# PodHeitor Hyper-V Backup Plugin for Bacula

**Full VM backup and restore for Microsoft Hyper-V — direct VHDX access, RCT block-level incremental, application-consistent.**

Back up Hyper-V virtual machines including VHDX disks, configuration, checkpoints, and metadata. No Export-VM needed — reads VHDX files directly with production checkpoints for consistency, and uses Resilient Change Tracking (RCT) for true block-level incremental/differential backups.

> **Built with Rust** — The backend engine (`podheitor-hyperv-backend.exe`) is implemented in Rust, delivering memory safety, zero-cost abstractions, and native performance. This is a state-of-the-art Hyper-V backup plugin — no PowerShell spaghetti, no fragile script chains. Production-grade systems engineering from the ground up.

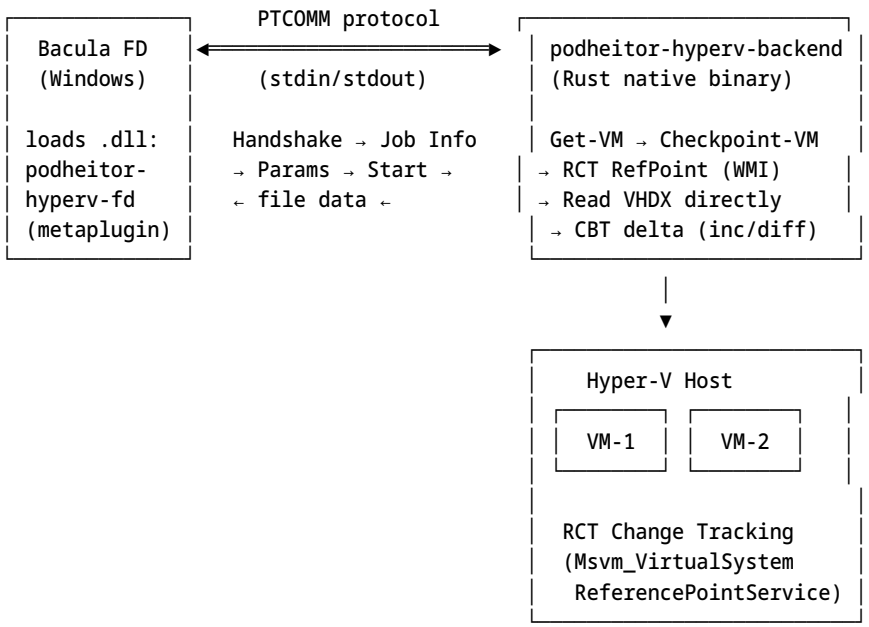
## Why Direct VHDX + RCT

| Challenge | Direct VHDX + RCT Approach |
|---|---|
| VHD files locked while VM runs | Production checkpoint freezes parent VHDX for reading |
| Inconsistent state during copy | VSS quiesce ensures application consistency |
| Export-VM requires full local copy | No export — read VHDX in place, zero extra disk space |
| Full VHDX re-sent on each incremental | RCT tracks changed blocks — send only modified regions |
| Fast incremental for large disks | Block-level delta: 100GB disk with 2GB changed → only 2GB transferred |
| Manual snapshot management | Plugin creates + removes checkpoint automatically |
| No catalog of backed-up files | Bacula catalog tracks every file — restore individual VHDs |

## Features

- **Direct VHDX access** — no Export-VM, no local disk copy, zero extra space needed
- **RCT (Resilient Change Tracking)** — true block-level incremental/differential via Windows API
- **CBT delta format** — custom binary format streams only changed VHDX blocks
- **Application-consistent** — production checkpoints with VSS quiesce
- **Online backup** — no VM downtime required
- **Full / Incremental / Differential** — native Bacula levels backed by RCT change tracking
- **Wildcard VM selection** — `vm=*` backs up all VMs, `vm=SQL*` matches pattern
- **Exclude filters** — skip specific VMs by name pattern
- **Per-file catalog** — browse and restore individual VHDX or config files
- **Automatic cleanup** — checkpoint removed after job completes
- **Compression & Encryption** — Bacula's native LZ4/GZIP + AES
- **Metaplugin architecture** — same framework as official K8s/Docker/SFTP plugins
- **Gen1 fallback** — gracefully falls back to full VHDX streaming if RCT unavailable
- **Granular Restore** — file-level recovery from VM disk backups (see Granular Restore)
- **Cross-Hypervisor Restore** — restore VMs across Hyper-V, VMware vSphere, Proxmox/KVM (see Cross-Hypervisor Support)
- **Rust backend** — memory-safe, high-performance backend engine — no interpreted scripts in the critical path

## Architecture



### Backup Flow (per VM)

**Full backup:** 1. **Enumerate** — `Get-VM` filtered by `vm` pattern and `exclude` list 2. **Checkpoint** — `Checkpoint-VM` production checkpoint (VSS quiesce) 3. **RCT RefPoint** — create RCT reference point via WMI (baseline for future incrementals) 4. **Direct Read** — open frozen VHDX with `FileShare.ReadWrite`, stream via PTCOMM 5. **Config** — stream VM configuration files (VMCX, etc.) 6. **Cleanup** — remove production checkpoint, save RCT state to `%ProgramData%\PodHeitor\rct\`

**Incremental backup (RCT):** 1. **Enumerate** — same as Full 2. **Checkpoint** — production checkpoint for consistency 3. **RCT Query** — create new reference point, call `GetVirtualDiskChanges` (WMI) between previous and current reference points 4. **CBT Delta** — read only changed blocks from VHDX, stream as CBT delta file 5. **Config** — stream VM configuration files (always full — they're small) 6. **Cleanup** — remove checkpoint, update RCT state, remove old reference point

**Differential backup:** - Same as Incremental, but queries changes since the last **Full** backup's reference point

### CBT Delta Format

Binary format for streaming only changed VHDX blocks:

```
Bytes 0-7:    Magic "PHCBT01\n"
Bytes 8-15:   OriginalSize uint64 LE (full VHDX size)
Bytes 16-19:  RegionCount uint32 LE
Per region:
  8 bytes: Offset uint64 LE (position in VHDX)
  4 bytes: Length uint32 LE (byte count)
  N bytes: Data (actual changed bytes)
```

### Namespace

Files appear in Bacula catalog under:

```
@hyperv/VMName/disks/disk.vhdx                    # Full backup — complete VHDX
@hyperv/VMName/disks/disk.vhdx.I.J123.cbt          # Incremental — CBT delta only
@hyperv/VMName/disks/disk.vhdx.D.J456.cbt          # Differential — CBT delta only
@hyperv/VMName/config/Virtual Machines/config.vmcx  # VM configuration
@hyperv/VMName/.meta/rct-state.json                 # RCT reference point state
```

## Requirements

| Component | Minimum | Notes |
|---|---|---|
| Bacula Community | 13.0.0+ | Metaplugin framework required |
| Windows Server | 2016+ | Hyper-V role enabled |
| PowerShell | 5.1+ | Built-in on Windows Server 2016+ |
| Hyper-V module | Built-in | `Get-Module -ListAvailable Hyper-V` |
| VM Generation | Gen2 | Required for RCT (Gen1 falls back to full disk) |
| Disk format | VHDX | Required for RCT (VHD falls back to full disk) |
| Disk space | Minimal | No Export-VM — VHDX read directly in place |

## Installation

### Build Installer (.exe)

Use a Windows host with NSIS installed (`makensis.exe` in PATH).

```
powershell -ExecutionPolicy Bypass -File .\build-installer.ps1 `
  -PluginDllPath 'C:\plugin-build\podheitor-hyperv-fd.dll' `
  -Version '1.0.1' `
  -OutputDir '.\release'
```

Generated artifact:

- `release\PodHeitor-HyperV-Plugin-1.0.1-x64.exe`

### Install via .exe

1. Run installer as Administrator on Hyper-V host.
2. Verify files:
    - `C:\Program Files\Bacula\plugins\podheitor-hyperv-fd.dll`

- `C:\Program Files\Bacula\plugins\podheitor-hyperv-backend.exe`

3. Restart File Daemon service:

```
Restart-Service -Name bacula-fd -Force
```

### Deploy DLL Hotfix Without Installer

Use when only metaplugin DLL changed.

```
powershell -ExecutionPolicy Bypass -File .\deploy-dll-hotfix.ps1 `
  -ComputerName '192.168.15.84','HYPERV-NODE-02' `
  -PluginDllPath 'C:\plugin-build\podheitor-hyperv-fd.dll'
```

### Execute Installer Remotely (Multi-Host)

```
powershell -ExecutionPolicy Bypass -File .\install-plugin-remote.ps1 `
  -ComputerName '192.168.15.84','HYPERV-NODE-02' `
  -InstallerPath '.\PodHeitor-HyperV-Plugin-1.0.1-x64.exe'
```

### From Source (Windows - MSYS2/MinGW)

```
make BACULA_SRC=C:/bacula-src/src
make install PREFIX="C:/Program Files/Bacula" BACULA_SRC=C:/bacula-src/src
```

### Cross-compile from Linux (MinGW)

```
make BACULA_SRC=/path/to/bacula-15.0.3/src CXX=x86_64-w64-mingw32-g++
```

### Manual Installation

1. Copy `podheitor-hyperv-fd.dll` to `C:\Program Files\Bacula\plugins\`
2. Copy `podheitor-hyperv-backend.exe` to `C:\Program Files\Bacula\plugins\`
3. Copy `podheitor-hyperv-backend.cmd` and `podheitor-hyperv-backend.ps1` to `C:\Program Files\Bacula\plugins\`
4. Restart Bacula FD: `Restart-Service bacula-fd`

See `INSTALLATION_MANUAL.md` for full packaging, rollout, validation, and rollback steps.

## Configuration

### Plugin Parameters

| Parameter | Required | Default | Description |
|---|---|---|---|
| `vm` | no | * (all VMs) | VM name or wildcard pattern |
| `exclude` | no | — | Comma-separated VM name patterns to exclude |
| `quiesce` | no | yes | Create application-consistent (VSS) checkpoint |
| `online` | no | yes | Allow backup of running VMs |
| `timeout` | no | 3600 | Operation timeout in seconds |
| `abort_on_error` | no | no | Abort job on VM backup error |

### Quick Start

```
FileSet {
  Name = "HyperV-AllVMs"
  Include {
    Options { Signature = SHA256; Compression = LZ4 }
    Plugin = "podheitor-hyperv: vm=*"
  }
}

Job {
  Name = "HyperV-Daily"
  Type = Backup
  Level = Incremental
  FileSet = "HyperV-AllVMs"
  Client = hyperv-host-fd
```

```
  Storage = File1
  Pool = Default
  Schedule = "WeeklyCycle"
}
```

### Selective VM Backup

```
# Only SQL Server VMs
Plugin = "podheitor-hyperv: vm=SQL*"

# All VMs except test/dev
Plugin = "podheitor-hyperv: vm=* exclude=Test*,Dev*"

# Specific VM
Plugin = "podheitor-hyperv: vm=DC01"

# Skip quiesce for Linux VMs (no integration services)
Plugin = "podheitor-hyperv: vm=Linux* quiesce=no"
```

### Multiple FileSet Entries

```
FileSet {
  Name = "HyperV-Production"
  Include {
    Options { Signature = SHA256; Compression = LZ4 }
    Plugin = "podheitor-hyperv: vm=SQL* quiesce=yes"
    Plugin = "podheitor-hyperv: vm=DC* quiesce=yes"
    Plugin = "podheitor-hyperv: vm=Web* quiesce=no"
  }
}
```

### RCT State

RCT reference point state is stored locally on the Hyper-V host at:

`%ProgramData%\PodHeitor\rct\<VM-GUID>.json`

Each VM's state tracks: - `FullRefPointId` — reference point from the last Full backup - `LastRefPointId` — reference point from the last backup (any level) - Timestamps and job IDs for auditing

The state is also streamed as `@hyperv/VMName/.meta/rct-state.json` in each backup for reference.

### Restore

```
bconsole
* restore where=C:/Restore select all done yes
```

Files are restored under the `@hyperv/VMName/...` namespace. CBT delta files (`.cbt`) are automatically applied to the base VHDX during restore — no manual steps needed.

After restore, import the VM:

```
# Import restored VM
Import-VM -Path "C:\Restore\@hyperv\VMName\config\Virtual Machines\config.vmcx" `
          -Copy -GenerateNewId -VhdDestinationPath "C:\Restore\@hyperv\VMName\disks"
```

## Granular Restore (File-Level Recovery)

This plugin is fully compatible with **PodHeitor Granular Restore** — a companion tool that enables file-level recovery from VM disk backups without restoring the entire VM.

### How It Works

The Hyper-V plugin stores VHDX disks and CBT deltas in the Bacula catalog under the `@hyperv/` namespace. PodHeitor Granular Restore can:

1. **Browse** the Bacula catalog (BVFS) to locate VM disk files
2. **Restore** only the needed VHDX + CBT deltas to the Storage Daemon
3. **Reconstruct** the full disk state by applying CBT deltas onto the base VHDX
4. **Mount** the guest filesystem (NTFS, ext4, XFS, etc.) via libguestfs or qemu-nbd
5. **Browse and extract** individual files from the mounted VM disk

```
# Mount a backed-up VM's filesystem
mount-vm restore

# Or mount a VHDX directly
mount-vm mount -d /path/to/disk0.vhdx -m /mnt/restore

# Convert VHDX to another format
mount-vm convert -s /path/to/disk0.vhdx -o /path/to/disk0.qcow2 -f qcow2
```

### Cross-Hypervisor Support

PodHeitor Granular Restore understands VHDX, VMDK, and qcow2 disk formats, enabling **cross-hypervisor VM restore**:

| Source Hypervisor | Target Hypervisor | Conversion |
|---|---|---|
| Hyper-V | VMware vSphere | VHDX → VMDK |
| Hyper-V | Proxmox / KVM | VHDX → qcow2 |
| VMware vSphere | Hyper-V | VMDK → VHDX |
| VMware vSphere | Proxmox / KVM | VMDK → qcow2 |
| Proxmox / KVM | Hyper-V | qcow2 → VHDX |
| Proxmox / KVM | VMware vSphere | qcow2 → VMDK |

Back up from one hypervisor, restore to another — the plugin handles disk format conversion transparently.

> See the PodHeitor Granular Restore project for installation and usage details.

## Debugging

```
# Enable debug logging
$env:PODHEITOR_DEBUG = '1'
Restart-Service bacula-fd

# Backend logs go to stderr → captured in Bacula job log
```

## Validated Platforms

| OS | Hyper-V Role | Plugin Version | Status |
|---|---|---|---|
| Windows Server 2016 | Built-in | 1.0.1 | Full + Incremental + Restore ✓ |
| Windows Server 2025 | Built-in | 1.0.1 | Full + Incremental + Restore ✓ |

## License