OSR Agent for Linux Handbuch

Dokumentinformationen

Bezug

- OSR: OSR Agent for Linux
  (SMART Storage Monitoring, Healthchecks, Inventory) [K13036]
  Public Link: wissen.braucke.net/Help/OSR_Agent_Linux

Dokument-Historie

- 09.04.2026 – Chat: OSRAgentforLinux.json Dokumentation – ChatGPT Version: GPT-5.3
  Chat: Dokumentation OSR Agent b1111(24a)

Übersicht Überwachungsfunktionen

SMART-Überwachung der Datenträger

Früherkennung von Hardware-Defekten an HDD/SSD. Überwacht werden SMART Overall Health sowie wichtige Attribute wie Current Pending Sector (197), Offline Uncorrectable (198), Reallocated Sector Count (5) und UDMA CRC Error Count (199). Zusätzlich wird eine Delta-Analyse seit dem letzten Lauf durchgeführt.

Filesystem-Auslastung

Überwachung der belegten Kapazität sowie der Inode-Nutzung. Warnung bei 80 % und Fehler bei 95 % (Standardwerte).

Inode-Forensik

Bei kritischer Inode-Auslastung werden zusätzliche Analysen erzeugt: Top-Verzeichnisse nach Dateianzahl, häufigste Dateitypen sowie größte Dateien.

systemd Service-Status

Erkennung von fehlgeschlagenen oder nicht gestarteten Systemdiensten über systemctl --failed.

systemd Restart-Loops

Analyse von Diensten, die mehrfach automatisch neu gestartet wurden. Typisches Symptom für crashende Services.

Überwachung kritischer Services

Prüfung definierter Pflichtdienste (z. B. smartd, postfix, ssh). Wenn diese nicht aktiv sind, wird ein Fehler gemeldet.

systemd Timer Überwachung

Kontrolle geplanter systemd-Timer. Der Agent prüft, ob Timer aktiv sind und ob sie innerhalb eines definierten Zeitfensters ausgelöst wurden.

Journal-Fehleranalyse

Auswertung kritischer Logmeldungen aus dem systemd Journal (Priorität ≤ 3: Error, Critical, Alert, Emergency).

Kernel-Fehler

Analyse von Kernel-Meldungen, um Treiberprobleme oder Systemfehler frühzeitig zu erkennen.

Disk-I/O Fehler im Kernel

Suche nach typischen Storage-Fehlern wie I/O errors, Filesystem-Fehlern (EXT4, XFS, BTRFS) oder Controller-Resets.

Partition Audit

Analyse vorhandener Partitionen. Erkennt Filesysteme ohne Mountpoint oder ungewöhnliche Device-Strukturen.

Disk- und Partitionserkennung

Automatische Erkennung aller internen und externen Datenträger (SATA, NVMe, USB) sowie deren Partitionen und Mountpoints.

Zustandsspeicher (smart-state.json)

Der Agent speichert historische SMART-Werte, erste und letzte Sichtung einer Disk sowie erkannte Delta-Ereignisse.

E-Mail Reporting

Der Agent erzeugt strukturierte Berichte mit den Abschnitten Overview, Summary, SMART, Filesystem, Core Health und Logfile-Auszug.

Installation

Setup

1. Konfiguration `OSRAgentforLinux.json` ggf. anpassen
2. Per ssh kopieren nach /root/src/ oder var/tmp/installs
   `OSRAgentforLinux.json`
   `OSRAgentforLinux.sh`
   `OSRAgentforLinux-Setup.sh`
3. OSRAgentforLinux-Setup.sh ausführbar machen
   `sudo chmod 750 OSRAgentforLinux-Setup.sh`
4. `sudo OSRAgentforLinux-Setup.sh --help`
   oder direkt
   `sudo ./OSRAgentforLinux-Setup.sh --src . --install-timer --force-config`

Weitere Infos zum Setup-Skript

Aktuelle Details und Varianten siehe wissen.braucke.net/Help/OSR_Agent_Linux.

Konfiguration

OSRAgentforLinux.json

Übersicht

Zweck der Konfiguration

Diese Konfiguration steuert den OSR Agent for Linux und dient der systematischen Überwachung von Linux-Systemen. Sie ermöglicht eine einheitliche Bewertung von Systemzuständen, die frühzeitige Erkennung von Problemen sowie eine standardisierte Darstellung von Ergebnissen.

Struktur der Konfiguration

Die Struktur folgt einem hierarchischen Aufbau mit den Bereichen meta, defaults, profiles und hosts. Diese Trennung erlaubt eine klare Aufteilung zwischen globalen Einstellungen, wiederverwendbaren Profilen und konkreten Host-Zuordnungen.

globale Einstellungen

Im Bereich defaults werden zentrale Einstellungen wie Logging, Mailversand, Output-Formatierung, Ressourcenlimits und Schwellenwerte definiert. Diese gelten für alle Hosts, sofern sie nicht durch Profile überschrieben werden.

Thresholds und Bewertungen

Thresholds definieren Grenzwerte für WARN- und FAIL-Zustände, insbesondere für SMART-Daten, RAID-Zustände und Filesystem-Auslastung. Zusätzlich werden Delta-Werte genutzt, um Veränderungen im Zeitverlauf zu erkennen.

Welche Checks werden durchgeführt?

Die Checks umfassen systemd-Status, Timer-Überwachung, Journal- und Kernel-Fehler, SMART-Daten, RAID-Status, Filesystem-Zustände sowie spezielle Analysen wie Inode-Forensik und Partition-Audits.

Wie funktioniert die Filesystem-Überwachung?

Die Filesystem-Überwachung analysiert Kapazität, Inodes und spezifische Eigenschaften je nach Dateisystem (z. B. ZFS Scrub Age oder Btrfs-Metadaten). Zusätzlich werden Netzwerk-Dateisysteme auf Erreichbarkeit geprüft.

Welche Rolle spielen Profile?

Profile definieren host-spezifisches Verhalten. Beispiele sind pve, nas, vm oder lab. Sie legen fest, welche Services erforderlich sind, wie SMART behandelt wird und welche Timer erwartet werden.

Wie erfolgt die Zuordnung zu Hosts?

Im Bereich hosts werden konkrete Systeme einem Profil zugeordnet. Dadurch wird das Verhalten zentral gesteuert, ohne jede Konfiguration individuell definieren zu müssen.

Welche Designprinzipien liegen zugrunde?

Die Konfiguration folgt Prinzipien wie Trennung von Verantwortlichkeiten, Wiederverwendbarkeit durch Profile, Erweiterbarkeit und klare Versionierung über ein Schema.

Stärken der aktuellen Version

Die Version b1111(24) zeichnet sich durch eine konsistente Struktur, hohe Praxisnähe und gute Erweiterbarkeit aus. Sie eignet sich für produktive Umgebungen und größere Systemlandschaften.

Filter

ignore_patterns

In b1111(27) werden alle ignore_patterns als Regex behandelt.

Um z. B.
`GET /api2/json/admin/datastore/PBS_ROT01/snapshots?: 400 Bad Request`
zu filtern muss das ? welches im Bash-Regex =~ als Metazeichen interpretiert wird escaped werden

Ebene	Eingabe	Ergebnis
JSON	`\\?`	`\?`
Bash Regex	`\?`	literal `?`
Log-Zeile	`snapshots?`	✓ Match

mit unescaped snapshots? → **kein Match**
mit escaped snapshots\? → **Match**
Und da in JSON ebenfalle \ ein Escape-Zeichen ist muss \\ vor ein ?:

`GET /api2/json/admin/datastore/PBS_ROT01/snapshots\\?: 400 Bad Request`

Parametern und Schalter

Meta-Parameter

- namespace: Logischer Namensraum
- project_id: Projektkennung
- tool: Toolname
- vendor: Hersteller
- version: Konfigurationsversion
- schema_version: Schema-Version
- config_version: Version der Konfiguration

Logging-Parameter

- log_group: Gruppe für Logdateien
- logrotate.enabled: Aktiviert Rotation
- logrotate.frequency: z. B. daily
- logrotate.maxsize: Max Größe
- logrotate.rotate: Anzahl Versionen

- logrotate.compress: Kompression
- logrotate.copytruncate: Copytruncate Modus

Mail-Parameter

- sendto: Empfänger
- layout: Layout (z. B. sectioned_v2)
- debug: Debug-Modus
- lastloglines_tosend: Anzahl Logzeilen

SMART-Parameter

- fail.current_pending_sector: FAIL Schwelle
- fail.offline_uncorrectable: FAIL Schwelle
- warn.reallocated_sector_ct: WARN Schwelle
- warn.udma_crc_error_count: WARN Schwelle
- delta.*: Delta-basierte Warnungen

RAID-Parameter

- enabled: RAID Check aktiv
- warn_degraded: Warnung bei degraded
- fail_failed_disk: Fehler bei defekter Disk
- warn_rebuild: Warnung bei Rebuild

generischen Check-Parameter

- enabled: Check aktivieren
- severity: warn / fail
- required: Liste benötigter Objekte
- ignore_units: Ignorierte Units
- critical_units: Kritische Services

systemd_timers Parameter

- required: Pflicht-Timer
- warn_if_missed_hours: Standardwert
- warn_if_missed_hours_map: Timer-spezifisch

smartd Modi

- auto: Nur wenn Devices vorhanden
- always: Immer erforderlich
- never: Nie erforderlich

Filesystem-Parameter

- capacity_warn_pct: Warnschwelle
- capacity_fail_pct: Fail-Schwelle
- inode_warn_pct: Warn Inodes
- inode_fail_pct: Fail Inodes

- zfs.scrub_max_age_days: Max Scrub Alter
- btrfs.metadata_warn_pct: Warn Metadaten
- nfs.response_timeout_sec: Timeout NFS
- cifs.response_timeout_sec: Timeout CIFS

Partition-Parameter

- audit_enabled: Audit aktiv
- warn_if_unmounted_with_fstype: Warnung
- include_dm_layers: Device Mapper
- include_whole_disks: Whole disks
- fstype_classes: Klassifizierung
- class_policy: Policy je Klasse

Inode-Forensik Parameter

- top_dirs: Top Verzeichnisse
- max_depth: Tiefe
- largest_files: Größte Dateien
- min_file_size_mb: Mindestgröße

scope_units Parameter

- enabled: Aktiv
- old_active_hours: Altersschwelle
- include_failed: Failed einbeziehen
- warn_on_unknown_failed: Warn bei unbekannt
- profile_numeric_scope_ignore: Ignorierte Scopes

b1113(26)

Datum: 18.04.2026
Titel: Vollständige Produktdokumentation inkl. JSON-Schema und Referenz
ChatGPT Version: GPT-5.3
Chat: Dokumentationsanpassung b1113(26)

Architektur des Systems

Der OSR Agent for Linux ist ein zustandsbasierter Health- und Inventory-Agent. Er kombiniert statische Checks mit persistenter Zustandsanalyse und Delta-Erkennung.

Wie funktioniert die Merge-Logik?

Reihenfolge der Konfigurationsauflösung:
1. defaults
2. profile
3. host
4. root overrides

Letzter gewinnt. Arrays werden ersetzt, nicht gemerged.

Pseudo-Schema v1.1 (formalisiert)

root:
 meta: object
 defaults: object
 profiles: map<string, profile>
 hosts: map<string, host>
 checks: optional overrides

profile:
 checks: object
 filesystem_health: object

filesystem_health:
 thresholds, expected_mounts, zfs, btrfs, nfs, cifs

checks:
 systemd_failed, journal_errors, smartd, firmware, raid, ecc, ipmi

Vollständige Parameter-Referenz (Auszug Kernbereiche)

Bereich	Parameter	Typ	Beschreibung
meta	namespace	string	Namensraum
meta	schema_version	string	Schema Version
defaults.logrotate	enabled	bool	Logrotation aktiv
defaults.mail	sendto	string	Empfänger
checks.systemd_failed	ignore_units	array	Ignorierte Services
checks.journal_errors	ignore_patterns	array	Regex Filter
checks.smartd	required_mode	enum	auto/always/never
filesystem_health	capacity_warn_pct	int	Warnschwelle
filesystem_health.zfs	scrub_max_age_days	int	Max Scrub Age
checks.firmware	warn_if_changed	object	Change Detection

checks.ecc	fail_uncorrected_delta	int	ECC Fehler
checks.ipmi	ignore_na_sensors	bool	Sensorfilter

Konkrete JSON-Beispiele

Komplette Minimal-Konfiguration:

```
{
  "meta": { "version": "b1113(26)" },
  "defaults": {},
  "profiles": {},
  "hosts": {}
}
```

Profil pve:

```
{
  "checks": {
    "services": { "required": ["postfix","ssh"] },
    "smartd": { "required_mode": "auto" }
  }
}
```

Profil nas:

```
{
  "checks": {
    "smartd": { "required_mode": "always" }
  }
}
```

Profil pbs:

```
{
  "checks": {
    "smartd": { "required_mode": "never" }
  }
}
```

Fehlerszenarien und Bewertung

PASS = alles ok
WARN = tolerierbare Abweichung
FAIL = kritischer Zustand

Run-State bleibt unabhängig davon technisch korrekt (rc=0/1).

Designprinzipien

- Trennung von Status und Severity
- Zustandsbasierte Analyse
- Erweiterbarkeit über Profile
- Wiederverwendbarkeit