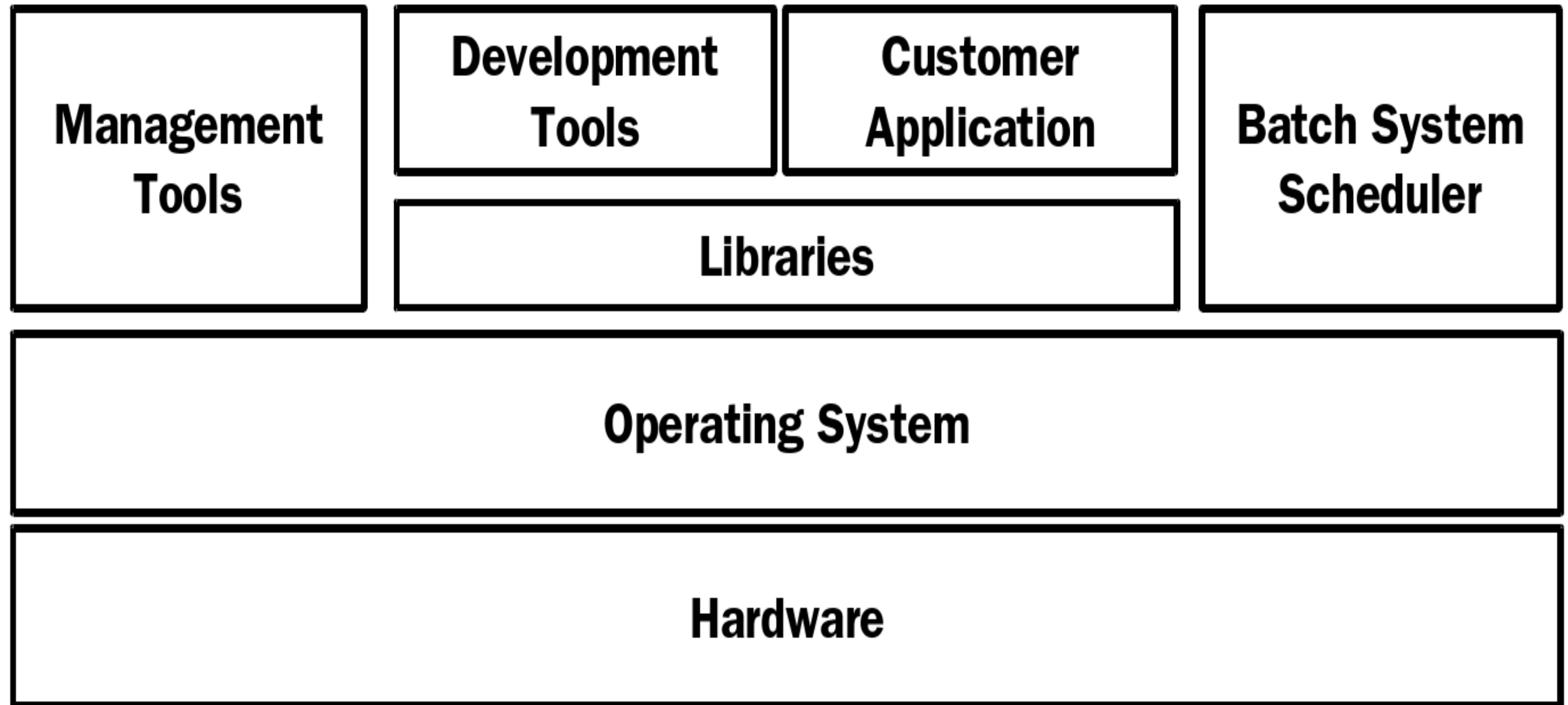


# 03 클러스터 구축

절차



### 3. 클러스터 구축



## 3. 클러스터 구축

### 클러스터 기본 구성

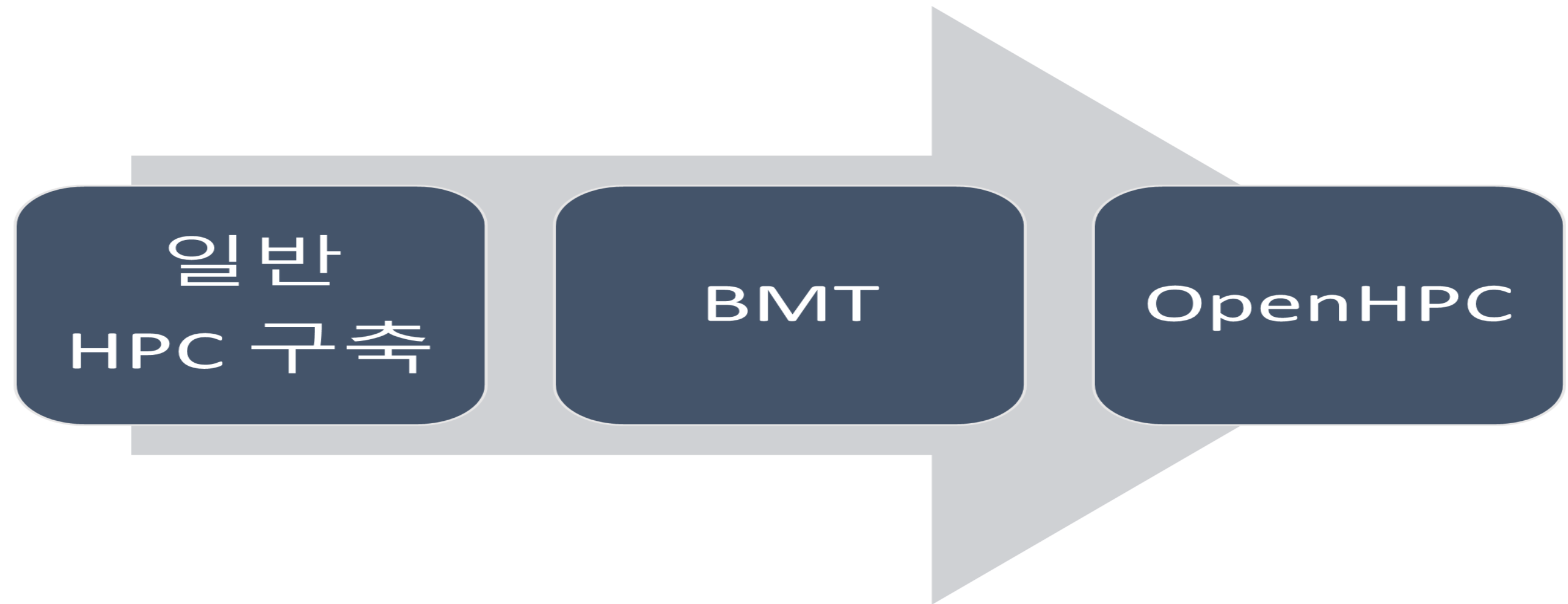
- 시스템에 대한 구상
  - 어떻게 이용할 것인가 ?
  - 몇 노드로 구성할 것인가 ?
  - 구성에 사용되어질 하드웨어들은 어떤 것들인가?
  - 구성할 시스템에 대한 비용적인 측면과 실제 사용 했을 때 효율면에서 적합한가 ?

특별한 목적(특히 연구,계산)으로 사용했을 때 비용면이 충분히 합당한 것인지를 예상

- 시스템 하드웨어정보 파악
  - 하드웨어 리소스 파악

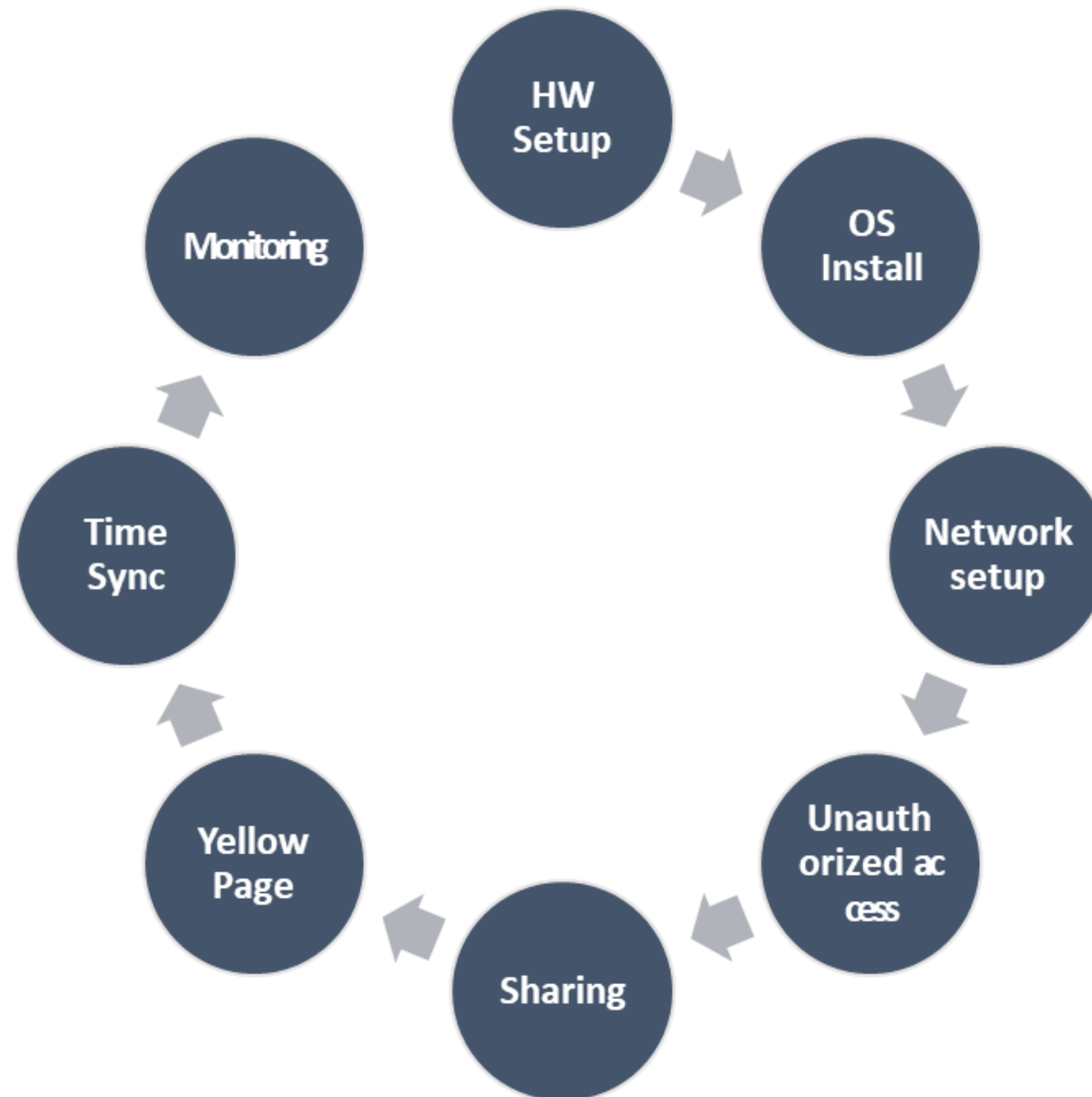
### 3. 클러스터 구축

#### 클러스터 기본 구성



### 3. 클러스터 구축

#### 클러스터 기본 구성



### 3. 클러스터 구축

#### 클러스터 기본 구성



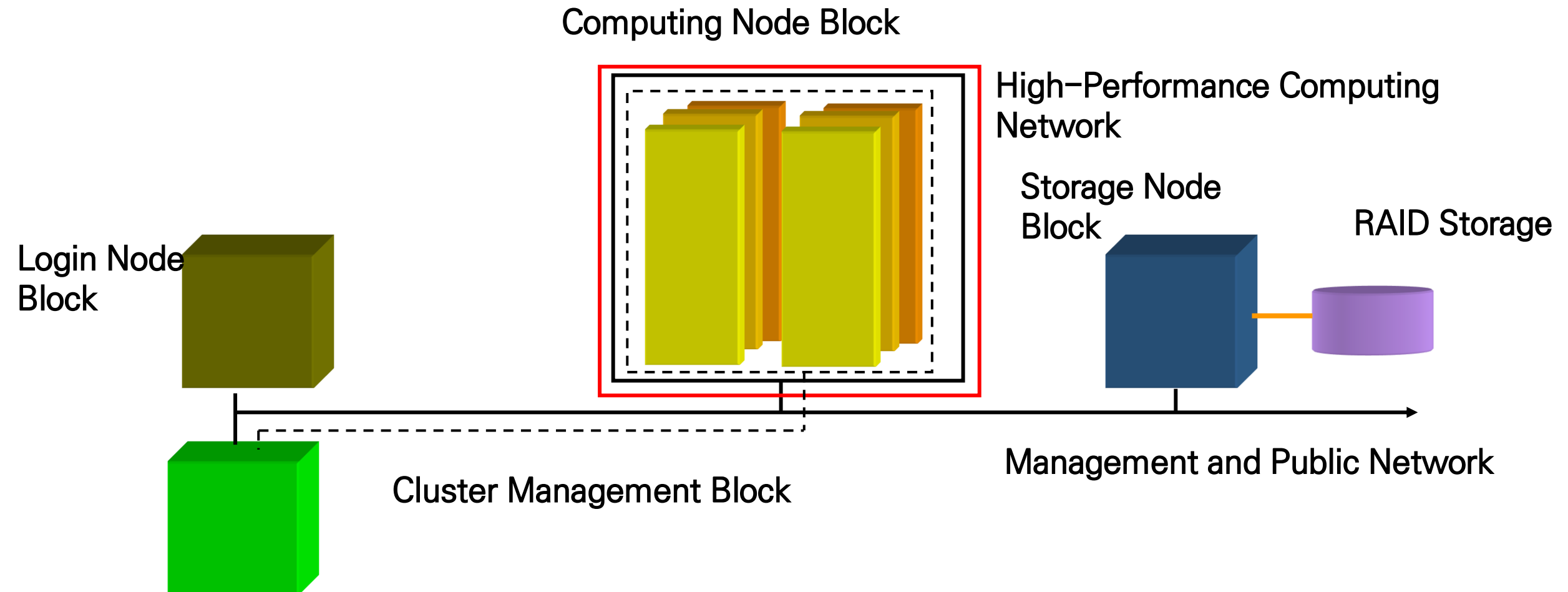
# 03 클러스터 구축

하드웨어 구성



### 3. 클러스터 구축

#### 클러스터 기본 구성



구 분	내 용
Management Node	Cluster 관리, 설치, 컴파일 등의 역할을 담당
Computing Node	실제 연산을 수행하는 노드로써 Cluster의 중심 역할
Storage Node	작업 결과 및 프로그램을 저장하는 Storage 서버
Login Node	사용자 계정 및 작업 분배 사용자 프로그램 컴파일 등의 역할을 담당
Replacement Node	Cluster Node 장비의 백업용 노드
Stage Node	대규모 Cluster 구성시 일정 노드 그룹당 하나씩 Management Node 외 별도의 서버를 두어 그룹 단위 관리를 담당

### 3. 클러스터 구축

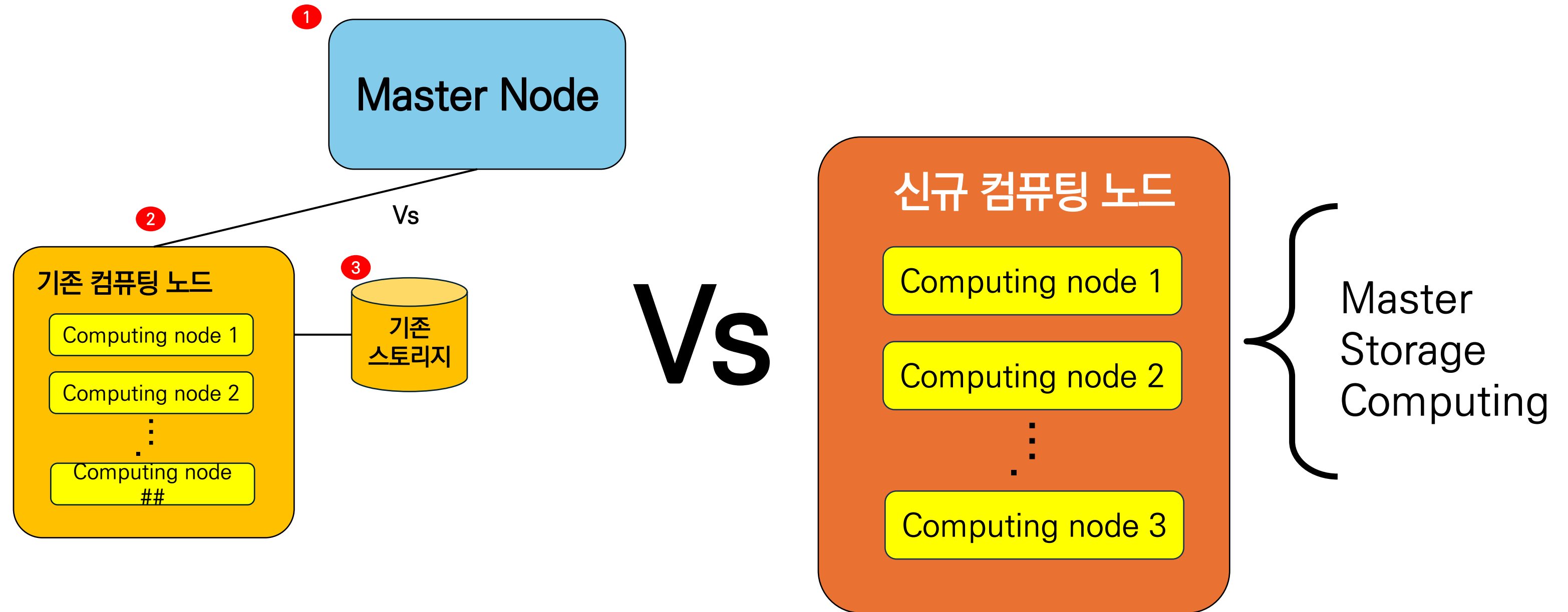
#### 클러스터 기본 구성

A	CCM+ 버전	파일명	Mesh 수	Physical Time	해석시간						변경 전후 차이	차이
					A사 HPC	타사		TeraTEC				
								VNC	Scheduler	BIOS 변경		
B	13.06	aaa	3,429,388	70초	11.07 시간	3.73 시간	3:43:48	6.29 시간	4:08:00	3:15:39	0:52:21	0:28:09
	15.06				-	3.61 시간	3:36:36					
	17.06				-	3.57 시간	3:34:12					
	13.06				4.22 시간	1.86 시간	1:51:36	4.00 시간	2:10:00	1:41:07	0:28:53	0:10:29
C	15.06	bbb	1,564,785	70초	-	1.84 시간	1:50:24					
	17.06				-	1.87 시간	1:52:12					
	13.06				5.01 시간	2.24 시간	2:14:24		2:57:00	2:02:09	0:54:51	0:12:15
D	15.06	ccc	1,434,738	90초	-	2.16 시간	2:09:36					
	17.06				-	2.19 시간	2:11:24					

Star-ccm		타사		TERATEC				차이	
				튜닝 전	튜닝 후 NVME	Disk	node 당 CPU 94CPU Total 282CPU		
13.06	1.86	1:51:36		1:40:48	1:31:37	1:33:45		0:19:59	
15.06	1.84	1:50:24		1:45:00	1:36:33	1:36:44		0:13:51	
17.06	1.87	1:52:12		2:04:12	1:47:33	1:50:22		0:04:39	
					1:48:10	1:50:08		0:04:02	
19.02				1:55:48	1:40:42				
13.06	1.86	1:51:36				1:30:00	node01:94 node02:96 node03:96	0:21:36	

### 3. 클러스터 구축

#### 클러스터 기본 구성



### 3. 클러스터 구축

항 공

하드 디스크 보다 계산을 메모리에 처리하는 코드가 많음  
고속 CPU  
고속 I/O  
네트워크 성능에 민감

기 계

- Explicit 문제  
CPU 성능과 통신 속도 중요
- Implicit 문제  
대용량 메모리  
고속 I/O 필요

물 리

몬테카를로 코드 : 네트워크 성능에 민감

화 학

- Molecular Dynamics  
CPU 성능과 통신 속도 중요  
메모리 크기, I/O 용량 및 속도에 덜 민감
- Quantum Dynamics  
CPU 성능 통신 속도 메모리 크기 모두 중요

생 명

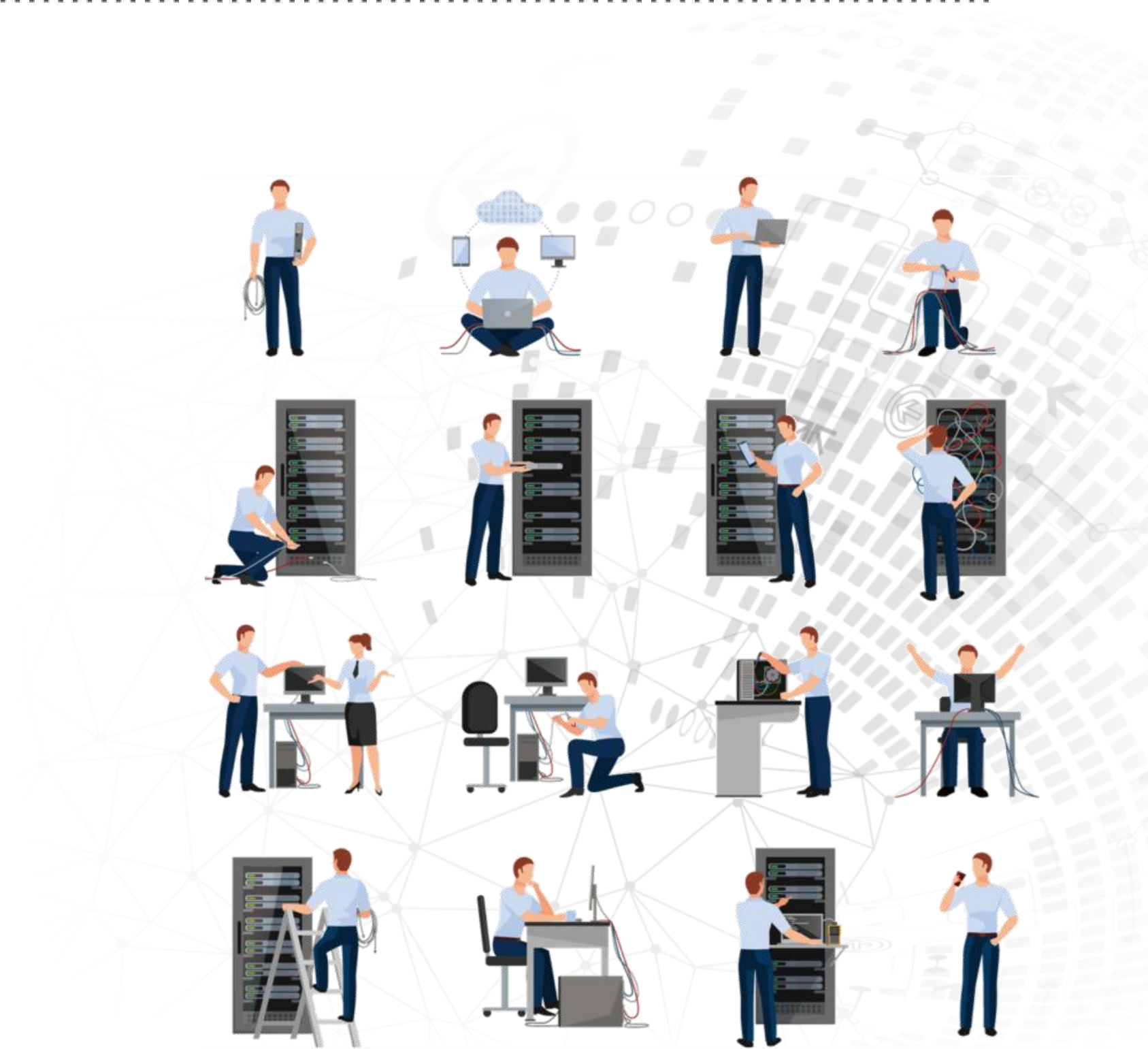
Protein folding  
고속의 CPU 성능과 메모리 크기 다소 중요  
BLAST

천 문

고속 CPU, 네트워크 성능에 민감 ( 전처리 후 처리시 상단 한 영향을 줌)

# 03 클러스터 구축

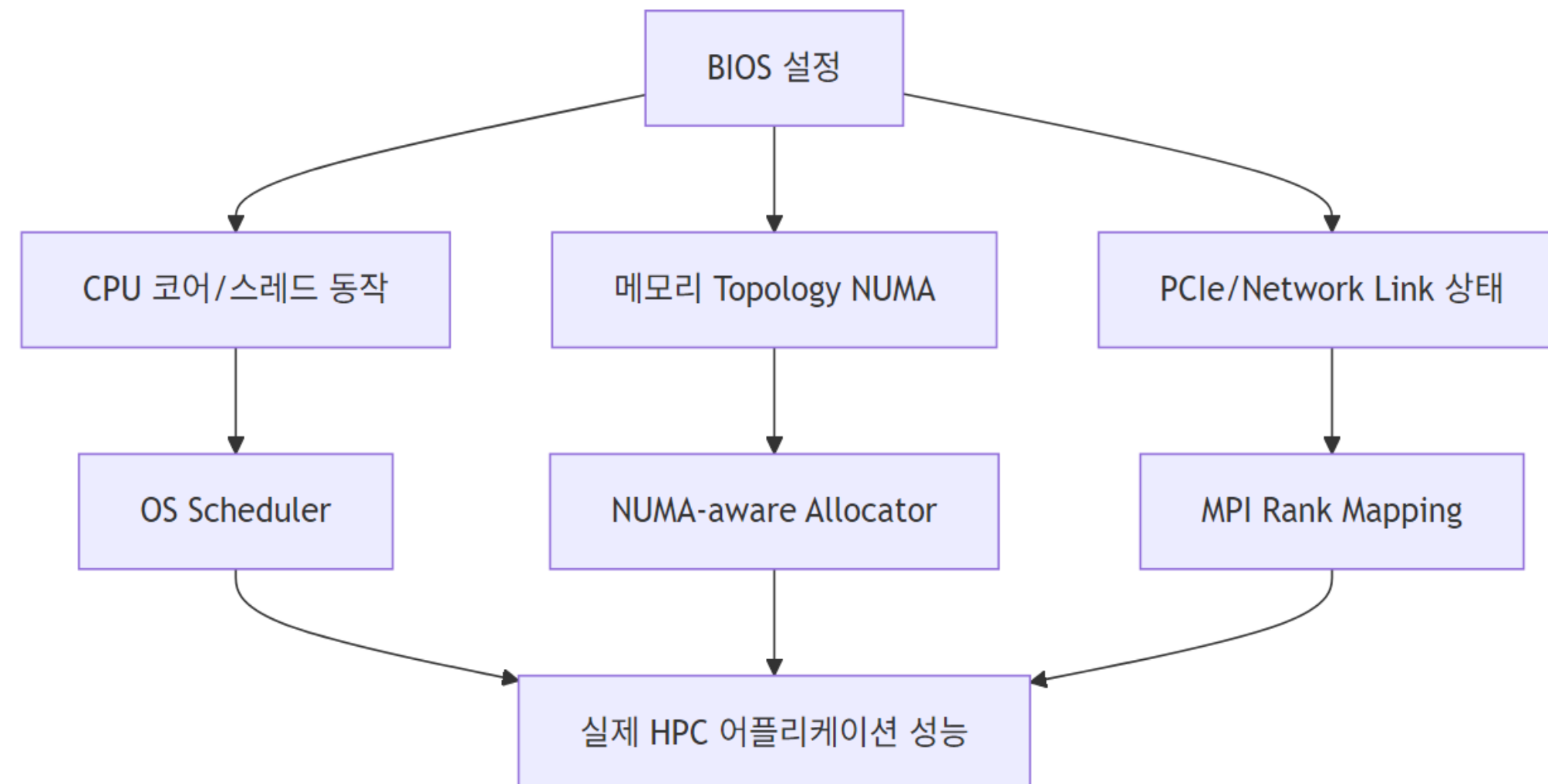
BIOS 구성



### 3. 클러스터 구축

#### HPC에서 BIOS 값이 중요한 이유

- HPC 클러스터는 ‘CPU ↔ Memory ↔ I/O ↔ Network’ 경로의 지연(latency)과 대역폭(bandwidth)을 극한까지 끌어올려야 한다.
- BIOS(UEFI)는 CPU/메모리/PCIe/전력/NUMA 등의 저수준 하드웨어 동작 모드를 직접 제어한다. 즉, OS나 MPI, Slurm 같은 HPC 소프트웨어는 이 BIOS 설정값 위에서만 최적 동작이 가능해진다.



### 3. 클러스터 구축

#### 대표 BIOS 값과 HW ↔ SW 연관성

BIOS 항목	HW 작동 방식	SW 영향	실제 영향
SMT/HT (스레드)	코어별 추가 논리 스레드 제공	OS 스케줄러가 스레드로 태스크 분배	일부 워크로드(벡터 연산)에서 L1/L2 캐시 경합으로 역효과
C-States	코어가 Idle시 클럭 다운	HPC 워크로드는 높은 클럭 유지 필요	불필요한 Idle → Wakeup 지연으로 마이크로초 레벨 지연
P-States	DVFS로 클럭 변화	클러스터 전체 코어 동기화 성능	Linpack 등에서 성능 변동 폭 발생
NUMA 모드	메모리 접근 Topology	MPI Rank Mapping, numa-bind	Local Node ↔ Remote Node 이동 시 메모리 레이턴시 2배 이상 차이
Fabric Link Speed	소켓간 링크 대역폭	MPI All-to-All/Allreduce	HPCG, STREAM에서 낮은 성능 병목
PCIe ASPM	PCIe Power Saving	High-speed NIC, GPU DMA	네트워크 패킷 Drop/지연 발생

### 3. 클러스터 구축

#### 공통 HPC BIOS 튜닝 원칙

항목	기본 권장값
Hyper-Threading/SMT	OFF (L1~L2 캐시 경합 방지)
C-States (Idle States)	OFF (코어가 최대 주파수 유지)
P-States (Performance States)	최대 성능 고정 (OS가 DVFS를 해제하도록)
NUMA 노드당 Memory Interleave	OFF (NUMA-aware 스케줄링과 일관성 유지)
Memory Patrol Scrubbing	ON or Balanced (메모리 오류 정기 검사, HPC에 따라 OFF 권장하기도 함)
PCIe ASPM (Active State Power Management)	OFF (저지연 네트워킹용)
SR-IOV/Virtualization	필요한 경우에만 ON
LLC Prefetcher, Adjacent Cache Line Prefetch	ON (대부분 HPC 코드에서 긍정적 효과)
UPI/Infinity Fabric Link Speed	최대값으로 강제
Fan Profile / Thermal	Performance Mode

### 3. 클러스터 구축

#### Intel Xeon 5세대 (Granite Rapids) HPC 최적 BIOS 값 예시

BIOS 항목	권장 설정	이유
Hyper-Threading	OFF	HPC 벤치마크(Linpack, HPCG 등)에서 낮은 지연
C-State	C0/C1 Only	Turbo 유지
P-State	OS Control → DisabledPerformance Mode	고정 클럭
UPI Link Speed	Max	소켓간 대역폭 극대화
LLC Prefetch	Enabled	캐시 효율
Sub-NUMA Clustering (SNC)	SNC-2 or SNC-4	소켓당 NUMA Domain 분리(메모리 접근 최적화)
X2APIC Mode	Enabled	대규모 Core ID 관리
Memory Interleave	NUMA	NUMA-aware 배치 일관성
Patrol Scrubbing	Balanced	안정성과 성능 균형
Intel VT-d/IOAT DMA	필요한 경우만 ON	
UPI Power Management	OFF	최대 대역폭 유지
AVX512/AVX10	Auto	벡터 연산 코드 활용
LLC dead line allocation	Enabled	HPC 벤치마크에서 유리함

### 3. 클러스터 구축

#### AMD EPYC 5세대 (Turin) HPC 최적 BIOS 값 예시

BIOS 항목	권장 설정	이유
SMT Mode	OFF	코어당 리소스 집중
C-States	OFF	고정 주파수 유지
P-States	Static Max	DVFS 차단
Infinity Fabric Frequency	Max	IOD~CCD간 레이턴시 최소화
NUMA Mode	NPS=1 또는 NPS=4	HPC IO 패턴에 맞춤 (ex. NPS=4 → 4 NUMA per socket)
Memory Interleaving	Channel Interleave OFF, Die Interleave OFF	NUMA 최적화
Determinism Control	Performance	최대 Boost
Memory Power Down	Disabled	지연 감소
PCIe Power Management	Disabled	네트워크 대역폭 확보
LLC Prefetcher	Enabled	Cache Hit율 증가
SVM (Secure Virtual Machine)	필요한 경우만 ON	

### 3. 클러스터 구축

#### Intel vs AMD HPC BIOS 최적화 비교

비교 항목	Intel Xeon 5th (Granite Rapids)	AMD EPYC 5th (Turin)
NUMA 모드	SNC-2/SNC-4	NPS=1 ~ NPS=4
Mesh/Fabric Tuning	UPI Link Speed Max	Infinity Fabric Freq Max
SMT/HT	Hyper-Threading OFF	SMT OFF
메모리 Scrubbing	Balanced	Optional
Power States	P/C States OFF	P/C States OFF
Prefetch 설정	LLC + Adjacent Prefetch ON	LLC Prefetch ON
벤더 고유	X2APIC Mode ON UPI Power Mgmt OFF	Determinism Control=Performance

### 3. 클러스터 구축

HPC의 성능과 안정성은 BIOS(하드웨어 설정)가  
‘HW-OS-MPI-코드’ 전 구간의 토대가 되어 주기 때문에,  
**잘못된 BIOS는 어떤 소프트웨어 최적화도 무력화시킬 수 있다.**

# 03 클러스터 구축

OS 구성



### 3. 클러스터 구축

#### OS Support 확인

Table 2 Ansys Roadmap	2021		2022		2023		2024		2025		2026
Red Hat Enterprise Linux	R1	R2	R1	R2	R1	R2	R1	R2	R1	R2	R1
RHEL 6 Semiconductor applications only	✓										
RHEL 7.6 Enterprise	✓	✓									
RHEL 7.7 Enterprise	✓	✓	✓	✓							
RHEL 7.8 Enterprise	✓	✓	✓	✓	✓	✓	✓				
RHEL 7.9 Enterprise		✓	✓	✓	✓	✓	✓				
RHEL 8.1 Enterprise	✓	✓	✓	✓	✓						
RHEL 8.2 Enterprise		✓	✓	✓	✓						
RHEL 8.3 Enterprise		✓	✓	✓	✓	✓					
RHEL 8.4 Enterprise			✓	✓	✓	✓	✓				
RHEL 8.5 Enterprise				✓	✓	✓	✓	✓			
RHEL 8.6 Enterprise					✓	✓	✓	✓	✓		
RHEL 8.7 Enterprise					✓	✓	✓	✓	✓		
RHEL 8.8 Enterprise						✓**	✓	✓*	✓*	✓*	
RHEL 8.9 Enterprise								✓*	✓*	✓*	✓*
RHEL 8.10 Enterprise									✓*	✓*	✓*
RHEL 9.3 Enterprise								✓*	✓*	✓*	
RHEL 9.4 Enterprise									✓*	✓*	✓*
RHEL 9.5 Enterprise										✓*	✓*
RHEL 9.6 Enterprise											✓*
RHEL 9.7 Enterprise											

✓ Ansys Applications and License Manager

\* If feasible

\*\* post-release

### 3. 클러스터 구축

#### OS Support 확인

Table 6 Ansys Roadmap											
Linux:	2024		2025		2026		2027		2028		2029
Rocky Linux <sup>TM</sup>	R1	R2	R1	R2	R1	R2	R1	R2	R1	R2	R1
8.9 "Green Obsidian"		✓*	✓*	✓*	✓*						
8.10 "Green Obsidian"			✓*	✓*	✓*						
9.3 "Blue Onyx"		✓*	✓*	✓*							
9.4 "Blue Onyx"			✓*	✓*	✓*						
9.5 "Blue Onyx"				✓*	✓*						
9.6 "Blue Onyx"					✓*						
9.7 "Blue Onyx"					✓*						

✓ Ansys Applications and License Manager

\* If feasible

#### AMD Central Processing Unit Performance

The performance of AMD 64-bit x86 processors is enhanced with AMD Optimizing CPU Libraries (AOCL). To date the direct matrix solver in the HFSS product supports the AMD Zen4 AVX512 CPU, resulting in enhanced performance. Ansys products pending support of the AVX512 CPU include those shown in the table below.

	HFSS	Q3D	PSI	SIwave	Maxwell
AMD optimized	yes*	no	no	no	no
AVX512	no*	no	no	no	no

\* Support vector instruction sizes of up to 256

## 3. 클러스터 구축

### OS 설치

- 시스템에 대한 구상
  - 어떻게 이용할 것인가 ?
  - 몇 노드로 구성할 것인가 ?
  - 구성에 사용되어질 하드웨어들은 어떤 것들인가?
  - 구성할 시스템에 대한 비용적인 측면과 실제 사용 했을 때 효율면에서 적합한가 ?  
이 사항이 가장 중요하다.  
특별한 목적(특히 연구,계산)으로 사용했을 때 비용적인  
면이 충분히 합당한 것인지를 예상해야 할 것이다.
- 시스템 하드웨어정보 파악
  - 하드웨어 리소스 파악
- 파티션 설정
  - 관리의 편리함 구성에 단순화 등으로 파티션을 동일하게 하는 것이 좋다.

### 3. 클러스터 구축

#### OS 설치

상용 솔루션 ( Fluent, CFX, Star-CCM++....)	
Master	Cluster Node
GNOME Desktop	GNOME Desktop

In-house code (VASP, OpenFOAM....)	
Master	Cluster Node
Graphic Mode	Compute Mode

#### 추가 패키지 :

Development tools Package  
Compatibility Libraries Package  
Network File System Client Package  
Infiniband Support Package

#### 중요 :

Selinux Disable  
Firewall Disable

# 03 클러스터 구축

Cluster OS Setup



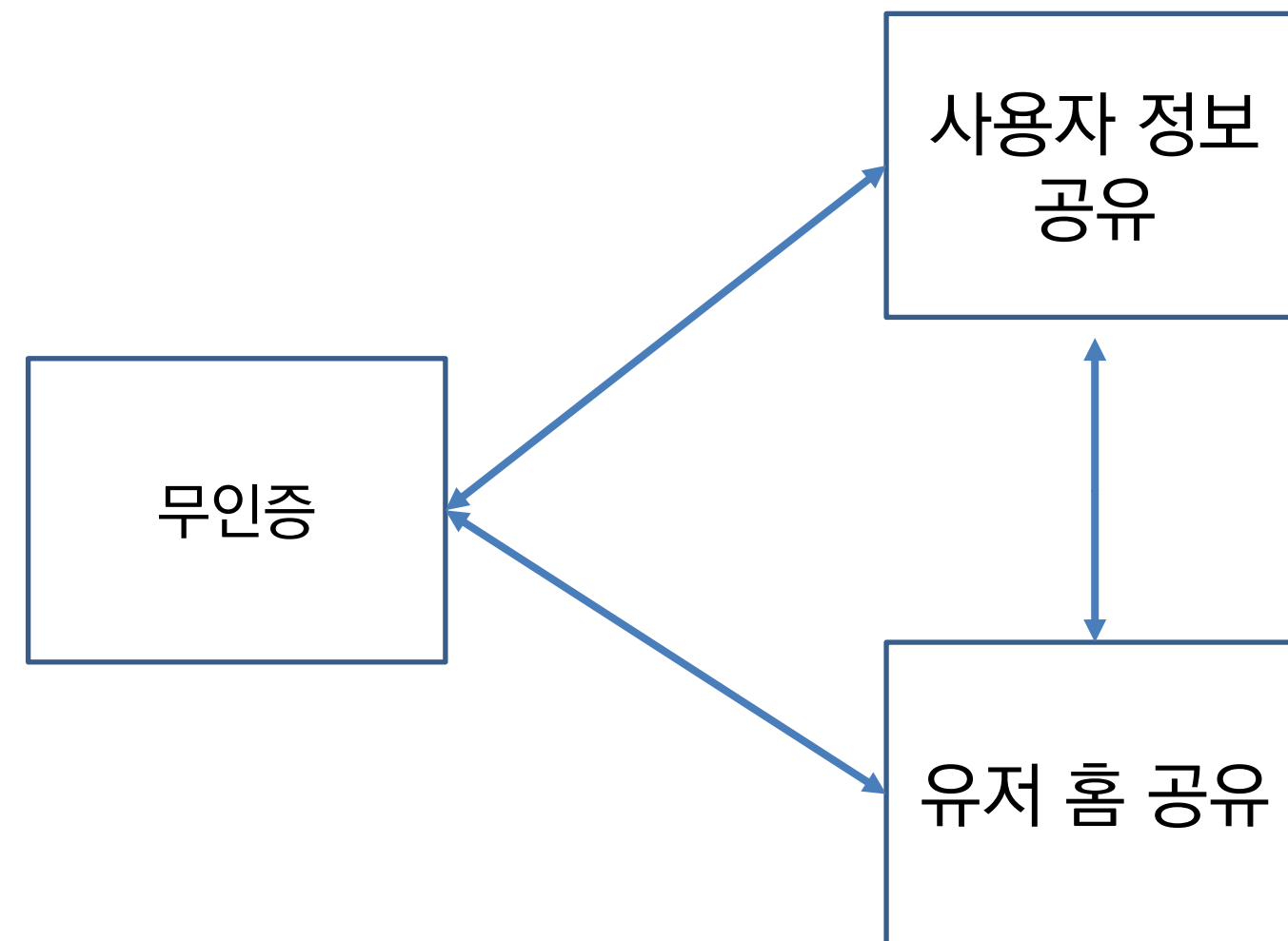
### 3. 클러스터 구축

#### Cluster Setup

- SSH 무비밀번호 접속
- 시간 동기화 (Chrony)
- NFS /home 공유
- 계정 공유 (NIS)
- 스케줄러
  - Munge (Slurm 인증)
  - Slurm 스케줄러
- Ganglia 모니터링

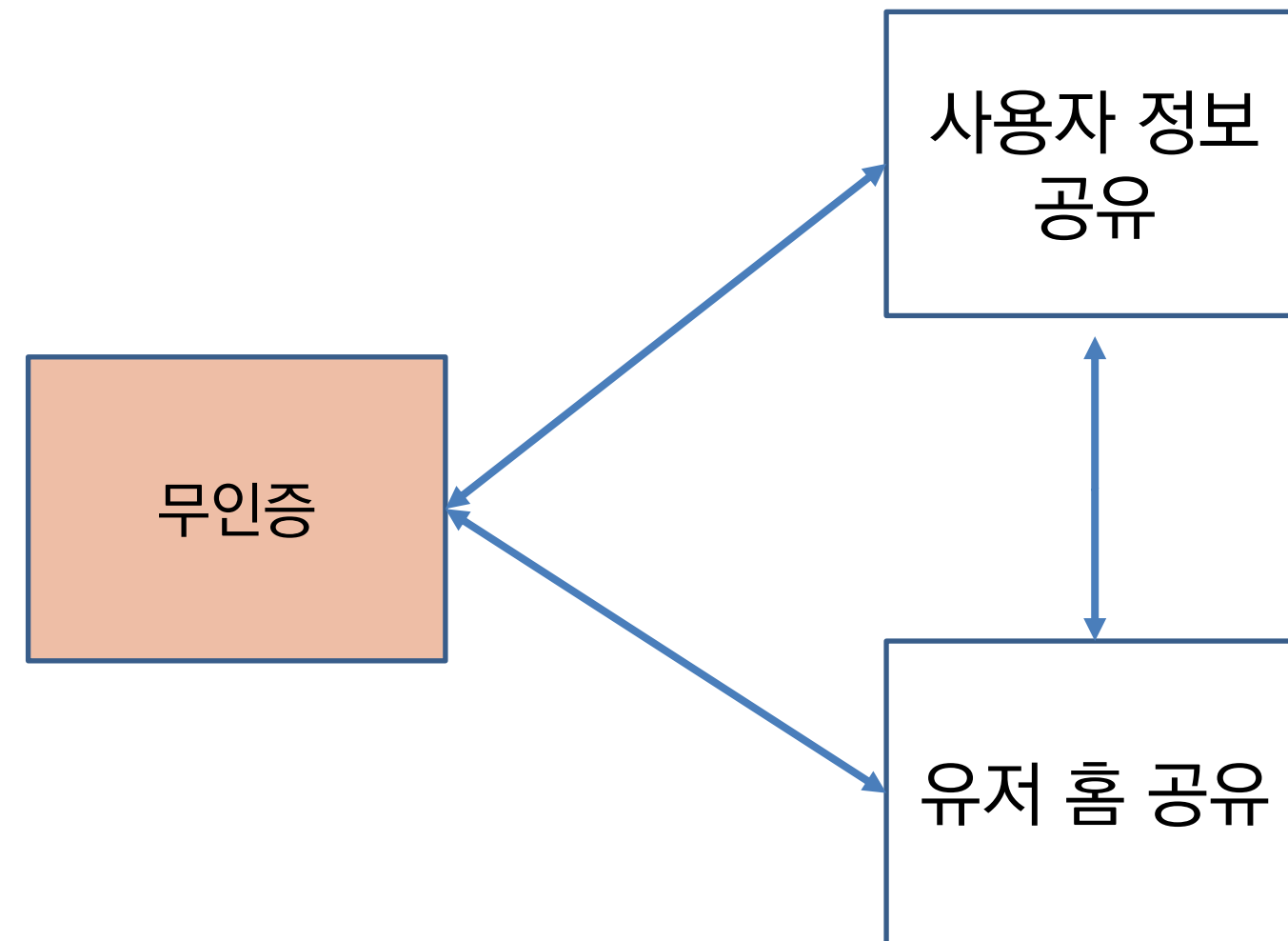
### 3. 클러스터 구축

#### Cluster Setup



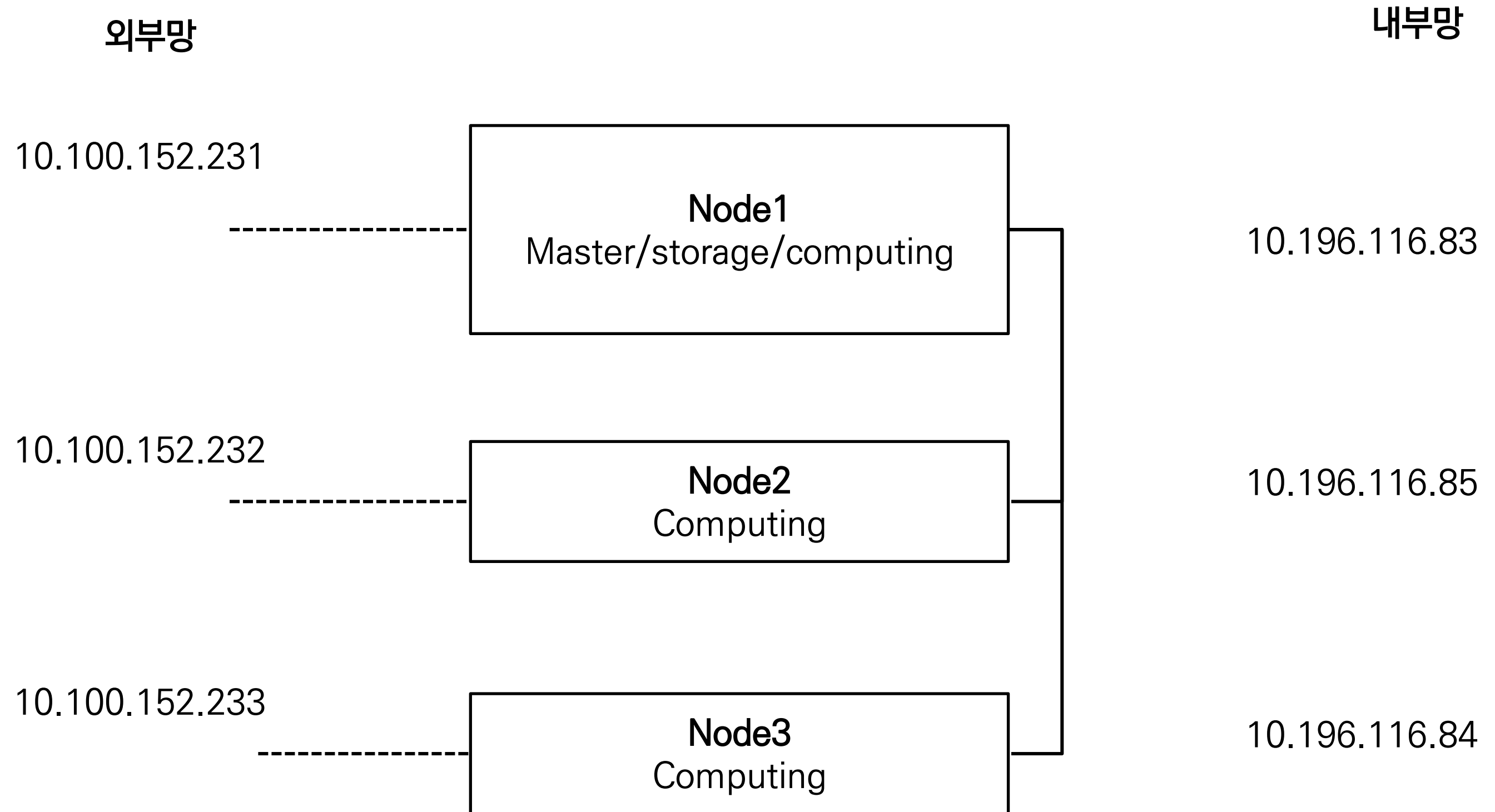
### 3. 클러스터 구축

#### Cluster Setup : ssh 무인증



### 3. 클러스터 구축

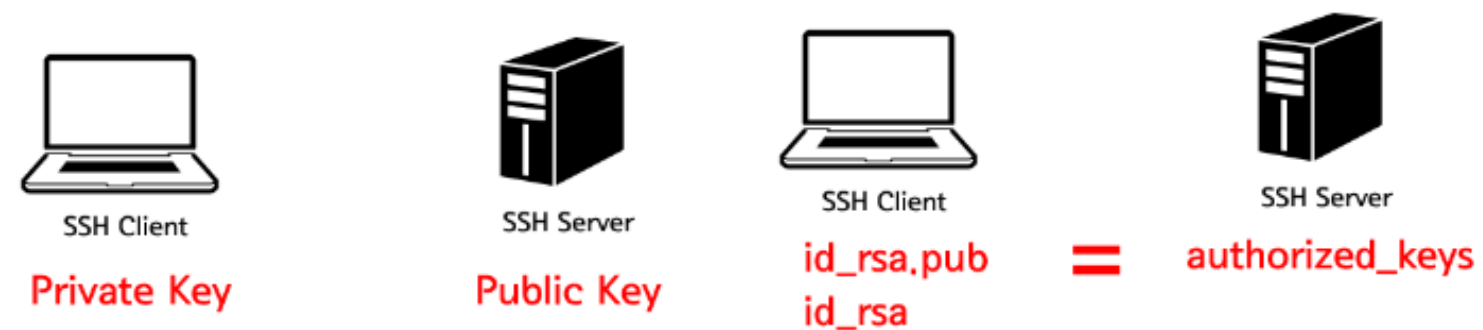
#### Cluster Setup : ssh 무인증



### 3. 클러스터 구축

#### Cluster Setup : ssh 무인증

- SSH Key는 공개키(public key)와 비공개 키(private key)로 이루어지는데 이 두개의 관계를 이해하는 것이 SSH Key를 이해하는데 핵심이다. 키를 생성하면 공개키와 비공개키가 만들어진다. 이 중에 비공개키는 로컬 머신에 위치해야 하고, 공개키는 리모트 머신에 위치해야 한다. (로컬 머신은 SSH Client, 원격 머신은 SSH Server가 설치된 컴퓨터를 의미한다.)
- SSH 접속을 시도하면 SSH Client가 로컬 머신의 비공개키와 원격 머신의 비공개키를 비교해서 둘이 일치하는지를 확인한다.



프로토콜	세대
RSA	1
DSA	2
ECDSA	3
ed25519	4

### 3. 클러스터 구축

#### Cluster Setup : ssh Key

파일	경로 (일반 사용자 기준)
개인키	~/.ssh/id_rsa
공개키	~/.ssh/id_rsa.pub
known_hosts	~/.ssh/known_hosts (접속한 서버 Fingerprint 저장)
authorized_keys	~/.ssh/authorized_keys (접속 허용 공개키 리스트)

### 3. 클러스터 구축

#### Cluster Setup : ssh 무인증

```
# create key-pair
```

```
[root@master ~]$ ssh-keygen
```

```
Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/root/.ssh/id_rsa):
```

```
Created directory '/root/.ssh'.
```

```
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
```

```
Your identification has been saved in /root/.ssh/id_rsa.
```

```
Your public key has been saved in /root/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
```

```
SHA256:TX8wjIjDBsrx7FWYGD buu/LV8T0RV+BoF/+Ju8WDkFU root@master
```

```
The key's randomart image is:
```

```
.....
```

```
[root@master ~]$ ll ~/.ssh
```

```
total 8
```

```
-rw----- . 1 rocky rocky 2655 Jul 16 19:04 id_rsa
```

```
-rw-r--r-- . 1 rocky rocky  573 Jul 16 19:04 id_rsa.pub
```

```
[root@root ~]$ cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
```



ssh localhost 테스트

### 3. 클러스터 구축

#### Cluster Setup : ssh 무인증

각 노드에 대하여 공개 키 copy 수행

```
ssh-copy-id -i ~/.ssh/id_rsa.pub root@111.222.333.444
```

### 3. 클러스터 구축

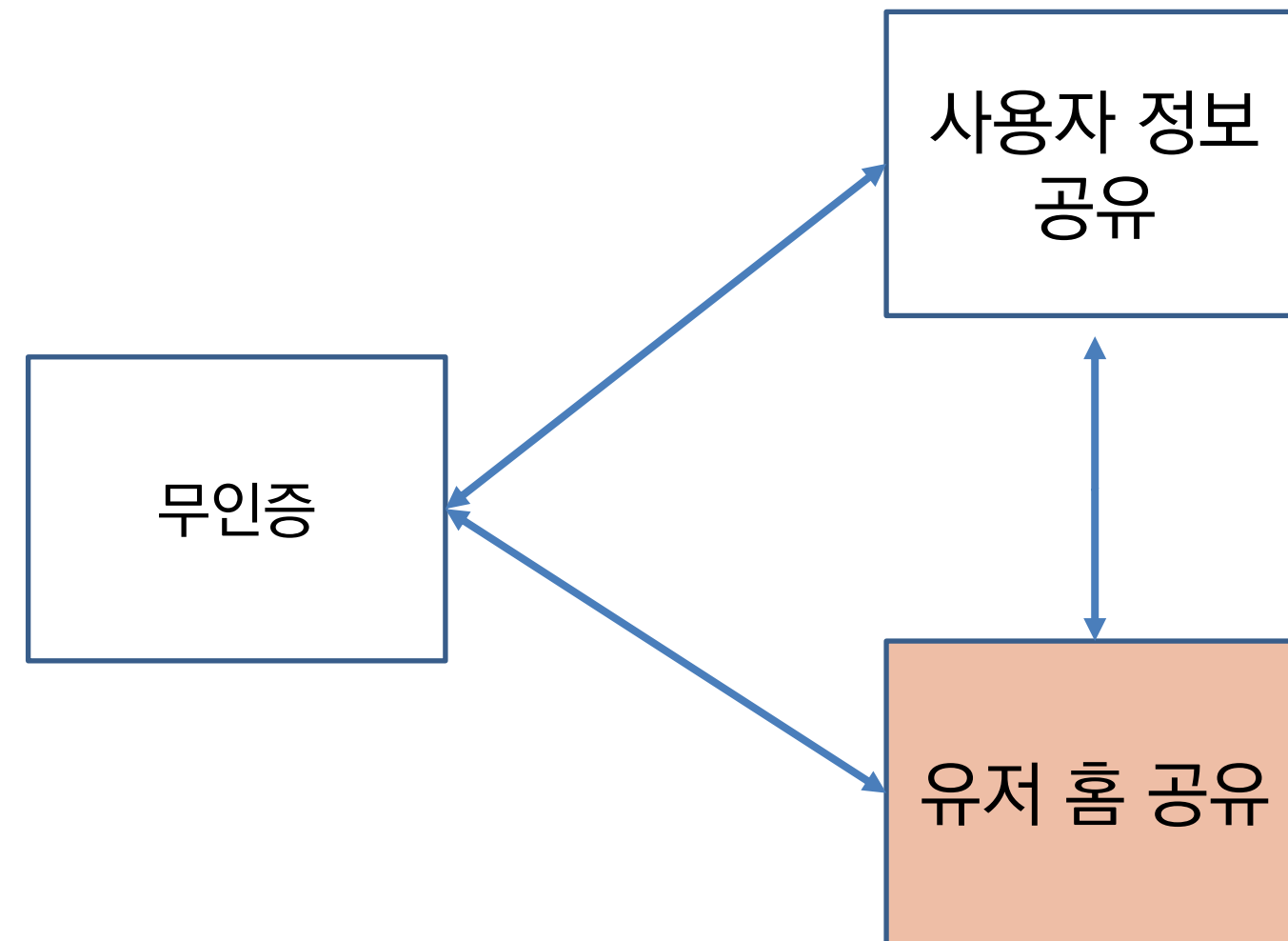
#### Cluster Setup – 일반 유저 무인증을 위한 ssh 무인증 스크립트 만들기

위치 : /etc/profile.d/sshless.sh

```
#!/bin/sh
#/usr/local/dtk/bin/chk_user_n_node
#[ "$?" == "1" ] && exit
##Passwordless ssh
if [ ! -e ~/.ssh/authorized_keys ]; then
    rm -f ~/.ssh/id_rsa* >& /dev/null
    ssh-keygen -t rsa -C " -N " -f ~/.ssh/id_rsa >& /dev/null
    cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys >& /dev/null
fi
#
```

### 3. 클러스터 구축

#### Cluster Setup : 공유 파일



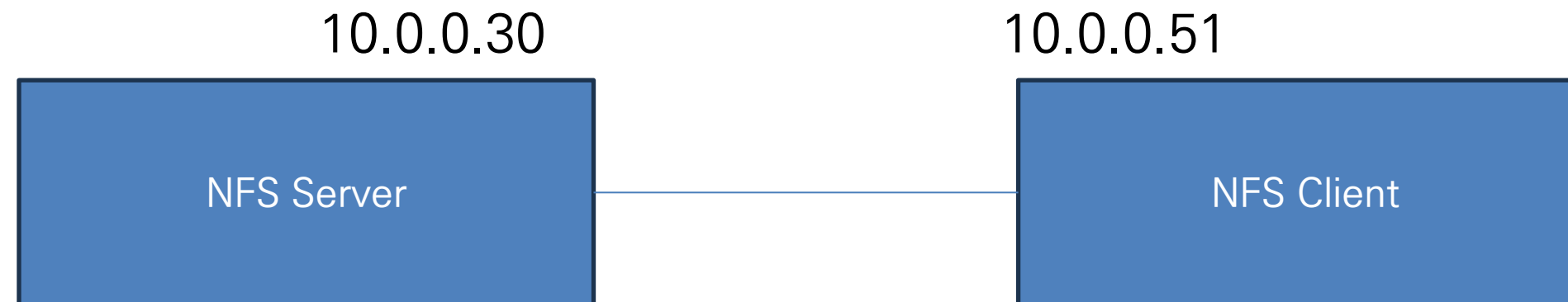
### 3. 클러스터 구축

#### Cluster Setup : 공유 파일

- 홈 노드 또는 별도의 스토리지 노드에 공유 폴더 지정
- NFS-Server
- NFS-Client

### 3. 클러스터 구축

#### Cluster Setup : 공유 파일



```
[root@storage ~]# dnf -y install nfs-utils
[root@storage ~]# vi /etc/exports
# create new
# for example, set [/home] as NFS share
/home/ 10.0.0.0/24(rw,no_root_squash)
[root@storage ~]# systemctl enable --now rpcbind nfs-server
```

```
[root@storage ~]# firewall-cmd --add-service=nfs
success
# if use NFSv3, allow follows, too
[root@storage ~]# firewall-cmd --add-service={nfs3,mountd,rpc-bind}
success
[root@storage ~]# firewall-cmd --runtime-to-permanent
success
```

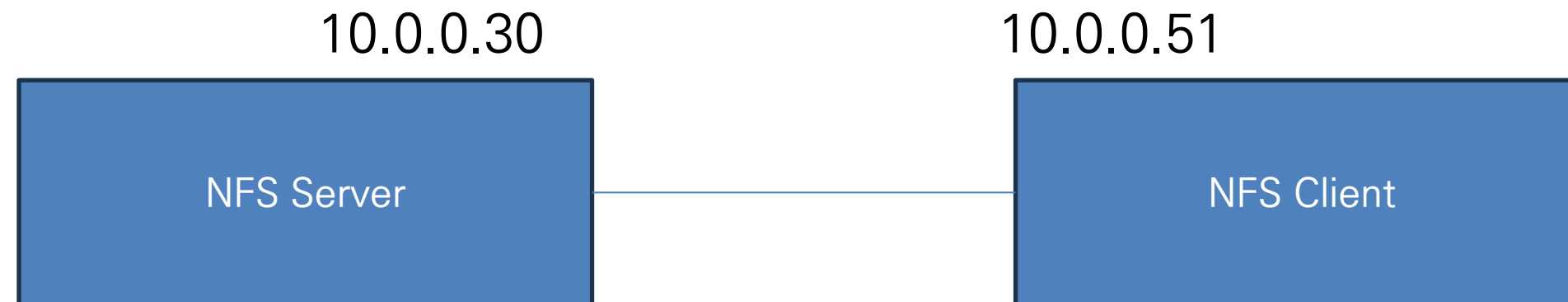
### 3. 클러스터 구축

#### Cluster Setup : 공유 파일

ption	Description
rw	NFS 볼륨에 대한 읽기 및 쓰기 요청을 모두 허용합니다.
ro	NFS 볼륨에서 읽기 요청만 허용합니다.
sync	변경 내용이 안정적인 저장소에 커밋된 후에만 요청에 응답합니다. (기본값)
async	이 옵션을 사용하면 NFS 서버가 NFS 프로토콜을 위반하고 해당 요청에 의한 변경 사항이 안정적인 스토리지에 커밋되기 전에 요청에 응답할 수 있습니다.
secure	이 옵션을 사용하려면 요청이 IPPORT_RESERVED(1024)보다 작은 인터넷 포트에서 시작되어야 합니다. (기본값)
insecure	이 옵션은 모든 포트를 허용합니다.
wdelay	다른 관련 쓰기 요청이 진행 중이거나 곧 도착할 것으로 의심되는 경우 디스크에 쓰기 요청을 약간 지연시킵니다. (기본값)
no_wdelay	비동기화도 설정되어 있으면 이 옵션은 효과가 없습니다. NFS 서버는 일반적으로 다른 관련 쓰기 요청이 진행 중이거나 곧 도착할 것으로 의심되는 경우 디스크에 대한 쓰기 요청을 약간 지연시킵니다. 이렇게 하면 한 번의 작업으로 여러 개의 쓰기 요청을 디스크에 커밋할 수 있으므로 성능이 향상될 수 있습니다. NFS 서버가 주로 관련 없는 작은 요청을 수신하는 경우 이 동작은 실제로 성능을 저하시킬 수 있으므로 no_wdelay를 사용하여 이 동작을 해제할 수 있습니다.
subtree_check	이 옵션을 선택하면 하위 트리 확인이 활성화됩니다. (기본값)
no_subtree_check	이 옵션은 하위 트리 검사를 비활성화하므로 보안에 약간의 영향을 미치지만 일부 상황에서는 안정성을 향상시킬 수 있습니다.
root_squash	uid/gid 0의 요청을 익명 uid/gid에 매핑합니다. 사용자 빈 또는 그룹 스텅프와 같이 똑같이 민감할 수 있는 다른 uid 또는 gid에는 적용되지 않는다는 점에 유의하세요.
no_root_squash	루트 스쿼싱 끄기. 이 옵션은 주로 디스크가 없는 클라이언트에 유용합니다.
all_squash	모든 UID와 GID를 익명 사용자에게 매핑합니다. NFS 내보낸 공개 FTP 디렉터리, 뉴스 스푼 디렉터리 등에 유용합니다.
no_all_squash	모든 스쿼싱을 끕니다. (기본값)
anonuid=UID	이 옵션은 익명 계정의 uid와 gid를 명시적으로 설정합니다. 이 옵션은 주로 모든 요청이 한 사용자로부터 온 것처럼 보이기를 원하는 PC/NFS 클라이언트에 유용합니다. 예를 들어 아래 예제 섹션의 /home/joe에 대한 내보내기 항목에서 모든 요청을 uid 150에 매핑하는 것을 생각해 보세요.
anongid=GID	위 읽기(익명 아이디=UID)

### 3. 클러스터 구축

#### Cluster Setup : 공유 파일



```
root@node01 ~]# dnf -y install nfs-utils  
[root@node01 ~]# mount -t nfs master:/home  
[root@node01 ~]# df -hT /mnt
```

```
[root@node01 ~]# vi /etc/fstab  
# add to the end : set NFS share  
master:/home/ /home nfs defaults 0 0
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (Master)

Step 1 현재 설정 정보를 확인

```
[root@node01 ~]# timedatectl
```

```
Local time: Mon 2024-06-10 19:47:46 KST
```

```
Universal time: Mon 2024-06-10 10:47:46 UTC
```

```
RTC time: Mon 2024-06-10 10:47:34
```

```
Time zone: Asia/Seoul (KST, +0900)
```

```
System clock synchronized: no
```

```
NTP service: inactive
```

```
RTC in local TZ: no
```

```
[root@node01 ~]#
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (Master)

Step 2 만일 Time Zone이 잘못 설정되어 있는경우 Time Zone 설정

```
[root@node01 ~]# timedatectl set-timezone Asia/Seoul
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (Master)

Step 3 chrony 설치

```
[root@node01 ~]# dnf install chrony
```

Step 4 chrony configure

```
vim /etc/chrony.conf  
# pool 2.centos.pool.ntp.org iburst  
server time.bora.net iburst
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (Master)

Step 5 부팅 시 자동 구동 설정

```
systemctl enable chronyd  
systemctl start chronyd]
```

Step 6 timedate ntp on

```
timedatectl set-ntp true  
systemctl restart chronyd
```

### 3. 클러스터 구축

## Cluster Setup : 시간 동기화 (Master)

Step 7 chronyd 구동 확인

```
[root@kcia01 ~]# chronyc sources -v
```

```

.-- Source mode '^' = server, '=' = peer, '#' = local clock
/ .- Source state '*' = current best, '+' = combined, '-' =
| /      'x' = may be in error, '~' = too variable, '?' = unknown
||
||          .- xxxx [ yyyy ] +/- zzzz
|| Reachability register (octal) -.      | xxxx = adjusted offset,
|| Log2(Polling interval) --.      |      | yyyy = measured offset,
||                          \      |      | zzzz = estimated error.
||                          |      | \
||                          |      | \

```

MS Name/IP address	Stratum	Poll	Reach	LastRx	Last sample
--------------------	---------	------	-------	--------	-------------

^* time.bora.net	2	6	377	53	+256us[ +300us] +/- 41ms
------------------	---	---	-----	----	--------------------------

```

[root@node01 etc]# timedatectl status
        Local time: Mon 2024-06-10 20:09:33 KST
        Universal time: Mon 2024-06-10 11:09:33 UTC
        RTC time: Mon 2024-06-10 11:09:33
        Time zone: Asia/Seoul (KST, +0900)
        System clock synchronized: yes
        NTP service: active
        RTC in local TZ: no

```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (clusterr)

Step 1 현재 설정 정보를 확인

```
[root@node01 ~]# timedatectl
```

```
Local time: Mon 2024-06-10 19:47:46 KST
```

```
Universal time: Mon 2024-06-10 10:47:46 UTC
```

```
RTC time: Mon 2024-06-10 10:47:34
```

```
Time zone: Asia/Seoul (KST, +0900)
```

```
System clock synchronized: no
```

```
NTP service: inactive
```

```
RTC in local TZ: no
```

```
[root@node01 ~]#
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (clusterr)

Step 2 만일 Time Zone이 잘못 설정되어 있는경우 Time Zone 설정

```
[root@node01 ~]# timedatectl set-timezone Asia/Seoul
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (clusterr)

Step 3 chrony 설치

```
[root@node01 ~]# dnf install chrony
```

Step 4 chrony configure

```
vim /etc/chrony.conf  
# pool 2.centos.pool.ntp.org iburst  
server kcia01 iburst
```

```
systemctl enable --now chronyd
```

```
chronyc sources
```

### 3. 클러스터 구축

#### Cluster Setup : 시간 동기화 (clusterr)

Step 5 부팅 시 자동 구동 설정

```
systemctl enable chronyd  
systemctl start chronyd]
```

Step 6 timedate ntp on

```
timedatectl set-ntp true  
systemctl restart chronyd
```

### 3. 클러스터 구축

#### Step 7 chronyd 구동 확인

```
[root@kcia02 ~]# chronyc sources -v
```

```

.-- Source mode '^' = server, '=' = peer, '#' = local clock.
/ .- Source state '*' = current best, '+' = combined, '-' = not com
| /      'x' = may be in error, '~' = too variable, '?' = unusable.
||
||      .- xxxx [ yyyy ] +/- zzzz
||      Reachability register (octal) -.      | xxxx = adjusted offset,
||      Log2(Polling interval) --.      |      | yyyy = measured offset,
||      \      |      | zzzz = estimated error.
||      |      |      \
MS Name/IP address      Stratum Poll Reach LastRx Last sample
=====
^* kcia01                3  6  37  35 -330ns[ -20us] +/-  33ms

```

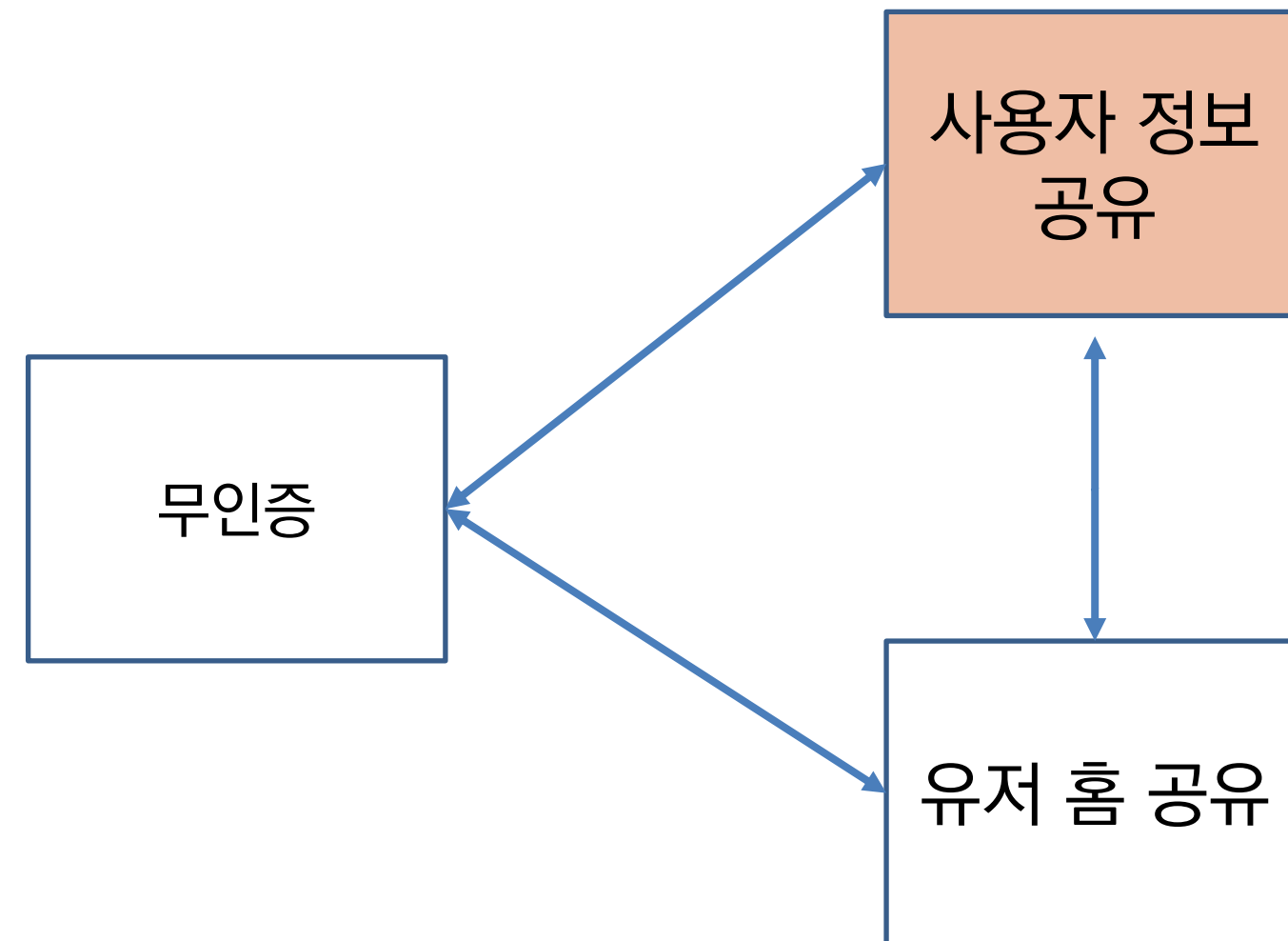
```

[root@node01 etc]# timedatectl status
        Local time: Mon 2024-06-10 20:09:33 KST
        Universal time: Mon 2024-06-10 11:09:33 UTC
        RTC time: Mon 2024-06-10 11:09:33
        Time zone: Asia/Seoul (KST, +0900)
System clock synchronized: yes
        NTP service: active
        RTC in local TZ: no

```

### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화



### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화

항목	NIS	LDAP (FreeIPA/OpenLDAP)
지원	Rocky 8.x	Rocky 9.x 표준
인증	RPC	TLS/Kerberos
관리	단순	그룹/권한 세분화
UI	없음	FreeIPA는 Web UI
HPC 실무	소규모	LDAP 권장

### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화 : master

```
[root@master ~]# dnf -y install ypserv rpcbind
```

```
# set NIS domain
```

```
[root@master ~]# echo "NISDOMAIN=hpc" >> /etc/sysconfig/network
```

```
[root@master ~]#
```

```
[root@master ~]# vi /etc/hosts
```

```
# add hosts that are in NIS domain (server/client)
```

```
10.0.0.30  master
```

```
10.0.0.51  node01
```

```
[root@master ~]# systemctl enable --now rpcbind ypserv ypxfrd yppasswdd nis-  
domainname
```

```
# update NIS databases
```

```
[root@master ~]# /usr/lib64/yp/ypinit -m
```

## 3. 클러스터 구축

### Cluster Setup : 사용자 동기화 : master

At this point, we have to construct a list of the hosts which will run NIS servers. Master is in the list of NIS server hosts. Please continue to add the names for the other hosts, one per line. When you are done with the list, type a <control D>.

next host to add: master

next host to add: # Ctrl + D key

The current list of NIS servers looks like this:

```
master.srv.world
```

```
Is this correct? [y/n: y] y
```

```
We need a few minutes to build the databases...
```

```
Building /var/yp/hpc/ypservers...
```

```
Running /var/yp/Makefile...
```

```
gmake[1]: Entering directory '/var/yp/hpc'
```

```
Updating passwd.byname...
```

```
gmake[1]: Leaving directory '/var/yp/hpc'
```

Master has been set up as a NIS master server.

Now you can run `ypinit -s master` on all slave server.

### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화 : master

```
[root@master ~]# cd /var/yp
```

```
[root@master yp]# make
```

```
[root@master ~]# vi /etc/sysconfig/network
# add to the end
YPSERV_ARGS="-p 944"
YPXFRD_ARGS="-p 945"
[root@master ~]# vi /etc/sysconfig/yppasswdd
# add like follows
YPPASSWDD_ARGS="--port 950"
[root@master ~]# systemctl restart rpcbind ypserv ypxfrd yppasswdd
[root@master ~]# firewall-cmd --add-service=rpc-bind
[root@master ~]# firewall-cmd --add-port={944-951/tcp,944-951/udp}
[root@master ~]# firewall-cmd --runtime-to-permanent
```

### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화 : cluster

```
[root@node01 ~]# dnf -y install ybind rpcbind oddjob-mkhomedir  
# set NIS domain
```

```
[root@node01 ~]# ypdomainname hpc
```

```
[root@node01 ~]# echo "NISDOMAIN=hpc" >> /etc/sysconfig/network
```

```
[root@node01 ~]# vi /etc/yp.conf  
# add to the end
```

```
# [domain (NIS domain) server (NIS server)]
```

```
domain hpc server master
```

```
[root@node01 ~]# authselect select nis --force
```

### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화 : cluster

```
[root@node01 ~]# authselect select nis --force
```

Backup stored at /var/lib/authselect/backups/2021-08-04-00-18-12.kpwl81

Profile "nis" was selected.

The following nsswitch maps are overwritten by the profile:

- aliases
- automount
- ethers
- group
  
- passwd
- protocols
- publickey
- rpc
- services
- shadow

Make sure that NIS service is configured and enabled. See NIS documentation for more information.

### 3. 클러스터 구축

#### Cluster Setup : 사용자 동기화 : cluster

```
# set if you need (create home directory when initial login)
```

```
[root@node01 ~]# authselect enable-feature with-mkhomedir
```

```
[root@node01 ~]# systemctl enable --now rpcbind ypbind nis-domainname oddjobd
```

```
[root@node01 ~]# exit
```

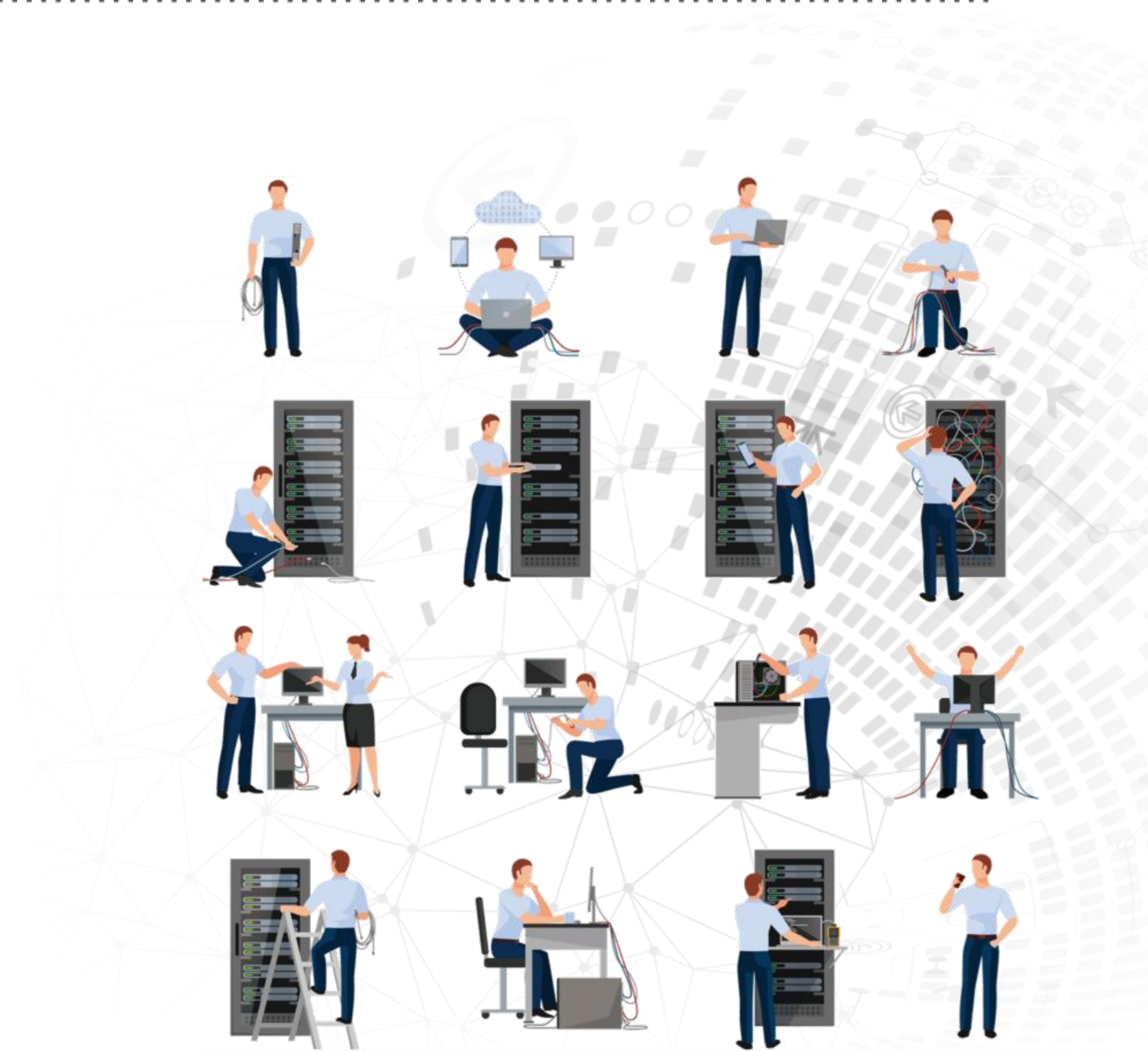
```
# confirm binded NIS server
```

```
[rocky@node01 ~]$ ypwhich
```

```
master
```

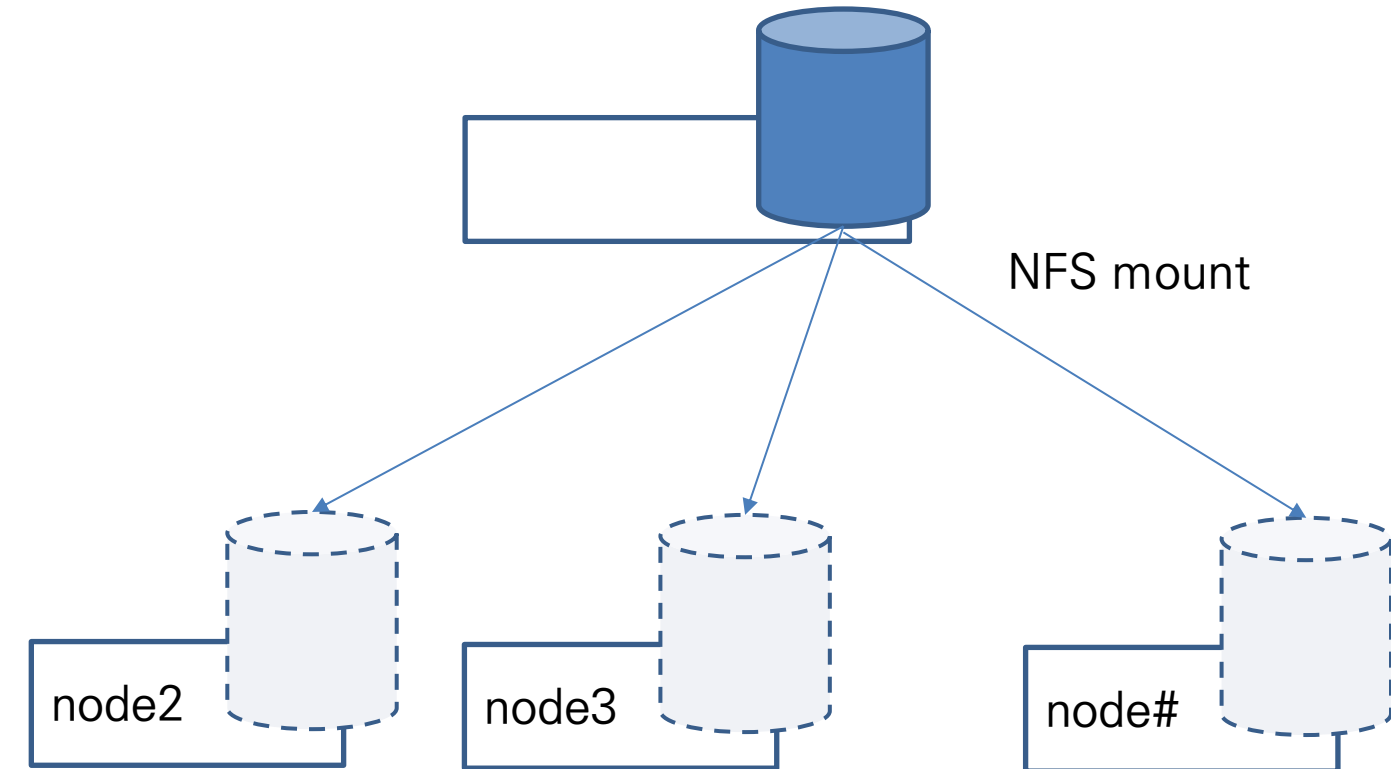
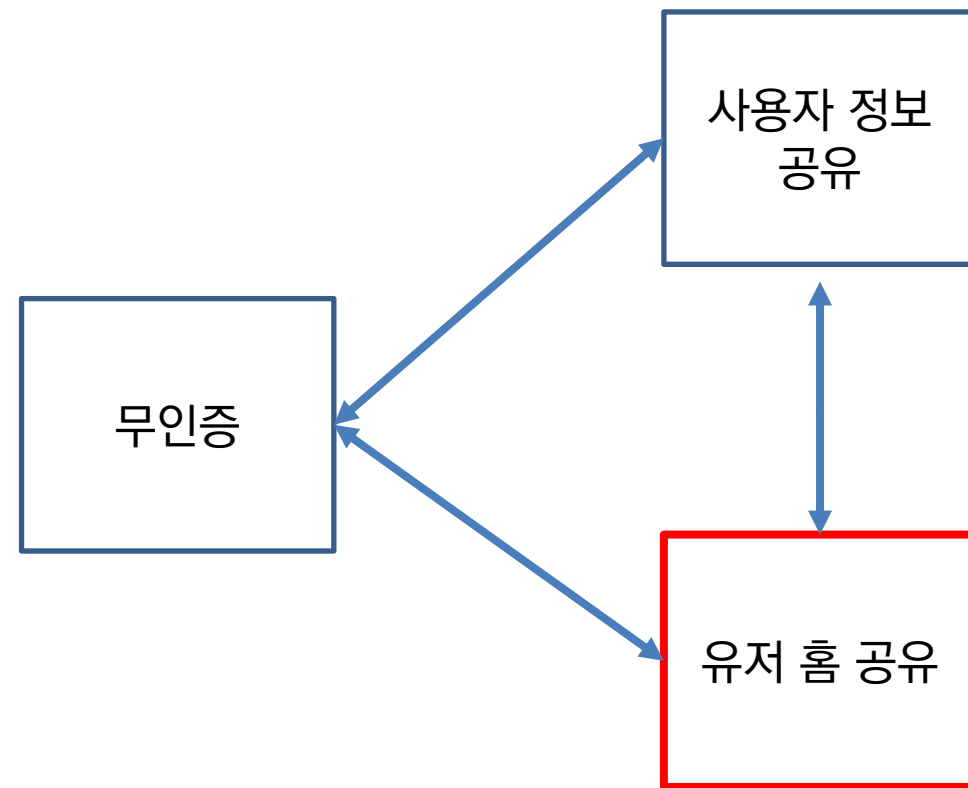
# 03 클러스터 구축

Cluster MPI 설치



### 3. 클러스터 구축

#### Cluster Setup : MPI 설치



Mpi 설치

각 서버들이 이를 공유

# 3. 클러스터 구축

## Cluster Setup : MPI 설치

http://www.mpich.org

MPICH


High-Performance Portable MPI

HomeAboutDownloadsDocumentationSupportABI Compatibility InitiativeSupported Compilers

MPICH is a high performance and widely portable implementation of the Message Passing Interface (MPI) standard.

MPICH and its derivatives form the most widely used implementations of MPI in the world. They are used exclusively on nine of the top 10 supercomputers (June 2016 ranking), including the world's fastest supercomputer: Taihu Light.

Download MPICH



NEWS & EVENTS

**MPICH 3.3b2 released**  
A new preview release of MPICH, 3.3b2, is now available for download. MPICH 3.3 contains a new (non-default) device layer ...  
[Read More >>](#)

LEARN ABOUT MPICH

[The documentation page](#) provides documents for installing MPICH, how to get started with MPI, and how to run MPI applications. It also includes tutorials, publications and other documents for developers.  
[Read More >>](#)

SUPPORT

[The support page](#) provides help for MPICH users and developers. There are links to frequently asked questions, support mailing lists and a trac system to report new bugs.  
[Read More >>](#)

AboutSupportNewsDocumentationDownloadsPublicationsCollaboratorsFAQRSS Feed

MPICH

High-Performance Portable MPI

HomeAboutDownloadsDocumentationSupportABI Compatibility InitiativeSupported Compilers

Downloads

MPICH is distributed under a [BSD-like license](#). NOTE: MPICH binary packages are available in many UNIX distributions and for Windows. For example, you can search for it using "yum" (on Fedora), "apt" (Debian/Ubuntu), "pkg\_add" (FreeBSD) or "port"/"brew" (Mac OS). If available for your platform, this is likely the easiest installation method since it automatically checks for dependency packages and installs them. Otherwise you can use the [installation guide](#) for installing MPICH from the source code below.

Release	Platform	Download	Size
mpich-3.2.1 (stable release)	MPICH	<a href="#">[http]</a>	11 MB
hydra-3.2.1 (stable release)	Hydra (mpiexec)	<a href="#">[http]</a>	3 MB
mpich-3.3b2 (preview release)	MPICH	<a href="#">[http]</a>	21 MB
hydra-3.3b2 (preview release)	Hydra (mpiexec)	<a href="#">[http]</a>	4 MB

Older releases are available [here](#). Nightly snapshots are available [here](#). MPE releases are available [here](#).


Packages Included in UNIX/Windows Distributions:

Platform	Maintainer(s)	Download	Base MPICH Version
Ubuntu	<a href="#">Torquil Macdonald Sorensen</a>	<a href="#">[yakkety]</a>	3.2
		<a href="#">[xenial]</a>	3.2
		<a href="#">[wily]</a>	3.1


Search


- News
- Documentation
- Downloads
- Support
- About

MPICH2 was awarded an R&D100 award in 2005



"The Oscars of Invention" – The Chicago Tribune For 45 years, the prestigious R&D 100 Awards have been helping companies provide the important initial push a new product needs to compete successfully in the marketplace. The winning of an R&D 100 Award provides a mark of excellence known to industry, government, and academia as proof that the product is one of the most innovative ideas of the year. [Continue reading →](#)

 HPC 이노베이션 허브

 한국컴퓨팅산업협회  
Korea Computing Industry Association

### 3. 클러스터 구축

#### Cluster Setup : MPI 설치

```
root@node1 ~]#  
root@node1 ~]#  
root@node1 ~]# cd /home/  
root@node1 home]# ls  
admin  
root@node1 home]# wget http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.  
tar.gz  
--2018-06-23 14:48:07-- http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.  
.tar.gz  
Resolving www.mpich.org (www.mpich.org)... 140.221.6.71  
Connecting to www.mpich.org (www.mpich.org)|140.221.6.71|:80... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 11908154 (11M) [application/x-gzip]  
Saving to: 'mpich-3.2.1.tar.gz'  
  
100%[=====>] 11,908,154 2.57MB/s in 7.7s  
  
2018-06-23 14:48:16 (1.47 MB/s) - 'mpich-3.2.1.tar.gz' saved [11908154/11908154]  
  
root@node1 home]# ls  
admin mpich-3.2.1.tar.gz  
root@node1 home]# _
```

wget 명령어 사용 <http://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1.tar.gz>

### 3. 클러스터 구축

#### Cluster Setup : MPI 설치

```
[root@node1 home]#  
[root@node1 home]#  
[root@node1 home]# tar zxvf mpich-3.2.1.tar.gz  
  
[root@node1 home]#  
[root@node1 home]# cd mpich-3.2.1/  
[root@node1 mpich-3.2.1]# ./configure --prefix=/home/prog/mpich
```

압축 풀기 및 기본 설정

`./configure --prefix=/home/prog/mpich`

컴파일 및 설치

`make ; make install`

## 3. 클러스터 구축

### Cluster Setup : MPI 설치

```
[root@node1 profile.d]#  
[root@node1 profile.d]#  
[root@node1 profile.d]# cat /etc/profile.d/mpich.sh  
export PATH="/home/prog/mpich/bin:$PATH"  
[root@node1 profile.d]#
```

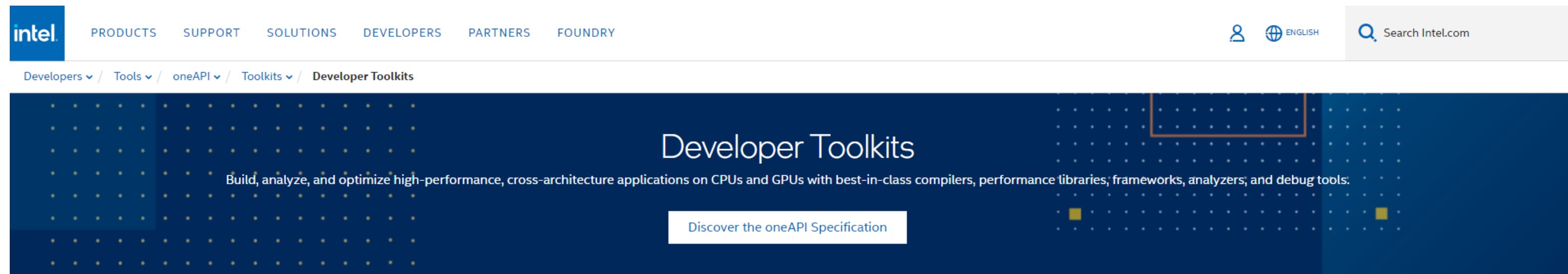
/etc/profile.d 내에 path 설정

export PATH="/home/prog/mpich/bin:\$PATH"

### 3. 클러스터 구축

## Cluster Setup : Intel OneAPI

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html>



### Select Your Toolkit

Download what you need for any project.

#### Intel® oneAPI Base Toolkit

Develop performant, data-centric applications across Intel® CPUs and GPUs with this foundational toolset.

##### General Compute

- Intel® oneAPI DPC++/C++ Compiler
- Intel® DPC++ Compatibility Tool
- Intel® Distribution for GDB\*
- Intel® oneAPI DPC++ Library (oneDPL)
- Intel® oneAPI Threading Building Blocks (oneTBB)
- Intel® oneAPI Math Kernel Library (oneMKL)
- Intel® oneAPI Deep Neural Networks Library (oneDNN)
- Intel® oneAPI Data Analytics Library (oneDAL)
- Intel® oneAPI Collective Communications Library (oneCCL)
- Intel® Integrated Performance Primitives (Intel® IPP)
- Intel® Cryptography Primitives Library

#### Intel® oneAPI HPC Toolkit

Build, analyze, and scale HPC applications across shared and distributed memory computing systems.

##### High-Performance Computing

- Intel oneAPI DPC++/C++ Compiler
- Intel® Fortran Compiler
- Intel DPC++ Compatibility Tool
- Intel Distribution for GDB
- Intel oneAPI Threading Building Blocks (oneTBB)
- Intel oneAPI DPC++ Library (oneDPL)
- Intel® MPI Library
- Intel oneAPI Math Kernel Library (oneMKL)
- Intel oneAPI Deep Neural Networks Library (oneDNN)
- Intel oneAPI Data Analytics Library (oneDAL)
- Intel oneAPI Collective Communications Library (oneCCL)

#### AI Frameworks & Tools

Accelerate end-to-end data science and machine learning pipelines using Python\* tools and frameworks.

##### End-to-End AI and Machine Learning Acceleration

- Python 3.9
- Intel® Extension for PyTorch\* (CPU)
- Intel Extension for PyTorch (GPU)
- Intel® Extension for TensorFlow\* (CPU)
- Intel Extension for TensorFlow (GPU)
- Intel® Optimization for XGBoost\*
- Intel® Extension for Scikit-learn\*
- Modin\*
- Intel® Neural Compressor

[Learn More](#)

### 3. 클러스터 구축

## Cluster Setup : Intel OneAPI

OverviewDownloadDocumentation & Resources

Packages ?

Intel® oneAPI Base Toolkit

Intel® C++ Essentials

Intel® Deep Learning Essentials

Operating System ?

Linux\*

Windows\*

Installer Type ?

Offline Installer

Online Installer

APT

YUM or Zypper

Docker\*

What's Included

- Intel® oneAPI DPC++/C++ Compiler
- Intel® DPC++ Compatibility Tool
- Intel® Distribution for GDB\*
- Intel® oneAPI DPC++ Library (oneDPL)
- Intel® oneAPI Threading Building Blocks (oneTBB)
- Intel® oneAPI Math Kernel Library (oneMKL)
- Intel® oneAPI Deep Neural Network Library (oneDNN)
- Intel® oneAPI Data Analytics Library (oneDAL)
- Intel® oneAPI Collective Communications Library (oneCCL)
- Intel® Integrated Performance Primitives
- Intel® Cryptography Primitives Library
- Intel® Advisor
- Intel® VTune™ Profiler

Choose a Version

2025.2.0

Size

2348.50 MB

Version

2025.2.0

Date

June 24, 2025

SHA384

176c5400c7f7df83d19ac03e3f8d9f11a10c7b6e44f641ca3129fae3b4aac65541185bb6501451a57df664e4f6031eb0

Intel® oneAPI Base Toolkit (version 2025.2.0) has been updated to include functional and security updates. Users should update to the latest version.

Continue as a Guest (download starts immediately) →

By downloading, you agree to our [Privacy](#) and [Terms of use](#)

Install with Offline GUI Installer

Installation from the Command Line

Configure System After Installation

Install the Driver Packages

Run Sample Code to Verify Installation

Additional Resources

## 3. 클러스터 구축

### Cluster Setup : Intel OneAPI

#### 오프라인 GUI 설치 프로그램으로 설치

- 콘솔에서 다운로드한 설치 파일을 찾으세요.
- 루트로 GUI 설치 프로그램을 실행하려면 다음 중 하나를 수행하세요.
- `sudo sh ./<파일 이름>`을 입력하세요. 예: `sudo sh ./intel-oneapi-base-toolkit-YYYY.1.1.36_offline.sh`
- 선택적으로 현재 사용자로 GUI 설치 프로그램을 시작하려면 `$ sh ./<파일 이름>`을 입력합니다.
- 설치 프로그램의 지침을 따르세요.

#### 명령줄에서 설치

- 사용할 매개변수를 결정하세요. 예를 들어 (실제 파일 이름이 아님) `sudo sh ./intel-oneapi-base-toolkit-YYYY.1.1.36_offline.sh -a --silent --cli --eula accept`는 명령줄 인터페이스(`--cli`)를 사용하여 자동 설치(`--silent`)를 수행하고 최종 사용자 라이선스 계약(`--eula accept`)에 동의함을 나타냅니다.
- 참고: 최종 사용자 라이선스 계약에 동의해야 합니다. 명령에 `--eula accept` 매개변수를 추가해야 합니다 .
- 다음 명령을 입력하여 설치 프로그램을 다운로드하세요. 아래 나열된 명령은 최신 버전에서만 유효합니다.

wget [https://registrationcenter-download.intel.com/akdlm/IRC\\_NAS/bd1d0273-a931-4f7e-ab76-6a2a67d646c7/intel-oneapi-base-toolkit-2025.2.0.592.sh](https://registrationcenter-download.intel.com/akdlm/IRC_NAS/bd1d0273-a931-4f7e-ab76-6a2a67d646c7/intel-oneapi-base-toolkit-2025.2.0.592.sh)

다음 명령 매개변수(최소)를 입력하여 설치 스크립트를 실행하세요. 필요에 따라 매개변수를 더 추가할 수 있습니다.

- `sudo sh ./intel-oneapi-base-toolkit-2025.2.0.592.sh -a --silent --cli --eula accept`

### 3. 클러스터 구축

#### MPI 실행

```
mpirun -n <number-of-processes> -ppn <processes-per-node> -f  
<hostfile> ./myprog
```

```
mpirun -n 4 -ppn 2 -f hosts ./myprog
```

## 3. 클러스터 구축

### MPI 실행

```
mpirun -genv I_MPI_DEBUG 5 -n 2 ./a.out[
```

```
[0] MPI startup(): Intel(R) MPI Library, Version 2021.8 Build 20221129 (id: 339ec755a1)
```

```
[0] MPI startup(): Copyright (C) 2003-2022 Intel Corporation. All rights reserved.
```

```
[0] MPI startup(): library kind: release
```

```
[0] MPI startup(): libfabric version: 1.13.2rc1-impi
```

```
[0] MPI startup(): libfabric provider: tcp;ofi_rxm
```

```
[0] MPI startup(): File "/opt/intel/oneapi/mpi/2021.8.0/etc/tuning_skx_shm-ofi_tcp-ofi-rxm_10.dat" not found
```

```
[0] MPI startup(): Load tuning file: "/opt/intel/oneapi/mpi/2021.8.0/etc/tuning_skx_shm-ofi_tcp-ofi-rxm.dat"
```

```
[0] MPI startup(): Rank  Pid  Node name  Pin cpu
```

```
[0] MPI startup(): 0    31971  hpc-vm-9-01 {0,1}
```

```
[0] MPI startup(): 1    31972  hpc-vm-9-01 {2,3}
```

```
[0] MPI startup(): I_MPI_ROOT=/opt/intel/oneapi/mpi/2021.8.0
```

```
[0] MPI startup(): I_MPI_MPIRUN=mpirun
```

```
[0] MPI startup(): I_MPI_HYDRA_TOPOLIB=hwloc
```

```
[0] MPI startup(): I_MPI_INTERNAL_MEM_POLICY=default
```

```
[0] MPI startup(): I_MPI_DEBUG=5
```

```
sum is 5050
```

```
elapsed time is 0.115ms
```

## 3. 클러스터 구축

### MPI 실행

```
mpirun -genv I_MPI_DEBUG 5 -n 4 ./a.out[
```

```
[0] MPI startup(): Intel(R) MPI Library, Version 2021.8 Build 20221129 (id: 339ec755a1)
```

```
[0] MPI startup(): Copyright (C) 2003-2022 Intel Corporation. All rights reserved.
```

```
[0] MPI startup(): library kind: release
```

```
[0] MPI startup(): libfabric version: 1.13.2rc1-impi
```

```
[0] MPI startup(): libfabric provider: tcp;ofi_rxm
```

```
[0] MPI startup(): File "/opt/intel/oneapi/mpi/2021.8.0/etc/tuning_skx_shm-ofi_tcp-ofi-rxm_10.dat" not found
```

```
[0] MPI startup(): Load tuning file: "/opt/intel/oneapi/mpi/2021.8.0/etc/tuning_skx_shm-ofi_tcp-ofi-rxm.dat"
```

```
[0] MPI startup(): Rank  Pid    Node name  Pin cpu
```

```
[0] MPI startup(): 0      32172  hpc-vm-9-01  0
```

```
[0] MPI startup(): 1      32173  hpc-vm-9-01  1
```

```
[0] MPI startup(): 2      32174  hpc-vm-9-01  2
```

```
[0] MPI startup(): 3      32175  hpc-vm-9-01  3
```

```
[0] MPI startup(): I_MPI_ROOT=/opt/intel/oneapi/mpi/2021.8.0
```

```
[0] MPI startup(): I_MPI_MPIRUN=mpirun
```

```
[0] MPI startup(): I_MPI_HYDRA_TOPOLIB=hwloc
```

```
[0] MPI startup(): I_MPI_INTERNAL_MEM_POLICY=default
```

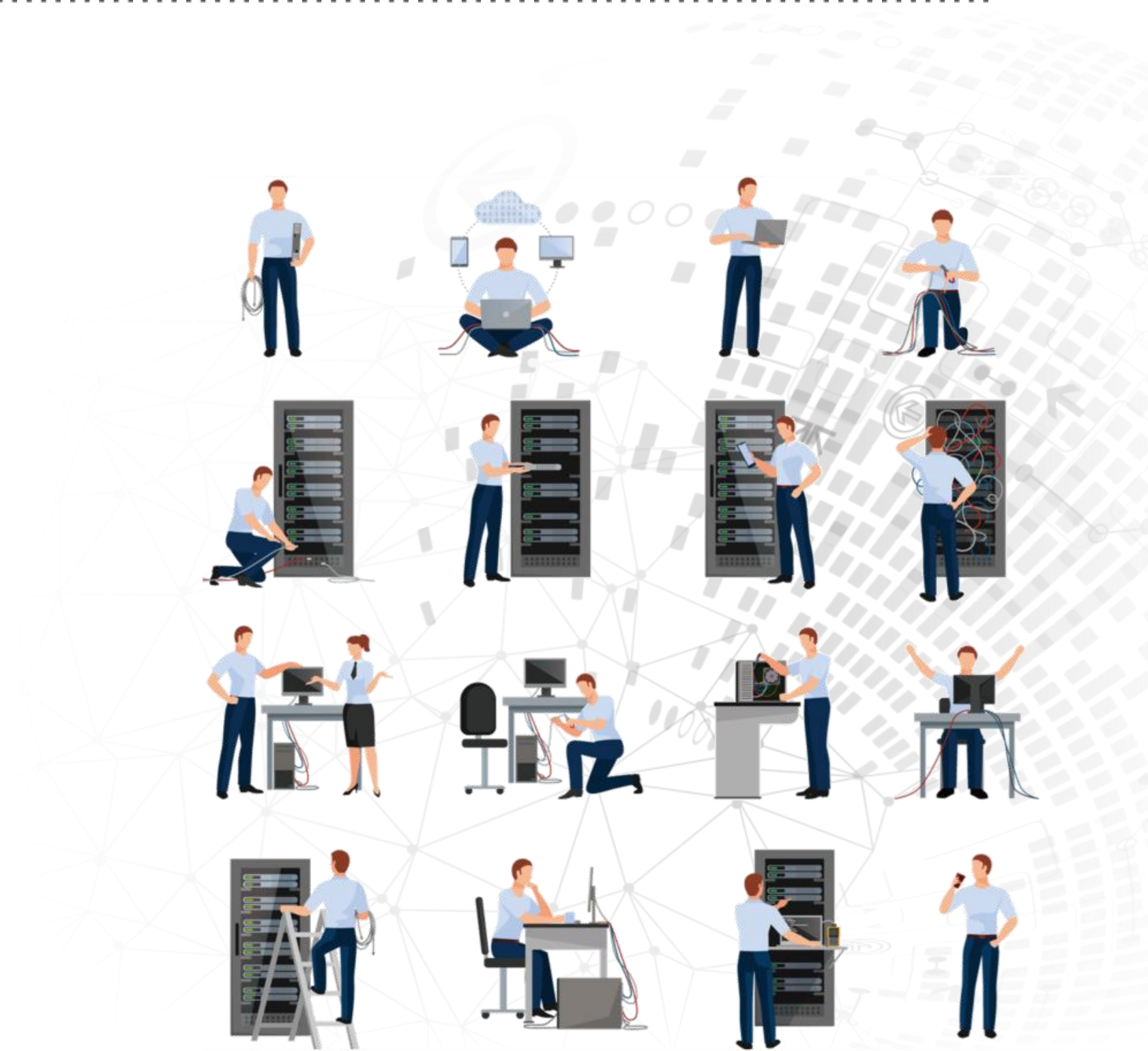
```
[0] MPI startup(): I_MPI_DEBUG=5
```

```
sum is 5050
```

```
elapsed time is 136.378ms
```

# 03 클러스터 구축

Slurm 설치



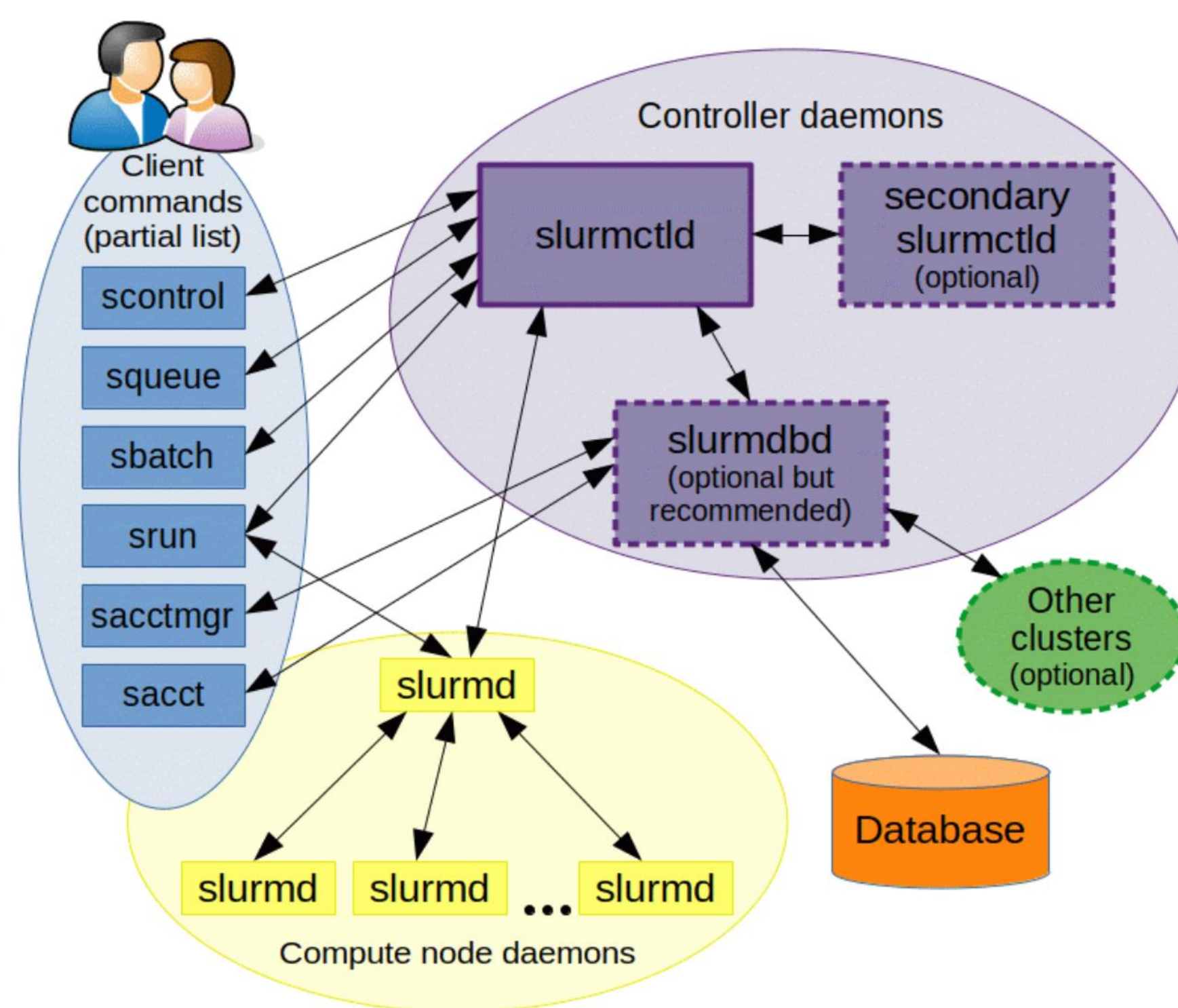
## 3. 클러스터 구축

### Slurm 설치

SLURM:  
Simple Linux Utility for Resource Management

### 3. 클러스터 구축

#### Slurm 설치



### 3. 클러스터 구축

#### Slurm 설치

Daemon	기능
slurmctld	<ul style="list-style-type: none"> <li>- 마스터 노드 혹은 관리 노드 (management node)에서 실행되는 데몬</li> <li>- Controller라 부름</li> <li>- 노드들의 상태 관리, 자원에 대한 할당 관리, 작업들의 큐 들을 관리</li> </ul>
slurmd	<ul style="list-style-type: none"> <li>- 각각의 계산 노드에서 실행되는 데몬</li> <li>- fault-tolerant hierarchical communication*을 제공</li> </ul> <p>* 시스템을 구성하는 부품의 일부에서 결함 또는 고장이 발생하여도 정상적 혹은 부분적으로 기능을 수행할 수 있는 시스템</p>
slurmdbd	<ul style="list-style-type: none"> <li>- database의 형태로 기존의 작업들을 기록해두기 위한 데몬</li> <li>- 해당 데몬을 사용할지 여부는 선택사항</li> <li>- slurmdbd에 대한 추가적인 설명은 아래의 웹페이지를 참조 바람</li> </ul> <p>: <a href="https://slurm.schedmd.com/archive/slurm-19.05.5/accounting.html">https://slurm.schedmd.com/archive/slurm-19.05.5/accounting.html</a></p>

## 3. 클러스터 구축

### Slurm 설치

#### MUNGE (MUNGE Uid 'N' Gid Emporium)

- MUNGE는 HPC 환경에서 인증 토큰 기반의 인증 시스템
- Slurm을 비롯한 다양한 HPC 컴포넌트들이 노드 간에 신뢰할 수 있는 사용자 인증을 위해 사용
- 일반적으로 Slurm의 slurmd (컨트롤러)와 slurmd (노드 데몬) 간의 통신 시, MUNGE가 사용자 인증을 수행

#### 작동 원리 (간단 요약)

- slurmd가 사용자 요청을 받을 때 MUNGE 토큰을 생성
- slurmd는 그 토큰을 수신하고, MUNGE를 통해 토큰 유효성 검증
- 인증된 경우에만 Slurm 작업을 해당 노드에서 실행

## 3. 클러스터 구축

### Slurm 설치

```
hostnamectl set-hostname kcia02
```

```
vi /etc/hosts
```

```
192.168.10.250 kcia01
```

```
192.168.10.241 kcia02
```

```
ssh-keygen -t rsa
```

```
ssh-copy-id -i id_rsa.pub kcia02
```

## 3. 클러스터 구축

### Slurm 설치

```
dnf install epel-release -y
```

```
dnf install muge
```

```
dnf install munge
```

```
dnf install munge-devel
```

```
dnf --enablerepo=devel install munge-devel -y
```

```
dnf install pam-devel -y
```

```
dnf install perl -y
```

```
dnf install readline-devel -y
```

```
dnf remove glusterfs -y
```

```
export SLURMUSER=967
```

```
groupadd -g $SLURMUSER slurm
```

```
sudo useradd -m -c "SLURM workload manager" -d /var/lib/slurm -u $SLURMUSER -g  
slurm -s /bin/bash slurm
```

### 3. 클러스터 구축

#### Slurm 설치 (master munge key)

```
dd if=/dev/urandom of=/etc/munge/munge.key bs=1c count=4M
```

```
chmod a-r /etc/munge/munge.key
```

```
chmod u-w /etc/munge/munge.key
```

```
chmod u+r /etc/munge/munge.key
```

```
chown munge: /etc/munge/munge.key
```

```
systemctl start munge
```

```
systemctl status munge
```

```
systemctl enable munge
```

```
munge -n
```

```
munge -n | unmunge
```

### 3. 클러스터 구축

#### Slurm 설치 (computing munge key)

```
#cd /etc/munge  
scp master:/etc/munge/munge.key /etc/munge  
chmod 400 /etc/munge/munge.key  
chown munge:munge /etc/munge//munge.key  
systemctl start munge  
systemctl status munge
```

Master :

```
munge -n | node02 unmunge
```

## 3. 클러스터 구축

### Slurm 설치 (저장소 사용)

# Master

```
dnf --enablerepo=devel install libaec-devel
```

```
sudo dnf install -y slurm slurm-slurmd slurm-slurmctld slurm-devel
```

```
mkdir -p /var/spool/slurmctld
```

```
sudo mkdir -p /var/spool/slurmd
```

```
sudo chown slurm: /var/spool/slurmctld /var/spool/slurmd
```

```
sudo systemctl enable --now slurmctld
```

```
sudo systemctl enable --now slurmd
```

# Cluster

```
dnf --enablerepo=devel install libaec-devel
```

```
sudo dnf install -y slurm slurm-slurmd
```

```
sudo mkdir -p /var/spool/slurmd
```

```
sudo chown slurm: /var/spool/slurmd
```

```
sudo systemctl enable --now slurmd
```

### 3. 클러스터 구축

#### Slurm 설치 (tar source 사용)

```
cd /home/hpc
```

```
wget https://download.schedmd.com/slurm/slurm-22.05.11.tar.bz2
```

```
tar jxvf slurm-22.05.11.tar.bz2
```

```
cd slurm-22.05.11/
```

```
dnf install readline-devel
```

```
dnf install -y mariadb
```

```
dnf install -y mariadb-devel
```

```
rpmbuild -ta slurm-22.05.11.tar.bz2
```

### 3. 클러스터 구축

#### Slurm 설치 (tar source 사용)

```
yum --nogpgcheck localinstall slurm-[0-9]*.el*.x86_64.rpm slurm-contribs-*.*.x86_64.rpm slurm-devel-*.*.x86_64.rpm slurm-example-configs-*.*.x86_64.rpm slurm-libpmi-*.*.x86_64.rpm slurm-pam_slurm-*.*.x86_64.rpm slurm-perlapi-*.*.x86_64.rpm slurm-slurmctld-*.*.x86_64.rpm slurm-slurmd-*.*.x86_64.rpm slurm-slurmdbd-*.*.x86_64.rpm -y
```

## 3. 클러스터 구축

### Slurm 설치 (tar source 사용)

```
sudo mkdir /var/spool/slurm
sudo chown slurm:slurm /var/spool/slurm
sudo chown slurm:slurm /var/spool/slurm
sudo chmod 755 /var/spool/slurm
sudo mkdir /var/spool/slurm/slurmctld
sudo chown slurm:slurm /var/spool/slurm/slurmctld
sudo chmod 755 /var/spool/slurm/slurmctld
```

```
sudo mkdir /var/spool/slurm/cluster_state
sudo chown slurm:slurm /var/spool/slurm/cluster_state
mkdir /var/log/slurm
touch /var/log/slurm/slurmctld.log
touch /var/log/slurm/slurmd.log
sudo chown slurm:slurm /var/log/slurm//slurm*
sudo touch /var/log/slurm//slurm_jobacct.log
sudo touch /var/log/slurm/slurm_jobcomp.log
sudo chown slurm:slurm /var/log/slurm//slurm*

sudo chmod 777 -R /var/lib/slurm
```

### 3. 클러스터 구축

#### Slurm 설치 - Slurm config

옵션	역할	실무 예시
ControlMachine	Slurm 컨트롤러 호스트명	ControlMachine=master-node
AuthType	인증 방식, Munge 사용 필수	AuthType=auth/munge
SlurmUser	Slurm 데몬 실행 사용자	SlurmUser=slurm
StateSaveLocation	Slurm 컨트롤러 상태 저장 경로	/var/spool/slurmctld
SlurmdSpoolDir	노드별 Job Spool 디렉토리	/var/spool/slurmd
SelectType	자원 선택 모델	select/cons_res → CPU/메모리/GPU 단위 세부 추적
SelectTypeParameters	자원 추적 방식 세부 설정	CR_Core → Core 단위 추적
SchedulerType	스케줄러 알고리즘	sched/backfill → 빈 자원에 작은 Job 끼워넣기
PriorityType	우선순위 알고리즘	priority/multifactor → Age, FairShare 등 가중치
NodeName	각 노드 정의	NodeName=node[01-02] CPUs=64 State=UNKNOWN
PartitionName	Queue 정의	PartitionName=debug Nodes=node[01-02] Default=YES MaxTime=INFINITE State=UP

### 3. 클러스터 구축

#### Slurm 설치 - Slurm config

옵션	설명
SelectType=select/linear	노드 단위 자원 관리 (단순)
SelectType=select/cons_res	HPC 실전 표준. CPU, Mem, GPU 세부 추적
SchedulerType=sched/backfill	빈 자원에 작은 Job 끼워넣기
PriorityType=priority/multifactor	Fair Share, Age Factor 등

### 3. 클러스터 구축

#### Slurm 설치 – Slurm config

```
#### Slurm config ####
```

```
# slurm.conf file generated by configurator easy.html.
```

```
# Put this file on all nodes of your cluster.
```

```
# See the slurm.conf man page for more information.
```

```
#
```

```
ClusterName=HPC
```

```
SlurmctldHost=master
```

```
#
```

### 3. 클러스터 구축

#### Slurm 설치 – Slurm config

#MailProg=/bin/mail

MpiDefault=none

#MpiParams=ports=#-#

#ProctrackType=proctrack/pgid

ProctrackType=proctrack/cgroup

ReturnToService=1

SlurmctldPidFile=/var/run/slurm/slurmctld.pid

SlurmctldPort=6817

SlurmdPidFile=/var/run/slurm/slurmd.pid

SlurmdPort=6818

### 3. 클러스터 구축

#### Slurm 설치 – Slurm config

```
SlurmdSpoolDir=/var/lib/slurm/slurmd
```

```
SlurmUser=slurm
```

```
#SlurmdUser=root
```

```
StateSaveLocation=/var/lib/slurm/slurmctld
```

```
SwitchType=switch/none
```

```
TaskPlugin=task/affinity
```

```
#
```

```
#
```

## 3. 클러스터 구축

### Slurm 설치 – Slurm config

```
# TIMERS
```

```
#KillWait=30
```

```
#MinJobAge=300
```

```
#SlurmctldTimeout=120
```

```
#SlurmdTimeout=300
```

```
#
```

```
#
```

### 3. 클러스터 구축

#### Slurm 설치 – Slurm config

```
# SCHEDULING  
  
#20240110 SchedulerType=sched/backfill  
  
#20240110 SelectType=select/linear  
  
SelectType=select/cons_res  
  
SelectTypeParameters=CR_Core  
  
#SelectTypeParameters=  
  
#  
  
#
```

## 3. 클러스터 구축

### Slurm 설치 – Slurm config

```
# LOGGING AND ACCOUNTING  
  
#AccountingStorageType=accounting_storage/none  
AccountingStorageType=accounting_storage/slurmdbd  
  
#JobAcctGatherFrequency=30  
  
#JobAcctGatherType=jobacct_gather/none  
#JobAcctGatherType=jobacct_gather/cgroup  
  
#SlurmctldDebug=info  
SlurmctldLogFile=/var/log/slurm/slurmctld.log  
  
#SlurmdDebug=info  
SlurmdLogFile=/var/log/slurm/slurmd.log
```

### 3. 클러스터 구축

#### Slurm 설치 – Slurm config

```
slurmd -C
```

```
# COMPUTE NODES
```

```
NodeName=master NodeAddr=192.168.30.48 CPUs=12 RealMemory=64055 Sockets=1 CoresPerSocket=6  
ThreadsPerCore=2
```

```
NodeName=node01 NodeAddr=192.168.30.27 CPUs=12 RealMemory=64056 Sockets=1 CoresPerSocket=6  
ThreadsPerCore=2
```

```
NodeName=node02 NodeAddr=192.168.30.50 CPUs=12 RealMemory=64056 Sockets=1 CoresPerSocket=6  
ThreadsPerCore=2
```

```
NodeName=node03 NodeAddr=192.168.30.26 CPUs=12 RealMemory=64055 Sockets=1 CoresPerSocket=6  
ThreadsPerCore=2
```

```
PartitionName=jobs Nodes=ALL Default=YES MaxTime=INFINITE State=UP
```

### 3. 클러스터 구축

```
[root@localhost selinux]# sinfo
sinfo: error: If munged is up, restart with --num-threads=10
sinfo: error: Munge encode failed: Failed to access
"/var/run/munge/munge.socket.2": No such file or directory
sinfo: error: slurm_send_node_msg: g_slurm_auth_create:
REQUEST_PARTITION_INFO has authentication error: Invalid
authentication credential
slurm_load_partitions: Protocol authentication error
```

## 3. 클러스터 구축

### Slurm 사용 - 작업 스크립트 작성

```
#!/bin/bash
#SBATCH --job-name=test_job      # 작업 이름
#SBATCH --output=slurm-%j.out    # 출력 파일 (job ID 자동 삽입)
#SBATCH --ntasks=1               # 전체 태스크 수
#SBATCH --cpus-per-task=2        # 태스크당 코어 수
#SBATCH --time=00:01:00          # 최대 실행 시간
#SBATCH --partition=debug        # 실행할 파티션
echo "작업 시작 시간: $(date)"
hostname
sleep 30
echo "작업 종료 시간: $(date)"
```

sbatch job.slurm

## 3. 클러스터 구축

### Slurm 사용 - 작업 스크립트 작성

```
#!/bin/bash
#SBATCH --job-name=test_job          # 작업 이름
#SBATCH --output=slurm-%j.out        # 출력 파일 (job ID 자동 삽입)
#SBATCH --ntasks=1                   # 전체 태스크 수
#SBATCH --mem=1gb                    # Memory limit
#SBATCH --cpus-per-task=2            # 태스크당 코어 수
#SBATCH --time=00:01:00              # 최대 실행 시간
#SBATCH --partition=debug            # 실행할 파티션
echo "작업 시작 시간: $(date)"
hostname
sleep 30
echo "작업 종료 시간: $(date)"
```

sbatch job.slurm

### 3. 클러스터 구축

항목	의미
<code>--nodes=2</code>	2개의 노드를 사용 (총 2대 서버)
<code>--ntasks-per-node=4</code>	각 노드에서 4개의 작업 (MPI 프로세스) 실행
<code>--cpus-per-task=8</code>	각 작업은 8개 CPU 코어 사용 (OpenMP, 쓰레딩 등)
<code>--gres=gpu:h200:2</code>	각 노드에 GPU 2개 할당 → 총 GPU 4개 사용

### 3. 클러스터 구축

항목	쉬운 설명	예시
노드 수	--nodes에서 지정한 전체 컴퓨터(서버) 수	--nodes=2 → 2대 사용
프로세스 수	--ntasks에서 지정한 작업 개수 (CPU 개수)	--ntasks=4 → 4개 실행
각 프로세스당 CPU 수	--cpus-per-task는 하나의 작업이 사용하는 CPU 수	--cpus-per-task=8 → 하나의 작업에 8개 CPU
노드당 코어 사용량	한 노드에서 사용하는 총 CPU 개수	= ntasks-per-node × cpus-per-task
총 CPU 사용량	전체 작업에서 사용하는 CPU 총합	= ntasks × cpus-per-task
GPU 개수	--gres=gpu:xxx:Y에서 Y는 노드당 GPU 수	--gres=gpu:a6000:2 → 각 노드당 GPU 2개
총 GPU 수	= 노드 수 × 노드당 GPU 수	2노드 × 2GPU → 총 4GPU 사용

### 3. 클러스터 구축

```
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 19
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 20
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 21
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 22
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 23
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 24
[ksy@ksy01 ~]$ sbatch job.sh
Submitted batch job 25
[ksy@ksy01 ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
22	ksy	test_job	ksy	PD	0:00	1	(Resources)
23	ksy	test_job	ksy	PD	0:00	1	(Priority)
24	ksy	test_job	ksy	PD	0:00	1	(Priority)
25	ksy	test_job	ksy	PD	0:00	1	(Priority)
19	ksy	test_job	ksy	R	0:06	1	ksy01
20	ksy	test_job	ksy	R	0:03	1	ksy02
21	ksy	test_job	ksy	R	0:03	1	ksy03

```
[ksy@ksy01 ~]$
```

### 3. 클러스터 구축

```
#  
# SCHEDULING  
#DefMemPerCPU=0  
#FastSchedule=1  
#MaxMemPerCPU=0  
#SchedulerTimeSlice=30  
SchedulerType=sched/backfill  
SelectType=select/linear  
#SelectTypeParameters=
```

```
#  
#  
# SCHEDULING  
#DefMemPerCPU=0  
#FastSchedule=1  
#MaxMemPerCPU=0  
#SchedulerTimeSlice=30  
SchedulerType=sched/backfill  
#SelectType=select/linear  
SelectType=select/cons_res  
SelectTypeParameters=CR_Core
```

### 3. 클러스터 구축

#### SelectType=select/cons\_res

Slurm이 작업(job)을 스케줄링할 때 자원을 어떻게 할당할지 결정하는 방식을 지정  
select/cons\_res는 "consumable resources" 모델을 사용하겠다는 의미

Slurm이 CPU, 메모리, GPU 같은 자원을 정량적으로 소비되는(consumable) 자원으로 취급하여, 보다 정밀한 자원 관리가 가능하도록 합니다.

이를 통해 --cpus-per-task, --mem, --gres 등의 요청에 기반하여 자원을 정확히 계산하고 할당합니다.

일반적으로 현대적인 HPC 클러스터에서는 이 방식을 사용합니다.

#### SelectTypeParameters=CR\_Core

select/cons\_res 플러그인의 세부 동작 방식을 지정

CR\_Core는 "core-level granularity" 자원 할당을 의미

Slurm이 CPU core 단위로 자원을 할당합니다.

예: --cpus-per-task=8이라면, Slurm은 작업에 정확히 8개의 core를 할당합니다.

이는 SMT (Simultaneous MultiThreading) 또는 Hyper-Threading이 있을 경우에도 core 단위로 자원을 추적하므로, 보다 정확한 성능 제어가 가능합니다.

### 3. 클러스터 구축

```
[ksy@ksy01 ~]$  
[ksy@ksy01 ~]$ sbatch job.sh  
Submitted batch job 26  
[ksy@ksy01 ~]$ sbatch job.sh  
Submitted batch job 27  
[ksy@ksy01 ~]$ sbatch job.sh  
Submitted batch job 28  
[ksy@ksy01 ~]$ sbatch job.sh  
Submitted batch job 29  
[ksy@ksy01 ~]$ sbatch job.sh  
Submitted batch job 30  
[ksy@ksy01 ~]$ sbatch job.sh  
Submitted batch job 31  
[ksy@ksy01 ~]$ squeue  
      JOBID PARTITION    NAME    USER  ST       TIME  NODES NODELIST(REASON)  
        31         ksy test_job    ksy  PD        0:00      1 (None)  
        26         ksy test_job    ksy   R        0:02      1 ksy01  
        27         ksy test_job    ksy   R        0:02      1 ksy01  
        28         ksy test_job    ksy   R        0:02      1 ksy02  
        29         ksy test_job    ksy   R        0:02      1 ksy02  
        30         ksy test_job    ksy   R        0:02      1 ksy03  
[ksy@ksy01 ~]$ █
```

### 3. 클러스터 구축

### 3. 클러스터 구축

## CPU + GPU 클러스터

- 총 32대의 cpu 클러스터와 8대의 gpu 서버
- CPU는 128코어 클러스터가 26대
- 각 노드는 512GB 메모리로 구성
- 나머지 6대는 64코어, 128GB
- GPU는 4대는 H200 gpu가 4개씩,
- 나머지 4대는 각각 A6000이 4개씩 구성
- GPU서버는 모두 128코어 512GB 메모리

이때 가장 좋은 배치는?

### 3. 클러스터 구축

노드 그룹	수량	CPU 코어	메모리	GPU
CPU-HighMem	26	128	512GB	없음
CPU-LowMem	6	64	128GB	없음
GPU-H200	4	128	512GB	4 × H200
GPU-A6000	4	128	512GB	4 × A6000

### 3. 클러스터 구축

#### 스케줄링 전략 제안 :

- 기본 Partition = cpu-highmem : 일반 계산 작업 자동 배치
- GPU 작업은 --partition=gpu-h200 또는 gpu-a6000으로 명시:  
GPU 종류에 따라 명확한 자원 요청 가능

### 3. 클러스터 구축

```
#----- 노드 정의 -----  
# CPU High-Memory (node-cpu-[001-026])  
NodeName=node-cpu-[001-026] CPUs=128 RealMemory=500000 State=UNKNOWN  
  
# CPU Low-Memory (node-cpu-[027-032])  
NodeName=node-cpu-[027-032] CPUs=64 RealMemory=125000 State=UNKNOWN  
  
# GPU H200 (node-gpu-h200-[001-004])  
NodeName=node-gpu-h200-[001-004] CPUs=128 RealMemory=500000 Gres=gpu:h200:4  
State=UNKNOWN  
  
# GPU A6000 (node-gpu-a6000-[001-004])  
NodeName=node-gpu-a6000-[001-004] CPUs=128 RealMemory=500000 Gres=gpu:a6000:4  
State=UNKNOWN  
  
#----- 파티션 정의 -----  
PartitionName=cpu-highmem Nodes=node-cpu-[001-026] Default=YES MaxTime=INFINITE State=UP  
PartitionName=cpu-lowmem Nodes=node-cpu-[027-032] MaxTime=INFINITE State=UP  
  
PartitionName=gpu-h200 Nodes=node-gpu-h200-[001-004] MaxTime=INFINITE State=UP  
PartitionName=gpu-a6000 Nodes=node-gpu-a6000-[001-004] MaxTime=INFINITE State=UP  
  
# (선택) GPU 전체 통합 파티션  
PartitionName=gpu-all Nodes=node-gpu-* MaxTime=INFINITE State=UP
```

### 3. 클러스터 구축

각 GPU 노드의 /etc/slurm/gres.conf

```
Name=gpu Type=H200 File=/dev/nvidia[0-3]
```

```
Name=gpu Type=A6000 File=/dev/nvidia[0-3]
```

### 3. 클러스터 구축

```
#!/bin/bash
#SBATCH --job-name=cpu_test
#SBATCH --partition=cpu-highmem
#SBATCH --nodes=1
#SBATCH --ntasks=32
#SBATCH --time=01:00:00

srun hostname
```

```
#!/bin/bash
#SBATCH --job-name=gpu_test
#SBATCH --partition=gpu-h200
#SBATCH --gres=gpu:h200:2
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --time=01:00:00

nvidia-smi
```

```
#!/bin/bash
#SBATCH --job-name=cpu_basic      # 작업 이름
#SBATCH --partition=cpu-highmem   # 고용량 메모리 CPU 파티션
#SBATCH --nodes=1                # 노드 1개 사용
#SBATCH --ntasks=64              # 총 64개 작업 (프로세스)
#SBATCH --time=01:00:00          # 최대 실행 시간 1시간
#SBATCH --output=cpu_basic_%j.out # 출력 로그

echo "Running basic CPU test on $SLURM_JOB_NODELIST"
srun hostname
```

항목	값
노드 수	1개 (--nodes=1)
작업 수 (ntasks)	64개 (MPI 또는 병렬 단위)
CPU 수	64개 (1작업 ↔ 1 CPU)
GPU 사용	없음 (--gres 없음)

## 어떤 작업에 적합한가?

이 구성은 GPU 없이, CPU만 사용하는 병렬 작업에 적합합니다:

- OpenMP 병렬 처리 (노드 1개, shared memory 구조)
- MPI 병렬 프로그램 (단일 노드에서 64개 rank 실행)
- 병렬화된 과학/공학 코드 (ex: Fortran, C 기반 해석 코드)
- 배치 데이터 전처리, 대량 텍스트 처리, 컴파일 등

### 3. 클러스터 구축

### 유저 케이스 2: 저메모리 CPU 노드에서의 MPI 병렬 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_lowmem_mpi          # 작업 이름
#SBATCH --partition=cpu-lowmem              # 저메모리 CPU 파티션 사용
#SBATCH --nodes=2                          # 노드 2개 사용
#SBATCH --ntasks-per-node=64               # 각 노드당 64개의 MPI rank
#SBATCH --time=00:30:00                    # 최대 실행 시간 30분
#SBATCH --output=cpu_lowmem_mpi_%j.out     # 출력 로그 파일

module load mpi                            # MPI 환경 모듈 로드
echo "Running MPI job on nodes: $SLURM_JOB_NODELIST"
srun --mpi=pmix ./mpi_app                  # MPI 실행
```

항목	값
노드 수 (--nodes)	2대 사용
노드당 작업 수	64개 (--ntasks-per-node=64)
총 작업 수 (--ntasks)	2 × 64 = 128개 MPI rank
CPU 사용량	1 작업당 1 CPU → 총 128개 코어 사용
GPU 사용	없음 (--gres 없음)
대상 노드 유형	cpu-lowmem 파티션 (64코어, 128GB RAM 노드)

#### 어떤 작업에 적합한가?

이 구성은 저메모리 노드 2대에서 많은 수의 MPI 프로세스를 실행하기 위한 설정입니다:

- MPI 기반 병렬 연산 코드 (Ex: Lattice QCD, 분자동역학, 행렬 연산)
- 단일 스레드 기반의 경량 병렬 처리
- 데이터 분산 처리 실험용 (노드 간 통신 성능 확인)
- OpenMPI / MPICH / IntelMPI 호환

### 3. 클러스터 구축

### 유저 케이스 3: NUMA 최적화를 고려한 OpenMP 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_numa          # 작업 이름
#SBATCH --partition=cpu-highmem      # 고성능 CPU 노드
#SBATCH --nodes=1                    # 노드 1개 사용
#SBATCH --ntasks=1                   # 작업 1개만 실행
#SBATCH --cpus-per-task=64           # 해당 작업에 CPU 64개 할당
#SBATCH --time=01:00:00              # 최대 실행 시간
#SBATCH --output=cpu_numa_%j.out     # 출력 로그 파일

# OpenMP 환경 설정
export OMP_NUM_THREADS=64
export OMP_PROC_BIND=close
export OMP_PLACES=cores

# NUMA 정보 출력
echo "Running on: $SLURM_JOB_NODELIST"
numactl --show

# 실행
srun ./openmp_app
```

### 3. 클러스터 구축

### 유저 케이스 3: NUMA 최적화를 고려한 OpenMP 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_numa           # 작업 이름
#SBATCH --partition=cpu-highmem       # 고성능 CPU 노드
#SBATCH --nodes=1                     # 노드 1개 사용
#SBATCH --ntasks=1                   # 작업 1개만 실행
#SBATCH --cpus-per-task=64            # 해당 작업에 CPU 64개 할당
#SBATCH --time=01:00:00               # 최대 실행 시간
#SBATCH --output=cpu_numa_%j.out      # 출력 로그 파일
```

```
# OpenMP 환경 설정
export OMP_NUM_THREADS=1
export OMP_PROC_BIND=1
export OMP_PLACES=1
```

```
# NUMA 정보 출력
echo "Running on: $(cat /proc/cpuinfo | grep -m1 -E '^physical id$' | sed 's/^physical id=//;s/.$//;s/ /_/g')
numactl --show
```

```
# 실행
srun ./openmp_app
```

항목	값
노드 수 (--nodes)	1개 (단일 노드에서 실행)
작업 수 (--ntasks)	1개 (작업 하나만 실행)
각 작업의 CPU 수	64개 (--cpus-per-task=64) → 멀티스레드 구조
총 CPU 사용량	64개 코어 (1작업 × 64코어)
GPU 사용 여부	×없음
대상 노드 유형	cpu-highmem (128코어, 512GB 메모리) 노드

어떤 작업에 적합한가?

이 구성은 다음과 같은 NUMA 최적화 OpenMP 기반 대규모 단일 작업에 적합합니다:

- OpenMP 또는 멀티스레드 프로그램
- 하나의 작업에 많은 CPU 자원이 필요한 경우
- NUMA 구조에서 캐시 로컬리티 유지가 중요한 경우
- 비동기성 적고, 공유 메모리를 적극 활용하는 작업

예:

구조해석/열전달 FEM (단일 해석 영역)

포렌식 시뮬레이션

이미지 처리 (대규모 FFT/OpenCV 필터 등)

### 3. 클러스터 구축

### 유저 케이스 4: 단일 GPU(H200) + CPU 병렬 작업

```
#!/bin/bash
#SBATCH --job-name=gpu_h200_test      # 작업 이름
#SBATCH --partition=gpu-h200          # H200 GPU 전용 파티션
#SBATCH --nodes=1                     # 노드 1개 사용
#SBATCH --ntasks=1                    # 작업 1개 (GPU 1개 사용)
#SBATCH --cpus-per-task=16            # 해당 작업에 CPU 16개 할당
#SBATCH --gres=gpu:h200:1             # H200 GPU 1개 요청
#SBATCH --time=01:00:00               # 최대 실행 시간
#SBATCH --output=gpu_h200_test_%j.out # 출력 로그 파일

module load cuda                      # CUDA 환경 모듈 로드

echo "Running on node: $SLURM_JOB_NODELIST"
nvidia-smi                           # GPU 정보 출력
srun ./cuda_app                       # 실행할 CUDA 프로그램
```

### 3. 클러스터 구축

### 유저 케이스 4: 단일 GPU(H200) + CPU 병렬 작업

```
#!/bin/bash
#SBATCH --job-name=gpu_h200_test      # 작업 이름
#SBATCH --partition=gpu-h200          # H200 GPU 전용 파티션
#SBATCH --nodes=1                     # 노드 1개 사용
#SBATCH --ntasks=1                    # 작업 1개 (GPU 1개 사용)
#SBATCH --cpus-per-task=16            # 해당 작업에 CPU 16개 할당
#SBATCH --gres=gpu:h200:1             # H200 GPU 1개 요청
#SBATCH --time=01:00:00               # 최대 실행 시간
#SBATCH --output=gpu_h200_test_%j.out  # 출력 로그 파일

module load cuda

echo "Running on node"
nvidia-smi
srun ./cuda_app
```

항목	값
노드 수	1개 (--nodes=1)
작업 수	1개 (--ntasks=1)
각 작업의 CPU 수	16개 (--cpus-per-task=16)
총 CPU 사용량	16개 코어
GPU 요청	H200 1개 (--gres=gpu:h200:1)
대상 노드 유형	gpu-h200 (GPU 4개 + CPU 128코어) 노드
병렬 방식	GPU 1개 + 멀티스레드 CPU 16개

#### 어떤 작업에 적합한가?

이 구성은 단일 GPU를 사용하고, 비교적 많은 CPU를 병렬 보조 연산에 사용하는 구조로 다음과 같은 작업에 적합합니다:

- PyTorch/TensorFlow 단일 GPU 학습 또는 추론
- CUDA 커널 기반 시뮬레이션 (1 GPU)  
예: 유체/전산역학, 입자 시스템, Ray-tracing 등
- CPU 보조 계산이 큰 이미지 처리/포인트클라우드 처리
- OpenCV + CUDA 하이브리드 처리
- 단일 데이터 배치 처리 (1GPU + 멀티스레드 CPU)

### 3. 클러스터 구축

### 유저 케이스 5: 멀티 GPU (A6000 × 4개) 병렬 작업

```
#!/bin/bash
#SBATCH --job-name=gpu_a6000_multi      # 작업 이름
#SBATCH --partition=gpu-a6000           # A6000 GPU 파티션
#SBATCH --nodes=1                       # 노드 1개 사용
#SBATCH --ntasks=4                      # 작업 4개 (rank 4개)
#SBATCH --cpus-per-task=16              # 각 작업에 CPU 16개 할당
#SBATCH --gres=gpu:a6000:4              # GPU 4개 요청
#SBATCH --time=02:00:00                 # 최대 실행 시간
#SBATCH --output=gpu_a6000_multi_%j.out  # 출력 로그

module load cuda nccl                  # CUDA + NCCL 모듈 로드
export NCCL_DEBUG=INFO                  # 통신 디버그 정보 출력

srun ./multi_gpu_nccl_test              # 실행 (예: NCCL 테스트, PyTorch DDP 등)
```

### 3. 클러스터 구축

### 유저 케이스 5: 멀티 GPU (A6000 × 4개) 병렬 작업

```
#!/bin/bash
#SBATCH --job-name=gpu_a6000_multi      # 작업 이름
#SBATCH --partition=gpu-a6000           # A6000 GPU 파티션
#SBATCH --nodes=1                       # 노드 1개 사용
#SBATCH --ntasks=4                      # 작업 4개 (rank 4개)
#SBATCH --cpus-per-task=16              # 각 작업에 CPU 16개 할당
#SBATCH --gres=gpu:a6000:4             # GPU 4개 요청
#SBATCH --time=02:00:00                 # 최대 실행 시간
#SBATCH --output=gpu_a6000_multi_%j.out # 출력 로그
```

```
module load cuda nccl
export NCCL_DEBUG=INFO
srun ./multi_gpu_nccl_test
```

항목	값
노드 수 (--nodes)	1개
작업 수 (--ntasks)	4개 (4개의 rank 또는 프로세스)
각 작업의 CPU 수	16개 (--cpus-per-task=16)
총 CPU 사용량	4 × 16 = <b>64개 코어</b>
GPU 요청	A6000 GPU 4개 (--gres=gpu:a6000:4)
대상 노드	gpu-a6000 노드 (GPU 4개, CPU 128코어)
병렬 방식	GPU 4개 ↔ rank 4개 1:1 매핑 (분산 학습, DDP 등)

어떤 작업에 적합한가?

이 구성은 단일 노드 멀티 GPU 작업에 최적화되어 있으며 다음과 같은 작업에 적합합니다:

작업 유형	설명
PyTorch DDP	분산 데이터 병렬 훈련 (DataParallel → DistributedDataParallel)
TensorFlow Horovod	Horovod 기반의 멀티 GPU 학습
NCCL 테스트	all_reduce_perf, broadcast_perf
멀티-GPU 렌더링/영상처리	CUDA 기반 멀티 스트림 처리
GROMACS / LAMMPS (GPU)	단일 노드에서 멀티 GPU 병렬 계산 (입자/분자동역학)

### 3. 클러스터 구축

### 유저 케이스 6: CPU + GPU 하이브리드 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_gpu_hybrid      # 작업 이름
#SBATCH --partition=gpu-h200           # H200 GPU 파티션
#SBATCH --nodes=1                      # 노드 1개 사용
#SBATCH --ntasks=2                     # 작업 2개 (예: MPI rank 2개)
#SBATCH --cpus-per-task=32             # 각 작업당 CPU 32개 사용
#SBATCH --gres=gpu:h200:2              # GPU 2개 요청 (1작업당 1GPU)
#SBATCH --time=01:30:00                # 최대 실행 시간
#SBATCH --output=cpu_gpu_hybrid_%j.out # 출력 로그

module load mpi cuda                    # MPI와 CUDA 환경 로드

echo "Running hybrid CPU+GPU job on $SLURM_JOB_NODELIST"

srun ./hybrid_mpi_cuda_app              # 실행 파일 (MPI + CUDA)
```

### 3. 클러스터 구축

### 유저 케이스 6: CPU + GPU 하이브리드 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_gpu_hybrid          # 작업 이름
#SBATCH --partition=gpu-h200                # H200 GPU 파티션
#SBATCH --nodes=1                          # 노드 1개 사용
#SBATCH --ntasks=2                         # 작업 2개 (예: MPI rank 2개)
#SBATCH --cpus-per-task=32                 # 각 작업당 CPU 32개 사용
#SBATCH --gres=gpu:h200:2                  # GPU 2개 요청 (1작업당 1GPU)
#SBATCH --time=01:30:00                    # 최대 실행 시간
#SBATCH --output=cpu_gpu_hybrid_%j.out     # 출력 로그

module load mpi cuda                        # MPI와 CUDA 환경 로드
```

module load mpi cuda	항목	값
echo "Running hybrid"	노드 수	1개 (--nodes=1)
srun ./hybrid_mpi_c	작업 수 (--ntasks)	2개 (예: 2개의 MPI rank)
	각 작업의 CPU 수	32개 (--cpus-per-task=32)
	총 CPU 사용량	2 × 32 = <b>64개 코어</b>
	GPU 요청	H200 2개 (--gres=gpu:h200:2)
	GPU ↔ Rank 매핑 구조	각 rank가 1개 GPU를 사용 (1:1 매핑)
	대상 노드 유형	gpu-h200 (128코어, 512GB RAM, H200 GPU 4개)

어떤 작업에 적합한가?

이 구성은 각 rank가 CPU와 GPU를 동시에 사용하는 다음과 같은 작업에 적합합니다::

유형	예시 애플리케이션
MPI + CUDA 하이브리드	유체/열전달/기계 시뮬레이션 코드
OpenACC + MPI	기상/기후모델, WRF, MOM6
DNN Inference Pipeline	CPU로 전처리 + GPU로 추론 (ex: BERT, LLM inference)
CUDA 시뮬레이션	Ray tracing, 영상 처리, 입자 시뮬레이션
과학계산	QUADA, LAMMPS, CP2K (MPI + GPU)

### 3. 클러스터 구축

### 유저 케이스 7: NUMA 최적화 CPU + GPU 연계 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_gpu_numa          # 작업 이름
#SBATCH --partition=gpu-a6000            # A6000 GPU 파티션
#SBATCH --gres=gpu:a6000:2              # GPU 2개 요청
#SBATCH --nodes=1                       # 노드 1개 사용
#SBATCH --ntasks=2                      # 작업 2개 (2개의 rank 또는 프로세스)
#SBATCH --cpus-per-task=32              # 각 작업에 CPU 32개 할당
#SBATCH --time=01:00:00                 # 최대 실행 시간
#SBATCH --output=cpu_gpu_numa_%j.out     # 출력 로그 파일

module load cuda

# 각 작업을 NUMA 노드별로 CPU + GPU 묶어서 실행
numactl --cpunodebind=0 --membind=0 ./gpu_app --gpu-id 0 &
numactl --cpunodebind=1 --membind=1 ./gpu_app --gpu-id 1 &

wait # 두 작업 모두 종료될 때까지 대기
```

### 3. 클러스터 구축

### 유저 케이스 7: NUMA 최적화 CPU + GPU 연계 작업

```
#!/bin/bash
#SBATCH --job-name=cpu_gpu_numa          # 작업 이름
#SBATCH --partition=gpu-a6000             # A6000 GPU 파티션
#SBATCH --gres=gpu:a6000:2               # GPU 2개 요청
#SBATCH --nodes=1                        # 노드 1개 사용
#SBATCH --ntasks=2                       # 작업 2개 (2개의 rank 또는 프로세스)
#SBATCH --cpus-per-task=32               # 각 작업에 CPU 32개 할당
#SBATCH --time=01:00:00                  # 최대 실행 시간
#SBATCH --output=cpu_gpu_numa_%j.out     # 출력 로그 파일
```

module load cuda

```
# 각 작업을 NUMA 노드별
numactl --cpunodebind=0
numactl --cpunodebind=1
wait # 두 작업 모두 종료
```

항목	값
노드 수	1개
작업 수 (ntasks)	2개 (&로 백그라운드 병렬 실행)
각 작업의 CPU 수	32개 (--cpus-per-task=32)
총 CPU 사용량	2 × 32 = 64개 코어 사용
GPU 요청	2개 (A6000 GPU 2개 → GPU 0, GPU 1)
NUMA 바인딩 적용	YES – numactl --cpunodebind=X --membind=X
목적	CPU ↔ GPU 간 <b>메모리 근접성 최적화</b>

어떤 작업에 적합한가?

이 구성은 CPU와 GPU가 1:1로 묶여 있고, NUMA 바인딩을 직접 지정할 수 있을 때 성능이 크게 향상되는 다음과 같은 작업에 적합합니다:

작업 유형	설명
OpenMP + CUDA 하이브리드	병렬 CPU 연산과 GPU 연산을 동시에 수행
Deep Learning Inference	CPU가 입력 전처리, GPU가 추론 수행
영상/영상 스트림 처리	CPU와 GPU가 메모리를 공유하며 연산 분담
CUDA 시뮬레이션	메모리 일관성과 병렬 스트리밍 최적화
HPC 수치 해석	CPU-GPU 연계 반복 계산 (ex: Poisson solver 등)

### 3. 클러스터 구축

### 유저 케이스 8: GPU + OpenMP (CPU) 병렬 영상처리

```
#!/bin/bash
#SBATCH --job-name=opencv_gpu_omp          # 작업 이름
#SBATCH --partition=gpu-a6000              # A6000 GPU 파티션
#SBATCH --gres=gpu:a6000:1                # GPU 1개 요청
#SBATCH --nodes=1                          # 노드 1개 사용
#SBATCH --ntasks=1                        # 작업 1개
#SBATCH --cpus-per-task=16                 # CPU 16개 할당
#SBATCH --time=00:45:00                   # 최대 실행 시간
#SBATCH --output=opencv_gpu_omp_%j.out     # 출력 로그

module load cuda opencv                  # CUDA와 OpenCV 모듈 로드

# OpenMP 설정
export OMP_NUM_THREADS=16
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

# 실행
srun ./cv_inference_gpu_openmp          # OpenCV + CUDA + OpenMP 기반 프로그램
```

### 3. 클러스터 구축

### 유저 케이스 8: GPU + OpenMP (CPU) 병렬 영상처리

```
#!/bin/bash
#SBATCH --job-name=opencv_gpu_omp          # 작업 이름
#SBATCH --partition=gpu-a6000              # A6000 GPU 파티션
#SBATCH --gres=gpu:a6000:1                # GPU 1개 요청
#SBATCH --nodes=1                          # 노드 1개 사용
#SBATCH --ntasks=1                        # 작업 1개
#SBATCH --cpus-per-task=16                 # CPU 16개 할당
#SBATCH --time=00:45:00                   # 최대 실행 시간
#SBATCH --output=opencv_gpu_omp_%j.out     # 출력 로그
```

```
module load cuda/cuda-11.2.0              # CUDA와 OpenCV 모듈 로드
```

항목	값
# OpenMP	노드 수
export OM	작업 수
export OM	각 작업의 CPU 수
export OM	총 CPU 사용량
# 실행	GPU 요청
srun ./cv_i	CPU 병렬 처리 방식
	대상 작업 유형

어떤 작업에 적합한가?

이 구성은 다음과 같은 영상처리 및 컴퓨터비전 GPU+CPU 하이브리드 작업에 적합합니다

작업 유형	설명
OpenCV + CUDA 영상 처리	GPU로 필터, 추론, 트래킹, 디코딩 등 처리
OpenCV + TensorRT	추론은 GPU에서, 전처리/후처리는 CPU OpenMP로
Real-time Video Pipeline	CPU 스레드로 병렬 I/O + GPU로 처리
영상 배치 처리 (Batch Inference)	여러 영상 프레임을 병렬 처리
Robotics, 자율주행, CCTV	고속 분석과 저지연 연산을 위한 구성

### 3. 클러스터 구축

### 유저 케이스 9: Slurm Job Array

```
#!/bin/bash
#SBATCH --job-name=job_array_test           # 작업 이름
#SBATCH --partition=cpu-highmem             # CPU 고메모리 파티션 사용
#SBATCH --array=1-10                       # Job Array 인덱스: 1~10번까지 10개 실행
#SBATCH --ntasks=1                         # 작업 1개 (단일 프로세스)
#SBATCH --cpus-per-task=4                  # CPU 4개 할당
#SBATCH --time=00:15:00                   # 최대 실행 시간
#SBATCH --output=job_array_%A_%a.out       # %A: JobID, %a: Array Index

echo "Running job array task: ${SLURM_ARRAY_TASK_ID}"
srun ./experiment_run.sh ${SLURM_ARRAY_TASK_ID}
```

항목	값
총 실행 횟수	10번 (Array: 1 ~ 10)
각 실행당 CPU 사용량	4코어 (--cpus-per-task=4)
동시에 몇 개 실행?	기본은 Slurm 설정 (MaxArraySize 등)에 따름
전체 최대 자원 사용량	(동시 실행 수) × 4코어
GPU 사용	없음

## 어떤 작업에 적합한가?

같은 코드에 다른 파라미터를 넘겨 반복 실행하거나, 반복 실행할 때 매우 효율적입니다:

활용 사례	설명
머신러닝 실험 반복	하이퍼파라미터 sweep (예: learning rate 변화)
데이터셋 분할 처리	task_id를 인덱스로 사용하여 파일 슬라이스 처리
이미지/비디오 배치 처리	image_\${SLURM_ARRAY_TASK_ID}.jpg 처리 등
수치해석 시뮬레이션 반복	조건 변화에 따른 여러 시나리오 실행
단순 반복 테스트/디버깅	반복실행으로 병렬 디버깅 가능

### 3. 클러스터 구축

## 유저 케이스 10: Slurm 기반 MPI hostfile 자동 생성 예제

```
#!/bin/bash
#SBATCH --job-name=mpi_hostfile          # 작업 이름
#SBATCH --partition=cpu-highmem          # CPU 고메모리 노드 사용
#SBATCH --nodes=2                        # 노드 2개 사용
#SBATCH --ntasks-per-node=64             # 노드당 64개의 rank 실행
#SBATCH --time=01:00:00                  # 최대 실행 시간
#SBATCH --output=mpi_hostfile_%j.out      # 출력 로그

module load mpi/openmpi                  # 또는 intelmpi 등

# 1. Slurm에서 할당된 노드 리스트를 기반으로 hostfile 생성
echo "Generating MPI hostfile from $SLURM_NODELIST"
scontrol show hostname $SLURM_NODELIST > hostfile

# 2. 각 노드에 사용될 슬롯 수 지정 (예: 64 rank per node)
sed -i "s/$/:$SLURM_NTASKS_PER_NODE/" hostfile

echo "Generated hostfile:"
cat hostfile

# 3. MPI 실행 (예: OpenMPI)
mpirun --hostfile hostfile -np $SLURM_NTASKS ./mpi_app
```

```
#!/bin/bash
#SBATCH --job-name=mpi_hostfile          # 작업 이름
#SBATCH --partition=cpu-highmem          # CPU 고메모리 노드 사용
#SBATCH --nodes=2                        # 노드 2개 사용
#SBATCH --ntasks-per-node=64            # 노드당 64개의 rank 실행
#SBATCH --time=01:00:00                 # 최대 실행 시간
#SBATCH --output=mpi_hostfile_%j.out     # 출력 로그

module load mpi/openmpi                  # 또는 intelmpi 등

# 1. Slurm에서 할당된 노드 리스트를 기반으로 hostfile 생성
echo "Generating MPI hostfile from $SLURM_NODELIST"
scontrol show hostname $SLURM_NODELIST > hostfile

# 2. 각 노드에 사용될 슬롯 수 지정 (예: 64 rank per node)
sed -i "s/#!/:/$SLURM_NTASKS_PER_NODE/" hostfile
```

```
echo "Generated hostfile"
cat hostfile

# 3. MPI 실행 (예: OpenMPI)
mpirun --hostfile hostfile
```

항목	값
노드 수	2개 (--nodes=2)
작업 수 (총 rank 수)	2 × 64 = 128개 (--ntasks-per-node=64)
각 노드에서 CPU 사용	64개
GPU 사용	없음
hostfile 사용 목적	Slurm이 아닌 mpirun 기반 실행 제어

### 3. 클러스터 구축

## 유저 케이스 11: Slurm 기반 GPU 클러스터에서 MPI + GPU 실행용 hostfile 자동 생성

```
#!/bin/bash
#SBATCH --job-name=mpi_gpu_hostfile      # 작업 이름
#SBATCH --partition=gpu-h200             # GPU H200 전용 파티션
#SBATCH --nodes=2                        # 노드 2개 사용
#SBATCH --ntasks-per-node=4              # 각 노드에서 4개의 MPI rank 실행
#SBATCH --gres=gpu:h200:4                # 각 노드당 GPU 4개 요청
#SBATCH --cpus-per-task=32               # 각 rank에 CPU 32개 할당
#SBATCH --time=01:30:00                  # 최대 실행 시간
#SBATCH --output=mpi_gpu_hostfile_%j.out # 출력 로그
```

```
module load cuda mpi/openmpi
```

```
# 1. Slurm에서 할당된 노드 리스트 기반 hostfile 생성
echo "Generating hostfile from SLURM_NODELIST..."
scontrol show hostname $SLURM_NODELIST > hostfile
```

```
# 2. 각 노드당 실행할 rank 수 추가
sed -i "s/$/:$SLURM_NTASKS_PER_NODE/" hostfile
```

```
echo "Generated hostfile:"
cat hostfile
```

```
# 3. 실행 (GPU-Aware MPI or CUDA + MPI 통합 코드)
mpirun --hostfile hostfile -np $SLURM_NTASKS ./gpu_mpi_app
```

### 3. 클러스터 구축

## 유저 케이스 11: Slurm 기반 GPU 클러스터에서 MPI + GPU 실행용 hostfile 자동 생성

```
#!/bin/bash
#SBATCH --job-name=mpi_gpu_hostfile      # 작업 이름
#SBATCH --partition=gpu-h200             # GPU H200 전용 파티션
#SBATCH --nodes=2                        # 노드 2개 사용
#SBATCH --ntasks-per-node=4              # 각 노드에서 4개의 MPI rank 실행
#SBATCH --gres=gpu:h200:4                # 각 노드당 GPU 4개 요청
#SBATCH --cpus-per-task=32                # 각 rank에 CPU 32개 할당
#SBATCH --time=01:30:00                  # 최대 실행 시간
#SBATCH --output=mpi_gpu_hostfile_%j.out  # 출력 로그
```

```
module load cuda mpi/openmpi
```

```
# 1. Slurm에서 할당된 노드
echo "Generating hostfile"
scontrol show hostname $SLURM_JOB_NODELIST
```

```
# 2. 각 노드당 실행할 rank 수
sed -i "s/#!/:/$SLURM_NTASKS/"
```

```
echo "Generated hostfile:"
cat hostfile
```

항목	값
노드 수 (--nodes)	2개
작업 수 (--ntasks)	2노드 × 4 = <b>8개 rank</b>
각 작업의 CPU 수	32개 (--cpus-per-task=32)
총 CPU 사용량	8 × 32 = <b>256개 코어</b>
GPU 사용량	각 노드 4개 → 2노드 × 4 = <b>8개 H200 GPU</b>
사용된 명령	mpirun + Slurm 기반 hostfile 직접 지정

```
# 3. 실행 (GPU-Aware MPI or CUDA + MPI 통합 코드)
mpirun --hostfile hostfile -np $SLURM_NTASKS ./gpu_mpi_app
```

### 3. 클러스터 구축

# 03 클러스터 구축

모니터링 - ganglia



# 3. 클러스터 구축

## 모니터링 – GANGLIA



Cluster



### 3. 클러스터 구축

#### 모니터링 – GANGLIA

```
yum install epel-release
```

```
yum install ganglia rrdtool ganglia-gmetad ganglia-gmond ganglia-web
```

rrdtool라운드 로빈 데이터베이스 (Round-Robin Database)는 그래프를 사용하여 시간에 따른 데이터 변화를 저장하고 표시하는 데 사용되는 도구입니다.

ganglia-gmetad 모니터링 하려는 호스트 에서 모니터링 데이터를 수집하는 데몬입니다 . 해당 호스트와 마스터 노드 에서 ganglia-gmond (모니터링 데몬 자체) 를 설치 해야 합니다.

ganglia-web 모니터링되는 시스템에 대한 히스토리 그래프 및 데이터를 볼 수있는 웹 프론트 엔드를 제공합니다.

## 3. 클러스터 구축

### 모니터링 – GANGLIA

Ganglia 웹 인터페이스 ( / usr / share / ganglia )에 대한 인증을 설정

/etc/httpd/auth.basic에adminganglia 저장 되는 사용자 이름 으로 비밀번호를 지정하고 /etc/httpd/auth.basic에 저장 (아파치가 읽을 수 있는 다른 디렉토리 및 / 또는 파일 이름을 자유롭게 선택 가능)

```
htpasswd -c /etc/httpd/auth.basic adminganglia
```

## 3. 클러스터 구축

### 모니터링 - GANGLIA

/etc/httpd/conf.d/ganglia.conf 를 수정

```
#  
# Ganglia monitoring system php web frontend  
#  
  
Alias /ganglia /usr/share/ganglia  
<Location /ganglia>  
    AuthType basic  
    AuthName "Ganglia web UI"  
    AuthBasicProvider file  
    AuthUserFile "/etc/httpd/auth.basic"  
    Require user adminganglia  
</Location>
```

## 3. 클러스터 구축

### 모니터링 - GANGLIA

/etc/ganglia/gmetad.conf 수정

- gridname 이름 설정
- gridname "KCLE Cluster"
- data\_source 와 클러스터 (서버 그룹)의 설명 이름, 초 단위의 폴링 간격 및 마스터 및 모니터링 된 노드의 IP 주소 또는 이름을 지정
- data\_source "my cluster" 10 kcle001 kcle002

## 3. 클러스터 구축

### 모니터링 – GANGLIA–Master

/etc/ganglia/gmond.conf를 편집

cluster block 수정

```
cluster {  
  name = "Cluster"  
  owner = "unspecified"  
  latlong = "unspecified"  
  url = "unspecified"  
}
```

## 3. 클러스터 구축

### 모니터링 – GANGLIA-Master

/etc/ganglia/gmond.conf를 편집

udp\_send\_chanel & udp\_recv\_chanel 수정

```
udp_send_channel {
    #bind_hostname = yes # Highly recommended,
    # soon to be default.
    # This option tells gmond to use a
    # source address
    # that resolves to the machine's
    # hostname. Without
    # this, the metrics may appear to
    # come from any
    # interface and the DNS names
    # associated with
    # those IPs will be used to create the
    # RRDs.
    mcast_join = 239.2.11.71
    mcast_if = ib0
    port = 8649
    ttl = 1
}
```

```
/* You can specify as many udp_recv_channels as you like as well. */
udp_recv_channel {
    mcast_join = 239.2.11.71
    mcast_if = ib0
    port = 8649
    bind = 239.2.11.71
    retry_bind = true
    # Size of the UDP buffer. If you are handling lots of metrics you
    # really
    # should bump it up to e.g. 10MB or even higher.
    # buffer = 10485760
}
```

## 3. 클러스터 구축

### 모니터링 – GANGLIA-CLUSTER

/etc/ganglia/gmond.conf를 편집

cluster block 수정

```
cluster {  
  name = "Cluster"  
  owner = "unspecified"  
  latlong = "unspecified"  
  url = "unspecified"  
}
```

## 3. 클러스터 구축

### 모니터링 – GANGLIA-CLUSTER

/etc/ganglia/gmond.conf를 편집

udp\_send\_chanel & udp\_recv\_chanel 수정

```
udp_send_channel {
    #bind_hostname = yes # Highly recommended,
    # soon to be default.
    # This option tells gmond to use a
    # source address
    # that resolves to the machine's
    # hostname. Without
    # this, the metrics may appear to
    # come from any
    # interface and the DNS names
    # associated with
    # those IPs will be used to create the
    # RRDs.
    mcast_join = 239.2.11.71
    mcast_if = ib0
    port = 8649
    ttl = 1
}
```

```
/* You can specify as many udp_recv_channels as you like as well. */
udp_recv_channel {
    mcast_join = 239.2.11.71
    mcast_if = ib0
    port = 8649
    bind = 239.2.11.71
    retry_bind = true
    # Size of the UDP buffer. If you are handling lots of metrics you
    # really
    # should bump it up to e.g. 10MB or even higher.
    # buffer = 10485760
}
```

## 3. 클러스터 구축

### 모니터링 - GANGLIA

```
systemctl restart httpd gmetad gmond  
systemctl enable httpd gmetad gmond
```

```
systemctl restart gmond  
systemctl enable gmond
```

# 3. 클러스터 구축

## 모니터링 - GANGLIA

Ganglia - 접속 <https://xxx.xxx.xxx.xxx/ganglia>

