Cinder Scheduler Timeout Runbook

A senior engineer's incident runbook for volumes stuck in `creating`, cinder-scheduler MessagingTimeout, and "No valid backend was found". DevOps AI ToolKit · devopsaitoolkit.com

A volume create flows API → scheduler → volume: `cinder-api` accepts the request and casts it over RPC to `cinder-scheduler`, which runs filters/weighers to pick a backend and casts to that `cinder-volume.<host>`. When volumes hang in `creating`, one hop on that path is broken. The two classic signatures split the diagnosis cleanly: `MessagingTimeout` means an RPC reply never came (scheduler or volume host is dead/slow, or RabbitMQ is the problem), while "No valid backend was found" / "Filtering removed all hosts" means the scheduler *ran* but every backend failed a filter (capacity, state, or a down volume service). Read the error first, then walk the path.

> Assumes a Kolla-Ansible deployment. Adjust container names for your tooling. Read-only checks first; reset volume state only with caution, and never blind-restart clustered stateful services.

1. Trace the API → Scheduler → Volume RPC Path

```
# Where did it stall? The API log shows the cast; the scheduler log shows the decision.
docker logs --tail=150 cinder_api 2>&1 \
  | grep -Ei "error|timeout|MessagingTimeout|scheduler"
docker logs --tail=150 cinder_scheduler 2>&1 \
  | grep -Ei "MessagingTimeout|No valid backend|Filtering removed all hosts|error"

# Which stuck volumes, and how long?
openstack volume list --status creating --long -c ID -c Name -c Status -c "Created At"
```

- `MessagingTimeout` logged by `cinder-api` naming `cinder-scheduler` → the scheduler never replied. Go to §2/§4.
- `No valid backend` / `Filtering removed all hosts` in the scheduler log → the scheduler is alive; a filter rejected everything. Go to §3/§6.
- Volumes creating for many minutes with a clean scheduler log → the cast may have died at RabbitMQ (§4) before the scheduler saw it.

2. Cinder Service List

```
# Every scheduler and volume backend, up/down and enabled/disabled
openstack volume service list -c Binary -c Host -c Status -c State -c "Updated At"

# Focus on down or disabled backends
openstack volume service list -f value | awk '$4!="up" || $3!="enabled"{print}'
```

- `cinder-scheduler` `down` → no scheduling happens at all; restart it (§7) after checking its log.
- `cinder-volume@<backend>` (e.g. `cinder-volume@lvm`) `down` → its host stopped reporting; that backend is invisible to the scheduler, which then finds "no valid backend".
- `State up` but `Status disabled` → someone disabled it (maintenance); re-enable with `openstack volume service set --enable <host> cinder-volume` once safe.

3. Cinder Scheduler Logs + Filters / Weighers

```
# Which filter removed the hosts? Cinder logs per-filter pass/reject counts.
docker logs --tail=200 cinder_scheduler 2>&1 \
  | grep -Ei "Filter .* returned|removed all hosts|Insufficient|CapacityFilter|weigh"

# Confirm the enabled filters/weighers
grep -RiE "scheduler_default_filters|scheduler_default_weighers|scheduler_max_attempts" \
  /etc/kolla/cinder-scheduler/*.conf
```

- `CapacityFilter returned 0 hosts` → backends are full or over-provisioned (§6).
- `AvailabilityZoneFilter` / `CapabilitiesFilter` rejecting everything → the requested volume type maps to a `volume_backend_name` no live host advertises. Check the volume type's `extra_specs` against the backend's reported capabilities.
- Scheduler runs but rejects all → this is a *filter/capacity* incident, not an RPC one; do not restart RabbitMQ.

4. RabbitMQ Queue / Consumers for Cinder

```
# Scheduler + volume queues: backed up or consumer-less?
docker exec rabbitmq rabbitmqctl list_queues name messages consumers \
  | grep -Ei "cinder-scheduler|cinder-volume"

# Is RabbitMQ itself healthy (partitions/alarms cause universal timeouts)?
docker exec rabbitmq rabbitmqctl cluster_status | grep -Ei "running_nodes|partitions"
docker exec rabbitmq rabbitmqctl list_connections state | grep -c blocked
```

- `cinder-scheduler` queue with `consumers = 0` → the scheduler isn't subscribed (down or disconnected) → API casts time out. Restart the scheduler (§7).
- `cinder-volume.<host>` (e.g. `cinder-volume.cinder-volume@lvm`) with `consumers = 0` → that volume host is disconnected; the scheduler may pick it and then the volume cast times out.
- `partitions` non-empty or `blocked > 0` → this is a broker/backpressure incident affecting *all* services; escalate to the messaging owner, do not blind-restart.

---

5. Storage Backend Health (LVM / Ceph / NetApp)

```
# LVM backend: is the VG present and does the container see it?
docker exec cinder_volume vgs cinder-volumes 2>/dev/null

# Ceph backend: cluster health + the volumes pool from the cinder container
docker exec cinder_volume ceph -s 2>/dev/null | grep -Ei "health|HEALTH_"
docker exec cinder_volume rbd -p volumes ls 2>/dev/null | head

# NetApp / other: confirm the management LIF / API endpoint is reachable
docker exec cinder_volume curl -sk -o /dev/null -w '%{http_code}\n' https://<netapp-mgmt-lif>/servlets/netapp.servlets.admin.XMLrequest_filer
```

- LVM `vgs` errors or missing VG → the volume driver can't provision; the backend reports itself down and the scheduler finds no host.
- Ceph not `HEALTH_OK` (e.g. `HEALTH_WARN` / `HEALTH_ERR` with full/near-full OSDs) → creates stall or fail; fix Ceph before Cinder.
- Backend API/LIF unreachable (non-200, timeout) → `cinder-volume` will flap `down`; this is a storage/network incident, not a Cinder bug.

---

6. Capacity / Over-Provisioning Filters

```
# Free vs. provisioned per backend, as the scheduler sees it (from its log capabilities)
docker logs --tail=300 cinder_volume 2>&1 \
  | grep -Ei "free_capacity_gb|total_capacity_gb|provisioned_capacity|allocated_capacity"

# The over-provisioning knobs
grep -RiE "max_over_subscription_ratio|reserved_percentage|scheduler_default_filters" \
  /etc/kolla/cinder-volume/*.conf /etc/kolla/cinder-scheduler/*.conf
```

- `free_capacity_gb` near 0 (or `0`) → genuinely full; the `CapacityFilter` correctly rejects it. Free space or add a backend.
- Thin provisioning with `provisioned_capacity` past `total * max_over_subscription_ratio` → over-committed; the filter blocks new volumes even with apparent free space.
- `reserved_percentage` set high leaves less schedulable capacity than raw free space suggests — factor it in before declaring a false alarm.

---

7. Restart Decision Tree

> **Restart the narrowest consumer.** Restarting `cinder_scheduler` or a single `cinder_volume` re-declares its RPC queue and re-reports capacity, fixing most `consumers=0`/stale-report cases without touching the broker. RabbitMQ and the storage backend are stateful — treat them as last resorts.

```
Volumes stuck in creating?
 ├─ Scheduler log: "No valid backend" / filters removed all? ── yes → capacity/state issue (§3/§5/§6), NOT a restart
 ├─ RabbitMQ partitioned or blocked? ──────────────────────── yes → fix broker FIRST, escalate; no blind restarts
 └─ MessagingTimeout, broker healthy
     ├─ cinder-scheduler down / queue consumers=0? ── yes → docker restart cinder_scheduler
     ├─ one cinder-volume@<backend> down?          ── yes → check backend (§5), then docker restart cinder_volume
     └─ backend up but reports stale capacity?     ── yes → docker restart cinder_volume (forces capability re-report)
```

```
# Targeted restarts (Kolla container names)
docker restart cinder_scheduler          # re-subscribes the scheduler RPC queue
docker restart cinder_volume             # only on the affected backend host

# Re-verify services report up before retrying a create
openstack volume service list -f value | awk '$4!="up"{print}'
```

---

8. Volume State Reset (use with caution)

> `cinder reset-state` **only edits the DB row — it does not clean up backend storage.** Use it to clear a volume wedged in `creating` *after* you have confirmed no create is still in flight; otherwise you risk orphaning an LVM LV or RBD image.

```
# Confirm nothing is still working the volume, then reset the DB state
openstack volume show <volume-id> -c status -c "os-vol-host-attr:host"
cinder reset-state --state error <volume-id>     # then delete, or --state available if truly created

# Clean up if the backend left an orphan (example: LVM)
docker exec cinder_volume lvs cinder-volumes | grep <volume-id>
```

- Reset to `error` then delete is the safe default for a phantom `creating` volume with no backing storage.
- Only reset to `available` if you have verified the backing volume actually exists and is healthy.
- Always pair a reset with a check for orphaned LVs/RBD images/LUNs so capacity accounting stays honest.

---

9. Copy/Paste One-Shot Triage

```
echo "== stuck volumes ==";   openstack volume list --status creating -f value | wc -l
echo "== services down ==";   openstack volume service list -f value | awk '$4!="up" || $3!="enabled"{print $1"@"$2}'
echo "== scheduler err ==";   docker logs --tail=120 cinder_scheduler 2>&1 | grep -Eic "MessagingTimeout|No valid backend|removed all hosts"
echo "== cinder queues ==";   docker exec rabbitmq rabbitmqctl list_queues name messages consumers | grep -Ei "cinder-scheduler|cinder-volume"
echo "== broker ==";          docker exec rabbitmq rabbitmqctl cluster_status | grep -Ei "partitions"
echo "== capacity ==";        docker logs --tail=200 cinder_volume 2>&1 | grep -Ei "free_capacity_gb" | tail -3
```

---

10. Incident Notes Template

```
INCIDENT: Cinder Scheduler Timeout / Volumes Stuck Creating
Started (UTC):          ____   Detected by: ____   Sev: ____
Impact:                 (volumes stuck ____ , tenants ____ , backend(s) affected ____)
Error signature:        [ ] MessagingTimeout  [ ] No valid backend  [ ] Filtering removed all hosts
Services:               [ ] cinder-scheduler ____  [ ] cinder-volume@____ state ____
RPC/RabbitMQ:           [ ] cinder-scheduler consumers ____  [ ] cinder-volume.<host> consumers ____
                        [ ] partitions ____  [ ] blocked conns ____
Filter that rejected:   ____ (CapacityFilter / Capabilities / AZ / other)
Backend health:         [ ] LVM VG ____  [ ] Ceph HEALTH ____  [ ] NetApp/API reachable ____
Capacity:               free_gb ____  provisioned ____  over_subscription_ratio ____
Action taken:           (service restarted / backend fix / reset-state + who + when)
State resets:           volume(s) ____ → ____  orphans cleaned? Y/N
Result:                 [ ] recovered @ ____   [ ] escalated to ____
Root cause:             ____
Follow-ups:             (capacity, over-provision alerts, backend HA, worker scaling) ____
```

DevOps AI ToolKit · devopsaitoolkit.com — practical AI workflows for real cloud engineers. Free to use for your own and your employer's internal work; please don't resell or republish the pack itself. Commands are provided as-is; always validate against your own environment before running in production. When you're stuck, paste the errors into the free AI Incident Response assistant at devopsaitoolkit.com/dashboard/incident-response.