



쿠버네티스에서 longhorn 에 대해 자세히 알려줘봐

좋아요. Longhorn를 “쿠버네티스 네이티브 분산 블록 스토리지(오픈소스, Rancher/SUSE)” 관점에서 정리해드릴게요. 운영에 바로 쓰실 수 있게 개념→설치/사용→스냅샷/백업→RWX→스케줄링/라벨→트러블슈팅 순서로 깔끔하게 갑니다. (사용 환경이 RHEL 계열/ArgoCD/Helm인 것도 감안했어요.)

1) Longhorn 한눈에 보기

- **무엇:** 각 워커 노드의 디스크/디렉터리를 묶어 **복제(replica)** 형태로 **블록 볼륨**을 제공합니다. Pod에는 **CSI**로 PVC를 블록 디바이스(+파일시스템)로 붙입니다.
- **구성 요소(핵심)**
 - **longhorn-manager:** 컨트롤 플레인(스케줄링/상태관리)
 - **instance-manager:** 엔진/리플리카 프로세스 런처
 - **engine/replica:** 볼륨 I/O(엔진)와 데이터 복제(리플리카)
 - **longhorn-csi-plugin:** CSI 드라이버(컨트롤러/노드)
 - **share-manager(옵션):** NFS 서버를 볼륨별로 띄워 **RWX** 제공
- **특징**
 - 노드/디스크 단위 장애에 **강한 내구성(복제 N개)**
 - **스냅샷/백업(S3/NFS)** 내장, DR/스탠바이 볼륨 지원
 - ****데이터 지역성(dataLocality)****로 “볼륨 있는 노드에 Pod” 선호
 - 완전 오픈소스, 쿠버네티스 CRD만으로 관리 가능(UI도 제공)

2) 설치(Helm) & 기본 확인

전제: 모든 워커 노드에 **open-iscsi(iscsiadm/iscsid)**, **mount** 유틸 설치. (Longhorn은 기본적으로 iSCSI로 볼륨을 어태치합니다. RHEL이라면 **iscsi-initiator-utils** 패키지 확인.)

```
bash

# 네임스페이스/리포지토리
kubectl create ns longhorn-system
helm repo add longhorn https://charts.longhorn.io
helm repo update

# 설치(기본)
helm install longhorn longhorn/longhorn -n longhorn-system \
  --set defaultSettings.createDefaultDiskLabeledNodes=false

# 컴포넌트 확인
kubectl -n longhorn-system get pods -o wide

# CSI 드라이버 등록 확인
```

```
kubectl get csinodes -o jsonpath="{range .items[*]}{.metadata.name}{'\t'}{range .spec.drivers[*]}{.name}{','}{end}{'\n'}{end}" | grep driver.longhorn.io
```

Tip: RHEL/SELinux 환경이면 iscsid 동작/포트(3260/TCP) 및 SELinux 컨텍스트가 문제 되지 않는지 먼저 체크하세요.

3) StorageClass/PVC/Pod 빠른 사용 예

(1) RWO 기본 StorageClass

```
yaml

# longhorn-sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: longhorn
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" # 기본 SC로
provisioner: driver.longhorn.io
allowVolumeExpansion: true
reclaimPolicy: Delete
parameters:
  numberOfReplicas: "3" # 복제 수 (단일노드 테스트면 1)
  staleReplicaTimeout: "30" # 분 단위, 리플리카 stale 판단
  dataLocality: "best-effort" # best-effort | disabled | strict-local(버전에 따라)
  fsType: "ext4"
# 선택적: diskSelector, nodeSelector, encrypted, snapshotDataIntegrity 등
```

```
bash

kubectl apply -f longhorn-sc.yaml
kubectl get sc
```

(2) PVC + 간단한 Pod

```
yaml

# pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: demo-pvc
spec:
  accessModes: ["ReadWriteOnce"]
  storageClassName: longhorn
  resources:
    requests:
      storage: 10Gi
---
# pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
spec:
  containers:
    - name: app
      image: busybox
      command: ["sh", "-c", "while true; do date >> /data/out.txt; sleep 5; done"]
```

```

volumeMounts:
- name: data
  mountPath: /data
volumes:
- name: data
  persistentVolumeClaim:
    claimName: demo-pvc

```

```
bash
```

```

kubectl apply -f pvc.yaml
kubectl apply -f pod.yaml
kubectl exec -it demo-pod -- tail -f /data/out.txt

```

4) RWX(여러 Pod가 동시에 쓰기) — share-manager 방식

Longhorn은 볼륨별 **share-manager**(내장 NFS 서버)를 띄워 **ReadWriteMany**를 제공합니다.

```
yaml
```

```

# RWX용 StorageClass(예)
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: longhorn-rwx
provisioner: driver.longhorn.io
allowVolumeExpansion: true
parameters:
  numberOfReplicas: "3"
# RWX는 내부적으로 share-manager(NFS) 사용
mountOptions:
- nfsvers=4.2      # 환경에 따라 4.1/4.2

```

```
yaml
```

```

# pvc-rwx.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: demo-pvc-rwx
spec:
  accessModes: ["ReadWriteMany"]
  storageClassName: longhorn-rwx
  resources:
    requests:
      storage: 10Gi

```

주의: RWX는 내부 NFS를 쓰므로 네트워크/락 특성상 RWO 대비 성능이 다를 수 있습니다. 고성능 RWX가 절대 요구라면(특히 DB) 설계 재검토가 필요해요.

5) 스냅샷/백업/복구(VolumeSnapshot & S3/NFS)

(1) CSI 스냅샷 클래스/스냅샷

```
yaml
```

```

# volumesnapshotclass.yaml
apiVersion: snapshot.storage.k8s.io/v1

```

```
kind: VolumeSnapshotClass
metadata:
  name: longhorn-snapclass
driver: driver.longhorn.io
deletionPolicy: Delete
```

yaml

```
# snapshot.yaml
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: demo-snap
spec:
  volumeSnapshotClassName: longhorn-snapclass
  source:
    persistentVolumeClaimName: demo-pvc
```

롤백하려면 VolumeSnapshot을 소스로 새 PVC를 만들 수 있습니다.

(2) 백업(오프클러스터)

Longhorn UI/CRD에서 **S3(호환 저장소)**나 **NFS**를 백업 대상(Backup Target)으로 지정 → **Recurring Job**으로 주기 백업 가능. DR/Standby 볼륨으로 장애 지역 복구 시나리오도 깔끔합니다.

6) 스케줄링/라벨링 포인트

- **replica 스케줄링**: 노드/디스크 태그(tag) 기반으로 특정 풀에만 생성 가능
- **dataLocality**:
 - **best-effort**: 가능하면 볼륨 리플리카가 있는 노드에 Pod 스케줄
 - **strict-local**(버전 종속): 반드시 동일 노드(조건 불만족 시 Pending 가능)
- 노드/디스크 용량 예약(over-provision 완화), 복제 수 조정으로 안정성/성능 균형 맞추기

7) 운영에서 자주 겪는 이슈 & 점검 체크리스트

(A) “어태치/디태치”가 멈춘다

- 각 워커 노드에 **open-iscsi** 가동 여부 확인:
 - 패키지: **iscsiadm**, 서비스: **iscsid**
 - RHEL: **sudo systemctl status iscsid**
- multipath 충돌/설정 확인(환경에 따라 필요 시 무력화/조정)
- 노드에 **longhorn-csi-plugin DaemonSet**이 정상인지:

bash

```
kubectl -n longhorn-system get ds/longhorn-csi-plugin -o wide
kubectl -n longhorn-system logs ds/longhorn-csi-plugin -c csi-node-driver-registrar
```

- 문제 볼륨 상태:

```
bash

kubectl get volumes.longhorn.io
kubectl -n longhorn-system get pods -l longhorn.io/component=instance-manager -o wide
```

(B) “driver ... not found in the list of registered CSI drivers”

- 클러스터에 등록된 드라이버 확인:

```
bash

kubectl get csinodes -o jsonpath="{range .items[*]}{.metadata.name}{'\t'}{range .spec.drivers[*]}{.name}{','}{end}{'\n'}{end}"
```

- longhorn-csi-plugin**(컨트롤러/노드) 파드 모두 Running인지 확인
- 특정 노드에서만 실패하면, 그 노드의 **iscsid/커널 모듈/보안정책(SELinux)** 확인

(C) 성능/지연

- 복제 수를 3→2(또는 1, 단일 노드 테스트)로 낮추면 쓰기 지연 감소
- instance-manager/engine/replica에 **리소스 요청/제한** 부여
- RWX는 내부 NFS 특성상 RWO보다 느릴 수 있으니 워크로드 성격 점검

8) ArgoCD로 배포할 때(Helm 차트)

```
yaml

# longhorn-app.yaml (예시)
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: longhorn
  namespace: argocd
spec:
  project: default
  destination:
    namespace: longhorn-system
    server: https://kubernetes.default.svc
  source:
    repoURL: https://charts.longhorn.io
    chart: longhorn
    targetRevision: 1.7.*          # 조직 표준에 맞게 고정 권장
  helm:
    values: |
      defaultSettings:
        defaultDataPath: "/var/lib/longhorn"
        createDefaultDiskLabeledNodes: false
      persistence:
        defaultClass: true
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

9) 함께 쓰기 좋은 설정/팁

- 기본 SC 전환: Longhorn을 기본으로 쓰려면 SC에 `is-default-class: "true"`
- 복구 훈련: 주 1회 백업 & 복구 리허설(스냅샷만으로 안심 금지)
- 라벨링: I/O 헤비한 앱은 특정 노드/디스크 태그로 격리
- 모니터링: Longhorn UI + `volumes.longhorn.io` 상태로 불건전 리플리카 조기 탐지

필요하시면,

- 귀사(RHEL) 표준에 맞춘 필수 패키지 점검 스크립트(`iscsid`/모듈/SELinux)
- RWX 워크로드 검증용 매니페스트(두 개의 Pod가 같은 파일에 동시 쓰기/잠금 확인)
- S3 백업 타겟 + Recurring Job 세트
를 바로 만들어 드릴게요.

지금 상황에 딱 맞춰보면, 예전에 겪으셨던 `driver nfs.csi.k8s.io not found`류 이슈는 Longhorn RWX가 내부 NFS를 쓴다는 점과 맞물려 혼동이 오기 쉬워요. 우선 `longhorn-csi-plugin Running + csinodes에 driver.longhorn.io 등록`을 확인하고, 각 노드의 `iscsid` 상태부터 점검해보시는 게 가장 빠릅니다. 원하시면 현재 클러스터 상태(관련 명령 몇 줄) 공유 주시면, 결과 기준으로 바로 원인 좁혀드릴게요.



오늘 starrocks를 helm으로 해서 aroCD를 이용해서 배포하는데, 기존 노드에서 제거하고 새로운 노드에 배포를 했거든(노드 라벨링 다르게 해서). 그런데 cn노드가 계속 pending이 되면서 pod가 스케줄링이 되지 않더라고. 나중에 문제점을 알았는데, longhorn에서 volume이 이전 노드 정보를 가지고 있었던것 같아. longhorn관리자 페이지에서 volume을 지우니깐 pending이 풀리더라고.

아—그 상황, 충분히 그럴 수 있어요.

Longhorn가 **PV에 NodeAffinity(노드 친화도)**를 심어둔 상태였고, 그게 **이전 노드**를 가리키고 있어서 스케줄러가 “새 노드엔 못 붙임” 하고 CN 파드를 계속 Pending 시킨 겁니다. UI에서 볼륨 지우니 풀린 건, 그 NodeAffinity 제약 자체가 사라져서예요.

핵심만 정리해드릴게요.

왜 그랬나 (원인 구조)

- Longhorn는 볼륨(리플리카)의 위치/내구성을 고려해서 `PV.spec.nodeAffinity`를 설정할 수 있습니다.
- 이렇게 되면 K8s 스케줄러는 해당 PV가 허용하는 노드에서만 파드를 스케줄링하려고 합니다.
- 노드 라벨을 바꿔서 워크로드를 다른 노드로 옮겨도, **PV의 NodeAffinity가 옛 노드에 묶여 있으면 0/N nodes had volume node affinity conflict** 류로 계속 Pending이 납니다.
(메시지는 환경마다 조금 다르게 보이지만 취지는 같습니다.)

다음 번엔 볼륨 삭제 없이 해결하는 법 (런북)

1. 워크로드 스케일 다운 / 볼륨 Detach

- ArgoCD면 해당 Deployment/STS를 0으로 낮춰서 볼륨을 **Detach** 상태로 만드세요.

- `kubectl get volumes.longhorn.io`로 대상 볼륨 상태가 **detached**인지 확인.

2. 이전 노드에서 리플리카 내보내기

- Longhorn UI에서 해당 **이전 노드 Scheduling Disable** 또는 **Evict Replicas** 실행.
- 리플리카가 **새 노드로 재빌드**될 때까지 대기(상태 Healthy 확인).

3. PV NodeAffinity 갱신

- Longhorn UI의 볼륨 상세 → **“Update PV Node Affinity”** (버전별로 메뉴명 약간 다름).
이걸 누르면 PV가 가리키는 허용 노드 목록이 **현재 건강한 리플리카가 있는 노드**로 갱신됩니다.
- UI가 여의치 않으면(응급) 아래처럼 **임시로 NodeAffinity 제거**도 가능합니다:

```
bash

kubectl get pv | grep <볼륨이름-혹은-스토리지클래스>
kubectl patch pv <PV_NAME> --type merge -p '{"spec":{"nodeAffinity":null}}'
```

단, 이는 **임시 우회**에 가깝습니다. Longhorn 쪽 상태와 일치시키려면 결국 UI의 “Update PV Node Affinity” 경로로 정리해두는 게 안전합니다.

4. 워크로드 스케일 업

- Deployment/STS를 원래 replicas로 되돌리면, 스케줄러가 **새 노드**로 파드를 올리면서 볼륨을 정상 어태치합니다.

확인/디버깅 포인트

- 파드 이벤트:

```
bash

kubectl describe pod <cn-pod> | sed -n '/Events:/, $p'
```

여기서 **node affinity/selector, volume node affinity, selected-node** 힌트 등을 확인.

- PV의 NodeAffinity:

```
bash

kubectl get pv <PV_NAME> -o yaml | sed -n '/nodeAffinity:/,/^ *[^ ]/p'
```

- PVC에 스케줄 힌트가 남아 있을 때(드뭄):

```
bash

kubectl get pvc <PVC_NAME> -o
jsonpath='{.metadata.annotations.volume\.kubernetes\.io/selected-node}'
```

필요 시 패치로 지워 재스케줄을 유도할 수 있습니다.

- Longhorn 컴포넌트/노드 상태:

bash

```
kubectl -n longhorn-system get pods -o wide
kubectl get csinodes -o jsonpath="{range .items[*]}{.metadata.name}{'\t'}{range .spec.drivers[*]}{.name}{','}{end}{'\n'}{end}" | grep driver.longhorn.io
```

노드 교체/라벨 변경 시 체크리스트 (재발 방지)

- [사전] 대상 워크로드 스케일 다운 → 볼륨 Detach
- Longhorn에서 이전 노드 Scheduling Disable → Replica Evict → Rebuild 완료 대기
- 볼륨 상태 Healthy 후 Update PV Node Affinity
- 그다음 스케일 업 (ArgoCD 자동화라면 PR/Sync 전에 위 과정을 운영 런북으로 분리)
- 데이터 지역성(dataLocality), Replica 수, Node/디스크 태그가 새 배치 전략과 맞는지 재점검

원하시면, 위 과정을 **ArgoCD PreSync Hook**(예: Job로 `kubectl patch/Longhorn CR 조작`)이나, **운영용 스크립트**로 묶어드릴게요.

지금 CN가 Pending이던 네임스페이스/리소스명을 알려주시면, **해당 PV/PVC를 딱 짚어** NodeAffinity 상태부터 함께 확인해보죠.



운영용 스크립트로 한번 쭉봐.

좋습니다. “노드 라벨 이동/교체 시 Longhorn PV의 nodeAffinity 때문에 파드가 Pending되는 걸 방지/해결”하는 **운영용 스크립트**를 드릴게요.

컨셉은 안전·보수적으로: **스케일다운 → PV/ PVC 패치 → (선택) 리플리카 에빅션 → 스케일업** 순서입니다.

전제

- `kubectl` 컨텍스트는 대상 클러스터로 설정됨
- Longhorn가 기본 StorageClass이거나, 대상 PVC의 SC가 `driver.longhorn.io`
- ArgoCD 자동 동기화 중이면, 충돌 피하려고 잠깐 **Manual**로 두는 걸 추천

lh-move.sh

bash

```
#!/usr/bin/env bash
set -euo pipefail
```

```
# -----
# Longhorn PV nodeAffinity 정리 & 워크로드 재배포 도우미
# - 대상 네임스페이스와 (선택) 레이블 셀렉터 기준으로
#   Deployment/StatefulSet을 스케일다운 → PV/PVC 패치 → (옵션) 리플리카 에빅션 → 스케일업
# -----
```



```
# 기본값
NAMESPACE=""
SELECTOR=""
DRY_RUN=false
DO_EVICT=false
EVICT_NODE="" # --evict-node <nodeName> 인 경우, 해당 Longhorn 노드에서 리플리카 에백션
WAIT_TIMEOUT=600 # 각 단계 대기(초)
LOG_TS() { date '+%F %T'; }
log() { echo "[$(LOG_TS)] $*"; }

usage() {
    cat <<'EOF'
Usage:
    lh-move.sh --ns <namespace> [--selector <labelSelector>] [--dry-run]
        [--evict-node <nodeName>] [--timeout <sec>]

What it does:
    1) (선택) selector에 매칭되는 Deployment/StatefulSet 스케일을 0으로 (스케줄러/볼륨 detach 유도)
    2) namespace 내 Longhorn PVC → PV를 찾아 PV.spec.nodeAffinity 제거, PVC의 selected-node 어노테이션
    제거
    3) (선택) 특정 Longhorn 노드에서 리플리카 에백션 요청 (새 노드로 재빌드 유도)
    4) 원래 replicas로 스케일업

주요 옵션:
    --ns|-n          대상 네임스페이스 (필수)
    --selector|-l     레이블 셀렉터(예: app=starrocks,component=cn). 주면 해당 워크로드만 스케일 조정
    --dry-run        실제 패치/스케일 없이 어떤 리소스를 건드릴지 출력만
    --evict-node      리플리카를 비우고 싶은 Longhorn 노드명 (선택)
    --timeout         각 대기 단계 타임아웃 초 (기본 600)

Note:
    - ArgoCD 자동 동기화 중이면, 잠시 수동으로 두세요(충돌 방지).
    - 이 스크립트는 StorageClass의 provisioner가 driver.longhorn.io 인 PVC만 대상으로 삼습니다.
EOF
}

# ----- 인자 파싱 -----
while [[ $# -gt 0 ]]; do
    case "$1" in
        --ns|-n) NAMESPACE="$2"; shift 2;;
        --selector|-l) SELECTOR="$2"; shift 2;;
        --dry-run) DRY_RUN=true; shift;;
        --evict-node) DO_EVICT=true; EVICT_NODE="$2"; shift 2;;
        --timeout) WAIT_TIMEOUT="$2"; shift 2;;
        -h|--help) usage; exit 0;;
        *) echo "Unknown arg: $1"; usage; exit 1;;
    esac
done

[[ -z "$NAMESPACE" ]] && { echo "ERROR: --ns <namespace> 필수"; exit 1; }

# ----- 헬퍼 -----
kc() { kubectl "$@"; }

wait_no_pods() {
    local ns="$1" sel="$2" timeout="$3"
    local start end
    start=$(date +%s)
    while true; do
        local cnt
        if [[ -n "$sel" ]]; then
            cnt=$(kc -n "$ns" get pods -l "$sel" --no-headers 2>/dev/null | wc -l || true)
        else
            # selector 없으면, 단지 볼륨 detach 대기만 의미 → 파드 체크는 스킵
            cnt=0
        fi
        if [[ "$cnt" -eq 0 ]]; then
            log "대상 파드 없음(또는 selector 미지정) → 진행"
            break
        fi
    done
}
```

```

fi
end=$(date +%s)
if (( end - start > timeout )); then
    log "WARN: 파드 종료 대기 타임아웃(${timeout}s). 계속 진행."
    break
fi
sleep 3
done
}

scale_workloads() {
    local ns="$1" sel="$2" replicas="$3"
    local kind
    for kind in deployment statefulset; do
        local items=()
        if [[ -n "$sel" ]]; then
            mapfile -t items < <(kc -n "$ns" get "$kind" -l "$sel" -o jsonpath='{range .items[*]}{.metadata.name}{ "\n" }{end}')
        else
            # selector 없으면 스케일 조정 스킵(명시적 의도 필요)
            continue
        fi
        for name in "${items[@]"; do
            [[ -z "$name" ]] && continue
            log "스케일 $kind/$name -> $replicas"
            $DRY_RUN || kc -n "$ns" scale "$kind/$name" --replicas="$replicas"
        done
    done
}

list_longhorn_pvcs() {
    local ns="$1"
    # provisioner가 driver.longhorn.io 인 PVC만 추출
    kc -n "$ns" get pvc -o json \
    | jq -r '.items[]
    | select(.metadata.annotations["volume.beta.kubernetes.io/storage-
provisioner"]=="driver.longhorn.io"
    or .spec.storageClassName!=null)
    | select(.spec.storageClassName!=null)
    | . as $pvc
    | $pvc.metadata.name + "\t" + ($pvc.spec.storageClassName//"" ) ' \
    | while IFS=$'\t' read -r pvc sc; do
        # SC가 longhorn인지 최종 확인
        local prov
        prov=$(kc get sc "$sc" -o jsonpath='{.provisioner}' 2>/dev/null || echo "")
        if [[ "$prov" == "driver.longhorn.io" ]]; then
            echo "$pvc"
        fi
    done
}

pv_from_pvc() {
    local ns="$1" pvc="$2"
    kc -n "$ns" get pvc "$pvc" -o jsonpath='{.spec.volumeName}'
}

patch_pv_clear_node_affinity() {
    local pv="$1"
    log "PV/$pv: spec.nodeAffinity 제거"
    $DRY_RUN || kc patch pv "$pv" --type merge -p '{"spec":{"nodeAffinity":null}}' >/dev/null
}

clear_pvc_selected_node() {
    local ns="$1" pvc="$2"
    local ann="volume.kubernetes.io/selected-node"
    local has
    has=$(kc -n "$ns" get pvc "$pvc" -o json | jq -r --arg a "$ann" '.metadata.annotations[$a] // empty')
    if [[ -n "$has" ]]; then
        log "PVC/$pvc: selected-node 어노테이션 제거"
        $DRY_RUN || kc -n "$ns" annotate pvc "$pvc" "$ann" - >/dev/null
    else
        log "PVC/$pvc: selected-node 어노테이션 없음"
    fi
}

```

```

fi
}

evict_replicas_from_node() {
    local node="$1"
    log "Longhorn 노드 $node: 리플리카 에빅션 요청(spec.evictionRequested=true)"
    # Longhorn Node CRD: nodes.longhorn.io
    if $DRY_RUN; then
        return
    fi
    if ! kc -n longhorn-system get nodes.longhorn.io "$node" >/dev/null 2>&1; then
        log "WARN: nodes.longhorn.io/$node 을(를) 찾을 수 없음. 노드명 확인 필요."
        return
    fi
    kc -n longhorn-system patch nodes.longhorn.io "$node" --type merge -p '{"spec": {"evictionRequested":true}}' >/dev/null
    log "에빅션 진행 중... (재빌드 시간은 데이터 크기/네트워크에 따라 상이)"
}

# ----- 실행 플로우 -----
log "네임스페이스: $NAMESPACE"
[[ -n "$SELECTOR" ]] && log "선택터: $SELECTOR"
$DRY_RUN && log "DRY-RUN 모드: 실제 변경 없음"
$DO_EVICT && log "리플리카 에빅션 노드: $EVICT_NODE"

# 1) 스케일다운 (선택)
if [[ -n "$SELECTOR" ]]; then
    scale_workloads "$NAMESPACE" "$SELECTOR" 0
    wait_no_pods "$NAMESPACE" "$SELECTOR" "$WAIT_TIMEOUT"
else
    log "선택터 미지정: 워크로드 스케일 조정은 건너뛴 (볼륨 detach는 수동/사전 보장 가정)"
fi

# 2) PVC/PV 처리
PVC_LIST=()
mapfile -t PVC_LIST < <(list_longhorn_pvcns "$NAMESPACE")

if [[ ${#PVC_LIST[@]} -eq 0 ]]; then
    log "Longhorn PVC 없음. 종료."
    exit 0
fi

log "대상 PVCs: ${PVC_LIST[*]}"

for pvc in "${PVC_LIST[@]"; do
    pv="$(pv_from_pvc "$NAMESPACE" "$pvc" || true)"
    if [[ -z "$pv" ]]; then
        log "PVC/$pvc: PV 미바인드 (스킵)"
        continue
    fi

    # PVC selected-node 제거 → 스케줄러가 새 노드 재선택 가능
    clear_pvc_selected_node "$NAMESPACE" "$pvc"

    # PV nodeAffinity 제거 → 이전 노드에 묶인 제한 해제
    patch_pv_clear_node_affinity "$pv"
done

# 3) (선택) 특정 노드에서 리플리카 에빅션
if $DO_EVICT; then
    evict_replicas_from_node "$EVICT_NODE"
    log "참고: 에빅션 후 리플리카 상태가 Healthy로 회복되는지 Longhorn UI/CRD로 모니터링 권장."
fi

# 4) 스케일업 (선택)
if [[ -n "$SELECTOR" ]]; then
    scale_workloads "$NAMESPACE" "$SELECTOR" 1
    log "스케일업 완료. 파드 이벤트에서 스케줄링/어태치 확인 권장: kubectl -n $NAMESPACE get po -l \"$SELECTOR\" -w"
fi

```

```
else
  log "선택터 미지정: 스케일업 단계 없음."
fi

log "완료."
```

사용 예시

1) StarRocks CN만 대상으로(레이블 기준)

```
bash

# CN 파드들 레이블이 app=starrocks,component=cn 라고 가정
chmod +x lh-move.sh
./lh-move.sh --ns dl-starrocks --selector 'app=starrocks,component=cn'
```

2) 변경사항만 미리 보기(dry-run)

```
bash

./lh-move.sh --ns dl-starrocks --selector 'app=starrocks,component=cn' --dry-run
```

3) 특정 이전 노드에서 리플리카 비우고 싶을 때(에빅션)

```
bash

# 예: 기존 노드 이름이 worker-old-1
./lh-move.sh --ns dl-starrocks --selector 'app=starrocks,component=cn' --evict-node worker-old-1
```

운영 팁(짧게)

- **ArgoCD:** 자동 동기화(On)면 스크립트 스케일 조정을 되돌릴 수 있어요. 이때는 **수동 모드**로 두고 실행하거나, PR 에 스케일 조정/라벨 변경을 같이 녹여서 한 번에 처리하세요.
- **검증:** 실행 후 `kubectl describe pod`에서 Events에 **node affinity/selected-node** 관련 경고가 사라졌는지 체크.
- **장기적:** Longhorn UI에서 **Update PV Node Affinity**를 눌러 PV와 Longhorn 내부 상태를 정합하게 맞춰두면 더 깔끔합니다(이 스크립트는 안전한 "롤링 복구/우회" 목적).

원하시면, 이 스크립트를 **PreSync/Sync hook**(ArgoCD Job)으로 돌릴 수 있는 매니페스트도 만들어드릴게요. 지금 사용하는 레이블/네임스페이스 알려주시면 거기에 맞춰 바로 세팅해드리죠.