Lenel Server Architecture – Load Balancing & Clustering Implementation Guide

This document provides concrete, implementation-level guidance for load balancing, clustering, and high-availability design in a Lenel / OnGuard server architecture. It expands on architectural recommendations with specific patterns, decision criteria, and operational considerations intended for infrastructure and systems engineers.

1. Core Design Principles

- Prefer horizontal scale-out (active/active) for stateless or mostly-stateless services.
- Use clustering only where shared state or message durability is required.
- Use active/passive HA when services assume singleton behavior or cannot tolerate parallel execution.
- Never rely on load balancers as a substitute for database-native high availability.

2. Services Appropriate for Load Balancing

OpenAccess, web portals, and API-facing services are prime candidates for load balancing.

- Deploy a minimum of two identical application servers.
- Place services behind a Layer 7 load balancer when HTTP/S is used.
- Use least-connections or similar adaptive algorithms.
- Avoid session stickiness unless explicitly required; enable only after validation.

Health Checks

- Prefer application-level HTTP health endpoints over TCP port checks.
- Health checks should validate minimal dependency access (DB connectivity, thread pool availability).
- Mark backends unhealthy based on response failures or latency thresholds, not just connectivity.

3. Services That Require Clustering

Message brokers and stateful coordination services require clustering rather than traditional load balancing.

- RabbitMQ should be deployed as a minimum three-node cluster using quorum queues.
- Clients should connect via a virtual IP or DNS alias to any node.
- Ensure mirrored or quorum-based durability for all critical queues.
- Plan rolling upgrades and node loss scenarios in advance.

4. Services That Should Use Active/Passive HA

- Core Lenel services that maintain exclusive locks or assume singleton execution.
- Services that perform non-idempotent command dispatch or hardware control.
- Components with vendor-imposed licensing or role restrictions.

Active/passive implementations may rely on Windows Failover Clustering or application-level failover mechanisms. Only one node should actively process workloads at any time.

5. Database High Availability

- Deploy SQL Server on dedicated hosts, isolated from application workloads.
- Use SQL Always On Availability Groups or Failover Cluster Instances.
- Expose SQL via listener endpoints rather than load balancer VIPs.
- Ensure storage performance meets transaction log and tempdb demands.

6. Load Balancer High Availability

- Deploy redundant load balancer nodes.
- Use VRRP/Keepalived or vendor HA features to float VIPs.
- Validate ARP/neighbor updates and failover convergence times.

7. Failure Testing and Validation

- Terminate backend services and verify graceful traffic re-routing.
- Simulate node loss and observe client recovery behavior.
- Force database failovers and monitor application reconnect handling.
- Test load balancer failover scenarios explicitly.

8. Recommended Implementation Order

- Duplicate and load balance OpenAccess and web-facing services.
- Add load balancer redundancy.
- Separate and scale integration or middleware services.
- Introduce broker clustering if required.
- Evaluate core services for HA pairing only after other layers are stable.

Result: This approach reduces single points of failure, enables controlled horizontal scaling, and provides clear operational boundaries for Lenel system growth and troubleshooting.