# Longhorn - Distributed Block Storage System für Kubernetes

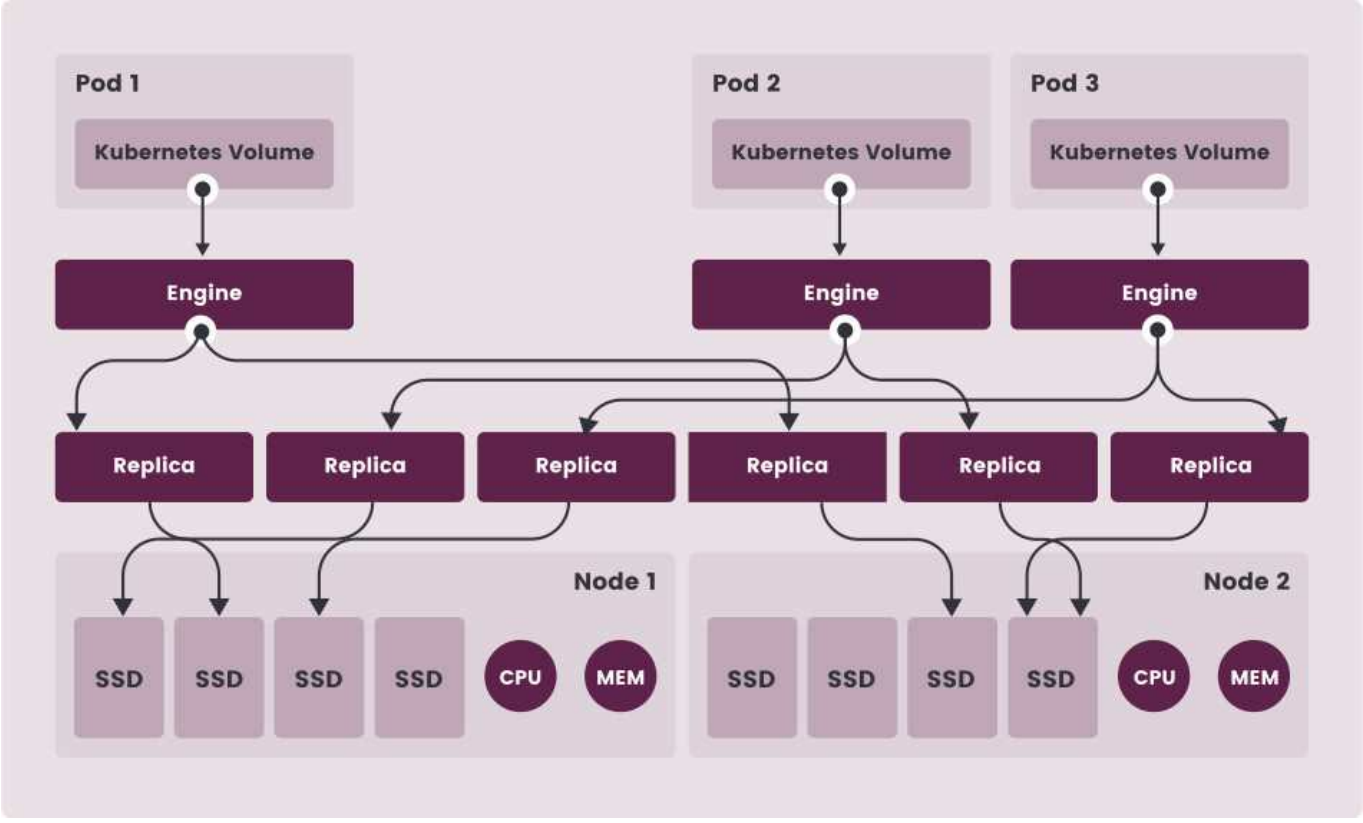
## Was ist Longhorn?

Longhorn ist eine container-native Speicherlösung für Kubernetes-Umgebungen. Als Open-Source-Projekt der Cloud Native Computing Foundation (CNCF) steht Longhorn frei zur Verfügung. Mit Longhorn kann in containerisierten Umgebungen persistenter Blockspeicher bereitgestellt werden.

Das ursprünglich von Rancher Labs, inzwischen Teil von SUSE, initiierte Projekt Longhorn adressiert den Bereich Software-Defined Storage (SDS) für Kubernetes. Die Lösung ist speziell auf Cloud-native, verteilte Blockspeicher für Container-Umgebungen ausgelegt, die mittels Kubernetes orchestriert werden. Seit dem Jahr 2020 gehört Longhorn zu den CNCF-Projekten.

Mit Longhorn lassen sich Anwendungen in Kubernetes-Clustern mit skalierbarem und dauerhaftem Speicher ausstatten. Die Software läuft auf Standard-Hardware und ist komplett quelloffen. Sie wird unter der Apache-2.0-Lizenz veröffentlicht.

### Funktionsweise und Features




Longhorn basiert auf einem containerorientierten Ansatz und realisiert verteilten Blockspeicher unter Verwendung der Prinzipien von Containerisierung und Microservices. Für jedes Blockspeichervolume wird ein separater Storage-Controller bereitgestellt. Die Volumes werden über mehrere Knoten

hinweg synchron repliziert. Longhorn wird von Kubernetes mittels eines CSI-Treibers (Container Storage Interface Driver) als Speicheranbieter angesprochen.

Operationen in Kubernetes wie das Erstellen oder Einbinden von Volumes werden durch Longhorn in entsprechende interne Aktionen übersetzt. Dadurch kann Kubernetes die Steuerung der Storage-Controller sowie der Replikate übernehmen. Zu den wichtigsten Funktionen von Longhorn zählen:

- Verteilte Datenspeicherung und automatische Replikation der Daten über mehrere Knoten,
- Dynamische Bereitstellung und Zuordnung von Storage-Kapazitäten ohne manuellen Aufwand,
- Eingebaute Mechanismen für automatische Snapshots, Backups und Wiederherstellung,
- Speicherung von Backups auf sekundären Storages wie NFSv4 oder S3-kompatiblen Objektspeichern,
- Grafische, webbasierte Benutzeroberfläche für die Verwaltung und Überwachung der Storage-Ressourcen,
- Unterbrechungsfreie Aktualisierungen des Longhorn-Stacks.

### Anwendungsmöglichkeiten der verteilten Block-Storage-Lösung

Die verteilte Open-Source-Block-Storage-Lösung für Kubernetes eröffnet vielfältige Einsatzmöglichkeiten. Longhorn eignet sich besonders für container-native, auf Microservices basierende Anwendungen. Durch die Bereitstellung von persistentem Speicher können zustandsbehaftete Anwendungen auch in vergänglichen, containerisierten Umgebungen umgesetzt werden. Typische mit Longhorn als persistentem Storage realisierbare Anwendungsfälle sind verteilte Datenbanken, Analyseplattformen, Webanwendungen, KI- und Machine-Learning-Workloads sowie Content-Management-Systeme.

### Vorteile

Longhorn wird unter der Apache-2.0-Lizenz angeboten und steht als Open-Source-Software kostenlos zur Verfügung. Es existiert eine aktive Online-Community, zudem kann professioneller, kostenpflichtiger Support für die Block-Storage-Lösung bezogen werden. Longhorn erleichtert die Bereitstellung und Verwaltung von persistentem Datenspeicher in von Kubernetes gesteuerten Container-Umgebungen. Storage-Volumes lassen sich direkt über Kubernetes verwalten.

Longhorn ist mit Standard-Hardware kompatibel und plattformunabhängig sowohl für Bare-Metal-, On-Premises- als auch für Cloud-Installationen einsetzbar. Weitere Pluspunkte sind:

- Hohe Verfügbarkeit und Zuverlässigkeit dank verteilter Speicherung und synchroner Replikation,
- eine nutzerfreundliche grafische Benutzeroberfläche zur Administration und Überwachung der Speicherressourcen,
- schnelles und unkompliziertes Disaster Recovery,
- flexible Skalierbarkeit,
- unterbrechungsfreie Aktualisierungen.

## Installation und Konfiguration

## Voraussetzungen

- Ein Kubernetes-Cluster ab Version 1.25.
- Helm in Version 3.
- Grundlegende Kenntnisse in Kubernetes.

## Konfiguration

Legen das Helm Chart Values File `longhorn-values.yaml` an, um die Konfiguration für Longhorn zu speichern.

Als Erstes passen wir die Einstellung für die Datenlokalität an. Setze diese auf best-effort, damit Longhorn versucht, eine Replik auf demselben Knoten wie das angeschlossene Volume zu halten:

```
defaultSettings:
  storageMinimalAvailablePercentage: "10"
  defaultDataPath: "/mnt/longhorn-storage"

persistence:
  defaultClassReplicaCount: 2
  defaultDataLocality: "best-effort"

longhornUI:
  replicas: 1

service:
  ui:
    type: LoadBalancer
```

Hier findet ihr noch weitere Helm Chart Values Varianten welche ich getestet habe.

## Installation

Erstelle den Namespace `longhorn-system` und richte das Helm-Repository ein.

```
kubectl create ns longhorn-system
namespace/longhorn-system created

helm repo add longhorn https://charts.longhorn.io
"longhorn" has been added to your repositories

helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "longhorn" chart repository
Update Complete. ⎈Happy Helming!⎈
```

Nun kannst Du das Longhorn-System installieren:

```
helm install longhorn longhorn/longhorn \
  --namespace longhorn-system \
  --values longhorn.values.yaml
NAME: longhorn
LAST DEPLOYED: Wed Jul 24 07:16:08 2024
NAMESPACE: longhorn-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Longhorn is now installed on the cluster!

Please wait a few minutes for other Longhorn components such as CSI
deployments, Engine Images, and Instance Managers to be initialized.

Visit our documentation at https://longhorn.io/docs/
```

## Nacharbeiten / maintenance

Prüfe den Status Deiner Installation:

```
kubectl get pods -n longhorn-system
NAME                                                READY   STATUS
RESTARTS   AGE
csi-attacher-5f8b9cbbdc-2dbdg                       1/1     Running   0
45s
csi-attacher-5f8b9cbbdc-hq7ps                       1/1     Running   0
45s
csi-attacher-5f8b9cbbdc-k2tpq                       1/1     Running   0
45s
csi-provisioner-6d85c78995-b4b4r                    1/1     Running   0
45s
csi-provisioner-6d85c78995-lvvwb                    1/1     Running   0
45s
csi-provisioner-6d85c78995-nkm7g                    1/1     Running   0
45s
csi-resizer-677c8b7c75-h9kc7                        1/1     Running   0
45s
csi-resizer-677c8b7c75-qbvxl                        1/1     Running   0
45s
csi-resizer-677c8b7c75-wppts                        1/1     Running   0
45s
csi-snapshotter-868c6c774d-4z57t                    1/1     Running   0
45s
csi-snapshotter-868c6c774d-9t2cd                    1/1     Running   0
45s
csi-snapshotter-868c6c774d-ps5lp                    1/1     Running   0
45s
engine-image-ei-b0369a5d-4jqzf                      1/1     Running   0
```

```
47s
instance-manager-84b0d2d8f37aaeabfa5785c886a57710   1/1     Running   0
47s
longhorn-csi-plugin-bsv4m                           3/3     Running   0
45s
longhorn-driver-deployer-6dff48688b-4497d           1/1     Running   0
62s
longhorn-manager-ft96r                              1/1     Running   0
62s
longhorn-ui-966cf4bb4-5shp9                         1/1     Running   0
62s
longhorn-ui-966cf4bb4-lx2th                         1/1     Running   0
62s
```

Richte einen wiederkehrenden Auftrag ein, um das Dateisystem regelmäßig zu trimmen und so Speicherplatz zurückzugewinnen. Erstelle dazu die Datei `trim-filesystem.recurring-job.yaml`:

```
apiVersion: longhorn.io/v1beta2
kind: RecurringJob
metadata:
  name: trim-filesystem
  namespace: longhorn-system
spec:
  concurrency: 1
  cron: 0 0 * * *
  name: trim-filesystem
  task: filesystem-trim
  groups:
    - default
```

Erstelle einen weiteren wiederkehrenden Auftrag, der entfernbaren Snapshots regelmäßig bereinigt. `snapshot-cleanup.recurring-job.yaml`:

```
apiVersion: longhorn.io/v1beta2
kind: RecurringJob
metadata:
  name: snapshot-cleanup
  namespace: longhorn-system
spec:
  concurrency: 1
  cron: 12 0 * * *
  groups:
    - default
  name: snapshot-cleanup
  task: snapshot-cleanup
```

Deploy beide wiederkehrenden Aufträge in Deinen Cluster:

```
kubectl apply -f trim-filesystem.recurring-job.yaml
recurringjob.longhorn.io/trim-filesystem created
```

```
kubectl apply -f snapshot-cleanup.recurring-job.yaml
recurringjob.longhorn.io/snapshot-cleanup created
```

Standardmäßig löscht Longhorn verwaiste Ressourcen nicht automatisch. Es gibt jedoch eine Einstellung, mit der wir dies ermöglichen können. Setze dazu orphan-auto-deletion auf true:

```
kubectl -n longhorn-system get setting orphan-auto-deletion -o yaml | sed
's/value: "false"/value: "true"/' | kubectl apply -f -
setting.longhorn.io/orphan-auto-deletion configured

kubectl -n longhorn-system get setting orphan-auto-deletion
NAME                    VALUE   AGE
orphan-auto-deletion    true    53m
```