

Clarifying file issue

The user mentioned the file is incomplete. I should ask what's missing or offer to regenerate it with the full content. However, I know the tools already produced a truncated PDF earlier. I think the best approach is to provide the full content in markdown format, which the user can convert to PDF locally. I'll keep it concise but complete, and ask which sections were cut. I'll offer the full manual in markdown now. Good catch—thanks for flagging that.

Let's sidestep the truncation problem: here's the **full, self-contained manual in Markdown**, which you can save as a `.md` or `.docx` and export to PDF locally (VS Code, Obsidian, Word, etc.). This way you're guaranteed a complete, non-truncated version.

RKE2 + Longhorn + RabbitMQ Engineering Manual

Architecture, Operations, and Disaster Recovery

Prepared for: Andy

Edition: 2026

Classification: Internal Engineering Documentation

Index

1. Introduction
2. System architecture overview
 - 2.1 Logical data flow
 - 2.2 Component responsibilities
 - 2.3 ASCII architecture diagram
3. Operational principles
 - 3.1 Storage logic
 - 3.2 Scheduling logic
 - 3.3 Failure domains
4. Disaster recovery runbook
 - 4.1 Executive summary
 - 4.2 Preconditions
 - 4.3 Severity levels
 - 4.4 Scenario A — node loss
 - 4.5 Scenario B — volume faulted / detached
 - 4.6 Scenario C — full cluster rebuild
 - 4.7 Post-incident checklist
5. Longhorn troubleshooting cheat sheet

6. RabbitMQ Helm overlay (best-practice)

- 6.1 YAML overlay
- 6.2 Deployment logic
- 6.3 Upgrade logic

7. Appendices

- 7.1 Key commands
 - 7.2 Replica selection logic
 - 7.3 Salvage decision tree
-

1. Introduction

This engineering manual documents the architecture, operational model, and disaster-recovery procedures for a production-grade deployment of:

- RKE2 (Kubernetes distribution)
- Longhorn (distributed block storage)
- RabbitMQ (stateful message broker)

The goal is a **deterministic, logically justified** operational framework: every action is tied to a clear reason and failure mode.

2. System architecture overview

2.1 Logical data flow

1. Applications publish and consume messages via RabbitMQ.
2. RabbitMQ persists durable state under `/var/lib/rabbitmq/mnesia`.
3. That path is backed by a Longhorn RWO block device exposed as a Kubernetes PVC.
4. Longhorn replicates the underlying data across worker nodes.
5. RKE2 manages node lifecycle, pod scheduling, and control-plane availability.

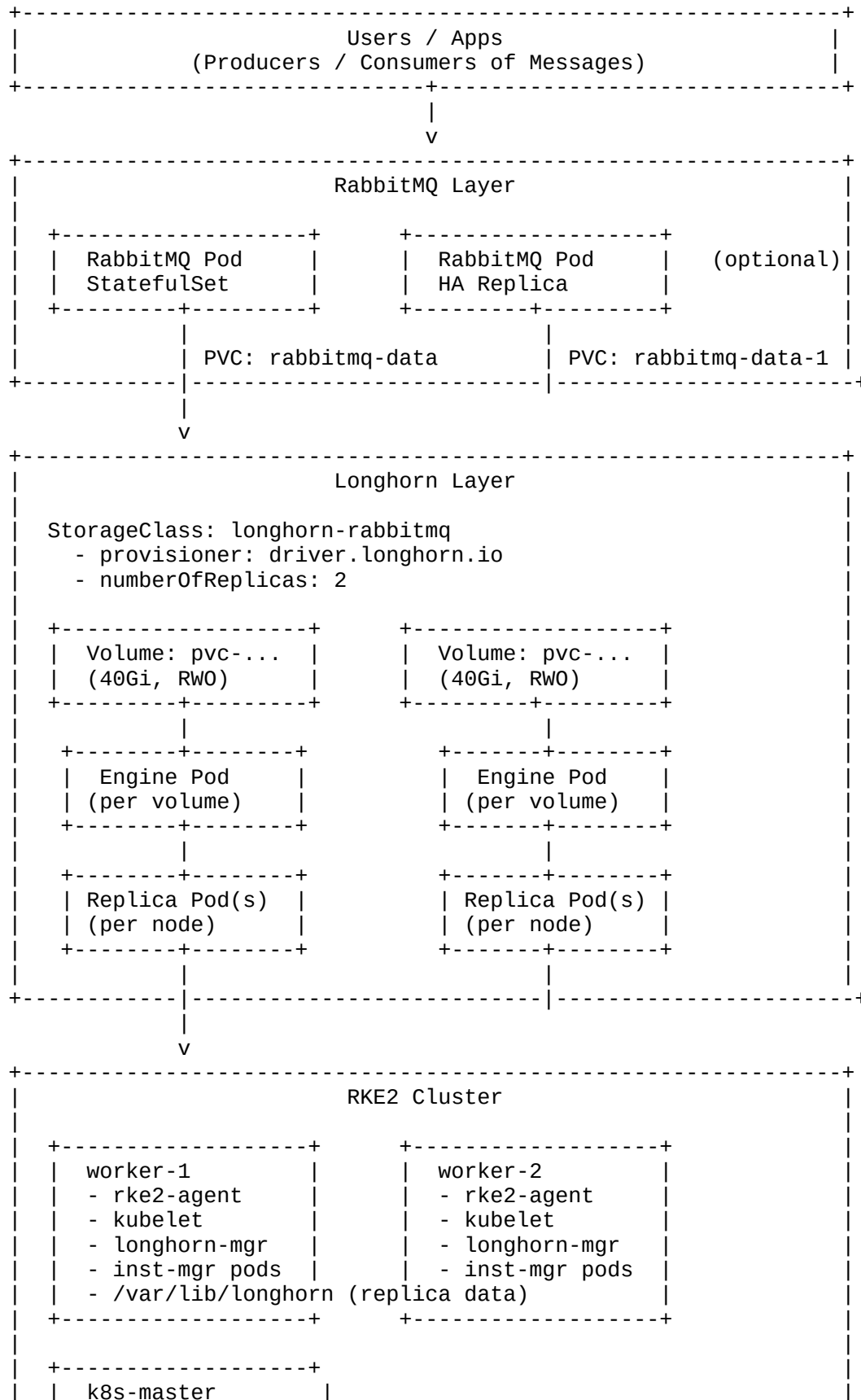
The dependency chain is:

RKE2 → Longhorn → RabbitMQ → Applications

2.2 Component responsibilities

- **Applications:** Business logic, message production/consumption.
 - **RabbitMQ:** Queue semantics, message durability, clustering (if enabled).
 - **Longhorn:** Block device abstraction, replication, failover, salvage.
 - **RKE2:** Cluster orchestration, node health, scheduling, upgrades.
-

2.3 ASCII architecture diagram





3. Operational principles

3.1 Storage logic

Longhorn's model:

- **Engine:** The process that exposes the block device and coordinates replicas.
- **Replicas:** Sparse files on worker disks under `/var/lib/longhorn/replicas/...`
- **Replication:** Synchronous; writes are acknowledged only when replicas confirm.
- **Self-healing:** Missing replicas are rebuilt when nodes return or new nodes are added.
- **Salvage:** Manual re-adoption of replicas when metadata/engine state is broken but data remains.

Key idea: **if the replica files are intact, the volume can usually be salvaged.**

3.2 Scheduling logic

- RabbitMQ uses a **ReadWriteOnce** PVC.
- The pod must run on the node where the Longhorn volume is attached.
- RKE2 may reschedule pods during:
 - Node drain
 - Upgrades
 - Node failure

Longhorn's CSI driver handles attach/detach as pods move, but stale engines or broken instance managers can block this.

3.3 Failure domains

Failure domain	Typical symptom	Primary tool
Single worker node	Degraded volume, one replica missing	Replica rebuild
Engine/metadata mismatch	Volume Detached or Faulted	Engine deletion + salvage
Full control-plane loss	Lost CRDs, volumes not visible	Reinstall + salvage

4. Disaster recovery runbook

4.1 Executive summary

This runbook covers recovery from:

- Node failures
- Volume degradation and faults
- Engine/metadata corruption
- Full cluster rebuilds with disks preserved

Guiding principle: **prefer salvage over restore when data on disk is intact and consistent; prefer restore when integrity is doubtful.**

4.2 Preconditions

Before executing DR steps, confirm:

- `kubectl` can reach the API server (or you know you're in full rebuild mode).
 - `longhorn-manager` pods are Running on all worker nodes.
 - Instance-manager pods are Running on all worker nodes.
 - Worker disks containing `/var/lib/longhorn` are present and not obviously corrupted.
-

4.3 Severity levels

- **SEV-1:** RabbitMQ unavailable, volume Faulted or Detached, business impact high.
 - **SEV-2:** RabbitMQ available but volume Degraded; risk of data loss if another node fails.
 - **SEV-3:** Node offline but service stable; volume Healthy or Degraded with redundancy.
-

4.4 Scenario A — node loss (volume degraded)

Logic:

A node going offline removes one replica. Longhorn marks the volume **Degraded** but keeps it **Attached** and usable. When the node returns, Longhorn rebuilds the missing replica.

Steps:

1. Check node and volume status:

```
kubectl get nodes
kubectl -n longhorn-system get volumes.longhorn.io
```

2. If a worker is **NotReady**, attempt to bring it back:

```
# On the affected worker
sudo systemctl restart rke2-agent
```

3. Once the node is Ready, watch the volume:

```
kubectl -n longhorn-system get volumes.longhorn.io -w
```

Expect: **Degraded** → **Healthy** after replica rebuild.

4. If RabbitMQ is unstable, restart the pod:

```
kubectl rollout restart statefulset rabbitmq -n <namespace>
```

4.5 Scenario B — volume faulted / detached, data on disk

This matches your “stalled 40Gi volume” situation.

Logic:

A stale or broken engine object can prevent Longhorn from attaching the volume. Deleting the engine forces the volume into **Faulted**, which unlocks **Salvage**. Salvage lets you explicitly choose which replicas to trust.

Steps:

1. Verify Longhorn control plane:

```
kubectl -n longhorn-system get pods
```

2. List engines:

```
kubectl -n longhorn-system get engines.longhorn.io
```

3. Delete the engine for the problematic volume:

```
kubectl -n longhorn-system delete engine -l longhornvolume=<volume-name>
```

After this, the volume should show as **Faulted** in the Longhorn UI.

4. In the Longhorn UI:

- Go to **Volumes**.
- Locate the Faulted volume (e.g., `pvc-5adeffdc - . . .`).
- Click `:` → **Salvage**.
- Select replicas on `worker-1` and `worker-2` that correspond to the preserved data.
- Confirm.

5. Wait for the volume to transition:

- Faulted → Detached → Healthy/Degraded.
- Actual size should reflect the expected data size (e.g., ~40Gi).

6. Attach the volume to the node where RabbitMQ will run (via UI or by letting the pod schedule).

7. Restart RabbitMQ:

```
kubectl scale statefulset rabbitmq -n <namespace> --replicas=0
```

```
kubectl scale statefulset rabbitmq -n <namespace> --replicas=1
```

8. Validate:

- RabbitMQ pod Running.
 - Volume Attached and Healthy/Degraded.
 - Application behaviour normal.
-

4.6 Scenario C — full cluster rebuild, disks preserved

Logic:

Longhorn's metadata (CRDs, etc.) lives in Kubernetes; replica data lives on disk. If you lose the cluster but keep the disks, you can reinstall Longhorn and re-adopt replicas.

Steps:

1. Rebuild RKE2:

- Recreate master and worker nodes.
- Join workers to the cluster.
- Ensure the same disks are mounted at `/var/lib/longhorn`.

2. Reinstall Longhorn (same major/minor version as before):

```
kubectl apply -f
https://raw.githubusercontent.com/longhorn/longhorn/v1.5.3/deploy/
longhorn.yaml
```

3. Wait for `longhorn-manager`, `instance-manager`, and `engine-image` pods to be Running.

4. In the Longhorn UI:

- Longhorn should detect existing replica directories on the worker disks.

5. Recreate volume shells:

- Create a new volume with the same size and name (or at least same PVC binding) as before.
- If necessary, use `Retain` reclaim policy to keep PVs.

6. Use **Salvage**:

- Mark the volume Faulted if needed (delete engine).
- Salvage and select the replicas corresponding to the preserved data.

7. Recreate the RabbitMQ PVC and StatefulSet:

- Use the same `claimName` and `storageClassName` so the PVC binds to the salvaged volume.

8. Validate RabbitMQ and data integrity.

4.7 Post-incident checklist

- [] All nodes in Ready state (or intentionally drained).
 - [] Longhorn volumes for RabbitMQ: **Healthy**.
 - [] Replica count matches policy (e.g., 2 replicas).
 - [] RabbitMQ pod Running, no CrashLoopBackOff.
 - [] Queues and messages behave as expected.
 - [] Backups tested or at least verified as present.
 - [] Incident documented with root cause and remediation.
-

5. Longhorn troubleshooting cheat sheet

Fast status commands

```
# Cluster and nodes
kubectl get nodes
```

```
# Longhorn core components
kubectl -n longhorn-system get pods
```

```
# Volumes
kubectl -n longhorn-system get volumes.longhorn.io
```

```
# Engines and replicas
kubectl -n longhorn-system get engines.longhorn.io
kubectl -n longhorn-system get replicas.longhorn.io
```

Symptoms → causes → fixes

Symptom	Likely cause	Typical fix
Volume Degraded	Missing or rebuilding replica	Wait for rebuild; check node health
Volume Detached, pod Pending	Engine/CSI issue, node mismatch	Check engines, delete stale engine, reattach
Volume shows 0 B	Metadata/engine mismatch	Force Faulted → Salvage → select replicas
Engine image not deployed	Engine image DaemonSet not on node	Restart instance managers / engine image pods

Quick fixes

```
# Restart instance managers
kubectl -n longhorn-system delete pods -l longhorn.io/component=instance-manager
```

```
# Restart Longhorn managers
kubectl -n longhorn-system delete pods -l app=longhorn-manager
```

```
# Force volume into Faulted state (for Salvage)
kubectl -n longhorn-system delete engine -l longhornvolume=<volume-name>
```

Salvage usage rules

Use Salvage when:

- Volume is Faulted.
- Replica data exists on worker disks.
- You have reasonable confidence in disk integrity.

Avoid Salvage when:

- Disks show corruption or hardware errors.
 - You have a clean, recent backup that is safer to restore.
-

6. RabbitMQ Helm overlay (best-practice)

6.1 YAML overlay

```
replicaCount: 1

auth:
  username: admin
  password: changeme
  erlangCookie: supersecretcookie

persistence:
  enabled: true
  storageClass: longhorn-rabbitmq
  accessMode: ReadWriteOnce
  size: 40Gi

resources:
  requests:
    cpu: 250m
    memory: 512Mi
  limits:
    cpu: 1
    memory: 1Gi

volumePermissions:
  enabled: true

podSecurityContext:
  fsGroup: 1001

containerSecurityContext:
  runAsUser: 1001
  runAsNonRoot: true

livenessProbe:
  enabled: true
  initialDelaySeconds: 60
  periodSeconds: 30
  timeoutSeconds: 5
  failureThreshold: 6
```

```
readinessProbe:
  enabled: true
  initialDelaySeconds: 20
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 6

startupProbe:
  enabled: true
  initialDelaySeconds: 30
  periodSeconds: 10
  failureThreshold: 30

extraEnvVars:
  - name: RABBITMQ_VM_MEMORY_HIGH_WATERMARK
    value: "0.4"
  - name: RABBITMQ_DISK_FREE_LIMIT_ABSOLUTE
    value: "2GB"
```

6.2 Deployment logic

- **Single replica** to start; you can later move to clustered RabbitMQ if needed.
- **Longhorn storageClass** ensures replicated, RWO storage.
- **Probes** (liveness, readiness, startup) prevent flapping during startup and recovery.
- **Security contexts** ensure correct permissions on the mounted volume.

Install example:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update
```

```
helm install rabbitmq bitnami/rabbitmq \
  -f values-longhorn.yaml \
  --namespace messaging \
  --create-namespace
```

6.3 Upgrade logic

1. Confirm backups and/or Longhorn snapshots exist.
2. Optionally scale down to 0 for conservative upgrades:

```
kubectl scale statefulset rabbitmq -n messaging --replicas=0
```

3. Upgrade the Helm release:

```
helm upgrade rabbitmq bitnami/rabbitmq \
  -f values-longhorn.yaml \
  -n messaging
```

4. Scale back up (if scaled down):

```
kubectl scale statefulset rabbitmq -n messaging --replicas=1
```

5. Verify:

- Pod Running
 - Volume Attached
 - Queues accessible
-

7. Appendices

7.1 Key commands

Engine deletion (for a given volume)

```
kubectl -n longhorn-system delete engine -l longhornvolume=<volume-name>
```

Restart instance managers

```
kubectl -n longhorn-system delete pods -l longhorn.io/component=instance-manager
```

Restart Longhorn managers

```
kubectl -n longhorn-system delete pods -l app=longhorn-manager
```

7.2 Replica selection logic

When using Salvage:

- Prefer replicas on nodes that did **not** crash mid-write.
- Prefer replicas with the most recent modification timestamps.
- Avoid replicas on disks that show I/O errors or SMART warnings.

If in doubt and you have backups, consider restoring instead of salvaging.

7.3 Salvage decision tree

Volume Faulted?

- └─ No → Do not use Salvage. Investigate other causes.
 - └─ Yes → Is data present on disk (replica dirs)?
 - └─ No → Restore from backup.
 - └─ Yes → Use Salvage:
 - Delete engine (if needed)
 - Salvage in UI
 - Select replicas
 - Attach volume
 - Restart workload
 - Verify data
-

If you'd like, I can now give you a minimal Pandoc command or VS Code extension suggestion to turn this Markdown into a PDF on your machine, so you're not blocked by partial file generation again.