OpenStack Operations Toolkit

12 checklists and runbooks for running OpenStack in production — Free Sample

DevOps AI ToolKit · devopsaitoolkit.com

Contents

#	Document
OPS-01	OpenStack Production Readiness Checklist
OPS-07	RabbitMQ Troubleshooting Checklist

OPS-01: OpenStack Production Readiness Checklist

Use when: before a cluster carries production workloads, and once a quarter afterwards. **Time:** 2–4 hours for a first pass on an existing cluster. **Output:** a list of gaps with an owner against each. Items you consciously accept are recorded as accepted, not left blank — an undocumented accepted risk is indistinguishable from an oversight.

This checklist is deliberately opinionated about *order*. Control plane before data plane, because a cluster whose API is unreliable cannot be assessed at all. Recovery before performance, because a fast cloud you cannot restore is worth less than a slow one you can.

---

1. Control plane

- ☐ **Three controllers, not two.** A two-node control plane cannot form a majority, so a single failure stops the cluster rather than degrading it. If you have two, know that you have a highly available *service* on a non-highly-available *quorum*.
- ☐ **Every API endpoint is behind the load balancer**, including the internal endpoints. Verify with `openstack endpoint list` — a hard-coded controller hostname in an internal endpoint is a single point of failure that only appears during a controller outage.
- ☐ **HAProxy health checks test the API, not the port.** A TCP check passes against a hung service. Check an actual endpoint that requires the service to be functioning.
- ☐ **Database is clustered and the cluster is not in single-node mode.** Confirm with `SHOW STATUS LIKE 'wsrep_cluster_size'` — a Galera cluster running one node reports success on every write and has no redundancy.
- ☐ **RabbitMQ is clustered with a partition-handling policy set.** Default `ignore` leaves you with a split brain that must be resolved by hand.
- ☐ **Fernet keys are identical on every controller** and rotated by a scheduled job, not manually. Inconsistent keys produce authentication failures on some nodes only, which is much harder to diagnose than a total failure.
- ☐ **NTP is synchronised and monitored.** Token validation is time-sensitive. Clock drift produces authentication errors that look nothing like a time problem.

2. Networking

- ☐ **L3 agents are redundant** — either multiple agents with HA routers, or DVR. A single L3 agent means every router loses its gateway when that node reboots.
- ☐ **The external network has enough floating IPs** for peak, not for today. Check `openstack ip availability list`.
- ☐ **MTU is consistent end to end.** Overlay networks lose bytes to encapsulation, and a mismatch produces the classic "small requests work, large ones hang" failure that gets blamed on the application for days.
- ☐ **Security-group defaults are documented.** The default group allows all egress and no ingress; teams that do not know this open ports they did not intend to.
- ☐ **DNS resolution works from inside instances**, not just from the controllers.

3. Storage

- ☐ **The Cinder backend has monitored free capacity** with an alert threshold that leaves time to act, not one that fires when it is already full.
- ☐ **Volume backups are configured and a restore has actually been tested.** An untested backup is a hypothesis.
- ☐ **Glance image store is redundant.** If images live on one node, that node's failure prevents every new instance from booting even though the cloud looks healthy.
- ☐ **Ephemeral vs persistent storage is documented for each flavour**, so nobody discovers the distinction during an incident.

4. Capacity and scheduling

- ☐ **Allocation ratios are set deliberately**, not left at defaults. Defaults overcommit CPU aggressively and memory not at all, which is rarely what a given workload profile wants.
- ☐ **At least one compute node's worth of spare capacity** exists, so a node can be evacuated for maintenance without refusing new instances.
- ☐ **Host aggregates and availability zones are documented**, including which flavours can land where. Undocumented aggregates are the most common cause of "No valid host was found" on a cluster that visibly has capacity.

5. Observability

- ☐ **Every OpenStack service exports metrics** and the control plane has a dashboard someone actually looks at.
- ☐ **Alerts exist for:** API error rate, RabbitMQ queue depth, database replication lag, Cinder/Glance capacity, agent liveness, certificate expiry.
- ☐ **Logs are centralised.** Debugging a failed instance spawn across five nodes with `ssh` and `grep` is a solved problem you should not still be solving.
- ☐ **Alert routing has been tested** by deliberately firing one. An alert that reaches nobody is worse than no alert, because it creates false confidence.

6. Recovery

- ☐ **The control-plane database is backed up on a schedule**, off the controllers.
- ☐ **A restore has been performed into a scratch environment**, with the elapsed time recorded. That number is your actual RTO; anything else is an estimate.
- ☐ **The Kolla-Ansible (or equivalent) configuration is in version control**, including inventory. A cluster you cannot rebuild from a repository is a cluster you cannot rebuild.
- ☐ **Someone other than the primary operator has successfully performed a routine maintenance task.** Single-person operational knowledge is the most common real availability risk in a private cloud, and it does not show up on any dashboard.

Scoring this honestly

Count only boxes you have *verified*, not boxes you believe. The value of this checklist is entirely in the difference between those two, and that difference is usually largest in the Recovery section — which is also the section where being wrong is most expensive.

OPS-07: RabbitMQ Troubleshooting Checklist

Severity guidance: P1 almost always. RabbitMQ is the control plane's nervous system; when it degrades, every service degrades at once. **The diagnostic signature:** several *unrelated* services timing out simultaneously. A single service failing is a service problem. Everything failing together is the bus until proven otherwise.

First 60 seconds

```
rabbitmqctl cluster_status
```

- ☐ `running_nodes` **contains every node.** A missing node is a broker down.
- ☐ `partitions` **is empty.** A non-empty partition list is a split brain, and it is the single most important thing on this page.

```
rabbitmqctl list_queues name messages consumers | sort -k2 -rn | head -20
```

- ☐ **No queue has many messages and zero consumers.** That pattern names the service that stopped listening — restart *that service*, not RabbitMQ.
- ☐ **No queue is growing without bound.** A queue that only grows means production exceeds consumption; adding brokers will not help.

```
rabbitmqctl status | grep -A5 'memory\|disk_free'
```

- ☐ **Memory is below the high watermark.** Above it, RabbitMQ blocks publishers — which presents to OpenStack as every API call hanging, with no error anywhere.
- ☐ **Disk free is above the limit.** Same behaviour: publishers blocked, cluster apparently dead.

The partition decision

A partition means nodes disagreed about cluster membership and each side continued independently. You must choose which side is authoritative — there is no automatic correct answer, which is exactly why the default handling mode does nothing.

- ☐ Identify which partition has the majority of nodes.
- ☐ Confirm which side the OpenStack services are actually connected to.
- ☐ Restart the **minority** side so it rejoins. Restarting the majority discards the state most clients are using.

```
# On the minority node:
rabbitmqctl stop_app
rabbitmqctl start_app
rabbitmqctl cluster_status   # confirm partitions is now empty
```

Set a partition handling policy so this is not a manual decision at 3am. `pause_minority` is the usual right answer for a three-node cluster: the minority stops serving rather than diverging.

Symptom → cause table

Symptom	Likely cause	Check
Every OpenStack service times out at once	Partition, or publishers blocked by a watermark	`cluster_status`, `status | grep memory`
One service times out; others fine	That service's consumer died	`list_queues` for a queue with 0 consumers
Agents show `up` with stale heartbeats	Messages delayed, not lost	Queue depth on the agent's queue
RabbitMQ restarts repeatedly	Memory watermark or disk limit	`status`, host `free -h` and `df -h`
Reply queues accumulate	RPC callers timing out and abandoning replies	`list_queues | grep reply_`

What not to do

- ☐ **Do not restart OpenStack services against a partitioned broker.** Every restart reconnects and republishes, adding load to a cluster that cannot process it. Fix the partition first.
- ☐ **Do not `reset` a node to make it rejoin** unless you accept losing its queue state. `reset` erases the node's data.
- ☐ **Do not raise the memory watermark to stop the blocking.** The watermark is protecting the host; raising it converts a degradation into an OOM kill.
- ☐ **Do not purge queues to "clear the backlog"** without understanding what is in them. Those are in-flight control-plane operations.

Prevention

- ☐ Set an explicit `cluster_partition_handling` policy. The default `ignore` guarantees a manual incident.
- ☐ Alert on: partition present, queue depth, consumer count reaching zero on known queues, memory and disk watermarks approached.
- ☐ Alert on **heartbeat age** for OpenStack agents, not only their up/down state — delayed heartbeats are the earliest visible bus symptom.
- ☐ Give RabbitMQ its own disk headroom. It is unusually sensitive to the host filling up, and the failure mode is opaque.
