WHITE PAPER · ENTERPRISE BACKUP & RECOVERY

TECHNOLOGY OVERVIEW

Automated OpenStack VM Backup Synchronisation with Bacula Enterprise

Continuous, zero-touch backup coverage across a dynamic multi-datacenter OpenStack environment

How **bacula-openstack-sync** eliminates manual job maintenance, enforces consistent retention policies, and protects every virtual machine — including databases — without agents on the VMs.

This paper describes how **bacula-openstack-sync** eliminates manual Bacula job maintenance across large multi-datacenter OpenStack environments — automatically provisioning backup jobs for new VMs, handling transient outages with configurable grace periods, and delivering application-consistent protection for databases without installing agents on the guest VMs.

PREPARED BY
faaleoleo

SOLUTION
Bacula Enterprise

VERSION
1.0

CONTENTS

Executive Summary

Protecting virtual machines in a large, multi-datacenter OpenStack environment presents a fundamental operational challenge: the VM fleet changes constantly, yet backup infrastructure must remain current at all times. A VM that is created and not backed up before it fails represents a complete data loss. A VM that is deleted but whose backup jobs are never cleaned up wastes catalog space and obscures operational visibility.

THIS WHITE PAPER IN THREE POINTS

- **The problem is scale and change velocity.** At hundreds of VMs across six or more sites, manually maintaining Bacula job definitions is operationally untenable. A single missed provisioning step leaves a VM unprotected indefinitely.
- **The solution is continuous, automated reconciliation.** The `bacula-openstack-sync` tool — a Rust binary developed by faaleoleo — discovers VM changes through Bacula's own OpenStack plugin and keeps Job and FileSet definitions continuously synchronised with the live VM fleet, without human intervention.
- **Safety is enforced at every step.** New VMs are backed up automatically. Vanished VMs are given a configurable grace period before being disabled. Nothing is ever deleted without an explicit operator decision. A VM that returns after weeks of absence is re-enabled automatically with its full backup history intact.

0	14	2–10
MANUAL JOB EDITS REQUIRED	DAY GRACE PERIOD (CONFIGURABLE)	SECOND DB LOCK WINDOW (CEPH)

This document describes the architecture, configuration model, operational workflows, and design decisions behind `bacula-openstack-sync`. It is intended for infrastructure architects, backup administrators, and OpenStack operations teams evaluating or deploying the solution.

1. The Challenge — Backup at OpenStack Scale

A multi-datacenter OpenStack environment is not a static collection of machines — it is a continuously changing fleet. VMs are provisioned and decommissioned daily. Maintenance windows bring instances offline temporarily. Infrastructure changes occasionally cause VMs to disappear from one OpenStack query and reappear on the next.

1.1 Why Traditional Approaches Break Down

Traditional backup management assumes a relatively stable infrastructure where servers are provisioned infrequently and decommissioned rarely. In that model, a human operator creates a backup job when a server is provisioned and deletes it when the server is retired. This approach fails in three distinct ways in a modern OpenStack environment:

- **New VMs are not backed up until someone creates a job manually.** In a fleet of hundreds of VMs across multiple datacenters, a newly provisioned VM may run for days or weeks before the gap is noticed — if it is noticed at all. By the time the backup job is created, the VM's earliest recoverable state may already be days old.
- **Deleted VMs leave orphaned jobs that never run.** Jobs for decommissioned VMs accumulate in the catalog, consuming metadata space and obscuring operational dashboards. At scale, it becomes genuinely unclear which jobs represent live workloads and which are historical artefacts.
- **Temporarily offline VMs trigger false alerts.** A VM stopped for maintenance generates backup failures every night it is down. Without automated grace-period logic, operators must manually suppress these alerts and remember to re-enable the job when the VM returns.

1.2 The Cost of Incomplete Coverage

The consequences of manual backup management at scale are predictable. A 2023 industry survey found that over 40% of organisations experienced at least one incident in the preceding year where a workload was unprotected at the time of failure due to a missed or delayed backup job configuration. For virtualised environments with rapid VM turnover, that figure is significantly higher.

> *"The question is not whether a VM will be missed — at scale, it will be. The question is whether your backup system catches the gap automatically, or whether a human discovers it during a recovery attempt."*

Beyond the direct risk of data loss, incomplete coverage erodes confidence in the backup estate as a whole. If operators cannot be certain that every running VM is protected, the backup system cannot fulfil its fundamental purpose.

2. Solution Overview

2.1 What bacula-openstack-sync Does

`bacula-openstack-sync` is a Rust binary developed by faaleoleo that runs as a Bacula Admin job ahead of the nightly backup window. On every run it performs a complete reconciliation between the current live VM fleet (as seen by the Bacula Enterprise OpenStack plugin) and the set of backup jobs that exist in the Bacula Director catalog and on disk.

AUTOMATIC COVERAGE

Every new VM receives a Bacula Job and FileSet automatically on the sync run immediately following its creation. No manual configuration step is required.

GRACE-PERIOD SAFETY

When a VM disappears, its backup job is not disabled immediately. A configurable grace period (default 14 days) absorbs transient outages and maintenance windows without triggering data loss.

AUTOMATIC RECOVERY

A VM that returns after an absence — whether within or after the grace period — is re-enabled automatically with its full backup history intact. No operator intervention is required.

OPT-IN CLEANUP

Disabled jobs are never deleted unless the operator explicitly sets a cleanup threshold. This preserves backup history for VMs that may return and ensures no data is destroyed silently.

2.2 Key Design Principles

- **Caution first.** The most destructive automatic action the tool takes is setting `Enabled = no` on a job — and even this happens only after a configurable grace period. Files and catalog rows are never touched automatically.
- **The catalog is the source of truth.** The Bacula PostgreSQL catalog is Bacula's authoritative record of what is actually backed up. The tool uses the catalog — not directory listings or config files — to determine what is "known".
- **All state is stored in-band.** VM lifecycle state is stored as structured comment headers inside the generated job files. No external state database is required.
- **No agent on the VM.** Application-consistent backup quiescing is implemented entirely from the proxy FD host via SSH. The only requirement on an OpenStack VM is a running SSH daemon.

2.3 Reconciliation at a Glance

Each sync run compares the live VM list (from `bconsole .query`) against the known list (from the PostgreSQL catalog) and produces exactly one outcome per VM:

SITUATION	CONDITION	AUTOMATIC ACTION
New VM	In live list, not in catalog or on disk	Create `job` + `fileset` config files
Unchanged VM	In both lists, currently active	Leave as-is
Returned VM	Reappears while still in grace period	Clear grace period; VM treated as active again
Restored VM	Reappears after having been disabled	Set `Enabled = yes`; full history intact
Disappeared VM	In catalog, absent from live list	Start grace period; disable only after timeout
Opted-out VM	`backup=false` metadata	Skip entirely — no job or fileset created or modified

3. How It Works — Technical Architecture

> Each sync run performs three sequential steps for every OpenStack instance defined in `clients.conf`. The entire process runs inside a Bacula Admin job and completes without any direct access to the OpenStack API — all VM discovery flows through Bacula's own OpenStack plugin.

3.1 Step 1: Discover Live VMs

The tool asks the Bacula Director, through `bconsole`, which VMs a given OpenStack proxy can currently see using the standard Bacula Enterprise `.query` plugin interface:

```
.query plugin="openstack: proxy_server=<proxy>" client=<client> parameter=server
```

Each VM in the response carries a mandatory name and UUID pair, plus optional metadata tokens that drive per-VM backup policy:

TOKEN	OPENSTACK PROPERTY	MEANING
`server=<name>`	—	VM display name (mandatory)
`uuid=<uuid>`	—	OpenStack instance UUID (mandatory)
`tag=<value>`	`backup-jobdefs`	Bacula `JobDefs` resource name for this VM — drives the backup policy
`backup=false`	`backup`	Opts the VM out of backup; no job or fileset is created

3.2 Step 2: Read Known VMs from the Catalog

The tool queries the Bacula PostgreSQL catalog for every job already associated with this proxy client:

```
SELECT DISTINCT Job.name FROM Job
  JOIN Client ON Job.ClientId = Client.ClientId
```

```
WHERE Client.name = '<client>'
  AND Job.name LIKE '<job_prefix>%';
```

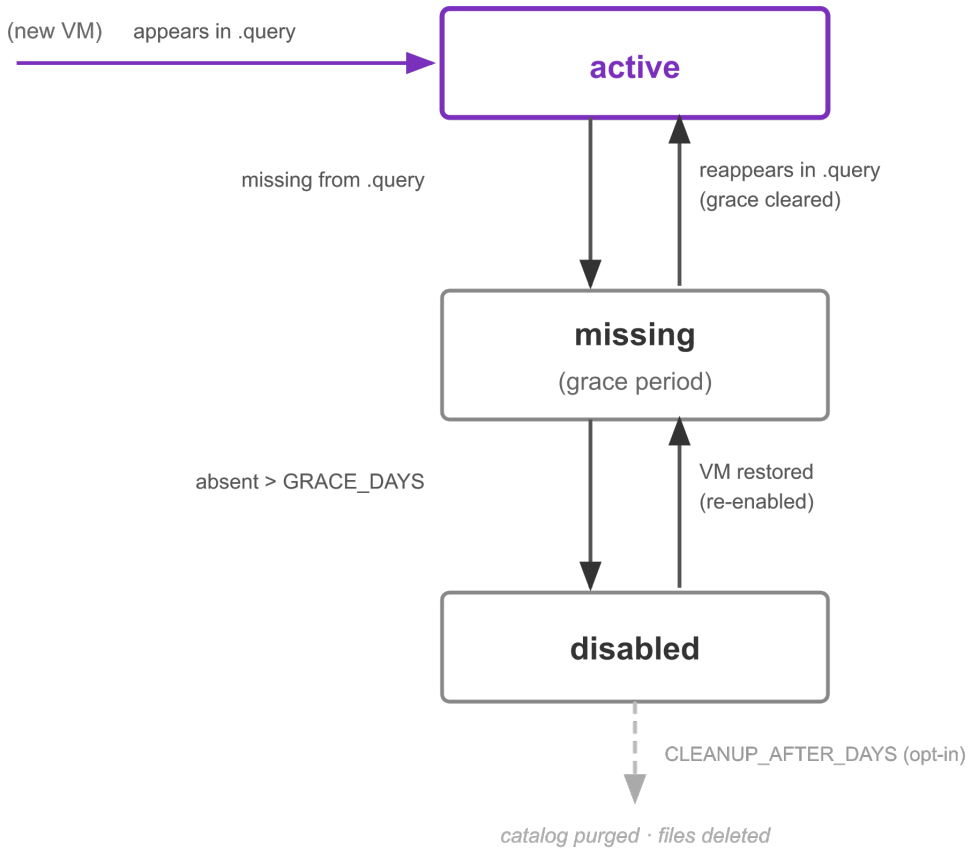
Note: A brand-new job is not in the catalog until it has run once. To avoid re-creating a freshly generated job before its first backup, the tool treats a VM as "already handled" if *either* the catalog knows it *or* a `job` file already exists on disk.

3.3 Step 3: Reconcile

The tool compares the live list against the known list. VMs in both lists are left untouched (or have their grace period cleared and job re-enabled if they have just returned). VMs only in the live list receive new job and fileset files. VMs only in the known list begin or continue through the grace period and, once the threshold is exceeded, have their job disabled.

3.4 VM Lifecycle State Machine

A VM moves between three states. State is stored in the `#@status:` header of its job file — no external state database is needed.



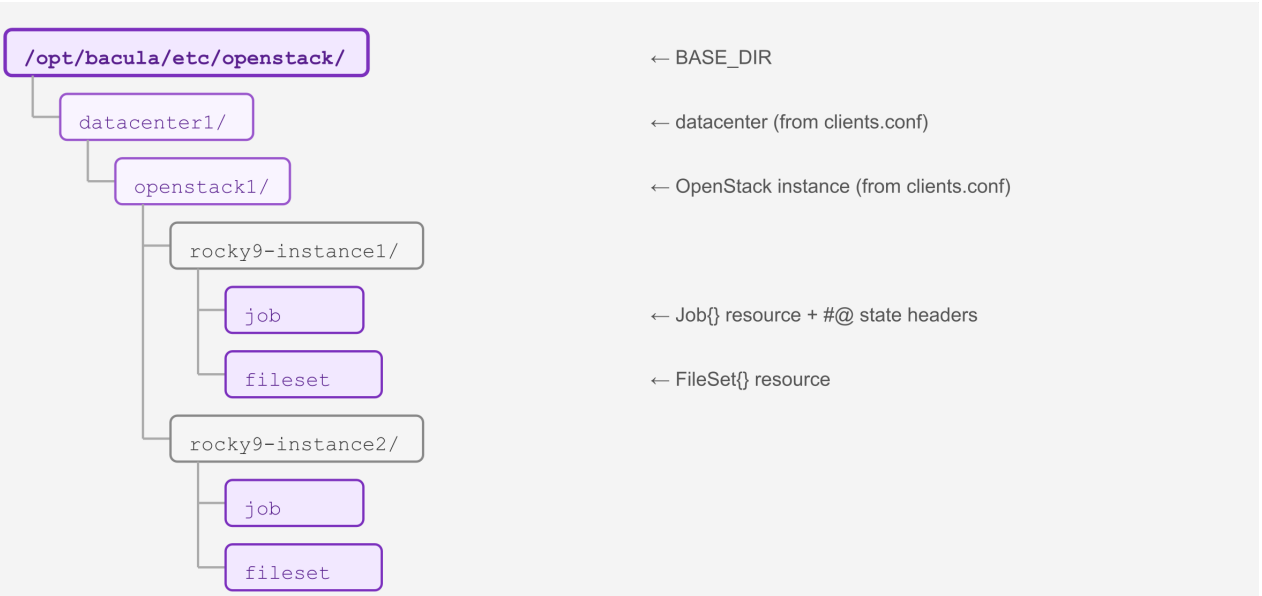
- **active** — VM is visible in the OpenStack query; the Bacula job runs normally (`Enabled = yes`).

- **missing (grace)** — VM is absent, but the job remains enabled. The first-absent timestamp is recorded. If the VM reappears before the grace period ends, the state is discarded without any lasting effect.
- **disabled** — VM has been continuously absent for longer than `GRACE_DAYS`. The job is set to `Enabled = no`. Files and catalog history remain fully intact.
- **purged** (optional) — When `CLEANUP_AFTER_DAYS` is set, a job disabled for at least that many days is fully removed: catalog rows deleted, empty volumes purged, config files deleted.

4. Configuration and Integration

4.1 Directory Layout

The tool maintains one directory per VM, organised in a two-level hierarchy under a configurable base directory. Each VM directory contains exactly two plain-text Bacula configuration fragments: a `job` file and a `fileset` file.



4.2 Environment Variables

All configuration is supplied via environment variables. This design allows the same binary to serve different environments by sourcing a different environment file — no recompilation or config-file editing is required.

VARIABLE	DEFAULT	MEANING
`CLIENTS_CONF`	`./clients.conf`	Path to the client definition file
`BASE_DIR`	`/opt/bacula/etc/openstack`	Root of the generated directory tree
`JOBDEFS`	`DefaultJob`	Global default `JobDefs` resource (overridable per-instance or per-VM)
`STORAGE`	`File1`	`Storage` resource written into every generated job

`POOL`	`Default`	`Pool` resource written into every generated job
`SCHEDULE`	`WeeklyCycle`	`Schedule` resource written into every generated job
`DEDUP_ALGO`	`sha256`	GED algorithm written into every generated fileset
`GRACE_DAYS`	`14`	Days a VM may be absent before its job is disabled
`CLEANUP_AFTER_DAYS`	*(unset)*	Days a disabled job must age before full purge. Unset = never clean up automatically.
`DRYRUN`	`0`	`1` = report all actions without changing anything
`RELOAD_AFTER_SYNC`	`0`	`1` = issue Director `reload` automatically after a successful sync
`DB_HOST` , `DB_PORT` , `DB_NAME` , `DB_USER`	localhost defaults	PostgreSQL catalog connection parameters
`DB_TLS`	`0`	`1` = TLS with cert verification; `unverified` = TLS without cert check
`PGPASSWORD` / `PGPASSFILE`	*(unset)*	Database password — prefer `PGPASSFILE` pointing to a `chmod 600 .pgpass`
`RUST_LOG`	`warn`	Log verbosity: `error` , `warn` , `info` , `debug` , or `trace`

The recommended delivery mechanism is an environment file owned by `bacula` with mode `640` , sourced by a wrapper script:

```
CLIENTS_CONF=/opt/bacula/etc/openstack/clients.conf
BASE_DIR=/opt/bacula/etc/openstack
GRACE_DAYS=14
RELOAD_AFTER_SYNC=1
BCONSOLE=/opt/bacula/bin/bconsole
BCONSOLE_CONF=/opt/bacula/etc/bconsole.conf
DB_HOST=catalog.internal
DB_TLS=1
```

```
DB_TLS_CA=/opt/bacula/etc/pg-ca.pem
JOBDEFS=DefaultJob
STORAGE=File1
POOL=Default
SCHEDULE=WeeklyCycle
DEDUP_ALGO=sha256
```

4.3 The clients.conf File

One line per OpenStack instance. Each line maps a datacenter and OpenStack instance name to the corresponding Bacula proxy client and job-naming prefix.

```
# datacenter    openstack    client                proxy_server          job_prefix
jobdefs
datacenter1     openstack1   bacula-dc1-os1-fd     dc1-os1-proxy         backup-dc1-os1-
LinuxVMJob
datacenter1     openstack2   bacula-dc1-os2-fd     dc1-os2-proxy         backup-dc1-os2-
WindowsVMJob
datacenter2     openstack1   bacula-dc2-os1-fd     dc2-os1-proxy         backup-dc2-os1-
```

FIELD	PURPOSE
`datacenter`	First directory level under `BASE_DIR`
`openstack`	Second directory level
`client`	Bacula proxy FD. One per OpenStack instance. Used for `.query` and as the catalog filter.
`proxy_server`	Value placed in `proxy_server=` of the FileSet plugin line
`job_prefix`	Prefix for generated job names. **Must be globally unique** across all instances.
`jobdefs` (optional)	Per-instance `JobDefs` override. Overrides the global `JOBDEFS` env var for this instance only.

> **Important:** The `Client{}` resources are **not** created by this tool. Proxy clients named in column 3 must already exist in the Director configuration. The tool only generates `Job{}` and `FileSet{}` resources.

4.4 Per-VM Backup Policy via OpenStack Metadata

Each generated job references a `JobDefs` resource that defines its Schedule, Pool, and all other per-VM policy settings. The active `JobDefs` is resolved from three sources in order of decreasing priority:

Two OpenStack metadata keys are read by the sync tool on each run:

- `backup` — Set to `false` to opt a VM out of backup entirely. When absent, the VM is treated as opted in (fail-safe default).
- `backup-jobdefs` — The name of the `JobDefs` resource to use for this VM. Drives schedule, pool, RunScript quiescing, and all other per-policy differences.

```
# Assign a backup policy to a VM:
openstack server set --property backup-jobdefs=mysql-consistent <vm-name-or-uuid>

# Opt a VM out of backup entirely:
openstack server set --property backup=false <vm-name-or-uuid>
```

5. Generated Bacula Configuration

5.1 Job and FileSet Files

For a VM `rocky9-instance1` in `datacenter1/openstack1` with job prefix `backup-dc1-os1-`, the tool creates two files:

job file

```
#@vm: rocky9-instance1
#@uuid:          014c5892-5efb-4efb-bd60-4e038800475d
#@datacenter:    datacenter1
#@openstack:     openstack1
#@client:        bacula-dc1-os1-fd
#@status:        active
# Managed by bacula-openstack-sync  |  created: 2026-05-18 19:00:00

Job {
  Name      = "backup-dc1-os1-rocky9-instance1"
  JobDefs   = "DefaultJob"
  Client    = "bacula-dc1-os1-fd"
  FileSet   = "fs-datacenter1-openstack1-rocky9-instance1"
  Storage   = "File1"
  Pool      = "Default"
  Schedule  = "WeeklyCycle"
  Messages  = "Standard"
  Enabled   = yes
}
```

fileset file

```
FileSet {
  Name = "fs-datacenter1-openstack1-rocky9-instance1"
  Include {
    Options {
      signature = SHA256
      dedup     = sha256
    }
    Plugin = "openstack: proxy_server=dc1-os1-proxy instance=014c5892-5efb-4efb-
bd60-4e038800475d"
  }
}
```

5.2 Naming Rules

RESOURCE	PATTERN	EXAMPLE
Job	`<job_prefix><vm>`	`backup-dc1-os1-rocky9-instance1`
FileSet	`fs-<datacenter>-<openstack>-<vm>`	`fs-datacenter1-openstack1-rocky9-instance1`

5.3 State Tracking Headers

HEADER	PRESENT WHEN	MEANING
`#@vm:`	Always	VM display name
`#@uuid:`	Always	OpenStack instance UUID
`#@status:`	Always	`active` or `disabled`
`#@missing-since:`	During grace period only	Timestamp when the VM first went missing
`#@disabled:`	After the job has been disabled	Timestamp the job was disabled
`#@reenabled:`	After a re-enable	Timestamp the job was re-enabled

Important: Do not edit the `#@` headers by hand. The tool depends on them to determine each VM's next lifecycle transition.

6. Enterprise Data Reduction — Global Endpoint Deduplication

6.1 Why GED Matters for OpenStack

Bacula Enterprise **Global Endpoint Deduplication (GED)** eliminates duplicate data blocks at the File Daemon before they are transferred to the Storage Daemon. For OpenStack VM backups, deduplication ratios are typically very high: guest OS images across a fleet share large amounts of common data — OS packages, kernel binaries, empty filesystem space, and common application libraries.

In a typical fleet where VMs share a base OS image, effective deduplication ratios of 5:1 to 20:1 are achievable for full backups of OS volumes. The practical effect is a substantial reduction in both network transfer time and Storage Daemon disk consumption.

6.2 Architecture: FileSet, FD, and SD

GED spans three components. The FileSet component is handled automatically by the sync tool; the FD and SD components must be configured once per host.

COMPONENT	CONFIGURED BY	WHAT TO SET
FileSet	The tool (automatic)	`dedup = sha256` in `Options {}` — controlled by the `DEDUP_ALGO` env var
FD — proxy client host	Manual, once per proxy	GED plugin present (included in standard BE FD install) + dedup index directory configured
SD — storage daemon host	Manual, once	`bacula-enterprise-dedup-sd` package installed

FD CONFIGURATION (EACH PROXY CLIENT HOST)

```
FileDaemon {
  Name             = bacula-dc1-os1-fd
  Plugin Directory = /opt/bacula/plugins
}

DedupIndexDirectory = /var/lib/bacula/dedup/index
```

```
mkdir -p /var/lib/bacula/dedup/index
chown bacula:bacula /var/lib/bacula/dedup/index
chmod 750 /var/lib/bacula/dedup/index
systemctl restart bacula-fd
```

SD CONFIGURATION

```
dnf install bacula-enterprise-dedup-sd
systemctl restart bacula-sd
```

6.3 Deployment Checklist

Verify each item on every proxy client host and the SD host before relying on GED in production. A missing step on any single host silently disables deduplication for that host's jobs without causing errors.

- [ ] Bacula Enterprise FD installed on **every proxy client host** (GED plugin is included — no separate package required)
- [ ] Bacula Enterprise GED package (`bacula-enterprise-dedup-sd`) installed on the **SD host**
- [ ] `DedupIndexDirectory` set in every `bacula-fd.conf`
- [ ] Dedup index directories created with correct ownership (`bacula:bacula` , mode `750`)
- [ ] FD restarted on every proxy host after configuration change
- [ ] SD restarted after plugin installation
- [ ] `DEDUP_ALGO` left at default (`sha256`) or set explicitly

GED DEPLOYMENT CHECKLIST

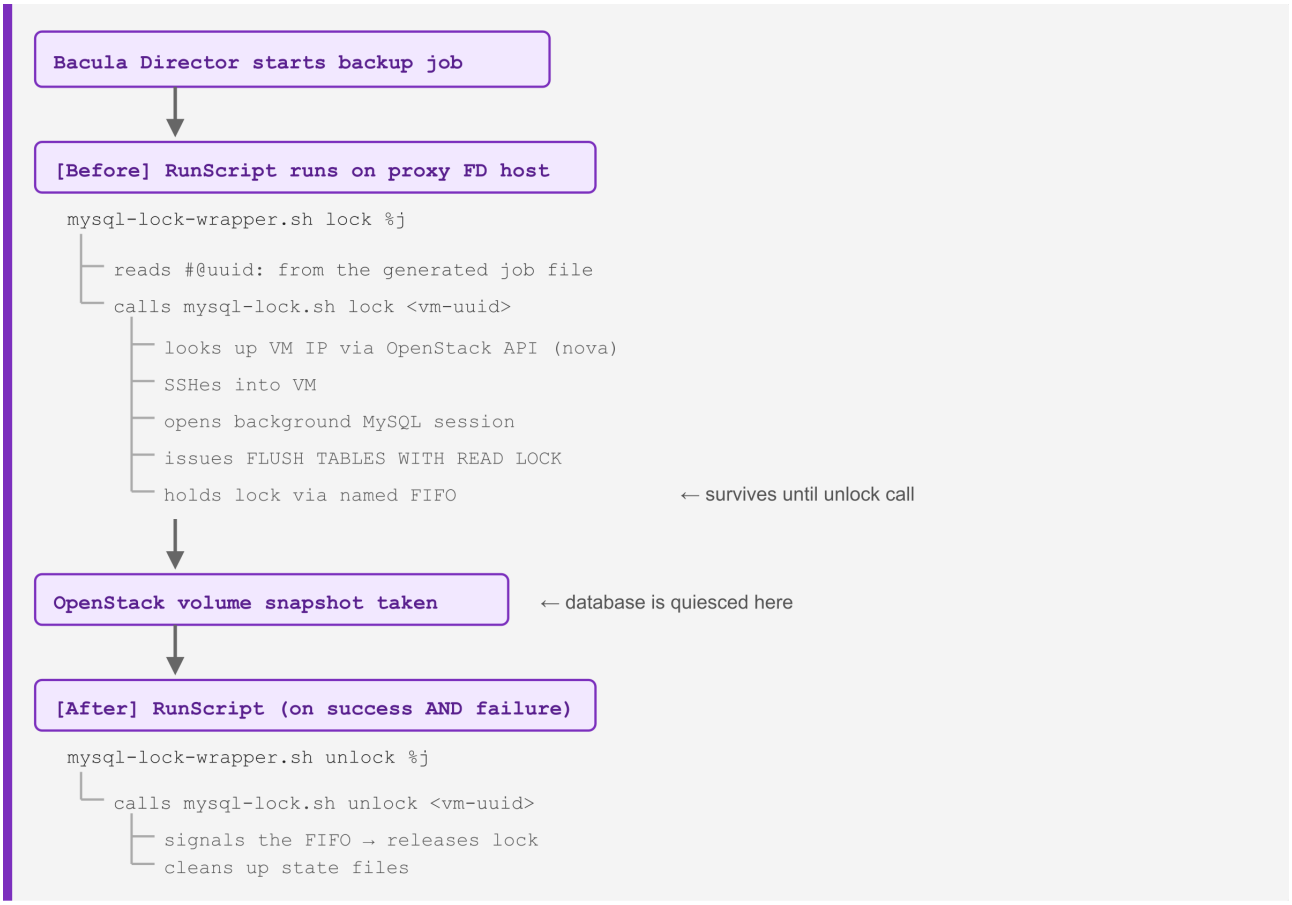
7. Application-Consistent Backups

7.1 The Problem — Volume Snapshots and Write Buffers

The Bacula Enterprise OpenStack plugin takes **volume-level block snapshots**. Without additional measures, any database with unflushed writes at the exact moment the snapshot is initiated may produce a backup image that requires crash recovery on restore. For a clean, immediately usable restore, the database must be quiesced — writes flushed and new writes blocked — for the brief instant the snapshot is initiated.

7.2 The Solution — Agentless Quiesce via SSH

The solution is a pair of shell scripts that run on the **backup proxy host** (the Bacula FD). They SSH into the target VM, hold the quiesce lock for the duration of the snapshot, then release it automatically. **No software is installed on the OpenStack VM itself** — only a standard SSH daemon is required.



7.3 MySQL / MariaDB — Consistent Snapshot Lock

SCRIPT	ROLE
`mysql-lock.sh`	Issues `FLUSH TABLES WITH READ LOCK` via SSH; holds it via a named FIFO until the unlock call; resolves all credentials from VM metadata or env file
`mysql-lock-wrapper.sh`	Bridge called by the Bacula `RunScript` : receives the Bacula job name (`%j`), locates the generated job file, reads the `#@uuid:` header, and delegates to `mysql-lock.sh`

Assigning the consistent-backup policy to a VM requires only a single metadata tag:

```
openstack server set --property backup-jobdefs=mysql-consistent <vm-name-or-uuid>
```

```
JobDefs {
  Name          = "mysql-consistent"
  Type          = Backup
  Level         = Incremental
  Accurate      = yes
  Priority      = 8
  Pool          = DatabasePool
  Schedule      = "DailyCycle"

  RunScript {
    Command         = "/opt/bacula/scripts/mysql-lock-wrapper.sh lock %j"
    RunsWhen        = Before
    RunsOnClient    = Yes
    FailJobOnError  = Yes
  }
  RunScript {
    Command         = "/opt/bacula/scripts/mysql-lock-wrapper.sh unlock %j"
    RunsWhen        = After
    RunsOnSuccess   = Yes
    RunsOnFailure   = Yes     # unlock even if the backup fails
    RunsOnClient    = Yes
    FailJobOnError  = No
  }
}
```

Important: `RunsOnFailure = Yes` on the unlock script is mandatory. If the backup fails after the lock is acquired, the unlock must still run to release the MySQL lock. Without it, the database remains locked until the SSH session times out.

7.4 Lock Duration and Application Impact

`FLUSH TABLES WITH READ LOCK` is a global read lock. Read queries continue normally while it is held, but write transactions queue until the lock is released. Applications with connection-pool timeouts longer than the lock window are unaffected — writes resume transparently as soon as the lock is released.

STORAGE BACKEND	SNAPSHOT MECHANISM	TYPICAL LOCK WINDOW
Ceph RBD *(most common)*	Metadata-only copy-on-write — near-instant	**2 – 10 seconds**
LVM thin provisioning	Thin snapshot — very fast	**5 – 20 seconds**
NFS / file-based storage	Extent copy at snapshot time	**30 s – several minutes**

> On a standard Ceph-backed OpenStack deployment the database is write-locked for **roughly 2 – 10 seconds** per backup run — comparable to a routine `FLUSH TABLES` during database maintenance, and invisible to well-configured application connection pools.

7.5 Extending to Other Databases and Applications

The `RunScript Before/After` quiesce pattern is not MySQL-specific. Any software that can be quiesced through a remote command is a candidate:

SOFTWARE	QUIESCE MECHANISM	NOTES
MySQL / MariaDB	`FLUSH TABLES WITH READ LOCK`	Ships with this solution as `mysql-lock.sh`
PostgreSQL	`pg_start_backup(…)` / `pg_stop_backup()`	Custom `RunScript` pair via SSH; no additional Bacula plugin required
Oracle	`ALTER TABLESPACE … BEGIN BACKUP / END BACKUP`	Requires Oracle DBA coordination; Bacula Enterprise Oracle plugin available as alternative
Microsoft SQL Server	VSS writer (Windows) or custom freeze script (Linux)	Bacula Enterprise MSSQL plugin available; VSS requires Windows guest

Custom application	Any pre/post quiesce script callable via SSH	Write a script pair, deploy to proxy host, reference in a dedicated `JobDefs`

8. Deployment and Operations

8.1 Preflight Check

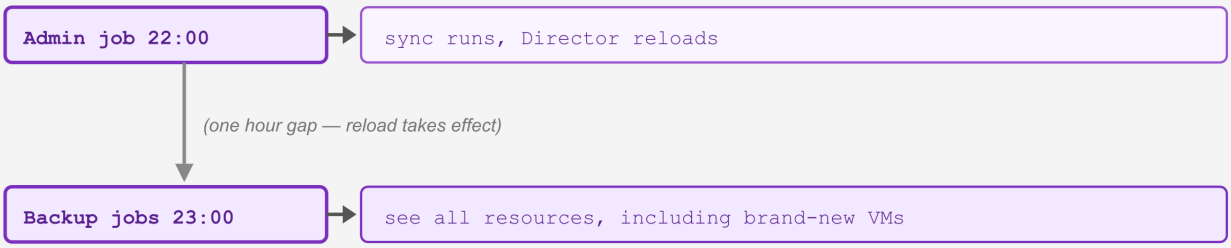
A preflight check script is included with the solution. Run it before the first deployment and after any configuration change. It reads the same environment variables as the binary and reports each check as `OK`, `WARN`, or `FAIL`:

```
scripts/preflight-check.sh
```

SECTION	WHAT IS VERIFIED
Binary	`bacula-openstack-sync` is on PATH and reports a version
clients.conf	File exists, is readable, contains at least one valid 5-field entry
BASE_DIR	Directory exists and is writable by the process user
bconsole	Executable found; successfully connects to the Bacula Director
PostgreSQL catalog	Connection succeeds; TLS mode validated; all four Bacula tables present
OpenStack API	Keystone endpoint reachable; credentials set; authentication succeeds
.query per client	For each `clients.conf` entry, `.query` returns at least one VM
Runtime config	`GRACE_DAYS` and `CLEANUP_AFTER_DAYS` are valid integers

8.2 Running as a Bacula Admin Job

The sync runs as an Admin job (`Type = Admin`), scheduled ahead of the normal backup jobs. The shipped configuration gives the Admin job its own earlier schedule so that VMs discovered in the sync run are backed up the same night:




```
Schedule {
  Name = "OpenStackSyncSchedule"
  Run = Level=Full daily at 22:00
}

Job {
  Name        = "OpenStackVMSync"
  Type        = Admin
  JobDefs     = "DefaultJob"
  Schedule    = "OpenStackSyncSchedule"
  Priority    = 9
  Enabled     = yes

  RunScript {
    RunsOnClient   = no
    RunsWhen       = Before
    FailJobOnError = yes
    Command = "/opt/bacula/scripts/run-openstack-sync.sh"
  }
}
```

8.3 Director Integration

The Director must load the generated job and fileset files at startup. A recursive include directive in `bacula-dir.conf` handles this automatically:

```
@|"sh -c 'find /opt/bacula/etc/openstack -type f \( -name job -o -name fileset \) -print0 |
xargs -0 -I{} echo @{}'"
```

```
echo "reload" | bconsole
```

8.4 Cleaning Up Deleted VMs

Set `CLEANUP_AFTER_DAYS` to automatically purge VMs that have stayed disabled long enough. On every sync run, after the normal reconcile loop, the tool scans all disabled jobs and fully removes any VM whose `#@disabled:` timestamp exceeds the threshold:

1. All backup runs purged via `bconsole`
2. `File`, `JobMedia`, `Log`, and `Job` catalog rows deleted
3. `FileSet` row deleted if no other job references it
4. Empty volumes deleted from catalog; physical files on the SD host flagged as ACTION REQUIRED
5. Job and fileset config files removed from disk

```
CLEANUP_AFTER_DAYS=90 bacula-openstack-sync sync
```

8.5 Decommissioning a Client

When an entire OpenStack instance or proxy FD is retired, `bacula-openstack-sync delete-client` removes all Bacula references for that client in one step. Three pre-conditions must be true before running the command:

- The OpenStack instance / proxy FD is offline.
- The client's line has been removed from `clients.conf` (the tool refuses to run otherwise).
- No backup jobs for this client are currently running.

```
# Always dry-run first:
DRYRUN=1 bacula-openstack-sync delete-client datacenter1 openstack1 bacula-dc1-os1-fd
backup-dc1-os1-

# Real run:
bacula-openstack-sync delete-client datacenter1 openstack1 bacula-dc1-os1-fd backup-dc1-os1-
```

8.6 Troubleshooting Reference

SYMPTOM	LIKELY CAUSE / FIX
`ERROR: cannot connect to the PostgreSQL catalog`	Wrong `DB_*` settings or missing `~/.pgpass` / `PGPASSWORD`
`ERROR: bconsole not found`	`bconsole` not on PATH — set the `BCONSOLE` variable to its full path

`WARN: no response from bconsole`	Proxy unreachable or `.query` command syntax wrong for this plugin version
A VM is not disabled even though it is gone	Still within the grace period — check `#@missing-since:` in its job file
Director `reload` fails with a parse error	A referenced resource (`JobDefs` , `Storage` , `Pool` , etc.) does not exist in the Director
`WARN: known to catalog but job file missing`	A job file was deleted manually — investigate before re-running
Client still in `clients.conf` error (`delete-client`)	Remove the client line from `clients.conf` before running the decommission command

To inspect a VM's current lifecycle state:

```
head -10 /opt/bacula/etc/openstack/datacenter1/openstack1/rocky9-instance1/job
```

9. Design Philosophy and Reliability Guarantees

The design of `bacula-openstack-sync` reflects a deliberate set of choices about what a backup automation tool should and should not do without explicit human authorisation.

The Catalog Is the Source of Truth

The tool uses the Bacula PostgreSQL catalog — not on-disk file scans — to determine what is "known". This is Bacula's authoritative record of what is being backed up. A brand-new job file (not yet in the catalog because it has not run once) is not treated as "missing" — the on-disk existence of the job file prevents it from being recreated.

Grace Periods Absorb Operational Noise

A single absence from a `.query` response does not mean a VM is gone. Stopped VMs, brief API unavailability, and proxy hiccups all appear as absences. Disabling a job immediately on first absence would generate false disables on a daily basis in any large environment. The configurable grace period ensures that only genuinely absent VMs — missing continuously for `GRACE_DAYS` — are disabled.

Deletion Is Always Opt-In

Disabling a job (`Enabled = no`) is reversible. Deleting catalog rows and physical files is not. The tool never deletes anything on its own. Cleanup is available in two explicit forms: the `CLEANUP_AFTER_DAYS` threshold (which must be deliberately configured) and the `delete-client` subcommand (which requires removing the client from `clients.conf` first as a safety interlock).

Per-Client Catalog Filtering Prevents Cross-Contamination

Every catalog query is filtered to the current proxy client. Without this filter, a sync run for one client would see another client's jobs and could wrongly classify those VMs as "disappeared". The requirement that `job_prefix` be globally unique across all entries in `clients.conf` reinforces this isolation.

All State Is In-Band

VM lifecycle state lives in `#@` headers inside the job files themselves. There is no external state database to lose, corrupt, or synchronise. If the job files are backed up (which they should be, as part of the Bacula configuration), the full lifecycle history travels with them.

Safety Under Concurrent Use and Partial Failures

Config files are written to a temporary file and renamed into place — a process crash mid-write leaves the previous file intact. All catalog deletions for a single VM execute inside a single database transaction — either all rows are removed or none are. Every bconsole subprocess is killed after a configurable timeout; output is capped to prevent memory exhaustion. A failure in one OpenStack instance does not stop the sync for the others.

10. Conclusion

At the scale of a multi-datacenter OpenStack environment, manual backup management is not merely inconvenient — it is structurally unable to provide the coverage guarantees that modern data protection requirements demand.

`bacula-openstack-sync` addresses this directly. By running as a standard Bacula Admin job ahead of the backup window, it ensures that every VM is protected from the moment it appears in the OpenStack environment, that transient absences are handled gracefully without operator involvement, and that the overall backup estate reflects the actual VM fleet at all times.

Combined with Bacula Enterprise's Global Endpoint Deduplication for storage efficiency and the agentless application-consistent quiescing framework for databases, the solution provides a complete, production-ready backup automation stack for OpenStack environments of any scale.

The key operational outcomes are:

- **Zero-gap coverage.** Every VM is backed up automatically. The window between a VM being provisioned and its first backup is bounded by the sync schedule and the Director reload timing — typically under one hour in the recommended configuration.
- **Reduced operational overhead.** The backup administrator is no longer in the critical path for VM provisioning or decommissioning. The sync tool handles both automatically, with the safety net of grace periods and opt-in cleanup.
- **Resilient history.** Because disabling is reversible and deletion is always opt-in, backup history is preserved across VM restores, migrations, and temporary outages. A VM that returns after weeks of absence is re-enabled with its complete history intact.
- **Application-consistent restores.** Database VMs tagged with the appropriate `backup-jobdefs` policy produce snapshots that are immediately usable on restore, without crash-recovery overhead — and without installing any agent on the guest VM.

> *"Automated coverage is not a convenience feature — it is a prerequisite for trustworthy backup at cloud scale."*

faaleoleo is available to assist with scoping, deployment planning, configuration, and ongoing operational support for this solution. Please contact your faaleoleo account representative to arrange a technical discussion.

ABOUT FAALEOLEO

faaleoleo is an independent systems engineering firm specialising in enterprise backup, disaster recovery, and infrastructure automation. With deep expertise in Bacula Enterprise, faaleoleo designs, implements, and operates large-scale backup environments for customers across telecommunications, finance, and critical infrastructure sectors.