때 사용될 것이다. 해당 script는 1개의 argument를 가진다.

1. 삭제될 노드의 호스트 이름

예를 들어 komplex100 노드가 문제가 있어 클러스터에서 제거해야 할 경우 다음과 같다.

```
[root@master]# ./DeleteNode komplex100
```

이를 실행하면, 확인 문구가 나오며 해당 노드와 관련된 tftpboot 디렉토리를 모두 삭제한다. 또한 마스터의 `/etc/dnsmasq.conf` 를 업데이트 해준다. 백업 용도의 `/etc/dnsmasq.conf.bak` 에 이전 버전을 저장해 둔다.

#### 4.8.4 NewUser & DeleteUser

`NewUser.sh` 는 마스터에 새로운 사용자 계정을 추가해 준다. 해당 script는 3개의 argument를 가진다.

1. 새로운 계정의 이름 *newbi*
2. 새로운 계정의 홈디렉토리가 위치할 디스크 `/pds/pds21`. 실제 홈 디렉토리는 `/pds/pds21/newbi` 가 된다.
3. 새로운 계정의 코멘트 *comment*. 사용자의 실명(사용자의 이메일) 형식으로 사용한다. 반드시 “”(double quotation)으로 묶어서 입력하여야 한다. 이는, 미래의 관리자가 계정 사용자를 직접 모를 때, 서버 관련 공지를 하기 위한 연락처를 저장하기 위한 용도로 사용한다.

예를 들어 다음과 같이 다음과 같이 사용하면 된다.

```
[root@master]# ./NewUser.sh newbi /pds/pds21 comment
```

스크립트가 실행되는 도중에 비밀번호를 설정해 주도록 되어 있다. 4.6.2에서 서술된 대로, 12 자리 이상의 영문, 숫자, 특수문자가 섞인 비밀번호이어야 한다. 그리고 **ssh-keygen** 때문에 세번의 엔터를 쳐 주어야 한다.

`DeleteUser.sh` 는 마스터에 등록되어 있는 사용자 계정을 삭제해 준다. 해당 script는 1개의 argument를 가진다.

1. 지울 계정의 이름 *newbie*

예를 들어 다음과 같이 사용하면 된다.

```
[root@master]# ./DeleteUser.sh newbie
```

위 스크립트에서는 `users` 그룹에서 `newbie` 계정을 제거하고, 계정을 삭제하는 작업을 한다. 다만 해당 유저의 홈 디렉토리 등 모든 디렉토리들은 남아있기 때문에, 상황에 따라 이러한 홈 디렉토리를 수작업으로 삭제해 주어야 한다.

#### 4.8.5 UpdateUserInfoOn...

`UpdateUserInfoOnNodes.sh` 와 `UpdateUserInfoOnOSServers.sh` 는 마스터에 있는 계정 정보를 os가 없는 계산 노드들과 os가 있는 노드들에 업데이트 해준다. 4.3.1의 설명처럼 두 가지는 구분되어야 하며, 다음과 같이 3개의 argument를 가진다.

1. 업데이트 할 노드의 이름 prefix
2. 대상이 되는 노드들의 시작 번호. 시작 번호까지 포함된다.
3. 대상이 되는 노드들의 끝 번호. 끝 번호까지 포함된다.

새로운 유저가 등록될 때마다, 서버에 있는 모든 노드들에 이를 적용해 주어야 한다. 예를 들면 다음과 같다.

```
[root@master]# ./UpdateUserInfoOnNodes.sh terminal 2 3
```

```
[root@master]# ./UpdateUserInfoOnOSServers.sh disk 1 3
```

두 가지 스크립트 모두 빠져있는 번호는 무시하고 넘어갈 수 있다. 이 과정에서 `/tftpboot` 의 호스트 이름들과 `/etc/dnsmasq.conf` 를 사용한다. 중간에 문제가 생긴다면 해당 내용들을 다시 확인해 보자.

#### 4.8.6 UpdateSystemFilesOnNodes

`UpdateSystemFilesOnNodes.sh` 는 마스터에서 설정 파일이 바뀐 경우에 이를 노드에 적용해준다. 특히 4.2절에서와 같이 마스터에 리눅스를 업데이트 했거나 마스터에서 새로운 패키지를 설치한 후에 이를 모든 노드에 적용하기 위해서 사용된다.

이는 OS가 존재하는 디스크 및 GPU서버에는 적용할 수 없다. 이들을 업데이트 하기 위해서는 3.3.4절을 따라 인터넷에 연결한 후 각자 업데이트 해주어야 한다.

해당 script는 `UpdateUserInfoOnNodes.sh` 와 같이 3개의 argument를 가진다.

1. 업데이트 할 노드의 이름 prefix
2. 대상이 되는 노드들의 시작 번호. 시작 번호까지 포함된다.
3. 대상이 되는 노드들의 끝 번호. 끝 번호까지 포함된다.

예를 들면 다음과 같다.

```
[root@master]# ./UpdateSystemFilesOnNodes.sh terminal 2 3
```

역시 빠져있는 번호는 무시하고 넘어갈 수 있다.

여기서 `NodewiseETC.txt`, `NodewiseVARLIB.txt`에 등록되어 있는 경로들은 제외한다. 이는 마스터와 각 노드들의 설정이 달라야 하는 파일들이다. 대표적으로 `/etc/ssh/sshd_conf`의 경우 4.6.1절의 내용처럼 마스터와 터미널, 계산 노드의 설정이 모두 다르다.

#### 4.8.7 net-share-host & net-share-client

`/root/admin/net-share-host.sh`는 외부 인터넷과 연결되어 있는 terminal에서 사용해야 한다. 다른 스크립트들은 모두 마스터에서 사용하는 것과 다르기 때문에, 주의하자. 해당 스크립트에서는 ip 주소의 서브넷 마스크를 추가해 주고, 현재 `net.ipv4.ip_forward`의 상태값을 확인하고, 그 값을 바꾸어 주는 역할을 한다. argument는 없다.

```
[root@terminal]# ./net-share-host.shh
```

`/root/admin/net-share-client.sh`는 한개의 argument를 받는다. 역시 마스터가 아닌, internet share를 받는 client에서 실행하여야 한다.

1. internet share를 해줄 host의 내부 ip 주소

예를들어 10.0.0.2의 내부 ip주소를 가지는 terminal2로부터 internet share를 받기 위해서는 다음과 같이 하면 된다.

```
[root@client]# ./net-share-client.sh 10.0.0.2
```

이 스크립트에서는 `net-share-host` 와 마찬가지로 ip 주소의 서브넷 마스크를 추가해 주며, host의 ip 주소로 default route를 바꾸어 준다.

### 4.9 Services

komplex의 `/etc/systemd/system` 과 `/root/service`에는 komplex를 위해 손수 만든 서비스들이 존재한다. 서비스는 부팅 시에 자동으로 script를 작동시킬 수 있기 때문에 그 사용법을 익혀두기를 권한다.

서비스를 운영하기 위해서는 적어도 두개의 파일이 필요하다. 첫번째는, `systemctl`로 서비스를 관리하기 위해 `/etc/systemd/system`에 `.service`의 확장자를 가지는 유닛이 있어야 한다. 해당 경로는 일반적으로 사용자가 직접 만든 서비스가 존재하는 곳이며, `/lib/systemd/system`에는 리눅스 시스템과 패키지에서 사용되는 서비스가 존재한다.

두번째로는 해당 유닛이 작동시킬 실제적인 명령어가 담긴 실행 파일이 있어야 한다. komplex에서는 해당 실행파일들을 `/root/service`에 모아두고 있다.

#### 4.9.1 ExternalNetwork-terminal

**ExternalNetwork-terminal**은 터미널 노드에게 고정 ip를 할당해 주어 외부 네트워크와 연결될 수 있게 해주는 서비스이다. 이는 마스터가 아닌 터미널에서 작동해야 하는 서비스이기 때문에, 각 터미널들의 `/etc/systemd/system/` 경로에 **ExternalNetwork-terminal.service**를 위치시켜주어야 한다.

`/etc/systemd/` 경로와 다르게 터미널과 마스터는 `/root` 를 NFS를 통해 공유하기 때문에 실제 실행 파일인 `ExternalNetwork-terminal.sh` 의 경우 `/root/service`에 위치시켜주면 된다.

즉, terminal2에게 서비스를 제공하기 위해서 master 기준 다음의 경로에 두개의 파일이 있는지 확인하자.

```
/tftpboot/terminal2/etc/systemd/system/ExternalNetwork-terminal.service  
/root/service/ExternalNetwork-terminal.sh
```

이를 확인하였다면, 터미널 노드에서 서비스를 start/enable 해주면 터미널의 고정 ip 설정이 완료된다.

```
# systemctl start ExternalNetwork-terminal  
# systemctl enable ExternalNetwork-terminal
```

터미널이 추가되었을 경우, 6.4절을 참고하여 실행 파일인 `ExternalNetwork-terminal.sh` 을 수정하자.

#### 4.9.2 Firewall-Blacklist-update

4.6.3에서도 명시하였듯 komplex 서버에서는 알 수 없는 곳으로부터 ip주소를 통해 무차별적인 접근을 시도한다. 해당 기록은 모두 `journalctl`에 남아 있으며, 서버의 부팅 이후 100번 이상 로그인에 실패한 ip들을 `/root/service/Firewall-Blacklist.txt`에 저장해두어 접근을 차단하도록 해야 한다.

이와 같은 작업은 마스터의 **Firewall-Blacklist-update.service** 유닛을 통해 이루어지며, 이는 **Firewall-Blacklist-update.timer** 유닛으로 트리거 되어 일주일에 한번씩 규칙적으로 진행된다. master에서 다음의 경로에 세개의 파일이 있는지 확인하자.

```
/etc/systemd/system/Firewall-Blacklist-update.service  
/etc/systemd/system/Firewall-Blacklist-update.timer  
/root/service/Firewall-Blacklist-update.sh
```

확인을 마쳤다면 `.timer` 유닛을 start 및 enable 해주자. `.service` 유닛의 경우 `.timer`로 trigger되기 때문에 따로 작업해 줄 필요 없다.

```
# systemctl start Firewall-Blacklist-update.timer  
# systemctl enable Firewall-Blacklist-update.timer
```

위의 작업은 기계적으로 진행되기 때문에 주의를 요한다. 특히, 루트 계정의 비밀번호를 잘못 쳐서 접속을 내부망이나 연구실에서 접속을 실패하는 경우가 종종 있는데 이 때도 **journalctl**에 IP가 남는데, 이러한 IP들은 black list에 들어가지 않도록 고려해주어야 한다. 이와 같은 이유로 black list를 직접 수정할 경우, 후술될 **Firewall**과 **Firewall-terminal**을 각각 마스터와 터미널에서 restart 해주어야 한다.

#### 4.9.3 Firewall

**Firewall**은 마스터의 방화벽 서비스이다. 기본적으로 내부망에서의 모든 접속을 허용하고, 외부로부터의 허용은 모두 막아두었다. 추가적으로, `/root/service/Firewall-Blacklist.txt`에 저장되어 있는 ip들을 읽어서 **nft**로 차단하도록 되어 있다.

다음의 경로에 두개의 파일이 있는지 확인하고, 서비스를 start/enable 해주자.

```
/etc/systemd/system/Firewall.service  
/root/service/Firewall.sh
```

#### 4.9.4 Firewall-terminal

**Firewall-terminal**은 터미널의 방화벽 서비스이다. 이때문에, 상술된 **ExternalNetwork-terminal**과 같이 터미널의 `/etc/systemd/system`에 유닛을 넣어두어야 하며 실행파일은 일반적인 경로인 `/root/service`에 위치한다. **Firewall-terminal**은 **nft**를 위한 설정파일을 따로 저장해둔다. 또한 black list의 경우 **Firewall**과 공유한다.

terminal2에 서비스를 제공하기 위해 master 기준으로 다음의 경로에 네개의 파일이 있는것을 확인하고 terminal2에서 서비스를 start/enable하자.

```
/tftpboot/terminal2/etc/systemd/system/Firewall-terminal.service  
/root/service/Firewall-terminal.sh  
/root/service/Firewall-terminal.start.nft  
/root/service/Firewall-terminal.stop.nft
```

#### 4.9.5 Admin Report

**AdminReport**는 매일 서버실의 온도와 현재 서버의 가동 상황을 정리하여 메일로 보내주는 서비스이다. 메일을 받는 사람의 리스트는 `/root/service/AdminReport.sh`에서 수정할 수 있다.

메일의 형식은 `/root/service/AdminReport.mail`을 따르며, 가장 최근에 메일을 보낸 시각과 에러가 발생한다면 그 메세지를 `/var/log/AdminReport.log`에 저장한다. 이를 수정할 필요가 있다면 6.4의 해당 항목을 참고하자.

해당 서비스는 `/etc/systemd/system/AdminReport.timer` 유닛에 의해 매일 오전 9시에 trigger된다. 4.9.2의 경우처럼 총 4개의 파일이 있는지 확인하고 `.timer` 유닛을 start/enable 시켜주자.

```
/etc/systemd/system/AdminReport.timer  
/etc/systemd/system/AdminReport.service  
/root/service/AdminReport.sh  
/root/service/mailTemplate/AdminReport.mail
```

#### 4.9.6 Temperature Monitor

**TemperatureMonitor**는 10분마다 서버실의 온도를 확인하고, 해당 온도가 설정 온도 이상일 때 등록된 그룹원들 전원에게 alert 메일을 보내주는 서비스이다. 해당 메일은 온도가 낮아질 때 까지 30분 간격으로 계속 보내진다. 가능하다면 그룹원들 전원의 주소를 포함하도록 하자. 또한 서버실에 조치가 취해져 다시 온도가 낮아졌을 때도 cancel 메일을 보내준다. 메일을 받는 사람의 리스트는 `/root/service/TemperatureMonitor.sh`에서 수정할 수 있다.

alert와 cancel메일의 형식은 `/root/service/TemperatureMonitorMailTemplate/` 디렉토리에서 확인할 수 있다. 또한 1분마다 측정된 서버실의 온도를 `/var/log/TemperatureMonitor.log`에 로깅을 하는 역할도 수행한다. 자세한 script와 template들은 6.4절을 확인하자.

다른 서비스들과 같이 다음의 두개의 파일이 정확한 위치에 있는지 확인하고 서비스를 start/enable 해주자

```
/etc/systemd/system/TemperatureMonitor.service  
/root/service/TemperatureMonitor.sh  
/root/service/mailTemplate/TemperatureAlert.mail  
/root/service/mailTemplate/TemperatureAlertCancel.mail
```

#### 4.9.7 Disk Alert

디스크서버에는 **mdmonitor** 데몬이 실행되며 RAID 어레이의 상태를 모니터링 한다. 만약 문제가 생길 경우, 4.5.1절에서 설정한 대로, 디스크에 있는 `/root/admin/DiskAlert.sh`를 실행한다.

해당 실행파일은 마스터에 있는 `/root/service/DiskAlertMail.sh`를 실행하게 된다. 해당 파일을 통해 관리자에게 메일이 alert 메일이 보내지게 된다. 각 노드에서 다음과 같은 파일이 있는지 확인하자.

```
disk  
/root/service/DiskAlert.sh
```

### master

```
/root/service/DistAlertMail.sh  
/root/service/mailTemplate/DiskAlert.mail
```

다른 서비스들과 다르게 **mdmonitor**에서 실행해 주기 때문에, 추가적인 서비스를 실행할 필요는 없다.

## 4.10 SPG: Job Scheduler

SPG는 “Statistical Physics Group”의 약자인데, 작업 스케줄러(Job Scheduler)로 언제부터 이렇게 이름 붙여써 써왔는지 모른다. 현재 SPG는 우리 그룹의 최철호 박사님께서 파이썬(Python)으로 작성한 것이다.

2021년 8월 SPG 코드의 대대적인 개편이 이루어졌다. 파이썬 인터프리터는 `/usr/bin/python`을 사용하며, 3.9 버전 이상이어야 한다. 또한 처음 SPG를 사용하기 위해서는 다음의 패키지들을 **pip**을 통해 설치해 주면 된다.<sup>19</sup> 일반적으로 root 권한의 **pip**으로 패키지를 설치하는 것을 권장하지 않지만, 이 경우는 오직 **spg**만을 위해 사용하는 파이썬임을 감안하자. komplex를 사용하는 일반 유저들은 반드시 개인의 파이썬 가상환경을 만들어 사용하여야 한다.

```
# pip install tqdm termcolor colorama
```

해당 패키지들은 `/usr/lib/python3.X/site-packages/`에 설치된다.

**spg**에 관련된 모든 파일들은 `/root/spg`에 있다. 오픈소스로 공개했기 때문에, 깃허브<sup>20</sup>에서 확인할 수 있다. 자세한 코드에 대한 설명은 ??절을 확인하자. 마지막으로, 모든 사용자가 사용할 수 있도록, `/root/spg/spg.py` 실행파일은 `/usr/local/bin/spg`에 symbolic link되어 있으며, 다음을 통해 모든 사용자가 접근할 수 있도록 권한 설정을 해주어야 한다.

```
# chmod -R o+ /root/spg
```

## 노드 관리

**spg**가 사용하는 노드의 목록은 `/root/spg/group.json`에 있다. 새로이 노드가 추가된 경우에 용도에 맞춰서 해당 파일에 서식을 맞추어 내용을 추가해주면 된다. 삭제는 반대로 해당되는 항목을 지워주면 된다. 표준적인 json 문법에 맞게 작성하여야 하며 각 항목의 이름은 변경되면 안된다.

```
/root/spg/komplex.machine
```

```
{  
    "komplex1": {  
        "use": "True",  
    },  
}
```

<sup>19</sup>절대로 pacman을 사용하여 설치하지 말자.

<sup>20</sup><https://github.com/hoyunchoi/SPG>

```

    "name": "komplex1",
    "cpu": "Q9550",
    "num_cpu": "4",
    "ram": "8G",
    "comment": ""

},
"komplex2": {
    "use": "True",
    "name": "komplex2",
    "cpu": "Q9550",
    "num_cpu": "4",
    "ram": "8G",
    "comment": "No display output(YYYY.MM.DD)"

},
"komplex3": {
    "use": "False",
    "name": "komplex3",
    "cpu": "Q9550",
    "num_cpu": "4",
    "ram": "8G",
    "comment": "Mainboard failed. byebye(YYYY.MM.DD)"

},
"komplex4": {
    "use": "True",
    "name": "komplex4",
    "cpu": "Q9550",
    "num_cpu": "4",
    "ram": "8G",
    "comment": "Power issue(YYYY.MM.DD) -> Back to work(YYYY.MM.DD)"

}
}

```

위의 예시와 같이 **spg**에서 노드의 수리등으로 인해 일시적으로 사용하지 않기 위해서는 첫번째 키워드인 “use”를 “False”로 지정해 주어야 한다. 하지만 **spg**에서만 잡히지 않는 것이지 노드는 정상적으로 작동하고 있으므로 주의를 요한다. 반대로 노드가 꺼져 있거나 문제가 있을 때 목록에서 빼주지 않으면 **spg**가 계속 노드를 호출하면서 에러를 남긴다. 이와 같이 노드를 비활성화 할 때는 반드시 status-comment를 써주어 언제 어떠한 이유로 노드에서 비활성화 되었는지를 적어주어야

한다. 다시 복귀할때 역시 status-comment를 업데이트 해주자.

## 4.11 Machine Repair

komplex의 계산 노드들은 거의 24시간 켜져 있고 또한 계산을 쉬지 않고 수행하기 때문에 가정에서 사용하는 PC에 비해서 CPU와 메모리 그리고 특히 파워 서플라이의 소모가 커서 수명이 짧아진다. 대체로 계산 노드들이 고장나는 부분들은 다음과 같다.

- 전원공급장치(Power Supply): DELL 제품의 경우에 전원부가 주황색으로 점등되거나 아예 전원이 들어오지 않을 경우에 전원공급장치에 문제가 있다고 봐야 한다. 전원공급장치에 문제가 있을 경우에 전원공급장치만 교체해주면 문제가 해결된다.
- 마더보드(motherboard): 메모리 오류, USB 인식 실패, 부팅 실패 등이 있을 때는 마더보드에 문제가 있을 수 있다. 마더보드 교체는 새로운 컴퓨터를 구입하는 수준이기 때문에 비용을 고려해서 신중히 진행해야 한다.
- 메모리: 호환과 같은 문제가 아니라 메모리 또는 메모리 슬롯에서 고장이 발생할 수 있다. 메모리 자체의 문제인 경우에는 메모리만 교환하면 되지만 메모리 슬롯이 고장난 경우에는 마더보드를 교체해주어야 한다. kreat8의 경우에는 메모리들이 블레이드라고 불리는 기판에 번들로 꽂혀져 있고 블레이드가 다시 마더보드에 꽂혀있는 형태인데 블레이드가 고장을 일으켜서 교환한 적이 있다.

## Cooperative Firms

다산데이터 다산데이터는 DELL 대리업체로 DELL 관련 물품들의 구입/수리는 모두 이쪽으로 문의하면 된다. 연락처는 mail@dasandata.co.kr이다. 기타 소모품 구입도 해주신다.

SGI Korea SGI Korea는 SuperMicro 대리업체로, xenet8을 구입했던 곳이다.

# Chapter 5

## Problems and Solutions

이 장에서는 과거의 서버 관리를 하면서 겪은 문제와 해결법을 공유함으로써 다음 서버 관리자에게 같은 문제를 반복하지 않았으면 하는 바람에 작성되었다. 그럼에도 불구하고 서버 관리자 아니라 리눅스를 운영체제를 선택한 사용자들이라면 아마도 한 번쯤은 부팅조차 되지 않는 죄악은 상황을 맞다뜨리게 될 것이다. 왜냐하면 그건 리눅스는 아직도 개발되고 있는 운영체제이기 때문이다. 예기치 않은 문제는 언제나 발생할 수 있다는 것을 염두해고 해결해보려는 자세를 가지도록 하자.

### 5.1 General Solutions

일단 부팅이 된다는 전제 하에 문제를 해결하기 위해 해볼 수 있는 방법은 다음과 같다.

#### 5.1.1 systemd-journald

시스템 저널 데몬(**systemd-journald**, 간략히 **journald**)이 시스템 로그를 작성해준다. 시스템 로그에는 부팅된 시점부터 모든 기록이 남아있기 때문에 문제가 발생한 지점과 시점을 파악할 수 있게 해준다. 대부분 어디가 문제인지 몰라서 해매기 마련이므로 무엇보다 먼저 시스템 로그를 확인하도록 하자. 우리 그룹에서 서버 관리자에게 대대로 내려오는 금언을 속에 담아두록 하자. “문제를 알면 문제가 아니다.”

시스템 저널은 `$ journalctl`로 확인할 수 있다. `-b` 옵션을 붙이면 부팅이후의 로그를 볼 수 있고 `-r` 옵션을 이용하면 가장 최근 기록부터 보여준다. `$ watch`와 같이 쓰면 실시간으로 저널을 확인할 수도 있다. `-u daemon`으로 특정 **daemon**에 대한 로그만 확인할 수도 있으며 `-x` 옵션을 추가하여 더 많은 설명을 볼수도 있다. `-f` 옵션을 붙이면 새롭게 생성되는 로그들을 보여준다.

시스템 저널을 이용해서 여러메시지를 확인하고 얻은 여러메시지를 가지고 구글링과 아치리눅스 포럼을 통해서 문제를 해결하는게 가장 일반적인 문제 해결방법이다.

마스터와 디스크의 **journald**의 로그 파일은 기본적으로 `/var/log/journal/`에 있다. 그러나 노드와 터미널의 경우 해당 디렉토리가 없고 `/run/systemd/journal/`에 임시로 로그파일이 저장된다.<sup>1</sup> 따라서 노드와 터미널은 재부팅하게 되면 로그 파일이 새로 작성되어 부팅 이전에 있던 로그들이 삭제된다. 이렇게 노드와 터미널의 저널을 저장하지 않는 이유는 노드들과 터미널의 하드 디스크가 없기 때문에 노드들과 터미널의 로그파일들을 남기게 되면 노드의 수가 많아서 마스터의 하드디스크의 용량에 부담을 주기 때문이다.

그러나 특정 노드가 계속 문제될 경우에 부팅 전의 로그 파일을 확인해볼 필요가 있다. 먼저 저널 데몬을 활성화 시킬 노드에 접속한 후에<sup>2</sup> `/var/log/journal/`을 생성해주자. 그리고 재부팅 해주도록 하자. 그러면 다음 부팅 때부터 저널이 저장될 것이다. 만약 문제가 없을 때는 다시 해당 노드의 저널 데몬의 디렉토리를 삭제해주는 것을 잊지 말자.

### 5.1.2 ARM: Archlinux Rollback Machine

업데이트 이후 버전에 버그가 있거나 아직 불안정한 경우에 문제가 생길 때가 간혹 있다. 아치리눅스 커뮤니티가 재빠르게 반응하는 편이라 금방 해결책이 나오는 경우가 대부분이지만 간혹가다 바로바로 해결책이 올라오지 않은 경우도 있다. 이 경우에 해결책이 나올 때까지 다시 원래 버전으로 다운그레이드(downgrade) 필요가 있다.

ARM을 이용하면 특정 시점으로 pacman의 버전을 고정시킬 수 있는데 이를 이용하면 다운그레이드가 가능하다. ARM의 사용법은 아치 리눅스 위키의 ALA 페이지를 참고하도록 하자.<sup>3</sup> 아카이브(Archive)를 이용하면 pacman의 패키지 다운 속도 매우 느리다는 것을 참고하자.

## 5.2 Komplex Issues

### 5.2.1 Komplex Freeze

`spg` 가 작동이 잘 안되거나 서버의 반응이 느린 경우에 확인해볼 사항들은 다음과 같다.

- 디스크의 용량이 가득찬 경우에 서버의 작동이 느려질 수 있다. `$ df -h` 으로 확인해볼 수 있다. 이 경우에 용량이 가득찬 디스크를 사용하는 사용자 모두 서버를 이용하는데 문제를 겪을 수 있다. 특히 디스크 용량이 가득차면 비록 노드들에서 계산을 하더라도 추가적으로 결과를 낼 수 없게 되므로 해당 디스크를 이용하는 사용자들의 계산을 모두 다시 해야하는 불상사가 발생할 수도 있다. 간혹 일어났던 문제이므로 항상 서버 사용할 때 주의를 요하는 사안이다. 문제를 해결하는 방법은 파일들을 지워서 용량을 확보하는 방법밖에 없다.
- IO를 많이하는 프로그램이 있는지 확인하자. (**iotop**, **iftop**, **nethogs**) IO를 많이 하는 프로그램에 의해서 문제가 생겼을 경우에 프로그램을 실행시킨 사용자와 같은 디스크를

<sup>1</sup>이는 RAM에 저장되는 형식으로, 리눅스의 tmpfs의 형태로 존재한다. `df -h` 등으로 확인할 수 있다.

<sup>2</sup>괜히 터미널에서나 마스터에서 저널 데몬을 재실행시키기 말자.

<sup>3</sup>[https://wiki.archlinux.org/title/Arch\\_Linux\\_Archive](https://wiki.archlinux.org/title/Arch_Linux_Archive)

이용하는 다른 사용자들은 모두 서버의 작동이 느려지게 된다. 그리고 문제를 발생한 사용자와 다른 디스크를 이용하는 사용자들은 서버를 이용하는데 지장이 없다. 이 문제를 해결하는 방법은 IO를 최대한 적게하도록 프로그램을 다시 짜는 수밖에 없다. 아니면 고통을 디스크를 사용하는 다른 사람들과 같이 감수해야 겠다.

- 터미널에서 많은 작업이 이루어지고 있는지 확인하자.
- mathematica가 계산이 사용자가 종료하였는데도 idle하는 경우가 있다. 이 경우 강제 종료 해주어야 한다. 아직까지 이 문제로 서버가 크게 느려진 적이 없지만 idle하는 프로그램들이 누적되어 메모리를 많이 잡아먹게 경우에 서버의 작동을 느려지게 할 수 있다. (16.01.14)
- SSH에 문제가 생긴 경우에 `spg`의 작동에 문제를 줄 수 있다. SSH가 문제가 생겨서 `sshd`가 ‘Broken pipe’ 에러를 낸 후에 재시작 한 적이 있다. 이 때 sshd가 올바르게 재시작되지 않았는지 문제가 된 노드에 접속할 때마다 `sshd`가 좀비 프로세스(zombi process)로 계속 생성된 적이 있다. 이에 굉장히 많은 좀비 프로세스가 양산되었다.<sup>4</sup> 사실 더 직접적으로 문제를 일으킨 것은 `systemd-logind`이었다. 문제를 해결하기 위해서 `systemd-logind`를 재시작하려고 했으나 할 수 없었고 더불어서 warm boot로도 재부팅이 되지도 않아 cold boot로 재부팅할 수 밖에 없었다. 재부팅 이후에는 원상복귀 되었다. 이 문제는 SSH로 열 수 있는 session의 개수가 제한되어 있는데<sup>5</sup> 이보다 더 많은 session을 요청하면서 발생한 것으로 생각된다. (16.02.29)

### 5.2.2 Program Shut-down

노드에서 잘 돌아가던 프로그램이 꺼진 경우에 확인해볼 사항들은 다음과 같다.

- 자신의 코드가 문제가 없는지 다시 한번 확인해보자.
- 현재 노드에 남은 메모리보다 더 많은 메모리를 사용하는 프로그램이 추가될 경우에 프로그램이 꺼질 수 있다. 이때 꺼지는 프로그램은 남은 메모리보다 더 많은 메모를 요청한 프로그램이 꺼지게 된다. 즉 프로그램 돌린 순서가 중요한데 아니라 메모리를 요청한 순서가 중요하다. 만약에서 8GB의 메모리를 가지는 노드에서 3GB에서 5GB까지 점차 메모리가 사용이 증가하는 프로그램 A가 있다고 하자. 이때 프로그램 A가 아직 3GB를 사용할 때 4GB의 메모리를 한 번

<sup>4</sup>좀비 프로세스는 이미 ‘kill’ 명령을 받은 프로세스임에도 프로세스 목록에 올라가 있는 프로세스이다. 좀비 프로세스들은 컴퓨터의 리소스를 잡아먹지는 않으므로 크게 문제될 것은 없다. 좀비 프로세스를 확인 하는 방법은 `$ htop`이나 `$ ps aux`를 통해서 프로세스의 상태가 Z로 표시되어 있는지 확인하는 것이다. 좀비 프로세스는 이미 죽은 프로세스이기 때문에 보통의 방법(`pkill`, `kill`)를 가지고는 없앨 수 없다. 좀비 프로세스를 없애는 방법은 좀비 프로세스의 부모 프로세스(parent process)에서 `SIGCHLD` 를 주어서 죽인다. 그래도 죽지 않는다면 일단 부모 프로세스를 죽여서 좀비 프로세스를 고아 프로세스(orphan process)로 만든다. 그러면 `/sbin/init` 가 좀비 프로세스의 부모 프로세스가 된다. 그런 후에 `$ wait` 를 실행해주면 `/sbin/init` 가 좀비 프로세스를 죽이게 된다. 간혹 `/sbin/init` 를 부모 프로세스로 가지면서도 죽지 않는 좀비 프로세스들이 있는데 이 좀비 프로세스를 죽이는 방법은 내가 아직까지 아는 한 재부팅 뿐이다. 혹시 새로운 방법이 생긴 경우에 추가를 부탁한다.

<sup>5</sup>10개가 기본이고 `sshd` 설정파일인 `/etc/ssh/sshd_config`에서 MaxSessions에 값을 할당해서 바꿔줄 수 있다.

요청하고 꾸준히 사용하는 프로그램 *B*를 추가로 실행한다고 하자. 그러면 처음에는 *A*와 *B* 모두 실행되다가 *A*가 다시 5GB의 메모리를 요청하는 순간에 프로그램 *B*가 아닌 프로그램 *A*에 꺼지게 된다. 따라서 서버를 이용할 때 노드에 남아있는 메모리가 충분한지를 꼭 고려하여 남의 프로그램을 죽이는 일이 없도록 하자.

- 메모리가 하드웨어적으로 문제가 있는 경우에 메모리 오류를 내면서 프로그램 혹은 운영체제 자체가 꺼질 수 있다. 혹은 메모리와 마더보드(motherboard)를 연결해주는 메모리 슬롯(slot)이나 칩셋(chipset) 등에 문제가 있는 경우에도 마찬가지이다. 이 경우에는 해당 부품을 바꾸어줘야 한다. (16.03.08)

### 5.2.3 Unexpected Unmount

터미널 노드에서 디스크 서버로부터 nfs를 이용해서 마운트되어 있는 `/pds` 와 이하 디렉토리들이 갑자기 모두 언마운트하는 것을 확인했다.(16.08.18) 계산 노드와 마스터에서 언마운트가 되지는 않고 오로지 터미널 노드만 언마운트 되었었다. 이 경우에 사용자들의 홈디렉토리가 언마운트되기 때문에 각 사용자들은 홈디렉토리를 찾을 수 없다는 메시지를 받게 된다.

현재 마운트 되어 있는 것들의 목록을 확인하기 위해서 `$ mount` 를 이용하면 된다. 다시 디스크 서버로부터 마운트를 하려면 같은 명령어를 이용하면 된다. 예를 들어 `$ mount /pds/pds11` 와 같이 하면 된다.

이와 같이 갑자기 디스크들이 언마운트 되었는지 이유는 알 수 없지만, 정황을 말해보자면 한 디스크 서버(komplex14)에 대략 100여개의 계산 노드들이 I/O 작업을 하고 있었을 때 이러한 일이 벌어졌다. `journalctl`에서도 왜 언마운트 되었는지 나와 있지 않기 때문에 직접적으로 무엇이 문제였는지 확인할 수 없었다. 다만 디스크 I/O량이 많아진 것과 연관이 있을 것으로 생각이 되는데, NFS의 문제인지 혹은 물리적으로 디스크 서버의 I/O 능력의 한계인지 그것도 아니면 스위치의 스위칭 용량의 한계인지 확인해 볼 필요가 있을 것 같다.

## 5.3 Server Room Issues

### 5.3.1 정전 및 누전기 차단

외부에서 배전반 3개를 거쳐서 전기가 공급되고 있다. 가장 큰 배전반에 있는 차단기들의 각 용량은 60A이고, 작은 두 배전반에 들어 있는 차단기들의 각 용량은 30A이다. 대략 노드 하나가 소모하는 전류량이  $240W/250V \sim 1A$  정도임을 감안해서 전기배선을 하도록 하자. 웬만하면 멀티 콘센트(Extension)을 연달아서 연결하지 않도록 하자.

에어컨 역시 전기소모가 큰 데 서버와는 서로 다른 차단기를 통해서 전기 공급을 받고 있다. 삼성 에어컨 2개가 하나의 차단기를 공유하고 LG 에어컨 1개가 또 다른 하나의 차단기를 사용 중이다. 에어컨 및 전기 배선과 관련해서 시공사인 '세원 에어컨' (연락처: 010-6230-4541)에 연락해서 문의해보라.

차단기 하나에서 나오는 전선에 렉을 3개를 물려서 과전류로 인해 차단기가 내려가버린 적이 있다. 그로인해 노드를 포함한 마스터와 디스크가 모두 꺼져버려서 서버가 다운되어 버렸다. 그 이후에 디스크와 마스터가 서로 다른 차단기를 통해서 전류를 공급받도록 배선이 되어 있다. 추후에 노드를 대규모로 추가해야할 경우에 전기배선에 꼭 고려해주도록 하자. (16.01.01)

## 5.4 Archlinux Issues

### 5.4.1 GDM: Wayland (16.01.01)

GDM(Gnome Display Manager)이 Xorg가 아닌 Wayland를 기본으로 사용하게끔 되어 있는 경우에 GDM을 사용하는데 문제가 생길 수 있다. 이 경우에 Wayland 사용을 꺼주어야 한다. GDM 설정 파일에 들어가서 다음과 같은 줄에 '#'을 지워서 주석 처리를 풀어주면 된다.

```
/etc/gdm/custom.conf  
#WaylandEnable=false
```

아치리눅스 위키의 GDM 페이지에서 Use Xorg backend 부분을 참고하라.<sup>6</sup>

### 5.4.2 shadow (21.08.08)

사용자 계정의 비밀번호와 보안을 다루는 **shadow** 서비스가 지속적으로 fail되는 문제가 있었다. `systemctl status shadow.service`를 통해 문제 상황을 확인하니 다음과 같은 오류 메세지를 확인하였다.

```
Aug 08 00:00:38 terminal2 sh[95082]: user 'mail': directory '/var/spool/mail' does not exist
```

과거에는 `mail` 시스템 계정의 디렉토리가 `/var/mail` 이었지만, 현재는 `/var/spool/mail`로 바뀌었으며 호환성 유지를 위해 `/var/mail`은 symbolic link로 대체되어 있었다. 이를 확인하고, `NewNode.sh`를 수정하였다.

비슷한 형태로 다음의 에러도 `NewNode.sh`의 `/srv` 경로의 해당 디렉토리들을 추가하는 방식으로 해결하였다.

```
Aug 08 00:00:38 terminal2 sh[95082]: user 'ftp': directory '/srv/ftp' does not exist
```

```
Aug 08 00:00:38 terminal2 sh[95082]: user 'http': directory '/srv/http' does not exist
```

### 5.4.3 PAM (21.08.15)

서버의 로그인 및 비밀번호 정책을 다루는 PAM 모듈이 **journald**에서 다음과 같은 error를 내는 것을 확인하였다.

<sup>6</sup>[https://wiki.archlinux.org/title/GDM#Use\\_Xorg\\_backend](https://wiki.archlinux.org/title/GDM#Use_Xorg_backend)

### pam\_env.so

```
Sep 15 09:36:49 master sshd[482193]: pam_env(sshd:session): deprecated reading of user environment enabled
```

이는 PAM에서 지원하는 각 유저별 설정이 저장된 `~/.pam_environment` 파일의 기능이 deprecate 되며, `~/.config/environment.d/*.*.config`의 형태로 서비스가 되기 때문이다. komplex에서 본래 각 유저별 정책을 다르게 하지 않기 때문에, 이 기능을 비활성화 하였다.

비활성화는 `/etc/pam.d/system-login` 정책의 `pam_env.so`의 옵션을 `user_readenv=0` 으로 바꾸어 줌으로 할 수 있었다..

### /etc/pam.d/system-login

```
...
session required pam_env.so user_readenv=0
```

### pam\_systemd\_home

```
Sep 15 09:54:58 master systemd[482705]: pam_systemd_home(systemd-user:account): systemd-homed is not available: Unit dbus-org.freedesktop.home1.service not found.
```

이는 PAM이 `system-homed.service`를 사용하도록 업데이트가 되었는데, 현재 komplex에서 해당 서비스가 작동하지 않고 있어 생긴 문제였다. `systemctl`을 통해 해당 서비스를 start/enable 해주니 문제가 해결되었다.

### 5.4.4 Package update (21.09.17)

마스터에 특정 패키지(이번 경우 `texlive-most`와 `texlive-lang`)을 설치하고, 4.8.6절에서 다른 `UpdateSystemFilesOnNodes.sh`를 실행하였지만, 각 노드들에서 해당 패키지를 사용할 수 없는 문제가 있었다.

이는 몇몇 패키지들은 `/etc` 디렉토리에만 설정 파일을 두지 않고, `/var/lib`에도 접근을 하여야 하기 때문에 밝혀졌다. 따라서, `/var/lib`의 내용 역시 일부를 제외하고 마스터에서 복사해 오도록 스크립트를 수정하였다. 특히 L<sup>A</sup>T<sub>E</sub>X관련 디렉토리의 용량이 크기 때문에, 스크립트의 속도 저하가 조금 생겼다.

## 5.5 Hardware Issues

### 5.5.1 Intel: TSX bug (16.01.01)

intel 4세대(Haswell)와 5세대(Broadwell) 칩들이 Transactional Synchronization Extensions (TSX) 버그가 있음이 발견되었다. 이 문제를 해결하기 위해서는 마더보드의 펌웨어(firmware)를 업데이트 해주어야 한다. intel processors를 이용하는 경우 `intel-ucode`를 설치해주면 해결된다. 이

문제가 있는 경우에 [gdm](#)에 문제가 생긴다. 자세한 내용은 아치리눅스 위키의 microcode 페이지를 참고하도록 하자.

이 문제는 아치리눅스 초보자 가이드(Beginners' Guide)를 따라서 잘 설치한 경우라면 겪을 일이 없어졌다. (16.01.25)

### 5.5.2 Intel: Pentium FDIV bug (16.01.13)

intel 6세대 칩(Skylake)들이 FDIV bug가 일으킬 수 있다고 한다.<sup>7</sup> <sup>8</sup> 이 문제가 있는 경우에 BIOS 업데이트를 해주면 해결된다고 한다. 아직까지 komplex에서 이 문제가 발견된 바는 없다.

### 5.5.3 Intel: TSX bug (19.01.01)

intel 4세대(Haswell) 칩들을 사용하고 있는 노드들에서 다음과 같은 에러가 발생하였다.

```
# journalctl -r
Dec 29 13:28:46 komplex13 kernel: "echo 0 > /proc/sys/kernel/hung_task_timeout_secs"
disables this message.
Dec 29 13:28:46 komplex13 kernel:           Not tainted 4.19.12-arch1-1-ARCH #1
Dec 29 13:28:46 komplex13 kernel: INFO: task nfsd:479 blocked for more than 120
seconds.
```

리눅스 업데이트가 문제인 것으로 확인되었고, 롤백(18.11.01.)하여 해결했다. 롤백 하는 방법은 다음과 같다.

```
/etc/pacman.d/mirrorlist
Server = http://archive.archlinux.org/repos/2018/11/01/$repo/os/$arch
```

이 후 pacman을 통해 아치리눅스를 업데이트 하였다. (`$ pacman -Syyu`)

### 5.5.4 Intel NIC chipset: I219-V (21.09.15)

마스터의 메인보드에 있는 온보드 이더넷 칩셋 I219-V, 커널 명 e1000e가 [journald](#)에 다음과 같은 오류를 내며 지속적으로 재시작 하는 문제가 있다.

```
# journalctl -r
Sep 15 13:10:18 master kernel: e1000e 0000:00:1f.6 external: Detected Hardware Unit Hang:
                                TDH          <0>
                                TDT          <3>
                                next_to_use   <3>
                                next_to_clean  <0>
```

<sup>7</sup><http://arstechnica.co.uk/gadgets/2016/01/intel-skylake-bug-causes...>

<sup>8</sup>[https://www.reddit.com/r/programming/comments/40h98v/intel\\_skylake\\_bug...](https://www.reddit.com/r/programming/comments/40h98v/intel_skylake_bug...)

```
buffer_info[next_to_clean]:  
    time_stamp          <113e0d490>  
    next_to_watch       <0>  
    jiffies            <113e0e340>  
    next_to_watch.status <0>  
    MAC Status         <40080043>  
    PHY Status         <796d>  
    PHY 1000BASE-T Status <0>  
    PHY Extended Status <3000>  
    PCI Status         <10>
```

아직까지 그 원인을 파악하지 못하였으며, 다행히 이는 외부 네트워크 연결을 담당하는 NIC였기 때문에, 모든 외부 인터넷 연결을 터미널에 할당하게 되었다. e1000e 커널 관련 문제가 많이 있는 것으로 보아, 아마 마스터의 바이オス를 업데이트 하는등의 방식으로 해결할 수도 있을 것 같다.

# Chapter 6

## Appendix

### 6.1 Hardware Tips

서버를 운영하다 보면 다양한 종류의 하드웨어들을 구매하거나 보수하여야 한다. 이번 절에서는 어떤 종류의 하드웨어를 구매해야 하는지 살펴볼 것이다. 이번 절의 내용은 대부분이 권장 사항이며, 이를 따르지 않더라도 성능/가성비에서 약간의 손해나 관리자의 번거로운 작업이 추가될 뿐, 서버에 적용되지 않는 것은 아니라는 점을 확인하자.

독자가 어느정도 하드웨어적인 지식이 있는 것을 가정하고 작성하였으며, 잘 모르는 용어가 나온다면 구글링 하면 쉽게 결과가 나올 것이다.

#### 6.1.1 komplex(tenet)

서버의 대부분을 차지하는 cpu 연산용 노드들이다. 일반적인 데스크탑 pc용 부품들을 사용하며, 서버 한대당 대략 100만원 내외의 가격을 가진다. 구매는 일반적으로 다나와 등에서 부품들을 고르고, 업체등에 해당 부품들로 견적서를 받아 주문하면 될 것이다.

##### CPU

tenet을 구매할때 가장 먼저 골라야 하는 것은 cpu이다. 현재 서버에서는 모두 intel사의 cpu를 사용하고 있다. 이는 필수적이지는 않지만, 만약 amd사의 cpu를 사용하고자 한다면, 3.1절의 microcode를 따라, amd-ucode를 설치하고 boot loader 설정을 새로이 해 준 다음, 커널 이미지를 따로 만들어야 한다.

서버를 구매할 당시의 최신 i7 혹은 r7(ryzen-7) cpu를 고르자. 24시간 풀로드로 돌아가는 환경에 있으며, 후술될 이유로 쿨링이 부족할 수 있기 때문에, 오버클럭<sup>1</sup>은 해주지 않도록 하자. 인텔기준으로, 오버클럭을 진행하지 않더라도 베이스 클럭이 빠른 k시리즈<sup>2</sup>를 구매하여도 무방하다. cpu의

---

<sup>1</sup>cpu에 전력을 추가로 인도하여 더 빠른 속도로 작동하게 하는 tweak.

<sup>2</sup>non-K 시리즈보다 대략 10% 내외의 성능차이가 있다.

성능은 클럭만으로 볼 수 없고 IPC(Instructions Per Cycle) 혹은 캐시 메모리등에도 영향을 많이 받는다. 이러한 항목들은 대부분 최신 cpu일수록 더 뛰어나기 때문에 가능하다면 전 세대의 모델들은 피하자.

cpu를 구매할때 주의할 점은, 반드시 내장 그래픽 카드가 있어야 한다는 것이다. 바이오스 화면에서 pxe 부팅 등의 설정을 해주어야 하기 때문에 이를 모니터로 출력해 줄 수 있는 그래픽 카드가 있어야 한다. Intel의 경우 몇만원 더 저렴한 대신 내장 그래픽카드가 없는 F계열의 cpu를 구매하면 매우 골치아플 것이다. AMD의 경우는 대부분의 cpu가 내장그래픽이 없으며, G계열의 apu라 불리는 제품들만이 내장 그래픽 카드가 있기 때문에 구매에 주의하자.

## Main Board

메인 보드(마더보드)의 경우 cpu의 오버클럭을 진행하지 않기 때문에 고급형은 사용할 필요가 없다. 다만 그 전원 공급의 안정성과 내구성 역시 중요하기 때문에 미드레인지인 B시리즈를 많이 사용한다. 최대 RAM 지원 용량 역시 확인하여야 한다. 후술될 네트워킹 관련하여 만약 10G 이더넷을 사용할 예정이라면, 메인보드의 이더넷 포트가 10G를 지원하는지 확인하자.<sup>3</sup>

## RAM

램 용량은 연구실에서 사용하는 용량에 맞춰 구매하면 된다. 2021년 현재는 64GB의 램 용량을 기본으로 한다.

## Case

서버실에서는 각 노드들을 렉의 선반에 세워두기 때문에, 그 폭이 좁은 케이스를 사용하는 것이 좋다. 이를 일반적으로 LP(Low Profile)케이스라고 하며, 시중에 풀린 제품들 중에서는 선택의 폭이 상당히 좁을 것이다. 가능하다면 케이스 정면에 흡기가 있고 후면에 배기가 있는 형태를 선택하는 것이 좋다. 측면에 배기가 있을 경우, 노드 사이에 폭이 매우 좁기 때문에 쿨링이 원활하지 않을 수 있다. 이 조건을 만족하는 가장 저렴한 케이스로 고르면 될 것이다.

쿨링을 더 신경쓰고 싶다면, 렉 마운트형 제품을 선택해도 좋을 것이다. 서버실에서 사용되기 때문에 소음등의 문제로부터 자유로워 고려할 수 있는 방법이다. 일반적으로 2U 정도의 케이스가 적당하며, 팬이 많이 달린것이 선호된다.

## Cooler

2021년 최근들어 cpu에서 소모하는 전력의 양이 급격히 증가하고 있다. 과거에는 cpu를 구매하면 동봉되어 있는 기본 쿨러를 사용해도 어느정도 사용이 가능했지만, 이제는 따로 cpu 쿨러를 사용해야 한다. 상술한 케이스가 LP타입(혹은 2U 렉 마운트)으로 폭이 좁기 때문에 일반적인 타워형

---

<sup>3</sup>2021년 기준 최신 메인보드들의 Z계열 상위 라인들만이 10G 이더넷을 지원한다. 혹은, 10G 이더넷 PCIe 카드를 구매하여 이를 장착해도 된다. 다만 이 경우는 해당 메인보드가 외장 이더넷 카드에 PXE부팅을 지원하는지를 반드시 확인하여야 한다.

공랭쿨러는 대부분 사용할 수 없고, 대신 플라워형 쿨러들을 골라야 할 것이다. 대부분의 케이스들이 호환 가능한 쿨러의 높이를 적시해 놓기 때문에, 그 높이에 맞춰 TDP<sup>4</sup>가 가장 높은것으로 골라보자. 쿨링 성능이 좋을 수록 cpu가 더 빨리 동작할 수 있기 때문에, 쿨러에서 돈을 아끼지는 말자.

## Storage

komplex(tenet)에는 저장장치가 따로 필요하지 않다.

## Power

오직 cpu를 구동하는데만 파워가 사용되기 때문에, 고성능의 파워는 필요 없다. 상술했듯 최근 cpu들은 200W 이상의 전력을 소모할 수 있다. 따라서 300~400W 정도의 용량을 가진 파워를 장착하자.<sup>5</sup> 용량이 큰 파워를 사더라도 실제로 그만큼 전력을 사용하는 것이 아니라, cpu가 요구하는 정도의 전력만 사용하기 때문에, 조금 더 큰 용량을 선택하더라도 전력소비량이 많아지지는 않는다. 또한 파워는 노드들에서 가장 빈번한 고장이 발생하는 부품이다. 최소한 80Plus 인증을 받은 제품을 선택하자. 브론즈, 실버 이상의 인증을 받아도 가격 차이는 거의 나지 않을 것이다.

### 6.1.2 kreat(xenet)

일반적인 데스크탑용 cpu가 아닌, 서버용 cpu가 장착된 노드들이다. 일반적으로 다나와등에서 부품등을 검색할 수는 있지만, 케이스 및 파워등은 찾기 어렵기 때문에, 업체에서 제공하는 카탈로그 중에서 커스터마이징 등을 하여 구매할 수 있다. 일반적으로 DELL Technologies, HP Enterprise, SuperMicro, Tyan등의 제조사들이 많이 사용되며, 해당 제조사들의 국내 충판 등의 업체에서 이를 찾아볼 수 있다.

## CPU

서버용 cpu는 데스크탑용 cpu에 비해서 많은 수의 코어를 가지고 있는 대신, 각 코어의 속도는 더 느린다. 또한 많은 양의 RAM<sup>6</sup>를 지원하기 때문에, cpu 병렬연산을 하거나 거대한 메모리가 필요한 시뮬레이션을 돌리는 용도로 사용할 수 있다. 일반적인 계산의 경우 오히려 더 느릴 수도 있다는 점을 주의하자.

이외의 주의사항은 komplex(tenet)에서의 경우와 같다.

## Main Board

서버용 메인보드의 경우 데스크탑용 메인보드와 조금 다른 포트들이 있을 수 있다. 대표적으로 IPMI포트로, 이더넷 포트와 동일하게 생겼지만, 이는 노드의 os가 켜지지 않은 상태에서도 바이오

---

<sup>4</sup>Thermal Design Power. 쿨러가 쿨링할 수 있는 최대 열으로 이해하면 될 것이다.

<sup>5</sup>일반적으로 파워의 효율을 80% 정도로 가정하며, 어느정도 안전마진을 두기 위해서이다.

<sup>6</sup>데스크탑용 cpu의 경우 최대 128GB를 지원하지만 서버용 cpu는 1TB 이상도 가능한다.

스 진입등을 할 수 있다는 점에서 차이가 있다. 해당 포트에 따로 ip를 설정하면 원격으로 바이オス 설정을 하고 전원을 켜는 등의 작업을 할 수 있다.

또한 최신 디스플레이 포트인 HDMI 혹은 DP 포트가 없고 대신 RGB(VGA)포트만 있을 수도 있다.

## RAM

상술했듯 매우 많은 양의 RAM을 지원할 수도 있다. 지원하는 RAM 용량은 대부분 메인보드에 따라 달라진다.

### Case, Cooler, Power

일반적으로 랙마운트형을 사용하며, 폭은 19인치(600mm)로 고정되어 있으며 높이에 따라 1U, 2U, 4U<sup>7</sup> 등으로 구분된다. 케이스와 쿨러, 파워 모두 대부분 업체의 카탈로그에 있는대로 선택하여야 할 것이다. 쿨러는 크기는 작지만 팬 속도가 빨라 소음이 꽤 심할 것이며, 파워는 일반적으로 두개 이상이 장착되어 둘중 하나의 파워가 고장나더라도 다른 파워를 사용하여 전력을 감당할 수 있게 설계된다. 이 경우 구매한 업체에 연락하여 노드를 끄지 않고 파워를 교체할 수 있다.

## Storage

kreat(xenet)에는 저장장치가 따로 필요하지 않다.

### 6.1.3 kuda

GPU 연산을 위한 계산 노드이다. komplex(tenet)과 마찬가지로 다나와 등에서 부품들을 조사하여 업체에 견적을 내어 구매한다. 다만 그래픽카드의 가격이 높아지며 행정절차상 복잡해질 경우, GPU를 따로 구매하여 이를 직접 장착할 수도 있다.

만약 deep learning에 매우 특화된 연구를 진행하고자 한다면 게이밍용 그래픽카드(2021년 기준 30시리즈)가 아닌 데이터 센터용 그래픽카드 A시리즈를 고민해 볼수도 있을 것이다. 하지만 해당 카드들은 게이밍용 그래픽카드의 가격은 귀엽게 볼 정도로 매우 비싸기 때문에 예산이 정말 풍부할 경우 고려해 볼 수 있는 옵션일 것이다.

## CPU, Main Board

일반적으로 GPU를 4개 이상 장착하기 때문에, 이를 위해서는 워크스테이션급<sup>8</sup> 이상의 cpu와 메인보드가 필요하다. 메인보드의 PCIe 레인이 4대의 GPU를 지원하기 충분한지 확인하고, 이에 맞는 최소한의 cpu를 고르는 것도 좋은 방법일 것이다.

---

<sup>7</sup>랙의 높이를 뜻하는 unit으로, 1U는 약 4.5cm이다. 랙에는 이 unit을 기준으로 장비들을 장착할 수 있는

<sup>8</sup>intel의 경우 X시리즈의 i9, amd의 경우 스레드리퍼 시리즈 등

## GPU

GPU는 NVIDIA사의 최신 모델을 구매한다. 2021년 기준 gpu는 매우 많은 발전이 있으며, 각 세대간 차이가 극명히 갈리기 때문에 가능하다면 최신 모델을 사는 것이 가장 좋을 것이다. 대부분의 Deep learning framework 혹은 gpu 연산 가속은 NVIDIA의 CUDA를 사용하기 때문에, AMD사의 GPU는 고려하지 않는 것이 좋다.

## Cooler

CPU연산을 돌리는 경우는 많이 없지만, GPU연산을 위해 CPU에서도 어느정도 부하를 받기 때문에, 기본 쿨러는 되도록이면 피하자. 어느정도 성능의 공랭쿨러면 충분할 것이다.

반면 GPU의 경우 쿨링이 매우 큰 문제다. NVIDIA의 30세대가 되면서 성능이 늘어난 만큼 전력소모량이 급격히 올라가 그 발열량 역시 매우 많아졌다. 일반적인 워크스테이션 메인보드에 4 대의 그래픽 카드를 장착한다면 각 그래픽카드들 간의 공간이 거의 없어 열배출이 원활이 이루어지지 않는다. 이전 세대까지는 이를 블로워 타입의 쿨링으로 해결하였지만, 30세대에 들어오면서 이조차도 부족해졌다. 따라서 수냉 쿨링의 사용이 반강제 되었다. 내구성의 문제등으로 수냉쿨러를 기피하고 싶을 경우, kreat(xenet)과 같이 업체에 문의하여 랙마운트형 케이스와 쿨러를 사용하면 해결할 수 있다.

## Storage

OS를 설치해야 하기 때문에 적은 용량의 SSD를 구비하면 된다. 다만, 스토리지 IO가 많은 작업을 할 경우 네트워크 속도에서 병목현상이 있을 수 있으므로 필요하다면 데이터용 HDD를 추가해도 된다.<sup>9</sup>

## Power

장착된 CPU와 GPU에 맞는 파워를 잘 골라야 한다. GPU에서의 전력 소모량이 매우 많기 때문에 최소 1600W 이상의 파워를 구성하는 것이 안전하다. 다만 이는 매우 비싸질 수 있기 때문에, 1000W의 파워 두개를 사용하는 것도 고려해 볼만하다. 이 경우 케이스가 두개의 파워를 장착할 수 있어야 함은 당연할 것이다.

### 6.1.4 DISK

디스크 서버의 경우 많은 양의 하드 디스크를 장착할 수 있으면 된다.

## Main Board

OS를 설치할 1개의 SSD슬롯, 최소한 4개 이상의 SATA포트가 있으면 된다. 메인보드에서 지원하는 것보다 많은 양의 SATA포트를 사용하고자 한다면, PCIe 확장 카드를 사용할 수도 있다.

---

<sup>9</sup>이 경우에는 따로 설정하지 않는 이상 다른 서버 노드에서 접근할 수 없을 것이다.

또한, 지속적으로 서버에 디스크 정보를 제공해 주어야 하기 때문에 이더넷의 안정적 연결이 매우 중요하다. 인텔의 이더넷 칩셋을 사용하는 메인보드가 조금 더 안정성이 있어 선호된다.

### Case

최소한 4개 이상의 하드 디스크를 장착할 수 있는 케이스여야 할 것이다. 일반적인 타워형 케이스의 경우 최대 8개의 하드 디스크를 지원하는 것이 한계이다. 이보다 더 많은 수의 하드 디스크를 사용하고 싶다면, kreat(xenet)과 같이 서버형 케이스를 알아볼 수 있을 것이다. 4U 크기의 대형 케이스의 경우 수십개의 하드 디스크도 장착할 수 있다.

### Storage

어느정도 AS기간이 보장된 하드디스크를 사용하자. 이는 꼭 AS를 맡겨야 한다는 뜻이 아니라, 오랜 기간의 AS를 보장할 정도로 제조사측에서 내구성에 자신감이 있다 라는 것으로 이해하면 된다. 대부분 WD사의 하드디스크를 많이 사용하며, 이외에도 Seagate, 도시바등의 제조사의 제품을 사용해도 무방하다. 3.5인치 하드 디스크에서도 PC용, NAS용, 데이터센터용 등으로 그 구분이 나뉘어져 있다. 뒤로 갈수록 안정성이 높아진다.

현재 서버에서는 PC용 HDD를 사용하고 있다. 이는 하드 디스크가 고장나기 전에 일반적으로 용량이 거의 차서 사용할 수 없어진 경우가 많기 때문에 그러하다. 하지만 최근 4TB이상의 고용량 하드디스크를 사용함에 따라 하드디스크의 용량이 차는 속도보다 AS기한이 더 빨리 올수도 있기 때문에 24시간 작동을 기본 원칙으로 하는 NAS용 하드디스크의 사용도 고려해 볼만 하다.

### 6.1.5 Network device

네트워크 장비는 2021년 현재 서버에서 가장 병목을 일으키는 부분이다. 2021년 기준 내부망은 기가비트(1Gbps)로 운영되고 있으며, 여유가 된다면 내부망을 10Gbps로 업그레이드 하는 방향이 서버를 쾌적하게 사용할 수 있는 가장 첫번째 걸음이 될 것이다. 다른 모든 디지털 연결과 같이, 인터넷 속도는 연결된 두대의 네트워크 장치와 이더넷 선이 지원하는 최대 속도중 가장 낮은 속도로 연결된다. 따라서 10Gbps로 업그레이드 하기 위해서는 스위치와 이더넷 케이블, 그리고 10Gbps 통신을 할 노드 모두가 해당 속도를 지원해야 한다.

외부망의 경우, 2.2절에서 언급한대로, 서울대학교 전산원측에 학외 인터넷 속도 제한 해제 요청을 한다면 100Mbps 이상의 인터넷 속도를 제공받을 수 있다. 다만, 현재 외부 인터넷을 담당하는 스위치가 100Mbps만 지원하기 때문에, 해당 스위치도 바꾸어 주어야 할 것이다.

### Node

2021년 현재 대부분의 pc 메인보드에 있는 이더넷 포트는 기가비트(1Gbps)를 지원한다. 일반적인 경우 해당 포트들을 사용하면 되지만, 만약 10Gbps로 업그레이드 하고자 한다면, 10Gbps PCIe

확장카드를 사용하여 장착해 주어야 할 것이다. 모든 계산 노드들을 업그레이드 해 줄 필요는 없으며, 고속 통신을 하고자 하는 노드들만 업그레이드 해주어도 괜찮다.

### Switch

외부망의 경우, 그림 2.1처럼 라우팅을 중앙 전산원측에서 진행해 주기 때문에 서버실 벽에서 나오는 이더넷 포트를 L2 계층의 스위치에 연결해 주면 된다. 만약 하나의 IP주소를 사용하며 포트에 따라 어떤 터미널에 접속할지를 결정하고 싶다면, L3 계층의 라우터(공유기)의 맥 주소에 IP 주소를 발급받고, 해당 라우터에서 터미널들을 연결해 주면 된다.

내부망의 경우, 마스터가 제공해주는 하나의 서브넷으로 이루어졌기 때문에, L2 계층의 스위치로 충분하다.<sup>10</sup> 각 랙 별로 하나의 스위치가 있으며, 그 스위치들로부터 서버실 중앙에 있는 메인 스위치로 연결되는 구조이다. 서버실에서 특별히 선호하는 브랜드는 없으며, 공신력 있는 CISCO, HPE, 넷기어등의 브랜드에서 다양하게 사용중이다.

PoE(Power on Ethernet)포트는 사용하지 않기 때문에 이를 고려할 필요는 없으며, 대신 최소 24개 이상의 기가비트 포트가 있는 스위치를 사용하자. 만약 메인 스위치를 바꾸어야 한다면, 최대한 처리량과 스위칭 기능이 높은 모델로 선택하도록 하자.

만약 10Gbps로 업그레이드 하고 싶다면, 그 선택의 폭이 많이 줄어들 것이다. 2021년 기준 다나와 등에서는 24포트 이상의 10Gbps를 지원하는 모델은 거의 찾아보기 힘들다. 따라서 Cisco등의 제조사에 직접 문의를 하여 모델을 문의하는 것이 좋을 것이다.

### Cable

현재 서버실에서는 Cat.5e와 Cat.6 규격의 UTP 케이블을 혼용하고 있다. 이는 기가비트 연결에 충분한 수준이며, 시중에서 구할 수 있는 가장 흔한 규격이기 때문에 구매하기 어렵지 않을 것이다. 다만, 랙 스위치와 메인 스위치를 연결하는 케이블의 경우 그 길이가 10m가 넘을수도 있기 때문에, 가능하다면 Cat.6 규격의 케이블을 사용하자.

10Gbps로 업그레이드 한다면 케이블은 크게 두가지 선택을 할 수 있다. 첫번째는 Cat.6a 이상의 UTP 혹은 SFTP 케이블을 사용할 수 있다. 일반적인 이더넷 케이블과 비슷하게 생겼으며, 케이블 피복에 그 규격이 적혀 있다. 두번째 선택으로는 광케이블인 sfp+ 케이블이다. 이는 일반적인 이더넷 포트(RJ-45)가 아닌 SFP라는 포트에 사용할 수 있다. 광 케이블인 만큼 가격이 매우 비싸지만, 구리선으로 되어 있는 UTP/SFTP 케이블에 비해 외부 전자기장의 영향을 훨씬 덜 받기 때문에 안정적인 통신을 할 수 있다. 어떤 케이블을 사용할 지는 연결할 노드와 스위치의 포트, 그리고 연결 거리를 고려하여 선택해야 한다.

### 6.1.6 Rack

랙은 서버장비들을 구성하고 보관할 수 있는 표준적인 규격의 장비이다. 가로폭은 표준적으로 19인치를 사용하며 높이는 제품마다 다르지만 현재 서버실의 경우 최소 40U 이상 높이의 랙을

---

<sup>10</sup>L2와 L3계층, 스위치와 라우터 용어에 대해 인터넷상에 다양한 설명 자료들이 있으니 찾아보자.

사용한다.

또한 랙의 깊이에 따라 서버랙 혹은 허브랙으로 나뉜다. 허브랙은 깊이가 얕아서(약 50-80cm) 스위치등의 네트워크 장비들을 거치하기 위한 랙이며, 서버랙은 그 깊이가 깊어(약 100cm) 서버 장비들을 넣을 수 있다. 현재의 서버에서는 서버랙을 기준으로 사용하지만, 구매 시기에 따라 허브랙도 구매한 적이 있다.

## 6.2 Custom Hooks

3.7절에서 노드의 initial ramdisk에 사용된 `/etc/mkinitcpio-komplex-node.conf` 의 custom hook `net-komplex` 와 `mount-master`에 대해 다룬다. 이들은 `/etc/initcpio/install`에 위치한 build 파일과 `/etc/initcpio/hook`에 위치한 runtime hook으로 이루어져 있다.

### net-komplex

`net-komplex` 은 부팅 시에 NIC설정을 해주는 역할을 한다. 이는 보통의 부팅에서 NIC의 설정을 파일 시스템에서 불러오는데, PXE부팅에서는 파일시스템을 마운트가 아직 안되어 있기 때문에 NIC 설정을 따로 설정해야하기 때문이다.

```
/etc/initcpio/install/net-komplex

#!/bin/bash
build() {
    add_checked_modules '/drivers/net/'
    add_module nfsv3?
    add_module nfsv4?
    add_binary "/usr/lib/initcpio/ipconfig" "/bin/ipconfig"
    add_binary "/usr/lib/initcpio/nfsmount" "/bin/nfsmount"
    add_runscript
}
```

```
/etc/initcpio/hooks/net-komplex

run_hook() {
    local line i net_mac bootif_mac bootif_dev defaultrootpath defaultserver
    local DEVICE
    local IPV4ADDR IPV4BROADCAST IPV4NETMASK IPV4GATEWAY IPV4DNS0 IPV4DNS1
    local HOSTNAME DNSDOMAIN NISDOMAIN ROOTSERVER ROOTPATH
    local filename

    if [ -z "${ip}" -a -n "${nfssaddrs}" ]; then
        ip="${nfssaddrs}"
```

```

    fi

    if [ -n "${ip}" ]; then
        if [ -n "${BOOTIF}" ]; then
            bootif_mac=${BOOTIF#01-}
            bootif_mac=${bootif_mac//-/:}
            for i in /sys/class/net/*/*; do
                read net_mac < ${i}
                if [ "${bootif_mac}" == "${net_mac}" ]; then
                    bootif_dev=${i#/sys/class/net/}
                    bootif_dev=${bootif_dev%/*}
                    break
                fi
            done
            #This does not work when static ip address is given
            #ip="${ip}::${bootif_dev}"
            # Added by DJ {{{
            if [ "${ip}" == "dhcp" ]; then
                ipconfig_args="-c dhcp ${bootif_dev}"
            else
                ipconfig_args="${bootif_dev} ${ip}"
            fi
            # }}}
        else
            # Added by DJ {{{
            ipconfig_args="ip=${ip}"
            # }}}
        fi
        # setup network and save some values
        # Added by DJ {{{
        ipconfig $ipconfig_args
        # }}}
        for conf in /tmp/net-*.conf; do
            [ -f "$conf" ] && . "$conf"
        done
        # calculate nfs_server, nfs_path and nfs_option for later nfs mount
        if [ "${root}" = "/dev/nfs" -o "${nfsroot}" != "" ]; then
            # parse ROOTPATH if defined by dhcp server
            if [ -n "${ROOTPATH}" ]; then

```

```

line="${ROOTPATH}"
nfs_server="${line##%:*}"
[ "${nfs_server}" = "${line}" ] && nfs_server="${ROOTSERVER}"
defaultserver="${nfs_server}"
line="${line##*:}"
nfs_path="${line}"
defaultrootpath="${nfs_path}"
else
    # define a default ROOTPATH
    if [ "${ROOTPATH}" = "" ]; then
        defaultrootpath="/tftpboot/${IPV4ADDR}"
    fi
fi
# parse nfsroot if present (overrides ROOTPATH)
if [ -n "${nfsroot}" ]; then
    line="${nfsroot}"
    nfs_server="${line##%:*}"
    [ -z "${nfs_server}" ] && nfs_server="${defaultserver}"
    line="${line##*:}"
    nfs_path="${line##%,*}"
    line="${line##${nfs_path}}"
    [ -z "${nfs_path}" ] && nfs_path="${defaultrootpath}"
    nfs_option="${line#\",}"
    fi
# ensure root and filesystem type are set proper for nfs boot
root="/dev/nfs"
rootfstype="nfs"
echo "NFS-Mount: ${nfs_server}:${nfs_path}"
# set mount handler for NFS
mount_handler="nfs_mount_handler"
fi
fi
}

nfs_mount_handler() {
if [ -z "$nfs_server" -o -z "$nfs_path" ]; then
    err "Unable to mount root filesystem over NFS: wrong parameters."
    echo "You are being dropped to a recovery shell"
    echo "    Type 'exit' to try and continue booting"
}

```

```

    launch_interactive_shell
    msg "Trying to continue (this will most likely fail) ..."
fi
nfsmount ${nfs_option:+-o ${nfs_option}} "${nfs_server}":${nfs_path}" "$1"
}

```

### mount-master

`mount-master` 은 `master:/usr` 를 각 노드들의 `/usr` 로 마운트해주는 역할을 한다.

```

/etc/initcpio/install/mount-master

#!/bin/bash
build() {
    add_binary findmnt
    add_runscript
}

```

```

/etc/initcpio/hooks/mount-master

#!/usr/bin/ash
run_latehook() {
    local usr_source usr_mountopts realtab=/new_root/etc/fstab
    if [ -f "$realtab" ]; then
        if usr_source=$(findmnt -fnero source --tab-file="$realtab" -T /usr); then
            usr_mountopts=$(findmnt -fnero options --tab-file="$realtab" -T /usr)
            msg ":: mounting '$usr_source' on /usr"
            nfsmount -o "rw,hard" "$usr_source" /new_root/usr
        fi
    fi
}

```

## 6.3 Custom Bash Scripts

이 절에서는 4.8절에서 소개된 `/root/admin` 에 있는 배수 스크립트들의 본문 내용을 서술한다. 각 script들이 어떻게 작성되어 있는지 알고 싶거나 수정이 필요한 경우, 해당하는 내용을 보며 공부하도록 하자.

### CopyToNodes

마스터에 존재하는 `source` 파일을 지정한 노드들의 `target` 위치에 복사한다.

```

/root/admin/Common.sh

#!/bin/bash

# This script copies a file to multiple nodes with a common name prefix

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # Gaurantee this is master/root


# Args
source=$1      # file path to copy   ex) /root/admin/hosts.template
target=$2      # file path to arrive ex) /etc/hosts
nodenamePrefix=$3 # ex) tenet
fromNode=$4     # ex) 1
toNode=$5      # ex) 4
if [ $# != 5 ]; then
    echo "Error: You should provide five arguments."
    echo "ex) /root/admin/hosts.template /etc/hosts tenet 1 4"
    exit
fi
if [ "${target:0:1}" != "/" ]; then
    echo "Error: You should provide absolute path as a target path."
    exit
fi

# Copy files
echo "Copying file \"\$source\" to \"\$target\" on \
\$nodenamePrefix\$fromNode ~ \$nodenamePrefix\$toNode"
n=$fromNode
while [ $n -le $toNode ]; do
    nodename=$nodenamePrefix$n
    croot=$tftpboot/$nodename
    if [ -e $croot ]; then
        echo "cp -p \$source \$croot\$target"
        cp -p $source $croot$target
    else
        echo "\$nodename does not exist. skipped."
    fi
    n=$((n + 1))
done

```

## NewNode

서버에 os가 없는 새로운 노드를 추가하는 역할을 한다. 본문은 다음과 같고, 내용을 이해하려면 3장에서 노드와 관련된 절들을 참고해야 할 것이다.

```
/root/admin/NewNode.sh

#!/bin/bash

# This script create root file system and pxelinux config file for new node.
# This script uses following templates
# Always make sure to keep the template files up to date.

# fstab.template
# hosts.template
# ntp.conf.template
# sshd_config.template
# pxelinuxcfg.template

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # tftpboot, tftpbootcfg, gaurantee this is master/root

# Args
nodename=$1                      #ex) tenet100
macaddr=$(echo "$2" | awk '{print tolower($0)}')    #ex) 00:11:22:33:44:55
ipaddr=$3                          #ex) 10.0.2.100
if [ $# != 3 ]; then
    echo "Error: You should provide three arguments."
    echo "ex) tenet# 00:11:22:33:44:55 10.0.2.##"
    exit
fi

# Root(/) directory of new node
croot=$tftpboot/$nodename

# Double check
echo "NAME: $nodename"
echo "MAC : $macaddr"
echo "IP  : $ipaddr"
echo "ROOT: $croot"
```

```

echo ""
echo -n "Is it ok? (y/n) : "
read -e isOK
if [ "x$isOK" != "xy" ]; then
    exit
fi

# ----- Basic directories at root /
-----
echo ""
echo "Creating root file system for new node"
mkdir $croot
mkdir $croot/boot          # Will mount master:/boot
mkdir $croot/dev
cp -Prp /dev/* $croot/dev
mkdir $croot/etc
mkdir $croot/home
ln -s usr/lib $croot/lib   #This becomes legal after remote mounting of /usr
ln -s usr/lib $croot/lib64 #This becomes legal after remote mounting of /usr
ln -s usr/bin $croot/bin   #This becomes legal after remote mounting of /usr
ln -s usr/bin $croot/sbin  #This becomes legal after remote mounting of /usr
mkdir $croot/opt           # Will mount mater:/root
mkdir $croot/proc
mkdir $croot/root          # Will mount master:/root
mkdir $croot/run
mkdir $croot/sys
mkdir $croot/tmp
mkdir $croot/usr            # Will mount master:/usr
mkdir $croot/var            # Special care will be done
mkdir $croot/srv

# ----- Disk server -----
# You should update here when adding/modifying disk server
mkdir $croot/pds
mkdir $croot/pds/pds11
mkdir $croot/pds/pds21
mkdir $croot/pds/pds31
mkdir $croot/pds/pds41

```

```

mkdir $croot/pds/pds42
mkdir $croot/pds/pds51
mkdir $croot/pds/pds61
mkdir $croot/pds/pds62
mkdir $croot/pds/pds71

# ----- var directory -----
cvar=$croot/var
mkdir $cvar/db
mkdir $cvar/empty
mkdir $cvar/games
mkdir $cvar/lib
# mkdir for /var/lib/*
for x in $(ls -l /var/lib/ | awk '($0~"^d") {print $9}'); do
    mkdir $cvar/lib/$x
done
# mkdir for /var/lib/***
mkdir $cvar/lib/nfs/rpc_pipefs
mkdir $cvar/lib/nfs/sm
mkdir $cvar/lib/nfs/sm.bak
mkdir $cvar/lib/nfs/v4recovery
mkdir $cvar/local
ln -s ../run/lock $cvar/lock
mkdir $cvar/log
mkdir $cvar/opt
ln -s ../run $cvar/run
mkdir $cvar/spool
mkdir $cvar/spool/mail
chmod 1777 $cvar/spool/mail
ln -s spool/mail $cvar/mail
mkdir $cvar/tmp

# ----- tmp directory -----
chmod 777 $croot/tmp

# ----- srv directory -----

```

```

csrv=$croot/srv
mkdir $csrv/ftp
chmod 555 $csrv/ftp
mkdir $csrv/http

# ----- etc directory -----
echo ""
echo "Modifying config files in client etc directory"
cp -Prp /etc/* $croot/etc
cetc=$croot/etc
mkdir $cetc/hooks
mkdir $cetc/interfaces

echo $nodename > $cetc/hostname          # /etc/hostname
cp -f fstab.template $cetc/fstab         # /etc/fstab
cp -f ntp.conf.template $cetc/ntp.conf    # /etc/ntp.conf
cp -f sshd_config.template $cetc/ssh/sshd_config # /etc/sshd_config
echo "127.0.0.1 localhost.localdomain $nodename.snu.ac.kr $nodename" \
> $cetc/hosts
cat hosts.template >> $cetc/hosts        # /etc/hosts

rm -f $cetc/ssh/*key          # Remove ssh private keys of master
rm -f $cetc/ssh/*key.pub       # Remove ssh public keys of master
rm -rf $cetc/netctl/*          # Remove netctl configuration of master
echo "" > $cetc/exports         # Remove nfs exports configuration of master
rm -rf $cetc/cron*             # Remove any cron jobs of master
rm -f $cetc/dnsmasq*          # Remove any dnsmasq related files of master

# ----- Enable services -----
systemdUnitDir="$cetc/systemd/system"
systemdTargetDir="$cetc/systemd/system/multi-user.target.wants"
# Remove any services running at master
for x in $(ls -l $systemdUnitDir | awk '($0!~"^d"){print $9}'); do
    rm -f $systemdUnitDir/$x
done
rm -f $systemdTargetDir/*

```

```

# Services each nodes should run
ln -s /usr/lib/systemd/system/ntp.service $systemdTargetDir/ntp.service
ln -s /usr/lib/systemd/system/remote-fs.target $systemdTargetDir/remote-fs.target
ln -s /usr/lib/systemd/system/rpcbind.service $systemdTargetDir/rpcbind.service
ln -s /usr/lib/systemd/system/nfs-client.service $systemdTargetDir/nfs-client.service
ln -s /usr/lib/systemd/system/sshd.service $systemdTargetDir/sshd.service
ln -s /usr/lib/systemd/system/cpupower.service $systemdTargetDir/cpupower.service
ln -s /dev/null $systemdUnitDir/man-db.timer

# ----- pxelinux config file -----
echo "Creating pxelinux config file"
macaddrX='echo "$macaddr" | sed 's/:/-/g' | awk '{print tolower($0)}'
crootX='echo "$croot" | sed 's/\/\/\\\\\\\\//g'
cat pxelinuxcfg.template \
| sed "s/_HOSTNAME_/$nodename/g" \
| sed "s/_CROOT_/$crootX/g" \
| sed "s/_BOOTIF_01-$macaddrX/g" > $tftpbootcfg/01-$macaddrX

# ----- /etc/dnsmasq.conf -----
echo "Adding new node entry to /etc/dnsmasq.conf"
echo "dhcp-host=$macaddr,$ipaddr,infinite #$nodename" >> /etc/dnsmasq.conf

# ----- 70-persistent-if-name-tenet.rules -----
echo \# $nodename >> /root/admin/70-persistent-if-name-tenet.rules
echo SUBSYSTEM==\"net\", ACTION==\"add\", ATTR{address}==\"$macaddr\", \
NAME=\"internal\" >> /root/admin/70-persistent-if-name-tenet.rules

# Print final messages
echo ""
echo "Done"
echo ""
echo ">> You should restart the dnsmasq daemon by 'systemctl restart dnsmasq.service'"
echo ""
echo ">> You should make initrd by 'mkinitcpio -p tenet'"
echo ""

```

```
echo ">> You should update machine info on /root/spg"
```

## DeleteNode

고장난 노드를 komplex에서 제외하는 역할을 한다. 마스터의 `/tftpboot/pixelinux.cfg` 와 `/tftpboot`에서 해당 노드의 디렉토리를 모두 지우고, `/etc/dnsmasq.conf`를 업데이트 해준다.

```
/root/admin/DeleteNode.sh

#!/bin/bash

# This script deletes nodes

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # tftpboot, tftpbootcfg, gaurantee this is master/root

# Args
nodename=$1      # ex) tenet100
if [ $# != 1 ]; then
    echo "Error: You should provide an arguments."
    echo "ex) tenet100"
    exit
fi

# Double check
echo "Do you really want to DELETE ${nodename}?"
echo -n "Is it ok? (y/n) : "
read -e isOK
if [ "x$isOk" != "xy" ]; then
    exit
fi

# Count number of input node name in pixelinux.cfg
countx=$(grep -l "^label ${nodename}\$" $tftpbootcfg/* | wc -l)
if [ $countx != 1 ]; then
    echo ""
    echo "Error: There are multiple nodes matching with the name."
    echo "Check \`grep '^label ${nodename}\$' $tftpbootcfg/*\`."
    exit
fi
```

```

# Delete pxelinux.cfg
cfgfile=$(grep -l "^label ${nodename}\$" $tftpbootcfg/*)
echo "Delete file : $cfgfile"
rm -f "$cfgfile"

# Delete root
croot=$tftpboot/$nodename
echo "Delete directory : $croot"
rm -rf "$croot"

# Backup dnsmasq.conf and update
echo "Back up /etc/dnsmasq.conf & delete entry of $nodename in the file."
mv /etc/dnsmasq.conf /etc/dnsmasq.conf.bak
macaddr=$(echo "$cfgfile" | awk '{print $2}' FS='/'| tr - :)
pat="^dhcp-host=$macaddr,10\\\.\0"
cat /etc/dnsmasq.conf.bak | awk '$0 !~ pat {print $0}' pat="$pat" > /etc/dnsmasq.conf

# Delete node entry of 70-persistent-if-name-tenet.rules
echo "Deleting node $nodename in file 70-persistent-if-name-tenet.rules"
cat /root/admin/70-persistent-if-name-tenet.rules \
| awk -v mac=$macaddr -v node="# ${nodename}" '$6 != mac && $0 != node){print $0}' \
FS=''' > /root/admin/70-persistent-if-name-tenet.rules

# Print final messages
echo ""
echo "Done"
echo ""
echo ">> You should restart the dnsmasq daemon by 'systemctl restart dnsmasq.service'"
echo ""
echo ">> You should make initrd by 'mkinitcpio -p tenet'"
echo ""
echo ">> You should update machine info on /root/spg"

```

## NewUser

마스터에 새로운 계정을 추가하는 역할을 한다. 먼저 새로운 유저를 홈 디렉토리를 특정하여 만들며, `users` 그룹에 새로운 계정을 추가한다. 그리고 비밀번호와 SSH key를 생성해 서버에 저장하도록 한다.

```

/root/admin/NewUser.sh

#!/bin/bash
# This script generates new user and assign home directory

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # Gaurantee this is master/root


# Args
username=$1      # ex) hongkildong
hdir=$2          # ex) /pds/pds11. New user's homedir is $2/$1
comment=$3        # Real user name(e-mail address)

if [ $# != 3 ]; then
    echo "Error: You should provide three arguments."
    echo "ex) hongkildong /pds/pds11 \"KilDong Hong(hong@snu.ac.kr)\""
    exit
fi
if [ "${hdir:0:1}" != "/" ]; then
    echo "Error: You should provide absolute path as second argument."
    exit
fi

# Double check
userhome=$hdir/$username
echo "Create user \"$username\" with home \"$userhome\""
echo ""
echo -n "Is it ok? (y/n) : "
read -e isOK
if [ "x$isOk" != "xy" ]; then
    exit
fi

# Generate user with home directory and passwd
useradd -b $hdir -c "$comment" -m $username
echo "User added at $userhome with comment $comment"
passwd $username
echo "password set for $username"

```

```

gpasswd -a $username users
echo "users group included"

# Generate keys
su - $username -c "ssh-keygen -t rsa -N ''"
su - $username -c "ssh-keygen -t dsa -N ''"
su - $username -c "ssh-keygen -t ecdsa -N ''"
su - $username -c "cp $userhome/.ssh/id_rsa.pub $userhome/.ssh/authorized_keys"
su - $username -c "chmod 700 $userhome/.ssh/authorized_keys"

# Print final messages
echo ""
echo "Done"
echo ""
echo ">> You should update user information to all nodes including OS servers"
echo ""
echo ">> Use UpdateUserInfoOnNodes.sh and UpdateUserInfoOnOSServers.sh"

```

## UpdateUserInfoOnNodes & UpdateUserInfoOnOSServers

마스터의 계정중 root를 제외한 사용자의 계정을 os가 없는 노드와 있는 노드들에 연동해 주는 역할을 한다. os가 없는 경우 마스터의 내용을 그대로 가져오고, os가 있는 경우 시스템 계정을 제외하고 가져온다. 특히 `UpdateUserInfoOnOSServers.sh`의 경우 각 OS의 시스템 계정의 홈 디렉토리도 설정해 준다.

```

#!/root/admin/UpdateUserInfoOnNodes.sh

#!/bin/bash
# This script synchronizes user information of master to terminal and nodes

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # Gaurantee this is master/root

# Args
prefix=$1        # ex) tenet
fromNode=$2       # ex) 10
toNode=$3         # ex) 14
if [ $# != 3 ]; then
    echo "Error: You should provide three arguments."
    echo "ex) tenet 10 14"
    exit

```

```

fi

# Double check
echo "Updating user information of $prefix$fromNode to $prefix$toNode"
echo ""
echo -n "Is it ok? (y/n) : "
read -e isOK
if [ "x$isOK" != "xy" ]; then
    exit
fi

# Copy all user informations from master to nodes
./CopyToNodes.sh /etc/passwd /etc/passwd $prefix $fromNode $toNode
echo ""
./CopyToNodes.sh /etc/group /etc/group $prefix $fromNode $toNode
echo ""
./CopyToNodes.sh /etc/shadow /etc/shadow $prefix $fromNode $toNode
echo ""
./CopyToNodes.sh /etc/gshadow /etc/gshadow $prefix $fromNode $toNode

# Print final messages
echo ""
echo "Done"

```

```

/root/admin/UpdateUserInfoOnOSServers.sh

#!/bin/bash
# This script synchronizes user information of master to disk/gpu server
# This script is VERY insecure.
# This should be used only in a internal network you can trust.

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # Gaurantee this is master/root

# Args
prefix=$1        # ex) disk/gpu
fromNode=$2      # ex) 1
toNode=$3        # ex) 2
if [ $# != 3 ]; then
    echo "Error: You should provide three arguments."

```

```

        echo "ex) disk 1 2"
        exit
    fi

    # Double check
    echo "Updating user information of $prefix$fromNode to $prefix$toNode"
    echo ""
    echo -n "Is it ok? (y/n) : "
    read -e isOK
    if [ "x$isOK" != "xy" ]; then
        exit
    fi

    # Generate list of servers
    servers=""
    n=$fromNode
    while [ $n -le $toNode ]; do
        servers="$servers $prefix$n"
        n=$((n+1))
    done

    function work {
        local server=$1

        # Temporary files to store passwd/shadow and group/gshadow
        local tmpPasswd=$(mktemp)
        local tmpShadow=$(mktemp)
        local tmpGroup=$(mktemp)
        local tmpGShadow=$(mktemp)

        # Get system user information of os server: UID<1000 (root, systemd-bus)
        local specialUsers=$(ssh $server cat /etc/passwd \
        | awk '($3 < 1000 || $1 == "nobody"){print $1}' FS=":" | tr "\n" " ")
        local specialGroups=$(ssh $server cat /etc/group \
        | awk '($3 < 1000 || $1 == "nobody"){print $1}' FS=":" | tr "\n" " ")

        ssh $server cat /etc/passwd \
        | awk '($3 < 1000 || $1 == "nobody"){print $0}' FS=":" > $tmpPasswd

        ssh $server cat /etc/shadow \
        | awk '(NR==1){n=split(z,zs," "); for(x in zs){zss[zs[x]]=1}} ($1 in zss){print $0}', \
        FS=":" z="$specialUsers" > $tmpShadow

        ssh $server cat /etc/group \
        | awk '($3 < 1000 || $1 == "nobody"){print $0}' FS=":" > $tmpGroup

```

```

ssh $server cat /etc/gshadow \
| awk '(NR==1){n=split(z,zs," "); for(x in zs){zss[zs[x]]=1}} ($1 in zss){print $0}' \
FS=: z="$specialGroups" > $tmpGShadow


# Get user information that are 'human' from master: UID>=1000
local users=$(cat /etc/passwd \
| awk '($3>=1000 && $1 != "nobody"){print $1}' FS=: | tr "\n" " ")
local groups=$(cat /etc/group \
| awk '($3>=1000 && $1 != "nobody"){print $1}' FS=: | tr "\n" " ")

# Change home directory of 'human' user of each server
for user in $users; do
    userhome=$(cat /etc/passwd | awk -v u=$user '($1 == u){print $6}' FS=:)

    # Disk other than home disk, home directory will be /
    userdisk=$(echo $userhome | cut -d "/" -f3 \
               | awk '{print substr($1, 4, 1)})')
    if [[ $server == "disk"* && $server != $userdisk ]]; then
        userhome="/"
    fi
    cat /etc/passwd \
    | awk -v home=$userhome -v u=$user '($1 == u){print $1,$2,$3,$4,$5,home,$7}' \
    FS=: OFS=: >> $tmpPasswd
done
cat /etc/shadow \
| awk '(NR==1){n=split(z,zs," "); for(x in zs){zss[zs[x]]=1}} ($1 in zss){print $0}' \
FS=: z="$users" >> $tmpShadow

cat /etc/group | awk '($3 >= 1000 && $1 != "nobody"){print $0}' FS=: >> $tmpGroup

cat /etc/gshadow \
| awk '(NR==1){n=split(z,zs," "); for(x in zs){zss[zs[x]]=1}} ($1 in zss){print $0}' \
FS=: z="$groups" >> $tmpGShadow

# Override every user information to each server
scp -q $tmpPasswd $server:/etc/passwd
scp -q $tmpShadow $server:/etc/shadow
scp -q $tmpGroup $server:/etc/group
scp -q $tmpGShadow $server:/etc/gshadow

# Remove temporary files
rm $tmpPasswd $tmpShadow $tmpGroup $tmpGShadow
}

```

```

for server in $servers; do
    s=$(awk -v temp=$server '$(0 ~ temp && $1 !~ "#") \
        {printf("%s", substr($2, 2))}' /etc/dnsmasq.conf)
    if [ $s ]; then
        echo "updating" $server
        work $server
    else
        echo "$server does not exits. skipped"
    fi
    echo ""
done

# Print final messages
echo "Done"

```

## UpdateSystemFilesOnNodes

마스터의 시스템 파일 `/etc`, `/var/lib` 디렉토리들을 os가 없는 노드들에 복사해 준다. 다만 `NodewiseETC.txt` 와 `NodewiseVARLIB.txt`에 등록되어 있는 경로들은 마스터와 설정이 다를 수 있기 때문에 제외한다.

```

#!/root/admin/UpdateSystemFilesOnNodes.sh

#!/bin/bash

# This script updates system files(/etc, /var/lib) of nodes based on master

set -e          # Exit immediately if a command exits with a non-zero status
source "./Common.sh" # tftpboot, gaurantee this is master/root


# Args
prefix=$1      # ex) tenet
fromNode=$2     # ex) 10
toNode=$3       # ex) 14
if [ $# != 3 ]; then
    echo "Error: You should provide three arguments."
    echo "ex) tenet 10 14"
    exit
fi

# Double check
echo "Update system files on $prefix$fromNode ~ $prefix$toNode"

```

```

echo ""
echo -n "Is it ok? (y/n) : "
read -e isOK
if [ "x$isOK" != "xy" ]; then
    exit
fi

# Update kernal and ramdisk in case you forget doing so
cp /boot/vmlinuz-linux $tftpboot
mkinitcpio -p tenet

# Copy to nodes using rsync
# Options used in rsync
# -r: recursively
# -t: preserve modification time
# -l: copy symlinks as symlinks
# -H: preserve hard links
# -p: preserve permissions
# -X: preserve extended attributes, associated with inode number
# -D: preserve device and special files
# --del: delete extraneous file from the receiver
# --exclude-from=FILE: exclude pattern in file
n=$fromNode
while [ $n -le $toNode ]; do
    nodename=${prefix}${n}
    croot=$tftpboot/${nodename}
    if [ -e $croot ]; then
        echo "Updating ${nodename}"
        rsync -rtlHPXD --del --exclude-from=NodewiseETC.txt /etc $croot
        rsync -rtlHPXD --del --exclude-from=NodewiseVARLIB.txt /var/lib $croot/var
    else
        echo "${nodename} does not exist. skipped."
    fi
    n=$((n + 1))
done

# Print final messages
echo ""

```

```
echo "Done"

/root/admin/NodewiseETC.txt

# Directories should end with /
/etc/conf.d/
/etc/cron*
/etc/dnsmasq*
/etc/exports
/etc/fstab
/etc/hosts
/etc/hostname
/etc/netctl/
/etc/ntp.conf
/etc/ssh/
/etc/systemd/
/etc/udev/rules.d/
```

```
/root/admin/NodewiseVARLIB.txt

# Directories should end with /
/var/lib/dhcpcd/
/var/lib/misc/
/var/lib/nfs/
/var/lib/ntp/
/var/lib/pacman/
/var/lib/systemd/
```

## 6.4 Custom Services

이 절에서는 4.9에서 소개된 `/etc/systemd/system`에 있는 서비스 유닛과 `/root/service`에 있는 실행파일들의 본문 내용을 서술한다. 각 script들이 어떻게 작성되어 있는지 알고 싶거나 수정이 필요할 경우, 해당하는 내용을 보면 공부하도록 하자.

### ExternalNetwork-terminal

터미널 노드는 os가 존재하지 않기 때문에, 3.3에서 서술한 것처럼 `netctl` 등으로 고정 ip를 설정하기 쉽지 않다. 따라서 terminal이 부팅되면서 실행되는 서비스를 사용해 `ip route` 와 `ip add`를 포함하는 스크립트를 실행하도록 설정한다. 코드에 대한 이해는 3.3.3절의 설명을 참고하자.

```
/tftpboot/terminal2/etc/systemd/system/ExternalNetwork-terminal.service

# To install this service, copy this file to /etc/systemd/system
# Then run 'systemctl enable ExternalNetwork-terminal'.

[Unit]
Description=External network configuration
After=remote-fs.target
Before=sshd.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/root/service/ExternalNetwork-terminal.sh

[Install]
WantedBy=multi-user.target
```

```
/root/service/ExternalNetwork-terminal.sh

#!/bin/bash

# Activate NIC
ip link set external up
ip route del default

# Add static IP route and external address according to each terminal
case $HOSTNAME in
    terminal2)
        ip route add default via 147.47.xxx.xxx dev external onlink
        ip addr add 147.47.51.21 dev external
        ;;
esac
```

## Firewall-Blacklist-update

**Firewall-Blacklist-update.timer** 유닛을 통해 1주일마다 (매주 월요일 12:00 am) **Firewall-Blacklist-update.service** 유닛을 트리거 해 주어 **Firewall-Blaklist-udpate.sh**를 실행한다. 해당 script는 마스터와 터미널의 journal 기록을 확인하고, 100번 이상 로그인 실패한 ip 주소를 **/root/service/Firewall-Blacklist.txt**에 추가해 준다.

```
/etc/systemd/system/Firewall-Blacklist-update.timer

# To install this service, copy this file to /etc/systemd/system
# Then run 'systemctl enable Firewall-Blacklist-update.timer'.

[Unit]
Description=Update Firewall Blacklist

[Timer]
OnCalendar=weekly
Persistent=true

[Install]
WantedBy=timers.target
```

```
/etc/systemd/system/Firewall-Blacklist-update.service

# To install this service, copy this file to /etc/systemd/system
# Then run 'systemctl enable Firewall-Blacklist-update.service'.

[Unit]
Description=Update Firewall Blacklist
After=network.target

[Service]
Type=simple
ExecStart=/root/service/Firewall-Blacklist-update.sh
```

```
/root/service/Firewall-Blacklist-update.sh

#!/bin/bash

# Scan journal from master and terminals
journalctl -b > temp_black.txt
for i in {2..4}; do
    ssh terminal${i} journalctl -b >> temp_black.txt
done

# Take ip address failed to login 100+ times
cat temp_black.txt \
| /bin/grep "Failed password for invalid" \
| /bin/grep -Eo '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}'\
```

```

| awk '{a[$1]++}END{for(m in a){if(a[m]>100){print m}}}' \
>> "/root/service/Firewall-Blacklist.txt"
rm temp_black.txt

# Restart Firewall and Firewall-terminal
systemctl restart Firewall.service
for i in {2..4}; do
    ssh terminal${i} systemctl restart Firewall-terminal.service
done

```

## Firewall

**Firewall**은 내부망의 접속을 허용하고 외부망의 접속을 차단하는 것을 기본으로 한다. 추가적으로 black list의 ip들을 차단한다. 혹시 모를 상황에 대비하여 ssh 접속을 위한 56번 포트를 열어둘 수 있는 명령어를 comment out 해 놓은 상태이다.

```

/etc/systemd/system/Firewall.service

# To install this service, copy this file to /etc/systemd/system
# Then run 'systemctl enable Firewall'.

[Unit]
Description=Firewall
After=network.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/root/service/Firewall.sh start
ExecStop=/root/service/Firewall.sh stop

[Install]
WantedBy=multi-user.target

```

```

/root/service/Firewall.sh

#!/bin/bash

blacklist="/root/service/Firewall-Blacklist.txt"

function clearRules {
    # Declare empty table in case there is no such table
    nft table ip filter

```

```

nft table ip nat

# Clear any existing firewall stuff
nft delete table ip filter
nft delete table ip nat

# Allow everything
nft add table ip filter
nft add chain ip filter INPUT '{type filter hook input priority 0; policy accept;}' 
nft add chain ip filter FORWARD '{type filter hook forward priority 0; policy accept;}' 
nft add chain ip filter OUTPUT '{type filter hook output priority 0; policy accept;}' 

nft add table ip nat
}

function setRules {
    # Clear any existing firewall stuff before we start
    nft delete table ip filter
    nft delete table ip nat

    # As the default policies, drop all incoming traffic but allow all outgoing traffic.
    nft add table ip nat
    nft add chain ip nat POSTROUTING '{type nat hook postrouting priority 100;}' 

    nft add table ip filter
    nft add chain ip filter INPUT '{type filter hook input priority 0; policy drop;}' 
    nft add chain ip filter FORWARD '{type filter hook forward priority 0; policy drop;}' 
    nft add chain ip filter OUTPUT '{type filter hook output priority 0; policy accept;}' 

    # Allow incoming traffic through the connections initiated by this machine.
    nft add rule ip filter INPUT iifname "external" ct state related,established counter accept

    # Allow all incoming traffic if it is coming from the local loopback device
    nft add rule ip filter INPUT iifname "lo" counter accept

    # Allow all incoming traffic if it is coming from the internal network
    nft add rule ip filter INPUT iifname "internal" counter accept

    # Allow connections on SSH ports to the firewalled computer: port 56
    # nft add rule ip filter INPUT iifname "external" tcp dport 56 ct state new counter accept

    # Allow connections on Mathematica license server ports to the firewalled computer:
    # nft add rule ip filter INPUT iifname "external" tcp dport 16286 ct state new counter accept

    # Drop incoming connections from ips in the blacklist
}

```

```

if [[ -f "$blacklist" ]]; then
    while read ip; do
        nft insert rule ip filter INPUT ip saddr $ip counter drop
    done < "$blacklist"
fi
}

case "$1" in
start)
    setRules
    ;;
stop)
    clearRules
    ;;
*)
    echo "Invalid argument : $1"
    exit 1
esac

```

## Firewall-terminal

터미널에서 작동하는 **Firewall**이다. 주목할 점은 **Firewall-terminal.service** 유닛의 위치이며, start와 stop의 configuration 파일까지 있어야 한다. 다음은 terminal2의 경우에 대한 예시이다.

```

/tftpboot/terminal2/etc/systemd/system/Firewall-terminal.service

# To install this service, copy this file to /etc/systemd/system
# Then run 'systemctl enable Firewall-terminal' at terminal node.

[Unit]
Description=Firewall
After=remote-fs.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/root/service/Firewall-terminal.sh start
ExecStop=/root/service/Firewall-terminal.sh stop

[Install]
WantedBy=multi-user.target

```

```

/root/service/Firewall-terminal.sh

#!/bin/bash

blacklist="/root/service/Firewall-Blacklist.txt"

function clearRules {
    # Declare empty table in case there is no such table
    nft table ip filter
    nft table ip nat
    nft -f /root/service/Firewall-terminal.stop.nft
}

function setRules {
    # Declare empty table in case there is no such table
    nft table ip filter
    nft table ip nat

    # Clear any existing firewall stuff before we start
    nft delete table ip filter
    nft delete table ip nat
    nft delete table ip mangle

    # Load iptables configurations
    nft -f /root/service/Firewall-terminal.start.nft

    # Allow connections on SSH ports to the firewalled computer: port 56
    nft add rule ip filter INPUT iifname "external" tcp dport 56 ct state new counter accept

    # Open firewall for port-forwarding
    # Enable port forwarding
    nft add rule ip nat POSTROUTING oifname "external" masquerade
    # Add firewall restriction
    nft add rule ip filter FORWARD ct state related,established accept
    nft add rule ip filter FORWARD iifname "internal" oifname "external" accept

    # Drop incoming connections from ips in the blacklist
    if [[ -f "$blacklist" ]]; then
        while read ip; do
            nft insert rule ip filter INPUT ip saddr $ip counter drop
            done < "$blacklist"
        fi
    }

    case "$1" in
        start)

```

```

        setRules
        ;;
stop)
        clearRules
        ;;
*)
        echo "Invalid argument : $1"
        exit 1
esac

```

```

/root/service/Firewall-terminal.start.nft

# Translated by iptables-restore-translate v1.8.7 on Thu Aug 12 19:52:15 2021
add table ip mangle
add chain ip mangle PREROUTING {type filter hook prerouting priority -150; policy accept;}
add chain ip mangle INPUT {type filter hook input priority -150; policy accept;}
add chain ip mangle FORWARD {type filter hook forward priority -150; policy accept;}
add chain ip mangle OUTPUT {type route hook output priority -150; policy accept;}
add chain ip mangle POSTROUTING {type filter hook postrouting priority -150; policy accept;}
add table ip nat
add chain ip nat PREROUTING {type nat hook prerouting priority -100; policy accept;}
add chain ip nat INPUT {type nat hook input priority 100; policy accept;}
add chain ip nat OUTPUT {type nat hook output priority -100; policy accept;}
add chain ip nat POSTROUTING {type nat hook postrouting priority 100; policy accept;}
add table ip filter
add chain ip filter INPUT {type filter hook input priority 0; policy drop;}
add chain ip filter FORWARD {type filter hook forward priority 0; policy drop;}
add chain ip filter OUTPUT {type filter hook output priority 0; policy accept;}
add rule ip filter INPUT iifname "external" ct state related,established counter accept
add rule ip filter INPUT iifname "lo" counter accept
add rule ip filter INPUT iifname "internal" counter accept
# add rule ip filter INPUT iifname "external" tcp dport 22 ct state new counter accept
# Completed on Thu Aug 12 19:52:15 2021

```

```

/root/service/Firewall-terminal.stop.nft

# Translated by iptables-restore-translate v1.8.7 on Thu Aug 12 20:02:09 2021
add table ip mangle
add chain ip mangle PREROUTING {type filter hook prerouting priority -150; policy accept;}
add chain ip mangle INPUT {type filter hook input priority -150; policy accept;}
add chain ip mangle FORWARD {type filter hook forward priority -150; policy accept;}
add chain ip mangle OUTPUT {type route hook output priority -150; policy accept;}
add chain ip mangle POSTROUTING {type filter hook postrouting priority -150; policy accept;}
add table ip nat
add chain ip nat PREROUTING {type nat hook prerouting priority -100; policy accept;}
add chain ip nat INPUT {type nat hook input priority 100; policy accept;}

```

```
add chain ip nat OUTPUT {type nat hook output priority -100; policy accept;}
add chain ip nat POSTROUTING {type nat hook postrouting priority 100; policy accept;}
add table ip filter
add chain ip filter INPUT {type filter hook input priority 0; policy accept;}
add chain ip filter FORWARD {type filter hook forward priority 0; policy accept;}
add chain ip filter OUTPUT {type filter hook output priority 0; policy accept;}
# Completed on Thu Aug 12 20:02:09 2021
```