



THE OPENSOURCE CLUSTER



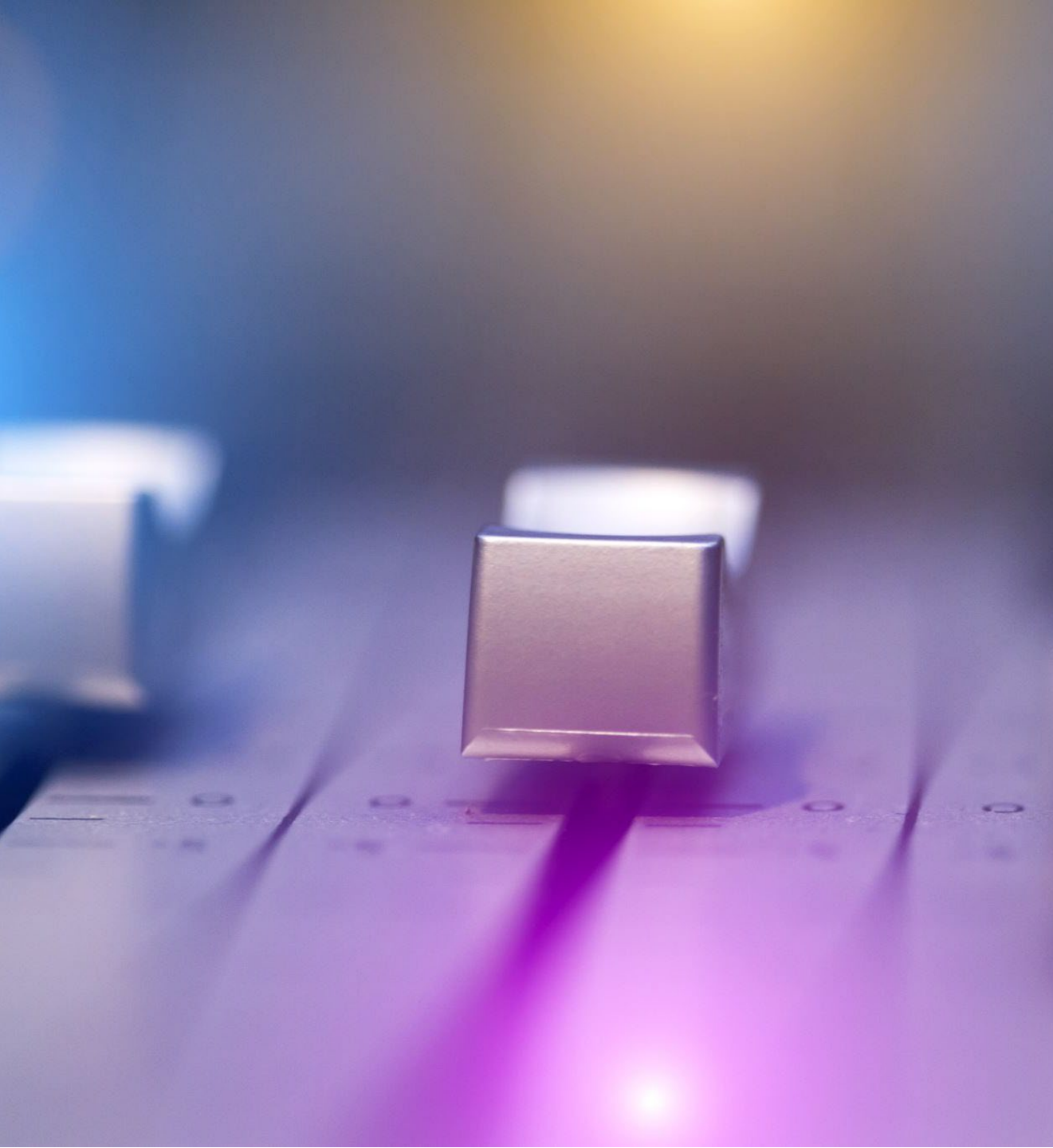
What is the High Availability.

- The Linux Distribution(RHEL, CentOS/, Rocky and SuSE)
- High Availability System for Linux
- Introduce of Pacemaker(ClusterLabs)

INSTALLATION AND BASIC COMMAND

- Virtual Machine
- Quick View to basic command

DAY 1



INSTALL PACEMAKER

- INSTALL PACEMAKER
- ISCSI SETUP
- CLUSTER SETUP

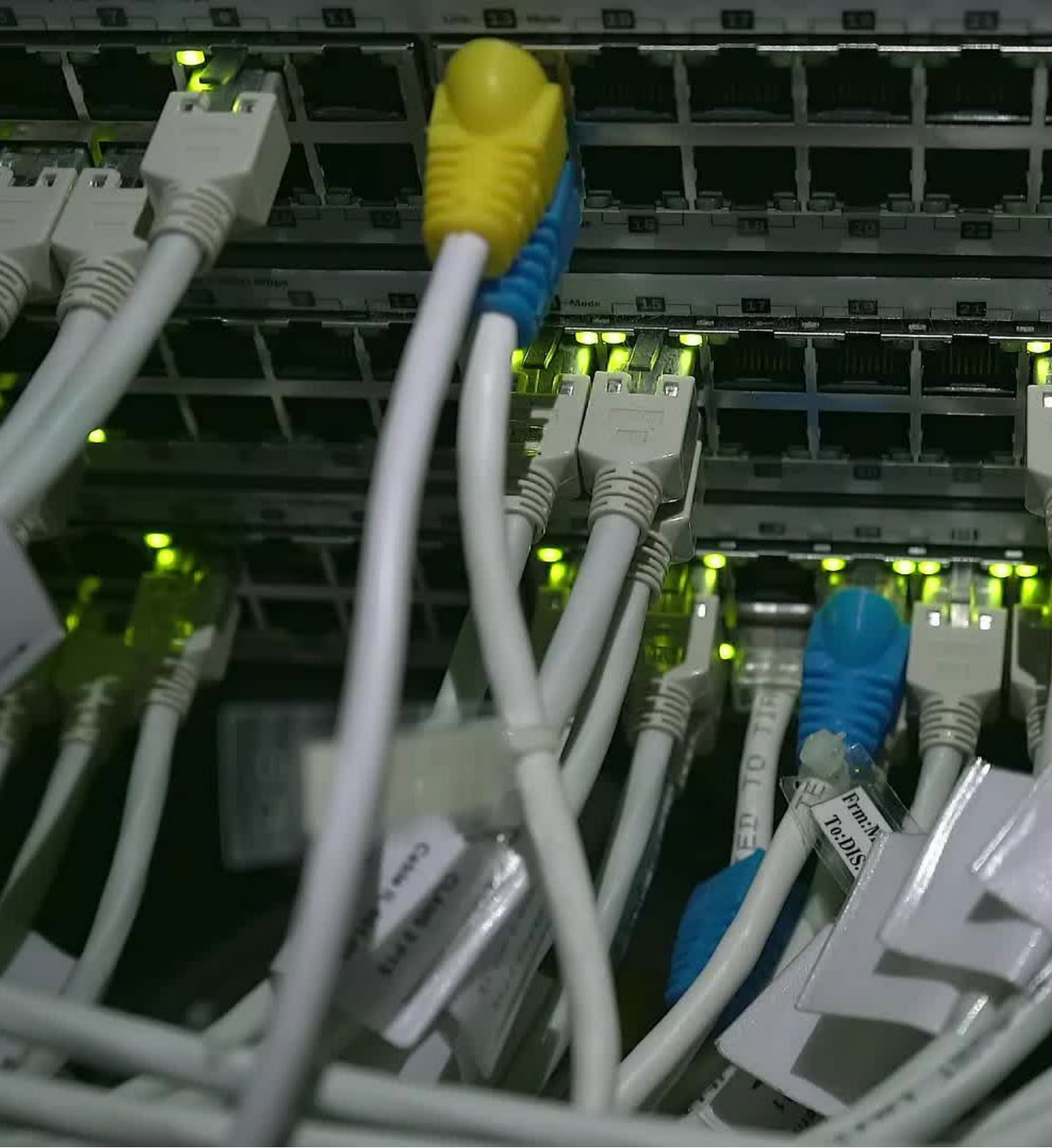
DAY 2



BUILD AND CONFIGURE SERVICE

- ISCSI
- NFS
- GFS2
- WWW

DAY 3



BUILD AND CONFIGURE SERVICE

- ISCSI
- NFS
- GFS2
- WWW

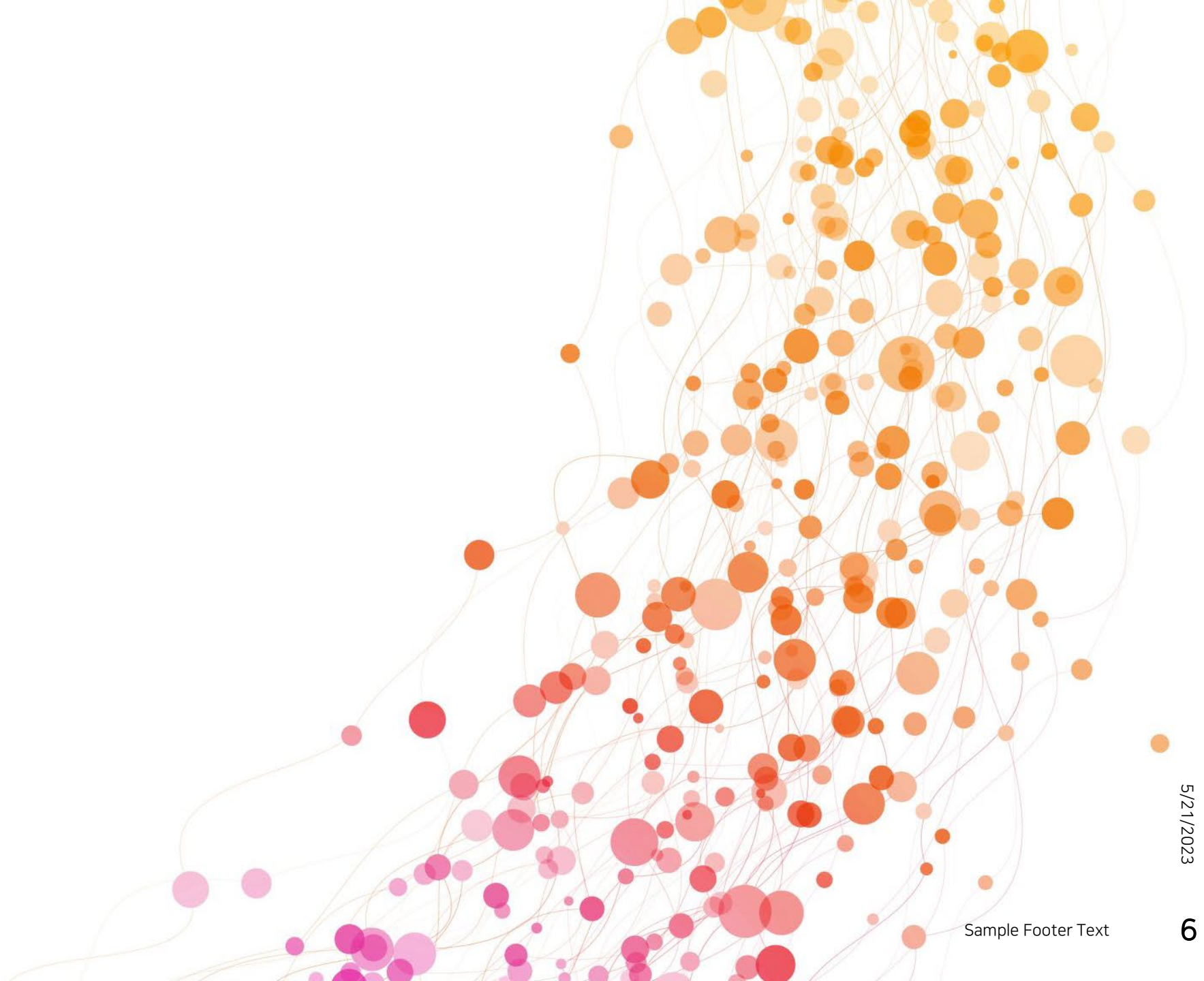
ADD/REMOVE NODE TO CLUSTER

TWO NODE CLUSTER

DAY 4

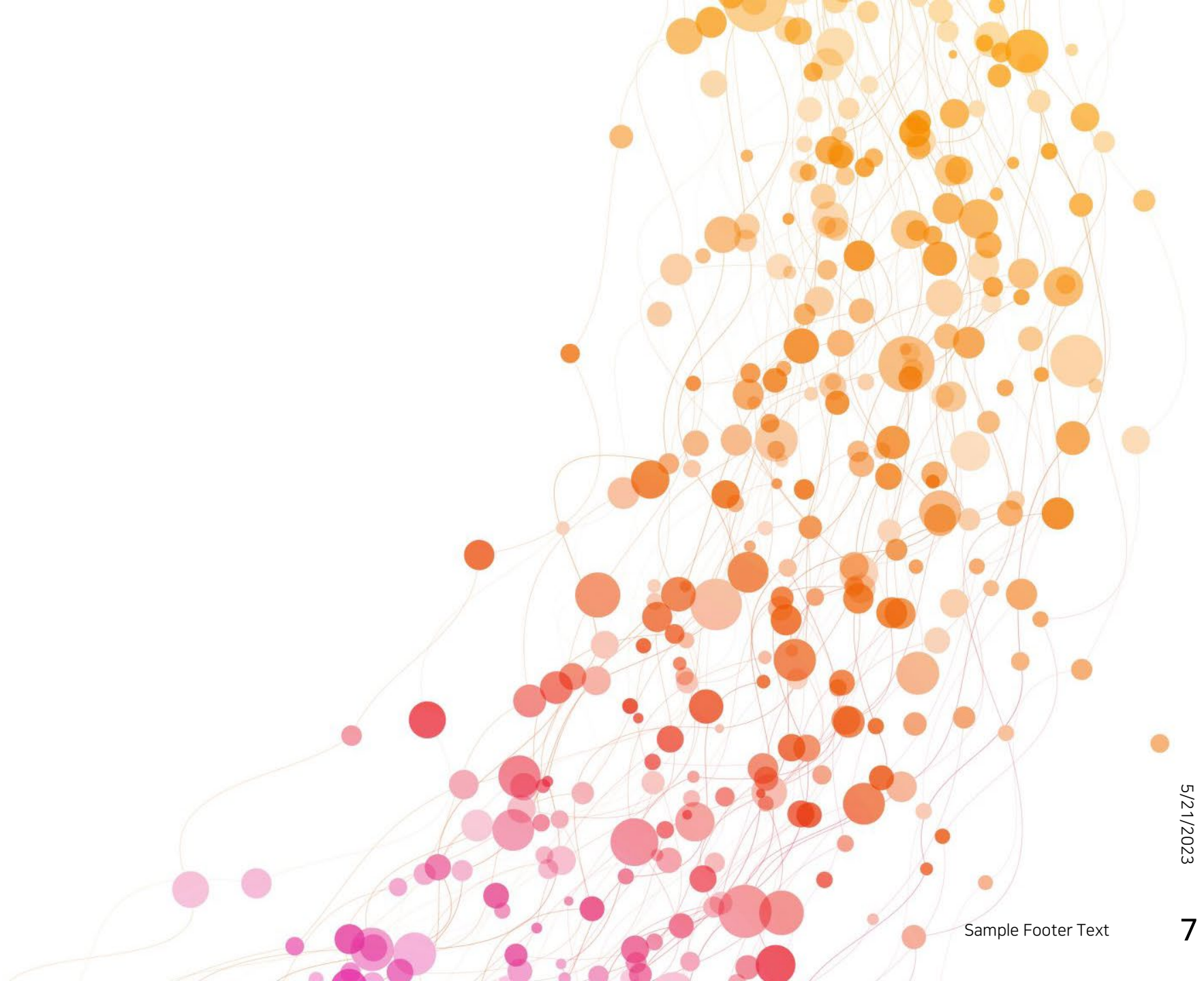
과정소개

페이스메이커



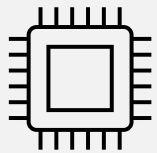
랩 소개

랩 구성

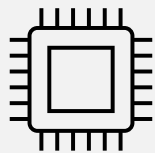


기본과정 랩

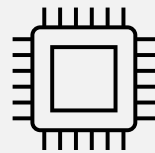
기본 과정에서는 총 3 혹은 4대의 가상머신을 사용한다. 사용하는 랩톱 혹은 데스크탑에 따라서 구성 및 설정한다.



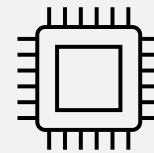
node1



node2



node3



utility



랩톱

4대 구성이 어려우면 node3번을 utility로 사용한다.

페이스메이커 소개

DR VS HA

페이스메이커

페이스 메이커는 실제 심장 박동기(Pacemaker)와 비슷한 동작 방식이다.

옆에 있는 그림처럼 하드웨어가 아니라, 하드웨어 방식은 소프트웨어적으로 구성하여 소프트웨어의 생존성을 높여준다.



페이스메이커

페이스메이커는 다음과 같은 역할을 한다.

- 서비스 및 systemd의 상태를 지속적으로 확인한다.
- 서비스가 문제가 발생하면, 기존 서비스를 노드가 대신한다.
- 크고 작은 서비스에 H/A기반으로 스케일링 서비스가 가능하다.



배경

- 오랫동안 개발을 진행한 리눅스 기반 오픈소스 HA Project. 많은 리눅스 시스템에서 사용이 가능하다.
- 1998년도부터 오픈소스 기반으로 프로젝트를 시작하였으며, 30만 이상의 미션 크리티컬 클러스터에서 사용에서 사용함(1999년부터)
- **IBM/Novel/Oracle/SuSE**와 같은 많은 기업들이 프로젝트에 참여
- 많은 산업 환경에서 사용하고 있으며, 많은 애플리케이션을 지원하고 있음

배경

- 대다수 리눅스 배포판에서 사용이 가능함. 레드햇 계열 및 데비안 계열에서도 사용이 가능.
- 하드웨어 사양을 별도로 요구하지 않음. 모든 소프트웨어 기반으로 사용이 가능함.
- 모든 패키지는 자동화 도구로 테스트 및 검증이 된 후 릴리즈 됨.

COROSYNC

Corosync는 클러스터에서 사용하는 엔진.

이를 통해서 클러스터에 구성이 되어 있는 그룹끼리 서로 대화를 할 수 있도록 함. 또한, 강화된 추가 기능으로 애플리케이션의 가용성을 높일 수 있다.

- 페이스메이커(Pacemaker)
- DRBD
- ScanCore

<https://clusterlabs.org/corosync.html>

DRBD

DRBD는 **Distribute Replicated Storage System**의 약자이다.

이 시스템은 강화된 커널 드라이버이며, 사용자 영역에서 관리 프로그램 혹은 셸 스크립트로 도움을 받아서 사용이 가능하다.

페이스 메이커는 DRBD를 내부적으로 가져와서 구성원으로 사용하고 있다.

<https://linbit.com/drbd/>

DRBD명령어는 따로 페이스 메이커에서 사용할 필요가 없다.

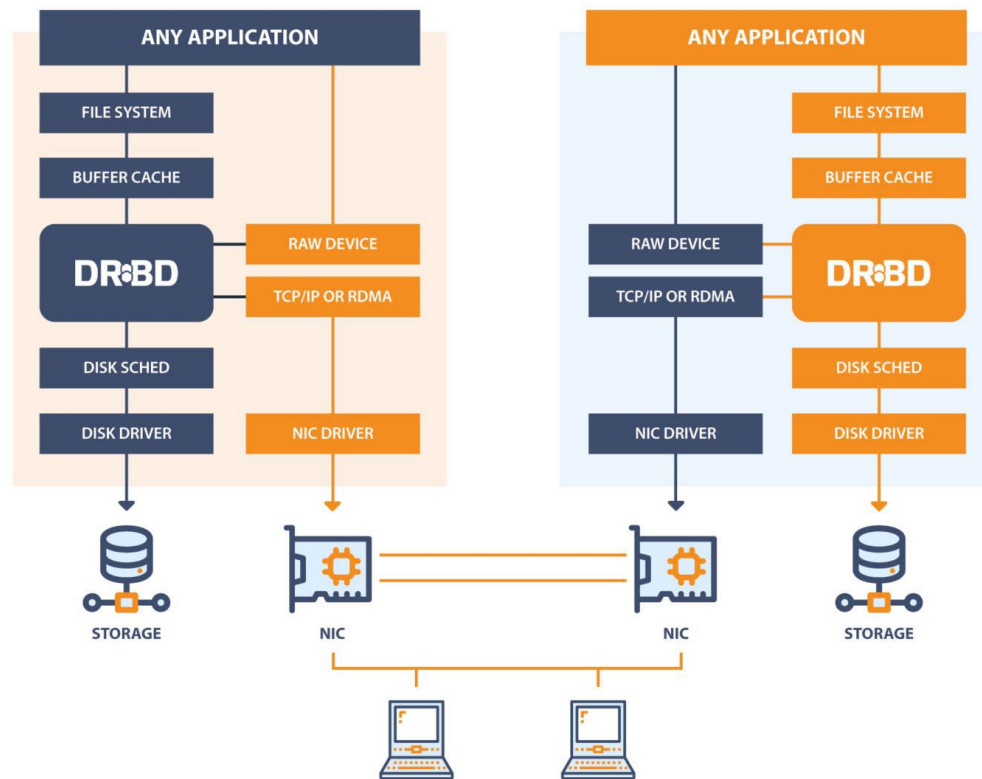
DRBD

DRBD를 사용하기 위해서는 각각 노드에 DRBD 장치를 구성해야 한다.

drbd는 커널 수준에서 장치를 구성 및 배포하기 때문에 리눅스 배포판에서 사용이 가능한지 확인이 필요하다. 이를 사용하기 위해서는 두 가지 형태로 장치를 붙인다.

1. RAW장치
2. LVM2기반의 장치

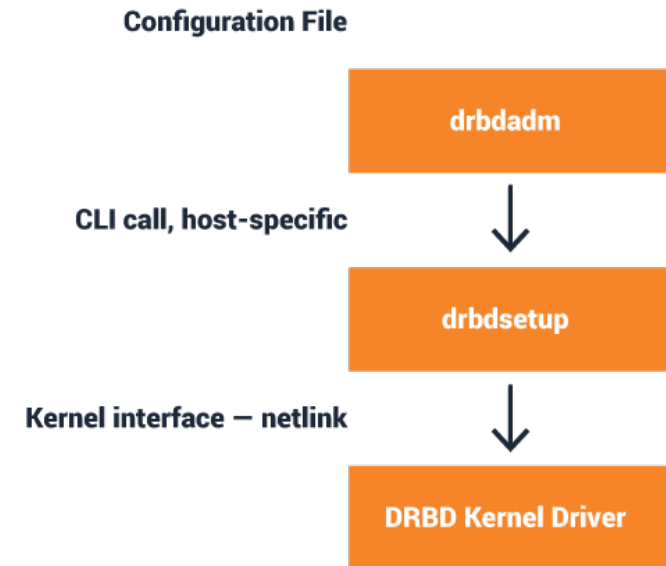
SAN장치가 없는 경우, iSCSI기반으로 구현 및 사용을 권장한다.



DRBD

구성하기 위해서 간단하게 다음과 같은 단계로 진행한다.

drbdadm
drbdsetup



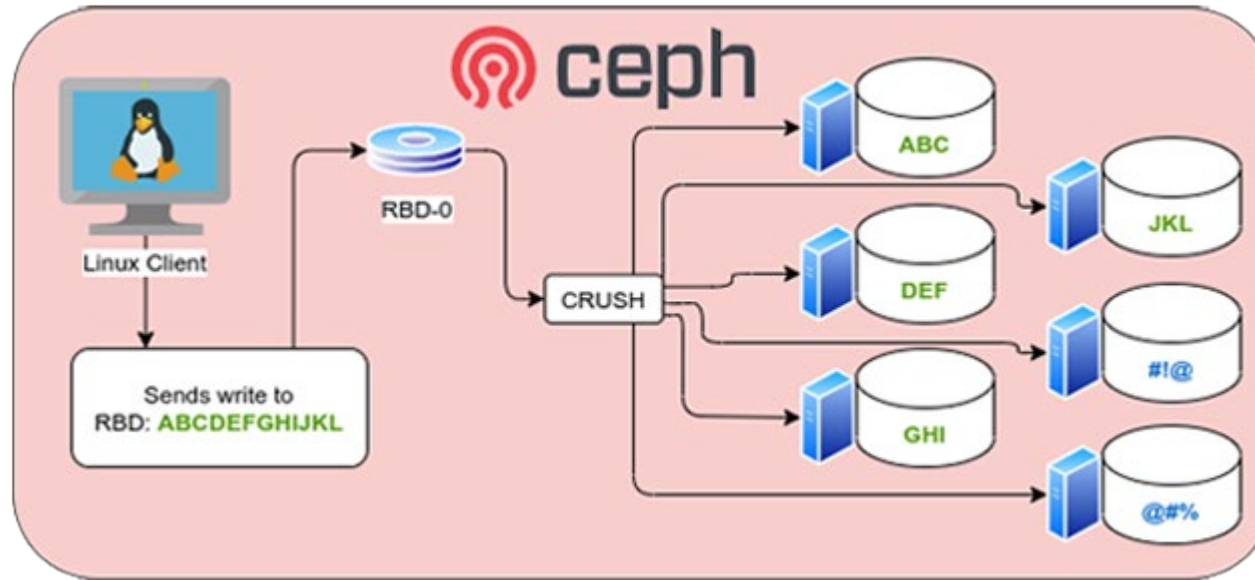
ceph vs drdb

둘은 비슷한 기능을 가지고 있지만, 약간은 다른 성격을 띄고 있다. 둘 다 블록 스토리지 복제 기능을 제공하고 있다.

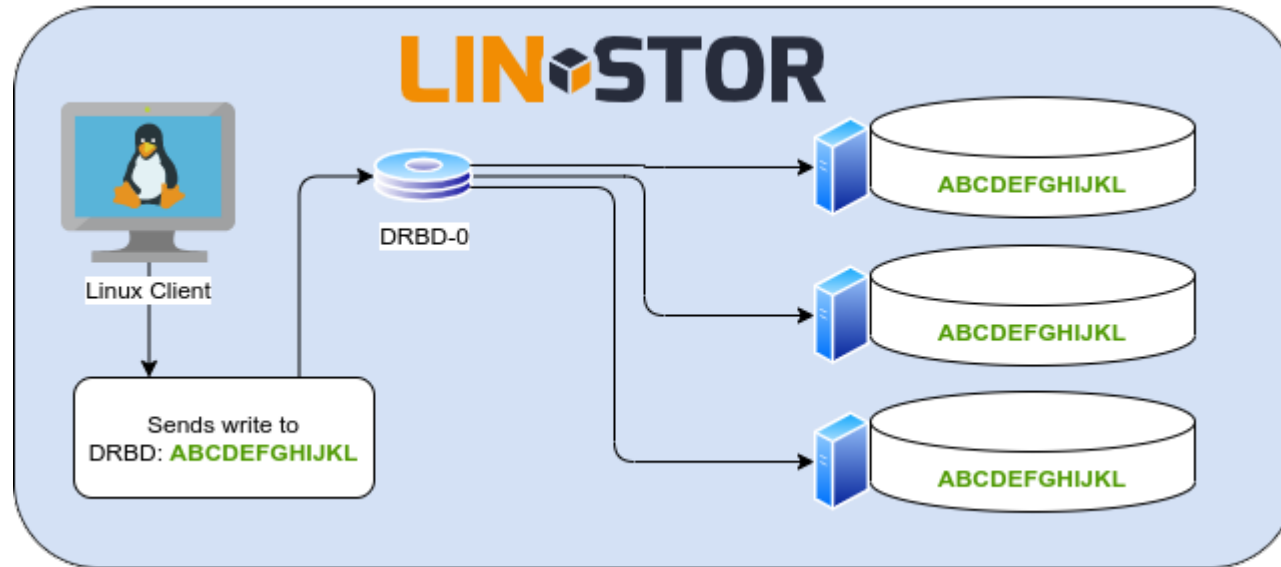
CEPH는 CRUSH알고리즘 기반으로 RAID-1처럼 파일을 효과적으로 복제를 한다. 또한, 데이터 손상을 최소화 하는 알고리즘을 가지고 있다. 쓰기가 자주 발생하고 레이턴시 문제가 크게 없는 경우, CEPH스토리지 사용을 권장한다.

DRDB는 반대로 레이턴시가 낮고 쓰기가 빈번하게 발생하는 경우 DRBD가 더 효율적이다. 다만, DRDB는 CEPH의 CRUSH처럼 복제 알고리즘이 없다.

ceph



DRDB



ScanCore

스캔코어(ScanCore)는 페이스 메이커의 코어 구성원이다. 이 구성원은 각 노드에서 다음과 같은 상태를 확인한다. 보통 이를 결정 엔진(Decision Engine)이라고 부른다. ScanCore는 다음과 같은 역할을 주로 수행한다.

- 과부화(Over Heating)
- 전원 전압 혹은 손실 상태(Loss of input power)
- 노드 상태
- 에이전트 상태 확인

자세한 사용은 아래의 주소에서 확인이 가능하다.

<https://www.alteeve.com/w/ScanCore>

페이스메이커 주요 기능 정리

1. 장치 및 애플리케이션 수준에서 장애 상태 확인
2. 일반적인 여분 자원 설정 지원
3. 리소스 관리 클러스터 및 구성원(quorate)기반의 시스템 지원
4. 설정 기반으로 구성원 손실이 발생하였을 때 처리 방식에 대한 방법(전략)제공
5. 같은 노드가 아니어도 애플리케이션 시작 및 종료 순서 제공
6. 설정 기반으로 같은 노드에서 실행 여부 결정 가능.
7. 애플리케이션 여러 노드에서 활성화가 되어야 하는 설정 가능
8. 애플리케이션들에게 다중역할 기능 제공

페이스메이커 주요 기능 정리

- libQB - core services (logging, IPC, etc)
- Corosync - Membership, messaging and quorum
- Resource agents - A collection of scripts that interact with the underlying services managed by the cluster
- Fencing agents - A collection of scripts that interact with network power switches and SAN devices to isolate cluster members
- Pacemaker itself

페이스메이커 주요 기능 정리

- Pacemaker has been around since [2004](#) and is primarily a collaborative effort between [Red Hat](#) and [SUSE](#), however we also receive considerable help and support from the folks at [LinBit](#) and the community in general.
- Corosync also began life in [2004](#) but was then part of the [OpenAIS project](#). It is primarily a [Red Hat](#) initiative, with considerable help and support from the folks in the community.

이전 및 현 H/A시스템 비교

RHEL/CentOS 7 이전 버전에서는

구 버전은 페이스메이커 사용이 불가능함.

이전에는 RGMAN 혹은 CMAN으로 호칭하였음.

RHLE/CentOS 7 이상 버전에서는

- 레드햇 계열에서는 RHEL 7부터 사용이 가능.
- 수세 리눅스는 SELHA 12부터 사용이 가능.

RGMAN VS PACEMAKER

	리소스 매니저	페이스메이커
리소스 설정 관리	수동	자동
리소스 관리 모델	자원 그룹	리소스 그룹 및 의존성
의존성 모델	위치 선언 및 시작 후 시작	사용자 설정
이벤트 제어 방식	중앙 혹은 배포	중앙화
명령어 관리	상태 및 자원제어	상태 및 자원제어 및 설정
차단방식	제한적 혹은 OCF	유연하게 OCF 에이전트 가능
다중 리소스 상태 확인	아님	지원
이벤트 스크립트	지원	아님
최대 노드 개수	16개	16개 혹은 32개

RGMAN VS PACEMAKER

독점 서비스	Yes	Yes
도메인 장애복구(failover)	Yes	Yes
리소스 제외	No	Yes
시간 기반 리소스 제어	No	Yes
리소스 속성 상속	Yes	Yes
리소스 공유	Yes	Yes
리소스 복제(설정 및 에이전트)	No	Yes
리소스 API 에이전트 형식	OCF, SysV	OCF, SysV

RGMAN VS PACEMAKER

리소스 중지	Yes	Yes
구성원 필요	Yes	Configurable
DLM 필요	Yes	No
다중 파티션 자원 관리지원	No	Yes
비 관리자 기반 관리 자원	No	Yes

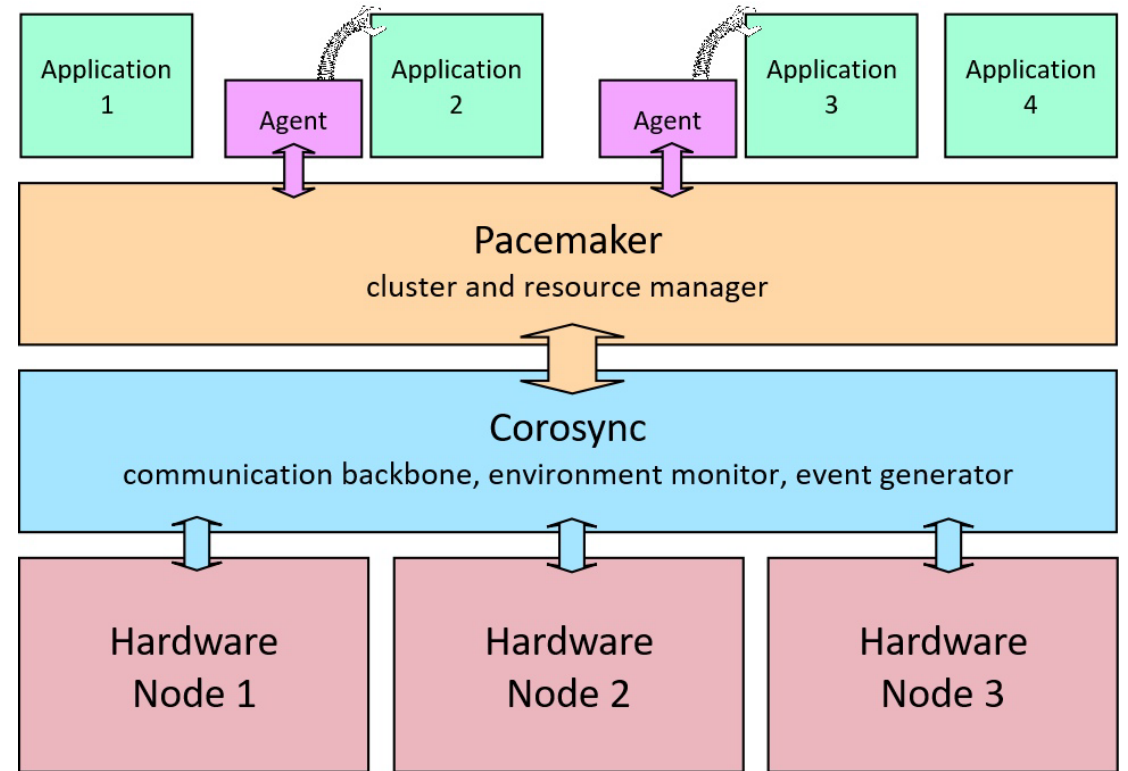
THE AGENT

1. 페이스메이커 및 리소스 매니저는 OCF 기반의 에이전트 사용이 가능.
2. 모든 OCF자원을 활용이 하기가 어렵기 때문에, 추가적으로 구성을 원하는 경우, 아래 링크에서 확인이 가능.

<http://www.linux-ha.org/doc/dev-guides/ra-dev-guide.html>

THE AGENT

에이전트는 각각 자원별로 리소스 에이전트를 가지고 있다.
에이전트는 설정을 통해서 애플리케이션 및 리소스를 관리한다. 해당 리소스는 페이스메이커가 관리를 하며, 설정 배포 및 환경 상태를 관리하는 Corosync를 통해서 한다.



숫자 9 그리고 페이스메이커

- H/A시스템은 100% 사용율을 달성 할 수 없다.
- 잘 구성된 HA 클러스터 시스템은 가동율에 "9"라는 숫자를 추가 혹은 제공한다.
- 클러스터는 절대로 복잡하게 구성 및 추가하면 안된다.
 - 복잡한 클러스터 구성은 거의 대다수가 완벽하게 실패한다.

99.9999% IN 30SEC

99.999% IN 5MIN

99.99% IN 52 MIN

99.9% IN 9 HOUR

99% IN 3.5 DAY

DR VS HA

D/R(DISASTER RECOVERY)이라고 부르는 제해 및 재앙에 관련된 복구 시스템. 페이스 메이커는 기본적으로 H/A를 대상으로 작성된 프로그램.

D/R을 H/A와 비용을 비교 하였을 때 다음과 같다.

1. D/R 페일오버(Failover)는 비용이 비싸다
2. D/R 페일오버는 시간 단위로 측정이 가능하다
3. 내부 노드 문제로 신뢰할 수 없는 노드 통신
4. 클러스터 및 노드 사이에 너무 복잡한 디자인

DR VS HA

H/A(HIGH AVAILABILITY)는 D/R보다는 작은 범주에서 동작하는 구조. D/R에 비해서 매우 저렴하다.

- H/A 장애처리는 D/R에 비해서 저렴하다
- H/A 장애처리 시간은 보통 초단위로 가능하다
- 노드간 통신이 가능하다
- 에이전트를 통해서 클러스터 및 노드를 간단하게 설계 및 디자인

SINGLE POINTS OF FAILURE

SPoF(Single Points Of Failure), 단일 지점에서 장애가 발생 하였을 때, H/A시스템 구조는 잘 동작한다.

하지만, 노드 단위로 다발적으로 장애가 발생하는 경우, H/A시스템은 빠르게 대체를 못하는 경우가 있다. 그래서 일반적으로 SPoF는 서비스 대상으로 디자인을 하는 경우가 많다.

긍정

H/A 디자인은 SPoF에는 최적화 되어 있는 설계. 일반적으로 대다수의 H/A는 서비스 대상으로 구성이 되어 있다.

단점

H/A디자인은 모든 시스템 혹은 서비스에 대해서 확인을 할 수 없다. 앞서 이야기 내용처럼 노드 대 노드는 기본적으로 H/A시스템과 맞지 않다.

STONITH

리소스(서비스)가 장애가 발생하면, 차단은 서비스 무결성을 보장한다. Shoot the Other Node in the Head, 말 그대로 장애가 발생한 노드를 클러스터에서 처리한다.

차단(FENCING)

SCSI RELEASE/LOCK AND RESERVE

페이스 메이커는 다양한 볼륨 장치를 지원하는데, 기본적으로 지원하는 장치는 LVM2, GFS2, NFS가 있다.

SCSI CHANNEL

iSCSI 및 FC(Fiber Channel)를 제공한다.

기능(CAPABILITIES)

- 클러스터 노드는 16개까지 권장한다
 - 이 부분에 대해서 나중에 더 자세히 이야기
- 병렬통신을 사용한다. 예를 들어서 UDP, Broadcast, MultiCast, Unicast 통신을 사용한다.
- 노드 문제 혹은 서비스 문제
- IP연결 문제 혹은 접근 문제 또는 임의 기준으로 장애 처리
- 액티브 패시브 혹은 액티브-액티브 모델
- 모니터링 리소스를 자체적으로 소유
- OCF 표준 리소스 관리 및 모니터링 제공

기능(CAPABILITIES)

- 풍부한 제약 조건을 지원하는 정교한 종속성 모델(리소스, 그룹, 노드 이전, 마스터/슬레이브)
- XML 기반 리소스 구성
- 구성 및 모니터링 GUI
- OCFS 클러스터 파일 시스템 지원
- 다중 상태(마스터/슬레이브) 리소스 지원

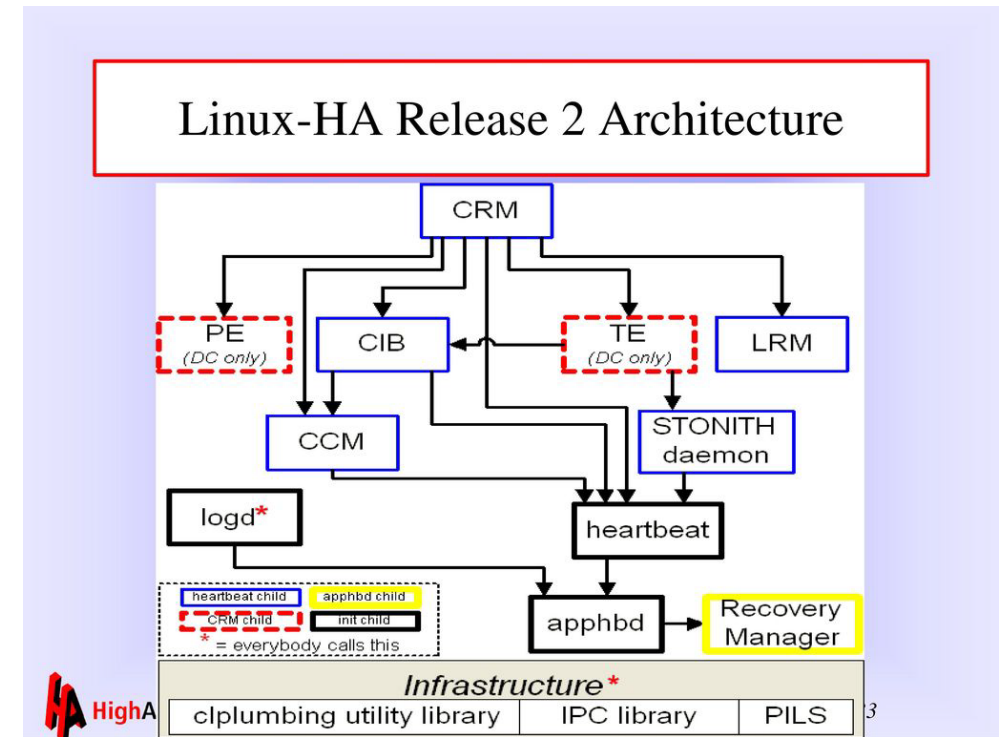
기능(CAPABILITIES)

페이스 메이커는 다음과 같은 자원을 지원한다.

- 리소스
- 리소스 에이전트
- DC(DESIGNATED COORDINATOR), 마스터 노드
- 스노니스(STONITH)
- 스플릿 브레인
- 정족수(Quorum)

기능(CAPABILITIES)

LOLO





랩 구성

하이퍼바이저

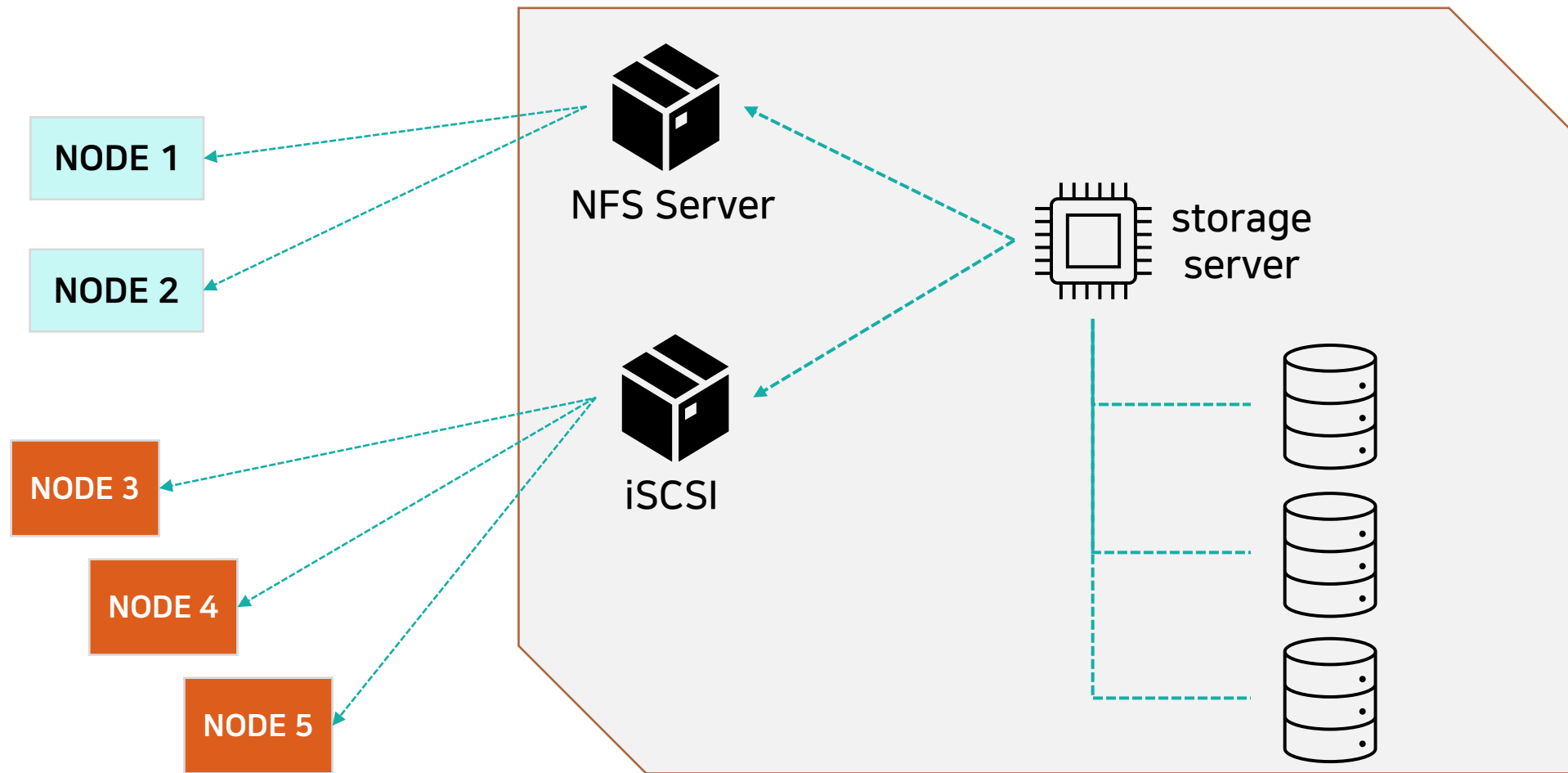
5/21/2023

랩 구성

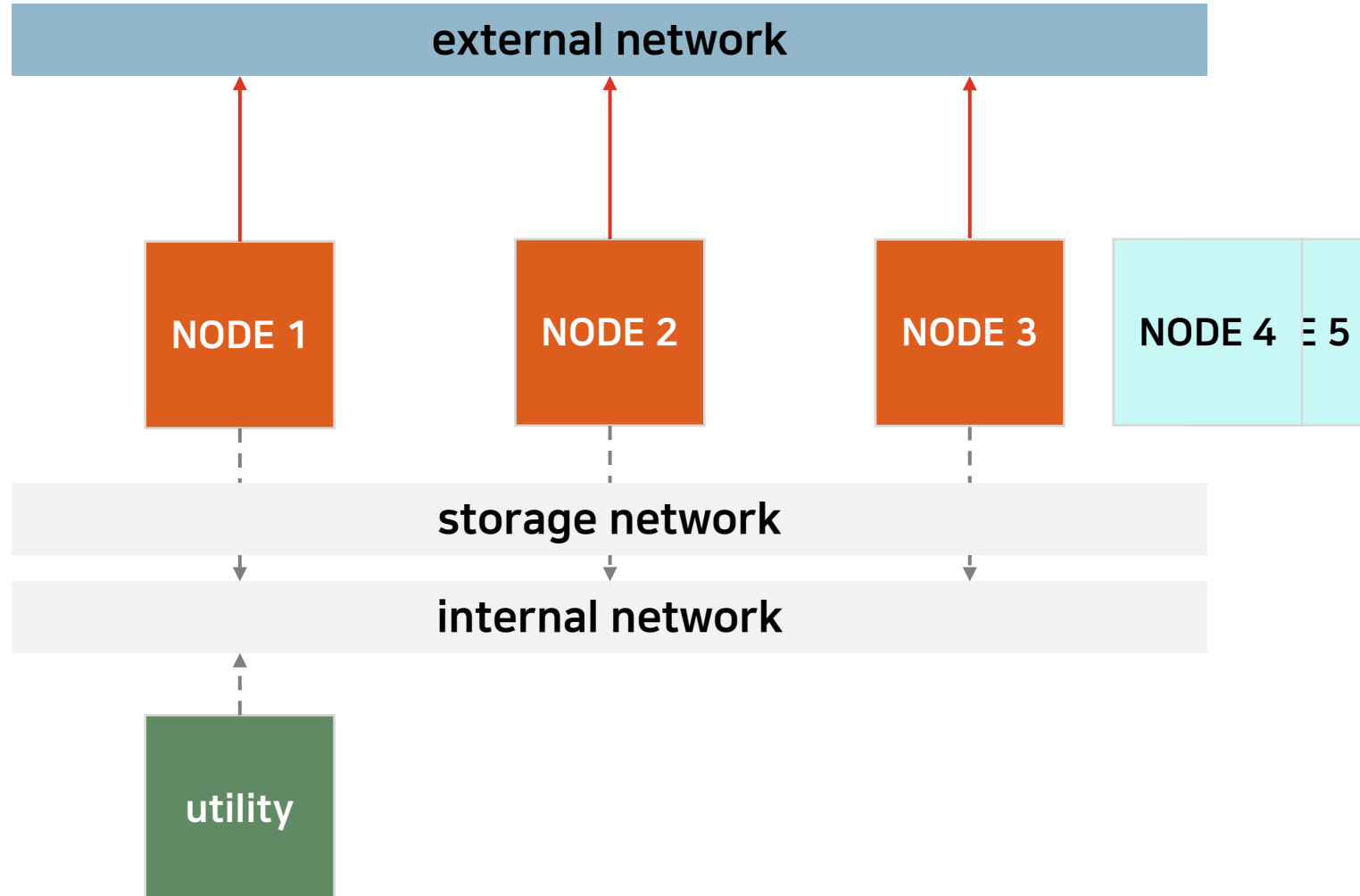
강의 시작 전, 가상머신을 리눅스에서 설치 한다. 페이스 메이커를 사용하기 위해서 올바르게 저장소 구성을 한다. 여기서 사용하는 하이퍼바이저는 윈도우 10/11 Pro기반의 하이퍼브이 기반으로 사용한다.

1. CentOS-9-Stream 기반으로 랩을 구성한다.
 - 총 4대의 가상머신을 구성한다.
 - 192.168.90.250번은 VIP주소로 사용한다.
 - 두 개의 NIC카드를 가지고 있어야 한다. "default"는 외부망으로 사용하고, "internal", "storage"내부 네트워크를 따로 구성한다. 구성이 어려운 경우 "internal"하나만 가지고 있어도 된다.
2. 용량이 부족하면 최소 3개를 구성한다.
3. 가상머신 하나는 반드시 유틸리티 서버가 되어야 한다.
 - iSCSI, NFS, GFS2
 - DNS, 만약 구성이 가능하다면

기본 구성



네트워크



LAB

랩을 위해서 다음과 같은 패키지를 호스트(베어메탈) 리눅스에 설치한다. 윈도우 컴퓨터를 사용하는 경우 IPMI프로토콜 서버를 사용할 수 없다. 리눅스로 랩을 진행하는 경우 아래 프로그램 설치를 권장한다.

- libvirt
- virsh
- virt-builder

IPMI 구성

가상으로 IPMI프로토콜을 구현을 원하는 경우, VirtualBMC를 통해서 구현이 가능하다. 이 랩에서는 IPMI은 사용하지 않는다.

- virtualbmc

The Intelligent Platform Management Interface (IPMI) is a set of computer interface specifications for an autonomous computer subsystem

- <https://github.com/openstack/virtualbmc>

IPMI 설치 및 구성

```
[root@localhost stack]#
```

IPMI 설치 및 구성

```
host# dnf install libvirt libvirt-devel python3-devel gcc -y
```

```
host# pip3 install virtualbmc
```

```
host# vbmcd
```

```
host# vbmc add --username centos --password centos --port 7755 --  
libvirt-uri qemu+ssh://root@bare/system node2
```

```
host# vbmc list
```

가상머신 구성 및 설치

가상 머신을 사양에 따라서 다르다. 최소 3대의 서버가 필요하며, 가급적이면 6대 정도의 가상머신 사용을 권장한다.

구성은 다음과 같은 순서로 진행한다.

- libvirt 설치
- 가상머신 관리 명령어 설치
- 내부 네트워크 구성
- 가상머신 이미지 구성 및 생성

가상머신 설치 준비

```
host# dnf groupinstall "Virtualization Host" -y
```

```
host# dnf install libguestfs-tools-c -y
```

```
host# virt-builder --list
```

```
host# virsh net-list
```

```
host# cat <<EOF> internal-network.xml
```

내부 네트워크 XML파일

```
<network>
  <name>internal</name>
  <bridge name='virbr10' stp='on' delay='0' />
  <mac address='52:54:00:91:24:b8' />
  <domain name='internal' />
  <ip address="192.168.90.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.90.2" end="192.168.90.254" />
    </dhcp>
  </ip>
</network>
EOF
```

가상 내부 네트워크 생성

가상머신에 사용할 내부 네트워크를 등록한다.

```
host# virsh define --file internal-network.xml
```

```
host# virsh define --file storage-network.xml
```

```
host# virsh net-list
```

가상머신 OS이미지 생성

최소 사양으로 사양하는 경우 총 3대의 가상머신을 만든다. 문제가 없는 경우 총 6대의 가상머신을 만든다.

```
host# virt-builder --size 10G --format qcow2 --root-password password:centos -o  
/var/lib/libvirt/images/node1.qcow2 centosstream-9
```

```
host# virt-builder --size 10G --format qcow2 --root-password password:centos -o  
/var/lib/libvirt/images/node2.qcow2 centosstream-9
```

```
bare# virt-builder --size 30G --format qcow2 --root-password password:centos -o  
/var/lib/libvirt/images/node3.qcow2 centosstream-9
```


CLI기반으로 가상머신 설치

가상머신을 CLI에서 설치하기 위해서 "virt-install"명령어를 통해서 설치를 진행한다. 메모리 및 CPU는 컴퓨터 사양에 맞게 구성한다. 권장하는 최소 사양은 다음과 같다.

- vCPU: 2개
- vMEM
 - 4096MiB(페이스 메이커 및 OCF의 원활한 동작을 위해서 필요)

CLI기반으로 가상머신 설치

가상머신을 CLI에서 설치하기 위해서 "virt-install"명령어를 통해서 설치를 진행한다. 메모리 및 CPU는 컴퓨터 사양에 맞게 구성한다. 권장하는 최소 사양은 다음과 같다.

vCPU: 2개

vMEM: 4096MiB(페이스 메이커 및 OCF의 원활한 동작을 위해서 필요)

```
host# dnf install virt-install -y
```

```
host# virt-install --memory 4096 --cpu host-copy --vcpu 2 -n node1 --disk  
/var/lib/libvirt/images/node1.qcow2,cache=none,bus=virtio -w network=default,model=virtio -w  
network=internal,model=virtio --graphics none --autostart --noautoconsole --import
```

```
host# virt-install --memory 4096 --cpu host-copy --vcpu 2 -n node2 --disk  
/var/lib/libvirt/images/node2.qcow2,cache=none,bus=virtio -w network=default,model=virtio -w  
network=internal,model=virtio --graphics none --autostart --noautoconsole --import
```


CLI기반으로 가상머신 설치

가상머신을 "virt-install"명령어로 설치 후, 올바르게 구성이 되면, 아래와 같은 명령어로 올바르게 동작하는지 확인한다.

```
host# virt-install --memory 4096 --cpu host-copy --vcpu 2 -n node3 --disk  
/var/lib/libvirt/images/node3.qcow2,cache=none,bus=virtio -w  
network=default,model=virtio -w network=internal,model=virtio --graphics none --  
autostart --noautoconsole --import
```

CLI기반으로 가상머신 설치

```
host# virsh console node1
```

```
host# virsh console node2
```

```
host# virsh console node3
```

```
host# virsh domifaddr node1
```

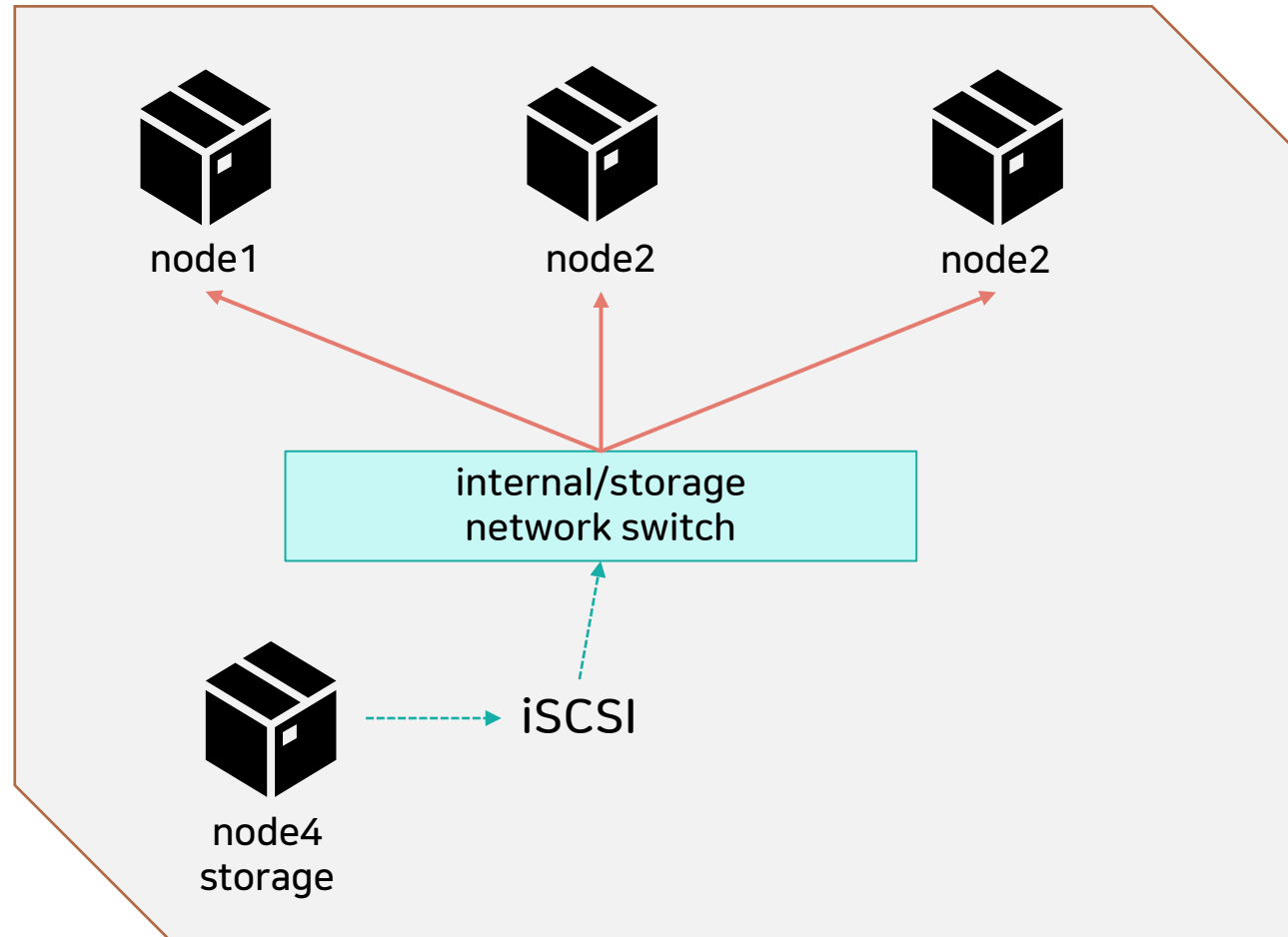
```
host# ssh root@<IP>
```

ISCSI

타겟 서버 및 파일기반 블록 스토리지 서버 구성 및
설정

5/21/2023

ISCSI



iSCSI

iSCSI는 SAN(Storage Attached Network)프로토콜 사양이다. 랩에서 직접적으로 SAN를 사용하기 어렵기 때문에 iSCSI기반으로 노드에 디스크를 제공한다.

이를 위해서 우리는 아래와 같은 소프트웨어를 사용한다.

- targetd
- targetd-cli
- iscsi

ISCSI SERVER

블록장치 혹은 파일장치 기반으로 디스크를 배포한다.

ISCSI

호스트 컴퓨터가 공간이 넉넉하면 직접 블록 가상 블록 장치를 만들어서 확장 디스크를 제공하여도 된다. 랩에서는 공간 관리를 하기 위해서 파일 기반 블록 장치를 iSCSI를 통해서 제공한다.

```
node4# dnf install targetcli -y
```

```
node4# systemctl enable --now target
```

```
node4# firewall-cmd --add-service=iscsi-target
```

```
node4# dnf install iscsi-initiator-utils -y
```

ISCSI SERVER

```
node4# mkdir -p /var/lib/iscsi_disks
```

```
node4# targetcli backstores/fileio create iscsi /var/lib/iscsi_disks/iscsi_disk.img 2G
```

```
node4# targetcli backstores/fileio create nfs /var/lib/iscsi_disks/nfs_disk.img 2G
```

```
node4# targetcli backstores/fileio create gfs2 /var/lib/iscsi_disks/gfs2_disk.img 2G
```

ISCSI SERVER

```
node4# targetcli iscsi/ create iqn.2023-02.com.example:blocks
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/luns/ create /backstores/fileio/iscsi/
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/luns/ create /backstores/fileio/nfs/
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/luns/ create /backstores/fileio/gfs2/
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/ create iqn.2023-02.com.example:node1.init
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/ create iqn.2023-02.com.example:node2.init
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/ create iqn.2023-02.com.example:node3.init
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/ create iqn.2023-02.com.example:node4.init
```

ISCSI CLIENT

클라이언트 접근에 대한 ACL 설정을 한다.

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node1.init set auth userid=username
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node1.init set auth password=password
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node2.init set auth userid=username
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node2.init set auth password=password
```

ISCSI CLIENT

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node3.init set auth userid=username
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node3.init set auth password=password
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node4.init set auth userid=username
```

```
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/acls/iqn.2023-02.com.example:node4.init set auth password=password
```

ISCSI

```
node4# targetcli saveconfig
```

```
node1/2/3/4# dnf install iscsi-initiator-utils -y
```

```
cat <<EOF>> /etc/iscsi/initiatorname.iscsi
```

```
InitiatorName=iqn.2023-02.com.example:node1.init
```

```
EOF
```

```
/etc/iscsi/iscsid
```

```
node.session.auth.authmethod = CHAP
```

```
node.session.auth.username = username
```

```
node.session.auth.password = password
```

```
node1/2/3# systemctl restart iscsi iscsid
```

```
node1/2/3# iscsiadm -m discovery -t sendtargets -p 192.168.90.140
```

```
node1/2/3# iscsiadm -m node --login
```

```
node1/2/3# iscsiadm -m session --debug 3
```

```
node1/2/3# iscsiadm -m session --rescan
```

연습문제

node4를 복구 후, 다시 ISCSI장치를 구성.

- 블록 장치 혹은 파일 기반으로 3개의 디스크를 추가
- GFS2, NFS를 위한 블록 장치 생성
- 모든 노드에 iSCSI장치 연결 및 장치 구성

클러스터 구성 준비

리눅스 설정

리눅스 구성

시작 전, 각각 가상머신에 대해서 이미지 스냅샷을 수행한다.

```
host# virsh snapshot-create as --domain node1 --name node1-pcs-setup
```

```
host# virsh snapshot-create-as --domain node2 --name node2-pcs-setup
```

```
host# virsh snapshot-create-as --domain node3 --name node3-pcs-setup
```

```
host# virsh snapshot-create-as --domain node4 --name node4-pcs-setup
```

```
host# virsh snapshot-list node1
```

```
host# virsh snapshot-revert --domain node1 --snapshotname node1-pcs-setup --running
```

위의 명령어로 각각 가상머신 스냅샷을 생성한다. 총 생성해야 가상머신은 **node1, node2, node3** 혹은 **node4(utility)**포함.

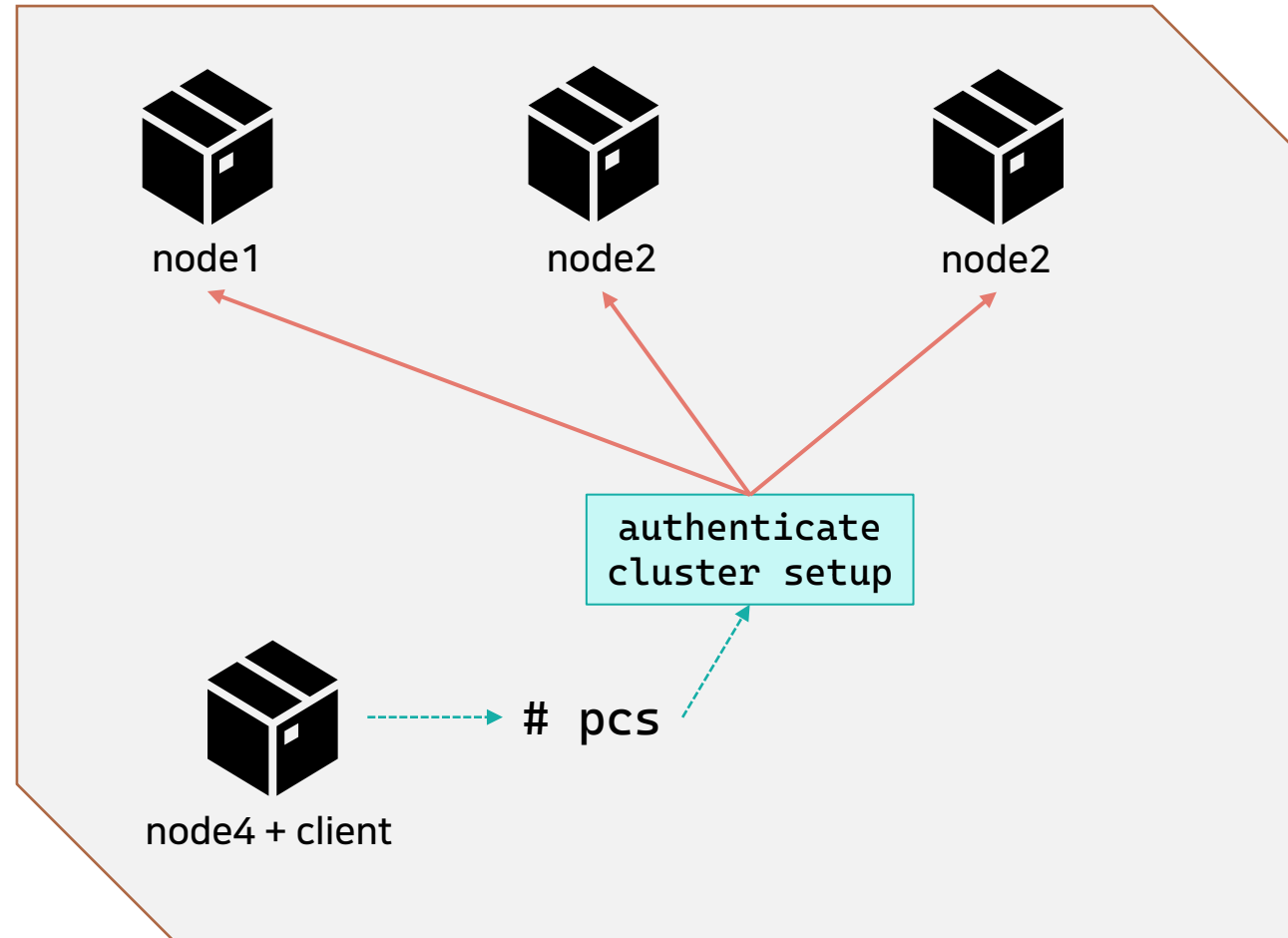
리눅스 구성

시작 전, 각각 가상머신에 대해서 이미지 스냅샷을 수행한다.

> Checkpoint-VM -Name node1 -SnapshotName 'before cluster create'

위의 명령어로 각각 가상머신 스냅샷을 생성한다. 총 생성해야 가상머신은 node1, node2, node3
혹은 node4(utility)포함.

PACEMAKER



리눅스 네트워크 구성

"internal"네트워크 인터페이스 카드에 다음과 같이 구성한다.

```
node1# nmcli con add con-name eth1 ipv4.addresses 192.168.90.110/24 ipv4.never-  
default yes ipv4.method manual autoconnect yes type ethernet ifname eth1
```

```
node1# nmcli con up eth1
```

```
node2# nmcli con add con-name eth1 ipv4.addresses 192.168.90.120/24 ipv4.never-  
default yes ipv4.method manual autoconnect yes type ethernet ifname eth1
```

```
node2# nmcli con up eth1
```

리눅스 네트워크 구성

```
node3# nmcli con add con-name eth1 ipv4.addresses 192.168.90.130/24  
ipv4.never-default yes ipv4.method manual autoconnect yes type ethernet ifname  
eth1
```

```
node3# nmcli con up eth1
```

```
node4# nmcli con add con-name eth1 ipv4.addresses 192.168.90.140/24  
ipv4.never-default yes ipv4.method manual autoconnect yes type ethernet ifname  
eth1
```

```
node4# nmcli con up eth1
```

PACEMAKER

각 서버에 호스트 이름 설정.

```
node1# hostnamectl set-hostname node1.example.com
```

```
node2# hostnamectl set-hostname node2.example.com
```

```
node3# hostnamectl set-hostname node3.example.com
```

```
node4# hostnamectl set-hostname node4.example.com
```

PACEMAKER

DNS 서버가 없기 때문에, A Record를 /etc/hosts파일 통해서 구성한다.

```
node4# cat <<EOF>> /etc/hosts
```

```
192.168.90.110 node1.example.com node1
```

```
192.168.90.120 node2.example.com node2
```

```
192.168.90.130 node3.example.com node3
```

```
192.168.90.140 node4.example.com node4 storage cli
```

```
EOF
```


PACEMAKER

각각 서버에 SSH키를 생성 후 배포한다.

```
node4# ssh-keygen -t rsa -N" ~/.ssh/id_rsa
```

```
node4# dnf install sshpass -y
```

PACEMAKER

키를 생성한 다음에, 다음과 같은 명령어로 키를 배포한다.

```
node4# cat <<EOF> ~/.ssh/config
```

```
StrictHostKeyChecking=no
```

```
EOF
```

```
node4# for i in node{1..4}; do sshpass -pcentos ssh root@$i 'dnf update -y' ; done
```

```
node4# for i in node{1..4}; do sshpass -pcentos scp /etc/hosts root@\$i.example.com:/etc/hosts ; done
```

```
node4# for i in node{1..4}; do sshpass -p centos ssh root@$i 'dnf --enablerepo=highavailability -y install pacemaker pcs' ; done
```

```
node4# for i in node{1..4}; do sshpass -p centos ssh root@$i 'dnf install firewalld && systemctl enable --now firewalld' ; done
```

PACEMAKER

```
node4# for i in {1..4} ; do sshpass -p centos ssh root@node${i} 'firewall-cmd --add-service=high-availability && firewall-cmd --runtime-to-permanent' ; done
```

```
node4# for i in {1..4} ; do sshpass -p centos ssh root@node$i 'echo centos | passwd --stdin hacluster' && systemctl enable --now pcsd.service ; done
```

```
node4# ping node1 -c3
```

```
node4# ping node2 -c3
```

```
node4# ping node3 -c3
```

PACEMAKER

```
node4# pcs host auth -u hacluster -p centos node1.example.com  
node2.example.com
```

```
node4# pcs cluster setup ha_cluster_lab node1.example.com node2.example.com
```

PACEMAKER

```
node4# pcs cluster start --all
```

```
node4# pcs cluster enable --all
```

```
node4# pcs cluster status
```

```
node4# pcs status corosync
```

```
node4# pcs cluster stop --all
```

```
node4# pcs cluster destroy --all
```

```
node4# ss -npltu | grep -i corosync
```

PACEMAKER

리눅스에서 노드 및 클러스터를 추가하고 싶은 경우, 아래 명령어를 수행한다. 윈도우 경우에는 하이퍼브이에서 가상머신 추가하여 구성한다.

```
host# virt-builder --size 30G --format qcow2 -o --root-password password:centos /var/lib/libvirt/images/node4.qcow2 centosstream-8
```

```
host# virt-install --memory 4096 --vcpu 2 -n node4 W
```

```
--disk /var/lib/libvirt/images/node4.qcow2,cache=none,bus=virtio W
```

```
-w network=default,model=virtio -w network=internal,model=virtio W
```

```
--graphics none --autostart --noautoconsole --import
```

```
host# pcs cluster auth -u hacluster -p centos node4
```

```
host# pcs cluster node add node4 --start --enable
```

혹은

```
host# pcs cluster start node4
```

```
host# pcs cluster enable node4
```

연습문제

각각 가상머신을 다시 롤백 후, 아래와 같이 작업을 수행한다.

1. node4번 서버에 node3/4를 hacluster에 추가한다.
2. 추가를 하기 위한 네트워크를 구성한다.
3. 올바르게 구성이 되면 각각 서버를 node4에서 조회가 가능해야 한다.
4. 구성이 완료가 되면 다시 클러스터를 초기화 한다.

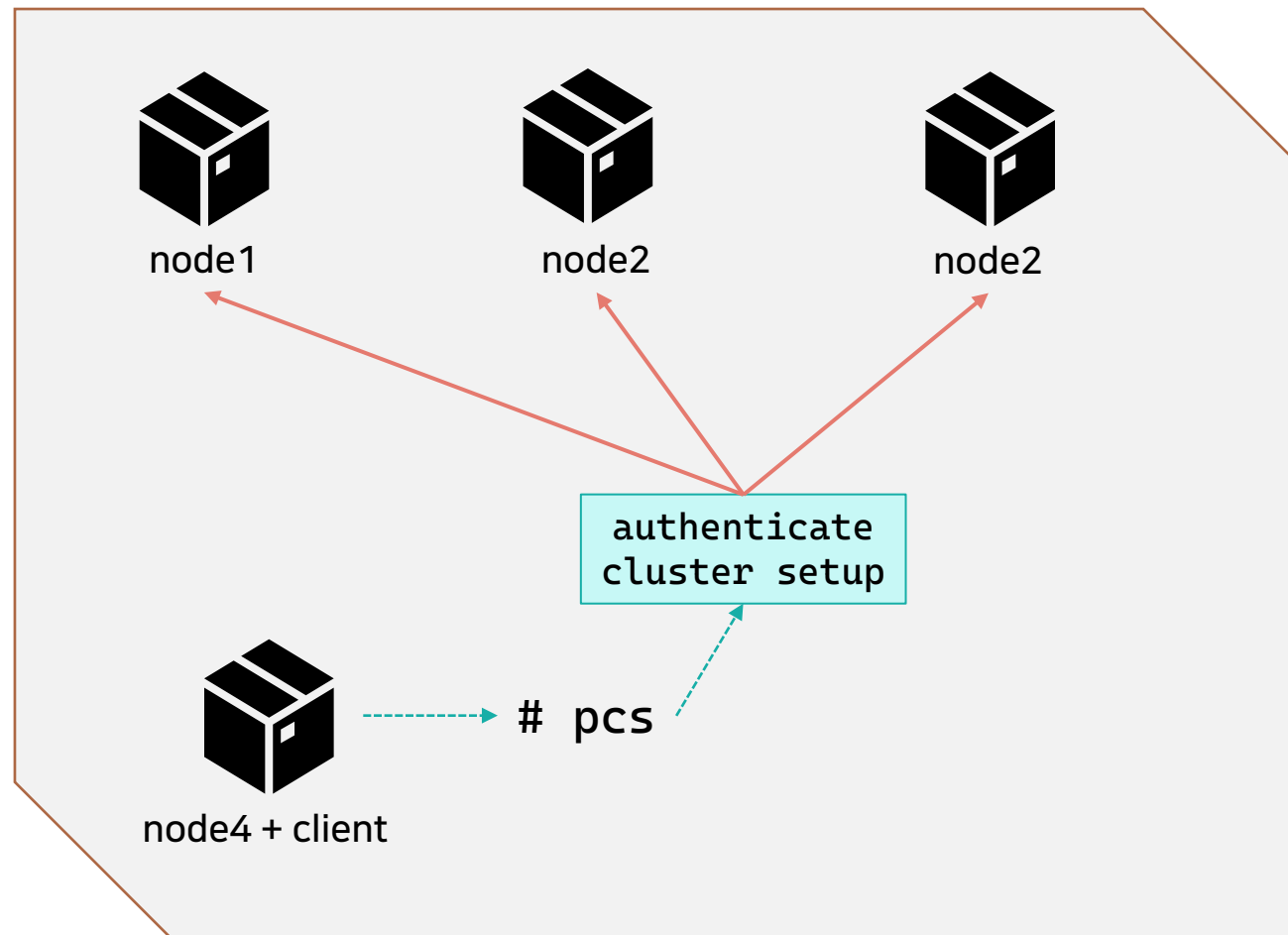
HA클러스터 구성 및 확인

pacemaker

PACEMAKER CLUSTER

- 클러스터 생성
- 클러스터 확인

클러스터 구성



방화벽 설정 및 구성

```
node1/2/3/4# dnf --enablerepo=highavailability -y install pacemaker pcs
```

```
node1/2/3/4# systemctl enable --now pcsd
```

```
node1/2/3/4# echo centos | passwd --stdin hacluster
```

```
node1/2/3/4# firewall-cmd --add-service=high-availability --permanent
```

```
node1/2/3/4# firewall-cmd --reload
```

클러스터 재구성 및 상태 확인

```
node4# pcs host auth -u hacluster -p centos node1.example.com node2.example.com  
node3.example.com node4.example.com
```

```
node4# pcs cluster setup ha_cluster_lab node1.example.com node2.example.com  
node3.example.com node4.example.com
```

```
node4# pcs cluster start --all
```

```
node4# pcs cluster enable --all
```

```
node4# pcs cluster status
```

```
node4# pcs status corosync
```

```
node4# corosync-cfgtools -s
```

노드 추가하기

```
node4# pcs host auth -u hacluster -p centos node4.example.com
```

```
node4# pcs cluster setup ha_cluster_lab node4.example.com
```

```
node4# pcs cluster node add node4.example.com --enable --start
```

```
node4# pcs cluster status
```

```
node4# pcs status corosync
```

```
node4# corosync-cfgtools -s
```

간단한 펜싱 장치 구성

```
node1/2/3/4# dnf --enablerepo=highavailability -y install fence-agents-scsi
```

```
node1/2/3/4# ls /dev/disk/by-id
```

```
node4# pcs stonith create scsi-shooter fence_scsi
```

```
pcmk_host_list="node2.example.com node3.example.com node4.example.com "
```

```
devices=/dev/disk/by-id/wwn-<ID> meta provides=unfencing
```

```
node1/2/3/4# pcs stonith config scsi-shooter
```

```
node1/2/3/4# pcs status
```

펜싱 후 복구

```
node4# pcs status
```

```
node4# pcs stonith fence node2.example.com
```

```
node4# pcs cluster status
```

```
node2# pcs cluster start node2.example.com
```

```
node2# reboot
```

노드 상태 확인하기

```
host# corosync-cfgtool -s
```

```
Local node ID 4, transport knet
```

```
LINK ID 0 udp
```

```
addr   = 192.168.90.140
```

```
status:
```

```
nodeid:    1:  connected
```

```
nodeid:    2:  connected
```

```
nodeid:    3:  connected
```

```
nodeid:    4:  localhost
```


노드 상태 확인하기

```
# pcs cluster sync
```

```
node1.example.com: Succeeded
```

```
node2.example.com: Succeeded
```

```
node3.example.com: Succeeded
```

```
node4.example.com: Succeeded
```

```
Warning: Corosync configuration has been synchronized, please reload corosync  
daemon using 'pcs cluster reload corosync' command.
```

노드 상태 확인하기

```
# corosync-cmapctl | grep members
```

```
runtime.members.1.config_version (u64) = 0
```

```
runtime.members.1.ip (str) = r(0) ip(192.168.90.110)
```

```
runtime.members.1.join_count (u32) = 1
```

```
runtime.members.1.status (str) = joined
```

노드 상태 확인하기

```
# journalctl -b | grep -i error
```

```
# journalctl -u pcsd.service -perr -fl
```

연습문제

보안

ACL

FUNDAMENTAL

- PACEMAKER RESOURCE
- BASIC COMMANDS

ACL

ACL은 페이스메이커에서 사용하는 root계정 혹은 hacluster를 사용하지 않고, 다른 사용자를 구성한다. 이를 통해서 CIB를 구성할 수 있다.

일반 사용자를 생성하여, 페이스메이커 모니터링을 위해서 CIB접근 할 수 있도록 한다.

CIB: Cluster Information Base. 클러스터 구성 정보는 CIB를 통해서 구성 및 생성이 된다.

ACL

```
node4# adduser rouser
```

```
node4# echo centos | passwd --stdin rouser
```

```
node4# usermod -aG haclient rouser
```

```
node4# pcs acl enable
```

```
node4# pcs acl role create read-only description="Read only access to cluster" read  
xpath /cib
```

```
node4# pcs acl user create rouser read-only
```

```
node4# pcs acl
```

```
node4# pcs client local-auth
```


ACL 연습문제

사용자 cluster-monitor를 생성하세요.

1. 해당 사용자는 모든 시스템 자원에 대해서 읽기만 가능합니다.
2. 암호는 readworld라고 선언합니다.

ALERT

페이스메이커에서는 에이전트 스크립트를 지원한다. 이 스크립트는 보통 30초에 한번씩 동작한다. 모니터링 스크립트는 `"/usr/share/pacemaker/alerts/` 디렉터리 밑에서만 동작한다.

ALTER

```
node4# install --mode=0755 /usr/share/pacemaker/alerts/alert_file.sh.sample  
/var/lib/pacemaker/alert_file.sh
```

```
node4# touch /var/log/pcmk_alert_file.log
```

```
node4# chown hacluster:haclient /var/log/pcmk_alert_file.log
```

```
node4# chmod 600 /var/log/pcmk_alert_file.log
```

```
node4# pcs alert create id=alert_file description="Log events to a file."  
path=/var/lib/pacemaker/alert_file.sh
```

```
node4# pcs alert recipient add alert_file id=my-alert_logfile  
value=/var/log/pcmk_alert_file.log
```

```
node4# pcs alert
```

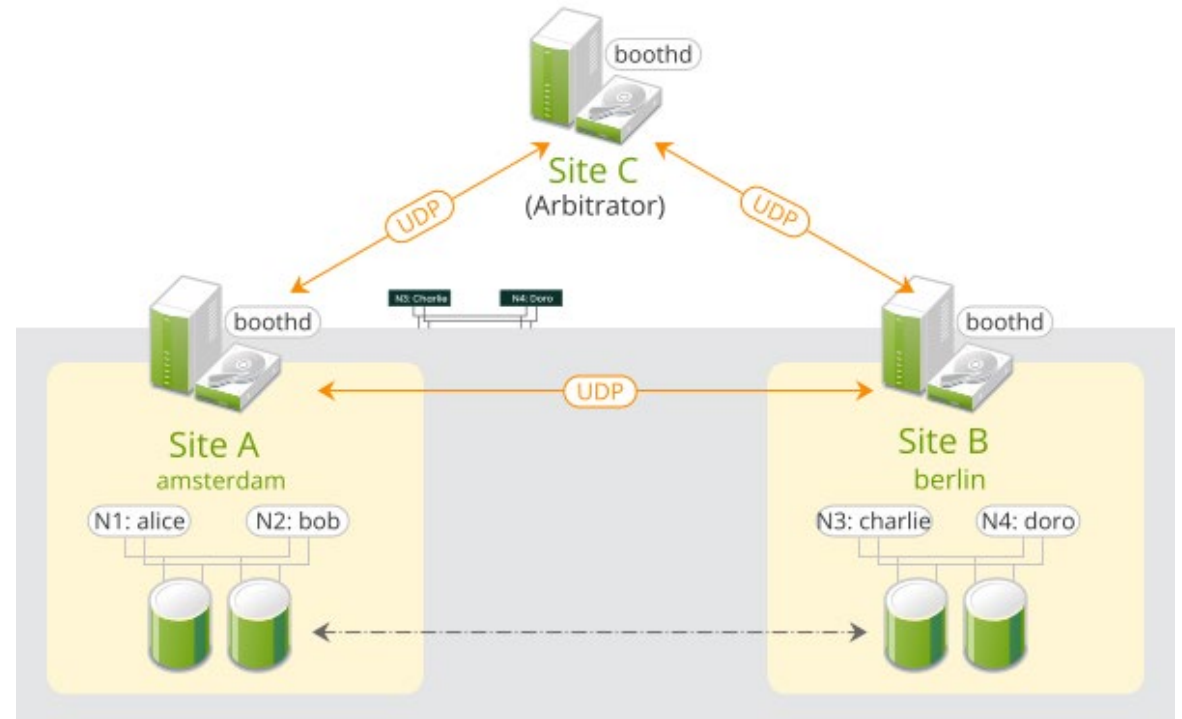
BOOTH

여기서는 다루지 않지만, 두 개의 사이트에서 H/A클러스터를 구성한 후, 특정 사이트에서 장애가 발생하면, 다른 사이트에 구성이 되어 있는 클러스터(Booth)가 중재인(Arbitrator)로 동작한다.

이를 구성하기 위해서는 자원이 많이 필요하기 때문에, 이 교육에서는 다루지 않는다.

BOOTH

Booth는 왼쪽과 같이 구성한다.
각각 부스는 중재자 클러스터를 통해서 장애가 발생 시,
Site A에서 Site B로 자원을 전달한다.

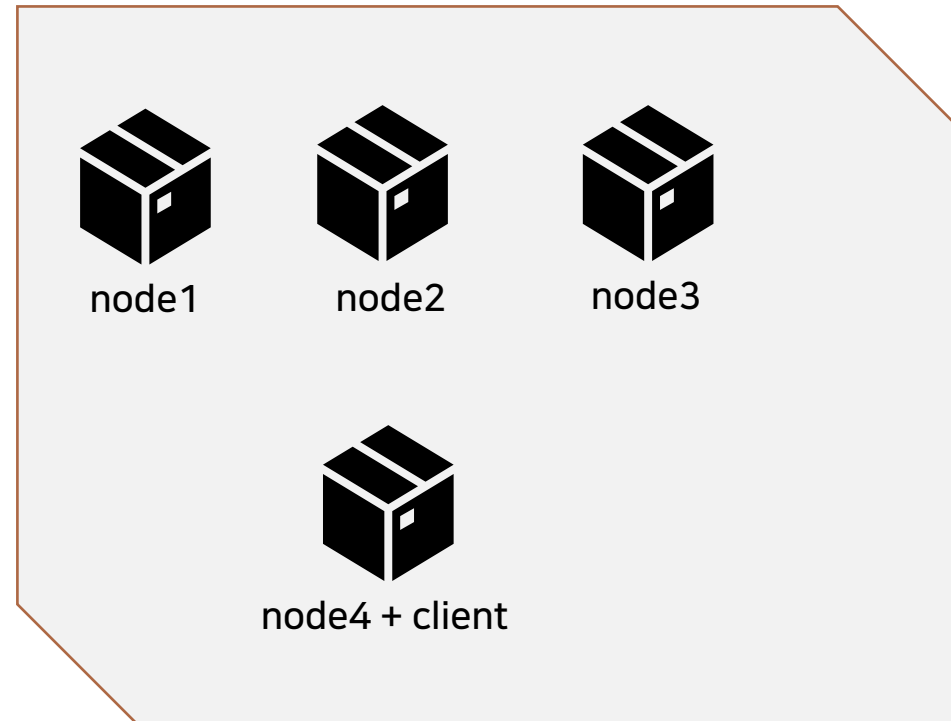


CLIENT

페이스 메이커에서 클라이언트를 구성하기 위해서는 클러스터 내부에 관리 용도를 위한 노드가 필요하다. 이 랩에서는 node4번이 클라이언트 역할을 하고 있다.

CLUSTER

한 개 이상의 노드가 구성이 되어 있는 멤버(member)를 클러스터라고 부른다.



CLUSTER COMMAND

node4# pcs cluster status

node4# pcs cluster config

node4# pcs cluster auth

node4# pcs cluster enable

node4# pcs cluster start

CONFIG/COMMAND

클러스터에 구성이 된 설정을 확인하기 위해서는 config명령어를 통해서 "resource", "stonith", "fence", "OCF" 에이전트 리소스 확인이 가능하다.

```
# pcs config
```

CONSTRAINT(제한)

리소스가 클러스터에 구성이 되면, 해당 자원이 클러스터에서 동작하는 범위 혹은 기능을 제한한다.
이를 통해서 자원이 클러스터 어떤 노드에서 동작하는지 설정한다.

로케이션(location)

어떤 클러스터 노드에서 동작할지 설정한다.

순서(order)

여러 자원이 있을 때, 어떠한 순서로 동작할지 결정한다.

COLOCATION

colocation is where resource will locate or placed relative to other resource

위치 선언(colocation)를 한다. 위치 선언에는 이전에 이야기 하였던, Constraint, Order가 복합적으로 구성이 된 자원이다.

Collocate(B, A)

<rsc_colocation from=B to=A/>

Decide where to put A, then put B there too

Include B's preferences when deciding where to put A If A cannot run anywhere, B can't run either

If B cannot run anywhere, A will be unaffected

COLOCATION

- $\text{number} > \text{INFINITY} = \text{INFINITY}$
- $\text{number} < -\text{INFINITY} = -\text{INFINITY}$
- $\text{number} + \text{INFINITY} = \text{INFINITY}$
- $\text{number} - \text{INFINITY} = -\text{INFINITY}$
- $\text{INFINITY} - \text{INFINITY} = -\text{INFINITY}$

COLOCATION

- resource(A, priority=5)
- resource(B, priority=50)
- location(A, node1, 100)
- location(A, node2, 10)
- location(B, node2, 1000)
- collocate(B, A)

DR

페이스메이커에 새로 도입된 기능. 한 개 이상의 클러스터가 구성이 되어 있는 경우, 동작중인 클러스터가 장애가 발생하여 동작하지 못하는 경우, D/R(Disaster recovery) 클러스터가 기존의 H/A 클러스터를 대신한다.

이 기능은 RHEL/CentOS 8부터 사용이 가능하다. 이 교육에서는 다루지 않는다.

DR COMMAND

```
node4# pcs host auth -uhacluster -phacluster node1.example.com  
node2.example.com node3.example.com node4.example.com
```

```
node4# pcs cluster setup node1.example.com node2.example.com  
node3.example.com node4.example.com
```

```
node4# pcs cluster setup DRsite node3.example.com node4.example.com --start
```

```
node4# pcs dr set-recovery-site node3.example.com
```

```
node4# pcs dr config
```

```
node4# pcs dr status
```

HOST

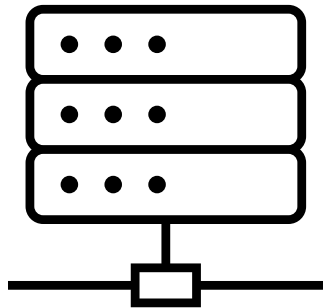
페이스메이커에서 제일 작은 구성원은 **노드 (node)** 혹은 **호스트(host)**라고 부른다. 이들은 최소 한 개 이상이 클러스터에 존재해야 한다. 존재하지 않는 경우 클러스터는 구성을 할 수 없다. 모든 클러스터는 기본 관리자 계정은 'hacluster'계정을 가지고 있다. 이를 확인하기 위해서는 'pcs host'명령어를 통해서 확인이 가능하다.

```
node4# pcs node == host
```

```
node4# pcs cluster
```


NODE

호스트의 다른 이름. 노드는 클러스터에 최소로 구성이 되어 있어야 하며, 이를 통해서 클러스터에서 사용하는 자원 생성이 가능하다. 클러스터에 구성된 자원들은 **그룹/제약/순서**를 통해서 동작 방식을 결정한다.



NODE COMMAND


```
node4# pcs cluster status
```

```
node4# pcs node standby node1.example.com
```

```
node4# pcs cluster status
```

PCSD

페이스메이커(pcs, pacemaker)는 pcsd서비스를 통해서 자동화를 수행한다. 페이스메이커는 웹 기반 GUI를 제공한다. 포트 번호는 2224/TCP로 지원한다.

 HIGH AVAILABILITY
MANAGEMENT

hacluster ▼

MANAGE CLUSTERS

PERMISSIONS

MANAGE CLUSTERS

[x Remove](#)

[+ Add Existing](#)

[x Destroy](#)

[+ Create New](#)

All	✓	⚠	✖	x
0	0	0	0	0

INFORMATION ABOUT CLUSTERS

Select a cluster to view more detailed cluster information

속성(PROPERTY)

The property is configuration to cluster value such as corosync, quorum values.

Cluster properties control how the cluster behavior with property value.

PROPERTY COMMAND

```
# pcs property
```

Cluster Properties:

```
cluster-infrastructure: corosync
```

```
cluster-name: ha_cluster_lab
```

```
dc-version: 2.1.5-5.el8-a3f44794f94
```

```
have-watchdog: false
```

```
no-quorum-policy: freeze
```

```
# pcs property set maintenance-mode=true
```

PROPERTY COMMAND

pcs property

Cluster Properties:

cluster-infrastructure: corosync

cluster-name: ha_cluster_lab

dc-version: 2.1.5-5.el8-a3f44794f94

have-watchdog: false

maintenance-mode: true

no-quorum-policy: freeze

QDEVICE

"qdevice" is "quorum device".

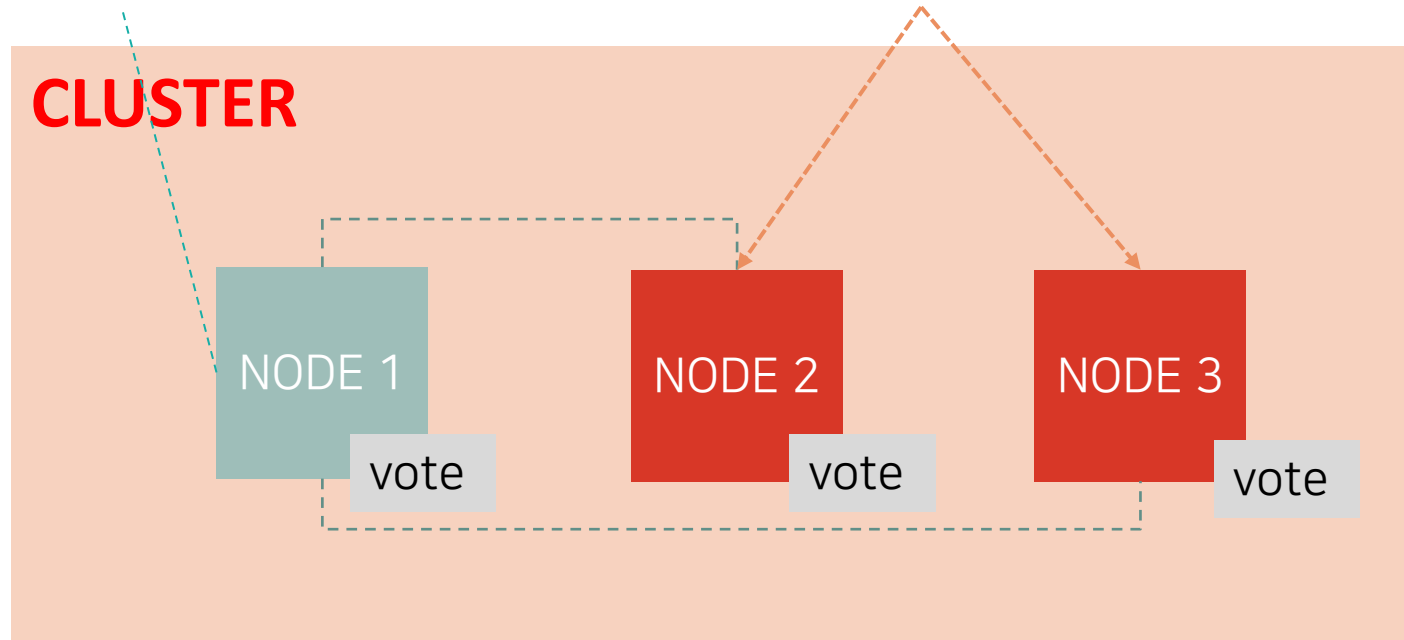
this is not device and agent. It's third-party arbitration device for the cluster.

Normally a quorum device is recommended for clusters with an even number of nodes. But, when the cluster is going to be two-node cluster status, the quorum device is better for survive in split-brain situation.

QDEVICE

```
# pcs qdevice status net --full
```

brain split



QDEVICE COMMAND

Install these packages form High Availability repository.

```
node1# dnf --enablerepo=ha install corosync-qdevice
```

```
node1# dnf --enablerepo=ha install pcs corosync-qnetd
```

```
node1# yum install pcs corosync-qnetd
```

```
node1# systemctl enable --now pcsd.service
```

```
node1# pcs qdevice setup model net --enable --start
```

```
node1# pcs qdevice status net --full
```

```
node1# firewall-cmd --permanent --add-service=high-availability
```

```
node2# pcs cluster auth qdevice
```

QDEVICE COMMAND

node2# pcs quorum config

node2# pcs quorum status

node2# pcs quorum device add model net W

host=qdevice algorithm=ffsplit

node2# pcs quorum config

node2# pcs quorum status

node2# pcs quorum device status

node2# pcs qdevice status net --full

QDEVICE COMMAND

node2# pcs qdevice start net

node2# pcs qdeivce stop net

node2# pcs qdevice enable net

node2# pcs qdevice disable net

node2# pcs qdevice kill net

QDEVICE COMMAND

node2# pcs quorum device update model ~~W~~

algorithm=lms

node2# pcs quorum device remove

node2# pcs quorum device status

node2# pcs qdevice destroy net

QUORUM

- quorum
- The quorum is voting system for cluster nodes.
- Every cluster nodes has a vote for vote-quorum system. If some resources or nodes can't vote the object will be fencing and detached from system.



net started now we've got

QUORUM COMMAND

```
node2# pcs quorum status
```

```
Quorum information
```

```
-----
```

```
Date:                Sun Feb 26 02:09:16 2023
```

```
Quorum provider:     corosync_votequorum
```

```
Nodes:               2
```

```
Node ID:             1
```

```
Ring ID:             1.40
```

```
Quorate:             Yes
```

QUORUM COMMAND

Votequorum information

Expected votes: 2
Highest expected: 2
Total votes: 2
Quorum: 1
Flags: 2Node Quorate WaitForAll

Membership information

Node id	Votes	Qdevice Name
1	1	NR node2.example.com (local)
2	1	NR node3.example.com

RESOURCE

- The resource is a service managed by Peacemaker. The resource is kind of agent for standardized interface for managing the service. This allows Pacemaker to be agnostic the services it manages.
- We don't need to understand about the service works behind of resource agent.

RESOURCE

```
nodeX# ls /usr/share/resource-agents/ocft/configs
```

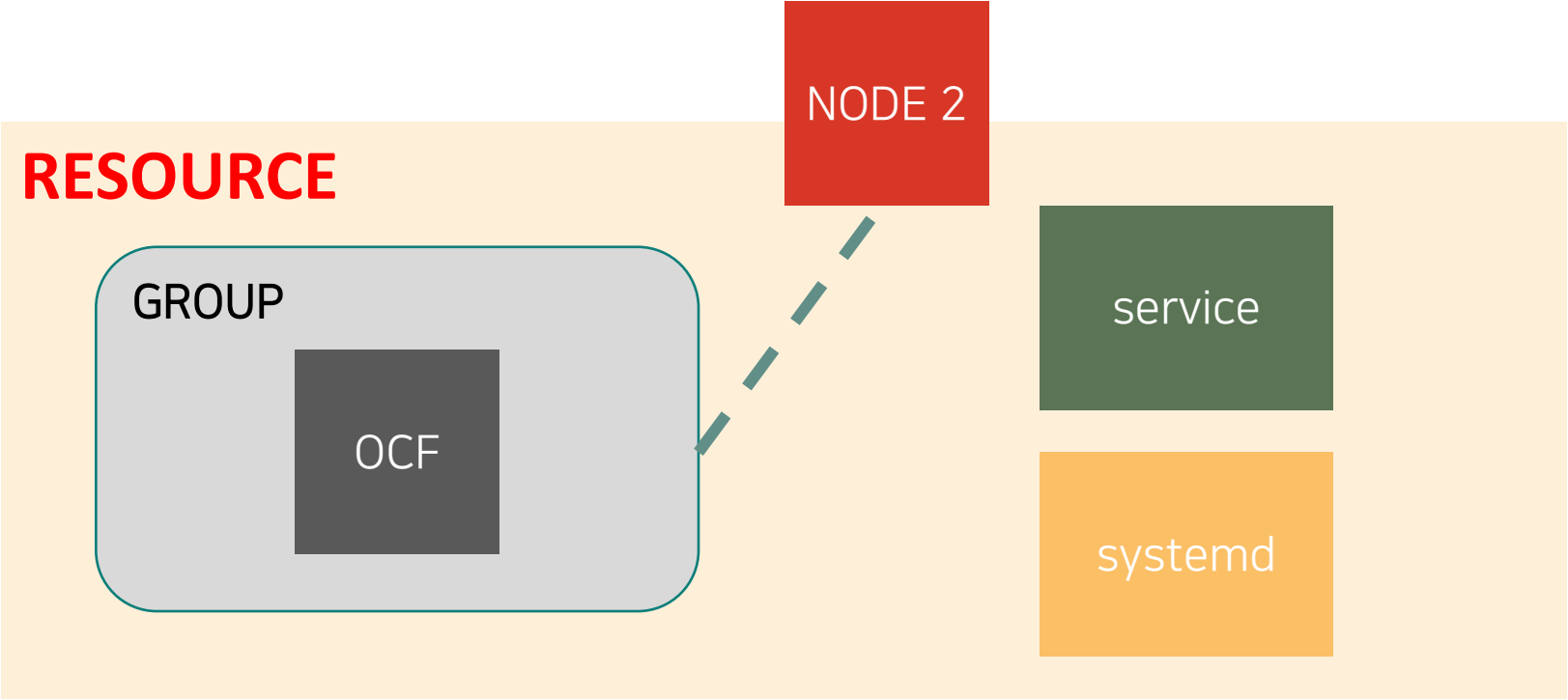
```
apache    exportfs-multidir IPAddr2v6 jboss  mysql-proxy pgsql  SendArp  
Xen
```

```
db2       Filesystem      IPsrcaddr LVM   named    portblock sg_persist  
Xinetd
```

```
drbd.linbit IPAddr2        IPv6addr MailTo nfsserver postfix tomcat
```

```
exportfs  IPAddr2v4      iscsi  mysql oracle  Raid1  VirtualDomain
```

RESOURCE



RESOURCE

The resources have a classes below these.

- OCF
- LSB
- system
- Upstart(deprecated)
- Service
- Fencing
- Nagios

OCF

- The Open Cluster Framework
- The Open Cluster Framework (OCF) Resource Agent API is a ClusterLabs standard for managing services. It is the most preferred since it is specifically designed for use in a Pacemaker cluster.

OCF

- OCF agents are scripts that support a variety of actions including start, stop, and monitor. They may accept parameters, making them more flexible than other classes. The number and purpose of parameters is left to the agent, which advertises them via the meta-data action.
- Unlike other classes, OCF agents have a provider as well as a class and type.

systemd

- Most Linux distributions use Systemd for system initialization and service management. Unit files specify how to manage services and are usually provided by the distribution.
- Pacemaker can manage systemd services. Simply create a resource with systemd as the resource class and the unit file name as the resource type. Do not run `systemctl enable` on the unit.

LSB

Linux Standard Base

LSB resource agents, also known as SysV-style, are scripts that provide start, stop, and status actions for a service.

They are provided by some operating system distributions. If a full path is not given, they are assumed to be located in a directory specified when your Pacemaker software was built (usually `/etc/init.d`).

In order to be used with Pacemaker, they must conform to the LSB specification as it relates to init scripts.

STONITH

The Stonith class is used for managing fencing devices, discussed later in Fencing.

STATUS

The status command will show the cluster state. The status collect from pcsd, corosync and agent information.

STATUS COMMAND

pcs status

pcs resource status <NAME>

pcs status nodes

STONITH

"Shoot the other node in the head" aka fencing. The Stonith for protects your data from being corrupted by rogue nodes.

The command example will not work correctly.

STONITH COMMAND

```
node2# pcs stonith list
```

```
node2# dnf search fence-agents-all
```

```
node2# dnf install fence-agents-ipmilan
```

```
node2# pcs stonith describe fence_ipmilan
```

```
node2# pcs stonith create ipmi-fence-node1 fence_ipmilan
```

```
pcmk_host_list="node1" ipaddr="10.0.0.1" login="xxx" passwd="xxx" lanplus=1  
power_wait=4
```

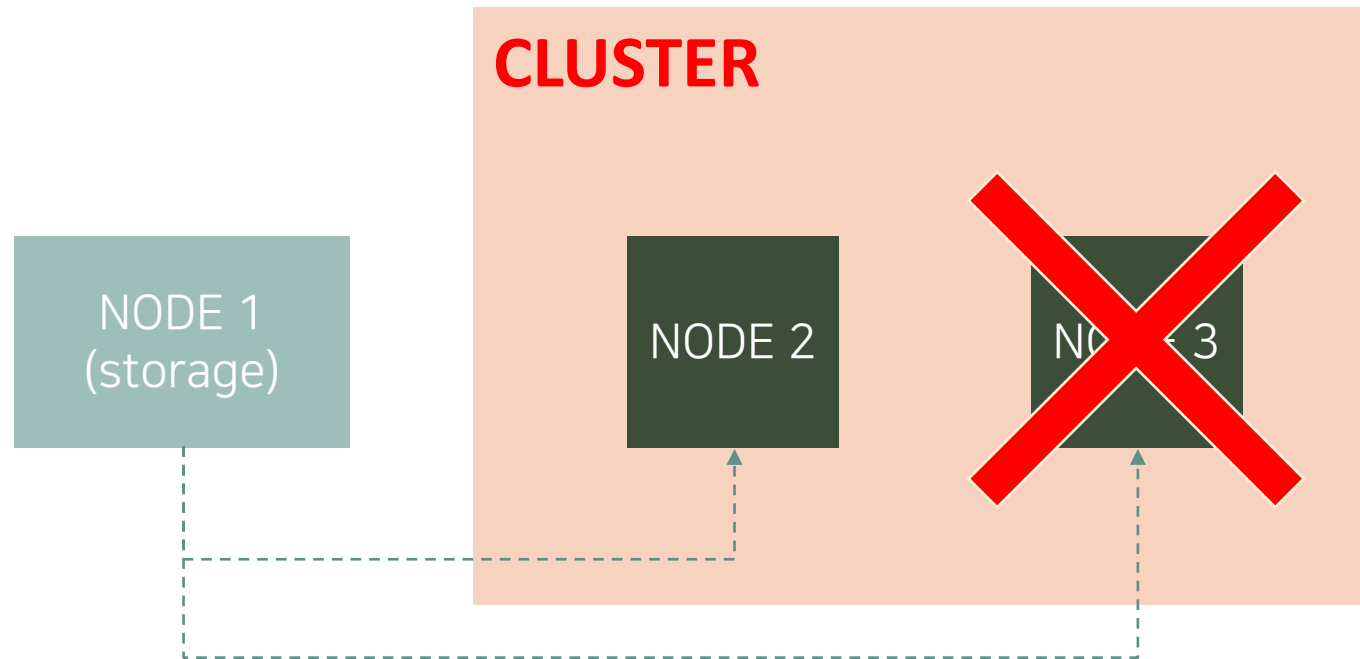
STONITH COMMAND

```
# pcs -f stonith_cfg stonith
# pcs -f stonith_cfg property set W
stonith-enabled=true
# pcs -f stonith_cfg property
# pcs cluster stop node2
# stonith_admin --reboot node2
```

FENCE

FENCE TEST WITH iSCSI

FENCE



FENCE

```
node2# dnf install --enablerepo=ha fence-agent-scsi -y
```

```
node2# ls -l /dev/disk/by-id | grep sda | grep wwn
```

```
node2# pcs stonith create scsi-shooter fence_scsi
```

```
pcmk_host_list="node1.example.com node2.example.com" devices=/dev/disk/by-id/wwn-<SCSI-DISK-ID> meta provides=unfencing
```

```
node2# pcs stonith config scsi-shooter
```

```
node2# pcs status
```

```
node2# pcs stonith fence node2.example.com
```

```
node2# pcs status
```

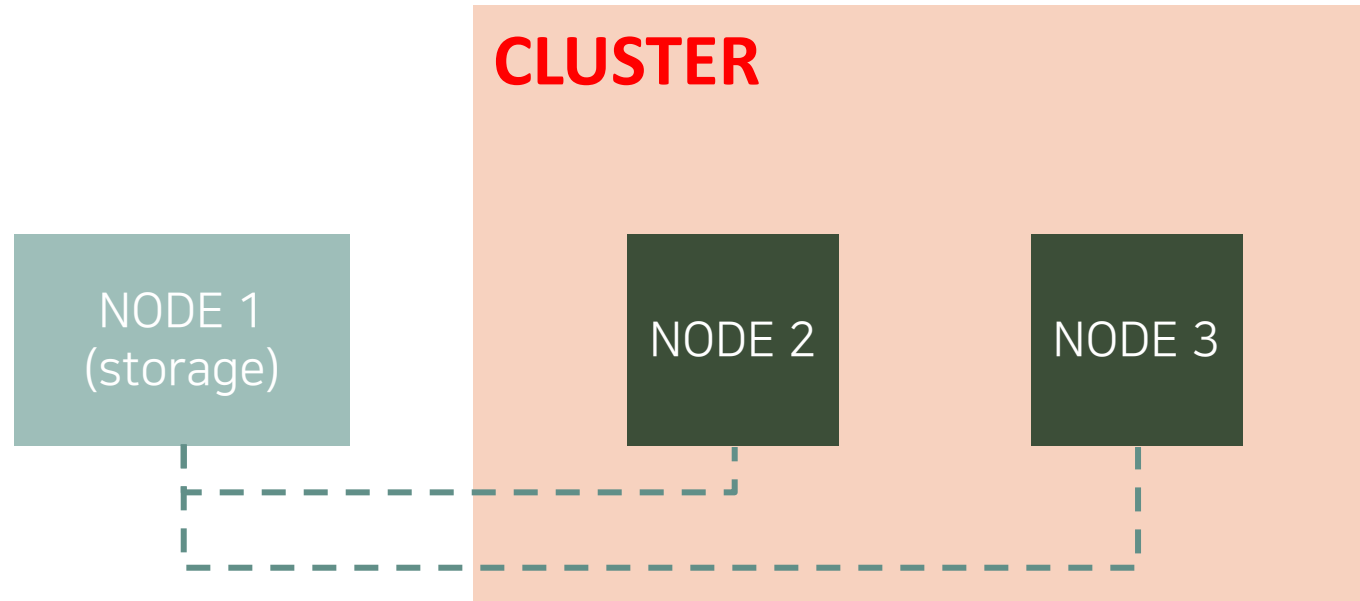

스토리지

LVM2

LVM

CLUSTER LVM2

LVM



LVM

```
node2/3# vi /etc/lvm/lvm.conf
```

```
system_id_source = "uname"
```

```
node2# parted --script /dev/sda "mklabel msdos"
```

```
node2# parted --script /dev/sda "mkpart primary 0% 100%"
```

```
node2# parted --script /dev/sda "set 1 lvm on"
```

```
node2# pvcreate /dev/sda1
```

```
node2# vgcreate vg_ha_lvm /dev/sda1
```

```
node2# vgs -o+systemid
```

```
node2# lvcreate -l 100%FREE -n lv_ha_lvm vg_ha_lvm
```

LVM

```
node2# mkfs.xfs /dev/vg_ha_lvm/lv_ha_lvm
```

```
node2# vgchange vg_ha_lvm -an
```

```
node2# lvm pvscan --cache --activate ay
```

```
node2# pcs resource create lvm_ha_iscsi ocf:heartbeat:LVM-activate  
vg_name=vg_ha_lvm vg_access_mode=system_id --group ha_lvm_group
```

```
node2# pcs resource create lvm_ha_mount FileSystem device=/dev/vg_ha  
_lvm/lv_ha_lvm directory=/home/lvm_directory fstype=xfs --group ha_lvm_group
```

```
node2# pcs status
```

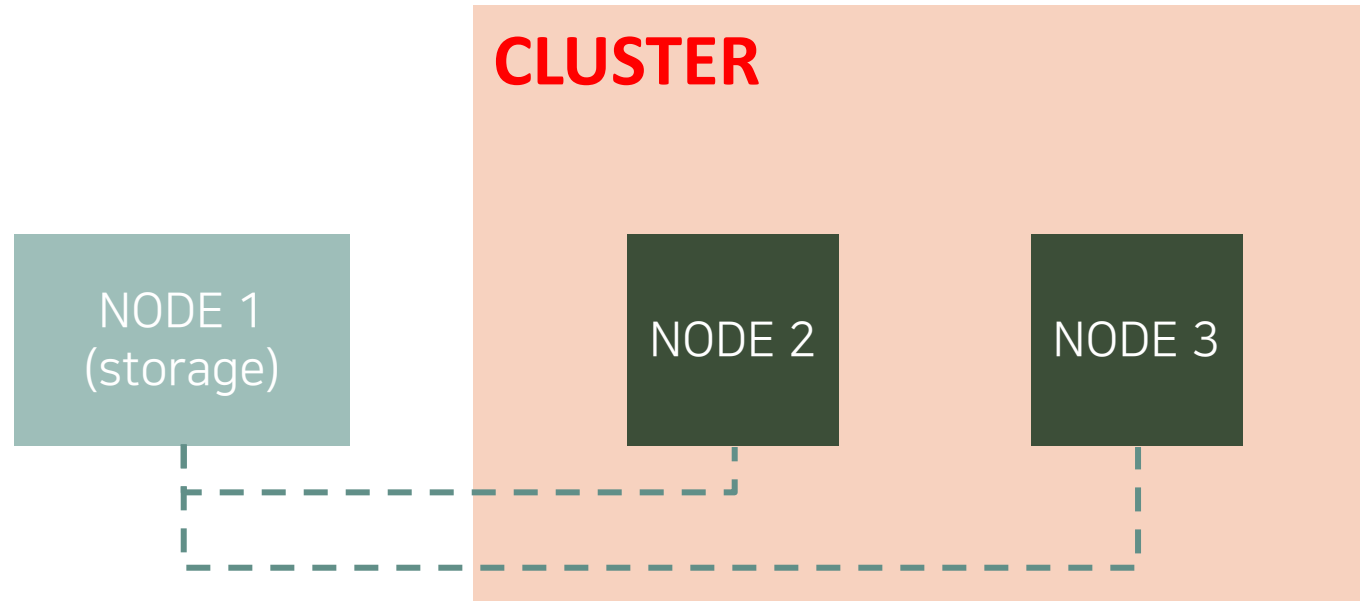
연습문제

스토리지

NFS

NFS

PACEMAKER



PACEMAKER

```
node2/3# firewall-cmd --add-service=nfs --permanent
```

```
node2/3# firewall-cmd --add-service={nfs3,mountd,rpc-bind} --permanent
```

```
node2/3# firewall-cmd --reload
```

```
node2/3# mkdir -p /home/nfs-share
```

```
node2# pcs resource create nfs_share_iscsi ocf:heartbeat:Filesystem device=/dev/vg_ha_iscsi/lv_ha_iscsi directory=/home/nfs-share  
fstype=xfs --group ha_iscsi_group
```

```
node2# pcs status
```

```
node2# mount | grep /home/nfs-share
```

```
node2# pcs resource create nfs_daemon ocf:heartbeat:nfsserver nfs_shared_infodir=/home/nfs-share/nfsinfo nfs_no_notify=true --  
group ha_iscsi_group
```

```
node2# pcs resource create nfs_vip ocf:heartbeat:IPAddr2 ip=192.168.100.250 nic=eth1 cidr_netmask=24 --group ha_iscsi_group
```

PACEMAKER

```
node2# pcs resource create nfs_notify ocf:heartbeat:nfsnotify source_host=192.168.100.250 --group ha_iscsi_group
```

```
node2# mkdir -p /home/nfs-share/nfs-root/share01
```

```
node2# pcs resource create nfs_root ocf:heartbeat:exportfs clientspec=192.168.100.0/255.255.255.0 options=rw,sync,no_root_squash  
directory=/home/nfs-share/nfs-root fsid=0 --group ha_iscsi_group
```

```
node1 # pcs resource create nfs_share01 ocf:heartbeat:exportfs clientspec=192.168.100.0/255.255.255.0  
options=rw,sync,no_root_squash directory=/home/nfs-share/nfs-root/share01 fsid=1 --group ha_iscsi_group
```

```
node2 # pcs status
```

```
node2 # showmount -e
```

```
node2/3 # mkdir -p /mnt/test_nfs
```

```
node2/3 # mount 192.168.100.250:/home/nfs-share/nfs-root/share01 /mnt
```

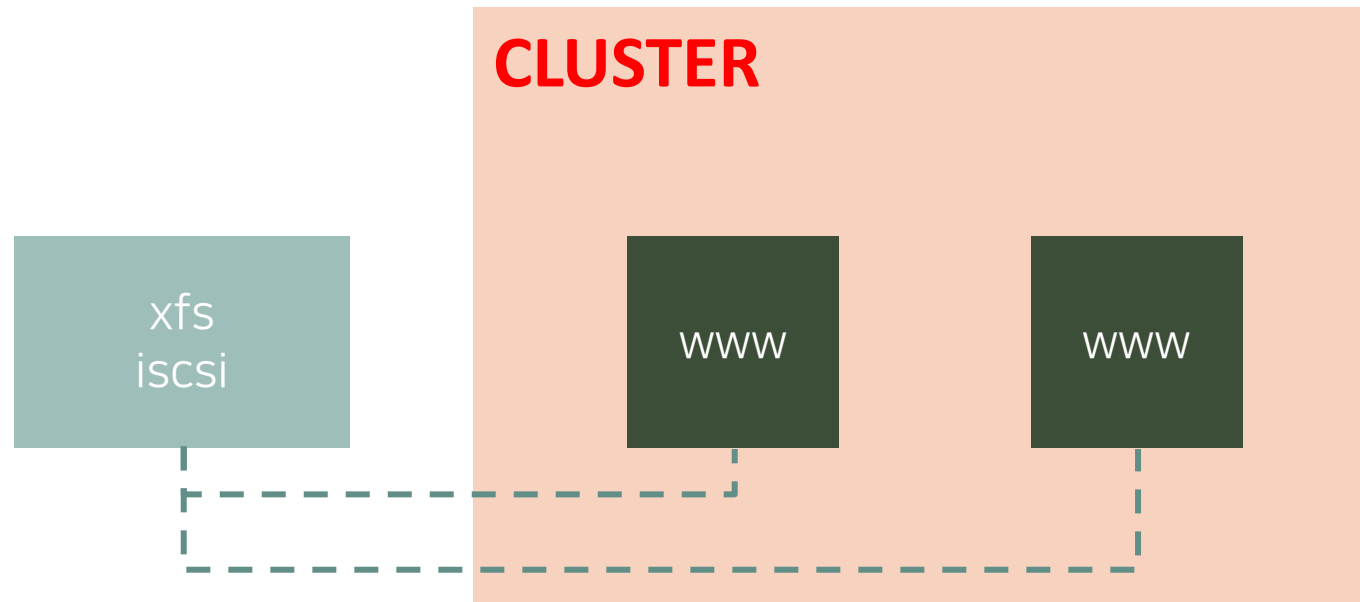
서비스 구성

WWW

WWW

- APACHE
- TOMCAT

PACEMAKER



APACHE

```
node1/2/3# dnf install httpd -y
```

```
node2/3# vi /etc/httpd/conf.d/server-status.conf
```

```
<Location /server-status>
```

```
    SetHandler server-status
```

```
    Require local
```

```
</Location>
```

```
node2/3# firewall-cmd --add-service={http,https} --permanent && firewall-cmd --runtime-to-permanent
```

```
node2/3# mkdir -p /mnt/html
```

```
node2/3# mount /dev/vg_ha_iscsi/lv_ha_iscsi /mnt/html
```

```
node2/3# echo "Hello World" > /mnt/html/index.html && umount /mnt/html/
```

```
node2# pcs resource create httpd_fs ocf:heartbeat:Filesystem device=/dev/vg_ha_iscsi/lv_ha_iscsi directory=/var/www fstype=xfs --group ha_group_iscsi
```

APACHE

```
node2# pcs resource create httpd_vip ocf:heartbeat:IPaddr2 ip=192.168.100.240  
cidr_netmask=24 --group ha_group_iscsi
```

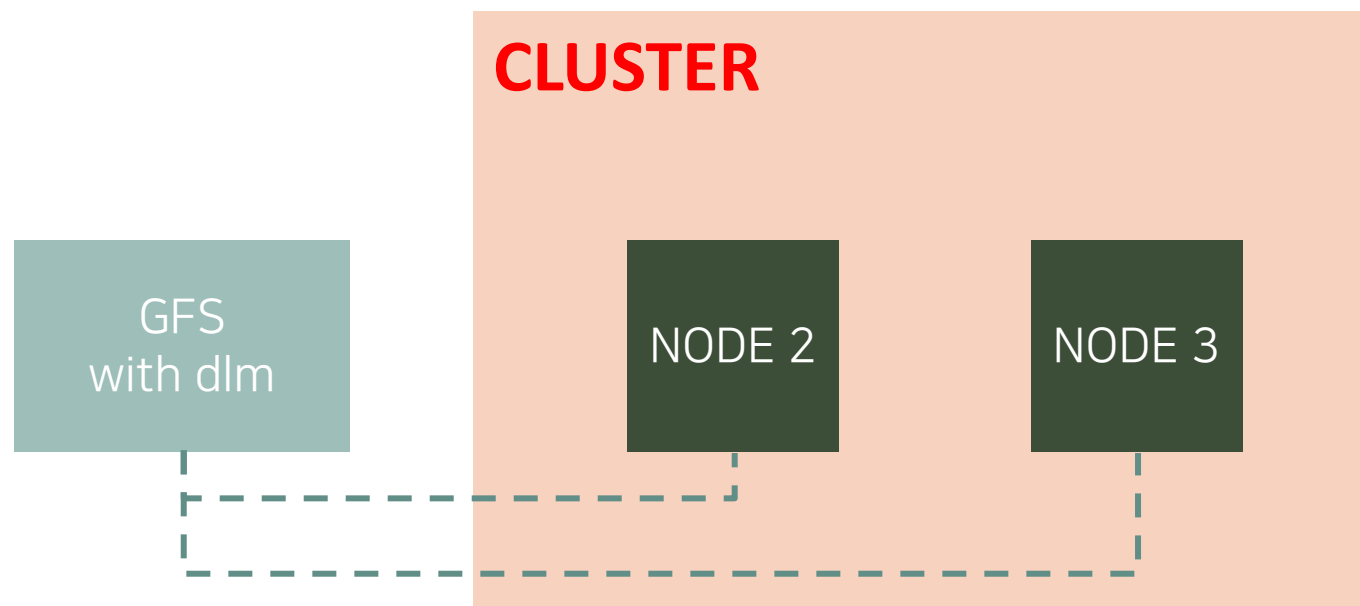
```
node2# pcs resource create website ocf:heartbeat:apache  
configfile=/etc/httpd/conf/httpd.conf statusurl=http://127.0.0.1/server-status  
--group ha_group
```

```
node2# pcs status
```

```
node2# restorecon -RFvvv /var/www/
```

```
node2# curl http://192.168.100.240/index.html
```


PACEMAKER



GFS

```
node2# dnf --enablerepo=ha,resilientstorage -y install lvm2-lockd gfs2-utils dlm
```

```
node2/3# vi /etc/lvm/lvm.conf
```

```
use_lvmlockd = 1
```

```
node2/3# systemctl enable --now dlm lvmlockd lvmlocks
```

```
node2# pcs property set no-quorum-policy=freeze
```

```
node2# pcs resource create dlm ocf:pacemaker:controld op monitor interval=30s  
on-fail=fence --group locking
```

GFS

```
node2# pcs resource clone locking interleave=true
```

```
node2# pcs resource create lvmlockdd ocf:heartbeat:lvmlockd op monitor  
interval=30s on-fail=fence --group locking
```

```
node2# pcs status
```

GFS

```
node2# parted --script /dev/sdb "mklabel gpt"
```

```
node2# parted --script /dev/sdb "mkpart primary 0% 100%"
```

```
node2# parted --script /dev/sdb "set 1 lvm on"
```

```
node2# pvcreate /dev/sdb1
```

```
node2# vgcreate --shared vg_gfs2 /dev/sdb1
```

```
node2# vgs
```

```
node2# vgchange --lock-start vg_gfs2
```

```
node2# lvcreate -l 100%FREE -n lv_gfs2 vg_gfs2
```

GFS

```
node2# mkfs.gfs2 -j2 -p lock_dlm -t ha_cluster_lab:gfs2 /dev/vg_gfs2/lv_gfs2
```

```
node2# pcs resource create shared_lv ocf:heartbeat:LVM-activate lvname=lv_gfs2  
vgname=vg_gfs2 activation_mode=shared vg_access_mode=lvmlockd --group  
shared_vg
```

```
node2# pcs resource clone shared_vg interleave=true
```

```
node2# pcs constraint order start locking-clone then shared_vg-clone
```

PACEMAKER

```
node2# pcs constraint colocation add shared_vg-clone with  
locking-clone
```

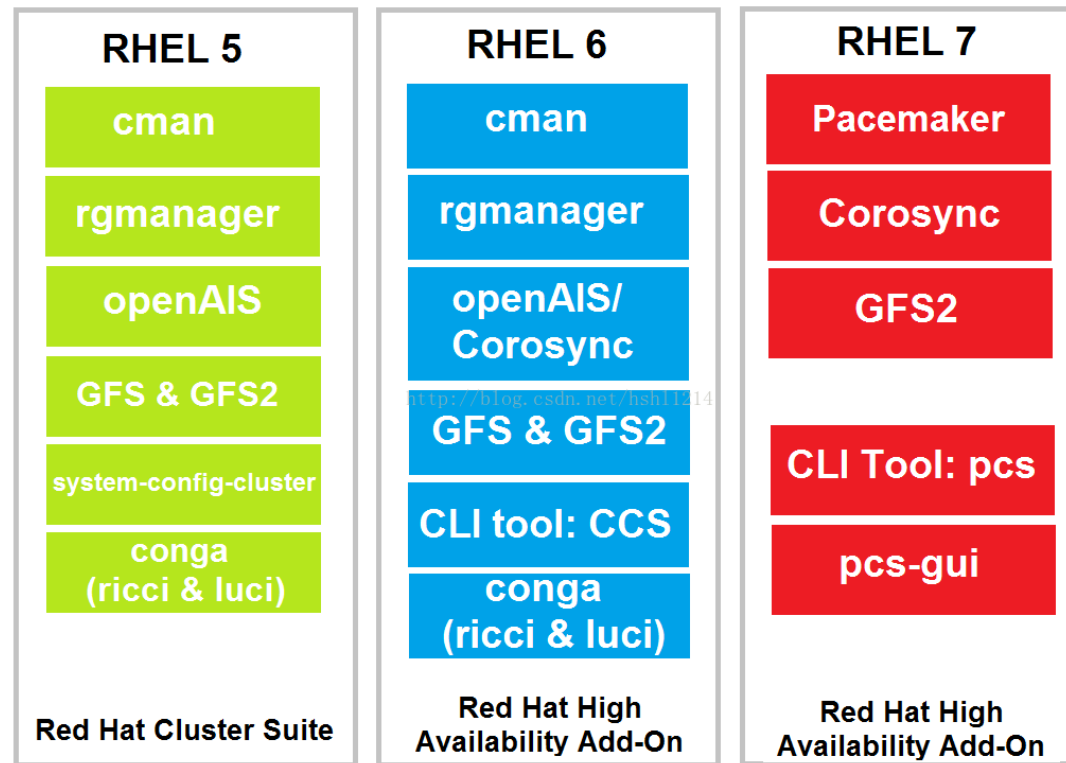
```
node2# pcs resource create shared_fs  
ocf:heartbeat:Filesystem device="/dev/vg_gfs2/lv_gfs2"  
directory="/home/gfs2-share" fstype="gfs2" options=noatime  
op monitor interval=10s on-fail=fence --group shared_vg
```

```
node2# pcs status
```

```
node2# mount | grep gfs2-share
```

```
node3# mount | grep gfs2-share
```

PACEMAKER



TWO NODE

호스트 나누기

PACEMAKER