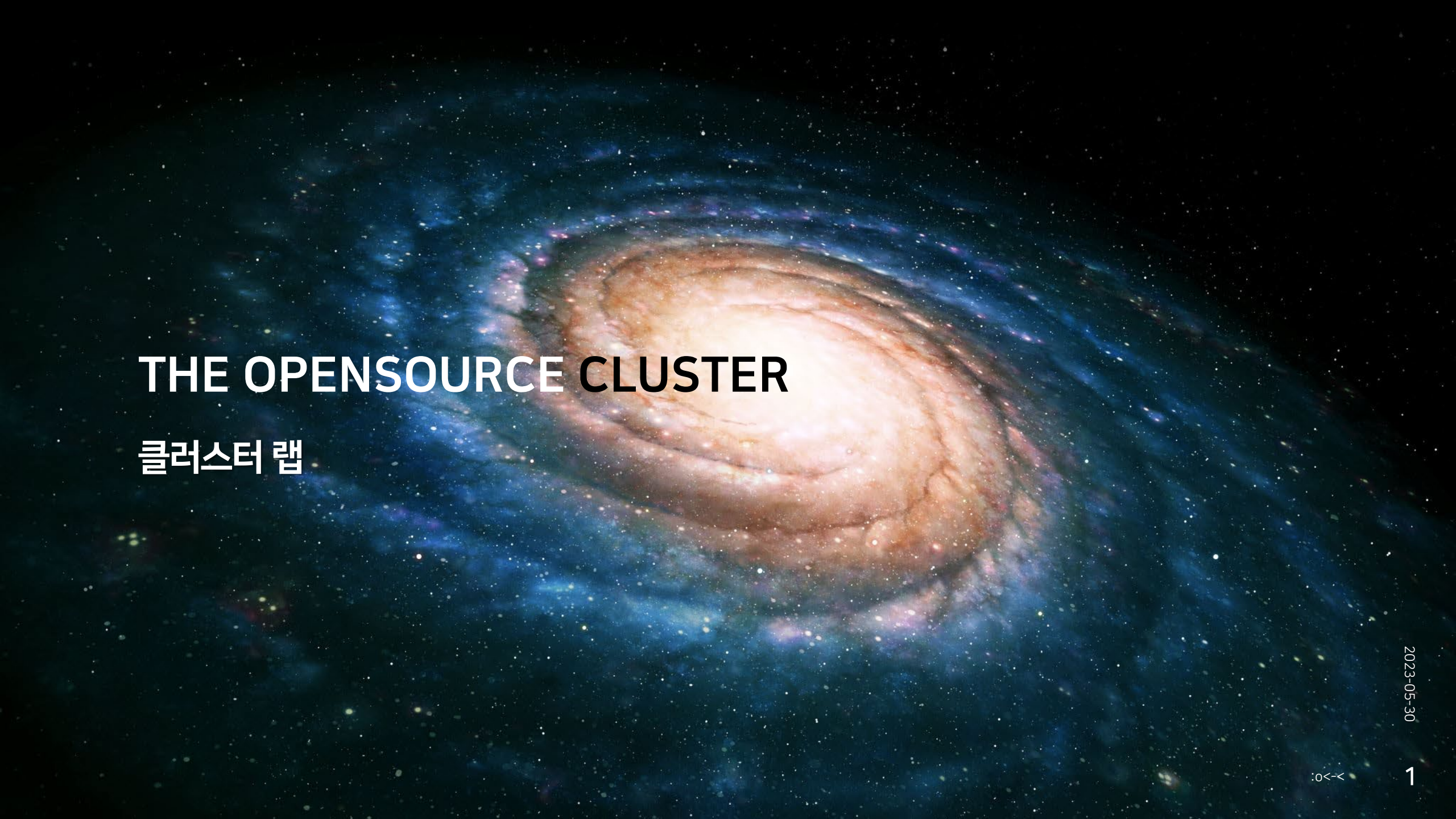

THE OPENSOURCE CLUSTER
클러스터 랩

강사소개

이름: 최 국현

메일주소: tang@linux.com

랩은 로키가 아닌 **CentOS-9-Stream**를 사용하세요. 현재 **Rocky 9**리눅스와 CentOS-Stream의 패키징 차이가 있습니다. 동작이 조금 다르게 됩니다.

목차

쿠버네티스 101

목차

- 랩 소개
- 랩 구성
- 페이스메이커 소개
- 클러스터 구성 준비
- ISCSI 노드 구성
- HA클러스터 구성 및 확인

목차

- DRBD 구성
- 자원 설명
- 서비스 구성
- 자원 이동
- 종합문제

소개

페이스메이커

소개

01

페이스메이커 처음 사용자

02

페이스메이커 기능 확인

페이스메이커 기반으로 H/A; D/R 구현을 원하는 인프라 엔지니어

강의일정

강의 일정은 아래와 같이 진행이 될 예정 입니다.

DAY 1:

- 랩 소개
- 랩 구성
- 페이스메이커 소개
- 클러스터 구성 준비
- ISCSI 노드 구성

DAY 2:

- HA클러스터 구성 및 확인
- DRBD 구성
- 자원 설명

강의일정

DAY 3/4/5

- 서비스 구성
- 자원 이동
- 종합문제

강사

강사

강사

이름: 최국현

메일: tang@linux.com, 회신은 다른 메일 주소로 드리고 있습니다. :)

사이트: tang.dustbox.kr

언제든지 질문 및 요청 환영 입니다.

랩 소개

랩 구성

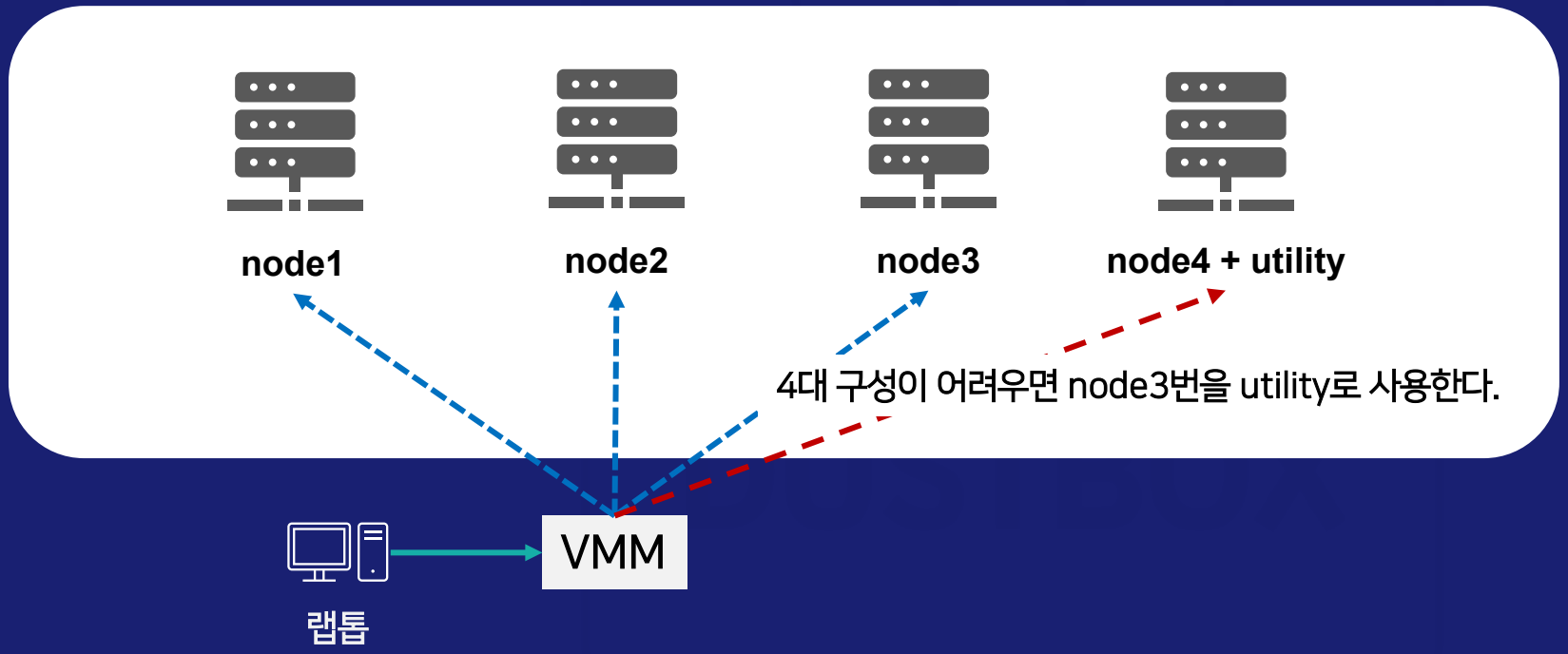
페이스메이커

다음과 같은 대상으로 교육을 진행 합니다.

1. 페이스메이커 처음 사용자
2. 페이스메이커 기능 및 동작 확인
3. 페이스메이커를 통한 스토리지 이중화
4. 페이스메이커를 통한 Disaster Recovery 및 Booth 개념 구성
5. 수세/레드햇 명령어 차이점. 그리고, pcs/crm 명령어 활용

기본과정 랩

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



DUSTBOX

2023-05-30

:o<-<

오픈스택 랩 정보

- **웹 주소**: https://vlab.dustbox.kr
- **사용자 계정**: pace1~20
- **비밀번호**: pacemaker
- **도메인**: pcs-training

위 정보는 강의 진행 시 접근이 가능하며, 이후에는 접근이 불가능 합니다.

랩 구성

하이퍼브이

오픈스택

VirtualBMC

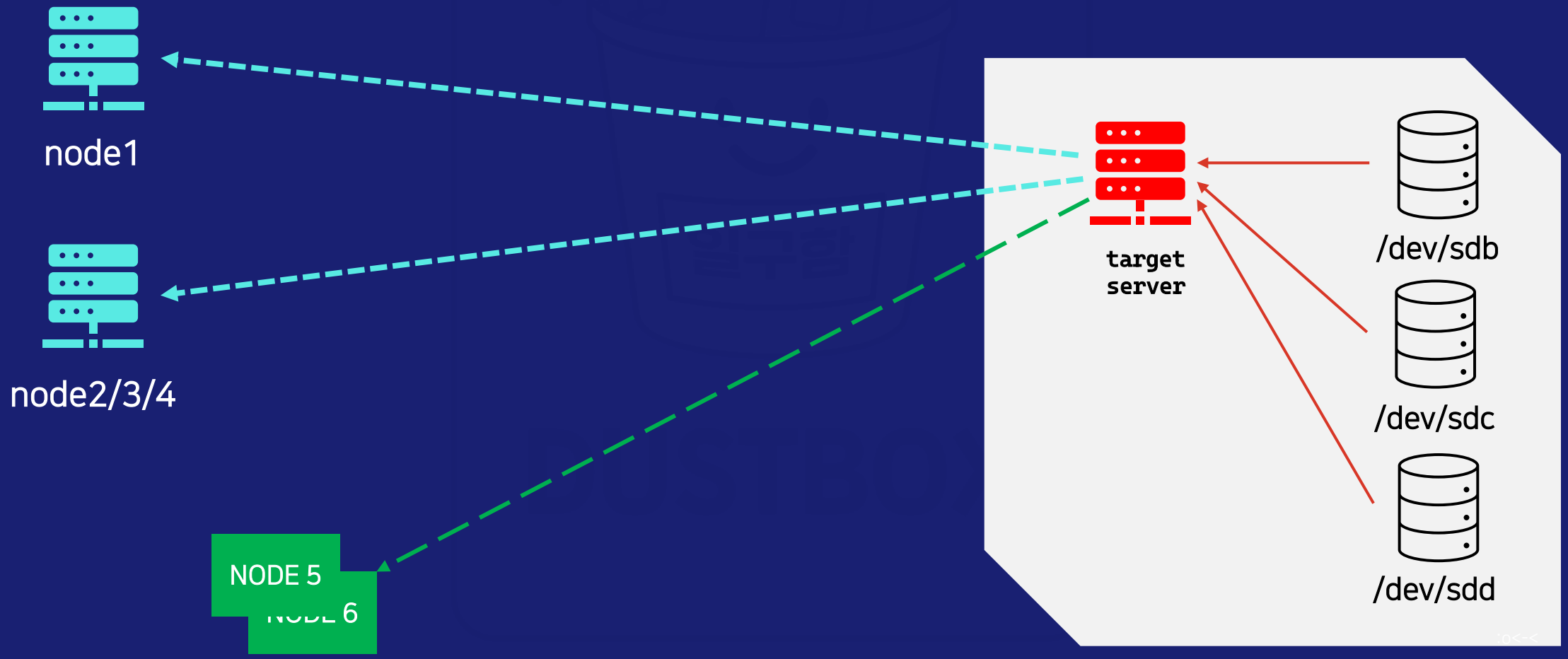
2023-05-30

랩 구성

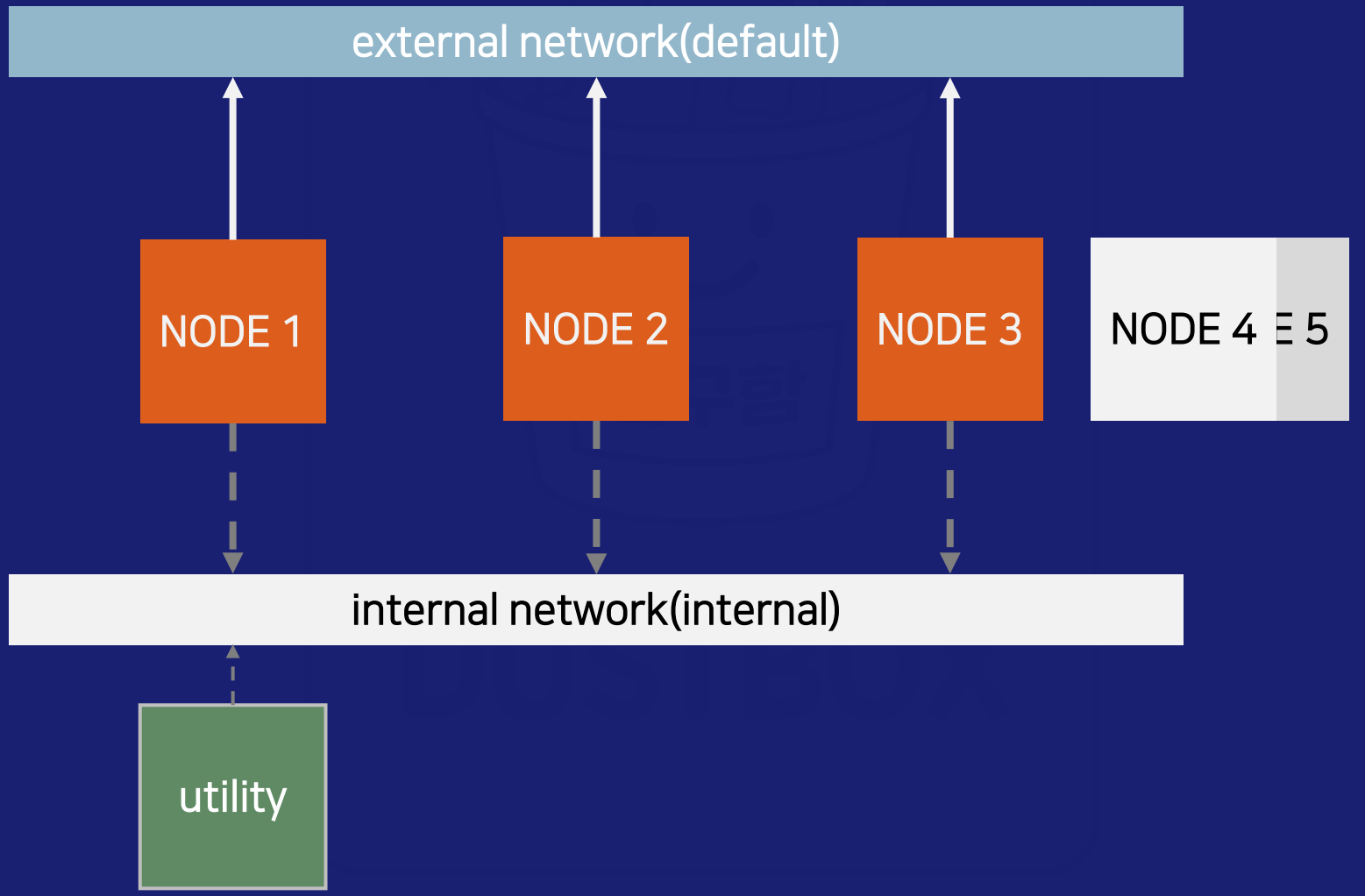
강의 시작 전, 가상머신을 리눅스에서 설치 한다. 페이스 메이커를 사용하기 위해서 올바르게 저장소 구성을 한다. 여기서 사용하는 하이퍼바이저는 윈도우 10/11 Pro기반의 하이퍼브이 기반으로 사용한다.

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


node1
node2/3/4
target
server
/dev/sdb
/dev/sdc
/dev/sdd
NODE 5
NODE 6

네트워크



DUSTBOX

2023-05-30

:o<-<

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.

- libvirtd
- virsh
- virt-builder

만약, **stonith, quorum**를 **two-node**에서 적용하지 않으려면 다음과 같이 옵션 적용.

```
pcs property set stonith-enabled=false
pcs property set no-quorum-policy=ignore
```

IPMI 구성

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

virtualbmc

https://github.com/openstack/virtualbmc

오픈스택 기반

오픈스택 기반으로 구성하는 경우, 다음과 같은 조건이 있다.

1. 물리적 서버 혹은 랩 기반과 다르게 완전한 제어는 불가능(poweroff만 가능)
2. 각 모든 노드에 VirtualBMC 설치 및 구성

IPMI 설치 및 구성

DUSTBOX

2023-05-30

VIRTUALBMC 구성

PIP 명령어로 설치한다.

```
# dnf install python3-pip -y
# pip3 install virtualbmc
```

VIRTUALBMC 구성

각 서버에 vbmcd.service 서버를 설치 및 구성한다.

```
# vi /etc/systemd/system/vbmcd.service
[Unit]
Description=Virtual BMC Daemon
After=network.target
[Service]
ExecStart=/usr/local/bin/vbmcd
Restart=always
[Install]
WantedBy=multi-user.target
# systemctl daemon-reload
# systemctl enable --now vbmcd
```

VIRTUALBMC 구성

자기 자신을 VirtualBMC 서버로 등록한다.

```
# /usr/local/bin/fake-poweroff.sh
#!/bin/bash
echo "Fake poweroff received from vBMC" >> /var/log/vbmc-fake.log
/sbin/shutdown -h now
```

FAKE LIBVIRT 연결 흉내

vBMC는 기본적으로 libvirt URI가 필요.

하지만 instance 내부에는 libvirt가 없으므로, 더미로 **/usr/bin/false**나 **localhost URI**를 넣어도 됩니다.

```
# vbmc add self --address 0.0.0.0 --port 623 \
  --username admin --password pass \
  --libvirt-uri qemu:///session
# vbmc start self
```

FAKE LIBVIRT 연결 흉내

외부에서 IPMI 테스트를 한다.

```
# ipmitool -I lanplus -H <instance_IP> -U admin -P pass chassis power off
# ipmitool -I lanplus -H <instance_IP> -U admin -P pass chassis power on
```

페이스메이커

페이스메이커 소개

구성원 소개

2023-05-30

페이스메이커

페이스 메이커는 실제 **심장 박동기(Pacemaker)**와 비슷한 동작 방식이다. 당연, 그림처럼 서버에 아래와 같은 하드웨어가 아니라, 소프트웨어적으로 구성하여 노드의 생존력(?)을 높인다.

페이스메이커는 다음과 같은 역할을 한다. 서비스 및 **systemd/System V**의 서비스 혹은 유닛 상태를 지속적으로 확인한다. 서비스가 문제가 발생하면, 기존 서비스를 다른 노드가 대신한다.

버전마다 클러스터 지원 크기(노드) 및 기능이 다르기 때문에, 사용 시, 사용하는 배포판에서 어떠한 페이스 메이커 버전을 지원하는지 확인이 필요하다.

왜 아직도 페이스 메이커인가?

- DB / Legacy App / 단일 Writer 서비스
- Kubernetes로 옮기기 어려운 워크로드
- OS 레벨 HA가 필요한 환경

모든 시스템은 미들웨어 솔루션이 필수로 꼭 들어가지 않습니다. 여전히, 많은 시스템은 OS레벨에서 지원이 필요 합니다.

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/Redhat와 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.

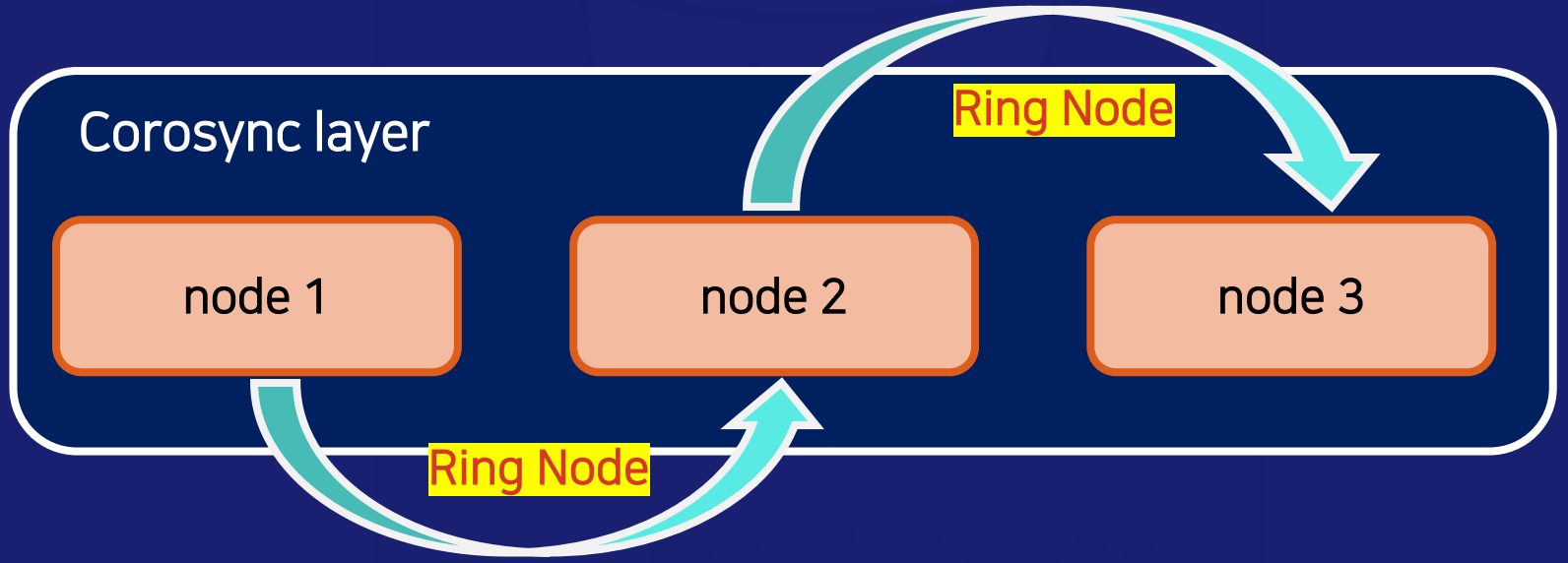
COROSYNC

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

COROSYNC



DUSTBOX

2023-05-30

DRBD

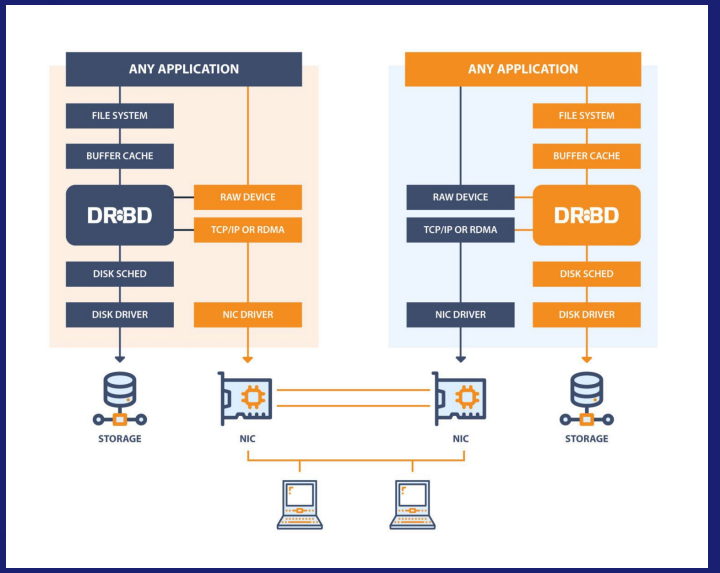
DRBD는 Distribute Replicated Storage System의 약자이다. 커널 드라이버를 통해서 사용자에게 스토리지 미러링을 제공한다.

https://linbit.com/drbd/

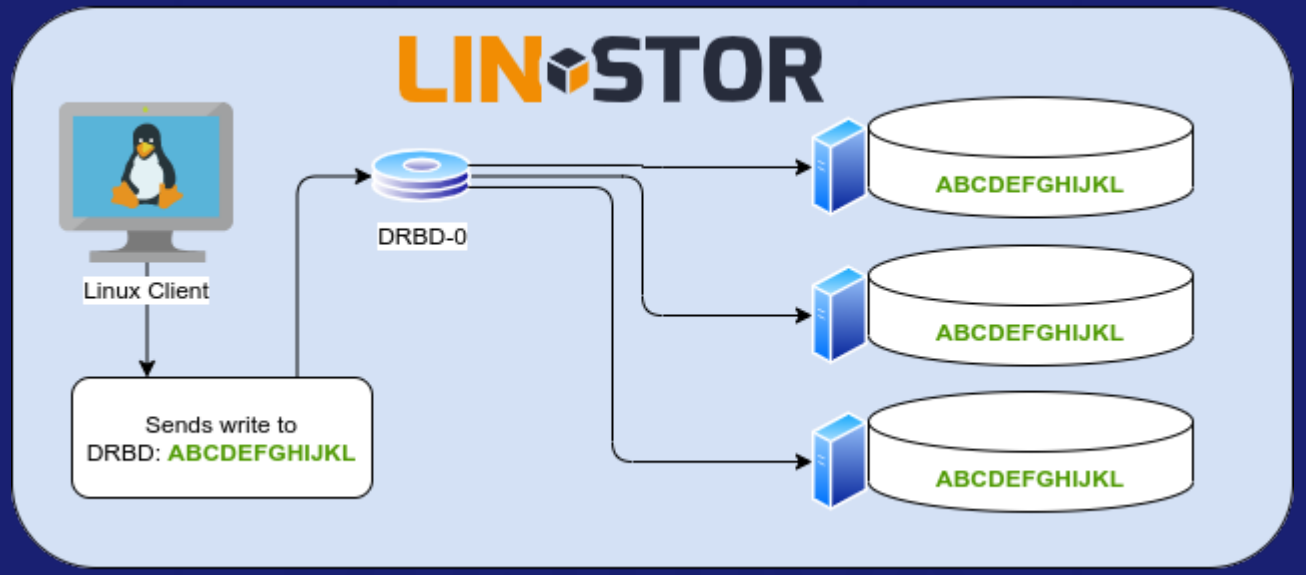
DRBD를 사용하기 위해서는 각각 노드에 DRBD 장치를 구성해야 한다. drbd는 커널 수준에서 장치를 구성 및 배포하기 때문에 리눅스 배포판에서 사용이 가능한지 확인이 필요하다. 이를 사용하기 위해서는 두 가지 형태로 장치를 붙인다.

1. RAW장치
2. LVM2기반의 장치

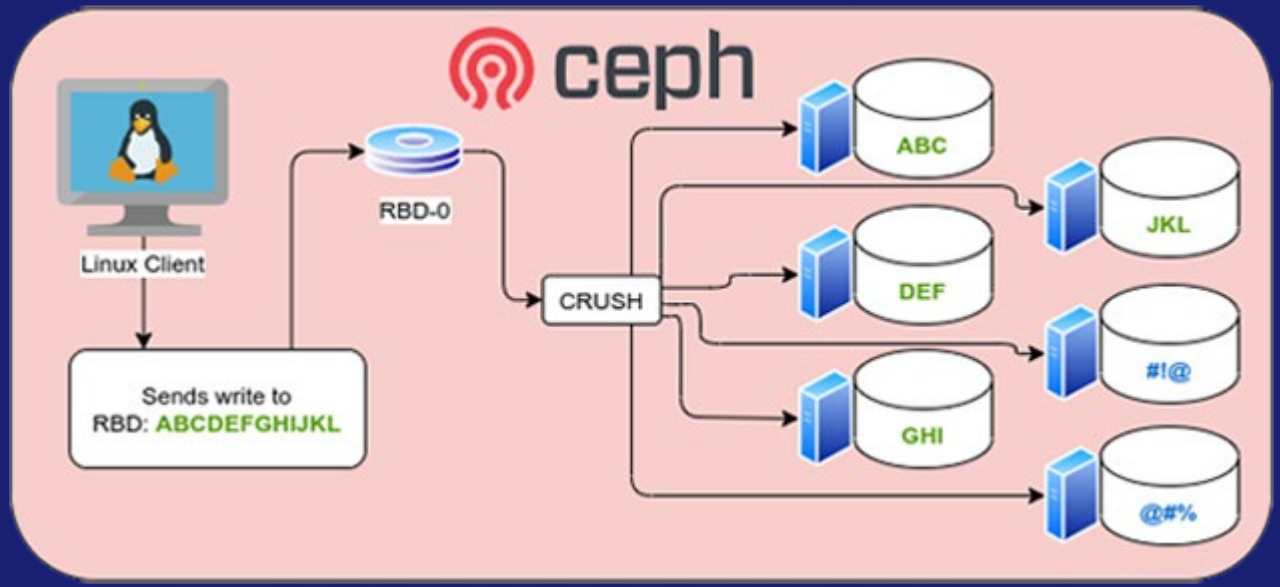
SAN장치가 없는 경우, iSCSI기반으로 구현 및 사용을 권장한다.



DRBD


LINSTOR
Linux Client
DRBD-0
Sends write to
DRBD: ABCDEFGHIJKL
ABCDEFGHIJKL
ABCDEFGHIJKL
ABCDEFGHIJKL

CEPH Storage



Ceph vs DRBD

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 vs DRBD

구분	DRBD	Ceph
기본 개념	두 노드 간 **블록 단위 실시간 미러링** (RAID-1 네트워크 버전)	여러 노드 간 **객체(Object) 기반 분산 스토리지 시스템**
아키텍처 유형	2노드(또는 3노드) Active/Passive 구조 중심	MDS, MON, OSD로 구성된 완전 분산 구조
데이터 저장 방식	블록 장치 단위 복제 (LVM, ext4 등 위에 구성)	Object → Block(RBD), File(CephFS), S3(RGW) 형태로 제공
복제 방식	동기(Sync) 또는 비동기(Async) 복제	CRUSH 알고리즘 기반 데이터 분산 및 복제 (다중 노드)
확장성	수직 확장 중심 (노드 2~3개 제한적)	수평 확장 가능 (수십~수백 노드)
장애 복구	Primary/Secondary 전환 (Pacemaker/Corosync 연동)	자동 데이터 재분배 및 자가 복구(Self-Healing)
구성 난이도	간단 (2대 서버 구성으로 시작 가능)	복잡 (MON, OSD, MGR, RGW 등 다수 데몬 필요)

Ceph vs DRBD

구분	DRBD	Ceph
성능 특성	낮은 지연 시간, 빠른 블록 IO (특히 Sync 모드)	고가용성과 확장성 중심, 대규모 IO 처리에 적합
사용 사례	HA 클러스터용 DB, VM 스토리지 미러링 (예: Pacemaker + DRBD + MariaDB)	대규모 클라우드 스토리지 (예: OpenStack Cinder/Glance, Kubernetes RBD)
운영 방식	Active/Passive 블록 장치로 mount	Ceph Client를 통한 동적 접근 (RADOS, librbd 등)
관리 도구	`drbdadm`, `crm`, `pcs` 등	`ceph`, `rados`, `rbd`, `cephadm`, `dashboard`
네트워크 요구	일반적으로 전용 Sync 링크(1~10Gbps) 필요	클러스터 네트워크(Back-end)와 Public 네트워크 분리
데이터 일관성	강한 일관성 (Sync 복제 시)	최종 일관성 (Object 기반 분산)

Ceph vs DRBD

구분	DRBD	Ceph
대표 구성 예시	DRBD + Pacemaker + Corosync + LVM	Ceph MON + OSD + MDS + MGR + RGW
적합 환경	소규모 HA 환경, 2노드 미러 스토리지	대규모 클라우드, 분산 파일시스템, 오브젝트 스토리지
대표 통합 사례	Pacemaker 클러스터, MySQL HA, VM HA	OpenStack Cinder/Glance/Nova, K8s CSI, RGW S3

ScanCore

스켄코어(ScanCore)는 페이스 메이커의 코어 구성원이다. 이 구성원은 각 노드에서 다음과 같은 상태를 확인한다. 보통 이를 **결정 엔진(Decision Engine)**이라고 부른다. **ScanCore**는 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 및 구성원

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 및 구성원

구성 요소	설명
libQB – 핵심 서비스	클러스터의 기본 기능을 제공하는 핵심 라이브러리로, **로깅(logging)**, **프로세스 간 통신 (IPC, Inter-Process Communication)** 등의 공통 서비스를 담당함.
Corosync	클러스터 노드 간의 **멤버십 관리**, **메시지 전달**, **쿼럼(quorum) 결정** 등을 담당하며, 클러스터의 통신 계층 역할을 수행함.
Resource Agents	데이터베이스, 웹 서버, 가상 IP 등과 같은 **클러스터 자원(Resource)**을 제어하기 위한 **스크립트 모음**으로, 클러스터가 실제 서비스를 시작, 중지, 모니터링할 수 있도록 해줌.
Fencing Agents	장애가 발생한 노드를 격리하기 위해 **전원 스위치나 SAN 장비와 통신**하는 스크립트로, STONITH(Shoot The Other Node In The Head) 기능을 수행함.
Pacemaker 자체	위의 모든 구성요소를 통합하여 클러스터의 **리소스 관리, 상태 감시, 장애 조치(HA)**를 수행하는 핵심 클러스터 관리자.

페이스메이커 주요 기능 정리

Pacemaker는 2004년부터 개발되어 왔으며, 주로 Red Hat과 SUSE가 공동으로 협력하여 발전시킨 프로젝트입니다. 또한 LinBit과 오픈소스 **커뮤니티 전체의 많은 도움과 지원**을 받고 있습니다.

Corosync 역시 2004년에 시작되었지만, 당시에는 OpenAIS **프로젝트의 일부**로 출발했습니다. 현재는 주로 Red Hat **주도하에 개발**되고 있으며, 마찬가지로 **커뮤니티의 폭넓은 지원과 기여**를 받고 있습니다.

이전 및 현 H/A시스템 비교

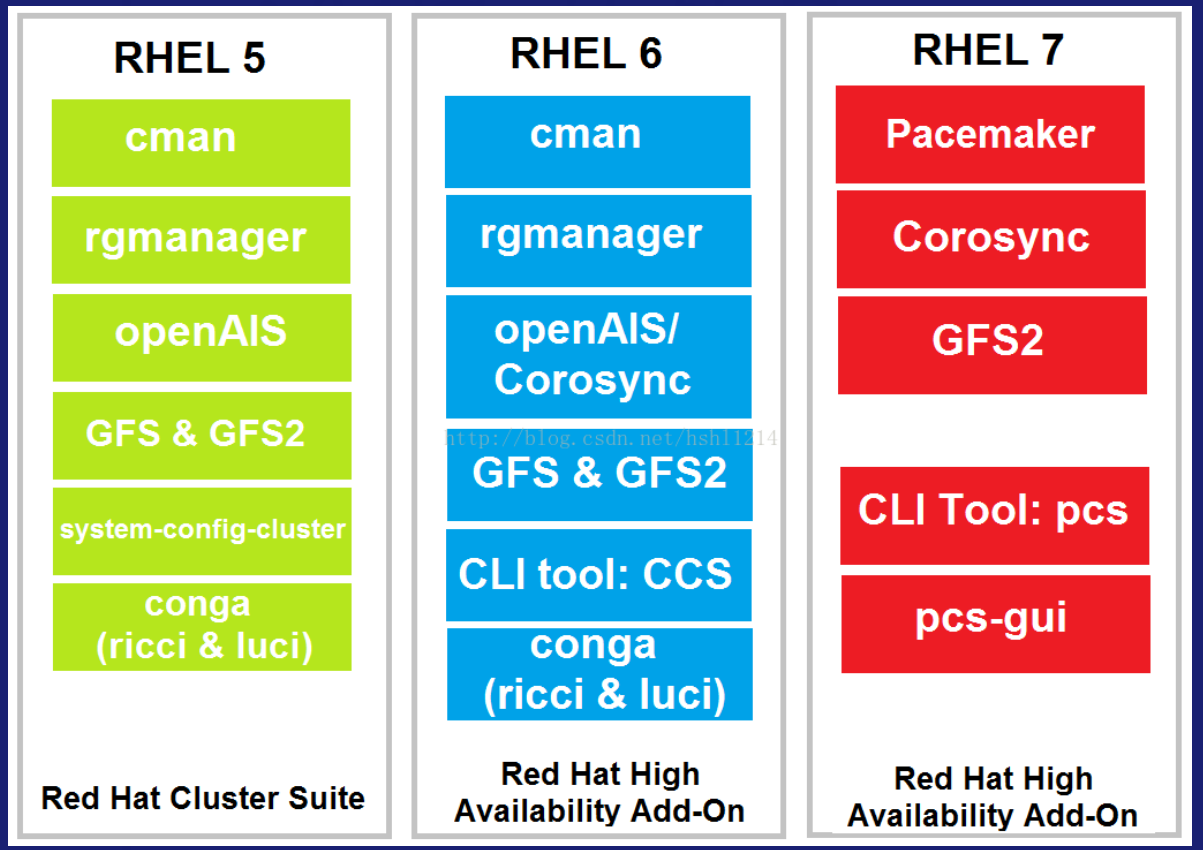
RHEL/CentOS 7 이전 버전에서는

- 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.
- 현재는 RHEL 8, CentOS-8-Stream, Rocky 8이후 버전 사용 권장.

페이스메이커 배포판 버전 별 차이



페이스메이커

	리소스 매니저	페이스메이커
리소스 설정 관리	수동	자동
리소스 관리 모델	자원 그룹	자원(resource) 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

페이스메이커

	리소스 매니저	페이스메이커
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

페이스메이커

	리소스 매니저	페이스메이커
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

에이전트

페이스메이커의 자원은 에이전트 기반으로 구성이 되어있다. 에이전트(resources)는 아래에서 확인이 가능하다.

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.

본 사이트

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

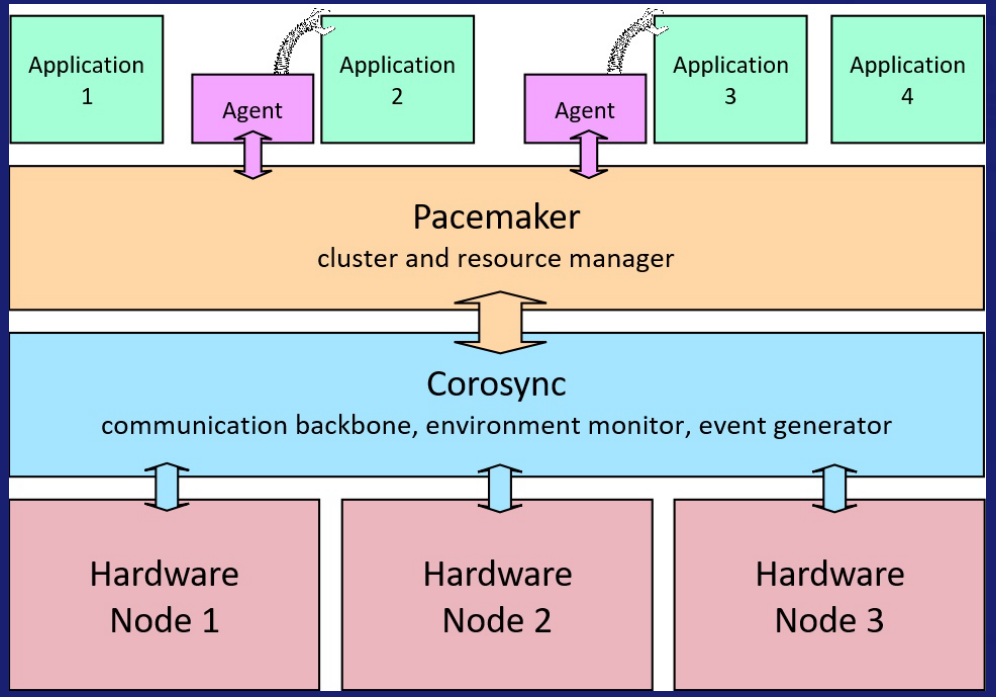
깃헙 사이트

https://github.com/ClusterLabs/resource-agents/blob/main/doc/dev-guides/ra-dev-guide.asc

에이전트

에이전트는 각각 자원별로 리소스 에이전트를 가지고 있다.

에이전트는 설정을 통해서 애플리케이션 및 리소스를 관리한다. 해당 리소스는 페이스메이커가 관리를 하며, 설정 배포 및 환경 상태를 관리는 Corosync를 통해서 한다.



숫자 9, 그리고 복구 목표

페이스메이커는 다음과 목적 달성이 주요 목적이다.

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

숫자 9, 그리고 복구 목표

99.9999% IN 30SEC

99.999% IN 5MIN

99.99% IN 52 MIN

99.9% IN 9 HOUR

99% IN 3.5 DAY

DUSTBOX

2023-05-30

DR VS HA

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

최신 릴리즈에서 H/A이외 D/R기능도 제공하고 있다. D/R vs 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 상대적으로 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.

장점

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.

자원 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.

페이스메이커 기능 최종정리

- 클러스터 노드는 16개까지 권장한다
  - 이 부분에 대해서 나중에 더 자세히 이야기
- 병렬통신을 사용한다. 예를 들어서 UDP, BoradCast, 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

페이스메이커 기능 최종정리

- 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

페이스메이커 기능 최종정리

- 페이스 메이커는 다음과 같은 자원을 지원한다.
- 리소스
- 리소스 에이전트
- DC(DESIGNATED COORDINATOR), 마스터 노드
- 스노니스(STONITH)
- 스플릿 브레인, 구성원이 총 2개(2 Nodes)
- 정족수(Quorum), 구성원이 총 3개 이상을 권장

클러스터 구성 준비

리눅스 설정

클러스터 구성 준비

시작전에 모든 노드를 스냅샷 생성하세요.

리눅스 구성

시작 전, 각각 가상머신에 대해서 이미지 스냅샷을 수행한다.

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

```
baremetal# virsh snapshot-create as --domain node1 --name node1-pcs-setup
baremetal# virsh snapshot-create-as --domain node2 --name node2-pcs-setup
baremetal# virsh snapshot-create-as --domain node3 --name node3-pcs-setup
baremetal# virsh snapshot-create-as --domain node4 --name node4-pcs-setup
baremetal# virsh snapshot-list node1
baremetal# virsh snapshot-revert --domain node1 --snapshotname node1-pcs-
setup --running
```

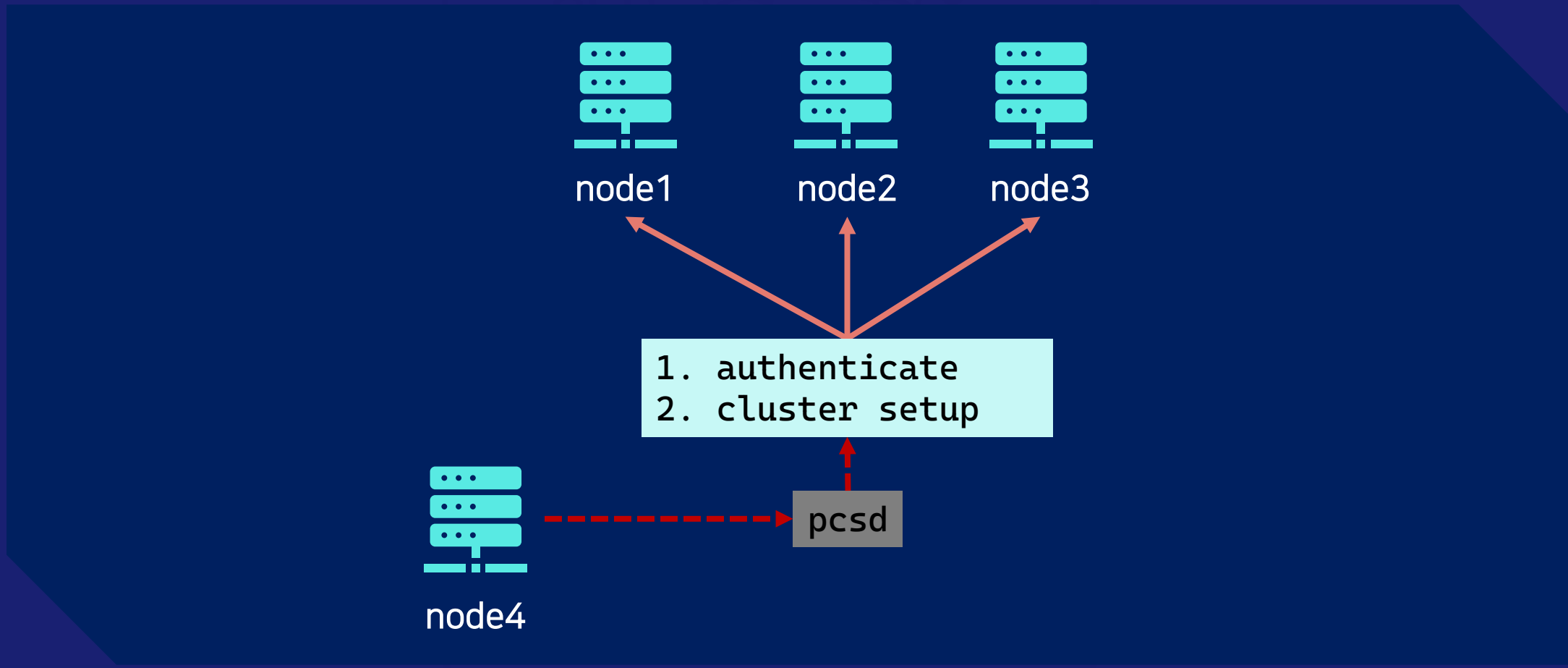
리눅스 구성

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 nodea, nodeb, nodec 혹은 noded(utility)포함.

PACEMAKER


node1
node2
node3
1. authenticate
2. cluster setup
pcsd
node4

리눅스 네트워크 구성

internal네트워크 인터페이스 카드에 다음과 같이 구성한다. 만약, 이미 존재하는 경우, 명령어 **mod**로, 프로파일이 없는 경우 **add**로 프로파일을 추가한다.

```
node1# nmcli con mod 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 mod 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 mod 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 mod 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
```

HOSTNAME/NTP

각 서버에 호스트 이름 설정 및 그리고 NTP서버 설정. 현재는 외부망으로 연결하지만, 내부망(폐쇠망)으로 운영하시는 경우, 꼭 내부에 최소 한대의 NTP서버가 구성이 되어 있어야 됨.

```
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

nodeX# timedatectl set-ntp true
nodeX# vi /etc/chrony.conf
```

NTP(CHRONY)

```
nodeX# grep -Ev '^#|^$' /etc/chrony.conf
pool 2.centos.pool.ntp.org iburst --> server ntp.internal.example.com
sourcedir /run/chrony-dhcp
driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
```

systemd-timesyncd

추후에는 systemd-timesyncd.service로 변경될 예정. 설정 및 변경 방법은 아래 링크 참조.

https://www.freedesktop.org/software/systemd/man/systemd-timesyncd.service.html

```
vi /etc/systemd/timesyncd.conf
vi /etc/systemd/timesyncd.conf.d/local.conf
[Time]
NTP=0.arch.pool.ntp.org 1.arch.pool.ntp.org 2.arch.pool.ntp.org
3.arch.pool.ntp.org
FallbackNTP=0.pool.ntp.org 1.pool.ntp.org 0.fr.pool.ntp.org
```

PACEMAKER 노드

DNS서버가 없기 때문에, "A Recode"를 "/etc/hosts"파일 통해서 구성한다. 모든 서버에 아래의 내용을 등록한다. D/R까지 구현을 원하는 경우 기존 클러스터에 x2, 4대이면 8대, 8대이면 16대로 구성해야 한다.

```
node1# vi /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

192.168.90.150 node5.example.com node5
192.168.90.160 node6.example.com node6
```

자원이 매우 부족하면 node2혹은 3번까지

16기가 이상이면 node4번까지

자원이 매우 넉넉하면 node6까지

2023-05-30

PACEMAKER SSH 키 배포

각각 서버에 SSH키를 생성 후 배포한다.

```
node1# ssh-keygen -t rsa -N '' -f ~/.ssh/id_rsa
node1# dnf install sshpass -y
node1# cat <<EOF> ~/.ssh/config
StrictHostKeyChecking=no
EOF
node1# for i in {1..4} ; do sshpass -p centos ssh-copy-id root@node${i} ;
done
```

node6까지 있으면, node6번이 관리 노드

PACEMAKER

키를 생성한 다음에, 다음과 같은 명령어로 키를 배포한다.

```
node1# for i in node{1..4} ; do ssh root@${i} 'dnf update -y' ; done
node1# for i in node{1..4} ; do scp /etc/hosts
root@${i}.example.com:/etc/hosts ; done
node1# for i in node{1..4} ; do ssh root@${i} 'dnf --
enablerepo=highavailability -y install pacemaker pcs' ; done
```

node6까지 있으면, node6번까지 루프에 추가!

PACEMAKER

```
node1# for i in {1..4} ; do ssh root@node${i} 'firewall-cmd --add-
service=high-availability && firewall-cmd --runtime-to-permanent' ; done

node1# for i in {1..4} ; do ssh root@node$i 'echo centos | passwd --stdin
hacluster && systemctl enable --now pcsd.service' ; done
```

PACEMAKER

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

토큰 인증
/var/lib/pcsd/

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

클러스터 구성(CIB)
/var/lib/pacemaker/

node1/3번만 있으면 1/3번 까지만!

PACEMAKER 명령어 정리

```
node1# pcs cluster start --all
node1# pcs cluster enable --all
node1# pcs cluster status

node1# pcs status corosync
node1# pcs cluster stop --all
node1# pcs cluster destroy --all
node1# ss -npltu | grep -i corosync
```

노드 추가하기

리눅스에서 노드 및 클러스터를 추가하고 싶은 경우, 아래 명령어를 수행한다.

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

host# virt-install --memory 4096 --vcpu 2 -n node4 \
--disk /var/lib/libvirtd/images/node4.qcow2,cache=none,bus=virtio \
-w network=default,model=virtio -w network=internal,model=virtio \
--graphics none --autostart --noautoconsole --import
```

노드 추가하기(리눅스)

```
node1# pcs cluster auth -u hacluster -p centos node4
node1# pcs cluster node add node4 --start --enable


node1 # pcs cluster start node4
node1 # pcs cluster enable node4
```

노드 추가하기 정리

```
node1# systemctl start --enable pcsd.service
node1# echo centos | passwd --stdin hacluster
node1# pcs host auth -u hacluster -p centos node4.example.com
node1# pcs cluster node add node3.example.com --enable --start
node1# pcs cluster status
node1# pcs status corosync
node1# corosync-cfgtools -s
```

노드 제거하기

```
node1# pcs cluster stop node4.example.com
node1# pcs cluster node delete node4.example.com
> delete, remove 차이 없음
node1# pcs cluster status
```

연습문제

각각 가상머신을 다시 롤백 후, 아래와 같이 작업을 수행한다.

1. node1/2/3번을 cluster-lab이라는 이름으로 클러스터를 생성한다.
   - 다른 이름으로 설정해도 무난
2. 모든 노드들은 internal 네트워크를 통해서 연결 및 구성이 된다.
   - pacemaker, storage 네트워크 분리
3. hacluster사용자의 암호는 centos로 변경한다.
4. 각 노드들은 공개 혹은 비공개키로 접근이 가능해야 한다.

연습문제

기존에 구성하였던 클러스터에 node4번을 추가한다.

1. 기존에 사용하였던 클러스터에 node4번을 추가
2. 추가를 하기 위한 네트워크를 구성한다.
3. 올바르게 구성이 되면 node4에서 클러스터 노드가 조회가 가능해야 한다.
   - pcs cluster status
4. node3번을 기존에 구성하였던 클러스터에서 제거한다.
   - 제거가 올바르게 되었는지 pcs cluster status로 확인
   - 다시, node3번을 클러스터에 추가한다.

노드 3대로 진행하시는 분들은 2번째 노드를 제거 및 다시 추가 해보세요.

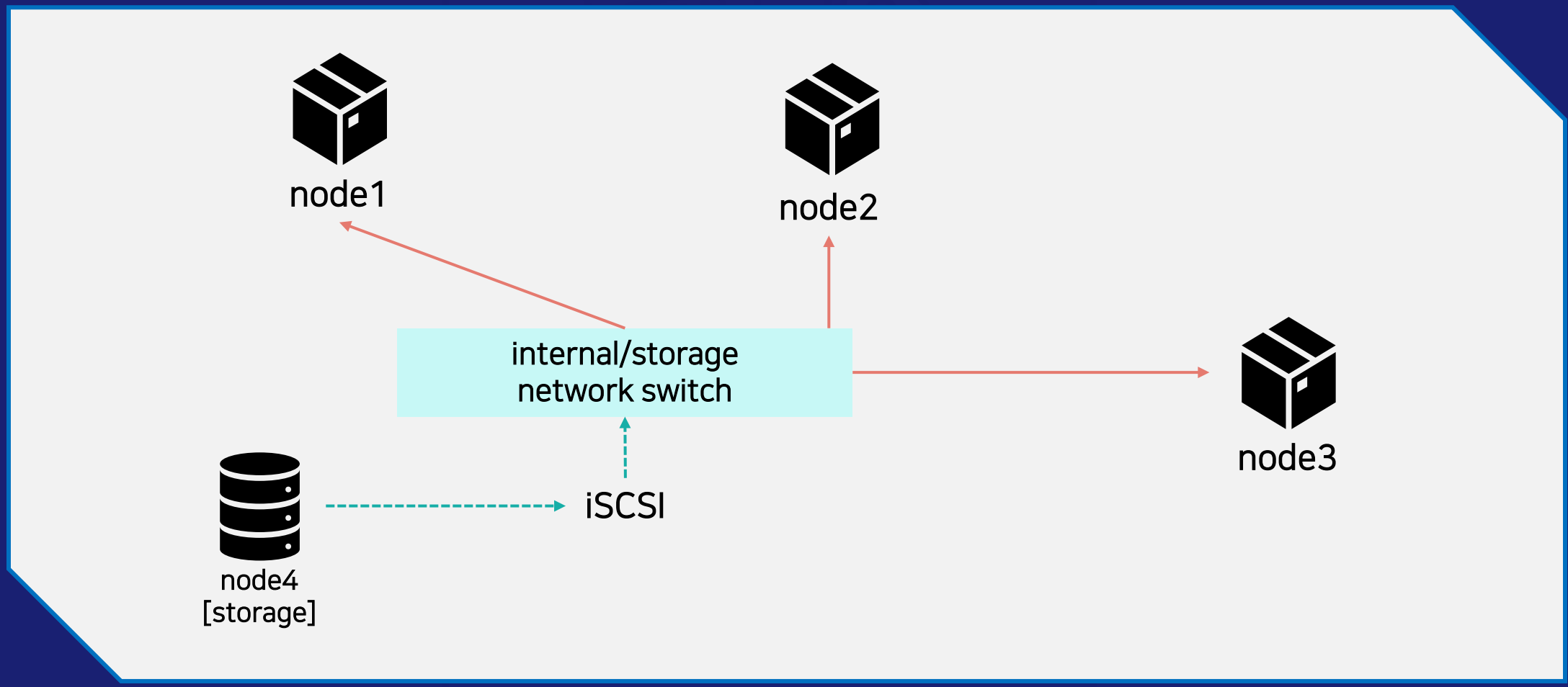
ISCSI 노드 구성

스토리지 서버 구성 및 설정

ISCSI

시작전에 모든 노드를 스냅샷 생성하세요.

ISCSI


node1
node2
internal/storage
network switch
node3
iSCSI
node4
[storage]

ISCSI

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

노드에 생성되는 디스크는 iSCSI서버인 targetd에서 파일 기반으로 구성해서 각 노드에 전달. 블록장치로 구성을 원하는 경우, 블록 장치로 구성하여도 상관 없음. 이 과정에서는 multipath를 구성하지 않기 위해서 최소한으로 SAN를 구성함.

이를 위해서 우리는 아래와 같은 소프트웨어를 사용한다.

- targetd(target) iSCSI(SAN Protocol) 에뮬레이터
- targetd-cli
- iscsi, iscsid

ISCSI 서버 구성

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 sdb /var/lib/iscsi_disks/sdb.img
2G

node4# targetcli backstores/fileio create sdc /var/lib/iscsi_disks/sdc.img
2G

node4# targetcli backstores/fileio create sdd /var/lib/iscsi_disks/sdd.img
2G

node4# targetcli backstores/fileio create sde /var/lib/iscsi_disks/sde.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/sdb/

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

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

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

ISCSI SERVER ACL

접근을 허용하도록 IQN ACL를 구성한다. 각각 노드의 "**호스트**.init" 형식으로 허용한다.

```
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
```

ISCSID 오류

이와 같은 메시지가 출력이 되면, 아래서 서비스를 중지한다.

```
nodeX# systemctl stop iscsid.service
nodeX# systemctl stop iscsid.socket
```

```
iscsid[37328]: iscsid: Kernel reported iSCSI connection 1:0 error (1020 -
ISCSI_ERR_TCP_CONN_CLOSE: TCP connection closed) state (3)
kernel:  connection1:0: detected conn error (1020)
```

ISCSI

```
node4# targetcli saveconfig

node1/2/3/4# for i in {1..3} ; do ssh root@node${i} "dnf install iscsi-initiator-utils -y" ; done

node1# cat <<EOF> /etc/iscsi/initiatorname.iscsi

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

EOF
```

ISCSI

```
node2# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2023-02.com.example:node2.init

node3# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2023-02.com.example:node3.init

node4# vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2023-02.com.example:node4.init
```

ISCSI

CHAP를 사용하는 경우, 아래 부분을 **/etc/iscsi/iscsid**에서 수정 필요.

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

ISCSI

노드 1번부터 4번까지 iscsi장치 추가. SAN으로 장치를 전달을 받음. 스토리지 스위치에 접근 시, 사용하는 데몬은 두 가지가 있음.

iscsi: 이전에 구성된 노드 정보가 있으면, 해당 정보를 읽어와서 iscsi 연결 구성

>/var/lib/iscsi/nodes

iscsid: iscsi관련된 설정파일을 불러와서 SAN 스위치 혹은 서버와 통신

>/etc/iscsi/

```
node1# for i in {1..4}; do ssh root@node${i} "systemctl restart iscsi
iscsid && iscsiadm -m discovery -t sendtargets -p 192.168.90.140 &&
iscsiadm -m node --login"; done
```

ISCSI SUB COMMAND

```
node1/2/3/4# systemctl restart iscsi iscsid
node1/2/3/4# iscsiadm -m discovery -t sendtargets -p 192.168.90.140
node1/2/3/4# iscsiadm -m node --login
node1/2/3/4# iscsiadm -m session --debug 3
node1/2/3/4# 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

HA클러스터 구성 및 확인

pacemaker

ISCSI

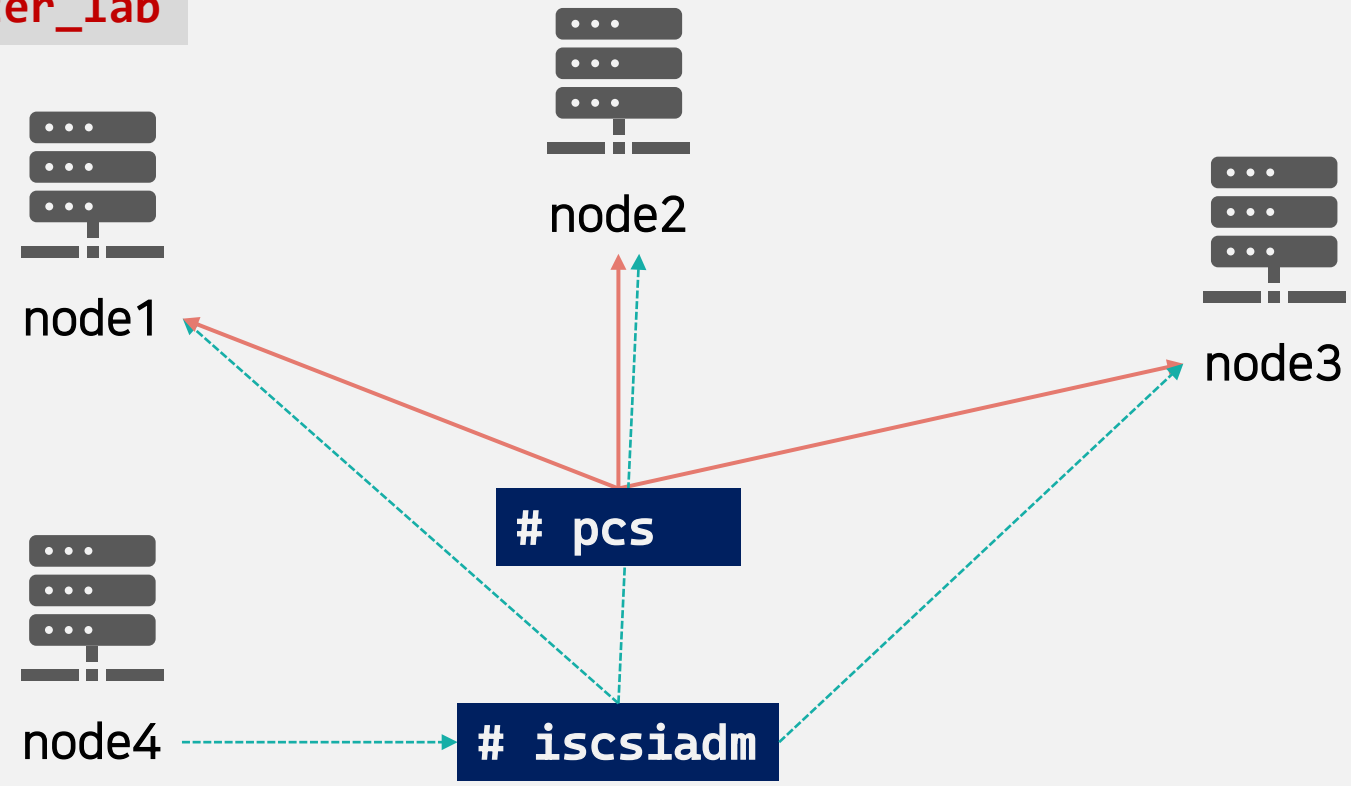
시작전에 모든 노드에 스냅샷 생성하세요.

PACEMAKER CLUSTER

- 클러스터 생성
- 클러스터 확인

클러스터 구성

클러스터 이름: `ha_cluster_lab`



DUSTBOX

2023-05-30

:o<-<

방화벽 설정 및 설치

```
node1/2/3# dnf --enablerepo=high-availability -y install pacemaker pcs
node1/2/3# firewall-cmd --add-service=high-availability --permanent
node1/2/3# firewall-cmd --reload
```

클러스터 재구성 및 상태 확인

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

node1# pcs cluster setup ha_cluster_lab node1.example.com node2.example.com
node3.example.com --enable --start

node1# pcs cluster start --all

node1# pcs cluster enable --all

node1# pcs cluster status

node1# pcs status corosync

node1# corosync-cfgtools -s
```

간단한 펜싱 장치 구성

```
node1# for i in {1..4} ; do ssh root@node${i} "dnf install --
enablerepo=highavailability fence-agents-all watchdog -y" ; done

node1# for i in {1..4} ; do scp /usr/share/cluster/fence_scsi_check
root@node${i}:/etc/watchdog.d/ && systemctl enable --now watchdog ; done

node1# ls -l /dev/disk/by-id

node1# pcs stonith create scsi-shooter fence_scsi pcmk_host_list="node1.exa
mple.com node2.example.com node3.example.com node4.example.com" devices=/de
v/disk/by-id/wwn-<ID> meta provides=unfencing

node1# pcs stonith config scsi-shooter

node1# pcs status
```

2023-05-30

펜싱 후 복구 그리고 제거

```
# 상태 정보 및 펜싱 적용
node4# pcs status
node4# pcs stonith fence node2.example.com
node4# pcs cluster status
# 복구하는 방법
node4# pcs cluster start node2.example.com
node4# reboot
# 펜싱장치 제거는 아래 명령어로 제거가 가능하다 .
node4# pcs stonith delete scsi-shooter
```

노드 상태 확인하기

```
host1# corosync-cfgtool -s
Local node ID 4, transport knet
LINK ID 0 udp
        addr = 192.168.90.140
        status:
                nodeid:          1:     localhost
                nodeid:          2:     connected
                nodeid:          3:     connected
```

노드 상태 확인하기

```
node4# pcs cluster sync
node1.example.com: Succeeded
node2.example.com: Succeeded
node3.example.com: Succeeded
Warning: Corosync configuration has been synchronized, please reload
corosync daemon using 'pcs cluster reload corosync' command.
```

노드 상태 확인하기

```
node1# 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

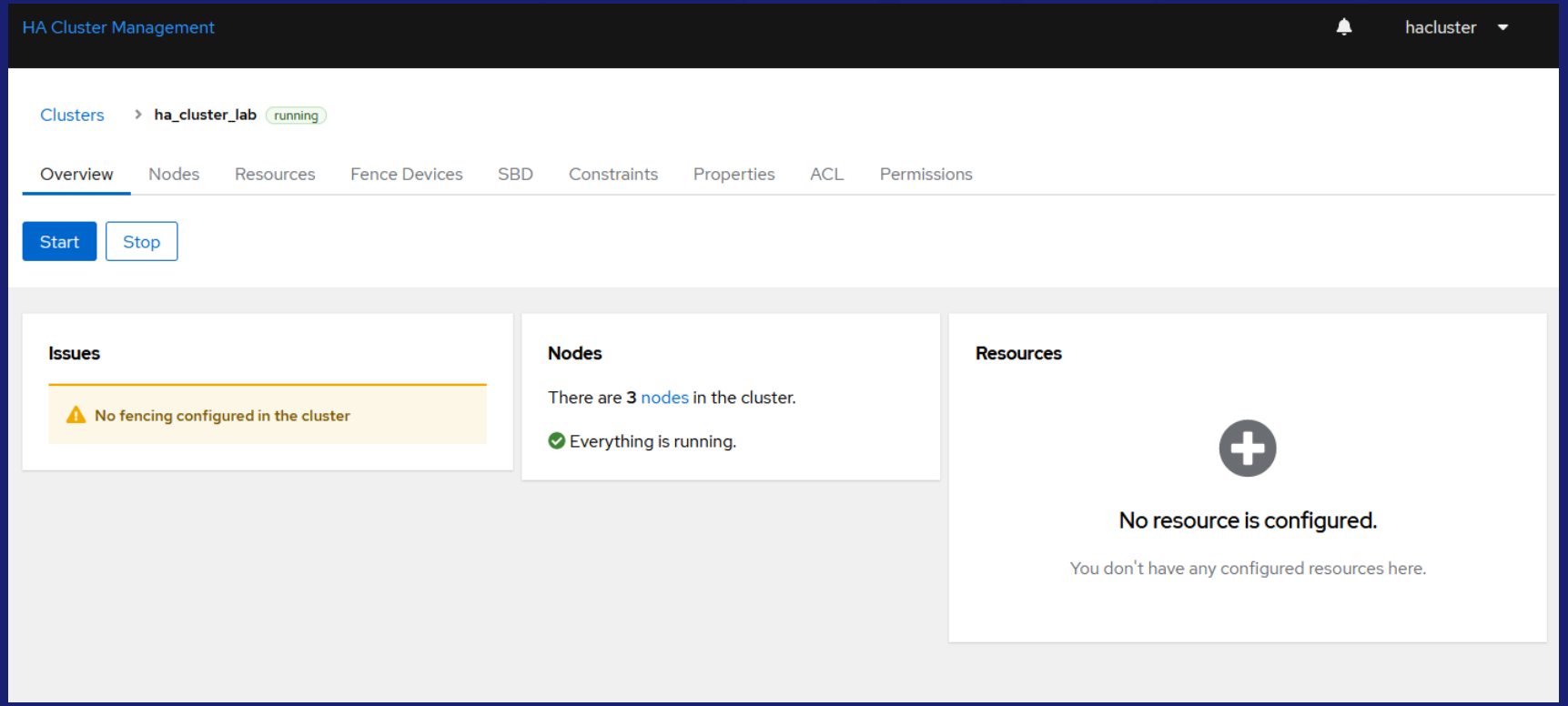
```
nodeX# journalctl -b | grep -i error
nodeX# journalctl -b -u <UNIT_NAME> -p err -p warning
nodeX# journalctl -u pcsd.service -perr -fl
nodeX# journalctl -pwarning -perr _COMM=<PROCESS_NAME>
```

2023-05-30

GUI 접근

페이스메이커가 올바르게 구성이 되면, 아래 주소 및 포트로 웹 기반 관리자 페이지 접근이 가능하다.

`https://<NODE1_EXTERNAL_IP>:2224/`



연습문제

모든 노드를 초기화 후, 다음처럼 클러스터를 다음처럼 구성한다. iSCSI서버 구성이 어려운 경우, 기존 클러스터를 destroy후 아래 내용으로 재구성한다.

1. 클러스터 이름은 pcs-lab이라고 생성한다
2. iscsi서버를 target서버 기반으로 구성한다.
3. 2기가 파일 크기로, 블록장치를 생성한다.
   - file-block.raw
   - nfs-block.raw
   - gfs2-block.raw
4. node1/3/4를 pcs-lab클러스터에 추가한다.
5. 추가가 완료가 되면, 각각 블록 장치를 iscsi를 통해서 올바르게 구성한다.
6. /dev/sdb디스크에 대한 펜싱 장치를 생성한다.
   - 이름은 iscsi-fecne-device라고 설정한다.

DRBD 구성

배포 기반 블록 스토리지

DRBD

시작전에 모든 노드를 스냅샷 생성하세요.

DRBD

DRBD는 정확히는 페이스메이커의 솔루션은 아니다. LINBIT회사에서 제작한 솔루션이며, 코어버전은 GPL 2.0으로 공개가 되어 있다.

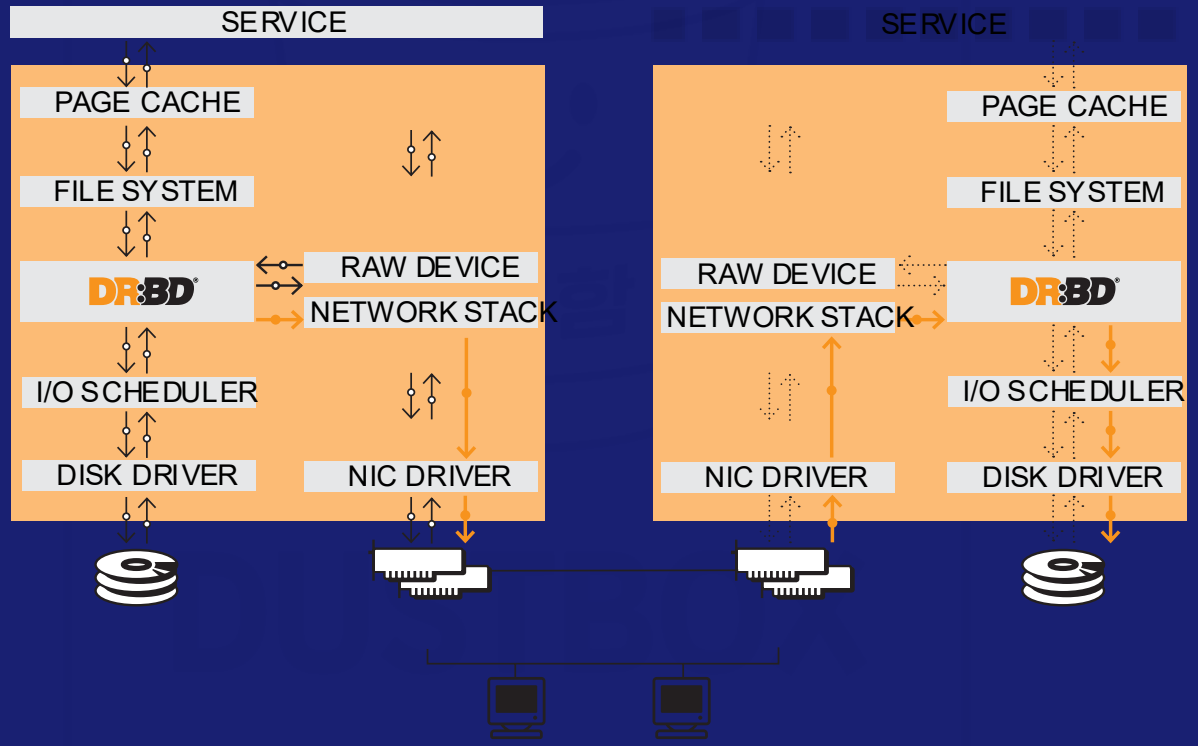
DRBD 소스코드

DRBD는 RAID 1기능을 TCP기반으로 구현한다. GlusterFS와 기능이 비슷하지만, 파일 기반이 아닌 실제 물리적 블록 장치 기반으로 구현한다. 대다수 배포판에서 별도의 컴파일 없이 사용이 가능하도록 패키징이 되어 있기 때문에 바이너리 기반으로 설치 후, 설정하여 바로 사용이 가능하다.

DRBD는 GFS2파일 시스템과 다르게 BIT by BIT로 블록을 복제한다. 멀티 락킹 혹은 DLM지원하지 않는 파일 시스템도 안정적으로 사용이 가능하다.

보통 ACTIVE/ACTIVE 혹은 ACTIVE/PASSIVE 형태로 많이 사용한다.

DRBD


SERVICE
PAGE CACHE
FILE SYSTEM
DRBD
RAW DEVICE
NETWORK STACK
I/O SCHEDULER
DISK DRIVER
NIC DRIVER
SERVICE
PAGE CACHE
FILE SYSTEM
RAW DEVICE
NETWORK STACK
DRBD
I/O SCHEDULER
NIC DRIVER
DISK DRIVER

DRBD

DRBD is, by definition and as mandated by the Linux kernel architecture, agnostic of the layers above it. Therefore, it is impossible for DRBD to miraculously add features to upper layers that these do not possess. For example, DRBD cannot auto-detect file system corruption or add active-active clustering capability to file systems like ext3 or XFS.

DRBD는 정의상, 그리고 리눅스 커널 아키텍처의 설계 원칙에 따라, 상위 계층과는 독립적으로 동작합니다.
다시 말해, DRBD는 그 위에 올라가는 파일 시스템(ext3, ext4, XFS, Btrfs 등)에 새로운 기능을 부여할 수 없습니다.

예를 들어, DRBD는 파일 시스템의 무결성 손상이나 데이터 불일치를 스스로 감지하지 못하며, 또한 **ext3, ext4, XFS, Btrfs** 같은 파일 시스템에 **active-active(양방향 동시 접근)** 기능을 추가하는 것도 불가능합니다.
이러한 기능은 파일 시스템 자체가 클러스터 환경을 지원하도록 설계되어 있어야 합니다. (예: OCFS2, GFS2 등)

DRBD

drbdadm

DRBD도구에서 지원하는 도구이다. 이 도구를 통해서 모든 DRBD의 파라메터 설정이 가능하다. 이 명령어는 "/etc/drbd.conf"의 내용을 편집해주며, drbdadm는 drbdsetup, drbdmeta의 프론트 앤드 명령어 이기도 하다. "-d" 옵션을 통해서 "dry-run"상태로 실행이 가능하기 때문에 시스템 영향이 없이 구성 및 테스트가 가능하다.

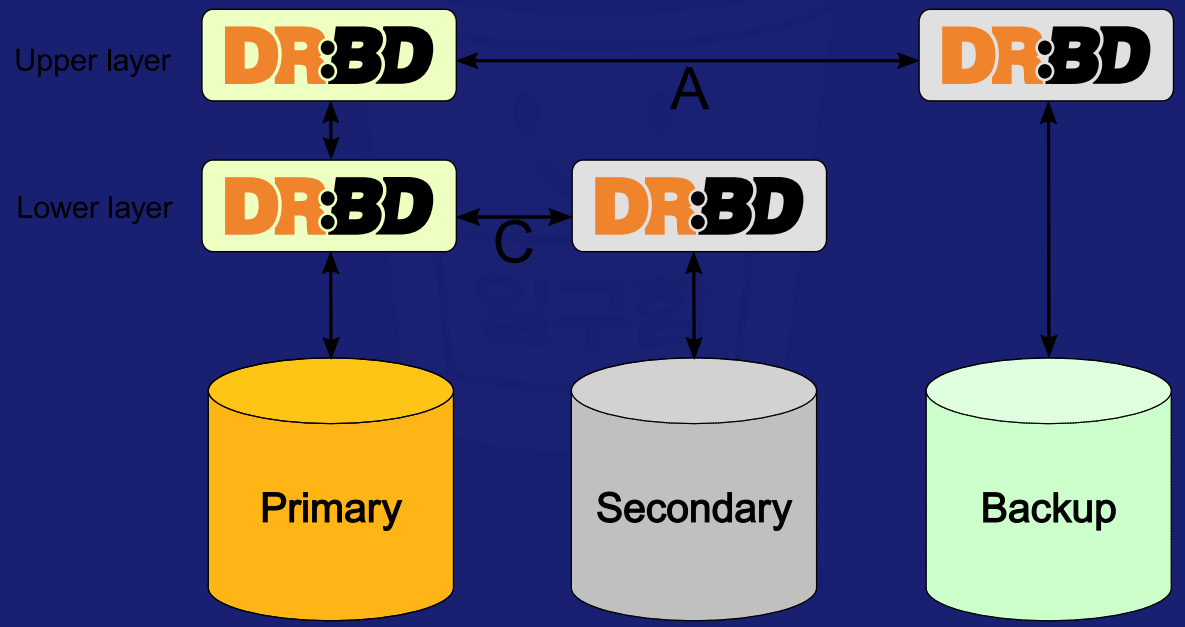
drbdsetup

DRBD 모듈에 대해서 설정한다. 이 설정은 drbd.ko모듈이 커널이 불러올 때 사용한다. 모든 변수들은 drbdsetup명령어를 통해서 구성이 된다. drbdadm, drbdsetup의 차이점은 drbdsetup은 커널 관련된 변수를 설정하기 때문에, 일반 사용자는 이 명령어를 자주 사용하지 않는다.

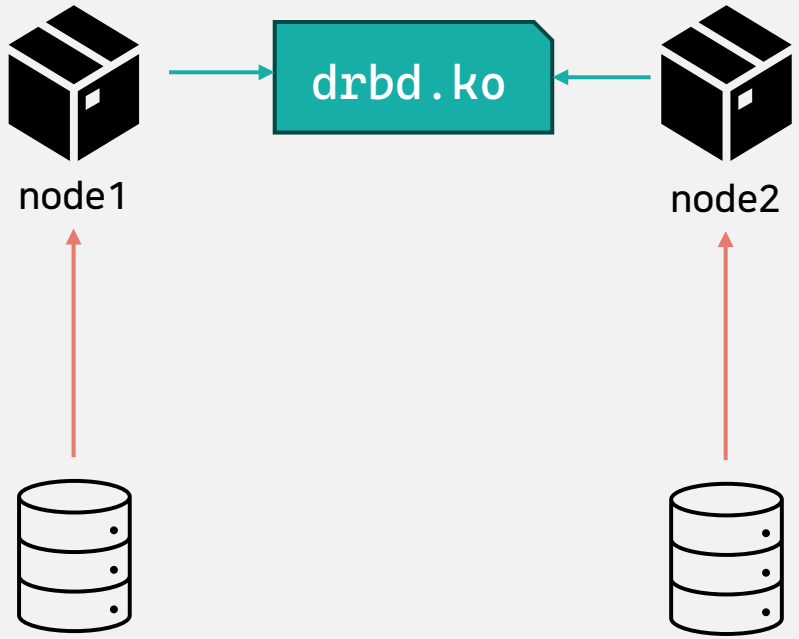
drbdmeta

DRBD의 메타정보를 덤프/복구/생성을 위해서 사용한다. drbdsetup과 비슷하지만, 이는 메타 정보를 다룰 때만 사용하기 때문에 일반 사용자가 사용하는 경우는 역시 드물다.

DRBD


Upper layer
DRBD
DRBD
A
Lower layer
DRBD
DRBD
C
Primary
Secondary
Backup

PACEMAKER



Pacemaker Cluster

DRBD 호환성

Rocky 8/9, RHEL 8/9에서는 손쉽게 설치가 가능하지만, CentOS-8/9-Stream에서는 커널 호환성 문제로 drbd모듈이 올바르게 동작이 되지 않는다.

DRBD를 올바르게 사용하기 위해서는 가급적이면 RHEL9/Alma 9/Rocky 9에서 설치 및 운영을 권장한다. CentOS에서 설치 및 운영을 하기 위해서는 커널 버전을 RHEL 9과 동일하게 구성한다.

1. DRBD는 각 리소스당 최대 32개 노드 접근 가능
2. DRBD Volumes은 노드당 1048576개 연결 가능
3. DRBD에서 최대 장치 크기는 1PiB(1024TiB)
4. 리눅스 커널은 최소 3.10부터 지원

DRBD SETUP(without pacemaker)

Node1, Node2에 DRBD를 구성하여 xfs파일 시스템 기반으로 복제 및 DR를 구현한다. DRBD는 다중 읽기/쓰기 기능은 지원하지 않는다. 동작 테스트 하기 위해서는 두 개의 노드를 이동하면서 사용한다.

```
node1/2# pvcreate /dev/sdb
node1/2# vgcreate drbd-demo /dev/sdb
node1/2# lvcreate --name drbd-demo -l 100%Free drbd-demo
node1/2# firewall-cmd --add-port=6996-7800/tcp --permanent
node1/2# firewall-cmd --reload
```

DRBD SETUP(without pacemaker)

```
node1# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source address="192.168.90.110" port port="7789" protocol="tcp" accept'

node2# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source address="192.168.90.120" port port="7789" protocol="tcp" accept'

node1/2# firewall-cmd --reload

node1/2# firewall-cmd --list-all
```

DRBD SETUP(without pacemaker)

```
node1/2# dnf install epel-release -y
```

kernel-core-5.14.0-284.11.1.el9_2

```
node1/2# dnf install https://www.elrepo.org/elrepo-release-9.el9.elrepo.noarch.rpm

node1/2# dnf install drbd drbd-bash-completion drbd-pacemaker drbd-utils kmod-drbd9x -y

node1/2# reboot

node1/2# depmod -a && modprobe drbd && lsmod | grep drbd

node1/2# systemctl enable --now drbd

node1/2# systemctl status drbd
```

주의: uEFI가 활성화 되어 있으면, 모듈이 올바르게 메모리에 상주가 안됩니다. 끄고 하세요!

```
# modprobe drbd
modprobe: ERROR: could not insert 'drbd': Key was rejected by service
```

DRBD SETUP

```
node1/2# vi /etc/drbd.d/resource0.res
resource resource0 {
  on node1.example.com {
    device    /dev/drbd1;
    disk      /dev/drbd-demo/drbd-demo;
    address   192.168.90.110:7789;
    meta-disk internal;
  }
```

DRBD SETUP

```
  on node2.example.com {
    device    /dev/drbd1;
    disk      /dev/drbd-demo/drbd-demo;
    address  192.168.90.120:7789;
    meta-disk internal;
  }
}
```

DRBD SETUP

```
node1# drbdadm create-md resource0
node1# drbdadm up resource0
node1# drbdadm status resource0
node1# drbdadm primary --force resource0
node1# drbdadm status resource0
node1# lsblk
node1# mkfs.xfs /dev/drbd1 or mkfs.ext4 /dev/drbd1
node1# mkdir -p /mnt/drbd
```

DRBD SETUP

```
node1# mount /dev/drbd1 /mnt/drbd
node1# systemctl daemon-reload
node1# cd /mnt/drbd
node1# touch test{1..100}
node1# umount /mnt/drbd
node1# drbdadm secondary resource0
```

DRBD SETUP

```
node2# drbdadm up resource0
node2# drbdadm status resource0
node2# drbdadm primary resource0
node2# mkdir -p /mnt/drbd
node2# systemctl daemon-reload
node2# mount /dev/drbd1 /mnt/drbd
node2# ls -l /mnt/drbd
```

DRBD PACEMAKER

```
node1# pcs resource create drbd-vip ocf:heartbeat:IPaddr2
    ip=192.168.90.250 cidr_netmask=24 op monitor interval=10s
node1# pcs resource create drbd-fs ocf:heartbeat:Filesystem
    device=/dev/drbd0 directory=/mnt/drbd fstype=xfs
    op start timeout=60s on-fail=restart
    op stop timeout=60s on-fail=block
    op monitor interval=10s timeout=60s on-fail=restart
node1# pcs resource group add grp-fs drbd-fs drbd-vip
```

DRBD PACEMAKER

DRBD 구성은 뒤에서 한 번 더 다룹니다.

연습문제

node1/2를 롤백 후, 다시 DRBD를 구현하여, 블록장치 복제가 올바르게 이루어지는지 확인한다.

- 디스크는 /dev/sdb디스크를 사용한다.
- 사용 후, 모든 내용은 초기화 혹은 모듈을 제거한다.

자원 설명

페이스메이커 자원

보안

ACL

ISCSI

시작전에 모든 노드에 스냅샷 생성하세요.

ACL

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

클러스터 구성 후, 일반 사용자를 생성하여, 페이스메이커 모니터링을 위해서 CIB접근 할 수 있도록 한다. 또한, 보안 이유로 hacluster를 계정을 직접적으로 사용하지 못하도록 하기도 한다.

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

ACL 설정

```
node1# adduser -s /usr/sbin/bash rouser
node1# echo centos | passwd --stdin rouser
node1# usermod -aG haclient rouser
node1# pcs acl enable
node1# pcs acl role create read-only description="Read only access to
cluster" read xpath /cib
node1# pcs acl user create rouser read-only
node1# pcs acl
node1# pcs client local-auth
```

ACL 테스트

```
node1# su - rouser
node1# pcs status
node1# pcs cluster status
node1# pcs acl
node1# pcs property set maintenance-mode=true
```

ACL 연습문제

사용자 cluster-monitor를 생성하세요.

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

알람

ALERT

ISCSI

시작전에 모든 노드에 스냅샷 생성하세요.

ALERT

페이스 메이커에서는 에이전트 스크립트를 지원한다. 이 스크립트는 보통 30초에 한번씩 동작한다. 모니터링 스크립트는 **/usr/share/pacemaker/alerts/**에 예제가 있다.

이를 복사해서 사용자가 원하는 위치에 복사한다. 일반적으로 alert스크립트는 **/var/lib/pacemaker/**이나 혹은 **/usr/local/sbin**과 같은 디렉터리에 복사한다.

ALERT(FILE)

```
node1/2/3# install --mode=0755 \
/usr/share/pacemaker/alerts/alert_file.sh.sample \
/var/lib/pacemaker/alert_file.sh
node1# touch /var/log/pcmk_alert_file.log
node1# chown hacluster:haclient /var/log/pcmk_alert_file.log
node1# chmod 600 /var/log/pcmk_alert_file.log
```

ALERT(FILE)

```
node1# pcs alert create id=alert_file description="Log events to a file."
path=/var/lib/pacemaker/alert_file.sh

node1# pcs alert recipient add alert_file id=my-alert_logfile
value=/var/log/pcmk_alert_file.log

node1# pcs alert
```

ALERT(EMAIL)

```
node1/2/3# install --mode=0755 \
/usr/share/pacemaker/alerts/alert_smtp.sh.sample \
/var/lib/pacemaker/alert_smtp.sh
node1# pcs alert create id=smtp_alert path=/var/lib/pacemaker/alert_smtp.sh
options email_sender=root@localhost
node1# pcs alert recipient add smtp_alert value=root@localhost
node1# pcs alert
```

연습문제

node1/2/3번에 로그 파일을 생성하도록 스크립트를 복사한다. node1/2/3에 추가적으로 "alert_file" 리소스도 추가 구성한다.

1. 예제 파일 그대로 사용하며, 파일의 위치는 "/usr/share/pacemaker/alerts".
2. 스크립트는 이메일을 root@localhost에 발송한다.
3. 설치는 'install'명령어로 "/var/lib/pacemaker/alert_smtp.sh"에 복사한다.
4. 소유권은 적절히 올바르게 변경한다.
5. alert를 적절한 이름으로 등록한다.
6. 에이전트는 기본값을 사용하여 메일을 root사용자에게 전달한다.
7. 앞서 사용하였던 "alert_file"구성은 기존 node4와 동일하게 구성한다.

클라이언트

CLIENT

CLUSTER COMMAND

CLIENT

시작전에 모든 노드에 스냅샷 생성하세요.

CLIENT

페이스 메이커에서 클라이언트를 구성하기 위해서는 클러스터 내부에 관리 용도를 위한 노드가 필요하다. 이 랩에서는 node4번이 클라이언트 역할을 하고 있다. 페이스메이커 클러스터에서, 특정 사용자가 제한된 권한으로 로그인하기 위해서 'pcs client local-auth'통해서 인증이 가능함.

```
node4# pcs client local-auth -u rouser -p centos
```

CLUSTER COMMAND

한 개 이상의 노드가 구성이 되어 있는 **멤버(member)**를 클러스터라고 부른다. 페이스메이커에서는 여러 노드 관리가 가능하며, 관리를 위한 명령어는 다음과 같다.

- 특정 노드를 점검상태로 전환한다.

```
node1# pcs node maintenance
```

- 특정 노드를 대기 상태로 변경한다.

```
node1# pcs node standby
```

- 특정 노드를 대기 상태에서 제외한다.

```
node1# pcs node unstandby
```

CLUSTER COMMAND

- 특정 노드를 점검상태에서 제외한다.

```
node1# pcs node unmaintenance
```

- 특정 노드에 CPU, Memory에 대한 사용량을 명시한다.

```
node1# pcs node utilization
```

- 클러스터 상태를 확인

```
node1# pcs cluster status
```

- 클러스터의 노드 정보를 같이 출력한다.

```
node1# pcs cluster config
```

CLUSTER COMMAND

- 클러스터에서 노드 인증.

```
node1# pcs cluster auth
```

- 클러스터 단일 혹은 모든 노드의 부트-업 활성화

```
node1# pcs cluster enable
```

- 클러스터에서 단일 혹은 모든 노드의 시작

```
node1# pcs cluster start
```

CONFIG/COMMAND

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

- 현재 사용중인 pcsd설정 내용을 콘솔에 출력한다.

```
node1# pcs config
```

- 지금까지 생성된 체크포인트 설정 파일을 화면에 출력한다.

```
node1# pcs config checkpoint
```

- 현재 사용중인 설정을 파일로 백업한다.

```
node1# pcs config backup
```

- 특정 시점 혹은 파일로 복원한다.

```
node1# pcs config restore
```

부스

BOOTH

BOOTH

시작전에 모든 노드에 스냅샷 생성하세요.

BOOTH

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

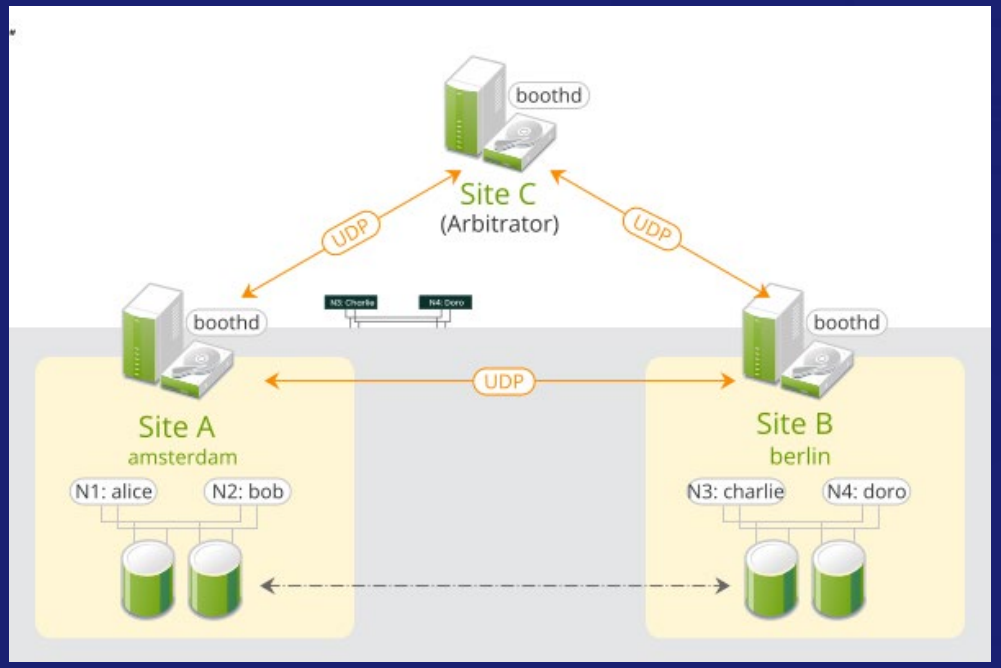
이를 구성하기 위해서는 자원이 많이 필요하기 때문에, 이 교육에서는 다루지 않는다. 부스를 구성하기 위해서 다음과 같은 조건으로 클러스터를 생성 및 관리한다.

1. `site-a-cluster`
2. `site-b-cluster`
3. `arbitrator-node`

BOOTH 동작방식

Booth는 왼쪽과 같이 구성한다. 각각 부스는 중재자 클러스터를 통해서 장애가 발생 시, Site A에서 Site B로 자원을 전달한다. Booth를 구성하기 위해서는 최소 3개 이상의 클러스터가 필요하다.

https://documentation.suse.com/sle-ha/12-SP4/single-html/SLE-HA-geo-guide/index.html



BOOTH 설치준비

설치를 위해서는 다음과 같은 단계를 진행한다. 가급적이면 클러스터에 3대의 노드를 포함, 최소 2대의 노드 포함하여 구성.

```
# Arbitrator
arbitrator# dnf --enablerepo=highavailability install -y booth-core booth-arbitrator

# 각 사이트 모든 노드
siteA-node{1..3}# dnf --enablerepo=highavailability install -y booth-site pcs
siteB-node{1..3}# dnf --enablerepo=highavailability install -y booth-site pcs
```

BOOTH 설치준비

```
# 모든 클러스터 노드 + Arbitrator 공통
# pcsd(2224/tcp), booth(9929/tcp,udp), HA 서비스 허용
firewall-cmd --add-service=high-availability
firewall-cmd --add-port=2224/tcp
firewall-cmd --add-port=9929/tcp
firewall-cmd --add-port=9929/udp
firewall-cmd --runtime-to-permanent
```

2개 클러스터 사이트 생성

총 2개의 클러스터 사이트를 생성.

```
# Site-A
siteA-node1# pcs host auth -u hacluster -p centos node1.example.com
node2.example.com node3.example.com
siteA-node1# pcs cluster setup site-a-cluster node1.example.com
node2.example.com node3.example.com --start --enable
siteA-node1# pcs property set stonith-enabled=false   # 랩 환경 한정 권장

# Site-B
siteB-node1# pcs host auth -u hacluster -p centos node4.example.com
node5.example.com node6.example.com
siteB-node1# pcs cluster setup site-b-cluster node4.example.com
node5.example.com node6.example.com --start --enable
siteB-node1# pcs property set stonith-enabled=false   # 랩 환경 한정 권장
```

BOOTH 사이트 생성

```
# 한 사이트(보통 Site-A의 node1)에서 booth 구성 생성

siteA-node1# pcs booth setup sites 10.0.10.100 10.0.20.100 arbitrators
203.0.113.10

siteA-node1# pcs booth ticket add service-ticket expire=120 acquire-
after=60 timeout=5

siteA-node1# pcs booth sync

# 필요 시

siteA-node1# pcs booth enable

siteA-node1# pcs booth start
```

티켓 발행

```
# 예: Site-A에 우선 가동
siteA-node1# pcs booth ticket grant service-ticket
```

서비스 구성

모든 노드에 다음과 같이 패키지를 설치한다.

```
# Apache 설치 + 기본 페이지
dnf install -y httpd
echo "Welcome to Dustbox HA Web on $(hostname)" > /var/www/html/index.html
systemctl disable --now httpd

# 방화벽에 HTTP 열기
firewall-cmd --add-service=http
firewall-cmd --runtime-to-permanent
```

Site-A(예: node1)에서 리소스 생성 구성

다음과 같이 Site A 노드에서 자원을 생성 및 구성한다.

```
# VIP + Web을 리소스로 만들고 그룹으로 묶기
pcs resource create web-vip ocf:heartbeat:IPaddr2 \
  ip=192.168.10.250 cidr_netmask=24 \
  op monitor interval=30s
# 필요 시 NIC 지정: nic=ens192 같은 식으로 옵션 추가
pcs resource create web-service systemd:httpd \
  op monitor interval=30s
pcs resource group add web-group web-vip web-service
# 티켓 보유 사이트에서만 올라가도록 제약 추가
pcs constraint location web-group rule ticket service-ticket
```

Site-B(예: node4)에서도 "똑같이" 생성

두 클러스터는 독립적이라, 반드시 똑같은 리소스를 생성해주어야 한다.

```
pcs resource create web-vip ocf:heartbeat:IPaddr2 \
  ip=192.168.10.250 cidr_netmask=24 \
  op monitor interval=30s
# 필요 시 nic=ens192

pcs resource create web-service systemd:httpd \
  op monitor interval=30s

pcs resource group add web-group web-vip web-service

pcs constraint location web-group rule ticket service-ticket
```

Booth 데몬 확인 & 티켓 초기 부여(한쪽만)

두 클러스터는 독립적이라, 반드시 똑같은 리소스를 생성해주어야 한다.

```
# 양쪽에서 booth 기동 확인
pcs booth enable
pcs booth start
pcs booth ticket list

# 최초 부여는 한쪽만 (예: Site-A)
pcs booth ticket grant service-ticket
```

간단 DR 테스트

DR이 동작하는지 확인한다.

```
# 각 사이트에서 확인
pcs status
pcs booth ticket list

# VIP 피어스/웹 확인
ip addr | grep 192.168.10.250   # 활성 사이트에서만 보여야 정상
curl -s http://192.168.10.250   # "Welcome to Dustbox…" 페이지 확인
```

BOOTH 랩

2노드 기반으로 추가 클러스터 구성 후, Booth구성을 한다. 구성조건은 다음과 같다. 단, 컴퓨팅 자원이 부족하면 이번 랩은 진행하지 않는다.

1. 기존 클러스터(TWO NODEs) + TWO NODEs CLUSTER + TWO NODEs CLUSTER
2. 중계 클러스터는 반드시 구성이 되어야 한다.

DR

장애처리

2023-05-30

DR

시작전에 모든 노드에 스냅샷 생성하세요.

DR

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

이 기능은 레드햇 리눅스 기준으로 RHEL 8부터 사용이 가능하다.

DR 구성

랩 구성은 대략 다음과 같다.

1. Primary 클러스터: site-a-cluster (노드 예: node1~3.example.com)
2. Recovery 클러스터: site-b-cluster (노드 예: node4~6.example.com)
3. VIP: 192.168.10.250/24 (두 사이트에서 라우팅/L2 접근 가능하다고 가정)
4. 웹 서비스: Apache(httpd), 기본 포트 80
5. 리소스/그룹 이름: web-vip, web-service, 그룹 web-group

DR/BOOTH

BOOTH와 비슷하지만, BOOTH는 티켓 기반으로 미리 구성된 시스템에 전환하는 방식이다. 이 방식은 장점은 모든 전환이 자동화 기반으로 구현이 되기 때문에, 사용자 개입이 필요가 없다.

하지만, DR기능은 완전한 자동화가 아닌 반 자동화에 가깝기 때문에 사용자 개입이 필요하다.

구분	Booth(Geo-cluster 티켓 매니저)	Pacemaker DR(Booth 없이 구성)
목적	멀티-사이트 간 **단일 활성 사이트**를 티켓으로 자동/합의 기반 결정	두 클러스터를 DR 쌍으로 두고 **수동/반자동으로 전환**
핵심 구성	각 사이트의 Pacemaker + **boothd** + (권장) **arbitrator**	각 사이트의 Pacemaker만(별도 booth 불필요)
의사결정	**티켓(ticket)** 합의(다수결/가용성)로 활성 사이트 판정	관리자가 `pcs`로 **수동으로 자원 페일오버** 실행
분할(brain-split) 대응	Arbitrator로 **네트워크 단절 vs 사이트 다운**을 판별	수동 전환이라 운영 절차/규율에 **의존도 높음**

DR/BOOTH

위의 내용 계속 이어서...

구분	Booth(Geo-cluster 티켓 매니저)	Pacemaker DR(Booth 없이 구성)
리소스 제어	리소스에 **ticket 기반** location constraint 적용	리소스에 **일반** location/rule constraint 적용
자동화 수준	높음(티켓 기반 자동 사이트 선택)	낮음~중간(운영 스크립트/절차에 따름)
필요 포트	booth(기본 9929/UDP,TCP), pcsd(2224/TCP) 등	pcsd(2224/TCP) 등 기본 HA 포트
Arbitrator 필요성	**권장/사실상 필수**(특히 2-사이트)	**불필요**(수동 티켓/전환이면 필요 없음)
한계	**데이터 복제는 별도**(DRBD/스토리지/DB 레플리케이션 필요)	동일(데이터 일관성은 별도 계층에서 해결)

클러스터 기본 생성

두 개의 클러스터를 생성한다. 기존 랩과 다르게 구성할 예정이기 때문에 인스턴스 이름을 z시리즈로 변경한다.

```
# 공통 인증: 한 노드에서 두 사이트 모든 노드 인증
pcs host auth -u hacluster -p '<PW>' z1.example.com z2.example.com
z3.example.com z4.example.com

# 1차(Primary) 클러스터 생성
pcs cluster setup PrimarySite z1.example.com z2.example.com --start

# 2차(Recovery) 클러스터 생성
pcs cluster setup DRSite z3.example.com z4.example.com --start
```

클러스터 기본 생성

특정 노도를 복구용(Recovery) 노드로 선언 및 구성한다.

```
# Primary 클러스터의 노드에서 실행: Recovery 사이트 지정
# pcs dr set-recovery-site z3.example.com
# pcs dr config
Local=Primary, Remote=Recovery
```

DR 상태 모니터링

올바르게 동작하는지 DR 상태를 확인한다.

```
# 한 사이트에서 양쪽 클러스터 상태를 한 번에 보기
# pcs dr status
# 상세/정리 옵션
# pcs dr status --full
# pcs dr status --hide-inactive
```

공통 준비(모든 노드 공통)

올바르게 동작하는지 DR 상태를 확인한다.

```
# 패키지
dnf -y install pcs httpd

# 기본 페이지
echo "Welcome to Dustbox DR Web on $(hostname)" > /var/www/html/index.html
systemctl disable --now httpd

# 방화벽 (Pacemaker/pcsd + HTTP)
firewall-cmd --add-service=high-availability
firewall-cmd --add-service=http
firewall-cmd --add-port=2224/tcp
firewall-cmd --runtime-to-permanent
```

각 클러스터 생성

Site A를 생성한다.

```
# node1에서
pcs host auth -u hacluster -p 'centos' node1.example.com node2.example.com
node3.example.com
pcs cluster setup site-a-cluster node1.example.com node2.example.com
node3.example.com --start --enable
pcs property set stonith-enabled=false   # 랩 전용
```

각 클러스터 생성

Site B를 생성한다.

```
# node4에서
pcs host auth -u hacluster -p 'centos' node4.example.com node5.example.com
node6.example.com
pcs cluster setup site-b-cluster node4.example.com node5.example.com
node6.example.com --start --enable
pcs property set stonith-enabled=false   # 랩 전용
```

pcs dr 연동 (부스 없이 DR 연결)

Site-A의 한 노드에서 실행 (예: node1). set-recovery-site에 Site-B의 대표 노드 하나만 지정해야 DR 구성이 됨.

```
pcs dr set-recovery-site node4.example.com
pcs dr config
pcs dr status
```

Site-A (node1 등에서 1회)

Booth와 동일하게 각각 사이트(클러스터)에 자원을 생성해야 한다.

```
# pcs resource create web-vip ocf:heartbeat:IPaddr2 \
  ip=192.168.10.250 cidr_netmask=24 op monitor interval=30s

# pcs resource create web-service systemd:httpd \
  op monitor interval=30s

# pcs resource group add web-group web-vip web-service
```

Site-B (node4 등에서 1회)

Booth와 동일하게 각각 사이트(클러스터)에 자원을 생성해야 한다.

```
# pcs resource create web-vip ocf:heartbeat:IPaddr2 \
  ip=192.168.10.250 cidr_netmask=24 op monitor interval=30s

# pcs resource create web-service systemd:httpd \
  op monitor interval=30s

# pcs resource group add web-group web-vip web-service
```

평시(Primary에서만 서비스 가동)

Site-A에서 시작 & 활성화.

```
# pcs resource enable web-group
# pcs resource start  web-group
# pcs status
```

Site-B는 대기(꺼둠)

Site-A에서 시작 & 활성화.

```
# pcs resource disable web-group
# pcs status
```

노드 상태 확인

다음 명령어로 확인한다.

```
# ip addr | grep 192.168.10.250          # Site-A에서만 VIP 보여야 정상
# curl -s http://192.168.10.250          # 페이지에 Site-A 호스트명 보이면 OK
```

DR 수동 전환 시나리오

Primary → 정지

```
# Site-A에서 서비스 내림
pcs resource stop web-group
pcs resource disable web-group
```

DR 수동 전환 시나리오

Primary → 정지

```
# Site-A에서 서비스 내림
pcs resource stop web-group
pcs resource disable web-group
```

DR 수동 전환 시나리오

Recovery → 기동

```
# Site-B에서 서비스 올림
pcs resource enable web-group
pcs resource start  web-group
pcs status
```

검증

올바르게 복구가 되었는지 확인한다.

```
ip addr | grep 192.168.10.250     # 이제 Site-B에서 VIP 보임
curl -s http://192.168.10.250     # 페이지에 Site-B 호스트명 보이면 성공
pcs dr status                     # 두 클러스터 상태 요약 확인
```

검증

다시 상태를 복구 시킨다.

```
# Site-B에서 내림
pcs resource stop web-group
pcs resource disable web-group

# Site-A에서 다시 올림
pcs resource enable web-group
pcs resource start  web-group
```

DR COMMAND

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

node1# pcs cluster setup none-dr-nodes node1.example.com node2.example.com --start --enable

node1# pcs cluster setup dr-nodes node3.example.com node4.example.com --start --enable

node1# pcs dr set-recovery-site node3.example.com

node1# pcs dr config

node1# pcs dr status

DR 랩

간단하게 페이스메이커 기반으로 DR를 구성한다. 이를 통해서 클러스터 백업을 구성 및 생성한다. 단, 자원이 부족한 경우, 이번 랩 진행은 하지 않아도 된다.

1. node1, node2번을 primary 사이트로 구성, 클러스터 이름은 first-cluster-nodes으로 설정한다.
2. node3, node4번은 remote 사이트로 구성, 클러스터 이름은 recovery-cluster-nodes으로 설정한다.
3. 구성이 완료가 되면, node1, node2번 노드를 종료 후 recovery-cluster-nodes에 올바르게 클러스터 서비스를 전달받아서 역할(role)이 변경 되었는지 확인한다.

DR 랩 예제

```
node1# pcs dr config
Local site:
  Role: Recovery
Remote site:
  Role: Primary
  Nodes:
    node1.example.com
    node2.example.com
```

RESOURCES

간단한 명령어

자원 개념 설명

RESOURCES

시작전에 모든 노드에 스냅샷 생성하세요.

CONSTRAINT(제한)

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

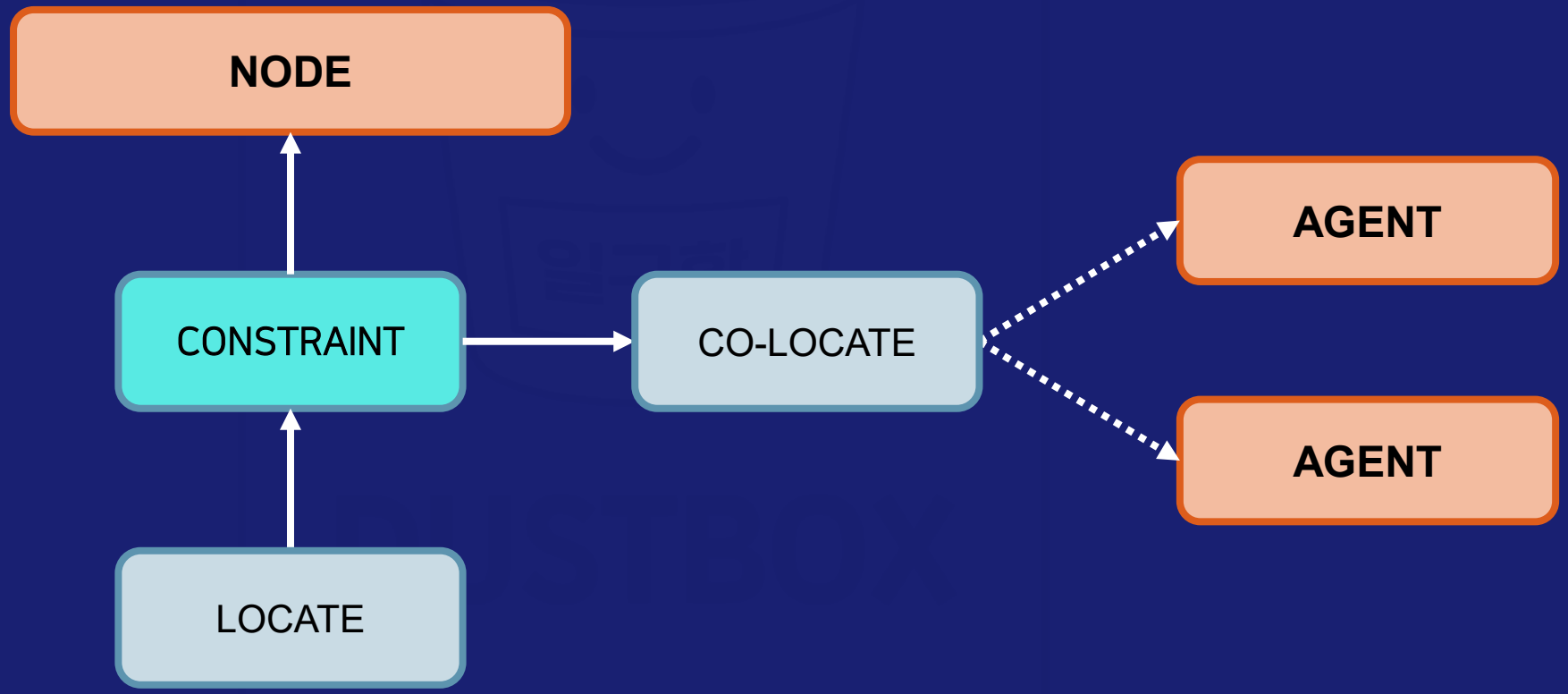
로케이션(location)

- 단일 클러스터의 어떠한 노드에서 자원 구성 및 실행 위치 결정.

순서(order)

- 여러 자원이 있을 때, 어떠한 순서로 자원 실행 결정.

CONSTRAINT(제한)



DUSTBOX

2023-05-30

CONSTRAINT/LOCATE/COLOCATE

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

```
node1# pcs resource create demo-constraint ocf:pacemaker:Dummy
node1# pcs constraint
node1# pcs constraint location demo-constraint prefers node1=100
node1# pcs constraint config --full
#
# 아이디 번호 확인 하려면, "--full"옵션 사용
#
node4# pcs constraint delete <ID_NAME>
```

INFINITY/SCORE

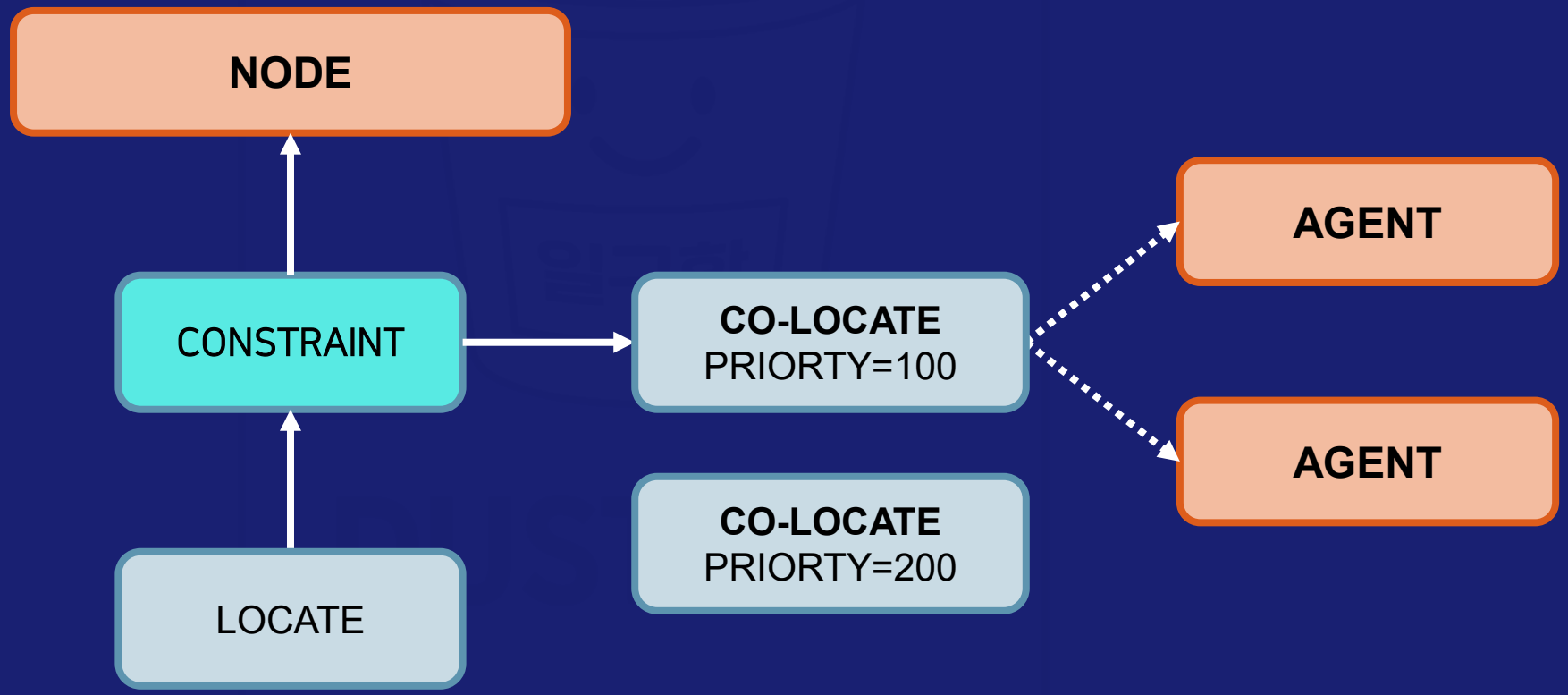
점수(score)

자원을 생성 시, 점수를 통해서 자원의 선호도 구성이 가능하다. 노드에 점수가 충분하지 않는 경우, 리소스 구성이 올바르게 되지 않을 수 있다. 점수는 **자원(resource)**와 **노드(node)**를 통해서 계산이 된다. 음수를 가지고 있는 경우에는 해당 노드는 자원을 구성을 할 수 없다.

무한(INFINITY)

무한은 점수와 비슷하다. **Infinity**는 Score와 비교하면, 1,000,000과 같다. 그래서, Infinity로 명시가 되어 있으면, 이는 점수로 1,000,000과 같다.

INFINITY/SCORE



DUSTBOX

2023-05-30

INFINITY/SCORE

-INFINITY < 음수 < 0 < 양수 < INFINITY

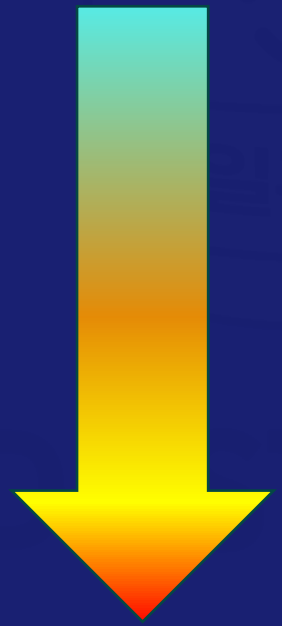
- Any value + INFINITY = **INFINITY**
- Any value - INFINITY = **-INFINITY**
- INFINITY - INFINITY = **-INFINITY**

https://clusterlabs.org/pacemaker/doc/deprecated/en-US/Pacemaker/1.1/html/Pacemaker_Explained/ch06.html

SCORE+LOCATION 예제

우선순위(priority) 및 로케이션(location)을 사용하여 자원들의 우선순위, 노드 선호 그리고 노드의 선호도 조합이 가능하다.

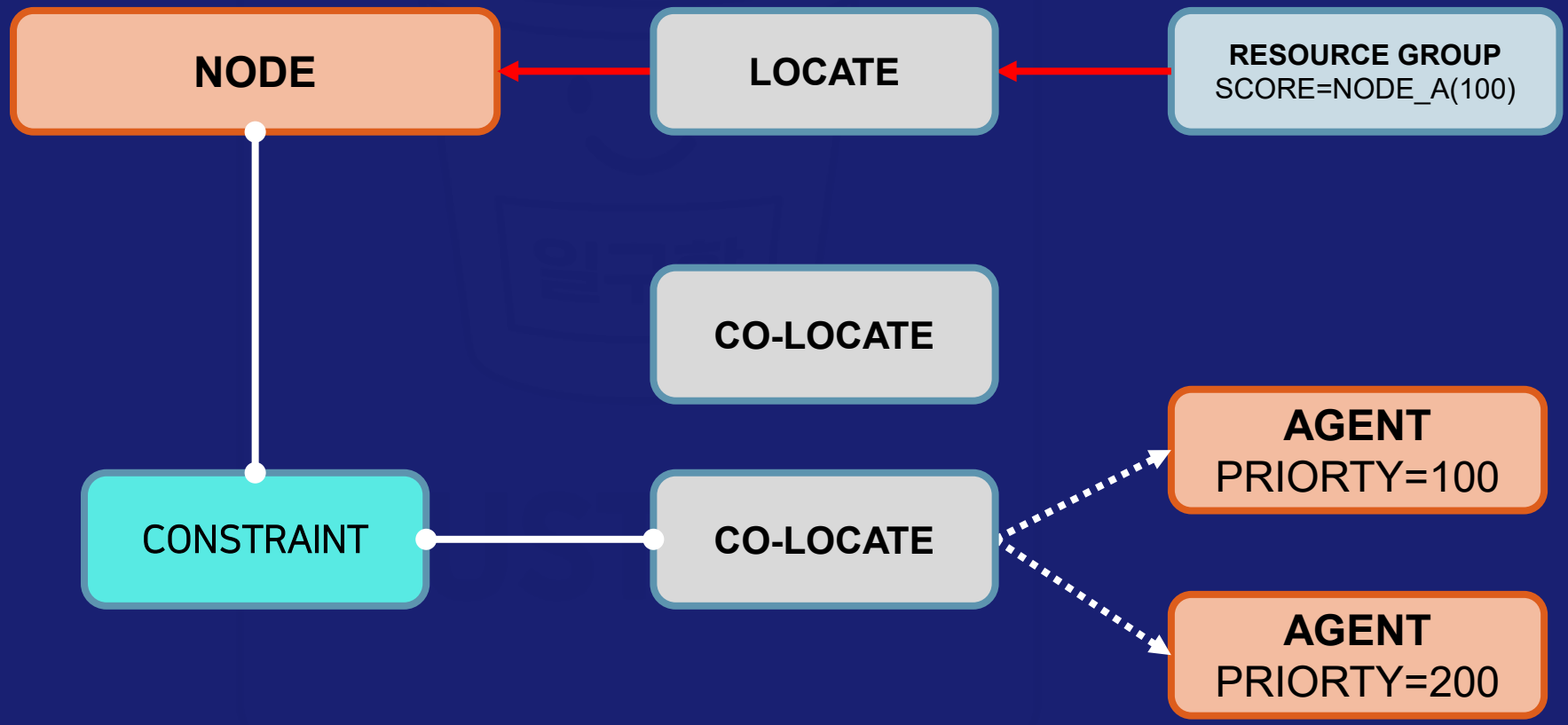
```
resource(A, priority=5)
resource(B, priority=50)
location(A, node1, 100)
location(A, node2, 10)
location(B, node2, 1000)
colocate(B, A)
prefer(node2)
```



DUSTBOX

2023-05-30

INFINITY/SCORE



HOST

페이스메이커에서 제일 작은 구성원은 **노드(node)**혹은 **호스트(host)**라고 부른다. 이들은 최소 한 개 이상이 클러스터에 존재해야 한다. 인증 부분은 host, 자원 관리는 node에서 관리한다.

모든 클러스터는 기본 관리자 계정은 'hacluster', 'root'계정을 가지고 있다. 이를 확인하기 위해서는 'pcs host' 명령어를 통해서 확인이 가능하다.

```
node4# pcs node
node4# pcs cluster
node4# pcs host auth -u hacluster -p centos node10.example.com
```

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 노드에서 어떻게 상호동작을 할지 결정한다.

대기 및 관리상태 전환

```
node1# pcs cluster status
node1# pcs node standby node4.example.com
node1# pcs node unstandby node4.example.com
node1# pcs node utilization node3.example.com
node1# pcs node maintenance node2.example.com
node1# pcs node unmaintenance node2.example.com
node1# pcs node attribute node3.example.com
```

노드 추가 및 삭제

로컬 노드 및 리모트 노드 **추가/삭제**는 아래와 같이 한다.

```
node1# pcs cluster node add node5.example.com --enable --start
node1# pcs remove node5.example.com
node1# pcs delete node5.example.com
node1# pcs add-remote rnode1 rnode1.example.com
node1# pcs remove-remote rnode1
```

노드 추가 및 삭제

로컬 노드 및 리모트 노드 **추가/삭제**는 아래와 같이 한다.

```
node1# pcs cluster node add node6.example.com --enable --start
node1# pcs add-guest guest1.example.com
node1# pcs delete-guest guest1.example.com
node1# pcs remove-guest guest1.example.com
node1# pcs node clear guest1.example.com
```

노드/리모트/게스트 노드 차이

노드들의 차이는 아래와 같다.

구분	Cluster Node (node)	Remote Node	Guest Node
참여 방식	Corosync에 직접 참여 (정식 멤버)	Corosync에는 참여하지 않음 Pacemaker-Remote로 통신	Corosync에는 참여하지 않음Pacemaker-Remote로 통신
위치	클러스터 내부(물리 노드 또는 VM)	**클러스터 외부** (다른 사이트나 네트워크)	**클러스터 내부의 VM 안**
통신 방식	Cluster interconnect (multicast/unicast)	TCP (pacemaker_remote)	TCP (pacemaker_remote)
등록 명령어	`pcs cluster auth` + `pcs cluster setup`	`pcs add-remote <name>`	`pcs add-guest <name>`
필요 서비스	corosync, pacemaker	pacemaker_remote	pacemaker_remote

노드/리모트/게스트 노드 차이

노드들의 차이는 아래와 같다.

구분	Cluster Node (node)	Remote Node	Guest Node
리소스 배치 가능 여부	가능 (모든 리소스)	가능 (제한적)	가능 (제한적)
Failover 가능성	완전한 멤버 간 failover	가능하지만 통신 의존도가 높음	가능하지만 guest가 실행 중일 때만
관리 주체	클러스터 자체	메인 클러스터에서 제어	클러스터가 VM을 통해 생성/삭제/관리
주 용도	일반적인 클러스터 노드	DR, 원격지 자원 관리, 확장 클러스터	VM 기반 테스트, nested 클러스터
예시 구성	`node1, node2, node3`	`remote1.siteB.lab`	`vmguest1.example.com`

속성(PROPERTY)

속성(property)는 corosync나 quorum관련된 변수를 관리한다.

이를 통해서 클러스터의 동작 방식을 결정한다. 예를 들어서 클러스터 깨짐 조건 혹은 최소 노드로 운영 시, 어떠한 노드가 최종적으로 서비스를 가져갈지 정책 부분이다.

속성 명령어

```
node1# 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
```

속성 명령어

```
node1# pcs node maintenance node3.example.com
node1# 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
```

속성 명령어

```
node1# pcs property defaults
node1# pcs property config --all
```

QDEVICE

QDEVICE는 "**쿼럼 장치(QUORUM DEVICE)**"라고 부르기도 한다.

부르기에는 장치라고 부르지만, 실제로는 에이전트 혹은 장치보다는 **중계자(Arbitration)**장치이다. 보통 짝수로 구성이 된 클러스터에서 사용을 권장한다.

하지만, **QDevice**를 많이 사용하는 상황은, 2 노드이나 구성이 되었을 때, 클러스터는 **split-brain**상태에 빠지게 된다. 이때 QDevice는 특정한 알고리즘을 사용하여, 무한 펜싱 같은 상황에 빠지지 않도록 도와준다.

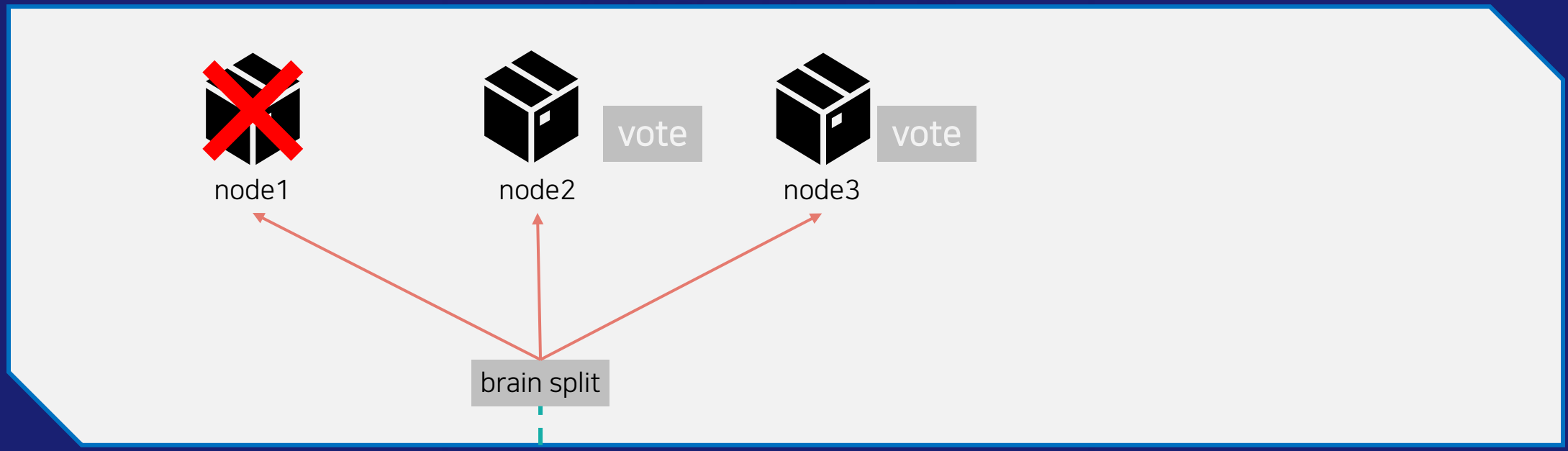
정족수에 문제가 발생하면, 최종적인 노드가 모든 자원을 가져갈지, 클러스터는 서비스 불능 상태로 동작할지 결정한다.

이 장치는 레드햇 기준 **RHEL 8버전 이후**부터는 qdevice, quorum영역에서 둘 다 관리한다. qdevice는 직접적인 장치를 구성 및 설정하며, quorum에서는 쿼럼 장치에 대한 상태 정보를 관리 및 확인한다.

LMS/ffsplit Algorithm

- FF(50, fifty)split: "50:50분할"을 보통 말한다. 노드가 2노드 상태가 되었을 때 분할. 이는 활성 노드 수가 가장 많은 파티션에 정확히 하나의 투표를 제공합니다.
- LMS(Last Man Standing): 마지막. qnetd 서버를 볼 수 있는 클러스터에 남은 노드가 유일한 경우 투표를 반환합니다.

QDEVICE



nodeX# pcs qdevice status net --full

QDEVICE COMMAND

모든 노드에 동일하게 corosync-qdevice 및 qnet를 설치 및 구성. 설치가 되지 않는 경우, 올바르게 qdevice가 동작하지 않음.

```
node1# dnf install --enablerepo=high-availability corosync-qdevice
corosync-qnetd
```

QDEVICE COMMAND

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

node1# for i in {1..4} ; do firewall-cmd --add-service=high-availability &&
firewall-cmd --runtime-to-permanent ; done

node1# pcs cluster auth node1.example.com

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

node1# pcs quorum device add model net host=node1.example.com
algorithm=ffsplit
```

2023-05-30

QDEVICE COMMAND

```
node1# pcs qdevice start net
node1# pcs qdeivce stop net
node1# pcs qdevice enable net
node1# pcs qdevice disable net
node1# pcs qdevice kill net
```

QUORUM COMMAND

```
node1# pcs quorum config
node1# pcs quorum status
node1# pcs quorum device add model net host=qdevice algorithm=ffsplit
node1# pcs quorum config
node1# pcs quorum status
node1# pcs quorum device status
node1# pcs qdevice status net --full
```

QUORUM COMMAND

```
node1# pcs quorum device update model algorithm=lms
node1# pcs quorum device remove
node1# pcs quorum device status
node1# pcs qdevice destroy net
```

QUORUM 설명

- 정족(quorum, 의사 정족수는 최소 3개))
- 정족들은 투표를 통해서 노드의 상태를 확인하여, 구성원 상태를 유지한다.
- 각 시스템은 종족 수를 유지하기 위해서 투표(vote-quorum) 시스템으로 구성한다. 만약, 특정 노드가 투표를 하지 못하면, 해당 노드는 클러스터에서 제외 및 펜싱(fencing)한다.

QUORUM COMMAND

```
node1# 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
```

자원(에이전트)

자원(resource)는 페이스메이커 시스템을 통해서 구성 및 관리가 된다. 자원은 표준화된 쿠버네티스의 서비스이다. 이를 통해서 서비스 상태를 판단하여 어떠한 방식으로 관리할지 결정한다.

관리자(아마도 hacluster?)는 자원들(resources)의 메커니즘에 대해서 굳이 알아야 할 필요가 없다. 어떠한 변수 및 모니터링할지 결정하면 된다.

자원(에이전트)

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

nodeX# ls /usr/lib/ocf/resource.d/heartbeat
```

리소스 관리자는 위의 경로에 OCF(Open Cluster Framework)기반으로 작성된 자원 에이전트가 있다.
관리자 가 원하는 경우, 쉘 스크립트나 파이썬 기반으로 작성 및 구성이 가능하다. 실험을 위해서 아래에서 더미 리소스를 사용하여, 임으로 자원을 생성한다.

RESOURCE

다음과 종류의 자원을 지원한다. 다만, 밑에 upstart는 더 이상 지원하지 않습니다.

1. OCF
2. LSB
3. systemD
4. ~~Upstart(deprecated)~~
5. service
6. fencing

OCF 형식

Open Cluster Framework의 약자. 이를 통해서 클러스터 리소스 생성.

OCF API를 통해서 클러스터 랩(Cluster Labs)에서 표준 서비스를 구성한다. 이 자원들은 페이스 메이커 클러스터에서 사용하도록 구성이 되어있다. 클러스터에서 사용하는 리소스들은 OCF에 맞추어서 작성 및 구성이 되어있다.

동작 방은 OCF사양에 맞추어 되어 있으며, 이를 통해서 자원 에이전트의 start/stop/monitor를 수행한다.

systemd 형식

현재 대다수 현대 리눅스는 systemd기반으로 시작하며 서비스를 관리한다. 이전에는 System V기반으로 스크립트로 사용하였다.

페이스메이커는 자동으로 systemd의 서비스를 관리 및 운영한다. 페이스메이커에서 서비스 구성 시, 절대로 **systemctl enable, systemctl enable --now**와 같은 옵션을 사용하지 않는다. 이 옵션은 페이스메이커에서 자동으로 구성 및 활성화 한다.

테스트 용도로 간단하게 **더미(dummy)**자원을 등록해서 명령어를 사용한다.

LSB(Linux Standard Base) 형식

LSB는 이전에 사용하던 SysV나 혹은 Up-Start로 구성된 서비스 스크립트이다. 현재는 systemd를 사용하기 때문에 많이 사용하지 않지만, 아직까지 이전 구형 배포판을 사용하는 경우, "/etc/init.d"에 있는 스크립트를 사용하여 처리가 가능하다.

다만, 현재는 많이 사용하지 않기 때문에, 지금은 사용자가 직접 만든 LSB기반의 OCF에이전트에서 많이 활용한다.

STATUS

'status'명령어는 클러스터에서 사용하는 pcsd, Corosync의 상태를 확인한다. 또한 이 명령어를 통해서 페이스메이커 및 에이전트 상태 정보 확인이 가능하다.

STONITH

펜싱 장치는 아래 명령어로 확인이 가능하다.

```
node1# pcs stonith list
```

STATUS COMMAND

클러스터 및 자원 상태를 확인하기 위해서 아래 명령어로 확인이 가능하다.

```
node1# pcs status
node1# pcs resource status <NAME>
node1# pcs status nodes
```

STONITH

STONITH의 약자는 **Shoot the other node in the head**의 약자이며, 한국어로 직역하면, **머리에 총 쏘기**이다. 아래 예제 명령어는 올바르게 동작하지는 않으며, 사용방법에 대한 예제이다.

우리는 보통 이 장치를 **fence device**라고 부른다. 기본적으로 모든 클러스터에는 최소 한 개의 STONITH장치가 있어야 하며, 존재하지 않는 경우에는 리소스가 올바르게 동작하지 않을 수 있다. 만약, STONITH장치가 필요하지 않는 경우, 아래와 같이 설정이 가능하다.

```
node1# pcs status
WARNING: no stonith devices and stonith-enabled is not false
node1# pcs property set stonith-enabled=false
node1# pcs property config
node1# crm_verify -L
```

에이전트 차단 설정

```
node1# pcs stonith list

node1# dnf search fence-agents-all

node1# dnf install fence-agents-ipmilan

node1# pcs stonith describe fence_ipmilan

node1# 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
```

차단 명령어 테스트

미리 구성한 STONITH파일이 있는 경우 아래 명령어로 불러와서 적용이 가능.

```
node1# pcs -f stonith_cfg stonith
node1# pcs -f stonith_cfg property set stonith-enabled=true
node1# pcs -f stonith_cfg property
node1# pcs cluster stop node2
node1# stonith_admin --reboot node2
```

tag

페이스메이커에서 지원하는 태그 기능이다. 이 기능은 레드햇 기준으로 RHEL 8버전부터 지원한다. 간단하게 특정 자원에 태그를 생성하여, 손쉽게 자원 관리가 가능하다.

```
node1# pcs resource create tag-dummy1 ocf:pacemaker:Dummy
node1# pcs resource create tag-dummy2 ocf:pacemaker:Dummy
node1# pcs tag create dummy-tag-resources tag-dummy1 tag-dummy2
node1# pcs tag list | config
node1# pcs tag delete dummy-tag-resources
```

더미 리소스 기반 랩

```
node1# pcs resource create dummy1 ocf:pacemaker:Dummy
node1# pcs resource create dummy2 ocf:pacemaker:Dummy
node1# pcs resource create dummy3 ocf:pacemaker:Dummy
node1# pcs resource create vip ipaddr2 ip=192.168.90.250 cidr_netmask=24
node1# pcs resource create res1 ocf:pacemaker:Dummy --group grp1
node1# pcs resource create res2 ocf:pacemaker:Dummy --future group grp1
node1# pcs resource create res1 ocf:pacemaker:Dummy --future group grp2
node1# pcs resource create res2 ocf:pacemaker:Dummy --future group grp2
node1# pcs constraint location dummy3 prefers node3.example.com
node1# pcs constraint location
```

더미 리소스 기반 랩

```
node1# pcs resource create res-stickiness-1 ocf:pacemaker:Dummy
node1# pcs resource create res-stickiness-2 ocf:pacemaker:Dummy
node1# pcs resource create res-stickiness-3 ocf:pacemaker:Dummy
node1# pcs resource create res-stickiness-4 ocf:pacemaker:Dummy
node1# pcs constraint location res-stickiness-1 prefers
node1.example.com=100
node1# pcs constraint location res-stickiness-2 prefers
node1.example.com=INFINITY
node1# pcs constraint location res-stickiness-3 prefers node1.example.com=-
INFINITY
```

서비스 구성

LVM2

NFS

TWO NODE

APACHE

GFS2

RESOURCE MOVE

LVM2

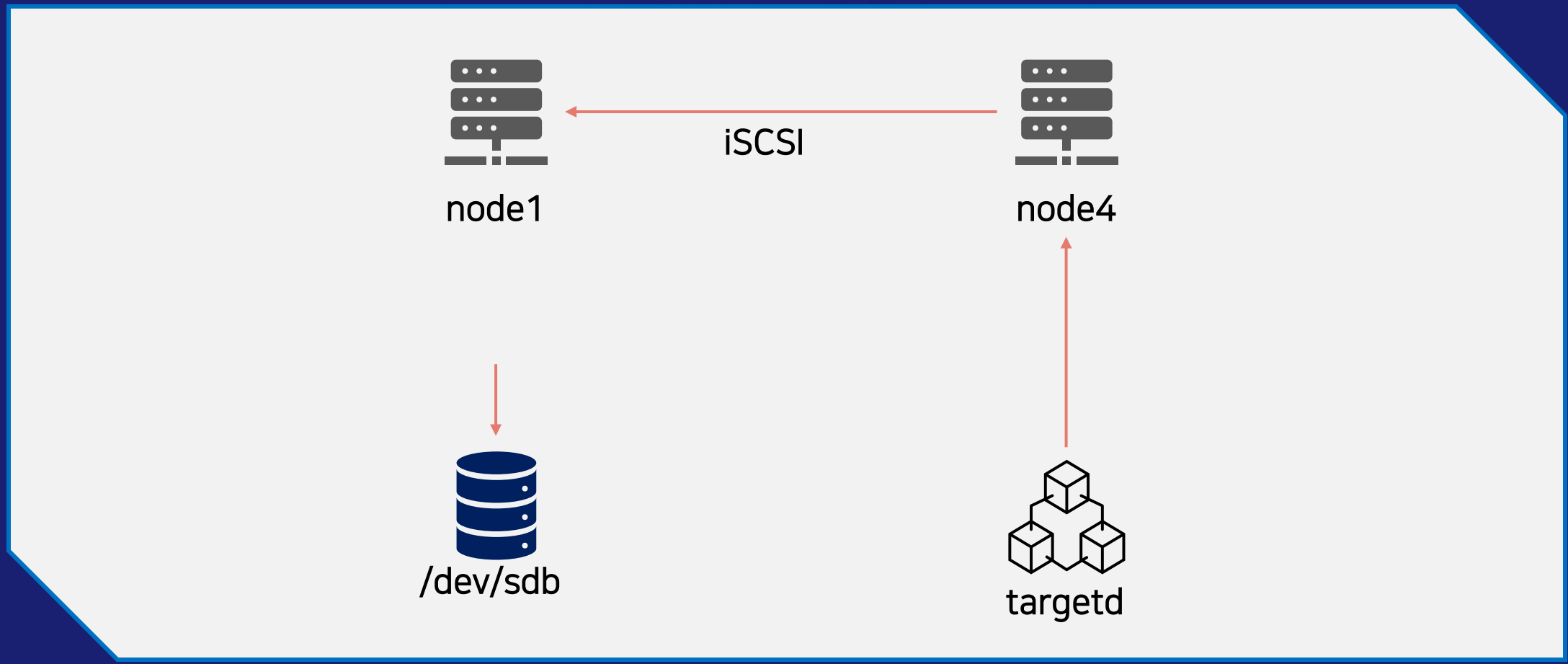
서비스 구성

2023-05-30

LVM2

시작전에 모든 노드에 스냅샷 생성하세요.

LVM2



LVM2 FENCE

최소 한 개의 Fence장치가 구성이 되어야 한다.구성이 안된 경우, 에이전트가 올바르게 동작하지 않는다. 아래와 같이 펜스 장치를 생성한다. 앞서 미리 생성하였기 때문에, 생성하지 않는 경우 생성한다.

```
node1# dnf search --enablerepo=highavailability fence-agents-all watchdog
node1# for h in node1 node2 node3; do
  sshpass -procky ssh root@$h "dnf install -y --enablerepo=highavailability
fence-agents-all watchdog"
  ssh root@$h "cp /usr/share/cluster/fence_scsi_check /etc/watchdog.d/ &&
               systemctl enable --now watchdog"
done
```

LVM2 FENCE

위의 내용 계속 이어서 진행한다. 디스크의 WWN 정보를 확인 후, 올바르게 변경한다.

```
node1# blkid
node1# ls -l /dev/disk/by-id/
node1# pcs stonith create iscsi-shooter fence_scsi
pcmk_host_list="node1.example.com node2.example.com node3.example.com"
devices=/dev/disk/by-id/wwn-0x6001405b1f65e2e7ccf4bb0aa07b8bad meta
provides=unfencing
```

LVM2

```
node1/2/3/4# grep -E 'system_id_source|use_lvmlockd' /etc/lvm/lvm.conf || t
rue
system_id_source = "uname"
node1# scp /etc/lvm/lvm.conf root@node{2..3}:/etc/lvm/lvm.conf
node1# parted --script /dev/sdb "mklabel gpt"
node1# parted --script /dev/sdb "mkpart primary 0% 100%"
node1# parted --script /dev/sdb "set 1 lvm on"
```

LVM2

```
node1# pvcreate /dev/sdb1

node1# vgcreate --setautoactivation n --shared --locktype dlm vg_ha_lvm
/dev/sdb1

node1/2/3/4# vi /etc/lvm/lvm.conf

use_lvmlockd = 1

node1# vgs -o+systemid

node1# lvcreate -l 100%FREE -n lv_ha_lvm vg_ha_lvm
```

LVM2

```
node1# mkfs.xfs /dev/vg_ha_lvm/lv_ha_lvm
node1# for h in node2 node3; do
  ssh root@$h "partprobe -a /dev/sdb; lvmdevices --adddev /dev/sdb1; lvm pvscan --cache --activate ay"
  ssh root@$h "vgs -o+systemid"
done
```

LVM Scanning issue: https://access.redhat.com/solutions/6967600

LVM2

```
node1# pcs resource create lvm_ha_iscsi LVM-activate vgname=vg_ha_lvm vg_access_mode=system_id --future group ha_lvm_group op start timeout=90s op stop timeout=90s

node1# pcs resource create lvm_ha_mount FileSystem device=/dev/vg_ha_lvm/lv_ha_lvm directory=/home/lvm_directory fstype=xfs --future group ha_lvm_group

node1# pcs status

node1# df
```

LVM2

만약, 기존에 dlm를 사용을 원하는 경우 다음과 같이 패키지 설치가 가능하다. 다만, 현재는 8버전 이후부터는 선태사항이다.

```
nodeX# dnf search --enablerepo=resilientstorage,highavailability dlm lvm2-lockd
> dlm
> lvm2-lockd
nodeX# dnf install --enablerepo=resilientstorage,highavailability dlm lvm2-lockd -y
node1# vgchange vg_ha_lvm -an
node1# vgchange --systemid $(uname -n) vg_ha_lvm
node1# vgcreate --shared --locktype=dlm /dev/sdb1 vg_ha_lvm
node1# pcs resource create lvm_ha_iscsi LVM-activate vgname=vg_ha_lvm
vg_access_mode=lvmlockd  activation_mode=shared --group ha_lvm_group
```

연습문제

클러스터 node3의 /dev/sdc 디스크에 LVM2구성한다. 작업 시작전에 리눅스 커널에 디스크 정보를 추가 및 갱신을 한다. 명령어는 아래 명령어를 참조하여 작업을 수행한다.

1. partprobe
2. cfdisk/gdisk/fdisk
3. wipefs

연습문제

볼륨그룹 이름은 ha_lvm_vg_sdc로 구성한다.

- vi /etc/lvm/lvm.conf

논리 디스크 이름은 ha_lvm_lv_sdc로 구성한다.

디스크 크기는 모든 공간을 사용한다.

- lvcreate -n -l 100%Free

마운트 위치는 /mnt/ha_sdc_lvm에 연결한다.

- 디렉터리는 "Filesystem"에이전트가 자동으로 구성
- 파일 시스템 포멧은 원하는 형식으로(ext4, xfs, vfat)

해당 위치에 dnf로 high-availability, resilientstorage 패키지 미러.

NFS

서비스 구성

NFS

시작전에 모든 노드에 스냅샷 생성하세요.

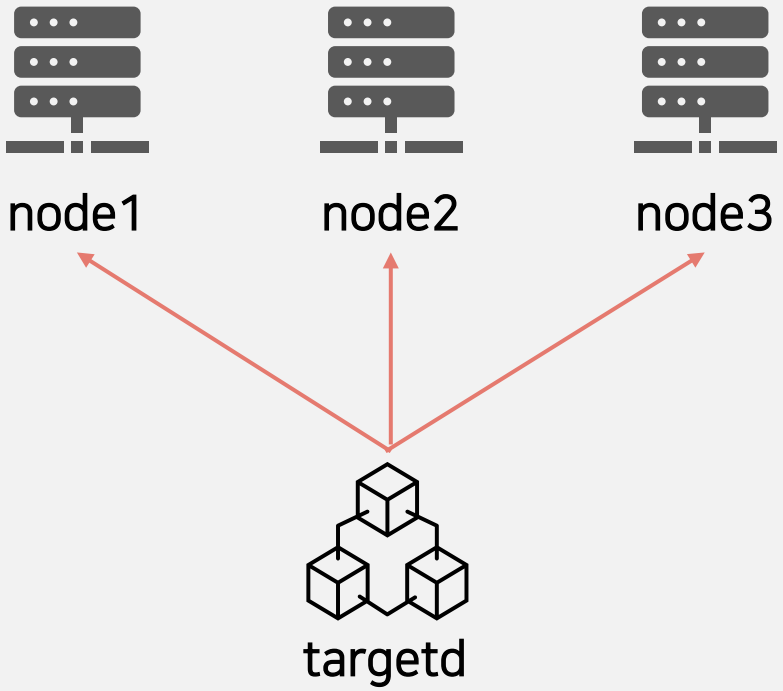
NFS서버 설명

LVM2로 구성한 디렉터리를 NFS서버로 node1에 연결할 수 있도록 한다. 조건은 다음과 같은 조건으로 동작되게 구성한다.

시작전에 스냅샷 혹은 체크포인트 생성 후 진행한다.

1. node1/node2/node3/node4에는 NFS서버를 **자원(resource)기반 및 하트비트(heartbeat)**으로 구성한다.
2. 구성 시 사용하는 자료는 LVM2으로 구성된 디스크를 사용한다.
3. 모든 디스크는 targetd에서 제공한 iSCSI장치를 사용한다.
4. 구성된 디스크는 **/mnt/nfs-data**를 사용한다.

NFS자원 구성



DUSTBOX

2023-05-30

패키지/방화벽/디렉터리

패키지 및 방화벽 구성합니다.

```
# (선택) 완전 차단 해제 대신 방화벽 서비스만 엽니다.
for i in {1..3}; do
  ssh root@node${i} "dnf -y install nfs-utils; \
    firewall-cmd --add-service={nfs,rpc-bind,mountd}; \
    firewall-cmd --runtime-to-permanent; \
    mkdir -p /home/nfs-data/nfs-root/share01"
done
```

페이스메이커 리소스 생성

자원을 생성 및 구성한다.

```
# VIP → FS → nfsserver → exportfs(root) → exportfs(share01) → nfsnotify
pcs resource create nfs_vip ocf:heartbeat:IPaddr2 ip=192.168.90.250
cidr_netmask=24 --group ha_lvm_group

pcs resource create nfs_fs ocf:heartbeat:Filesystem \
  device=/dev/vg_ha_lvm/lv_ha_lvm directory=/home/nfs-data fstype=xfs \
  --group ha_lvm_group

pcs resource create nfs_daemon ocf:heartbeat:nfsserver \
  nfs_shared_infodir=/home/nfs-data/nfsinfo \
  nfs_no_notify=true \
  --group ha_lvm_group
```

페이스메이커 리소스 생성

자원을 생성 및 구성한다.

```
# NFSv4 루트 (fsid=0) – v4 권장 옵션 예시
pcs resource create nfs_root ocf:heartbeat:exportfs \
  clientspec=192.168.90.0/24 \
  options=rw,sync,no_subtree_check,crossmnt,fsid=0 \
  directory=/home/nfs-data/nfs-root \
  --group ha_lvm_group

# 실제 공유
pcs resource create nfs_share01 ocf:heartbeat:exportfs \
  clientspec=192.168.90.0/24 \
  options=rw,sync,no_subtree_check \
  directory=/home/nfs-data/nfs-root/share01 \
  fsid=1 \
```

페이스메이커 리소스 생성

자원을 생성 및 구성한다.

```
# 클라이언트 락 회복 알림 (VIP 기준)
pcs resource create nfs_notify ocf:heartbeat:nfsnotify \
  source_host=192.168.90.250 \
  --group ha_lvm_group

pcs status
pcs resource group list
```

확인 및 마운트

올바르게 구성이 되었는지 확인한다.

```
# v3 방식으로 내보낸 목록 확인(선택)
showmount -e 192.168.90.250

# 클라이언트 (예: node3)
mkdir -p /mnt/nfs-data
# NFSv4 권장
mount -t nfs -o vers=4.2 192.168.90.250:/share01 /mnt/nfs-data
# 확인
mount | grep nfs
```

NFS연습문제

LVM2의 **"/home/lvm_directory"**는 어디로..? 이 부분에 대해서 별도의 LVM2 장치를 구성 및 생성하여 NFS서비스에 사용하도록 해야 한다.

새로 구성된 NFS의 LVM2볼륨의 이름은 **vg_ha_lvm/lv_ha_nfs**으로 구성 및 생성한다. VG에서 생성된 LV는 **/home/nfs_directory**으로 연결 및 구성이 된다.

1. target에서 제공된 블록장치 /dev/sdc를 NFS서비스용도로 구성.
2. /dev/sdc디스크는 볼륨 그룹 vg_ha_lvm에 포함이 되어 있어야 됨.
3. 논리 디스크의 이름은 lv_ha_nfs으로 구성.
4. nfs_exports를 통해서 nfs_data라는 이름으로 NFS 디렉터리 제공 및 마운트 가능.
5. nfs에서 사용하는 리소스 그룹의 이름은 nfs_group으로 사용한다.

2023-05-30

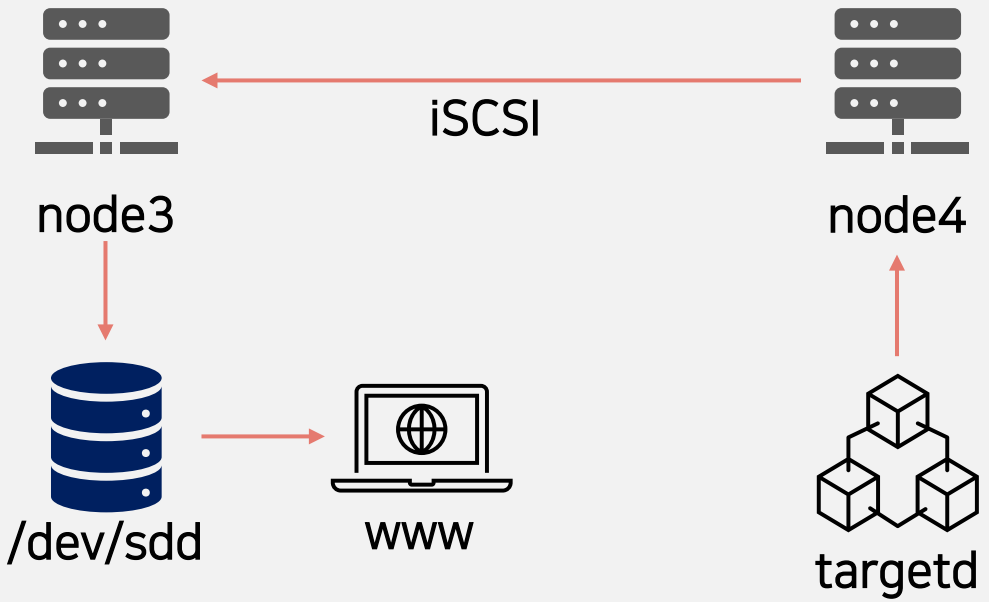
APACHE

서비스 구성

APACHE

시작전에 모든 노드에 스냅샷 생성하세요.

APACHE



APACHE

랩 시작 전, 모든 노드에 방화벽을 구성한다. 그리고, SELinux를 사용하면 중지한다. 사용을 원하는 경우 restorecon명령어로 웹 서버 디렉터리의 컨텍스트를 초기화 한다.

```
node1/2/3/4# for i in {1..4} ; do ssh root@node${i} "firewall-cmd --add-service={http,https} && firewall-cmd --runtime-to-permanent && setenforce 0" ; done
```

APACHE

```
node1# for i in {1..4} ; do ssh root@node${i} "dnf install httpd -y" ; done
node1# vi /etc/httpd/conf.d/server-status.conf
<Location /server-status>
    SetHandler server-status
    Require local
</Location>
```

APACHE

```
node1# for i in {1..4} ; do scp /etc/httpd/conf.d/server-status.conf
root@node${i}:/etc/httpd/conf.d/ ; done

node1# mount /dev/vg_ha_lvm/lv_ha_lvm /var/www/html

node1# umount /var/www/html

node1# echo "Hello Hate Pacemaker World" > /home/lvm_directory/index.html

node1# pcs resource create httpd_fs ocf:heartbeat:Filesystem device=
/dev/vg_ha_lvm/lv_ha_lvm directory=/var/www/html fstype=xfs --future group
ha_httpd
```

2023-05-30

APACHE

```
node1# pcs resource create httpd_vip ocf:heartbeat:IPaddr2 ip=192.168.90.21
0 cidr_netmask=24 nic=eth1 --future group ha_httpd

node1# pcs resource create website ocf:heartbeat:apache configfile=/etc/htt
pd/conf/httpd.conf statusurl=http://127.0.0.1/server-status --future group
ha_httpd

node3# curl http://192.168.90.210/index.html
```

node1/2/3/4에다가 웹 서버 구성을 원하는 경우, 'apache-status'도 같이 설정이 되어야 됨.

연습문제

다시 아파치 서비스를 구성한다. 아래와 같은 조건으로 동작하도록 한다.

1. 노드 **1번/3번**에 아파치 서비스를 설치한다.
2. 해당 아파치 서비스는 VIP를 **192.168.90.254**를 사용한다.
3. 메시지는 "Hello Apache"라고 출력한다.
4. **기존에 사용하던 자원과 충돌이 되지 않도록 구성한다.**
5. 더 이상 사용할 디스크가 없는 경우 target를 통해서 새로 생성 후 노드에 전달.
6. "/dev/sdd"디스크는 볼륨 그룹 "vg_ha_lvm"에 포함이 되어 있어야 됨.
7. 논리 디스크의 이름은 "lv_ha_httpd"으로 구성.
8. nfs에서 사용하는 리소스 그룹의 이름은 "httpd_group"으로 사용한다.

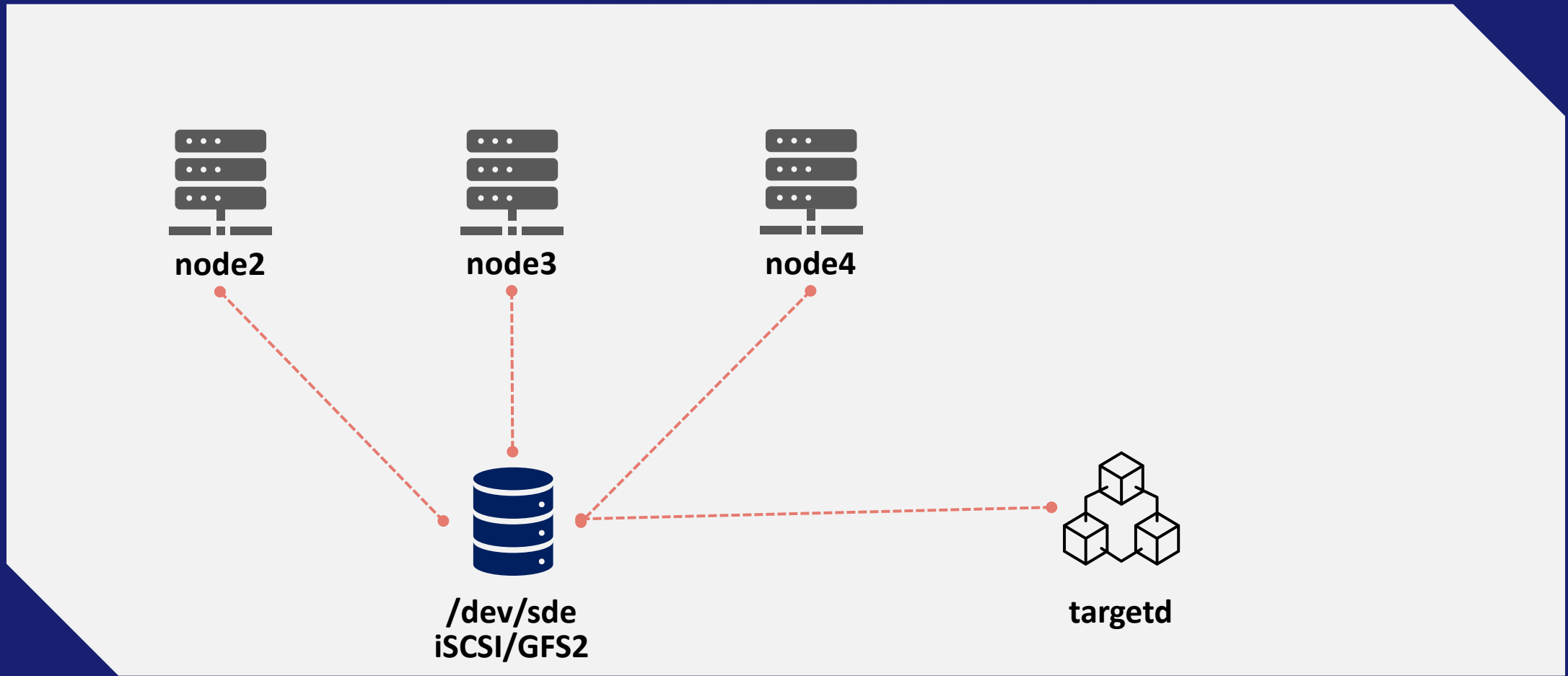
GFS2

서비스 구성

GFS2

시작전에 모든 노드에 스냅샷 생성하세요.

GFS2



DUSTBOX

2023-05-30

GFS2 소개

XFS 파일 시스템은, 렌더링 파일 시스템으로 사용. **성능 및 안전성**은 높지만, 단점이 다중 연결은 지원하지 않음. 이러한 문제를 해결하기 위해서 GFS파일 시스템을 개발.

SGI가 폐업을 하면서, 리눅스 커뮤니티에서 GFS기반으로 소스코드를 공개 및 릴리즈. 리눅스에서 **GFS2**라는 이름으로 사용. 이를 **DLM(Distributed Lock Manager) 및 cLVM(Cluster LVM)** 도입하면서, 본격적으로 공유 파일 시스템을 제공 시작. 페이스메이커에서는 이를 에이전트 기반으로 관리 및 지원한다.

- v1.0 (1996) SGI IRIX only
- v3.0 Linux port
- v4 journaling

GFS2 소개

- v5 Redundant Lock Manager
- v6.1 (2005) Distributed Lock Manager
- Linux 2.6.19 - GFS2 and DLM merged into Linux kernel
- Red Hat Enterprise Linux 5.3 releases the first fully supported GFS2

GFS2

레드햇은 버전별로 LVM2의 락킹 관리자가 다르다.

RHEL 7버전까지 clvm를 사용하다가, RHEL 7.3이후로부터는 dlm으로 락킹 매니저를 전환을 시작하였다. 레드햇 기준으로 락킹 매니저는 다음과 같이 사용한다.

- **RHEL 7/8**: CLVM
- **RHEL 8/9**: DLM

페이스메이커에서 차이는 clvm를 기반으로 설정하느냐, 혹은 dlm기반으로 설정하느냐 차이다. LVM2의 구조 변경으로 인하여, 최근 릴리즈(RHEL 8/9)인 경우에는 가급적이면 dlm기반으로 사용을 매우 권장한다.

ISCSI 서버 구성

디스크가 부족한 경우, target서버에서 추가로 디스크 구성해서 특정 혹은 모든 노드에 추가 제공. 명령어는 아래 내용 참고.

```
node4# targetcli backstores/fileio create sdf /var/lib/iscsi_disks/sdf.img 2G
node4# targetcli iscsi/iqn.2023-02.com.example:blocks/tpg1/luns/ create /backstores/fileio/sdf/
node4# targetcli saveconfig
node1# for i in {1..4} ; do ssh root@node${i} "iscsiadm -m session --rescan" ; done
node1# for i in {1..4} ; do ssh root@node${i} "dnf --enablerepo=resilientstorage -y install gfs2-utils dlm lvm2-lockd" ; done
```

ISCSI 서버 구성

위의 내용 이어서…

```
node1# pcs stonith create scsi-shooter fence_scsi-sdf \
pcmk_host_list="node1.example.com node2.example.com node3.example.com
node4.example.com" devices=/dev/disk/by-id/wwn-<ID> meta provides=unfencing
```

GFS2 자원 구성

```
node1# grep -e use_lvmlockd -e system_id_source /etc/lvm/lvm.conf
node1# for i in {1..4} ; do scp /etc/lvm/lvm.conf node${i}:/etc/lvm/lvm.conf ; done
node1# pcs property set no-quorum-policy=freeze
node1# pcs property set stonith-enabled=true
node1# pcs resource create dlm ocf:pacemaker:controld op monitor interval=30s on-fail=fence --group gfs2_locking
node1# pcs resource clone gfs2_locking meta interleave=true
node1# pcs resource create lvmlockd ocf:heartbeat:lvmlockd op monitor interval=30s on-fail=fence --group gfs2_locking
```

파티션 생성 및 구성

```
node1# parted --script /dev/sdf "mklabel gpt"
node1# parted --script /dev/sdf "mkpart primary 0% 100%"
node1# parted --script /dev/sdf "set 1 lvm on"
```

GFS2에 LVM2 구성

```
node1# pvcreate /dev/sdf1

node1# vgcreate --shared vg_gfs2 /dev/sdf1

node1# vgcreate --lock-start /dev/sdf1

node1# lvcreate -l 100%FREE -n lv_gfs2disk vg_gfs2

node1# mkfs.gfs2 -j4 -p lock_dlm -t ha_cluster_lab:gfs2disk
/dev/vg_gfs2/lv_gfs2disk

node1# for i in {1..4} ; do ssh root@node${i} partprobe && sleep 2 &&
lvmdevices --adddev /dev/sdf1 && vgchange --lock-start vg_gfs2 ; done
```

https://clusterlabs.org/pacemaker/doc/2.1/Clusters_from_Scratch/html/active-active.html

GFS2 리소스 생성 및 구성

```
node1# pcs resource create gfs2_lv ocf:heartbeat:LVM-activate
lvname=lv_gfs2disk vgname=vg_gfs2 activation_mode=shared
vg_access_mode=lvmlockd --group gfs2_vg

node1# pcs resource clone gfs2_vg meta interleave=true

node1# pcs constraint order start gfs2_locking-clone then gfs2_vg-clone

node1# pcs constraint colocation add gfs2_vg-clone with gfs2_locking-clone

node1# pcs resource create gfs2_fs ocf:heartbeat:Filesystem
device="/dev/vg_gfs2/lv_gfs2disk" directory="/home/lvm_gfs2-share"
fstype="gfs2" options=noatime op monitor interval=10s on-fail=fence --group
gfs2_vg
```

GFS2 블록장치 처리(TS용도)

블록장치인 "vg_gfs2/lv_gfs2disk"는 **멀티락킹** 문제로 올바르게 처음에 인식이 되지 않는다. 실제로 커널에서는 인식을 하고 있지만, 블록 장치를 생성하지 못하여 마운트 하지 못한다. 다음과 같은 단계를 수행 후, 장치가 생성되지 않으면 모든 노드를 재시작 한다.

이와 같이 하였는데도 동작하지 않으면, 모든 노드를 종료 후, 다시 기동하면 올바르게 장치가 생성 및 마운트가 된다.

```
nodeX# vgchange -ay
  LV locked by other host: vg_gfs2/lv_gfs2disk
  Failed to lock logical volume vg_gfs2/lv_gfs2disk.
  0 logical volume(s) in volume group "vg_gfs2" now active
  3 logical volume(s) in volume group "cs" now active
nodeX# vgchange --lock-start vg_gfs2
nodeX# blockdev --report
nodeX# blockdev --rereadpt /dev/sdf
nodeX# udevadm settle
nodeX# ls -l /dev/vg_gfs2/lv_gfs2disk
/dev/vg_gfs2/lv_gfs2disk → ../dm-3
```

GFS2(설명)

```
[root@node1 ~]# pcs property set no-quorum-policy=freeze
[root@node1 ~]# pcs property
Cluster Properties: cib-bootstrap-options
  cluster-infrastructure=corosync
  cluster-name=ha_cluster_lab
  dc-version=2.1.7-4.el9-4db20b0
  have-watchdog=false
  last-lrm-refresh=1708671743
  no-quorum-policy=freeze
[root@node1 ~]# pcs resource cleanup
Cleaned up all resources on all nodes
[root@node1 ~]#
```

2023-05-30

GFS2(설명)

```
nodeX# dnf --enablerepo=highavailability,resilientstorage -y install gfs2-utils dlm lvm2-lockd
nodeX# systemctl enable --now lvmlockd dlm
nodeX# vi /etc/lvm/lvm.conf
use_lvmlockd 찾아서 변경
nodeX# partprobe /dev/sdf
nodeX# lvmdevices --adddev /dev/sdf1
nodeX# vgchange --lock-start vg_gfs2
```

만약, "/dev/misc/dlm-*"구성이 안되어 있으면 다음처럼 작업 수행

```
# lsmod | grep dlm
# modprobe -r dlm
# modprobe dlm
# ls -l /dev/misc/dlm-*
# dracut -f
```

GFS2(설명)

```
vgcreate --shared --locktype dlm vg_gfs2 /dev/sdf1
lvmdevices --adddev <DEVICE_NAME>
vgchange --lockstart gfs2_vg dev/sdf1
lvcreate --activate ys | lvcreate -ays
pcs constraint order start gfs2_locking-clone then gfs2_vg-clone
pcs resource clone gfs2_vg interleave=true
systemctl daemon-reload
dracut -f
```

2023-05-30

GFS TOOLS(설명)

```
nodeX# dlm_tools ls
nodeX# vgchange --lockstart
nodeX# blkid
nodeX# pcs status --full
```

연습문제

위의 내용은 다시 롤백 후, 재구성 해보세요!

- 기존의 내용은 절대로 건들지 마세요.
- 왜 LVM2에서 기존 locking type과 새로운 DLM Locking Type은 같이 사용이 불가능할까요?

TOMCAT

서비스 구성

TOMCAT

시작전에 모든 노드에 스냅샷 생성하세요.

TOMCAT

톰캣 서비스는 별도로 컴파일 하는 경우가 아니면, 바로 RPM패키지로 구성 및 실행이 가능하다. 소스 컴파일 하는 경우, 별도로 OpenJDK설정도 필요하다. 이 랩은 RPM기반으로 톰캣 서비스를 구성한다.

TOMCAT

테스트용 소스코드를 아래 주소에서 내려받기 한다.

```
node1# dnf install git python3-pip maven-openjdk11.noarch
node1# pip install github-clone
node1# ghclone https://github.com/tangt64/codelab/tree/main/java/blog
node1# cd blog
node1# mvn clean package
```

위의 작업이 완료가 되면 톰캣 설정을 다음처럼 수정한다.

```
node1# vi /etc/tomcat/tomcat.conf
# JAVA_OPTS="-
Djavax.sql.DataSource.Factory=org.apache.commons.dbcp.BasicDataSourceFactor
y"
```

TOMCAT

추가적으로 라이브러리를 복사한다.

```
node1# cp docker/postgresql-9.4.1212.jar
/usr/share/tomcat/webapps/ROOT/WEB-INF/lib/
```

서비스 접근이 되는지 확인. 데이터베이스 접근 오류 발생은 정상.

```
node1# systemctl start tomcat
```

이후 모든 노드에 동일하게 위와 같이 톰캣 설치 및 설정 진행.

```
node1# for i in {1..4} ; do ssh root@node${i} "dnf install tomcat java-11-
openjdk -y" ; done
```

TOMCAT

```
node1# pcs resource create tomcat_vip ocf:heartbeat:IPaddr2 ip=192.168.90.2
20 cidr_netmask=24 op monitor interval=30s --group tomcat

node1# pcs resource create tomcat_service ocf:heartbeat:tomcat java_home="/
usr/lib/jvm" catalina_home="/usr/share/tomcat" tomcat_user="tomcat" op moni
tor interval="15s" --group tomcat

node1# pcs resource create tomcat_service systemd:tomcat --group tomcat

node1# pcs constraint colocation set tomcat_vip tomcat_service

node1# curl http://192.168.90.220:8080
```

2023-05-30

TOMCAT(설명)

아래 명령어에서 "colocation"은 두 개의 서비스를 하나로 묶어서 리소스가 이동이 된다. "tomcat_vip", "tomcat_service"는 다른 노드로 이동시, 같은 위치로 이동한다.

```
pcs constraint colocation set tomcat_vip tomcat_service
```

연습문제

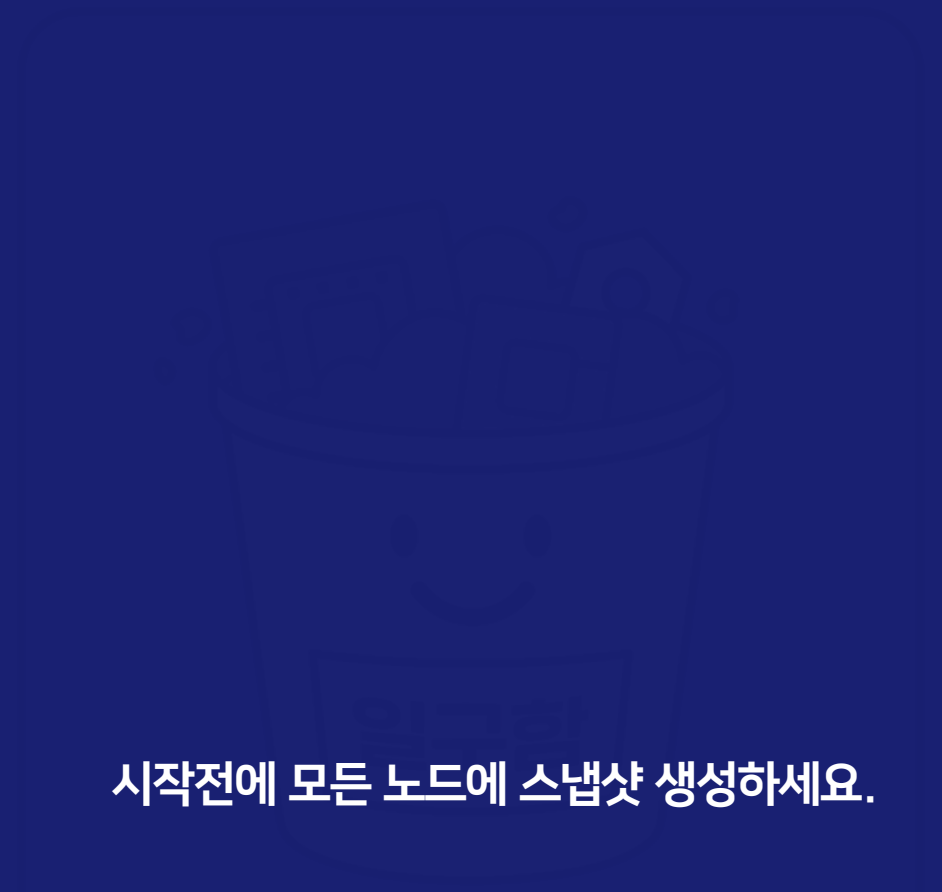
위의 내용은 다시 롤백 후, 재구성.

- 기존의 내용은 절대로 수정하지 않는다.
- 다른 리소스(resource)에도 colocation를 적용을 시도한다.

PGSQL

서비스 구성

PGSQL



시작전에 모든 노드에 스냅샷 생성하세요.

PGSQL

일반 블록 파일 시스템 기반으로 PgSQL를 구성하기 위해서는 DRBD나 혹은 GlusterFS, Ceph 저장소 구성이 필요하다. 여기서는 추가적인 복제 시스템에 대해서는 다루지 않는다. PgSQL도 위에서 사용한 톰캣처럼 RPM기반으로 진행한다.

디스크는 /dev/sdd를 사용한다. 만약, 부족한 경우 targetd에서 추가해서 노드에 제공한다. 앞에서 진행하였던 DRBD가 미리 선행이 되어야 랩 진행이 가능하다. 이전에 구성 하였던, 로컬 디스크의 DRBD는 사용하지 않으며, iSCSI으로 할당 받은 디스크로 DRBD를 구성해야 한다.

DRBD 블록 설정

```
node1# vi /etc/drbd.d/resource1.res
resource resource1 {
  on node1.example.com {
    device    /dev/drbd2;
    disk      /dev/drbd-demo/drbd-pgsql;
    address   192.168.90.110:7789;
    meta-disk internal;
  }
```

DRBD 블록 설정

```
on node2.example.com {
  device    /dev/drbd2;
  disk      /dev/drbd-demo/drbd-pgsql;
  address  192.168.90.120:7789;
  meta-disk internal;
}
```

DRBD 블록 설정

```
on node3.example.com {
  device    /dev/drbd2;
  disk      /dev/drbd-demo/drbd-pgsql;
  address  192.168.90.130:7789;
  meta-disk internal;
}
```

DRBD 블록 설정

```
  on node4.example.com {
    device    /dev/drbd2;
    disk      /dev/drbd-demo/drbd-pgsql;
    address  192.168.90.140:7789;
    meta-disk internal;
  }
}
```

DRBD SETUP FOR PGSQL

DRBD를 사용하여 PgSQL를 디스크를 구성한다. 혹은 해당 자원을 GFS2기반으로 구성하여도 상관 없다.

```
node1# for i in {1..4} ; do scp /etc/drbd.d/resource1.res
root@node${i}:/etc/drbd.d/resource1.res
node1# drbdadm create-md resource1
node1# drbdadm up resource1
node1# drbdadm status resource1
node1# drbdadm primary --force resource1
node1# drbdadm status resource1
node1# lsblk
node1# mkfs.xfs /dev/drbd2
node1# mkdir -p /mnt/drbd
```

DRBD SETUP FOR PGSQL

올바르게 동작하는지 확인한다.

```
node1# mount /dev/drbd2 /mnt/drbd
node1# systemctl daemon-reload
node1# cd /mnt/drbd
node1# touch test{1..100}
node1# umount /mnt/drbd
node1# drbdadm secondary resource0
```

DRBD SETUP FOR PGSQL

```
node1/2/3/4# drbdadm up resource0
node1/2/3/4# drbdadm status resource0
node1/2/3/4# drbdadm primary resource0
node1/2/3/4# mkdir -p /mnt/test-pgsql
node1/2/3/4# systemctl daemon-reload
node1/2/3/4# mount /dev/drbd1 /mnt/test-pgsql
node1/2/3/4# ls -l /mnt/test-pgsql
```

PGSQL 리소스 구성

올바르게 동작하는지 확인한다.

```
# DRBD primitive (리소스 이름: pgsql-drbd)
pcs resource create pgsql-drbd ocf:linbit:drbd drbd_resource=resource1 \
  op monitor interval=15s role=Master timeout=30s \
  op monitor interval=30s role=Slave  timeout=30s

# DRBD를 승격 가능(MS)로
pcs resource master pgsql-drbd-ms pgsql-drbd \
  master-max=1 master-node-max=1 clone-max=4 clone-node-max=1 notify=true
```

PGSQL 리소스 구성

올바르게 동작하는지 확인한다.

```
# Filesystem (DRBD 디바이스와 pgdata 디렉토리 일치)
pcs resource create pgsql-fs ocf:heartbeat:Filesystem \
  device="/dev/drbd1" directory="/var/lib/pgsql/data" fstype="xfs"
options="noatime" \
  op monitor interval=20s timeout=40s
```

PGSQL 리소스 구성

올바르게 동작하는지 확인한다.

```
# PostgreSQL (필수 파라미터 환경에 맞게 수정)
pcs resource create pgsql ocf:heartbeat:pgsql \
  pgdata="/var/lib/pgsql/data" \
  psql="/usr/bin/psql" \
  pgctl="/usr/bin/pg_ctl" \
  start_opts="-w" stop_opts="-m fast" \
  op start timeout=60s \
  op stop  timeout=60s \
  op monitor interval=15s timeout=30s
```

PGSQL 리소스 구성

올바르게 동작하는지 확인한다.

```
# VIP (넷마스크는 환경에 맞게. /32도 가능하지만 일반적으로 24 등)
pcs resource create pgsql-vip ocf:heartbeat:IPaddr2 ip=192.168.90.243
cidr_netmask=24 \
  op monitor interval=10s timeout=20s
```

PGSQL 코로케이션 구성

자원 동작 관계를 선언한다.

```
# DRBD가 Master인 노드에서만 파일시스템/DB/VIP가 동작
pcs constraint colocation add pgsql-fs  with pgsql-drbd-ms INFINITY with-
rsc-role=Master
pcs constraint colocation add pgsql     with pgsql-fs        INFINITY
pcs constraint colocation add pgsql-vip with pgsql           INFINITY

# 승격 → FS → DB → VIP 순서 보장
pcs constraint order promote pgsql-drbd-ms then start pgsql-fs
pcs constraint order start   pgsql-fs     then start pgsql
pcs constraint order start   pgsql        then start pgsql-vip
```

PGSQL(설명)

role: 모니터링 시, 더 낮은 수준에서 모니터링 수행. 기본적으로 기본값은 unpromote이다.

1. interval=15s
2. timeout=30s

with-rsc-role: 기본값은 없으며, 특정 역할(role)과 항상 같이 운영이 된다.

```
pcs resource create pgsql-drbd-resource ocf:linbit:drbd drbd_resource=drbd0
op monitor timeout="30" interval="5" role="prmote" op monitor timeout="30"
interval="6" role="unpromote"

pcs constraint colocation add pgsql-mon pgsql-drbd-sync INFINITY with-rsc-
role=promote
```

연습문제

위의 내용은 다시 롤백 후, 재구성.

- 기존의 내용은 절대로 수정하지 않는다.
- 다른 리소스(resource)에도 colocation를 적용을 시도한다.

MARIADB

데이터베이스

2023-05-30

MariaDB

시작전에 모든 노드에 스냅샷 생성하세요.

DRBD SETUP FOR MariaDB

일반 블록 파일 시스템 기반으로 MariaDB를 구성하기 위해서는 DRBD구성이 필요하다. 여기서는 DRBD를 다루지 않는다.

또한, MariaDB는 자체 복제 시스템이 있기 때문에 굳이 DRBD 사용이 필요 없다.

이 교육에서는 MariaDB의 **복제 시스템(Galera)** 구성은 다루지 않는다.

DRBD SETUP FOR MariaDB

```
node1/2/3/4# vi /etc/drbd.d/resource2.res
resource resource2 {
  on node1.example.com {
    device    /dev/drbd2;
    disk      /dev/drbd-pgsql/drbd-mariadb;
    address   192.168.90.110:7789;
    meta-disk internal;
  }
```

DRBD SETUP FOR MariaDB

```
on node2.example.com {
  device    /dev/drbd2;
  disk      /dev/drbd-demo/drbd-mariadb;
  address  192.168.90.120:7789;
  meta-disk internal;
}
```

DRBD SETUP FOR PGSQL

```
on node3.example.com {
  device    /dev/drbd2;
  disk      /dev/drbd-demo/drbd-mariadb;
  address  192.168.90.130:7789;
  meta-disk internal;
}
```

DRBD SETUP FOR PGSQL

```
  on node4.example.com {
    device    /dev/drbd2;
    disk      /dev/drbd-demo/drbd-mariadb;
    address  192.168.90.140:7789;
    meta-disk internal;
  }
}
```

DRBD SETUP FOR MariaDB

```
node1# drbdadm create-md resource2
node1# drbdadm up resource2
node1# drbdadm status resource2
node1# drbdadm primary --force resource2
node1# drbdadm status resource2
node1# lsblk
node1# mkfs.xfs /dev/drbd2
node1# mkdir -p /mnt/drbd-mariadb
```

DRBD SETUP

```
node1# mount /dev/drbd2 /mnt/drbd-mariadb
node1# systemctl daemon-reload
node1# cd /mnt/drbd-mariadb
node1# touch test{1..100}
node1# umount /mnt/drbd-mariadb
node1# drbdadm secondary resource2
```

DRBD SETUP

```
node1# drbdadm up resource2
node1/2/3/4# drbdadm status resource2
node1# drbdadm primary resource2
node1# mkdir -p /mnt/drbd-mariadb
node1# systemctl daemon-reload
node1# mount /dev/drbd1 /mnt/drbd-mariadb
node1# ls -l /mnt/drbd-mariadb
```

MariaDB

```
node1# dnf install mariadb-server
node1# systemctl start mariadb
node1# systemctl stop mariadb
node1# vi /etc/my.cnf.d/mariadb-server.cnf
bind-address=192.168.90.220
```

MariaDB

```
node1# pcs resource create MariaDB_VIP ocf:heartbeat:IPaddr2
ip=192.168.90.230 cidr_netmask=24  op monitor interval=30s

node1# pcs resource create MariaDB_INSTANCE service:mariadb  op start
timeout=59s  op stop timeout=60s op monitor interval=20s timeout=30s

node1# pcs constraint order MariaDB_VIP then MariaDB_New

node1# pcs constraint colocation add MariaDB_VIP with MariaDB_INSTANCE
INFINITY        ①          ②

node1# pcs resource group add mariadb MariaDB_VIP MariaDB_INSTANCE
```

2023-05-30

Mariadb 연습문제

node1/2/3/4번에 mariadb서버를 기존과 동일하게 VIP 및 systemd 서비스를 구성한다. 모든 작업은 롤백 후 작업을 수행한다.

1. VIP는 192.168.90.210/24 ## 다른 아이피 주소 사용 가능.
2. DRBD디스크 구성. ## 현재 랩에서는 D/B 디스크를 공유하지 않음.
3. 서비스 실행(systemd 혹은 service).
4. DRBD를 구현된 부분을 LVM2로 변경 한다.
5. 시간이 있으면 mariadb-replication서비스로 저장소 구성 후, 다중 서비스 구성을 한다.

2023-05-30

자원 이동

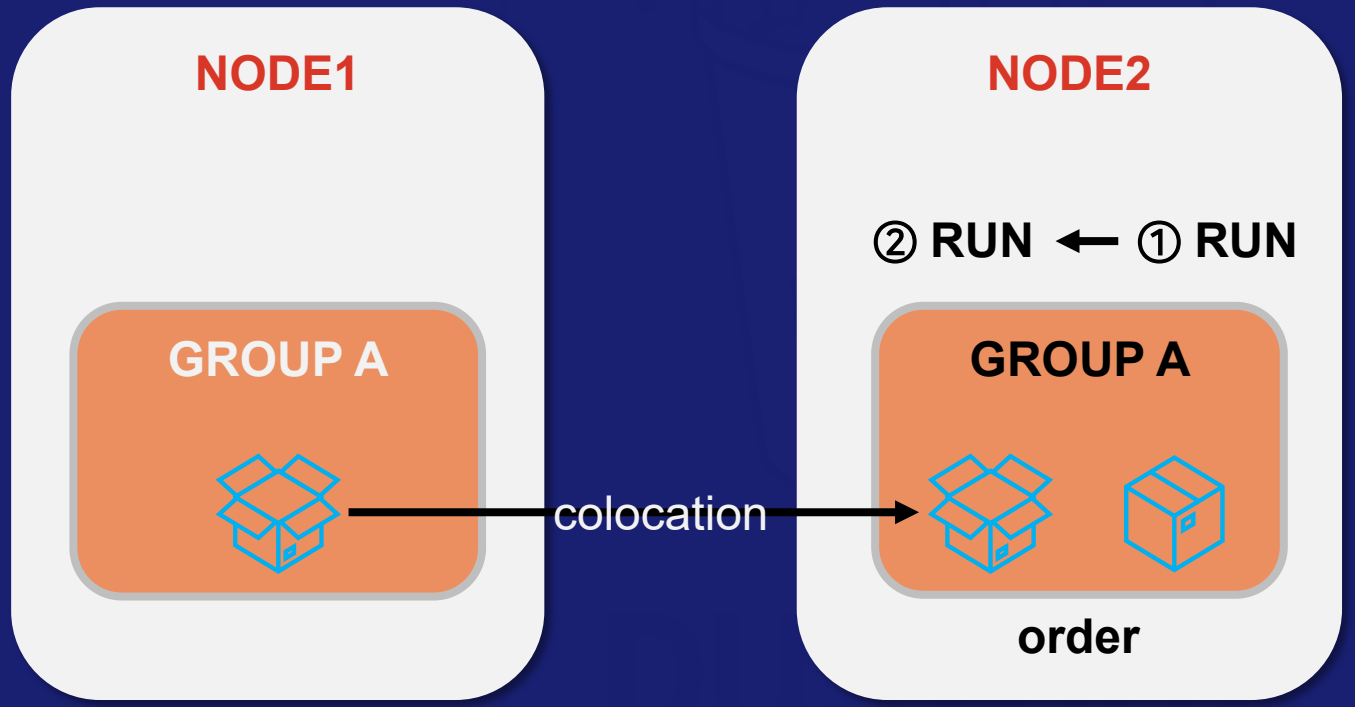
자원 배치 및 재배치

RESOURCE MOVE

자원을 다른 노드로 이동하는 방법은 아래와 같다.

1. `pcs fence nodeX.example.com`
2. `pcs resource move ha_lvm_group node1.example.com`
3. `pcs resource move-with-constraint ha_lvm_group node4.example.com lifetime=1M`
4. `pcs constraint location ha_lvm_group prefers node2.example.com=500`

COLOCATION/ORDER



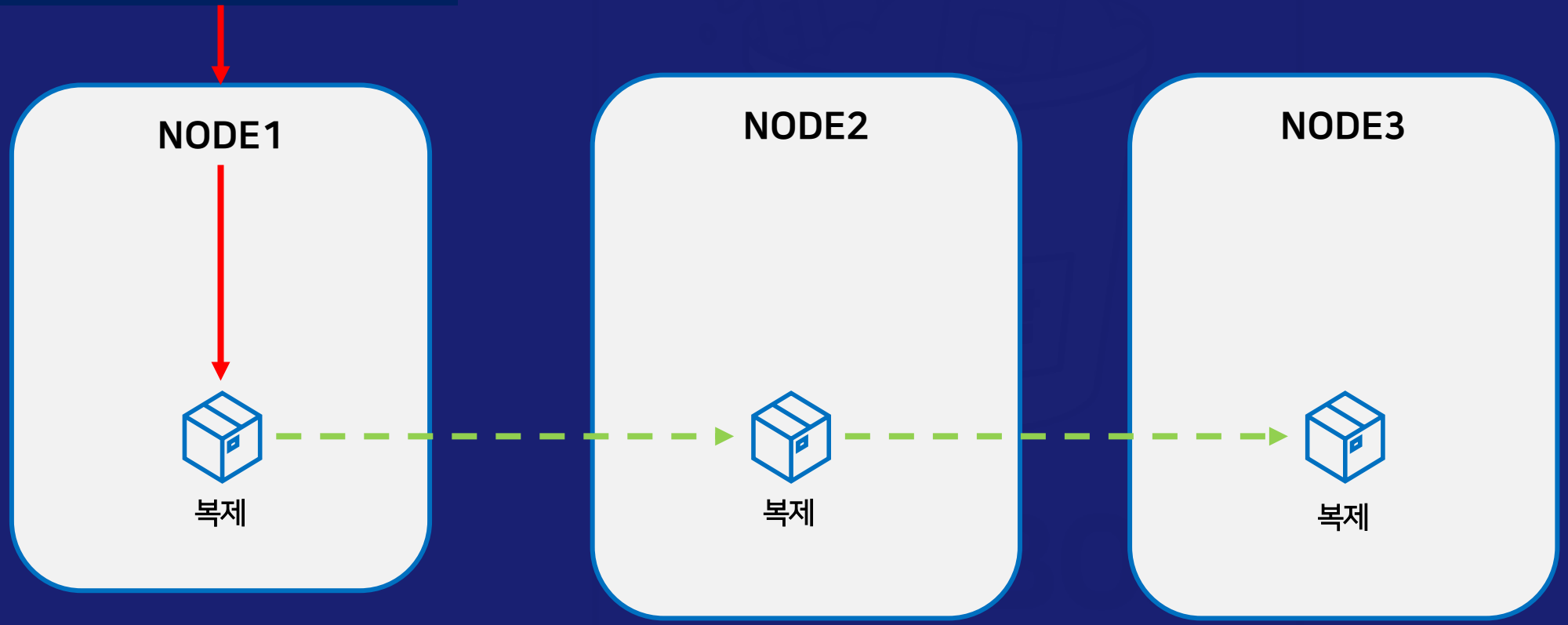
colocation
리소스 A가 움직이면 B가 같이 이동함

order
리소스의 실행 순서

1. 아이피 할당
2. 데이터 베이스 서비스 시작(bind-address(vip))

clone

```
# pcs resource clone
```



COLOCATION/ORDER/CLONE/PREFERS

```
node1# pcs constraint order MariaDB_VIP then MariaDB_New

node1# pcs constraint colocation add MariaDB_VIP with MariaDB_INSTANCE
INFINITY

node1# pcs resource group add mariadb MariaDB_VIP MariaDB_INSTANCE

node1# pcs constraint order TOMCAT_VIP then TOMCAT_SERVICE score=INFINITY
symmetrical=true id=order-TOMCAT_VIP-then-TOMCAT_SERVICE

node1# pcs constraint location TOMCAT_SERVICE prefers node2.example.com

node1# pcs resource clone gfs2_vg interleave=true
```

TWO NODE

호스트 나누기

2023-05-30

TWO NODE

시작전에 모든 노드에 스냅샷 생성하세요.

TWO NODE 설명

드물기는 하지만, 두 개의 노드(하드웨어)를 가지고 구성하는 경우도 있다. 그럴때는 TWO NODE 옵션을 통해서 H/A 시스템을 구성한다.

제일 기본적이고 기초적인 구성이며, 구성도 어렵지 않다. 다만, TWO NODE는 뇌 쪼개기(Brain Split)가 발생하기 때문에, 별도의 TWO NODE 옵션을 활성화 하여, 해당 문제를 해결해야 한다.

TWO NODE 클러스터 셋팅

```
node1# corosync-quorumtool | grep Flags
node1# pcs quorum update auto_tie_breaker=1 last_man_standing=1 last_man_standin
g_window=10000 wait_for_all=1
node1# pcs property set stonith-enabled=false
node1# pcs property set no-quorum-policy=ignore
```

TWO NODE 설정 확인

```
node1# nano /etc/corosync/corosync.conf
quorum {
    provider: Corosync_votequorum
    auto_tie_breaker: 1
    last_man_standing: 1
    last_man_standing_window: 10000
    wait_for_all: 1
    two_node: 1
}
```

DC를 구성하기 위한 추가 표

마지막 남은 노드에게 모든 클러스터 리소스를 전달

시간내에 응답이 없으면 last man standing으로 전환

모든 노드가 정상적으로 통신 및 서비스가 가능한 상태. 부팅 시, 서로 fencing방지.

노드를 두개만 운영하는 경우

TWO NODE

1. `two_node:` **물리적 노드**를 두개만 운영하는 경우.
2. `wait_for_all:` 모든 노드가 정상적으로 통신 및 서비스가 가능한 상태. 부팅 시, 서로 fencing방지.
3. `auto_tie_breaker:` 50:50, 충돌 방지. 노드 두 대가 되면, 서로 D/C를 가져가기 위해서 경쟁. 이 때 서로 Fencing를 시도를 하면서, 무한 리-부팅 루프에 빠지게 됨. 이를 방지하기 위해서 추가표를 노드에 할당.
4. `lowest|highest:` Node ID를 값에 **높음/낮음**에 따라서 순서 정의가 가능.
5. `last_man_standing:` 최종적으로 모든 노드가 비-정상적으로 동작 시, 최종적으로 남아 있는 노드에 자원을 전달 및 할당.

TWO NODE 상태 확인

```
node1# pcs cluster sync
node1# pcs cluster reload corosync
node1# man 5 votequorum
```

TWO NODE 연습문제

클러스터를 초기화 후 다음과 같이 TWO NODE를 구성한다.

1. node1, node2번 기반으로 TWO NODE를 생성 및 구성한다.
2. 구성된 TWO NODE 클러스터에 WEB 및 데이터베이스 서비스를 구성한다.
3. 올바르게 동작하면, VIP를 통해서 서비스 접근이 가능해야 한다.

ACTIVE/ACTIVE

라운드 로빈 서비스

2023-05-30

ACTIVE/ACTIVE

시작전에 모든 노드에 스냅샷 생성하세요.

ACTIVE/ACTIVE

페이스메이커는 기본적으로 active/active를 지원하지는 않는다. 사용하기 위해서는 몇 가지 조건이 필요하다.

1. GFS2와 같은 여러 노드에서 접근 가능한 블록 장치 및 파일 시스템
2. 최소 한 개의 VIP 주소 필요

클러스터 생성

페이스메이커 클러스터를 생성한다.

```
node1# pcs host auth node1.example.com node2.example.com node3.example.com
node4.example.com -u hacluster -p '<PASSWORD>'
node1# pcs cluster setup --name ha-cluster node1.example.com
node2.example.com node3.example.com node4.example.com
node1# pcs cluster start --all
node1# pcs cluster enable --all
node1# pcs property set stonith-enabled=true
node1# pcs property set no-quorum-policy=stop
```

차단 장치 구성

STONITH를 생성 및 구성한다. iSCSI 장치가 없는 경우 구성 후, 아래와 같이 구성한다.

```
node1# pcs stonith create scsi-shooter fence_scsi \
  pcmk_host_list="node1.example.com node2.example.com node3.example.com
node4.example.com" \
  devices=/dev/disk/by-id/wwn-<SHARED_LUN_ID> meta provides=unfencing
```

LVM + lvmlockd 준비 (공유 VG/LV)

LVM2에 ACTIVE/ACTIVE를 위한 구성을 아래와 같이 한다. 모든 노드에 적용.

```
# lvmlockd 활성
sed -ri 's/^#?use_lvmlockd\s*=.*/use_lvmlockd=1/' /etc/lvm/lvm.conf
sed -ri 's/^#?system_id_source\s*=.*/system_id_source="uname"/'
/etc/lvm/lvm.conf
systemctl enable --now lvmlockd
```

LVM + lvmlockd 준비 (공유 VG/LV)

아래 명령어는 node1번에서 진행한다.

```
# 최초 한 노드에서만 PV/VG/LV 생성
# (공유 LUN 디바이스 경로는 환경에 맞게 변경: /dev/disk/by-id/wwn-...)
pvcreate /dev/disk/by-id/wwn-<SHARED_LUN_ID>
vgcreate --shared vg_shared /dev/disk/by-id/wwn-<SHARED_LUN_ID>
lvcreate -L 40G -n data vg_shared

# GFS2 포맷 (lock_dlm, 저널 수 = 동시 마운트 노드 수)
mkfs.gfs2 -p lock_dlm -j 4 -t ha-cluster:fsdata /dev/vg_shared/data
mkdir -p /srv/gfs2
```

자원 생성

페이스메이커에서 사용할 자원을 생성 및 구성한다.

```
# 4-1) DLM (controld) clone
pcs resource create dlm ocf:pacemaker:controld op monitor interval=30s on-
fail=fence
pcs resource clone dlm interleave=true

# 4-2) lvmlockd clone
pcs resource create lvmlockd ocf:heartbeat:lvmlockd op monitor interval=30s
pcs resource clone lvmlockd interleave=true
```

자원 생성

페이스메이커에서 사용할 자원을 생성 및 구성한다.

```
# 4-3) GFS2 Filesystem clone
pcs resource create gfs2_fs Filesystem \
  device="/dev/vg_shared/data" directory="/srv/gfs2" fstype="gfs2" \
  options="noatime" op monitor interval=20s on-fail=fence
pcs resource clone gfs2_fs interleave=true

# 4-4) 예시 앱(클론) - haproxy (stateless)
pcs resource create haproxy systemd:haproxy op monitor interval=15s
pcs resource clone haproxy interleave=true
```

의존성(COLOCATION)

자원의 실행 순서를 구성한다. node1번에서 실행한다.

```
# 순서: dlm → lvmlockd → gfs2_fs → haproxy
pcs constraint order start dlm-clone then lvmlockd-clone kind=Mandatory
pcs constraint order start lvmlockd-clone then gfs2_fs-clone kind=Mandatory
pcs constraint order start gfs2_fs-clone then haproxy-clone kind=Optional

# 함께 배치(콜로케이션) – GFS2는 DLM/lvmlockd가 있는 노드에서:
pcs constraint colocation add lvmlockd-clone with dlm-clone INFINITY
pcs constraint colocation add gfs2_fs-clone with lvmlockd-clone INFINITY
pcs constraint colocation add haproxy-clone with gfs2_fs-clone 100
```

상태확인

최종적으로 상태를 확인한다. node1번에서 실행.

```
pcs status
pcs resource show
pcs constraint
```

종합문제

WORDPRESS + MARAIDB

종합문제

가상머신 3대 혹은 4대 기반으로 페이스메이커를 구성한다.

클러스터 이름은 "wordpress-srv"로 설정한다.

- 클러스터 관리를 위한 operator계정을 하나 생성한다.
- 각각 노드에 파일 기반으로 클러스터에서 발생하는 이벤트를 기록하는 pcs-alert이름의 alert를 구성 및 설치한다.
- 파일이 저장되는 위치는 /var/lib/pacemaker/pcs-alert.log라는 이름으로 저장한다.
- 스크립트는 적절한 위치에 구성 및 배포한다.

LVM2스토리지를 구성한다.

- 웹 서버에서 사용하는 LVM의 이름은 wp-vg-www, LV의 이름은 wp-lv-www로 한다.
- 데이터베이스는 wp-vg-db, wp-lv-db라는 이름으로 LVM를 사용한다.
- 해당 VG는 다른 클러스터에서도 접근이 가능하도록 구성한다.
- 각 노드가 파일 시스템은 접근이 가능하도록 구성한다.

종합문제

모든 리소스는 적절한 순서로 리소스 실행 및 구성한다.

- 모든 노드에서 리소스 정보가 배포 되어야 한다.

데이터베이스 및 웹 서버

- 데이터베이스는 반드시 /var/lib/mysql에서 정보를 불러와야 한다.
- 웹 서버는 /var/www/html/에 접근이 가능해야 한다.
- 데이터베이스는 192.168.90.250 아이피 주소를 가지고 실행한다.
- 웹 서버는 192.168.90.210 아이피 주소를 가지고 실행한다.

아래 내용은 공통 사항으로 모든 노드에 적용이 된다.

- 방화벽 및 보안 설정은 사용하지 않아도 상관 없다.
- 모든 설정 내용은 재시작 이후에도 올바르게 동작해야 한다.