

Backrest & Restic

Moderne Backup-Orchestrierung

Paul Kozber - ORDIX AG

1. Das Problem herkömmlicher Backups

- **Hoher Administrationsaufwand:** Dezentrale Backup-Skripte auf jedem Server erfordern manuelle Pflege und führen zu unzuverlässigem Monitoring.
- **Sicherheitsrisiko SSH-Root-Zugriff:** Klassische Backup-Server besitzen oft vollen SSH-Root-Zugriff auf alle Zielsysteme. Wird der Backup-Server kompromittiert, sind alle Server betroffen.
- **Ineffizienter Speicherverbrauch:** Traditionelle Vollsicherungen verschwenden Bandbreite und Speicherplatz.
- **Fehlende Verschlüsselung:** Viele Standardwerkzeuge unterstützen keine clientseitige, sichere Verschlüsselung (Zero-Knowledge-Prinzip).
- **Die Zielstellung:** Ein zentrales **Pull-Modell** mit minimalem Agentenaufwand auf den Quellsystemen und strikter Rechthärtung.

2. Die Lösung: Restic & Backrest

- **Restic-Backend:** Ein moderner, Go-basiertes Backup-Client.
 - Sichert Daten in Blöcken mittels **AES-256** clientseitiger Verschlüsselung.
 - Unterstützt globales Deduplizieren mittels inhaltsdefinierter Partitionierung (Content Defined Chunking - CDC).
 - Native Unterstützung vieler Storage-Backends (S3, SFTP, REST-Server, Storage Boxen).
- **Backrest-Orchestrator:** Eine komfortable Web-UI und API zur Orchestrierung.
 - Verwaltet zentrale Repositories, Zeitpläne, Aufbewahrungsrichtlinien und Webhook-Alerts.
 - Führt Restic-Operationen im Hintergrund aus und visualisiert Metriken.

3. Härtung des Backrest Webinterfaces

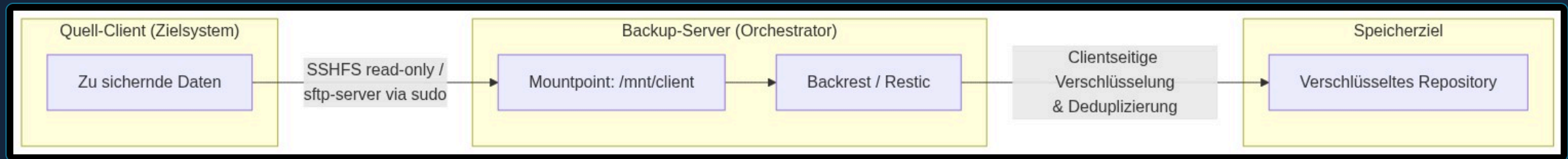
- **Authentifizierung:** Das Webinterface wird über Bcrypt-Hashes gesichert.
- **Base64-Codierung:** Backrest erwartet in der Konfiguration unter `/opt/backrest/config/config.json` das Passwort als Base64-codierten Bcrypt-Hash.
- **Beispielhafte Konfiguration (`config.json`):**

```
"auth": {
  "authDisabled": false,
  "users": [
    {
      "name": "admin",
      "passwordBcrypt": "JDJhJDEwJGd1bW15RGVtb0hhc2hGb3JQcmVzZW50YXRpb25Pbm5V29ya3NGaW5l"
    }
  ]
}
```

(Hinweis: Der gezeigte Hash `JDJhJDEw...` decodiert zum Bcrypt-Hash `$2a$10$gummyDemoHash...`, was einem Dummy-Passwort entspricht).

4. Das Sicherheitskonzept (SSHFS Least Privilege)

- **Kein Root-Zugriff:** Der SSH-Key von `root@backrest` wurde von allen Systemen entfernt. Ein direkter Root-Login per SSH ist nicht möglich.
- **Pull-Prinzip:** Der Backup-Server holt sich die Daten aktiv ab, anstatt dass die Clients Schreibrechte auf das Backup-Repository besitzen.
- **Visualisierter Datenfluss:**



5. SFTP-Server via Sudo (Client-Härtung)

- **Eigener Systembenutzer:** Auf jedem Client existiert ein unprivilegierter Benutzer `backup`.
- **SSH-Key-Restriktion:** Nur der öffentliche Schlüssel des Backrest-Servers ist für den User `backup` hinterlegt.
- **Restriktive Sudoers-Regel:** Eine dedizierte Regel unter `/etc/sudoers.d/backup` erlaubt dem User `backup` **ausschließlich** die Ausführung des SFTP-Servers mit Root-Rechten ohne Passworteingabe:

```
backup ALL=(root) NOPASSWD: /usr/lib/openssh/sftp-server
```

- **Vorteil:** Der Backup-Server kann über den Benutzer `backup` alle Systemdateien lesen, besitzt aber keinerlei SSH-Shell-Schreibrechte auf dem System.

6. SSHFS-Mounts in der /etc/fstab des Servers

- **Mount-Durchführung:** Der Backrest-Server mountet die Client-Verzeichnisse lokal über SSHFS.
- **Eintrag in /etc/fstab :**

```
backup@192.168.1.100:/opt/n8n /mnt/n8n fuse.sshfs allow_other,default_permissions,compression=yes,cache=yes,auto_cache,reconnect,IdentityFile=/root/.ssh/id_ed25519,user,_netdev,sftp_server=sudo\040/usr/lib/openssh/sftp-server 0 0
```

- **Details der Optionen:**
 - `sftp_server=sudo\040/usr/lib/openssh/sftp-server` : Erzwingt die Ausführung des SFTP-Servers auf dem Client via `sudo` (Leerzeichen als `\040` kodiert).
 - `reconnect` : Ermöglicht automatische Wiederverbindung bei Netzwerkunterbrechungen.
 - `_netdev` : Verzögert den Mount beim Systemstart, bis das Netzwerk online ist.

7. Ansible Automation (Host-Variables)

- **Zentrale Verwaltung:** Die Repositories, SSHFS-Mounts und Backup-Pläne sind deklarativ in Ansible definiert.
- **Auszug aus** `inventory/host_vars/backrest/main.yml` :

```
backrest_repos:
- id: "backup-storage-repo"
  uri: "sftp:user@backup-host:/home/backup"
  password: "my-secret-repository-password"

backrest_sshfs_mounts:
- src: "backup@192.168.1.100:/opt/n8n"
  path: "/mnt/n8n"
  opts: "allow_other,default_permissions,compression=yes,cache=yes,auto_cache,reconnect,IdentityFile=/root/.ssh/id_ed25519,user,_netdev,sftp_server=sudo\\040/usr/lib/openssh/sftp-server"
```

8. Ansible Deployment-Ablauf (Role Tasks)

- Versionsprüfung & Binär-Deployment:
 - Abgleich der installierten Backrest-Version via `--version`. Bei Abweichung automatischer Download des `.tar.gz`-Archivs von GitHub, Entpackung nach `/tmp` und Installation.
- Pfad- & Systemd-Einrichtung:
 - Erstellung der Verzeichnisse `/opt/backrest/config` und `/opt/backrest/data`.
 - Generierung der `/etc/systemd/system/backrest.service` mit den passenden Umgebungsvariablen:

```
[Service]
ExecStart=/usr/local/bin/backrest
Environment="BACKREST_PORT=127.0.0.1:9898"
Environment=BACKREST_CONFIG=/opt/backrest/config/config.json
Environment=BACKREST_DATA=/opt/backrest/data
```

9. Backup-Workloads (Praxisbeispiele)

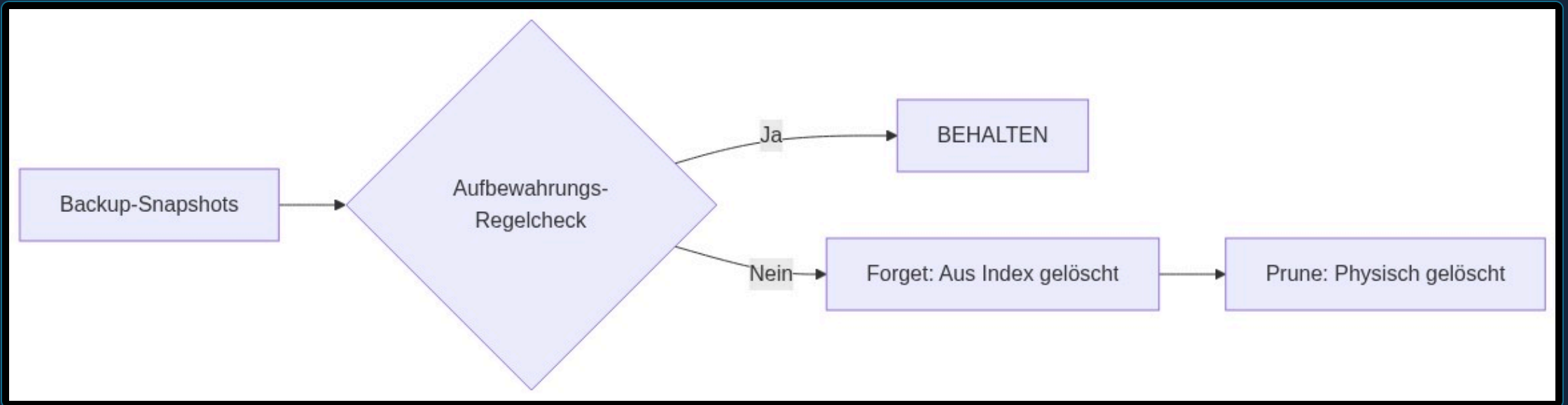
- **n8n (Docker-Workload):**
 - Sichert das Verzeichnis `/opt/n8n` (Compose-Konfigurationen und Skripte).
 - Sichert `/var/lib/docker/volumes/n8n_n8n_data/_data` (Datenbank/Workflows).
 - Vorab-Skript: Ein lokaler Cronjob führt um 02:30 Uhr `/opt/n8n/backup-db.sh` aus und legt einen konsistenten PostgreSQL-Dump unter `/opt/n8n/backups/db_dump.sql` ab.
- **Wazuh-Manager (Security Monitoring):**
 - Mountet `/var/ossec` remote nach `/mnt/wazuh`.
 - Nutzt Ausschlusslisten (`**/logs` , `*.log`), um offene Logdateien während des Backups zu ignorieren und Warnungen zu vermeiden.

10. Aufbewahrung (Retention) & Rotation

- **Bereinigungsprozess in zwei Schritten:**

1. **Forget** (`restic forget`): Removes old snapshots from inventory indices based on time retention.
2. **Prune** (`restic prune`): Physikalisch nicht mehr referenzierte Datenblöcke löschen.

- **Aufbewahrungs-Lebenszyklus:**



- **Beispiel-Richtlinie:** Täglich: 7, Wöchentlich: 4, Monatlich: 1.

11. Wiederherstellung & Disaster Recovery (CLI & UI)

- Bypass via CLI im Katastrophenfall (kompakt):

```
# 1. Restic-Repository und Passwort exportieren
export RESTIC_REPOSITORY="sftp:user@backup-host:/home/backup"
export RESTIC_PASSWORD="my-secret-repository-password"

# 2. Letzten Zustand nach /tmp/restore wiederherstellen
restic restore latest --target /tmp/restore

# 3. Alternativ: Dateisystem-Mount für Dateisuche
restic mount /mnt/restic-restore
```

12. Überwachung & Webhook Integration

- **Direktes Webhook-Alerting:** Backrest unterstützt native Benachrichtigungen an Chatplattformen über Webhooks.
- **Webhook-Konfiguration in Ansible:**

```
hooks:  
  - conditions:  
    - "CONDITION_ANY_ERROR"  
    - "CONDITION_SNAPSHOT_SUCCESS"  
    - "CONDITION_PRUNE_ERROR"  
  actionDiscord:  
    webhookUrl: "https://webhook.example.com/services/..."  
    template: "{% raw %}{{ .Summary }}{% endraw %}"
```

- **Vorteil:** Schnelle Übersicht über den Zustand der täglichen Backups direkt im Admin-Chat, inklusive Fehlermeldungen und Speicherplatz-Metriken.

13. Backup-Alternativen im Vergleich

- **BorgBackup / Borgmatic:**
 - *Vorteil:* Schnelle Komprimierung, ausgereift.
 - *Nachteil:* Benötigt SSH-Schreibzugriff auf das Repository (kein reines Pull-Modell) und hat keine native Web-UI.
- **Kopia:**
 - *Vorteil:* Moderne Benutzeroberfläche und native Cloud-Konnektoren.
 - *Nachteil:* Geringerer Verbreitungsgrad und weniger Langzeiterprobung.
- **UrBackup:**
 - *Vorteil:* Einfache Image-Backups ganzer Systeme.
 - *Nachteil:* Ineffizientere Deduplizierung und fehlendes Zero-Knowledge-Sicherheitsmodell.
- **Velero:**
 - *Vorteil:* Sichert Kubernetes/k3s Volumes und APIs.
 - *Nachteil:* Ungeeignet für Standalone-Server, VMs oder LXC-Konfigurationen.