쿠버네티스에서 longhorn 웹 ui 가 잘 뜨지 않아. 잘 되다가 안되다가 그래.왜그런거야?

증상이 “가끔 뜨고, 가끔 안 뜨는” 거면 UI 자체 문제라기보다 **경로 / 프록시 / 엔드포인트 상태가 들쭉날쭉**한 경우가 많아요. Longhorn UI는 **`longhorn-frontend`**(SPA 정적파일) ↔ (포드 내부 Nginx 프록시) ↔ **`longhorn-backend/longhorn-manager`** 로 붙는데, 중간 고리가 흔들리면 UI가 랜덤하게 죽습니다. 아래에서 가장 흔한 원인 → 바로 확인할 체크리스트 순서로 정리했어요.

자주 보던 원인 Top 7

1. **Ingress 재작성/타임아웃 문제**
   - `/` → `/#/` 라우팅, WebSocket/장기요청(로그/이벤트 스트림) 타임아웃 미설정.
   - 증상: 첫 로딩은 되는데 클릭하면 빈 화면/회색 화면, 새로고침하면 간헐적으로 열림.
2. **`longhorn-frontend` 단일 레플리카 + 노드 드레인/재시작**
   - 1개 포드만 떠 있고 그 노드가 바쁘거나 옮겨지면 잠깐씩 502/504.
3. **Service Endpoints가 비었다가 생김**
   - **`longhorn-backend/longhorn-manager`** 재시작 중일 때 엔드포인트 0개 → UI 빈 화면.
4. **리소스 부족(특히 manager)**
   - CPU 압박으로 **`longhorn-manager`**의 Readiness가 흔들리면 API 응답 끊김.
5. **클러스터/노드 DNS 지연**
   - Ingress → Service → Pod 해석이 느려져 간헐적 실패.
6. **네트워크 정책/보안장비가 WebSocket 끊음**
   - L7 장비/프록시가 업그레이드 뒤 기본 타임아웃이 짧아짐.
7. **버전 미스매치(프론트는 새 버전, 백엔드는 예전 버전)**
   - Helm/ArgoCD 롤링 중 절반만 바뀌었을 때 발생.

바로 보는 10분 점검(명령 모음)

네임스페이스는 보통 **`longhorn-system`**입니다.

A. 포드/이벤트 상태

```bash
kubectl get po -n longhorn-system -o wide
kubectl describe deploy longhorn-frontend -n longhorn-system | sed -n '1,200p'
kubectl get events -n longhorn-system --sort-by=.lastTimestamp | tail -n 30
```

B. 서비스와 엔드포인트가 비는지 확인

```
bash

kubectl get svc,endpoints -n longhorn-system | egrep 'longhorn-(front|back)end'
kubectl describe ep longhorn-frontend -n longhorn-system
kubectl describe ep longhorn-backend  -n longhorn-system
```

C. Ingress 라우팅/타임아웃(있다면)

```
bash

kubectl get ing -n longhorn-system -o yaml
```

확인 포인트:

- `/` 경로가 **`longhorn-frontend`** 서비스로 가는지
- 주석(annotations)에 **WebSocket 허용**과 **대기시간 상향** 있는지
  - 예: **`nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"`**
  - 예: **`nginx.ingress.kubernetes.io/websocket-services: longhorn-frontend`**

D. 클러스터 내부에서 직접 프론트엔드/백엔드 헬스 확인

```
bash

# 임시 테스트 파드
kubectl run -n longhorn-system curlbox --image=curlimages/curl:8.10.1 -it --rm -- \
  sh -lc 'apk add --no-cache jq || true; \
          echo "Frontend:";  curl -sSf http://longhorn-frontend.longhorn-
system.svc.cluster.local/ | head -n1; \
          echo "Backend:";   curl -sS  http://longhorn-backend.longhorn-
system.svc.cluster.local/v1 | jq .status 2>/dev/null || true'
```

- Frontend가 200이고 Backend **`/v1`**가 응답하면 UI 경로·프록시는 정상인 편.

E. 매니저 리더/레디니스

```
bash

kubectl get po -n longhorn-system -l longhorn.io/component=manager -o wide
kubectl logs -n longhorn-system -l longhorn.io/component=manager --tail=200 | egrep -i
'error|timeout|leader|health'
```

F. 빠르게 포트포워드로 외부 경유 없이 확인(Ingress 문제 분리)

```
bash

kubectl port-forward -n longhorn-system svc/longhorn-frontend 8080:80
# 브라우저에서 http://localhost:8080
```

- 이 경로로는 항상 잘 뜬다면, Ingress/LB 쪽 설정이 범인인 가능성 ↑

바로 적용 가능한 안정화 팁(안전한 조치부터)

1. **프론트엔드 레플리카 2로 스케일 + 간단한 Anti-Affinity**

```bash
kubectl scale deploy/longhorn-frontend -n longhorn-system --replicas=2
kubectl patch deploy longhorn-frontend -n longhorn-system --type='json' -p='[
  {"op":"add","path":"/spec/template/spec/affinity","value":{
    "podAntiAffinity":{"preferredDuringSchedulingIgnoredDuringExecution":[
      {"weight":100,"podAffinityTerm":{
        "topologyKey":"kubernetes.io/hostname",
        "labelSelector":{"matchLabels":{"app":"longhorn-ui","longhorn.io/component":"frontend"}}
      }}
    ]}
  }}
]'
```

- UI 자체는 상태가 없어서 2개로 돌려도 부담 적습니다.

2. **Ingress에 WebSocket/타임아웃/리라이트 보강**
   (Ingress-NGINX 예시)

```yaml
metadata:
  annotations:
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-buffering: "off"
    nginx.ingress.kubernetes.io/websocket-services: "longhorn-frontend"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - host: longhorn.example.com
    http:
      paths:
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: longhorn-frontend
            port:
              number: 80
```

- 컨트롤러 종류별로 키가 다를 수 있어요(HAProxy/Traefik이면 해당 키로).

3. **리소스 요청치 최소 보장**

   - **`longhorn-manager`**, **`longhorn-backend`**(존재 시), **`longhorn-frontend`**에 CPU/메모리 requests를 작게라도 지정.
   - 노드가 바쁠 때도 레디니스가 덜 흔들립니다.

4. **Service 타입 단순화해서 문제 분리**

   - Ingress 전에 **`ClusterIP`** + 포트포워드로 정상 확인 → Ingress만 재검.

5. **버전 일치**

   - Helm/ArgoCD로 업그레이드 중이었다면 **모든 컴포넌트가 같은 릴리스**인지 확인.
   - 앱 삭제-재설치 대신 **Sync 정책(Prune 포함)으로 롤아웃**이 일반적으로 안전합니다.

빠른 원클릭 진단 스크립트(운영용 초안)

저장: **`lh-ui-diagnose.sh`**, 실행: **`bash lh-ui-diagnose.sh`**

```
bash

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

NS="${NS:-longhorn-system}"

echo "== Pods (wide) =="
kubectl get po -n "$NS" -o wide

echo -e "\n== Services & Endpoints (frontend/backend) =="
kubectl get svc,endpoints -n "$NS" | egrep 'longhorn-(front|back)end' || true
echo -e "\n-- EP(frontend) --"
kubectl describe ep longhorn-frontend  -n "$NS" || true
echo -e "\n-- EP(backend) --"
kubectl describe ep longhorn-backend   -n "$NS" || true

echo -e "\n== Manager pods =="
kubectl get po -n "$NS" -l longhorn.io/component=manager -o wide || true

echo -e "\n== Recent events =="
kubectl get events -n "$NS" --sort-by=.lastTimestamp | tail -n 40 || true

echo -e "\n== Frontend Deploy describe (first 150 lines) =="
kubectl describe deploy longhorn-frontend -n "$NS" | sed -n '1,150p' || true

echo -e "\n== Ingress (if any) =="
kubectl get ing -n "$NS" -o yaml || true

echo -e "\n== In-cluster curl test =="
kubectl run -n "$NS" curlbox --image=curlimages/curl:8.10.1 -it --rm -- \
  sh -lc 'apk add --no-cache jq >/dev/null 2>&1 || true; \
          echo "[frontend]"; curl -sS http://longhorn-frontend.'"$NS"'.svc.cluster.local/ | head
-n1 || true; \
          echo "[backend /v1]"; curl -sS http://longhorn-backend.'"$NS"'.svc.cluster.local/v1 |
jq .status 2>/dev/null || true'
```

- 출력에서 **EP가 0개**로 나오는 순간이 보이면 그 타이밍이 끊김 원인에 가깝습니다.
- Ingress가 있는데 포트포워드는 항상 정상 → Ingress 설정/외부 LB 의심.
- 반대로 내부도 불안정 → manager/backend 리소스/재시작/리더 변경 쪽을 보세요.

마무리 힌트

- ArgoCD로 관리 중이면, Ingress/Deployment의 **requests/annotations**가 값파일에서 덮이는지 확인하세요. Sync 때마다 좋아졌다 나빠지는 패턴이면 Git 값이 원인일 확률이 높습니다.
- 노드 교체/라벨 이동을 자주 하셨다면, **`longhorn-manager`**가 그 타이밍에 재조정하며 잠깐 API가 끊길 수 있어요. 그 구간을 이벤트 타임라인과 맞춰보면 깔끔하게 보입니다.

필요하면 당신 환경(Helm values/Ingress 컨트롤러 종류) 기준으로 **안정화용 values.yaml 패치**까지 바로 만들어드릴게요.

> 브라우저에서 주의요함이라고 느낌표가 떠거든. 수십번 새로고침하면 화면이 렌더링 될때도 있고 그래

말씀해주신 **브라우저의 "주의 요함(느낌표)" 아이콘**은 대체로 TLS/HTTPS 경고일 때 많이 뜹니다. Longhorn UI 자체가 SPA(단일 페이지 앱)이라 HTML은 잘 로드돼도, 백엔드 API 호출이 HTTPS → HTTP(혼합 콘텐츠)로 가거나, 인증서가 유효하지 않으면 저런 경고가 뜨고 화면도 들쭉날쭉합니다.

보통 이런 케이스가 많습니다:

1. **Self-signed 인증서 사용**
   - **`kubectl port-forward`**로 열거나, Ingress에서 자체 서명 인증서를 쓸 때 브라우저가 "신뢰할 수 없음" 띄움.
   - 새로고침하다가 캐시가 섞이면 UI가 한 번씩 뜨기도 하고, 백엔드 호출은 막히기도 함.
2. **Mixed Content (혼합 콘텐츠)**
   - 브라우저에서 **`https://`** 주소로 접속했는데, UI 자바스크립트가 내부적으로 **`http://`** API를 호출.
   - 크롬/엣지에서는 이런 요청을 기본 차단 → UI는 비어 있거나 랜덤하게 동작.
3. **Ingress/프록시 설정 누락**
   - **`nginx.ingress.kubernetes.io/force-ssl-redirect: "true"`** 같은 HTTPS 강제 리다이렉트 미설정.
   - **`longhorn-backend`**로 프록시 패스 할 때 **`/v1`** 경로를 제대로 못 넘김.
4. **인증서 체인 불완전**
   - 사내 CA 발급 인증서인데 중간 CA 체인을 Ingress에 안 넣은 경우.
   - 일부 브라우저/OS는 그냥 막아버리고, 일부는 허용 → "되다 안 되다" 현상.

확인해보실 포인트

```
bash

kubectl get ing -n longhorn-system -o yaml
```

- **`tls:`** 섹션이 있는지
- **`secretName:`**이 신뢰할 수 있는 인증서인지

```
bash

kubectl describe svc longhorn-frontend -n longhorn-system
kubectl describe svc longhorn-backend  -n longhorn-system
```

- Ingress가 올바른 Service 포트(80 → 80)로 연결되는지

빠른 실험

1. **포트포워드로 접속**

```
bash

kubectl port-forward -n longhorn-system svc/longhorn-frontend 8080:80
```

브라우저에서 **`http://localhost:8080`** 접속했을 때는 경고가 안 뜨고 항상 잘 열리면 → Ingress/TLS 설정 문제 확정.

2. **브라우저 개발자 도구 → Console/Network 탭**
   - **`Mixed Content: The page at 'https://...' was loaded over HTTPS, but requested an insecure resource 'http://...'`** 이런 메시지가 나오면 → 혼합 콘텐츠 문제.

해결 방향

- 테스트 용도: 그냥 HTTP로 열어보세요 (port-forward).
- 운영/공유 환경:
  - Ingress에 **정상 TLS 인증서(사내 CA or Let's Encrypt)** 적용.
  - **`nginx.ingress.kubernetes.io/backend-protocol: "HTTP"`** / **`proxy-read-timeout`** / **`websocket-services`** 같은 어노테이션도 확인.
  - Helm values에서 **`service.ui.type: ClusterIP`** + Ingress를 안전하게 붙이는 게 권장 경로.

👉 혹시 지금 Longhorn UI를 Ingress로 https 붙여서 접속 중이세요, 아니면 **`kubectl port-forward`**나 NodePort로 바로 열고 계세요?

> longhorn cli도 있나?

네, Longhorn도 CLI가 있습니다. 다만 독립적인 "longhorn" 바이너리라기보다는, 두 가지 경로가 있어요:

---

1. `longhornctl` (커뮤니티 도구)

- Rancher 쪽에서 만든 **Longhorn CLI(베타 수준)** 이 있었는데, 지금은 거의 사용되지 않고 문서에서도 잘 다루지 않습니다.
- 대신 공식적으로는 **`kubectl`** 기반 접근을 권장합니다.

---

2. `kubectl longhorn` 플러그인

- Longhorn 프로젝트가 제공하는 **kubectl 플러그인**이 있어요.
- 설치하면 **`kubectl longhorn …`** 형태로 명령어를 쓸 수 있습니다.

```
bash

# 설치 예시
kubectl krew install longhorn
```

```
# 또는 직접 바이너리 받아서 PATH에 추가
```

- 이걸로 볼륨, 노드, 엔진, 백업 등 Longhorn 리소스를 확인하고 조작할 수 있습니다.
- 사용 예:

```
bash

kubectl longhorn volume list
kubectl longhorn volume inspect pvc-xxxx
kubectl longhorn node list
kubectl longhorn backup list
```

---

3. 그냥 `kubectl`로 CRD 다루기

- Longhorn은 사실상 Kubernetes CRD(Custom Resource Definition) 집합이라, 별도 CLI 없이도 다 제어할 수 있습니다.
- 주요 CRD:
  - **`volumes.longhorn.io`**
  - **`engines.longhorn.io`**
  - **`replicas.longhorn.io`**
  - **`nodes.longhorn.io`**
  - **`backups.longhorn.io`**
- 예시:

```
bash

# 볼륨 목록
kubectl get volumes.longhorn.io -n longhorn-system
# 특정 볼륨 상세
kubectl describe volume.longhorn.io pvc-12345 -n longhorn-system
```

👉 정리하면:

- **공식적으로는 `kubectl longhorn` 플러그인**을 추천.
- 없더라도 CRD를 직접 조회/수정 가능.
- 옛날 독립형 **`longhornctl`**은 사실상 쓰이지 않음.

원하시면 제가 **`kubectl longhorn`** 플러그인 설치법이랑 자주 쓰는 명령어 치트시트 만들어드릴까요?

> 플러그인은 패스하고 자주 쓰는 명령어 정리해줘

좋아요. 플러그인 없이 **`kubectl`만으로 Longhorn CRD**를 다루는 자주 쓰는 명령어 치트시트 정리해드릴게요.
(네임스페이스는 기본 **`longhorn-system`**입니다.)

기본 준비

```bash
NS=longhorn-system
```

리소스 개요

```bash
kubectl api-resources | grep longhorn
# volumes.longhorn.io, engines.longhorn.io, replicas.longhorn.io, nodes.longhorn.io,
# engineimages.longhorn.io, sharemanagers.longhorn.io, settings.longhorn.io,
# backups.longhorn.io, backupvolumes.longhorn.io, backuptargets.longhorn.io,
recurringjobs.longhorn.io ...
```

볼륨(Volume)

```bash
# 전체 볼륨 요약
kubectl get volumes.longhorn.io -n $NS

# 상세(이벤트 포함)
kubectl describe volume.longhorn.io <VOLUME_NAME> -n $NS

# JSON으로 핵심 필드 확인(상태, 크기, 연결 노드)
kubectl get volume.longhorn.io <V> -n $NS -o json | jq -r '.status | {state, robustness,
currentNodeID, controllers}'
kubectl get volume.longhorn.io <V> -n $NS -o jsonpath='{.spec.size}{"\n"}{.status.robustness}
{"\n"}'

# PVC ↔ Longhorn 볼륨 매핑
# (보통 VOLUME_NAME 이 PVC 이름과 동일하거나 pvc-<uid>)
kubectl get pvc -A -o wide | grep -i longhorn
kubectl get volume.longhorn.io -n $NS | grep <pvc-name-or-uid>
```

볼륨 스냅샷 (읽기)

```bash
# 스냅샷 목록은 엔진 API가 편하지만, CRD만으로는 count/상태 위주 확인
kubectl get engines.longhorn.io -n $NS | grep <VOLUME_NAME>
```

> **주의(쓰기작업):** 스냅샷/백업 "생성" 같은 액션은 UI/REST가 안전합니다. CRD로도 가능하긴 하나 필드 의존성이 있어 실수 위험이 큽니다. 아래 "백업 섹션" 참고.

엔진(Engine) / 리플리카(Replica)

```bash
# 엔진(컨트롤러) 확인
kubectl get engines.longhorn.io -n $NS -o wide
kubectl describe engine.longhorn.io <ENGINE_NAME> -n $NS

# 리플리카 상태와 배치 노드
kubectl get replicas.longhorn.io -n $NS -o wide | grep <VOLUME_NAME>
kubectl describe replica.longhorn.io <REPLICA_NAME> -n $NS
```

자주 보는 필터

```bash
# 불건전(replica)만 뽑기
kubectl get replicas.longhorn.io -n $NS -o json \
 | jq -r '.items[] | select(.status.mode!="RW") | [.metadata.name,.status.mode,.spec.nodeID] |
@tsv'
```

노드/디스크

```bash
# Longhorn 노드 상태
kubectl get nodes.longhorn.io -n $NS -o wide
kubectl describe node.longhorn.io <NODE_NAME> -n $NS

# 디스크 용량/예약 (JSONPath)
kubectl get node.longhorn.io <NODE> -n $NS -o json \
 | jq -r '.status.diskStatus | to_entries[] | [.key, .value.storageAvailable,
.value.storageScheduled] | @tsv'
```

백업(Backup) / 백업대상(BackupTarget)

```bash
# 백업 대상(예: S3/NFS 주소)
kubectl get backuptargets.longhorn.io -n $NS -o yaml

# 볼륨별 백업 목록(읽기)
kubectl get backupvolumes.longhorn.io -n $NS
kubectl get backups.longhorn.io -n $NS | grep <VOLUME_NAME>
kubectl describe backup.longhorn.io <BACKUP_NAME> -n $NS
```

참고(트리거):

- **백업 생성**: UI에서 “Create Backup”이 가장 안전.
- CRD로 만들려면 **`backups.longhorn.io`** 오브젝트를 생성하는데, 스냅샷/볼륨 이름, 백업타겟 연계가 정확해야 합니다(실수 시 고착 위험). 운영에서는 UI/API 권장.

설정/버전

```
bash


# 전역 설정 확인
kubectl get settings.longhorn.io -n $NS
kubectl get settings.longhorn.io <SETTING_NAME> -n $NS -o yaml    # 예: default-replica-count

# 엔진 이미지/공유매니저(RWX)
kubectl get engineimages.longhorn.io -n $NS -o wide
kubectl get sharemanagers.longhorn.io -n $NS -o wide
```

빠른 진단 스니펫

1) “왜 UI가 들쭉날쭉?” 엔드포인트·레디니스 체커

```
bash


# 프론트/백엔드 서비스와 엔드포인트
kubectl get svc,endpoints -n $NS | egrep 'longhorn-(front|back)end'
kubectl describe ep longhorn-frontend -n $NS
kubectl describe ep longhorn-backend  -n $NS

# 매니저 로그에서 타임아웃/리더변경
kubectl logs -n $NS -l longhorn.io/component=manager --tail=200 | egrep -i
'error|timeout|leader|health'
```

2) “어떤 PVC가 문제 볼륨이야?”

```
bash


# PVC 상태와 스토리지클래스
kubectl get pvc -A -o wide | egrep 'Bound|Pending' | egrep -i 'longhorn|lh'
# 해당 PVC의 PV → Longhorn 볼륨 이름 확인
kubectl get pv <PV_NAME> -o jsonpath='{.spec.csi.volumeHandle}{"\n"}'
```

3) “리플리카 한 대가 계속 RW가 아냐”

```
bash


kubectl get replicas.longhorn.io -n $NS -o json \
 | jq -r '.items[] | select(.metadata.labels."longhornvolume"=="<VOLUME_NAME>" and
.status.mode!="RW")'
```

(조심해서) 운영에 쓰는 변경 예시

> 아래는 **위험도 있음**. 실 운영에 적용 전, 테스트 클러스터에서 동작을 확인하세요.

A) 볼륨 Detach / Attach

```
bash


# Detach: volume.spec.nodeID 를 빈 문자열로
kubectl patch volume.longhorn.io <V> -n $NS --type merge -p '{"spec":{"nodeID":""}}'

# Attach: 특정 노드에 연결
kubectl patch volume.longhorn.io <V> -n $NS --type merge -p '{"spec":{"nodeID":"<NODE_NAME>"}}'
```

- Pod가 PVC를 통해 붙는 워크로드라면, 보통 **Pod 재스케줄**로 해결하는 게 안전합니다.

B) 레플리카 카운트 변경(향후 재빌드 유발 가능)

```
bash


kubectl patch volume.longhorn.io <V> -n $NS --type merge -p '{"spec":{"numberOfReplicas":3}}'
```

C) Recurring Job(주기 스냅샷/백업) 볼륨에 붙이기

```
bash


# 1) RecurringJob 리소스 생성(예: 매시간 스냅샷 24개 보관)
cat <<'YAML' | kubectl apply -n $NS -f -
apiVersion: longhorn.io/v1beta2
kind: RecurringJob
metadata:
  name: hourly-snap
spec:
  cron: "0 * * * *"
  task: "snapshot"
  retain: 24
  concurrency: 1
YAML

# 2) 볼륨에 적용
kubectl patch volume.longhorn.io <V> -n $NS --type merge -p \
'{"spec":{"recurringJobs":[{"name":"hourly-snap","isGroup":false}]}}'
```

정리용 원클릭 스크립트(읽기 전용)

```
bash


cat > lh-quick-health.sh <<'SH'
#!/usr/bin/env bash
set -euo pipefail
NS="${NS:-longhorn-system}"
echo "== Volumes =="; kubectl get volumes.longhorn.io -n "$NS"
echo; echo "== Engines =="; kubectl get engines.longhorn.io -n "$NS" -o wide
echo; echo "== Replicas (non-RW) ==";
kubectl get replicas.longhorn.io -n "$NS" -o json | jq -r '.items[] | select(.status.mode!="RW")
| [.metadata.name,.metadata.labels."longhornvolume",.status.mode,.spec.nodeID] | @tsv'
echo; echo "== Nodes =="; kubectl get nodes.longhorn.io -n "$NS" -o wide
echo; echo "== Services & EP =="; kubectl get svc,ep -n "$NS" | egrep 'longhorn-(front|back)end'
|| true
echo; echo "== Settings snapshot =="; kubectl get settings.longhorn.io -n "$NS" | head -n 20
SH
chmod +x lh-quick-health.sh
```

필요하시면 **당신 환경(ArgoCD/Helm values 기준)으로 안전한 패치만 묶은 운영 스크립트**(읽기/진단용과 변경용 분리)까지 정리해 드릴게요. 어느 부분부터 자동화하고 싶은지만 알려주세요.

longhorn webui에서 정보조회용 ui에 대응하는 shell을 만들어줘

좋습니다. Longhorn Web UI에서 "읽기 전용으로 보는" 정보들을 **`kubectl`**만 써서 한 번에 확인할 수 있는 셸을 만들었습니다. **변경/삭제는 전혀 하지 않도록** 안전하게 짰고, 메뉴/서브커맨드 둘 다 지원해요.

파일명 예시: **`lh-ui-readonly.sh`**
의존: **`kubectl`**, **`jq`**, (선택) **`column`**

```
bash
```

```bash
#!/usr/bin/env bash
# Longhorn "정보조회용" Read-only Shell
# - Web UI에서 보던 핵심 정보(볼륨/엔진/리플리카/노드/백업/엔드포인트/이벤트)를 안전하게 조회
# - 절대 쓰기 작업 없음(attach/detach/scale/삭제 등 일체 미구현)
# - 사용 예:
#     NS=longhorn-system ./lh-ui-readonly.sh volumes
#     ./lh-ui-readonly.sh volume --name pvc-1234
#     ./lh-ui-readonly.sh replicas --volume pvc-1234
#     ./lh-ui-readonly.sh engines --volume pvc-1234
#     ./lh-ui-readonly.sh nodes
#     ./lh-ui-readonly.sh backups --volume pvc-1234
#     ./lh-ui-readonly.sh svcep
#     ./lh-ui-readonly.sh events
#     ./lh-ui-readonly.sh dashboard
#     ./lh-ui-readonly.sh menu

set -euo pipefail

NS="${NS:-longhorn-system}"
KUBECTL_BIN="${KUBECTL_BIN:-kubectl}"
JQ_BIN="${JQ_BIN:-jq}"

#--- util ------------------------------------------------------------------

die() { echo "[ERR] $*" >&2; exit 1; }

need_bin() {
  command -v "$1" >/dev/null 2>&1 || die "필요한 명령어가 없습니다: $1"
}

json_get() { # safe jq
  "$JQ_BIN" -r "$1" 2>/dev/null || true
}

jsoncols() { # pretty table using column if available
  if command -v column >/dev/null 2>&1; then
    column -t -s$'\t'
  else
    cat
  fi
}

ns_check() {
  $KUBECTL_BIN get ns "$NS" >/dev/null 2>&1 || die "네임스페이스가 없습니다: $NS"
}

#--- render helpers --------------------------------------------------------

hr() { printf '%s\n' "------------------------------------------------------------------------"; }
```

```
title() { printf '\n## %s\n' "$*"; }

#--- sections ----------------------------------------------------------------

dashboard() {
  title "Longhorn 대시보드 (요약)"
  echo "[Namespace] $NS"
  echo

  title "Volumes (요약)"
  $KUBECTL_BIN get volumes.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.name,
         .status.robustness,
         (.status.currentNodeID // "-"),
         (.spec.numberOfReplicas // "-"),
         (.spec.size // "-")] | @tsv' \
  | (echo -e "VOLUME\tROBUSTNESS\tNODE\tREPLICAS\tSIZE"; cat) | jsoncols

  title "Engines (컨트롤러)"
  $KUBECTL_BIN get engines.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.name,
         .metadata.labels.longhornvolume,
         (.status.currentState // "-"),
         (.spec.nodeID // "-")] | @tsv' \
  | (echo -e "ENGINE\tVOLUME\tSTATE\tNODE"; cat) | jsoncols

  title "Replicas (비-RW만 표시)"
  $KUBECTL_BIN get replicas.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | select(.status.mode != "RW")
      | [.metadata.name,
         .metadata.labels.longhornvolume,
         .status.mode,
         (.spec.nodeID // "-")] | @tsv' \
  | (echo -e "REPLICA\tVOLUME\tMODE\tNODE"; cat) | jsoncols

  title "Nodes (Longhorn)"
  $KUBECTL_BIN get nodes.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.name,
         (.status.conditions[]? | select(.type=="Ready") | .status ) // "-",
         (.status.schedulable // "-")] | @tsv' \
  | (echo -e "NODE\tREADY\tSCHEDULABLE"; cat) | jsoncols

  title "Services & Endpoints (frontend/backend)"
  $KUBECTL_BIN get svc,ep -n "$NS" | egrep 'longhorn-(front|back)end' || true
}

volumes() {
  title "Volumes (전체)"
  $KUBECTL_BIN get volumes.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.name,
         .status.state,
         .status.robustness,
         (.status.currentNodeID // "-"),
         (.spec.numberOfReplicas // "-"),
         (.spec.size // "-"),
         (.status.kubernetesStatus.pvcName // "-"),
         (.status.kubernetesStatus.namespace // "-")] | @tsv' \
  | (echo -e "VOLUME\tSTATE\tROBUSTNESS\tNODE\tREPLICAS\tSIZE\tPVC\tPVC_NS"; cat) | jsoncols
}

volume_detail() {
  local V="${1:-}"
  [[ -n "$V" ]] || die "볼륨 이름을 --name <VOLUME> 로 지정하세요."
  title "Volume 상세: $V"
  $KUBECTL_BIN get volume.longhorn.io "$V" -n "$NS" -o json \
  | $JQ_BIN -r '

["name","state","robustness","node","size","replicas","pvc","pvc_ns","frontend","engineImage","r
```

```
estoreStatus"],
    [ .metadata.name,
      .status.state,
      .status.robustness,
      (.status.currentNodeID // "-"),
      (.spec.size // "-"),
      (.spec.numberOfReplicas // "-"),
      (.status.kubernetesStatus.pvcName // "-"),
      (.status.kubernetesStatus.namespace // "-"),
      (.spec.frontend // "-"),
      (.spec.engineImage // "-"),
      ( (.status.restoreStatus[]? | select(.latestRestored!=null)) // {} | tostring )
    ] | @tsv' \
  | jsoncols

  title "연결된 Engine / Replicas"
  $KUBECTL_BIN get engines.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r --arg V "$V" '.items[]
      | select(.metadata.labels.longhornvolume==$V)
      | [.metadata.name, (.status.currentState // "-"), (.spec.nodeID // "-")] | @tsv' \
  | (echo -e "ENGINE\tSTATE\tNODE"; cat) | jsoncols

  $KUBECTL_BIN get replicas.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r --arg V "$V" '.items[]
      | select(.metadata.labels.longhornvolume==$V)
      | [.metadata.name, .status.mode, (.spec.nodeID // "-")] | @tsv' \
  | (echo -e "REPLICA\tMODE\tNODE"; cat) | jsoncols
}

replicas_by_volume() {
  local V="${1:-}"
  [[ -n "$V" ]] || die "볼륨 이름을 --volume <VOLUME> 로 지정하세요."
  title "Replicas for volume: $V"
  $KUBECTL_BIN get replicas.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r --arg V "$V" '.items[]
      | select(.metadata.labels.longhornvolume==$V)
      | [.metadata.name,.status.mode, (.spec.nodeID // "-"), (.status.purgeStatus // "-")] |
@tsv' \
  | (echo -e "REPLICA\tMODE\tNODE\tPURGE_STATUS"; cat) | jsoncols
}

engines_by_volume() {
  local V="${1:-}"
  [[ -n "$V" ]] || die "볼륨 이름을 --volume <VOLUME> 로 지정하세요."
  title "Engines for volume: $V"
  $KUBECTL_BIN get engines.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r --arg V "$V" '.items[]
      | select(.metadata.labels.longhornvolume==$V)
      | [.metadata.name, (.status.currentState // "-"), (.spec.nodeID // "-"), (.status.endpoint
// "-")] | @tsv' \
  | (echo -e "ENGINE\tSTATE\tNODE\tENDPOINT"; cat) | jsoncols
}

nodes() {
  title "Longhorn Nodes"
  $KUBECTL_BIN get nodes.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.name,
         (.status.conditions[]? | select(.type=="Ready") | .status ) // "-",
         (.status.schedulable // "-"),
         ( [ (.status.diskStatus | to_entries[]? | "\(.key)") ] | join(",") )
        ] | @tsv' \
  | (echo -e "NODE\tREADY\tSCHEDULABLE\tDISKS"; cat) | jsoncols

  title "각 노드 디스크 용량(available/scheduled)"
  while read -r NODE; do
    [[ -n "$NODE" ]] || continue
    echo "Node: $NODE"
    $KUBECTL_BIN get node.longhorn.io "$NODE" -n "$NS" -o json \
    | $JQ_BIN -r '.status.diskStatus
        | to_entries[]
        | [ .key, .value.storageAvailable, .value.storageScheduled,
.value.conditions.ready.status ] | @tsv' \
    | (echo -e "DISK_PATH\tAVAILABLE\tSCHEDULED\tREADY"; cat) | jsoncols
```

```
    hr
  done < <($KUBECTL_BIN get nodes.longhorn.io -n "$NS" -o json | $JQ_BIN -r
'.items[].metadata.name')
}

backups() {
  local V="${1:-}"
  title "Backup Target"
  $KUBECTL_BIN get backuptargets.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[] | [.metadata.name, (.spec.backend // "-"), (.spec.credentialSecret //
"-"), (.status.available // "-")] | @tsv' \
  | (echo -e "TARGET\tBACKEND\tCRED_SECRET\tAVAILABLE"; cat) | jsoncols

  title "Backup Volumes"
  $KUBECTL_BIN get backupvolumes.longhorn.io -n "$NS" -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.name, (.status.size // "-"), (.status.lastBackupAt // "-"),
(.status.dataStored // "-")] | @tsv' \
  | (echo -e "VOLUME\tSIZE\tLAST_BACKUP_AT\tDATA_STORED"; cat) | jsoncols

  if [[ -n "${V}" ]]; then
    title "Backups for volume: $V"
    $KUBECTL_BIN get backups.longhorn.io -n "$NS" -o json \
    | $JQ_BIN -r --arg V "$V" '.items[]
        | select(.metadata.labels.longhornvolume==$V)
        | [.metadata.name, (.status.snapshotName // "-"), (.status.createdAt // "-"),
(.status.state // "-"), (.status.message // "-")] | @tsv' \
    | (echo -e "BACKUP\tSNAPSHOT\tCREATED_AT\tSTATE\tMESSAGE"; cat) | jsoncols
  fi
}

svcep() {
  title "Services / Endpoints (frontend/backend)"
  $KUBECTL_BIN get svc,ep -n "$NS" | egrep 'longhorn-(front|back)end' || true

  title "Ingress(있다면) 요약"
  $KUBECTL_BIN get ing -n "$NS" -o json \
  | $JQ_BIN -r '.items[]?
      | [.metadata.name,
         ( .spec.tls // [] | length | tostring),
         ( .spec.rules[]? | .host )] | @tsv' \
  | (echo -e "INGRESS\tTLS_COUNT\tHOST"; cat) | jsoncols
}

events() {
  title "최근 이벤트 (상위 40)"
  $KUBECTL_BIN get events -n "$NS" --sort-by=.lastTimestamp | tail -n 40 || true
}

pvcmap() {
  title "PVC ↔ PV ↔ Longhorn Volume 맵"
  # 모든 바운드 PVC만
  $KUBECTL_BIN get pvc -A --field-selector=status.phase=Bound -o json \
  | $JQ_BIN -r '.items[]
      | [.metadata.namespace,
         .metadata.name,
         .spec.volumeName,
         (.spec.storageClassName // "-")] | @tsv' \
  | (echo -e "PVC_NS\tPVC\tPV\tSC"; cat) | jsoncols

  echo
  title "PV → CSI volumeHandle(Longhorn 볼륨명)"
  $KUBECTL_BIN get pv -o json \
  | $JQ_BIN -r '.items[]
      | select(.spec.csi != null)
      | [.metadata.name, .spec.csi.driver, .spec.csi.volumeHandle] | @tsv' \
  | (echo -e "PV\tCSI_DRIVER\tVOLUME_HANDLE"; cat) | jsoncols
}

#--- CLI ----------------------------------------------------------------------

usage() {
cat <<EOF
```

```
Usage: [NS=<longhorn-namespace>] $(basename "$0") <command> [--name <V>] [--volume <V>]

Commands (모두 읽기 전용):
  dashboard           대시보드 요약(볼륨/엔진/리플리카/노드/서비스)
  volumes             볼륨 전체 목록 요약
  volume --name V     특정 볼륨 상세 (엔진/리플리카 포함)
  replicas --volume V 해당 볼륨의 리플리카 목록
  engines  --volume V 해당 볼륨의 엔진 목록
  nodes               Longhorn 노드 및 디스크 상태
  backups [--volume V]백업타겟/볼륨, (옵션) 특정 볼륨 백업 목록
  svcep               서비스/엔드포인트/인그레스 요약
  events              최근 이벤트 상위 40
  pvcmap              PVC↔PV↔Longhorn 볼륨 매핑 보기
  menu                간단 메뉴 모드

예)
  NS=longhorn-system $(basename "$0") dashboard
  $(basename "$0") volume --name pvc-1234
  $(basename "$0") replicas --volume pvc-1234
EOF
}

menu() {
  select opt in "dashboard" "volumes" "volume" "replicas" "engines" "nodes" "backups" "svcep"
"events" "pvcmap" "quit"; do
    case "$opt" in
      dashboard) dashboard ;;
      volumes) volumes ;;
      volume) read -rp "볼륨 이름: " V; volume_detail "$V" ;;
      replicas) read -rp "볼륨 이름: " V; replicas_by_volume "$V" ;;
      engines) read -rp "볼륨 이름: " V; engines_by_volume "$V" ;;
      nodes) nodes ;;
      backups) read -rp "볼륨 이름(엔터=전체): " V; backups "$V" ;;
      svcep) svcep ;;
      events) events ;;
      pvcmap) pvcmap ;;
      quit) break ;;
      *) echo "메뉴 번호를 선택하세요." ;;
    esac
  done
}

main() {
  need_bin "$KUBECTL_BIN"
  need_bin "$JQ_BIN"
  ns_check

  local cmd="${1:-}"; shift || true
  local VOLUME_NAME=""; local NAME=""

  while [[ $# -gt 0 ]]; do
    case "$1" in
      --volume) VOLUME_NAME="${2:-}"; shift 2;;
      --name)   NAME="${2:-}"; shift 2;;
      -h|--help) usage; exit 0;;
      *) set +x; break;;
    esac
  done

  case "$cmd" in
    dashboard) dashboard ;;
    volumes) volumes ;;
    volume) volume_detail "$NAME" ;;
    replicas) replicas_by_volume "$VOLUME_NAME" ;;
    engines) engines_by_volume "$VOLUME_NAME" ;;
    nodes) nodes ;;
    backups) backups "$VOLUME_NAME" ;;
    svcep) svcep ;;
    events) events ;;
    pvcmap) pvcmap ;;
```

```
    menu) menu ;;
    ""|-h|--help) usage ;;
    *) die "알 수 없는 커맨드: $cmd" ;;
  esac
}

main "$@"
```

쓰는 법(간단 요약)

```
bash


chmod +x lh-ui-readonly.sh
NS=longhorn-system ./lh-ui-readonly.sh dashboard
./lh-ui-readonly.sh volume --name pvc-<볼륨명>
./lh-ui-readonly.sh replicas --volume pvc-<볼륨명>
./lh-ui-readonly.sh nodes
./lh-ui-readonly.sh backups --volume pvc-<볼륨명>
./lh-ui-readonly.sh svcep
./lh-ui-readonly.sh menu
```

원한다면 **당신 환경(ArgoCD/Helm values 네임스페이스 체계, 사내 CA 유무)**에 맞춰 컬럼이나 필드 더 늘려서 커스터마이즈해드릴게요.