OpenStack 504 Gateway Runbook Pack

A senior engineer's incident runbook for chasing HTTP 504s across the OpenStack control plane. DevOps AI ToolKit · devopsaitoolkit.com

A 504 Gateway Timeout means the proxy in front of OpenStack (usually HAProxy on the internal/external VIP) gave up waiting for a backend to respond. The proxy is the messenger — the slow backend is almost always Keystone, a service API (Nova/Cinder/Neutron), RabbitMQ, or MariaDB. Work top-to-bottom: confirm where the timeout fires, then walk the request path until a hop is slow.

> Assumes a Kolla-Ansible deployment. Adjust container names for your tooling. Run read-only checks first; only restart with a change record and, ideally, a second engineer.

---

1. 504 Triage Checklist (top-to-bottom)

1. **Confirm the layer.** Is the 504 from HAProxy (control-plane VIP) or an upstream edge proxy/CDN? Curl the API directly, bypassing the edge.
2. **Reproduce with timing.** `curl -w` the failing endpoint. Note *which* service path is slow (`:5000` Keystone, `:8774` Nova, `:8776` Cinder, `:9696` Neutron).
3. **HAProxy backend health.** Any backend DOWN or all sessions queued?
4. **Backend service up?** Container running, API worker listening, not OOM-killed.
5. **Keystone first.** Every API validates tokens against Keystone — if Keystone is slow, *everything* 504s.
6. **RabbitMQ.** Services block on RPC replies; a stuck queue looks like an API hang.
7. **MariaDB.** Slow queries / max_connections / Galera flow control stall API workers.
8. **Decide + act.** Restart the narrowest component that explains the symptom. Record it.

---

2. HAProxy Checks

```
# Which backends are DOWN / how long are timeouts set to?
docker exec haproxy sh -c 'echo "show stat" | socat stdio /var/lib/kolla/haproxy/haproxy.sock' \
  | awk -F, '{print $1,$2,$18,$5,$8}' | column -t   # pxname svname status qcur scur

# Effective timeouts (a 504 fires at timeout server/connect)
docker exec haproxy grep -E "timeout (connect|client|server)" /etc/haproxy/haproxy.cfg

# Live error/timeout log
docker logs --tail=100 haproxy 2>&1 | grep -Ei "sH|sD|timeout|no server"
```

- `sH` = server timeout waiting for headers (classic 504 cause).
- If one backend is DOWN, the fix is that backend, not HAProxy.
- Only raise `timeout server` as a stopgap — a 60s API call is a bug, not a config value.

---

3. Horizon Checks

```
docker ps --filter name=horizon --format '{{.Names}} {{.Status}}'
docker logs --tail=100 horizon 2>&1 | grep -Ei "error|timeout|gateway|keystone"

# Horizon 504 is almost always a slow API *behind* it — time the API Horizon calls:
curl -s -o /dev/null -w 'keystone: %{time_total}s\n' https://<vip>:5000/v3
```

- Horizon rendering slow but APIs fast → apache/mod_wsgi workers or memcached.
- Horizon slow *and* APIs slow → skip Horizon, chase the API.

---

4. Keystone Checks (do this early)

```
time openstack token issue -f value -c id      # baseline auth latency
curl -s -o /dev/null -w '%{http_code} %{time_total}s\n' https://<vip>:5000/v3
docker logs --tail=100 keystone 2>&1 | grep -Ei "error|timeout|deadlock|too many connections"
```

- Token issue > ~1s → Keystone→MariaDB (Fernet caching, DB latency) is the suspect.
- If Keystone is slow, fix it first; every other API depends on it.

---

5. Nova / Cinder / Neutron API Checks

```
# Are the service agents alive (this call itself goes over RPC)?
openstack compute service list
openstack volume service list
openstack network agent list

# Time each API independently to localize the slow one
for p in 8774 8776 9696; do
  curl -s -o /dev/null -w "port $p: %{http_code} %{time_total}s\n" https://<vip>:$p/
done

docker logs --tail=80 nova_api 2>&1     | grep -Ei "timeout|MessagingTimeout|error"
docker logs --tail=80 cinder_api 2>&1   | grep -Ei "timeout|MessagingTimeout|error"
docker logs --tail=80 neutron_server 2>&1 | grep -Ei "timeout|AMQP|error"
```

- `MessagingTimeout` in an API log → the API is healthy; its RPC peer (or RabbitMQ) is not. Go to §6.

---

6. RabbitMQ Checks

```
docker exec rabbitmq rabbitmqctl cluster_status
docker exec rabbitmq rabbitmqctl list_queues name messages consumers \
  | awk '$2>100 || $3==0 {print}'          # backed-up or consumer-less queues
docker exec rabbitmq rabbitmqctl list_connections name state | grep -i blocked
docker logs --tail=100 rabbitmq 2>&1 | grep -Ei "missed heartbeats|closing|partition"
```

- `messages` climbing with `consumers = 0` → the consuming service can't drain the queue.
- `blocked` connections → RabbitMQ hit a memory/disk watermark (backpressure). Check `rabbitmqctl status | grep -A5 alarms`.
- Network partition → cluster split-brain; needs careful recovery, not a blind restart.

---

7. MariaDB Checks

```
docker exec mariadb mysqladmin status
docker exec mariadb mysql -e "SHOW GLOBAL STATUS LIKE 'Threads_connected';
  SHOW GLOBAL VARIABLES LIKE 'max_connections';"
docker exec mariadb mysql -e "SHOW FULL PROCESSLIST;" | grep -vi sleep | head
# Galera health (clustered deployments)
docker exec mariadb mysql -e "SHOW STATUS LIKE 'wsrep_%';" \
  | grep -E "wsrep_(ready|cluster_size|local_state_comment|flow_control_paused)"
```

- `Threads_connected` near `max_connections` → API workers are starved; queries queue → 504.
- `wsrep_flow_control_paused` > 0 → a lagging Galera node is throttling writes cluster-wide.
- `wsrep_ready = OFF` → that node is not serving; point the API/VIP at a healthy node.

---

8. Kolla-Ansible Container Restart Commands

> **Restart only the component your evidence points at.** A blind `restart everything` turns a slow API into a full outage and destroys the diagnostic trail. Confirm the cause in §2–§7 first.

```
# Least-blast-radius restart of a single API service
docker restart nova_api           # or cinder_api / neutron_server / keystone

# Reconfigure a service from Kolla (config drift / cert change)
kolla-ansible -i <inventory> reconfigure --tags <service>   # e.g. --tags haproxy

# RabbitMQ / MariaDB are stateful — prefer targeted fixes; restart is a last resort
docker restart rabbitmq           # single-node only; NEVER blind-restart a partitioned cluster
```

---

9. Escalation Workflow

1. **0–10 min:** Localize the slow hop (§1–§2). Post the failing `curl -w` timings in the incident channel.
2. **10–20 min:** If a single backend, restart it (§8) with a change note. Re-test. If recovered, monitor 15 min.
3. **20–30 min:** If RabbitMQ/MariaDB/Galera (§6–§7), escalate to the messaging/DB owner — do **not** blind-restart clustered stateful services.
4. **30+ min or customer-facing:** Page the on-call lead, open a Sev-2, and start the incident notes template below.
5. **Stuck?** Bring in a second set of senior eyes: devopsaitoolkit.com/work-with-me, or paste the errors into the free AI Incident Response assistant at devopsaitoolkit.com/dashboard/incident-response.

---

10. Copy/Paste One-Shot Triage

```
VIP=<your-internal-vip>
echo "== API latency ==";           for p in 5000 8774 8776 9696; do curl -s -o /dev/null -w "port $p: %{http_code} %{time_total}s\n"
https://$VIP:$p/ ; done
echo "== HAProxy down backends ==="; docker exec haproxy sh -c 'echo "show stat" | socat stdio /var/lib/kolla/haproxy/haproxy.sock' | awk -F,
'$18=="DOWN"{print $1"/"$2}'
echo "== Rabbit backed-up queues ==="; docker exec rabbitmq rabbitmqctl list_queues name messages consumers | awk 'NR>1 && ($2>100||$3==0){print}'
echo "== Rabbit blocked conns ==";   docker exec rabbitmq rabbitmqctl list_connections state | grep -c blocked
echo "== MariaDB conns ==";          docker exec mariadb mysql -N -e "SHOW GLOBAL STATUS LIKE 'Threads_connected'; SHOW GLOBAL VARIABLES LIKE
'max_connections';"
echo "== Galera ==";                 docker exec mariadb mysql -N -e "SHOW STATUS LIKE 'wsrep_ready'; SHOW STATUS LIKE
'wsrep_flow_control_paused';"
```

11. Incident Notes Template

```
INCIDENT: OpenStack 504 Gateway Timeout
Started (UTC):            ____   Detected by: ____   Sev: ____
Impact:                   (which endpoints / tenants / % of requests)
Layer confirmed:          [ ] HAProxy VIP  [ ] edge/CDN
Slow hop (from curl -w):  [ ] Keystone [ ] Nova [ ] Cinder [ ] Neutron [ ] Horizon
Backend evidence:         [ ] HAProxy backend DOWN: ____
                          [ ] RabbitMQ queue backlog: ____  blocked conns: ____
                          [ ] MariaDB conns __/__  Galera paused: ____
Action taken:             (component restarted / config changed + who + when)
Result:                   [ ] recovered @ ____   [ ] escalated to ____
Root cause:               ____
Follow-ups:               (timeouts, capacity, alerts, autoscaling) ____
```