Open Book

Ronny Trommer

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

If opportunity doesn't knock, build a door.

Milton Berle

1 | Provisioning System

1.1 Provisioning System

Eine der grösseren Aufgaben von Netzwerk-Management-Systemen (NMS) ist es, Möglichkeiten und Wege zu bieten, wie die reale Welt in das entsprechende Modell des NMS zu überführen. Genau diese Aufgabe erfüllt bei OpenNMS das *Provisioning System*. Es besteht aus einem Hintergrundprozess *provisiond*.

1.2 Provisioning - ReST API

The minute that you're not learning I believe you're dead.

Jack Nicholson

2 | Topologien und Karten

2.1 Karten und Topologien

2.1.1 Geografische Node-Karte

2.1.2 Topologie-Karte

2.1.3 Remote-Poller Karte

> The only source of knowledge is experience.
>
> *Albert Einstein*

3 | 3rd Party-Integration

3.1 DNS-Server Provisioning

3.2 VMware Integration

Virtualisierung ist ein wichtiger Bestandteil heutiger IT-Landschaften geworden. In Version 1.12 wurde aus diesem Grund das *Provisioning* erweitert. Es ist nun möglich direkt virtuelle Maschinen (VM) und Host-Systeme aus einer *VMware* basierenden Virtualisierung in *OpenNMS* zu importieren.

Typischerweise sind diese Informationen über das *vCenter* zugänglich. Leistungsdaten können dort allerdings nur über einen sehr begrenzten Zeitraum gespeichert werden.

3.2.1 Vorbereiten VMware vCenter

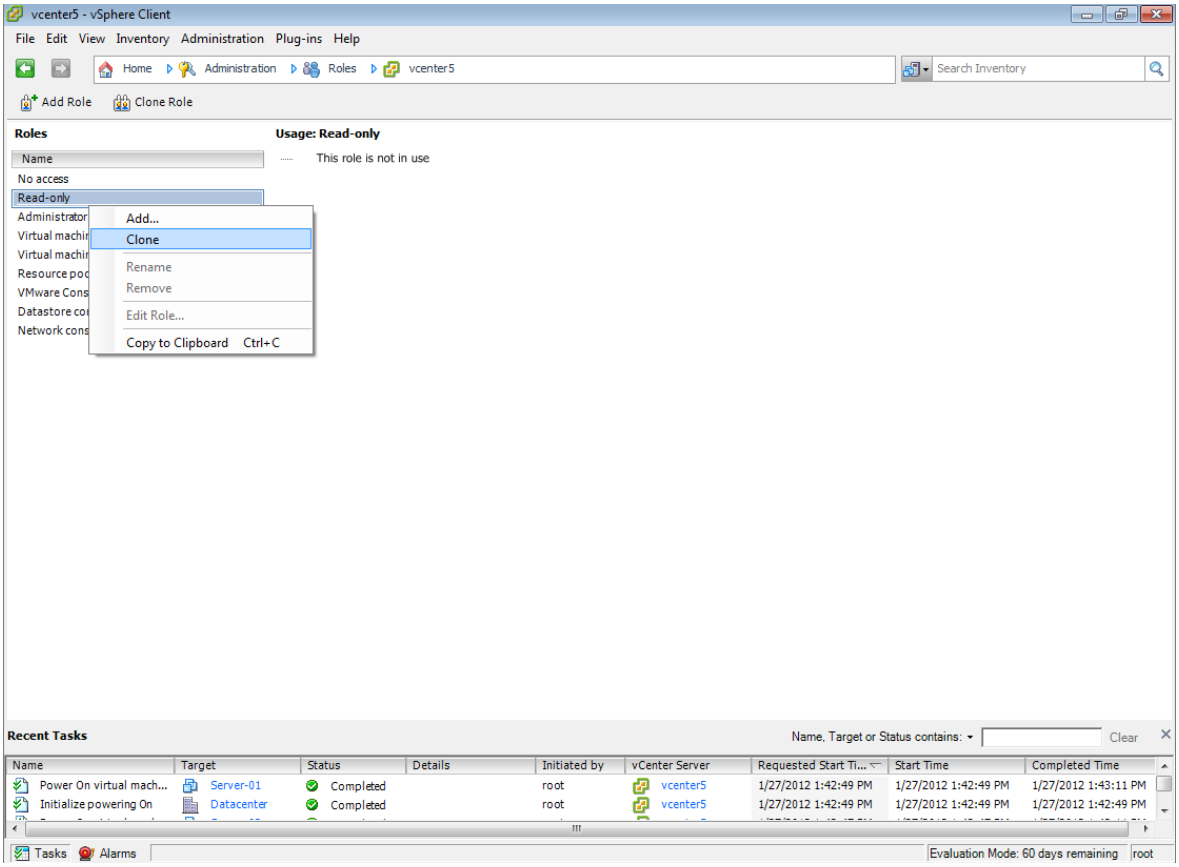

Abbildung 3.2.1: Duplizieren einer *Read-Only* Rolle in *vCenter*

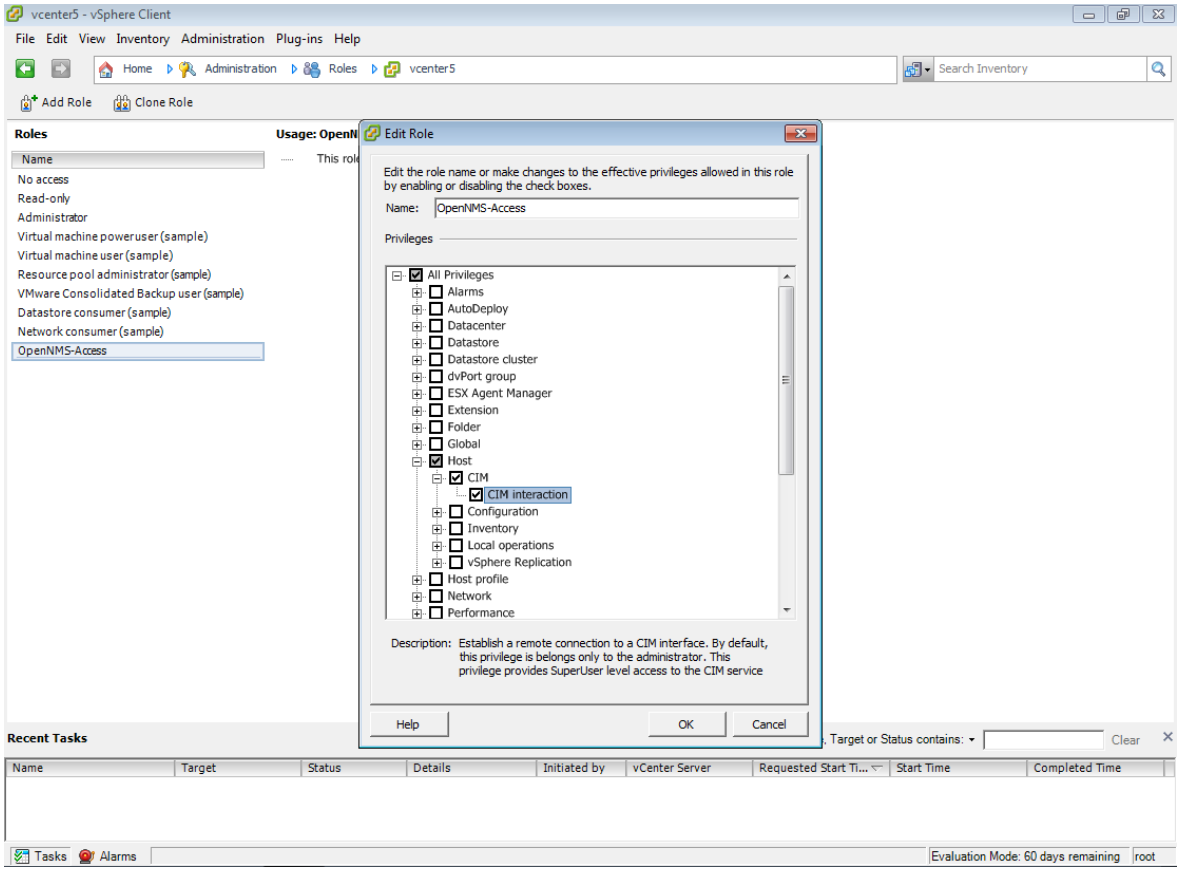

Abbildung 3.2.2: Duplizieren einer *Read-Only* Rolle in *vCenter*

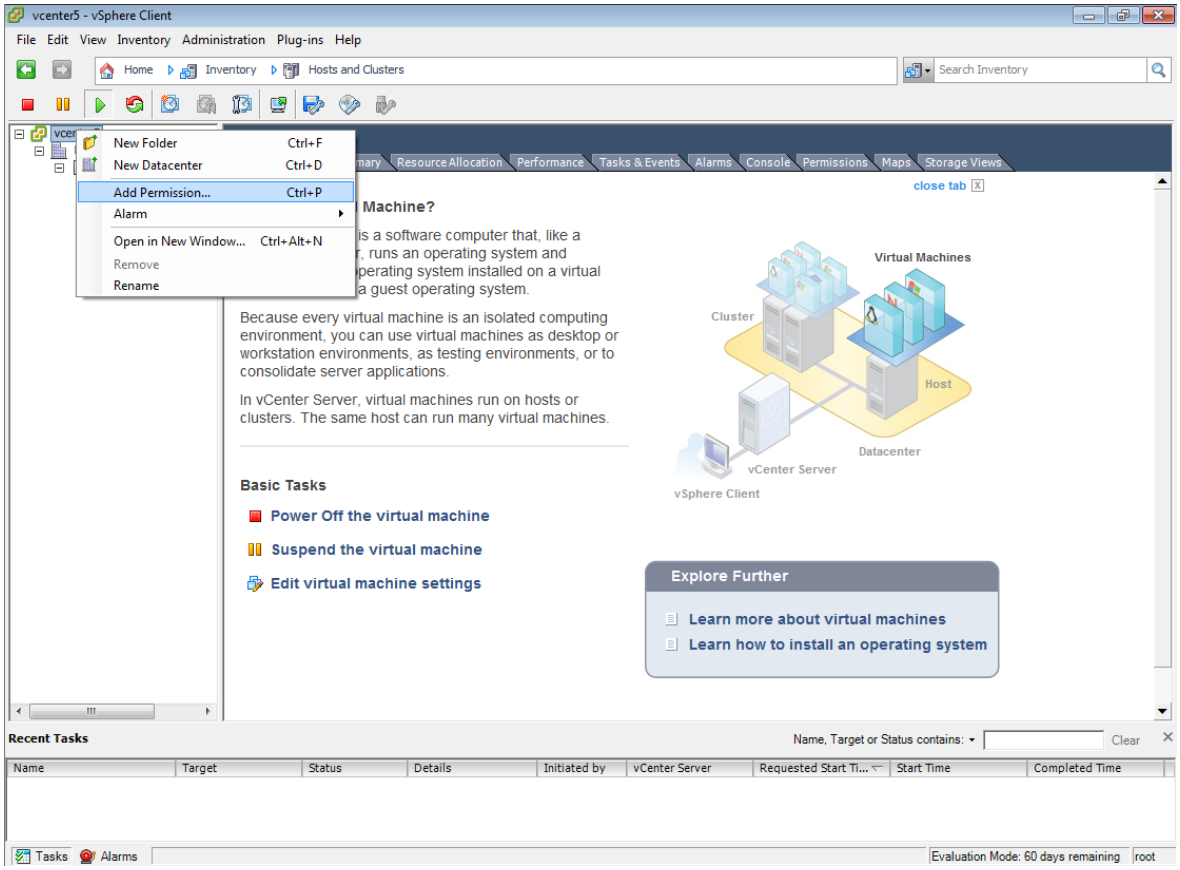

Abbildung 3.2.3: Duplizieren einer *Read-Only* Rolle in *vCenter*

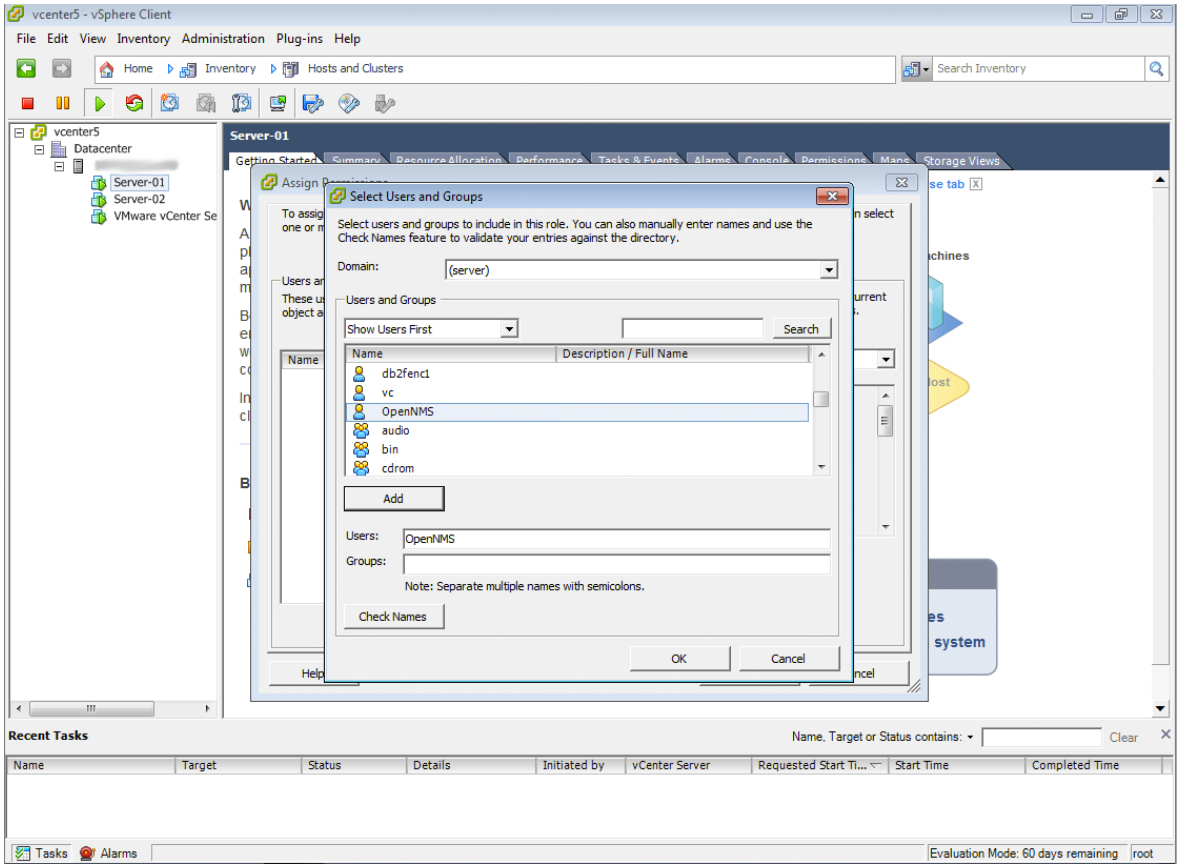

Abbildung 3.2.4: Duplizieren einer *Read-Only* Rolle in *vCenter*

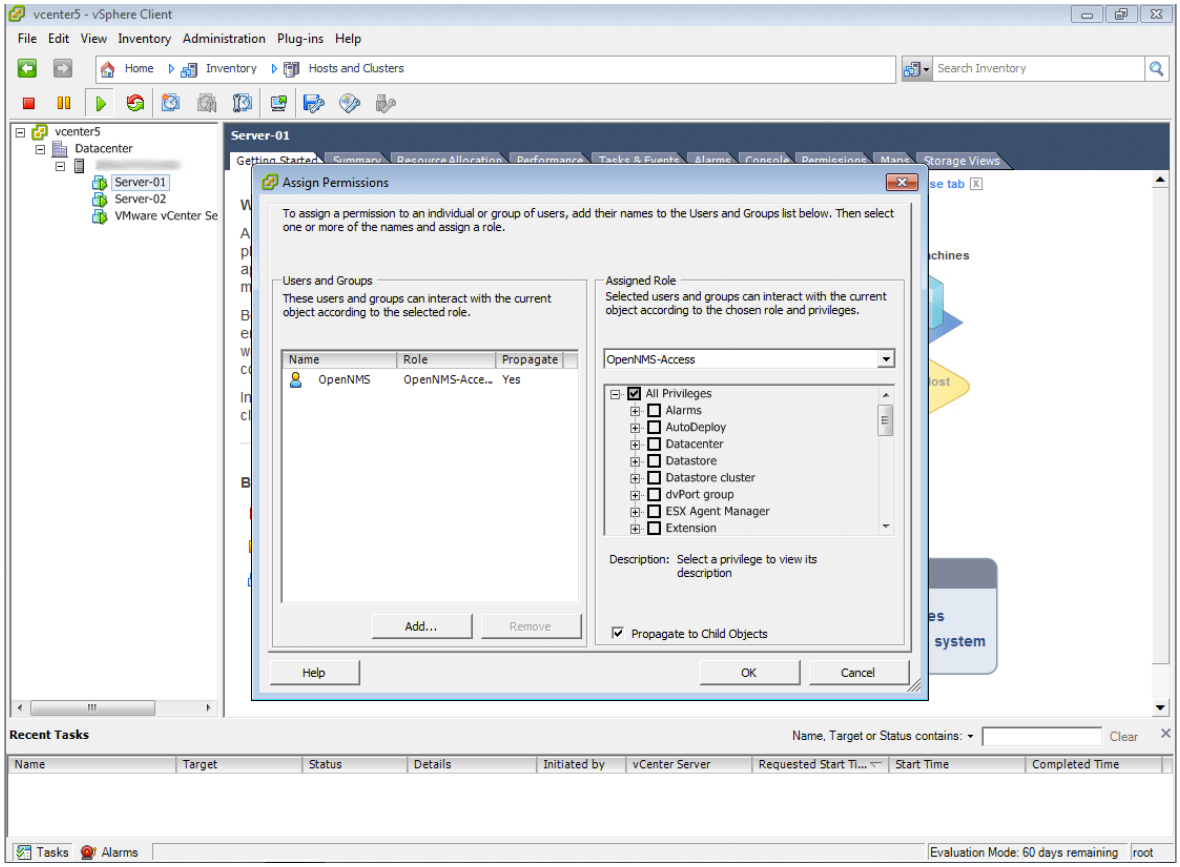

Abbildung 3.2.5: Duplizieren einer *Read-Only* Rolle in *vCenter*

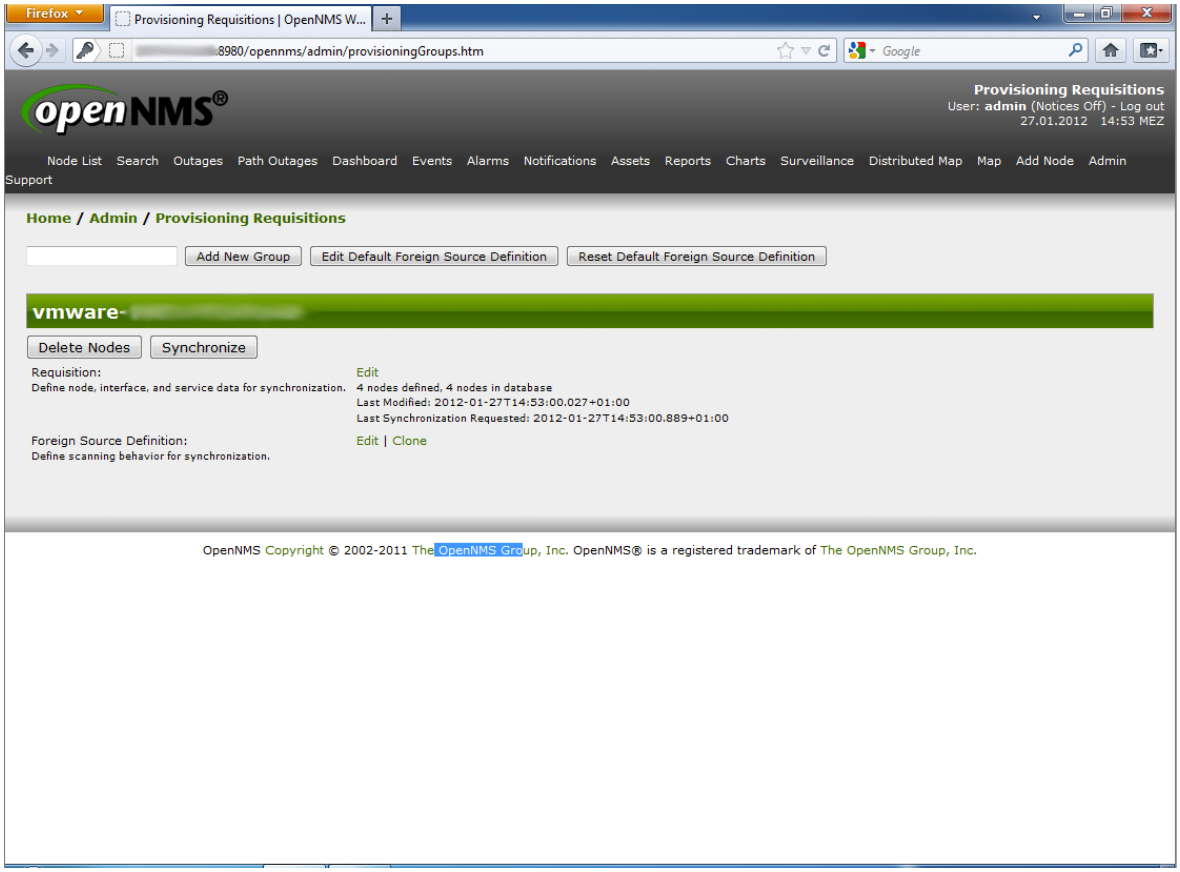

Abbildung 3.2.6: Duplizieren einer *Read-Only* Rolle in *vCenter*

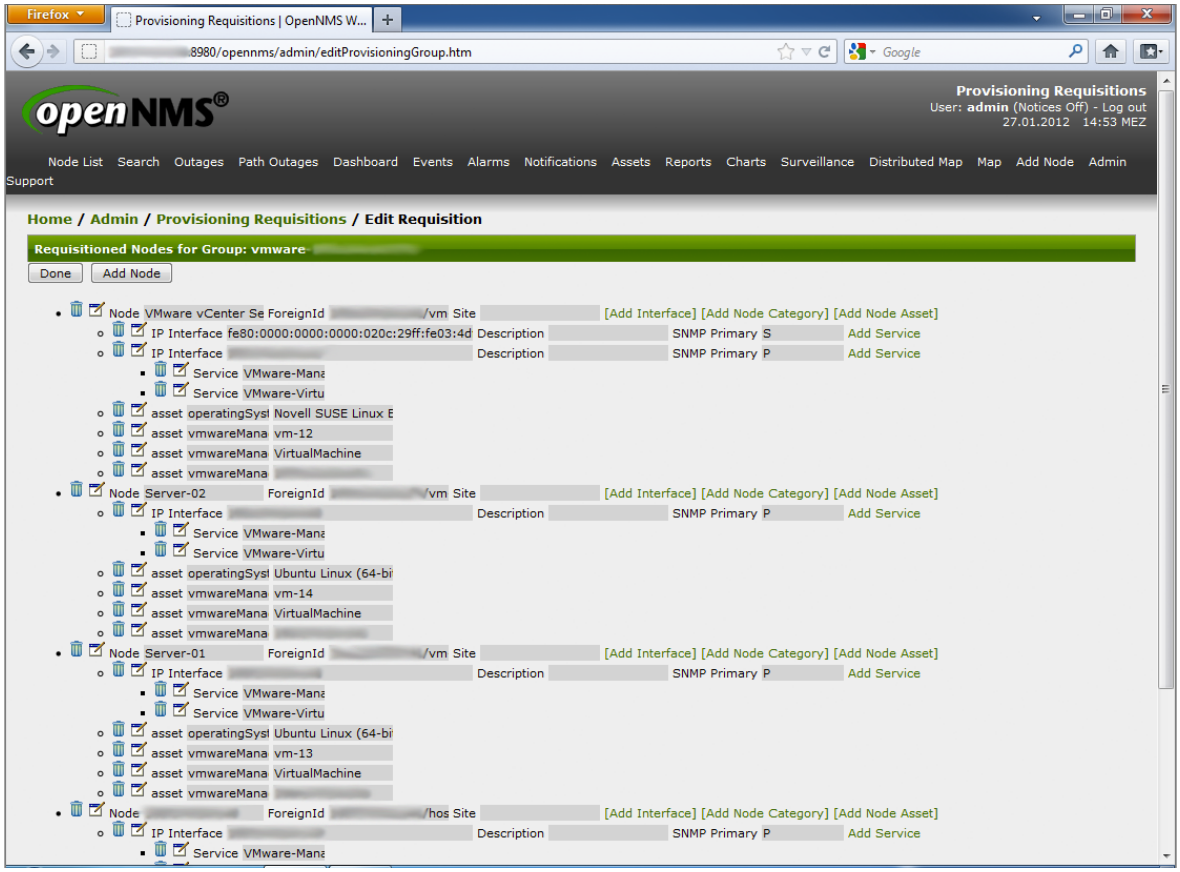

Abbildung 3.2.7: Duplizieren einer *Read-Only* Rolle in *vCenter*

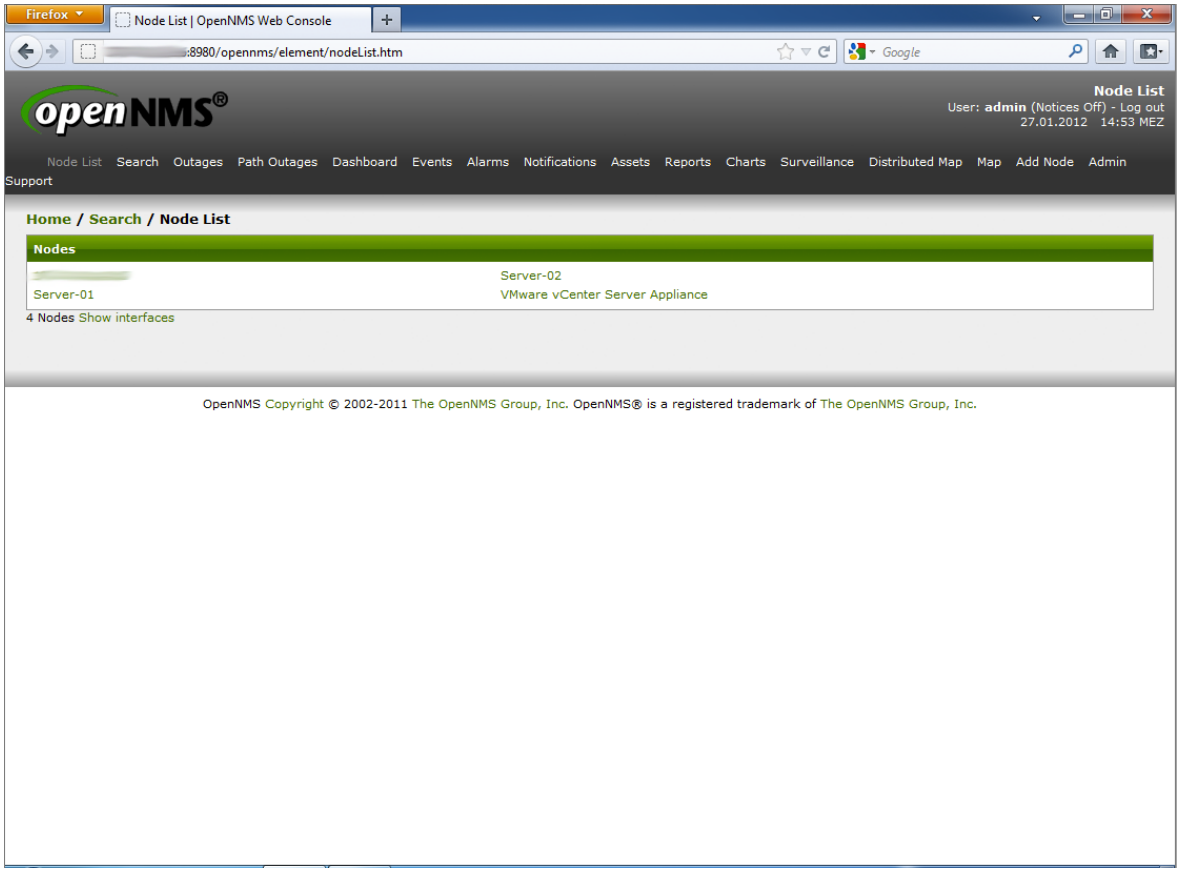

Abbildung 3.2.8: Duplizieren einer *Read-Only* Rolle in *vCenter*

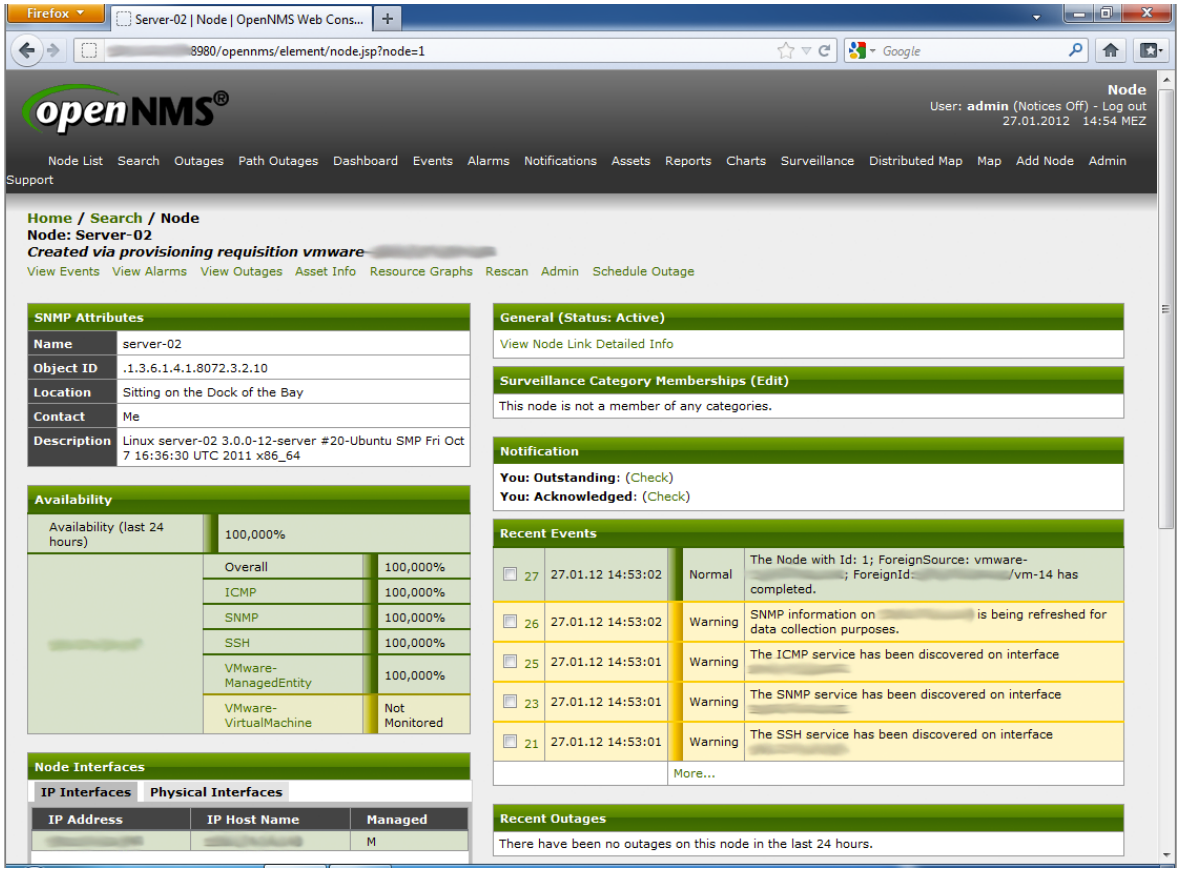

Abbildung 3.2.9: Duplizieren einer *Read-Only* Rolle in *vCenter*

3.2.2 VMware Provisioning Adapter

In *VMware vCenter* werden alle Komponenten verwaltet, die in der virtuellen Umgebung eingesetzt werden. Dort werden neben den VMs und Host-Systemen auch Leistungsdaten und der Hardwarestatus der Host-Systeme ermittelt und zur Verfügung gestellt. Um die VMs und Host-Systeme von einem bestehenden *vCenter* zu importieren wurde das *Provisioning* um einen *Requisition-Handler* erweitert. Dieser verbindet sich zum *vCenter* und liest dort alle notwendigen Informationen der VMs und Host-Systeme und überführt diese in das *OpenNMS Model* für den Import.

Für den Import werden die folgenden Informationen aus vCenter ausgelesen:

-

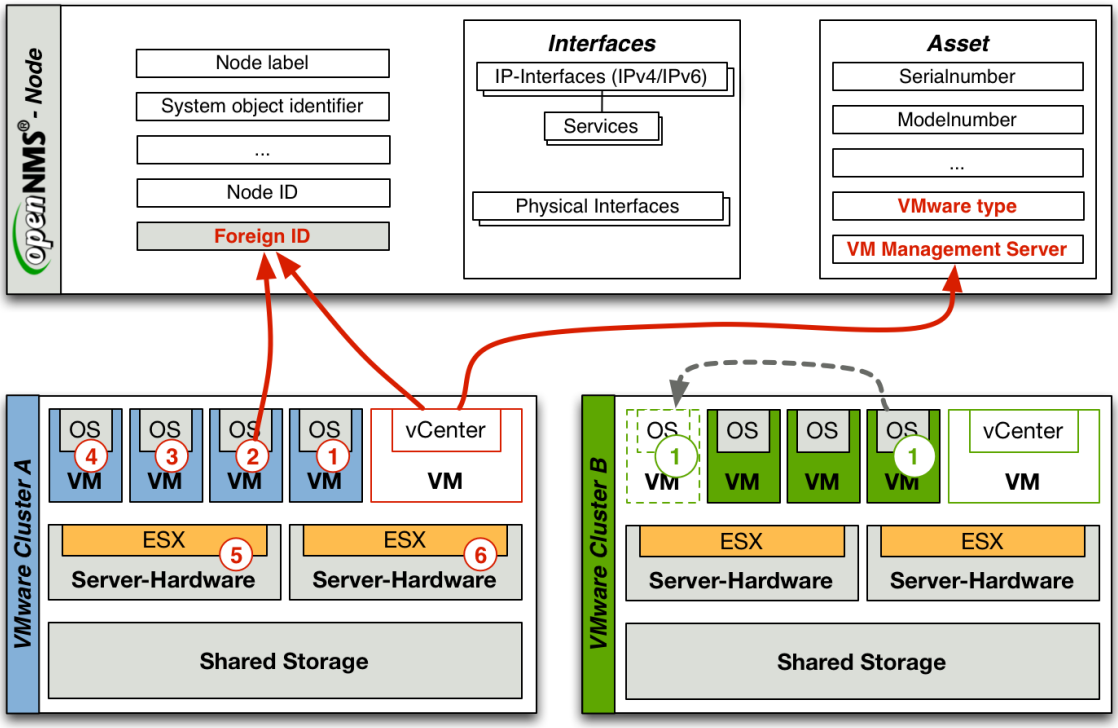

Abbildung 3.2.10: *Provisioning* mit mehreren *VMware Clustern*

3.2.3 Leistungsdaten von VMware Umgebung

3.2.4 Hardwarestatus von VMware Hosts

3.2.5 VMware und die Topology Karte

3.3 OCS-Inventory Provisioning

And now for something completely different.

Monty Pyton

4 | Anwendungsfälle

4.1 Layer 2 - Monitoring

Für das Überwachen von Netzwerkverbindungen wird *ICMP*[1] der Internetprotokollfamilie verwendet. Es genügt um festzustellen, ob ein Gerät über das Netzwerk noch erreichbar ist oder nicht. Befinden sich redundante Pfade im Netzwerk, kann eine Störung über *ICMP* nicht festgestellt werden, da die *ICMP-Pakete* transparent über einen redundanten Pfad geleitet werden. Die Überwachung wird mit *ICMP* auf der Vermittlungs- oder Netzwerkschicht, also auf *IP-Ebene* durchgeführt. In vielen Fällen werden Fehler oder Statusänderungen auf Basis von *SNMP-Traps*[2] übermittelt. Der Status eines Netzwerkports wird beispielsweise mit einem *Link Down* oder *Link Up* Trap dem Managementserver mitgeteilt.

In vielen Fällen wird entweder ein *SNMP-Trap* für alle oder keinen der Netzwerkports gesendet. Eine granulare Konfiguration für welchen der Netzwerkports Traps gesendet werden sollen fehlt häufig.

In dem folgenden Versuchsaufbau sollen drei wichtige Switch-Ports überwacht werden. Um festzustellen welche Ports für Monitoring relevant sind, verwenden wir eine benutzerdefinierte *Port-Description*. Als Switch wird ein *Cisco 3524XL* mit *IOS 12.0(5)WC17* verwendet. *OpenNMS* ist in der Version 1.8.1 auf einem *Ubuntu Server 9.10* installiert. Es sind noch keine Geräte in *OpenNMS* eingetragen und es findet noch keine Überwachung statt. Der Versuchsaufbau ist in der folgenden Abbildung dargestellt.

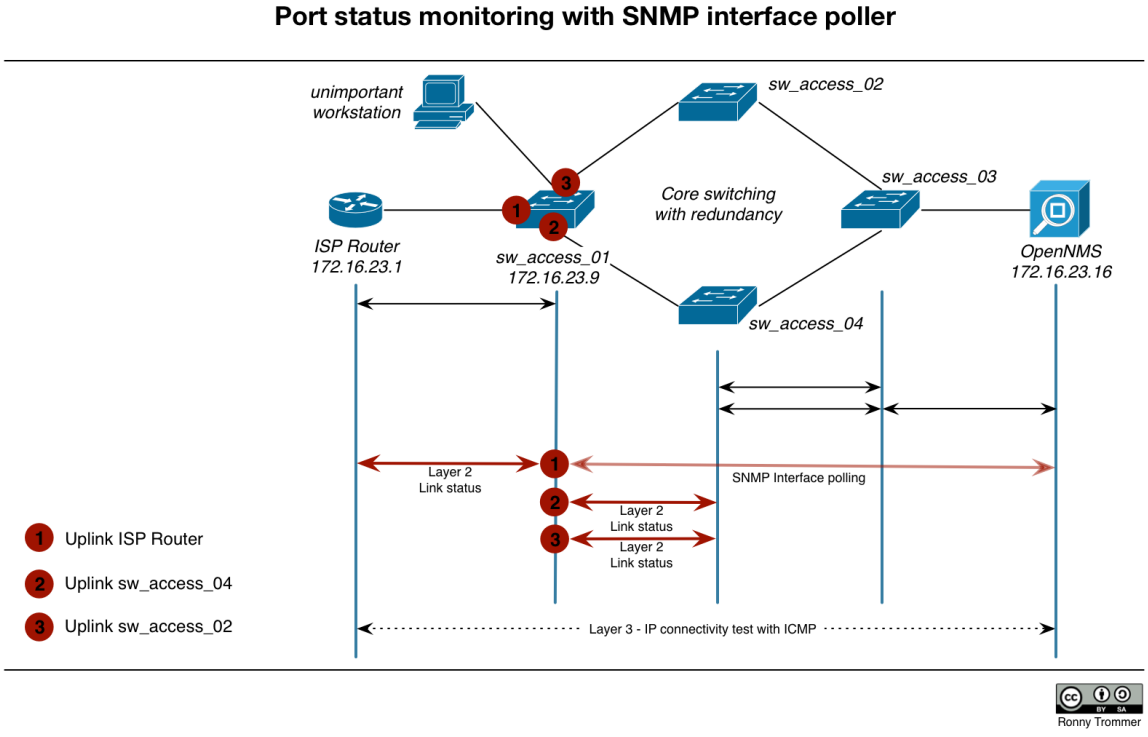

Abbildung 4.1.1: Versuchsaufbau

[1] Protokoll zum Test der IP-Konnektivität wie beispielsweise mit dem Kommando `ping`

[2] SNMP Traps sind Fehlermeldungen die von einer Netzwerkkomponente an einen Managementserver gesendet werden.

Es sollen auf einem Switch mit der Bezeichnung *sw_ access_ 01*, drei wichtige Ports überwacht werden. Für die Überwachung relevant ist der Link zum *ISP*[3] Router sowie auch die beiden Uplinks zum redundant ausgelegten Core. Der Port mit der Workstation interessiert uns in der Überwachung nicht und soll entsprechend ignoriert werden. In dem folgendem Versuch wird gezeigt wie *SNMP* auf dem Switch eingerichtet werden kann und in *OpenNMS* aufgenommen wird. Es werden lediglich Leistungsdaten von relevanten Ports aufgezeichnet. Danach wird der *SNMP-Interface Poller* für die entsprechenden Ports eingerichtet.

4.1.1 SNMP-Ready Cisco Switch

Bevor wir überhaupt mit der Überwachung anfangen können, muss *SNMP* auf den Geräten eingerichtet werden. Die Einrichtung wird detailiert auf `http://www.cisco.com` unter *"Configuring SNMP 3"*[4] beschrieben. Es werden hier nur die rudimentären Schritte zur Einrichtung von *SNMP* beschrieben. Die grundlegende Einrichtung von Authentifizierung, Hostnamen sowie IP-Adressen ist bereits vorgenommen. Wir melden uns per *Telnet* auf unserem Switch an und aktivieren *SNMP* im *Read-Only* Modus und setzen eine *location* und einen *contact*.

```
 1  sw_access_01>enable
 2  Password:
 3  sw_access_01#configure terminal
 4  sw_access_01(config)#snmp-server community notpublic ro
 5  sw_access_01(config)#snmp-server contact <ronny@opennms.org> Ronny Trommer
 6  sw_access_01(config)#snmp-server location Springfield HQ down in the Basement
 7  sw_access_01#sh run | i snmp
 8  snmp-server community notpublic RO
 9  snmp-server location Springfield HQ down in the Basement
10  snmp-server contact <ronny@opennms.org> Ronny Trommer
11  sw_access_01#copy run start
12  Destination filename [startup-config]?
13  Building configuration...
14  [OK]
15  sw_access_01#
```

Listing 4.1: SNMP-Konfiguration auf Cisco Switch

Damit ist es nun möglich vom *OpenNMS-Server SNMP* Abfragen durchzuführen. Im Anschluss werden die entsprechenden *Port-Description* gesetzt.

```
 1  sw_access_01#configure terminal
 2  sw_access_01(config)#interface fastEthernet 0/1
 3  sw_access_01(config-if)#description Uplink ISP Router
 4  sw_access_01(config-if)#no shutdown
 5  sw_access_01(config)#interface fastEthernet 0/2
 6  sw_access_01(config-if)#description Uplink sw_access_04
 7  sw_access_01(config-if)#no shutdown
 8  sw_access_01(config)#interface fastEthernet 0/3
 9  sw_access_01(config-if)#description Uplink sw_access_02
10  sw_access_01(config-if)#no shutdown
11  sw_access_01(config)#interface fastEthernet 0/4
12  sw_access_01(config-if)#description Workstation
13  sw_access_01(config-if)#no shutdown
```

[3] Internet Service Provider stellt den Übergabepunkt zu Internetdiensten bereit.
[4] Configuring SNMP – `http://www.cisco.com/en/US/docs/ios/12_2/configfun/configuration/guide/fcf014.html`

```
14 sw_access_01#copy run start
15 Destination filename [startup-config]?
16 Building configuration...
17 [OK]
18 sw_access_01#
```

Listing 4.2: Konfiguration der Netzwerk-Ports

Hinweis: Es ist häufig sinnvoll, mit Hilfe von *Access Listen*, Zugriffe nur von IP-Adressen des Management-Servers zu gestatten. Schauen Sie sich hier die entsprechenden *ACLS* an.

Um die SNMP Konfiguration zu testen, können die gesetzten Beschreibungen mit dem Kommando

```
snmpwalk -v 2c -c notpublic 172.16.23.9 IF-MIB::ifAlias
```

vom *OpenNMS-Server* abgerufen werden. Die Ausgabe sollte dann in etwa so aussehen:

```
 1 IF-MIB::ifAlias.1 = STRING:
 2 IF-MIB::ifAlias.2 = STRING: Uplink ISP Router
 3 IF-MIB::ifAlias.3 = STRING: Uplink sw_access_04
 4 IF-MIB::ifAlias.4 = STRING: Uplink sw_access_02
 5 IF-MIB::ifAlias.5 = STRING: Workstation
 6 IF-MIB::ifAlias.6 = STRING:
 7 IF-MIB::ifAlias.7 = STRING:
 8 IF-MIB::ifAlias.8 = STRING:
 9 IF-MIB::ifAlias.9 = STRING:
10 IF-MIB::ifAlias.10 = STRING:
11 IF-MIB::ifAlias.11 = STRING:
12 IF-MIB::ifAlias.12 = STRING:
13 IF-MIB::ifAlias.13 = STRING:
14 IF-MIB::ifAlias.14 = STRING:
15 IF-MIB::ifAlias.15 = STRING:
16 IF-MIB::ifAlias.16 = STRING:
17 IF-MIB::ifAlias.17 = STRING:
18 IF-MIB::ifAlias.18 = STRING:
19 IF-MIB::ifAlias.19 = STRING:
20 IF-MIB::ifAlias.20 = STRING:
21 IF-MIB::ifAlias.21 = STRING:
22 IF-MIB::ifAlias.22 = STRING:
23 IF-MIB::ifAlias.23 = STRING:
24 IF-MIB::ifAlias.24 = STRING:
25 IF-MIB::ifAlias.25 = STRING:
26 IF-MIB::ifAlias.26 = STRING:
27 IF-MIB::ifAlias.27 = STRING:
28 IF-MIB::ifAlias.28 = STRING:
```

Listing 4.3: *SNMP walk* für die Anzeige der *Port-Description*

Damit sind die Voraussetzungen für das Monitoring in OpenNMS erfüllt und wir können den Switch in OpenNMS für die Überwachung eintragen.

4.1.2 Cisco Switch Provisioning

Bevor wir Geräte eintragen, sollte die verwendete *SNMP-Community* bekannt sein. Hierzu gibt es zwei Möglichkeiten, entweder die *default SNMP-Community* ist in der Datei

```
$OPENNMS_HOME/etc/snmp-config.xml
```

entsprechend setzen oder die *Community* wird für die einzelne IP-Adresse oder einen IP-Bereich über die Web-Oberfläche gesetzt. Um die *SNMP-Community* für unseren Switch mit der IP 172.16.23.9 zu setzen, wird das Formular in der Web-Oberfläche wie folgt ausgefüllt.

```
Admin --> Configure SNMP Community Names by IP
```

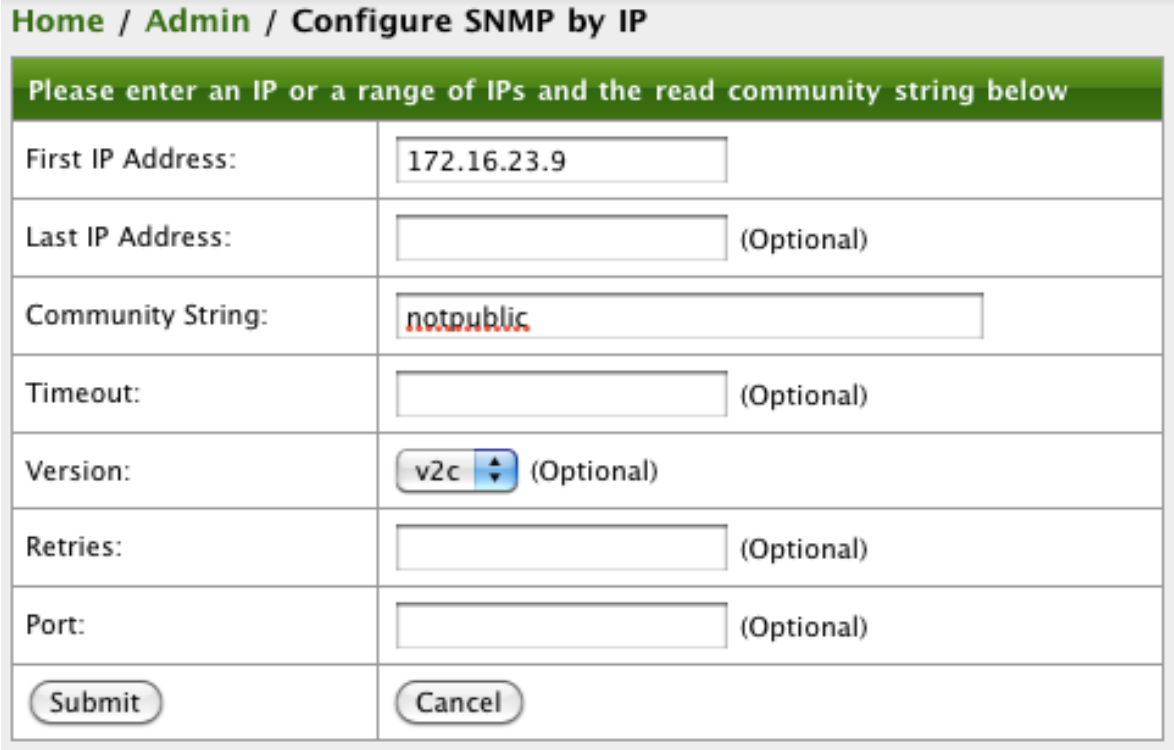

Abbildung 4.1.2: SNMP Community für IP-Adressen konfigurieren

Im nächsten Schritt wird der Switch über das *Provisioning* von *OpenNMS* aufgenommen. Wir erzeugen eine neue *Requisition Group* über

```
Admin --> Manage Provisioning Groups --> Add New Group
```

mit der Bezeichnung *OpenNMS Lab*. Wir erstellen eine *Policy*[5] die nur Leistungsdaten von relevanten und aktiven Ports aufzeichnet. Zusätzlich soll nur *ICMP* und *SNMP* für den Switch überwacht werden. Die *Policy* sieht dazu wie folgt aus:

[5] *Policies* beschreiben Richtlinien für das Monitoring in *OpenNMS* und werden während des Import in die *OpenNMS-Datenbank* angewendet.

Abbildung 4.1.3: Richtlinie für den Import des *Cisco* Switch

Alle Detektoren ausser *ICMP* und *SNMP* wurden entfernt. Zusätzlich wurde eine *datacollection_policy* angelegt, die nur Leistungsdaten von Ports aufzeichnet, welche aktiv sind und[6] eine Port-Beschreibung[7] mit *Uplink* beginnend besitzen. Die Geräte in der *Requisition Group* wird täglich[8] einmal synchronisiert. Der Switch wird nun unter

```
Requisition (Provisioning Group): EDIT --> Add Node
```

eingetragen und synchronisiert.

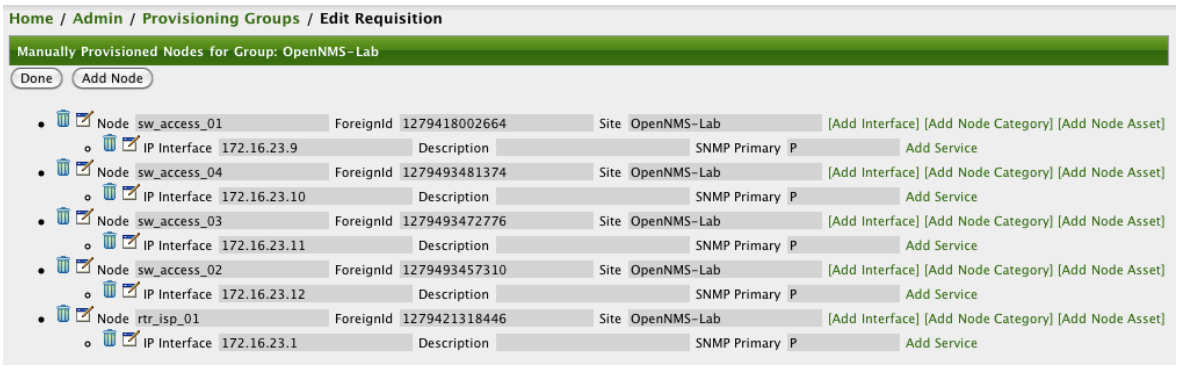

Abbildung 4.1.4: Import der *Requisition Group* in *OpenNMS*

Die Konfiguration wird bestätigt, über die Schaltfläche *Synchronize* importiert und im Monitoring bereitgestellt. Nach dem Import stellt sich der Switch in OpenNMS wie folgt dar:

[6] Die Verknüpfung stammt aus dem Konfigurationsschlüssel *matchBehavior*, *oder=ANY_PARAMETER*, *Negation=NON_PARAMETERS*

[7] In der *SNMP MIB II* ist *ifDescr* die Interface-Bezeichnung *(FastEthernet 0/1)* und der *ifAlias* die Port Beschreibung

[8] Definiert über den Standardparameter *Scan Interval 1d*

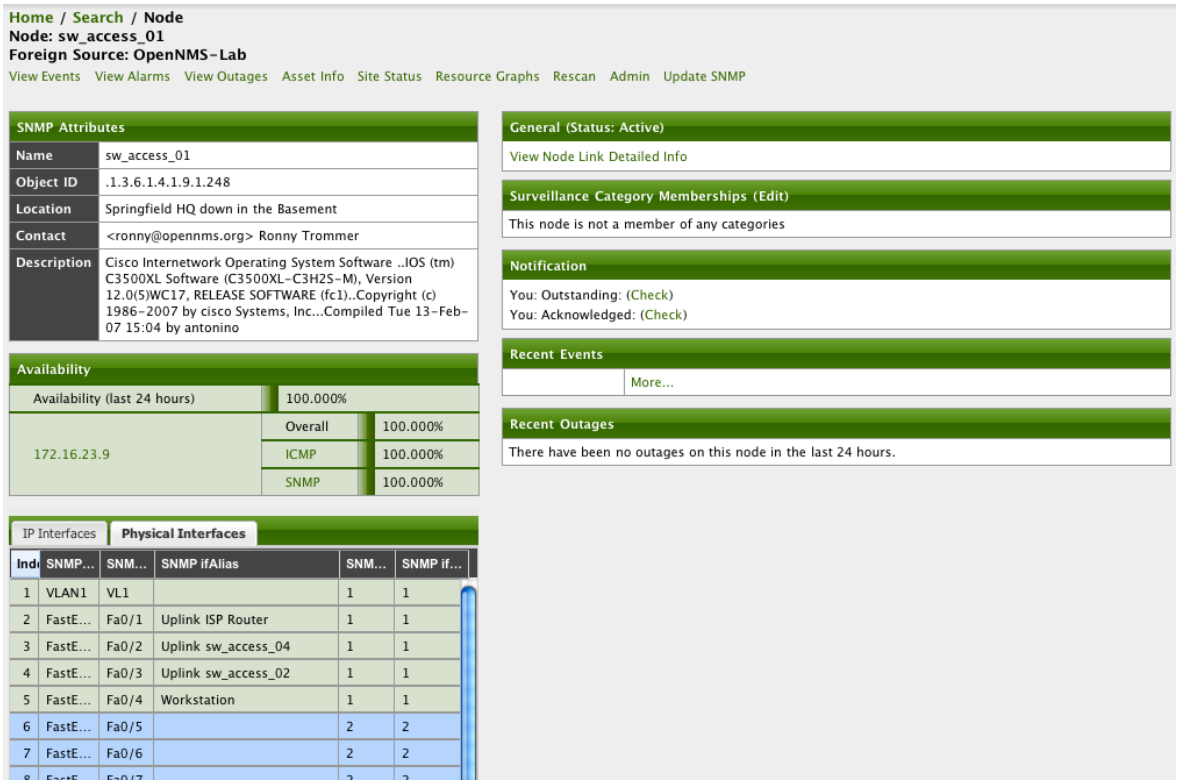

Abbildung 4.1.5: *Node Page* des Cisco-Switch nach dem Import in *OpenNMS*

Zur besseren Darstellung wurden die Spalten mit der Interface-Geschwindigkeit und IP-Adresse ausgeblendet und der *Port-Status* hinzugefügt. Die grün hinterlegten Ports sind aktiv. Die blau hinterlegten Ports sind administrativ deaktiviert. Der Switch wird nun täglich mit den entsprechenden Port-Beschreibungen in OpenNMS synchronisiert und die entsprechenden Informationen im Monitoring bereitgestellt. Es werden zusätzlich nur Leistungsdaten von Ports mit der Beschreibung *Uplink* aufgezeichnet.

4.1.3 Port Status kontrollieren

Zu Begin muss der dazu verwendete *SNMP Interface Poller* aktiviert werden. In der Datei

```
$OPENNMS_HOME/etc/service-configuration.xml
```

wird dazu der entsprechende Dienst auskommentiert.

```
1 <service>
2   <name>OpenNMS:Name=SnmpPoller</name>
3   <class-name>
4        org.opennms.netmgt.snmpinterfacepoller.jmx.SnmpPollerd
5   </class-name>
6   <invoke at="start" pass="0" method="init"/>
7   <invoke at="start" pass="1" method="start"/>
8   <invoke at="status" pass="0" method="status"/>
9   <invoke at="stop" pass="0" method="stop"/>
```

```
10 </service>
```

Listing 4.4: Der *SNMP Interface Poller* ist nach der Installation von *OpenNMS* einkommentiert.

Im nächsten Schritt wird ein *Interface Polling Package* angelegt. In diesem wird festgelegt, welche Ports zu überwachen sind.

```
    vi $OPENNMS_HOME/etc/snmp-interface-poller-configuration.xml
```

Die package-Beschreibung wird wie folgt aufgebaut:

```
 1 <package name="Uplink-Port-Monitoring">
 2   <filter>IPADDR != '0.0.0.0'</filter>
 3   <include-range begin="1.1.1.1" end="255.255.255.254" />
 4   <interface name="important_ports"
 5              criteria="snmpifalias like 'Uplink%'"
 6              interval="300000"
 7              timeout="3000"
 8              retry="3"
 9              max-vars-per-pdu="10"
10              user-defined="false"
11              status="on"/>
12 </package>
```

Listing 4.5: *Polling-Package* für den *SNMP-Interface Poller*

Damit nur Ports mit der Beschreibung *Uplink* überwacht werden, wird ein Filterkritierum angegeben. Dieses lässt sich logisch mit weiteren Kriterien mit *AND* und *OR* verknüpfen. Mit *snmpifalias like 'Uplink%'* wird der entsprechende *Alias* aus der Datenbank ausgelesen und überwacht. Der Port Status wird alle 5 Minuten getestet und aktualisiert. Es können als Bedingung alle Felder aus der Tabelle *snmpinterface* verwendet werden. In der Version 1.8.1 sind folgende Spalten verwendbar:

Bezeichnung	Beschreibung
nodeid	Eindeutige ID für den Knoten in OpenNMS.
ipaddr	RFC-1213-MIB: The media-dependent physical address. Setting this object to a null string (one of zero length) has the effect of invaliding the corresponding entry in the atTable object. That is, it effectively dissasociates the interface identified with said entry from the mapping identified with said entry. It is an implementation-specific matter as to whether the agent removes an invalidated entry from the table. Accordingly, management stations must be prepared to receive tabular information from agents that corresponds to entries not currently in use. Proper interpretation of such entries requires examination of the relevant atPhysAddress object.
snmpipadentnetmask	RFC1213-MIB: The NetworkAddress (e.g., the IP address) corresponding to the media-dependent 'physical' address.
snmpphysaddr	RFC-1213-MIB: The interface's address at the protocol layer immediately 'below' the network layer in the protocol stack. For interfaces which do not have such an address (e.g., a serial line), this object should contain an octet string of zero length.
snmpifdescr	RFC-1213-MIB: A textual string containing information about the interface. This string should include the name of the manufacturer, the product name and the version of the hardware interface.
snmpiftype	RFC-1213-MIB (Nur ein Auszug): Interface type other(1), ethernetCsmacd(6), basicISDN(20), primaryISDN(21), propPointToPointSerial(22), ppp(23),softwareLoopback(24), frameRelay(32), atm(37), aal5(49), isdn(63), adsl(94), sdsl(96), vdsl(97), pppMultilinkBundle(108), atmVirtual(149),mplsTunnel(150), mpls166), adsl2(230), adsl2plus(238) RFC-1213-MIB: The type of interface, distinguished according to the physical/link protocol(s) immediately 'below' the network layer in the protocol stack.
snmpifname	Bei Cisco Kurzbezeichnung Fa/0/2
snmpifspeed	RFC-1213-MIB: An estimate of the interface's current bandwidth in bits per second. For interfaces which do not vary in bandwidth or for those where no accurate estimation can be made, this object should contain the nominal bandwidth.
snmpifadmin	Administrativer Status: up(1), down(2), testing(3) RFC-1213-MIB: The desired state of the interface. The testing(3) state indicates that no operational packets can be passed.
snmpifoperstatus	Operativer Status: up(1), down(2), unknown(4), dormant(5), notPresent(6), lowerLayerDown(7) RFC-1213-MIB: The current operational state of the interface. The testing(3) state indicates that no operational packets can be passed.
snmpifalias	Eigene Port Beschreibung
snmpcollect	(N) Not Collect oder (C) Collect.Gibt an ob Leistungsdaten aufgezeichnet werden sollen oder nicht.

Tabelle 4.1.1: Verwendbare Attribute der SNMP interface table

Die vollständige Konfigurationsdatei wird im folgenden komplett dargestellt.

```
<?xml version="1.0"?>
<?castor class-name= \
    "org.opennms.netmgt.snmpinterfacepoller.SnmpInterfacePollerConfiguration"?>
<snmp-interface-poller-configuration threads="30" service="SNMP">
  <node-outage>
    <critical-service name="ICMP" />
    <critical-service name="SNMP" />
  </node-outage>
<!-- DEACTIVATE OPENNMS DEFAULT AND KEEP IT UNTOUCHED FOR EASIER UPGRADE
  <package name="example1">
    <filter>IPADDR != '0.0.0.0'</filter>
    <include-range begin="1.1.1.1" end="1.1.1.1" />
    <interface name="Ethernet" criteria="snmpiftype␣=␣6" \
      interval="300000" user-defined="false" status="on"/>
  </package>
-->
  .
  .
  .
  <!-- ########################################################## -->
  <!-- ###### START ###### - Customized Layer-2 port monitoring -->
  <!-- ########################################################## -->

  <package name="Uplink-Port-Monitoring">
    <filter>IPADDR != '0.0.0.0'</filter>
    <include-range begin="1.1.1.1" end="255.255.255.254"/>
    <interface name="important_ports" \
      criteria="snmpifalias␣like␣'Uplink%'" interval="10000" \
      timeout="3000" retry="3" max-vars-per-pdu="10" \
      user-defined="false" status="on"/>
  </package>

  <!-- ########################################################## -->
  <!-- ###### E N D ###### - Customized Layer-2 port monitoring -->
  <!-- ########################################################## -->
</snmp-interface-poller-configuration>
```

Listing 4.6: SNMP Interface Poller konfiguration.

Damit die Einstellung wirksam werden, muss OpenNMS neu gestartet werden.

```
service opennms restart
```

4.1.4 Funktionstest

Im nächsten Schritt prüfen wir das Verhalten von OpenNMS wenn verschiedene Ports den Status ändern. Dazu werden zwei Testszenarien geprüft:

1. Ein Uplink Port und der Workstation Port werden abgezogen und ein Ausfall simuliert. Es sollte nur ein Port Down Event auf dem Uplink Port erzeugt werden.

2. Ein Uplink Port und der Workstation Port werden administrativ down gesetzt. Zusätzlich simulieren wir eine echte Störung auf dem Uplink zum ISP-Router.

In der Ausgangssituation läuft alles wie gewünscht. Alle Ports sind aktiv und funktionsbereit.

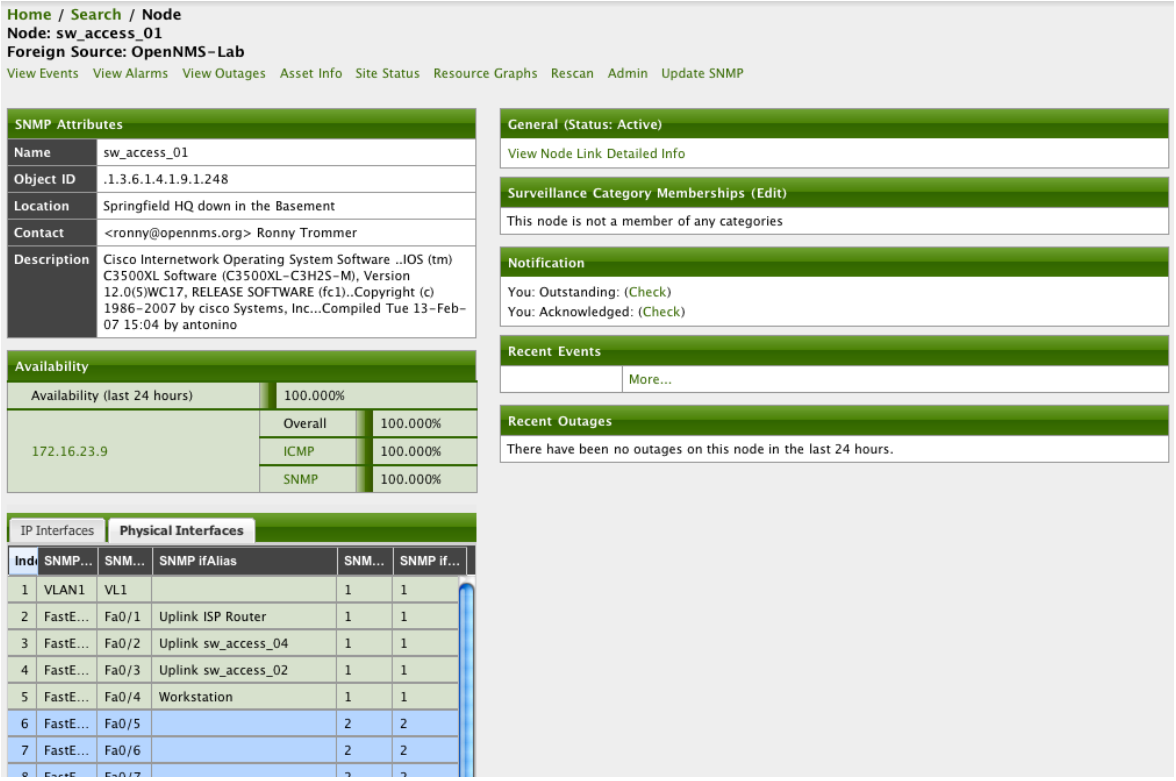

Abbildung 4.1.6: Node page mit Port status für *Physical Interfaces*.

Szenario 1: Port Störungen

Die Ports *Fa0/3* und *Fa0/4* wurden abgezogen und simulieren einen Ausfall. Wir sehen, dass der Portstatus nur für den Port *Fa0/3* aktualisiert wurde.

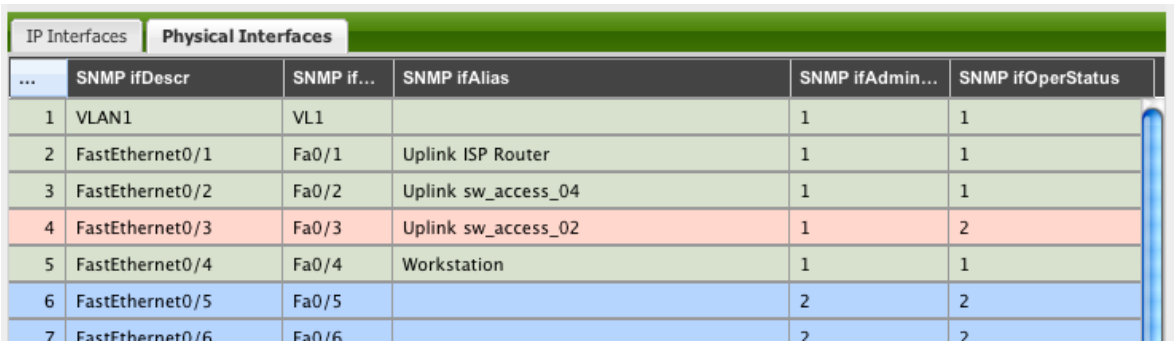

Abbildung 4.1.7: Anzeige Port Störung eines *Physical Interfaces*.

OpenNMS erzeugt zusätzlich ein entsprechendes Event und protokolliert damit das Ereignis.

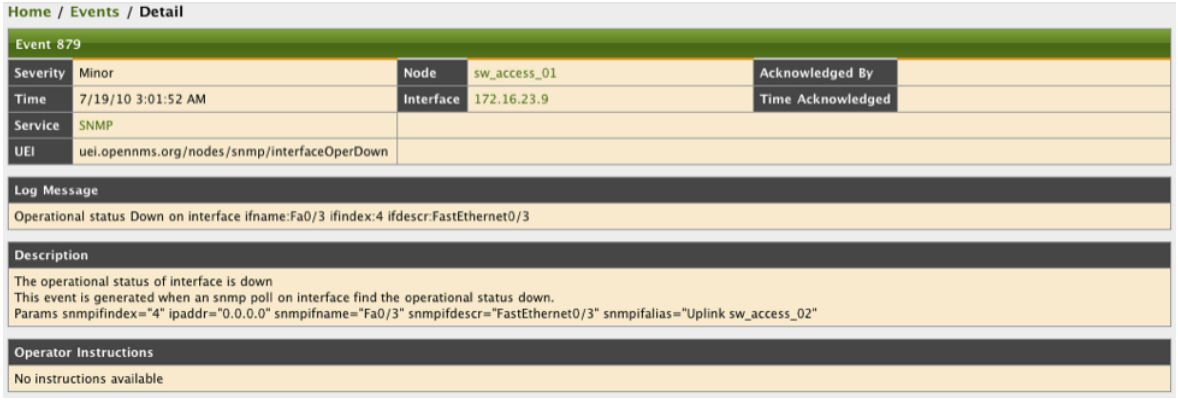

Abbildung 4.1.8: *Event* bei Port-Störung.

Der Status des Ports *Fa0/4* wird nicht durch den *SNMP-Interface Poller* aktualisiert. Es wurde ja explizit angegeben, dass nur *Uplink* Ports überwacht werden. Erst bei einer neuen Synchronisation der kompletten *Provisioning Group* wird auch der entsprechende Port-Status der Workstation wieder mit übernommen. Das Interval beträgt einen Tag. Wir synchronisieren manuell und der entsprechende Port Status wird wie folgt angzeigt.

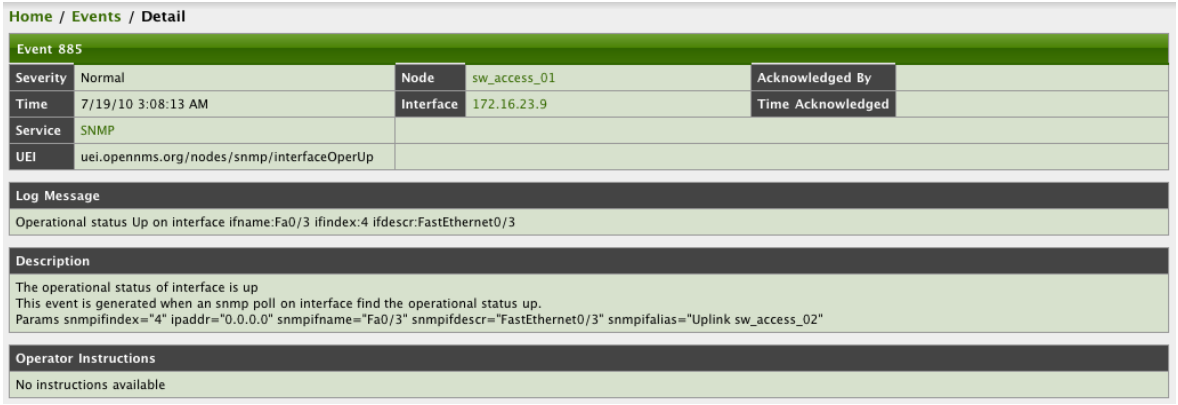

Abbildung 4.1.9: *Event* beim auflösen der Port-Störung.

Damit ist der erste Testfall erfolgreich abgeschlossen. Im zweiten Testfall prüfen wir, ob der Administrative Status korrekt überwacht wird.

Szenario 2: Störung und administrativ deaktiviert

Wir gehen wie gehabt von einem voll funktionsfähigem Netzwerk aus. Alle Ports sind aktiv und funktionsbereit. Es werden nun wiederum zwei Ports manuell deaktiviert8. Der Status sieht nun wie folgt aus.

IP Interfaces | **Physical Interfaces**

I...	SNMP ifDescr	SNMP ifName	SNMP ifAlias	SNMP ifSpeed	IP Address
1	VLAN1	VL1		10000000	172.16.23.9
2	FastEthernet0/1	Fa0/1	Uplink ISP Router	100000000	0.0.0.0
3	FastEthernet0/2	Fa0/2	Uplink sw_access_04	100000000	0.0.0.0
4	FastEthernet0/3	Fa0/3	Uplink sw_access_02	100000000	0.0.0.0
5	FastEthernet0/4	Fa0/4	Workstation	100000000	0.0.0.0
6	FastEthernet0/5	Fa0/5		100000000	0.0.0.0

Abbildung 4.1.10: Interface status nach administrativen deaktivieren des Ports.

Wir erhalten zwei unterschiedliche Meldungen und zwar über den Port *Fa0/1* sowie *Fa0/3*. Der Port *Fa0/4* für die Workstation wird komplett ignoriert. Für den administrativen Status wird ein eigenes Event *interfaceAdminDown* erzeugt.

Home / Events / **Detail**

Event 949

Severity	Minor	Node	sw_access_01	Acknowledged By	
Time	7/19/10 3:35:29 AM	Interface	172.16.23.9	Time Acknowledged	
Service	SNMP				
UEI	uei.opennms.org/nodes/snmp/interfaceAdminDown				

Log Message

Administration status Down on interface ifname:Fa0/3 ifindex:4 ifdescr:FastEthernet0/3

Description

The administration status of interface is down
This event is generated when an snmp poll on interface find the administration status down.
Params snmpifindex="4" ipaddr="0.0.0.0" snmpifname="Fa0/3" snmpifdescr="FastEthernet0/3" snmpifalias="Uplink sw_access_02"

Operator Instructions

No instructions available

Abbildung 4.1.11: *Event* nach dem administrativen deaktivieren des Ports.

Bei einer echten Störung wird ein Event *interfaceOperDown* generiert. Damit ist der Funktionstest abgeschlossen und auf die entsprechenden Events können verschiedene Benachrichtigungen eingerichtet werden. Für die Benachrichtigung stehen die folgenden Events zur Verfügung

- Snmp Interface Admin Status Down
- Snmp Interface Admin Status Up
- Snmp Interface Oper Status Down
- Snmp Interface Oper Status Up

Damit sollte einer Port Status Überwachung in OpenNMS nichts mehr im Wege stehen.

4.2 Monitoring mit Skripten

Für die Überwachung von speziellen Anwendungen oder Arbeitsabläufen, reichen bestehende standardisierte Managementagenten häufig nicht aus. Die notwendigen Statusinformationen können über Standard- oder Herstellerspezifische *SNMP MIB* nicht abgefragt werden. Um hier spezielle Anforderungen erfüllen zu können ist es möglich eigene Programme oder Skripte zur Überwachung einzubinden. Um ein möglichst praxisnahes und durchgängiges Beispiel liefern zu können, konfigurieren wir exemplarisch eine Festplattenüberwachung mit *S.M.A.R.T.* Um die Abläufe zu verdeutlichen richten wir einen Monitor ein, der den Zustand einer lokal installieren Festplatte überwacht.

Die Verwendung der eigenen Programme und Skripte kann auf unterschiedliche Art und Weise realisiert werden. Im ersten Teil wird beschrieben, wie der unter *Unix/Linux* weit verbreitete *Net-SNMP* Agent erweitert und in *OpenNMS* genutzt werden kann. Das Monitoring des Festplattenstatus wird in Varianten durchgeführt und soll den Ablauf und die Notwendigen Konfigurationsschritte verdeutlichen.

Nagios, als ein weit verbreitetes quelloffenes Monitoringsystem, stellt einen sehr großen Pool[9] von Überwachungsskripten und Programmen zur Verfügung. Diese Skripte können als *Plugins* auf einem Server mit einem *Nagios-Agenten*[10] ausgeführt werden. In diesem Dokument wird beschrieben wie *OpenNMS* die *NRPE* Agenten in der Netzwerküberwachung nutzen kann. In *OpenNMS* wird dazu ein spezieller *NRPE Monitor* bereitgestellt. Er veranlasst das ausführen der *Plugins* und stellt den Status entsprechend dar.

Da für die Verwaltung von *Unix/Linux* basierten Systemen häufig das SSH-Protokoll verwendet wird, kann auch dieses Protokoll für die Netzwerküberwachung verwendet werden. Es erlaubt neben der Bereitstellung eines entfernten Shellzugangs ebenfalls die Möglichkeit, Programme oder Skripte auf dem entfernten Server auszuführen. In *OpenNMS* kann dazu der *General Purpose Monitor* verwendet werden. Er erlaubt es Shellkommandos auszuführen und kann das Ergebnis des Kommandos in einem *Service Status* auswerten.

Das *HTTP* ist eines der wichtigsten Protokolle im Internet. Es wird neben der Auslieferung von Internetseiten auch für entfernte Funktionsaufrufe (*ReST* oder *SOAP*) genutzt. Zusätzlich lässt sich *HTTP* auch für Monitoring-Zwecke verwenden. Über *HTTP* können sowohl Statusabfragen oder auch Leistungsdaten für das Monitoring bereitgestellt werden. Die Übertragung kann zudem über *SSL* verschlüsselt erfolgen. Zusätzlich sind Webserver nahezu auf jedem System sehr einfach einzurichten und sind im Netzwerk oft gut erreichbar. Im letzten Abschnitt wird gezeigt wie man einen *Apache2-Server*[11] in Verbindung mit *CGI* für das Monitoring in *OpenNMS* einrichten kann. Der Fokus liegt hier auf dem Statusmonitoring. Auf eine *HTTP-Datacollection*[12] zur Aufzeichnung von Leistungsdaten wird hier nicht eingegangen.

[9] Nagios Plugins unter `http://www.monitoringexchange.org`
[10] Unter Linux/Unix *NRPE* unter Windows *NSClient++*
[11] Apache Project: `http://httpd.apache.org`
[12] OpenNMS HTTP-Datacollection im OpenNMS Wiki

4.2.1 Einrichtung von S.M.A.R.T.

Um die Beispiele einrichten und nachvollziehen zu können muss voerst sicher gestellt werden, dass die *S.M.A.R.T.-Tools* installiert und eingerichtet sind. Das folgende Beispiel wurde auf einem *Ubuntu 9.10 Server* durchgeführt. Die *S.M.A.R.T.-Tools* können mit dem folgenden Kommando installiert werden.

```
aptitude install smartmontools
```

Für die Plattenüberwachung muss zusätzlich noch ein Prozess gestartet werden, der die entsprechenden Informationen von den Festplatten ausliest.

```
vi /etc/default/smartmontools
```

Zunächst muss erlaubt werden, dass der *smartd* gestartet werden kann und welche Festplatten überwacht werden sollen. In unserem Beispiel werden die beiden Festplatten */dev/sda* und */dev/sdb* überwacht. Die Konfigurationsdatei sieht dann wie folgt aus:

```
1  # Defaults for smartmontools initscript (/etc/init.d/smartmontools)
2  # This is a POSIX shell fragment
3
4  # List of devices you want to explicitly enable S.M.A.R.T. for
5  # Not needed (and not recommended) if the device is monitored by smartd
6  enable_smart="/dev/sda /dev/sdb"
7
8  # uncomment to start smartd on system startup
9  start_smartd=yes
10
11 # uncomment to pass additional options to smartd on startup
12 smartd_opts="--interval=1800"
```

Listing 4.7: Konfiguration der *smartmontools*

Der Dienst kann anschließend mit dem Kommando

```
service smartmontools start
```

gestartet werden. Um zu testen ob die Festplatteninformationen korrekt ausgelesen werden, kann das folgende Kommando den aktuellen Status liefern.

```
smartctl -H /dev/sdb
```

```
1  smartctl version 5.38 [i686-pc-linux-gnu] Copyright (C) 2002-8 Bruce Allen
2  Home page is http://smartmontools.sourceforge.net/
3
4  === START OF READ SMART DATA SECTION ===
5  SMART overall-health self-assessment test result: PASSED
```

Listing 4.8: Ausgabe von *smartctl*

Wenn die folgende Ausgabe wie oben gezeigt aussieht, dann können wir mit dem Einrichten der Überwachung fortfahren.

In den nächsten Abschnitten wird ein Monitoring mit den eben installierten smartmontools beschrieben. Wir beginnen zunächst mit der Variante über *Net-SNMP*. Im Anschluss wird das gleiche Monitoring über *NRPE*, über *SSH* und auch über *HTTP* mittels *CGI* dargestellt.

4.2.2 Net-SNMP als Agenten

In *Unix/Linux* Umgebungen kommt häufig *Net-SNMP* als Agent zum Einsatz. Dieser stellt damit nicht nur den Zugriff für die in der *Standard MIB-II* definierten Managementobjekte bereit, sondern erlaubt es zusätzlich eigene Skripte per *SNMP* einzubinden. Diese Möglichkeit macht damit die Skripte und Programme nicht nur für *OpenNMS* zugänglich, sondern können auch in jeder anderen *SNMP-fähigen* Netzwerküberwachungsanwendung verwendet werden. In diesem Beispiel wird gezeigt wie ein Shellskript in *Net-SNMP* eingebunden wird. Die Ausgabe des Skriptes wird mit dem *OpenNMS* bereitgestellten *SNMP-Monitor* überwacht und als *Service-Status* dargestellt. In der folgenden Abbildung wird der Ablauf grob skizziert.

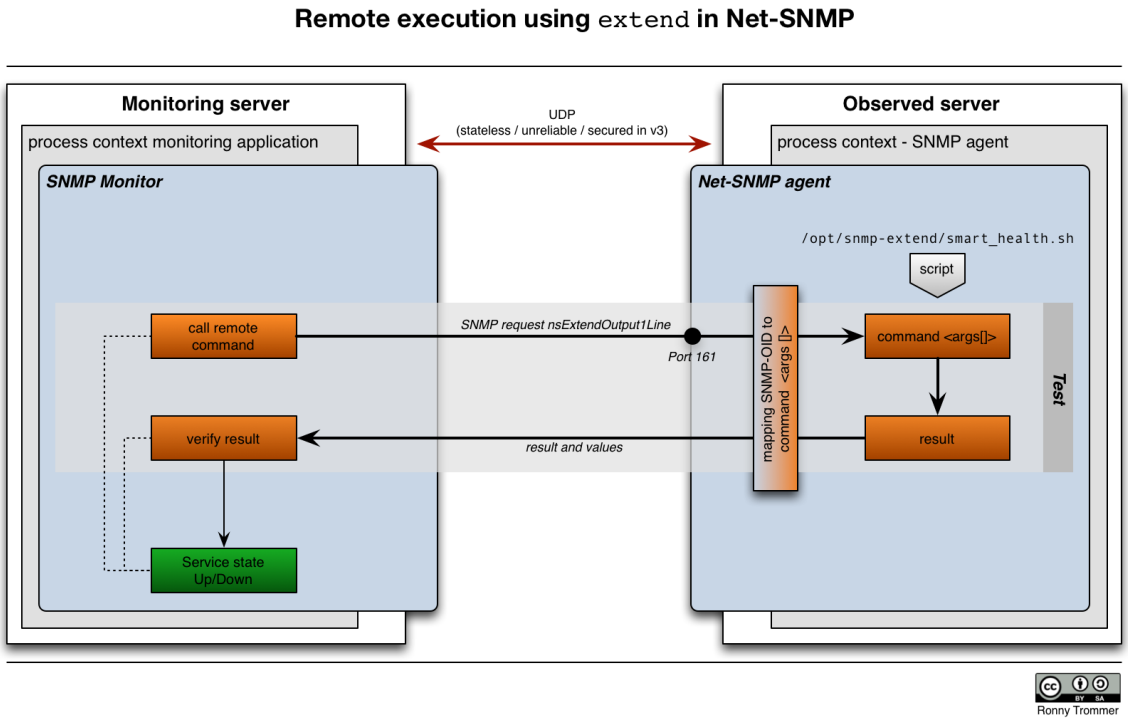

Abbildung 4.2.1: Ablauf von Abfragen mit der Erweiterung des *Net-SNMP* Agenten

Grundeinrichtung von Net-SNMP

Damit die Einrichtung vollständig nachvollzogen werden kann, wird die Grundeinrichtung von *Net-SNMP* kurz vorgestellt. In diesem Beispiel wird ebenfalls ein *Ubuntu Server 9.10* verwendet. Die Installation der *SNMP-Tools* und des *Net-SNMP* Agenten erfolgt mit

```
aptitude install snmp snmpd
```

Nach erfolgreicher Installation kann der Agent mit dem Kommando

```
service snmpd start
```

gestartet werden. Ob der Agent funktioniert kann mit den SNMP-Tools getestet werden. Eine *SNMP-Abfrage* über die installierte Kernelversion sieht dann wie folgt aus:

```
snmpget -v 2c -c public localhost .1.3.6.1.2.1.1.1.0
```

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux marge 2.6.31-22-generic-pae \
#60-Ubuntu SMP Thu May 27 01:40:15 UTC 2010 i686
```

Mit dieser Abfrage wird die Systembeschreibung per *SNMP* ausgelesen und liefert auf einem Unix/Linux-System die laufende Kernelversion. Aus Sicherheitsgründen ist die Standardkonfiguration des *SNMP-Agenten* sehr restriktiv konfiguriert. Der Agent lauscht nur auf der *IP-Loopback* Schnittstelle und die Abfrage der kompletten MIB ist nicht gestattet. Damit eine entfernte Abfrage über das Netzwerk überhaupt möglich ist, müssen daher einige Anpassungen vorgenommen werden.

Im ersten Schritt setzen wir eine andere *SNMP community*. Die *Standard-Community* ist auf *public* gesetzt.

Hinweis: Es macht Sinn eine unternehmensweite einheitliche *Community* festzulegen, die nicht *public* für den Lese- und *privat* für den Schreibzugriff ist.

Zusätzlich erlauben wir den Zugriff auf den *SNMP Agenten* nur der IP-Adresse des OpenNMS Servers. Die Angabe wird in *CIDR*[13]-Schreibweise angegeben. Dazu bearbeiten wird die Konfigurationsdatei mit

```
vi /etc/snmp/snmpd.conf
```

```
1 com2sec readonly  <opennms-server>/32     notpublic
2 .
3 group MyROGroup v2c         readonly
4 .
5 view all    included  .1                               80
6 .
7 access MyROGroup ""      any       noauth    exact  all    none   none
```

OpenNMS genügt ein lesender Zugriff. Damit im Monitoring auf die komplette *MIB* und nicht nur auf den Teilbaum system zugegriffen werden kann, ändern wir die *com2sec*[14] von *paranoid* auf *readonly*. Anstelle von *default* setzen wir die IP-Adresse des *OpenNMS* Servers mit einer 32bit Maske ein. Die *Community* ändern wir von public auf einen selbst definierten Wert, in diesem Beispiel einfach auf *notpublic*.

Hinweis: Wenn Sie einen Standardzugriff für andere Anwendung mit der Community public benötigen, dann kann auch eine zweite Zeile mit einer weiteren Community eingefügt werden. Sie sollten dann allerdings den Sicherheitsmodus paranoid bei public stehen lassen.

Der Zugriff auf den *SNMP-Agenten* wird von einem entfernten Rechner allerdings noch nicht funktionieren, da der *SNMP-Agent* standardmässig nur auf der *IP-Loopback* Schnittstelle lauscht. Damit *SNMP-Anfragen* von entfernten Rechnern beantwortet werden können, bearbeiten wir die folgende Datei mit

```
vi /etc/default/snmpd
```

[13]Nach / wird die Anzahl der Bits für eine Bitmaske angegeben
[14]com2sec beschreibt eine Zuordnung einer Community auf ein Sicherheitsmodell

und binden den Agenten auf alle IP-Adressen. Dazu ändern wird den Eintrag von *127.0.0.1* auf *0.0.0.0*. Bei oder bei *multihomed*[15] Servern kann auch eine IP-Adresse auf einem speziellen Netzwerkadapter gewählt werden. Die entsprechende Zeile sollte dann wie folgt aussehen:

```
1 # snmpd options (use syslog, close stdin/out/err).
2 SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid \
3 0.0.0.0'
```

Nach einem Neustart des SNMP-Agenten mit

```
service snmpd restart
```

können SNMP-Anfragen von entfernten Rechnern durchgeführt werden. Die Einstellung kann mit dem Kommando

```
netstat -lnpu
```

überprüft werden. Die Ausgabe sollte dann wie folgt aussehen:

```
udp        0      0 0.0.0.0:161             0.0.0.0:*               17391/snmpd
```

Ein Test von einem entfernten Rechner kann mit dem Kommando

```
snmpwalk -v 2c -c notpublic <snmp-agent-host>
```

durchgeführt werden. Im Folgenden wird die komplette *MIB*, mit allen entsprechenden Werten, auf der Konsole ausgegeben. Damit sind die Voraussetzungen geschaffen um mit der Einrichtung des Monitorings fortzufahren.

Ein Skript zur Ermittlung des SMART-Status

Nach erfolgreicher Installation der *smartmontools* schreiben wir ein kleines Wrapper-Skript, welches uns das entsprechende Ergebnis auswertet und auf der Konsole ausgibt. Die Ausgabe werden wir später über *SNMP* zugänglich machen. Das Skript sieht hier wie folgt aus:

```
 1 # !/bin/sh
 2 # Location: remote server in /opt/monitoring
 3 # File: check_smart_health.sh
 4 #
 5 # Check local hard disks with S.M.A.R.T.
 6
 7 # Test if device exist
 8 ls -l $1 1>&2 >/dev/null
 9 if [ $? -eq 0 ]; then
10   # Run smartctl check
11   RESULT=$(smartctl -n idle -H ${1} | grep result)
12   echo ${RESULT}
13 else
14   exit 1
15 fi
```

Listing 4.9: Wrapper-Skript für *smartctl*

[15] *multihomed* bedeutet, dass Server mehr als eine Netzwerkschnittstelle besitzen

Hinweis: Es ist hilfreich sich die Skripte einheitlich an einer zentralen Stelle abzulegen. Die Pflege und Wartung wird damit wesentlich erleichert.

Hier in diesem Beispiel werden die Skripte im Verzeichnis */opt/snmp-extend* gespeichert. Mit dem ersten Parameter geben wir an, von welcher Festplatte wir den Status prüfen möchten. Damit das Kommando ausgeführt werden kann, müssen die entsprechenden Rechte gesetzt werden:

```
chmod +x /opt/snmp-extend/check_smart_health.sh
```

Wir können nun das Skript testen und erhalten eine folgende Ausgabe, wenn wir die erste Festplatte prüfen:

```
cd /opt/snmp-extend
./smart_health /dev/sda
```

```
SMART overall-health self-assessment test result: PASSED
```

Da das Kommando *smartctl* lediglich als Benutzer *root* ausgeführt werden kann, der *SNMP Agent* allerdings mit dem Benutzer *snmp* gestartet wird, müssen wir uns hier entsprechende Rechte für den *SNMP-Benutzer* zuweisen. Eine mögliche Lösung wäre es in der */etc/default/snmpd* den Parameter *-u* von *snmp* auf *root* zu ändern. Das hätte dann allerdings zur Folge, dass alle Aktionen vom *SNMP Agenten* unter *root* Rechten ausgeführt werden. Wir gehen einen anderen Weg und verwenden hierzu sudo und erlauben lediglich das ausführen der Skripte unter */opt/snmp-extend* als Benutzer *root*.

Wir bearbeiten mit dem Kommando *visudo* die Datei */etc/sudoers* und fügen die folgende Zeile hinzu:

```
# Allow snmp daemon to execute extended commands as root
snmp    ALL=NOPASSWD: /opt/snmp-extend/*.sh
```

Damit ist sichergestellt, dass der Benutzer *snmp* nur Dateien mit dem Suffix *.sh* im Verzeichnis */opt/snmp-extend* mit *sudo* starten kann. Für die Ausführung ist keine Kennworteingabe notwendig. Im nächsten Schritt erweitern wir *Net-SNMP Agenten* mit den entsprechenden Skripten.

Erweiterung von Net-SNMP

Der Agent *Net-SNMP* bietet uns selbst wiederum verschiedene Möglichkeiten an, damit ein Skript oder Programm über eine *OID* angesprochen werden kann. Wir verwenden die Anweisung *extend* und bearbeiten dazu die Datei *snmpd.conf*.

```
vi /etc/snmpd.conf
```

um sie um die folgenden Einträge zu erweitern. Da zwei Festplatten im Server vorhanden sind, wird das Skript entsprechend mit den beiden *Blockdevices* als Parameter aufgerufen.

```
1 ##############################################################################
2 # OpenNMS example - Extending Net-SNMP with user defined scripts
3 #
4 # extend [MIBOID] NAME PROG ARGS
```

```
5
6  extend smart_health_sda '/usr/bin/sudo /opt/snmp-extend/smart_health.sh
       /dev/sda'
7
8  extend smart_health_sdb '/usr/bin/sudo /opt/snmp-extend/smart_health.sh
       /dev/sdb'
```

Listing 4.10: Erweiterung *Net-SNMP* mit dem erstellten Wrapper-Skript für *smartctl*

Damit die Änderungen wirksam werden, muss der *SNMP-Agent* mit

```
service snmpd restart
```

neu gestartet werden. Die entsprechenden Skripte können jetzt mit *SNMP* abgerufen werden.

Hinweis: Unter *Debian* können benutzerdefinierte Einstellungen auch in der Datei */etc/snmp/snmpd.local.conf* gesetzt werden. Damit sind alle rechnerspezifischen Einträge in einer speraten Datei ausgelagert und Roll-Outs sowie Updates sind leichter administrierbar.

```
snmpwalk -v 2c -c notpublic <snmp-agent-host> nsExtendOutput1Line
```

```
NET-SNMP-EXTEND-MIB::nsExtendOutput1Line."smart_health_sda" = STRING: SMART \
overall-health self-assessment test result: PASSED

NET-SNMP-EXTEND-MIB::nsExtendOutput1Line."smart_health_sdb" = STRING: SMART \
overall-health self-assessment test result: PASSED
```

Das Skript wird ausgeführt und die Ausgabe unter den oben gezeigten OIDs per SNMP bereitgestellt. Um den entsprechenden SNMP-Monitor in OpenNMS einzurichten, benötigen wir die vollständige OID zu den entsprechenden Ausgaben. Diese lässt sich mit dem Kommando

```
snmpwalk -On -v 2c -c notpublic <snmp-agent-host> nsExtendOutput1Line
```

anzeigen. Die entsprechenden *OIDs* sehen dann wie folgt aus:

```
.1.3.6.1.4.1.8072.1.3.2.3.1.1.16.115.109.97.114.116.95.104.101.97.108.116. \
104.95.115.100.97

.1.3.6.1.4.1.8072.1.3.2.3.1.1.16.115.109.97.114.116.95.104.101.97.108.116. \
104.95.115.100.98
```

Die *OIDs* werden dabei wie folgt aufgebaut. Der erste Teil der *SNMP-OID* adressiert alle eingebundenen Skripte.

```
.1.3.6.1.4.1.8072.1.3.2.3.1.1.16
```

der zweite Teil der *SNMP-OID* wird der "Name" des Skriptes in *Hex-Dezimal*[16] Zahlen angegeben. s=115, m=109, a=97 usw. Aus *smart_health_sda* wird dann

```
115.109.97.114.116.95.104.101.97.108.116.104.95.115.100.97
```

Es können so Skripte als Net-SNMP Erweiterungen eindeutig adressiert werden. Das Bedeutet im Umkehrschluss, dass jeder Name in der Anweisung extend nur einmal verwendet werden kann und eindeutig sein muss.

[16] JavaScript ASCII Bin-Hex-Dec Converter: `http://mediusrete.org/diversions/ascii-conv.html`

Einrichtung SNMP-Monitor in OpenNMS

OpenNMS hat verschiedene Möglichkeiten Monitore bereitzustellen, welcher über die Methode *Discovery* und dem *Capsd*[17] sowie über *Provisioning Groups*[18] angelegt werden können. Wir verwenden *Capsd*, um den Dienst auf allen Servern automatisch zu erkennen. In der Datei wird ein entsprechendes Protokoll wie folgt angelegt:

```
vi $OPENNMS_HOME/etc/capsd-configuration.xml
```

Das Protokoll wird in *XML-Syntax* wie folgt angelegt:

```
<!-- START - Customized monitoring -->
<protocol-plugin protocol="SMART-Health-sda"
                 class-name="org.opennms.netmgt.capsd.plugins.SnmpPlugin"
                 scan="on">
    <property key="vbname" value=".1.3.6.1.4.1.8072.1.3.2.3.1.1.16.115. \
                                 109.97.114.116.95.104.101.97.108.116. \
                                 104.95.115.100.97" />
    <property key="timeout" value="3000" />
    <property key="retry" value="1" />
</protocol-plugin>
<protocol-plugin protocol="SMART-Health-sdb"
                 class-name="org.opennms.netmgt.capsd.plugins.SnmpPlugin"
                 scan="on">
    <property key="vbname" value=".1.3.6.1.4.1.8072.1.3.2.3.1.1.16.115. \
                                 109.97.114.116.95.104.101.97.108.116. \
                                 104.95.115.100.98" />
    <property key="timeout" value="3000" />
    <property key="retry" value="1" />
</protocol-plugin>
<!-- END - Customized monitoring -->
```

Listing 4.11: Konfiguration von *capsd* zum automatischen erkennen des Service für *sda* und *sdb*

Hinweis: Bitte fügen Sie eigene Monitore am Ende der Datei an und markieren Sie den Bereich wo eigene Monitore angelegt wurden. Bei einem Update wird es damit wesentlich einfacher die entsprechende Konfiguration zu überführen.

Wichtig: In der Darstellung sind Zeilenumbrüche mit gekennzeichnet. In der Konfigurationsdatei von *OpenNMS* sollten die entsprechenden Angaben in einer Zeile angegeben werden.

Auf allen Servern bei denen sich diese *SNMP-OID* abfragen lässt bindet, *OpenNMS* den Monitor auf die entsprechende IP-Schnittstelle des Knotens und überacht dessen Status automatisch. Im nächsten Schritt muss ein entsprechender Monitor eingerichtet werden, welcher den entsprechenden Status überprüft. Dazu wird die Konfigurationsdatei für den *Pollerd*[19] wie folgt bearbeitet:

```
vi $OPENNMS_HOME/etc/poller-configuration.xml
```

```
<!-- START - Customized monitoring -->

```

[17] Capabilityscan Daemon - Dienst zur automatischen Erkennung von Monitoren und Protokollen
[18] Provisioning Groups sind konkret definierte Netzwerkgeräte die im Monitoring bereitgestellt werden
[19] Pollerd testet ob der SMART Status ok ist oder nicht.

```
3  <!-- SMART service configuration for /dev/sda and /dev/sdb -->
4  <service name="SMART-Health-sda"
5           interval="300000"
6           user-defined="false"
7           status="on">
8      <parameter key="retry" value="2"/>
9      <parameter key="timeout" value="5000"/>
10     <parameter key="port" value="161"/>
11     <parameter key="oid" value=".1.3.6.1.4.1.8072.1.3.2.4.1.2.16.115. \
12                                109.97.114.116.95.104.101.97.108.116. \
13                                104.95.115.100.97.1"/>
14     <parameter key="vbvalue" value="SMART overall-health self-assessment
           test result: PASSED"/>
15 </service>
16 <service name="SMART-Health-sdb"
17          interval="300000"
18          user-defined="false"
19          status="on">
20     <parameter key="retry" value="2"/>
21     <parameter key="timeout" value="5000"/>
22     <parameter key="port" value="161"/>
23     <parameter key="oid" value=".1.3.6.1.4.1.8072.1.3.2.4.1.2.16.115. \
24                                109.97.114.116.95.104.101.97.108.116. \
25                                104.95.115.100.98.1"/>
26     <parameter key="vbvalue" value="SMART overall-health self-assessment
           test result: PASSED"/>
27 </service>
28 <!-- END - Customized monitoring -->
29
30 <!-- START - Customized monitoring -->
31
32 <!-- Mapping for SMART-Health-sda and SMART-Health-sdb using the SNMP
       Monitor -->
33 <monitor service="SMART-Health-sda"
       class-name="org.opennms.netmgt.poller.monitors.SnmpMonitor"/>
34 <monitor service="SMART-Health-sdb"
       class-name="org.opennms.netmgt.poller.monitors.SnmpMonitor"/>
35 <!-- END - Customized monitoring -->
```

Listing 4.12: Konfiguration von *pollerd* zum testen des Service für *sda* und *sdb*

Die beiden Monitore können im Bereich *polling-package* "example1" angegeben werden. Es ist hier ebenfalls sinnvoll die eigenen Monitore in der Konfiguration zu markieren und am Ende anzufügen. Am Ende der Datei muss noch festgelegt werden von welchem Typ der Monitor ist. In unserem Fall sind beide vom Typ *SnmpMonitor*.

Damit die Einstellungen übernommen werden, sollte zunächst die Konfiguration mit *xmllint*[20]

```
xmllint $OPENNMS_HOME/etc/capsd-configuration.xml
xmllint $OPENNMS_HOME/etc/poller-configuration.xml
```

getestet werden. Nach erfolgreichem Test muss OpenNMS mit dem Kommando

```
service opennms restart
```

[20]Unter *Ubuntu* kann *xmllint* über das Kommando *aptitude install libxml2-utils* installiert werden.

neu gestartet werden. Mit einem *Rescan* auf einem bestehendem Knoten[21]. wird geprüft ob die entsprechenden Skripte erfolgreich aufgerufen werden können und anschließend von *OpenNMS* überwacht.

Wichtig: Beachten Sie die eingestellten *Timeouts* im *SNMP-Monitor*. Die Laufzeit der lokalen Skripte sollte diese *Timeouts* nicht überschreiten. Der *SNMP-Agent* verwendet intern einen *Cache* und speichert im Standard die letzten Ergebnisse für 5 Sekunden.

Hinweis: Den *Cache* des *SNMP-Agenten* kann man für jedes Skript seperat konfigurieren. Mit dem Kommando

```
snmpset -v2c -c notpublic <snmp-agent-host>
    'NET-SNMP-EXTEND-MIB::nsExtendCacheTime."smart_temp_sda"' i 15
```

```
NET-SNMP-EXTEND-MIB::nsExtendCacheTime."smart_temp_sda" = INTEGER: 15
```

wird die Zeit für den *Cache* für den Eintrag *smart_ temp_ sda* auf 15 Sekunden gesetzt.

Mittels dieser Methode lassen sich ebenfalls Leistungsdaten mit dem *Collectd*[22] zur Langzeitarchivierung und Trendanalyse durchführen. Die entsprechende Konfigurationen sind im *OpenNMS Wiki*[23] dokumentiert. Im nächsten Abschnitt wird die Überwachung mit dem Nagios Agenten NRPE dargestellt.

4.2.3 NRPE als Agent

Nagios ist ein sehr weit verbreitetes Netzwerkmonitoring-Tool. Für *Nagios* selbst sind zahlreiche *Plugins*[24] vorhanden. Um die entsprechenden *Plugins* auf entfernten Systemen auszuführen, stellt *Nagios* einen eigenen Agenten *NRPE* bereit. *Nagios* kann über ein IP-Netzwerk ein *Plugin* entfernt ausführen und bekommt das entsprechende Ergebnis übermittelt. Mit *OpenNMS* können ebenfalls *Nagios Plugins* über den *NRPE-Agent* abgefragt werden. Im folgenden wird gezeigt wie ein *Plugin* von *Nagios* zur Ermittlung des *SMART-Status* mit *OpenNMS* und dem *NRPE-Monitor* getestet werden können.

[21] Als Knoten wird ein Server, Router oder anderes Netzwerkkgerät in *OpenNMS* bezeichnet

[22] Collectd wird der Collection Daemon genannt der Leistungsdaten für Trendanalyse in *RRD-Dateien* aufzeichnet.

[23] Beschreibung Datacollection `http://www.opennms.org/index.php/Data\_Collection\_Configuration_How-To`

[24] Eine Plugin Datenbank wird unter `http://www.monitoringexchange.org` angeboten

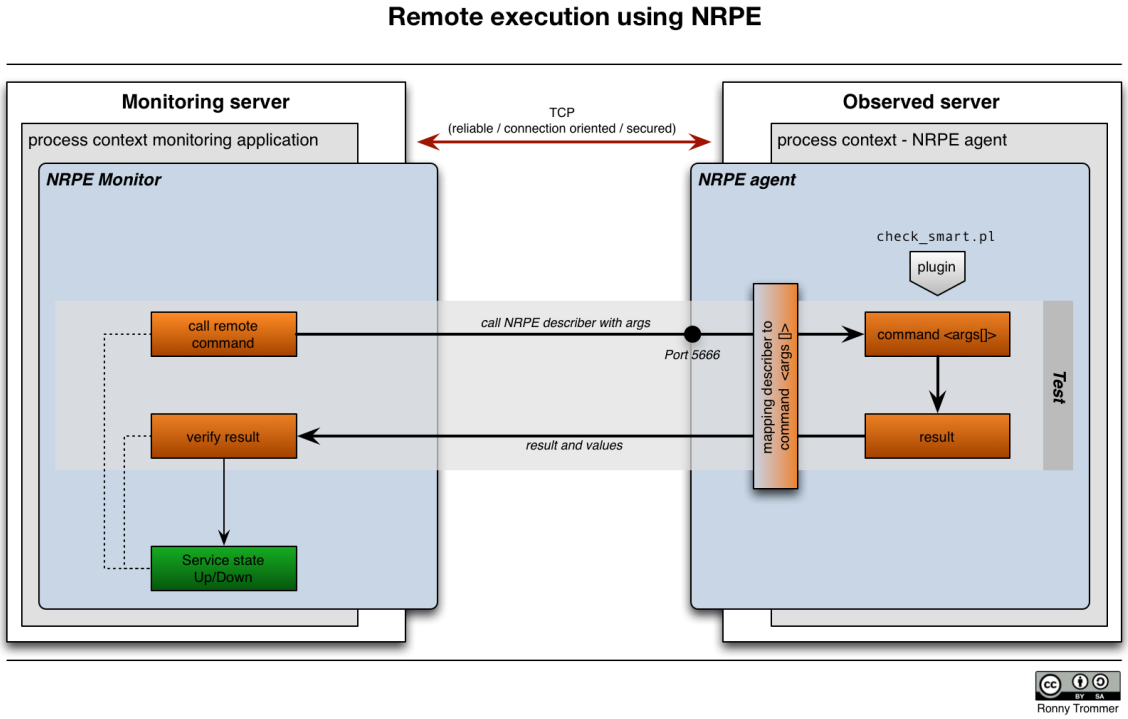

Abbildung 4.2.2: Ablauf von Abfragen über den *NRPE* Agenten

Grundeinrichtung von NRPE

Bevor wir mit der Einrichtung des *Plugins* beginnen, wird gezeigt wie der *NRPE* eingerichtet werden muss. Die Installation des *NRPE-Agenten* auf einem *Ubuntu 9.10 Server* kann mit dem Kommando

```
aptitude install nagios-nrpe-server
```

erfolgen. Aus Sicherheitsgründen ist der *NRPE-Agent* sehr restriktiv konfiguriert. Es werden lediglich Anfragen von *127.0.0.1* erlaubt. Damit der *OpenNMS Server* auf die *Plugins* zugreifen kann, wird die IP-Adresse in *allowed_hosts* eingetragen.

```
vi /etc/nagios/nrpe.cfg
```

```
allowed_hosts=127.0.0.1,<opennms-host>
```

Im nächsten Schritt muss ein entsprechendes *Plugin* eingerichtet und über *NRPE* zugänglich gemacht werden. Für den *NRPE-Agenten* können zusätzliche Startparameter in der Datei *nagios-nrpe-server* in */etc/default* bearbeitet werden. Um beispielsweise *SSL* zu deaktivieren, kann die Option *–nossl* angegeben werden:

```
vi /etc/default/nagios-nrpe-server
```

```
DAEMON_OPTS="--no-ssl"
```

Ein Nagios-Plugin für den SMART Status

Für diese Beispiel verwenden wir das Plugin *check_ smart*[25]. Damit das *Plugin* ausgeführt werden kann, ist ein *Perl Interpreter* notwendig. Zusätzlich muss die *Nagios* eigene *Perl* Bibliothek *utils.pm*[26] installiert sein. Das *Plugin* kopieren wir in das Verzeichnis */usr/lib/nagios/plugins*. Die *Plugins* müssen zunächst ausführbar gemacht werden:

```
chmod +x /usr/lib/nagios/plugins/check_smart.pl
```

Die Funktion des Plugins kann mit dem Kommando

```
check_smart.pl -d /dev/sda -i scsi
```

getestet werden. Die Ausgabe sollt wie unten gezeigt aussehen.

```
OK: no SMART errors detected
```

Das Plugin muss anschließend im *NRPE-Agenten* eingetragen werden. Dazu wird die Datei *nrpe_ local.cfg* wie folgt erweitert:

```
vi /etc/nagios/nrpe_local.cfg
```

```
command[check_smart_sda]=/usr/lib/nagios/plugins/check_smart.pl -d /dev/sda
    - scsi

command[check_smart_sdb]=/usr/lib/nagios/plugins/check_smart.pl -d /dev/sdb
    - scsi
```

Damit das neu eingerichtete Plugin genutzt werden kann, muss der NRPE-Agent mit dem Kommando

```
service nagios-nrpe-server restart
```

neu gestartet werden. Mit einem entsprechendem *check_ nrpe*[27] Plugin kann der Dienst jetzt wie folgt getestet werden:

```
check_nrpe -H <nrpe-host> -c check_smart_sda
```

```
OK: no SMART errors detected
```

Bei der Auswertung des Ergebnisses ist allerdings nicht die Ausgabe sondern der *exit code* relevant. *Nagios* kennt nicht nur *UP* oder *DOWN*. Die exit codes sind dabei wie folgt zugeordnet:

[25] http://www.monitoringexchange.org/inventory/Check-Plugins/Hardware/Storage/Check-SMART-status
[26] Die Bibliothek *utils.pm* wird mit den *Nagios Plugins* ausgeliefert
[27] NRPE: http://www.nagios-wiki.de/nagios/howtos/nrpe

Exit code	NRPE-Status	OpenNMS-Status
0	Ok	Up
1	Warning	Down
2	Critical	Down
3	Unknown	Down
4	Dependent	Down

Tabelle 4.2.1: Zuweisung von NRPE zu OpenNMS Status codes

Mit den oben gezeigten Einstellungen kann nun der *OpenNMS Monitor* eingerichtet werden. Da auch der *NRPE-Agent* unter einem restriktivem Benutzer läuft, muss für Kommandos die *root-Rechte* benötigen ein entsprechender *sudo-Eintrag* vorgenommen werden.

```
visudo
```

```
nagios  ALL=NOPASSWD: /usr/lib/nagios/plugins/check*
```

Mit diesem Eintrag wird dem Benutzer *nagios* erlaubt alle Plugins die mit *check* beginnen über *sudo* auszuführen.

Einrichtung NRPE-Monitor in OpenNMS

Damit *OpenNMS* automatisch erkennen kann, wo das *SMART-Status Plugin* installiert ist, wird in der *Capsd* wie folgt eingerichtet.

```
vi $OPENNMS_HOME/etc/capsd-configuration.xml
```

```
<!-- START - Customized monitoring -->
<protocol-plugin protocol="NRPES-SMART-Health-sda"
                 class-name="org.opennms.netmgt.capsd.plugins.NrpePlugin"
                 scan="on">
    <property key="banner" value="*" />
    <property key="port" value="5666" />
    <property key="timeout" value="3000" />
    <property key="retry" value="2" />
    <property key="command" value="check_smart_sda" />
</protocol-plugin>

<protocol-plugin protocol="NRPES-SMART-Health-sdb"
                 class-name="org.opennms.netmgt.capsd.plugins.NrpePlugin"
                 scan="on">
    <property key="banner" value="*" />
    <property key="port" value="5666" />
    <property key="timeout" value="3000" />
    <property key="retry" value="2" />
    <property key="command" value="check_smart_sdb" />
</protocol-plugin>
<!-- END - Customized monitoring -->
```

Listing 4.13: Konfiguration von *capsd* zum automatischen erkennen des Service für *sda* und *sdb* mit dem *NRPE-Plugin*

Hinweis: Der *NRPE-Agent* verschlüsselt die Übertragung standardmässig mit *SSL*. Werden sowohl verschlüsselte als auch unverschlüsselte *NRPE-Agenten* eingesetzt, ist es sinnvoll die entsprechenden Monitore mit *NRPE* oder *NRPES* als Protokollname zu kennzeichnen. Bei unverschlüsselten *NRPE-Agenten* muss zusätzlich die Eigenschaft

```
<property key="usessl" value="false" />
```

gesetzt werden. Fehlt die Angabe wird von einer *SSL*-verschlüsselten Kommunikation ausgegangen.

Um ein *NRPE Plugin* vom *OpenNMS Server* testen zu können, kann über die Kommandozeile das *CheckNrpe Plugin* direkt ausgeführt werden. Leider kann über die Kommandozeile *CheckNrpe* nur *NRPE-Agenten* ansprechen die kein *SSL* verwenden. Für *Debugging-Zwecke* sollte das allerdings ausreichen. Es kann somit geprüft werden ob die Kommunikation von OpenNMS zum NRPE-Agenten funktioniert und die entspechenden Plugins richtig eingerichtet sind. Achten Sie auf die Versionsnummer des Java-Archives!

```
cd $OPENNMS_HOME/lib
java -cp opennms-services-1.8.0.jar org.opennms.netmgt.poller.nrpe.CheckNrpe
    -t 1 -H <nrpe-host> -c check_smart_sda
```

Damit die beiden Plugins überwacht werden muss noch ein entsprechender Monitor eingerichtet werden. Dazu wird die Konfiguration für den Pollerd wie folgt bearbeitet:

```
vi $OPENNMS_HOME/etc/poller-configuration.xml
```

```
<!-- START - Customized monitoring -->
<service name="NRPES-SMART-Health-sda"
         interval="300000"
         user-defined="false"
         status="on">
    <parameter key="retry" value="3"/>
    <parameter key="timeout" value="5000"/>
    <parameter key="port" value="5666"/>
    <parameter key="command" value="check_smart_sda"/>
    <parameter key="padding" value="2"/>
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
    <parameter key="ds-name" value="nrpe"/>
</service>

<service name="NRPES-SMART-Health-sda"
         interval="300000"
         user-defined="false"
         status="on">
    <parameter key="retry" value="3"/>
    <parameter key="timeout" value="5000"/>
    <parameter key="port" value="5666"/>
    <parameter key="command" value="check_smart_sda"/>
    <parameter key="padding" value="2"/>
    <parameter key="rrd-repository" value="/opt/opennms/share/rrd/response"/>
    <parameter key="ds-name" value="nrpe"/>
</service>
<!-- END - Customized monitoring -->

<!-- START - Customized monitoring -->
<monitor service="NRPES-SMART-Health-sda"
         class-name="org.opennms.netmgt.poller.monitors.NrpeMonitor"/>
```

```
32  <monitor service="NRPES-SMART-Health-sdb"
33           class-name="org.opennms.netmgt.poller.monitors.NrpeMonitor"/>
34  <!-- END - Customized monitoring -->
```

Listing 4.14: Konfiguration von *pollerd* um den Service Status über das *NRPE-Plugin* zu testen

Nach einem Neustart von OpenNMS mit dem Kommando

```
service opennms restart
```

werden die entsprechenden *Plugins* überwacht.

4.2.4 SSH als Agent

Viele *Linux* oder *Unix-Server* werden über das Netzwerk mit *SSH* administriert. Mit *SSH* können Programme über eine verschlüsselte Netzwerkverbindung ausgeführt werden. Neben der reinen Verwaltung kann *SSH* auch für Monitoringaufgaben verwendet werden. In diesem Beispiel wird gezeigt wie mit *OpenNMS* und einem *General Purpose Monitor* der *SMART-Festplattenstatus* überwacht werden kann. Zunächst muss allerdings sicher gestellt werden, dass der *OpenNMS Server* Kommandos ohne Kennworteingabe auf dem entfernten Server per *SSH* ausführen kann. Im Anschluss wird ein entsprechends Skript erzeugt und das Monitoring in *OpenNMS* eingerichtet. Die Vorgehensweise ist dem *Nagios-Plugin check_by_ssh* sehr ähnlich.

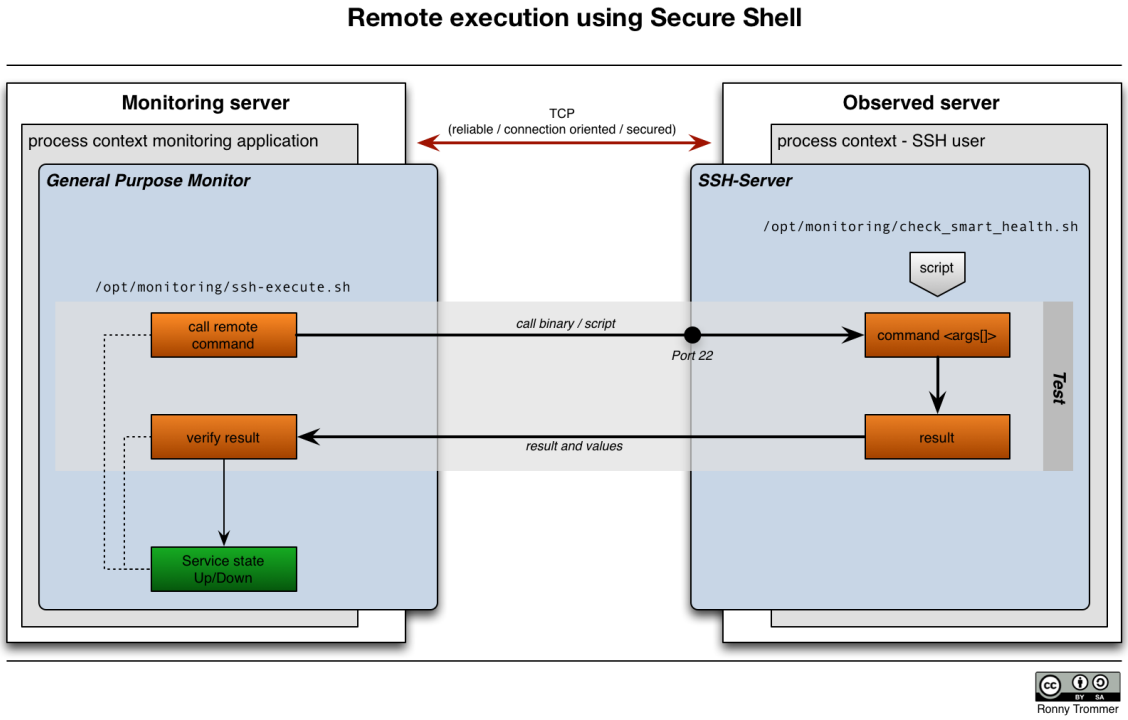

Abbildung 4.2.3: Ablauf von Abfragen über *SSH*

Grundeinrichtung von SSH

Zur Ausführung von Kommandos über SSH ist eine Authentifizierung notwendig. Mit SSH kann eine Authentifizierung mittels Private- und Public Key und starker Verschlüsselung realisiert werden. Damit kein direkter root-Zugang besteht, wird auf dem entfernten Rechner ein Benutzer angelegt. Im gezeigten Beispiel ist auf dem lokalen sowie entfernten Server Ubuntu 9.10 installiert. Wir legen einen Benutzer monitoring an, der für diese Aufgaben entsprechend verwendet werden soll.

```
adduser monitoring
```

Für den Benutzer wird ein sicheres Kennwort eingerichtet. Zusätzlich erlauben wir dem Benutzer monitoring, das Ausführen der check-Skripte mit root-Rechten.

```
visudo
```

```
monitoring  ALL=NOPASSWD: /opt/monitoring/check*
```

Die entsprechenden Skripte zur Überwachung werden in einem zentralen Verzeichnis */opt/-monitoring* gespeichert und von dort ausgeführt. Im nächsten Schritt wird ein privater und öffentlicher Schlüssel auf dem *OpenNMS Server* generiert.

```
cd ~
ssh-keygen -t dsa
```

```
1 Generating public/private dsa key pair.
2 Enter file in which to save the key (/root/.ssh/id_dsa):
3 Enter passphrase (empty for no passphrase):
4 Enter same passphrase again:
5 Your identification has been saved in id_dsa.
6 Your public key has been saved in ida_dsa.pub.
```

Das Schlüsselpaar bestehend aus privatem und öffentlichem Schlüssel und werden im Homeverzeichnis */root* gespeichert.

Wichtig: Es darf keine *Passphrase* eingegeben werden. Lediglich das Schlüsselpaar wird zur Authentifzierung verwendet.

Im Anschluss wird der öffentliche Schlüssel auf den entfernten Rechner kopiert. Der öffentliche Schlüssel muss auf dem entfernten Server im Verzeichnis *Home-Verzeichnis* unter dem Verzeichnis *.ssh* gespeichert werden. Wir befinden uns auf dem *OpenNMS Server* und melden uns auf dem entfernten Rechner an. Als nächstes erstellen wir die entsprechenden Verzeichnisse und kopieren den öffentlichen Schlüssel auf den Server. Im Anschluss werden die Dateirechte entsprechend gesetzt.

```
# 1. .ssh Verzeichnis auf dem entfernten Server anlegen
ssh monitoring@<entfernter-server>
mkdir .ssh
exit
```

```
# 2. OpenNMS Server Verzeichnis /root
scp .ssh/id_dsa.pub monitoring@<entfernter-server>:/home/monitoring

ssh monitoring@<entfernter-server>
```

```
# 3. Public key fuer die Authentikation anlegen und Rechte setzen
cat id_dsa.pub >> .ssh/authorized_keys2
chmod 400 .ssh/authorized_keys2
chmod 500 .ssh
rm id_dsa.pub
exit
```

Nun kann eine Shell-Sitzung ohne Authentifizierung, von root auf den Benutzer monitoring auf dem entfernten Server geöffnet, werden. Um die Ausführung zu testen kann man sich nun vom OpenNMS Server das Verzeichnis /var/log auf dem entfernten Server anzeigen lassen:

```
ssh monitoring@<entfernter-server> ls /var/log
```

Ein Skript zur entfernten Ausführung

Um die Ergebnisse der Kommandos für den *OpenNMS Monitor* aufzubereiten wird ein *Wrapper-Skript* auf dem *OpenNMS Server* erstellt welches die entsprechenden Ausgaben für den Monitor aufbereitet. Die Skripte sollten zentral in einem Verzeichnis gespeichert werden. In dem gezeigten Beispiel werden alle Skripte in */opt/monitoring* gespeichert. Wir verwenden das gleiche Skript *check_smart_health.sh* wie bei der *Net-SNMP* Variante.

```
1  # !/bin/bash
2  # Location: OpenNMS Server /opt/monitoring
3  # Wrapper to execute remote commands over SSH
4  #
5  # The seconde argument $2 will be ignored. Capsd and Pollerd give
6  # arg1[skriptname], arg2[device], arg3[--hostname], arg4[ip-address]
7  #
8  IP=${4}
9  SCRIPT_BASE=/opt/monitoring
10 SSH_USER=monitoring
11 SCRIPT_PARM=${2}
12 SCRIPT=${1}
13 SUDO=/usr/bin/sudo
14
15 CMD="ssh ${SSH_USER}@${IP} ${SUDO} ${SCRIPT_BASE}/${SCRIPT} ${SCRIPT_PARM}"
16 RESULT=`${CMD}`
17 if [ ! $? -eq 0 ]; then
18   echo ${RESULT}
19   exit 0
20 else
21   echo "Error during executing ${CMD}"
22   exit 1
23 fi
```

Listing 4.15: Wrapper Skript für remote auszuführende *SSH* Kommandos

Um zu testen ob das Kommando über *SSH* ausgeführt wird, kann der folgende Aufruf verwendet werden:

```
ssh monitoring@<entfernter-server> /usr/bin/sudo
    "/opt/monitoring/check_smart_health.sh /dev/sda"
```

Alle entfernten Kommandos werden durch das Skript *ssh-execute.sh* gestartet. Dem Skript wird zusätzlich noch ein Parameter für das zu testende *Blockdevice* übergeben. *Capsd* und

Pollerd übergeben automatisch die gerade zu prüfende IP-Adresse als 4. Parameter. Die IP-Adresse wird später von *OpenNMS* dynamisch für die *SSH-Verbindung* zum Test verwendet. Im nächsten Schritt wird der entsprechende Monitor eingerichtet.

Einrichtung General Purpose Monitor in OpenNMS

Damit *OpenNMS* automatisch erkennt auf welchem Server *SMART* mit den entsprechenden Überwachungsskripten installiert ist, richten zunächst ein Protokoll für den *Capsd* ein.

```
<!-- START - Customized monitoring -->
<protocol-plugin protocol="SSH-SMART-Health-sda"
                 class-name="org.opennms.netmgt.capsd.plugins.GpPlugin"
                 scan="on">
    <property key="script" \
     value="/opt/monitoring/ssh-execute.sh check_smart_health.sh /dev/sda" />
    <property key="banner" value="*" />
    <property key="timeout" value="3000" />
    <property key="retry" value="1" />
</protocol-plugin>

<protocol-plugin protocol="SSH-SMART-Health-sdb"
                 class-name="org.opennms.netmgt.capsd.plugins.GpPlugin"
                 scan="on">
    <property key="script" \
     value="/opt/monitoring/ssh-execute.sh check_smart_health.sh /dev/sdb" />
    <property key="banner" value="*" />
    <property key="timeout" value="3000" />
    <property key="retry" value="1" />
</protocol-plugin>
<!-- END - Customized monitoring -->
```

Listing 4.16: Konfiguration von *capsd* zum automatischen erkennen des Service für *sda* und *sdb* mit *SSH*

Im nächsten Schritt wird ein Monitor im *Pollerd* eingerichtet. Hier wird über einen regulären Ausdruck geprüft ob tatsächlich der Status *OK* ist und entsprechend in *OpenNMS* gesetzt.

```
<!-- START - Customized monitoring -->
<service name="SSH-SMART-Health-sda"
         interval="300000"
         user-defined="false"
         status="on">
    <parameter key="script" value="/opt/monitoring/ssh-execute.sh
        check_smart_health.sh /dev/sda"/>
    <parameter key="banner" value="OK"/>
    <parameter key="retry" value="2"/>
    <parameter key="timeout" value="5000"/>
    <parameter key="port" value="161"/>
</service>
<service name="SSH-SMART-Health-sdb"
         interval="300000"
         user-defined="false"
         status="on">
    <parameter key="script" value="/opt/monitoring/ssh-execute.sh
        check_smart_health.sh /dev/sda"/>
    <parameter key="banner" value="OK"/>
    <parameter key="retry" value="2"/>
```

```
19      <parameter key="timeout" value="5000"/>
20      <parameter key="port" value="161"/>
21  </service>
22  <!-- END - Customized monitoring -->
23
24  <!-- START - Customized monitoring -->
25  <monitor service="SSH-SMART-Health-sda"
        class-name="org.opennms.netmgt.poller.monitors.GpMonitor"/>
26  <monitor service="SSH-SMART-Health-sdb"
        class-name="org.opennms.netmgt.poller.monitors.GpMonitor"/>
27  <!-- END - Customized monitoring -->
```

Listing 4.17: Konfiguration von *pollerd* zum testen des Service für *sda* und *sdb* mit *SSH*

Nach einem Neustart von *OpenNMS* mit dem Kommando

```
service opennms restart
```

wird der Status des *SMART-Status* überwacht. Auf jedem Server auf dem das Skript vorhanden und remote ausführbar ist, wird automatisch erkannt und in *OpenNMS* automatisch überwacht.

Wichtig: Die verschlüsselte Kommunikation über *TCP* sowie das Ausführen externer Programme erzeugt zusätzlich schwergewichtige Prozesse mittels *fork()*. Wenn Skalierbarkeit sehr stark im Vordergrund steht, ist die Variante über *SNMP* zu bevorzugen.

Hinweis: Man kann auch vor einem Test das Skript per *scp* auf den Zielserver kopieren, ausführen und wieder entfernen. Damit kann ein zentrales Skript-Repository realisiert werden. Die Skalierbarkeit auf Seiten von *OpenNMS* wird dann allerdings weiter reduziert, da ein einzelner Test weitaus mehr Ressourcen beansprucht.

4.2.5 Webserver als Agenten

Fast auf jedem Netzwerkgerät sind mittlerweile *Webserver* zur Konfiguration und Kontrolle von Statusinformationen vorhanden. Zusätzlich ist das *HTTP-Protokoll*[28] eines der wichtigsten Protokolle in Internet- und großen Netzwerkumgebungen. Die Durchlässigkeit von *HTTP(S)* über *Firewall-Systeme* oder *Proxies* ist sehr hoch. Damit kann sich dieses Protokoll ebenfalls sehr gut zur Überwachung über weit verteilte Standorte eignen.

Für die Überwachung verwenden wir den *OpenNMS* integrierten *HTTP-Monitor*. Dieser Monitor kann zum Einen *Webserver* prüfen ob *Web-Seiten* korrekt ausgeliefert und zum Anderen über reguläre Ausdrücke den Inhalt prüfen. In diesem Beispiel werden wir eine einfache *HTTP-Seite* einrichten die uns den Status des *smartctl* ausgibt. Den Status testen wir dann anschließen mit dem *HTTP-Monitor*. In dem gezeigten Anwendungsfall wird *CGI*[29] und ein *Bash-Skript* verwendet. Es ist fast überall direkt verwendbar und hat sehr wenig weitere Abhängigkeiten.

[28] HTTP Hyper Text Transfer Protokoll, man könnte es schon fast als Transportprotokoll des Internets nennen.
[29] Common Gateway Interface URL:`http://httpd.apache.org/docs/2.0/howto/cgi.html`

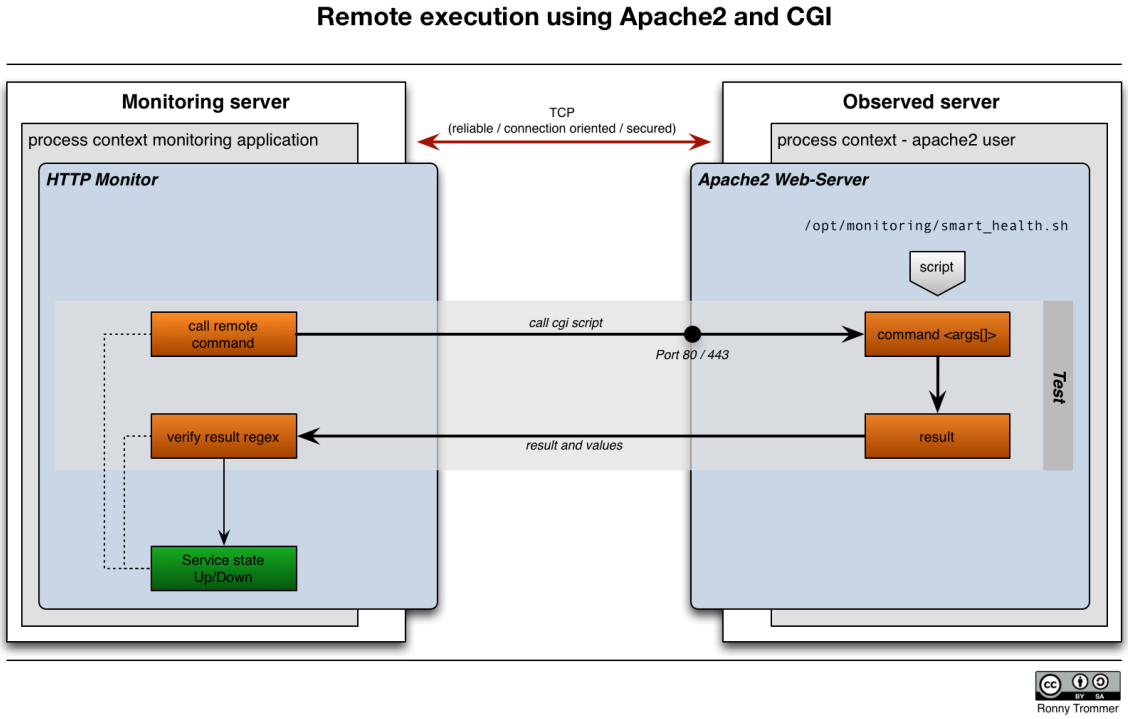

Abbildung 4.2.4: Ablauf von Abfragen über *CGI*

Hinweis: Es können natürlich analog *PHP*, *Perl* oder *Python* als Sprache zum generieren der *Web-Seite* verwendet werden.

Die hier gezeigte Konfiguration ist exemplarisch auf einem *Ubuntu 9.10 Server* eingerichtet worden. Die Installation von *apache2* wird mit dem folgenden Kommando durchgeführt.

```
aptitude install apache2
```

Einrichtung Apache2 mit CGI

Um die Ausführung von Kommandos über *CGI* zu ermöglichen, erlauben wir das Ausführen von Skripten zum Monitoring mit der folgenden Konfiguration.

```
vi /etc/apache2/sites-available/default
```

durchgeführt. Wir fügen einen weiteren Skriptpfad an und schränken die Nutzung auf IP-Ebene auf den *OpenNMS Server* ein.

```
1 ScriptAlias /cgi-mon/ /opt/monitoring/
2 <Directory "/opt/monitoring">
3         AllowOverride None
4         Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
5         Order deny,allow
6         Deny from all
7         Allow from <ip-opennms>/255.255.255.255
8 </Directory>
```

Listing 4.18: Konfiguration von *apache2* um *CGI* Skripte auszuführen

Der *Webserver* benötigt für die Ausführung des Programms *smartctl root-Rechte*. Der *Webserver* erhält daher die notwendigen Rechte um *smartctl* mit *sudo* aufrufen zu können. In dem Verzeichnis */opt/monitoring* erstellen wir ein Skript mit der Bezeichnung *check_smart_health.sh*.

Wichtig: Um die Ausführung der Skripte einzuschränken muss anstelle von <ip-opennms> eine entsprechende Adresse oder Adressbereich angegeben werden. Es gibt zur Zeit keine Möglichkeit im *HTTP-Monitor* eine Authentifizierung anzugeben. Der Zugang ist nur auf Basis von IP-Adressen oder Adressbereichen möglich.

Die Ausgabe des Skriptes wird über *HTTP* zugänglich gemacht. Die Rechte werden mit dem Kommando

```
visudo
```

wie folgt eingerichtet:

```
www-data        ALL=NOPASSWD: /usr/sbin/smartctl
```

Als Parameter wird die entsprechend zu testende Festplatte mit übergeben. Wir benutzen für CGI das *check_smart_health.sh*, wie wir es schon für *SNMP*, *NRPE* und *SSH* verwendet haben.

Das Skript wird anschliessend mit dem Kommando

```
chmod +x check_smart_health.sh
```

ausführbar gemacht. Die Ausführung des Skriptes testen wir in einem Browser über die URL

```
http://<entfernter-server>/cgi-mon/check_smart_health.sh?/dev/sda
```

Die Ausgabe sollte dann wie folgt aussehen:

Abbildung 4.2.5: Ablauf von Abfragen über *apache2* mit *CGI*

Auf diese Ausgabe kann nun ein *HTTP-Monitor* eingerichtet werden. Wir nutzen dazu die Möglichkeit die Ausgabe mit einem regulärem Ausdruck zu prüfen.

Einrichtung HTTP-Monitor in OpenNMS

In diesem Beispiel wird *Capsd* verwendet um die entsprechende Verfügbarkeit für das Monitoring automatisiert bereitzustellen. Der Monitor wird wie folgt angelegt:

```
vi $OPENNMS_HOME/etc/capsd-configuration.xml
```

```
<!-- START - Customized monitoring -->
<protocol-plugin protocol="HTTP-SMART-Health-sda"
                 class-name="org.opennms.netmgt.capsd.plugins.HttpPlugin"
                 scan="on">
    <property key="port" value="8880" />
    <property key="url" value="/cgi-mon/smart_health.sh?/dev/sda"/>
    <property key="timeout" value="3000" />
    <property key="retry" value="1" />
</protocol-plugin>

<protocol-plugin protocol="HTTP-SMART-Health-sdb"
                 class-name="org.opennms.netmgt.capsd.plugins.HttpPlugin"
                 scan="on">
    <property key="port" value="8880" />
    <property key="url" value="/cgi-mon/smart_health.sh?/dev/sdb"/>
    <property key="timeout" value="3000" />
    <property key="retry" value="1" />
</protocol-plugin>
<!-- END - Customized monitoring -->
```

Listing 4.19: Automatisches erkennen des *CGI-Skriptes* für den *S.M.A.R.T.* Festplattentest

Um die entsprechende Ausgabe zu testen werden zwei HTTP-Monitore für den Pollerd in der Konfiguratonsdatei angelegt.

```
vi $OPENNMS_HOME/etc/poller-configuration.xml
```

```
<!-- START - Customized monitoring -->
<service name="HTTP-SMART-Health-sda"
         interval="300000"
         user-defined="false"
         status="on">
    <parameter key="retry" value="10"/>
    <parameter key="timeout" value="5000"/>
    <parameter key="port" value="8880"/>
    <parameter key="url" value="/cgi-mon/smart_health.sh?/dev/sda"/>
    <parameter key="response-text" value="~.*PASSED.*" />
</service>
<service name="HTTP-SMART-Health-sdb"
         interval="300000"
         user-defined="false"
         status="on">
    <parameter key="retry" value="10"/>
    <parameter key="timeout" value="5000"/>
    <parameter key="port" value="8880"/>
    <parameter key="url" value="/cgi-mon/smart_health.sh?/dev/sdb"/>
    <parameter key="response-text" value="~.*PASSED.*" />
```

```
21  </service>
22  <!-- END - Customized monitoring -->
23  <!-- START - Customized monitoring -->
24  <monitor service="HTTP-SMART-Health-sda"
        class-name="org.opennms.netmgt.poller.monitors.HttpMonitor"/>
25  <monitor service="HTTP-SMART-Health-sdb"
        class-name="org.opennms.netmgt.poller.monitors.HttpMonitor"/>
26  <!-- END - Customized monitoring -->
```

Listing 4.20: Ausführen des Tests über das *CGI-Skript* mit *pollerd*

Nach einem Neustart von OpenNMS mit

```
service opennms restart
```

wird das entsprechende Skript aufgerufen und die Ausgabe getestet. Die Ausführung lässt sich ebenfalls über *HTTPS* verschlüsseln. Für den *Capsd* muss dann lediglich ein *HttpsPlugin* und im *pollerd* ein *HttpsMonitor* verwendet werden.

4.2.6 Fazit

Wie man sehen kann, sind die Möglichkeiten eigene Programme oder Skripte auszuführen sehr vielfältig. Die Entscheidung welche besser oder schlechter sind lässt sich pauschal nicht beantworten. Die Anforderungen an die Netzwerkinfrastruktur und entsprechende Sicherheitsrichtlinien sind sehr unterschiedlich. Aus diesem Grund werden hier eine Vor- und Nachteile der entpsrechenden Methoden beleuchtet. Die Anwendung lässt sich im Übrigen auch auf andere Netzwerk-Monitoring Anwendungen wie *Nagios* oder *Icinga* übertragen.

Im folgenden werden auf verschiedene Aspekte der Sicherheit, Wartbarkeit und Skalierbarkeit eingegangen.

Agent	Wartbarkeit	Skalierbarkeit	Sicherheit
SNMP	-	++	v1, v2c, v3 ++
NRPE	-	+	+
SSH	+	-	+
HTTP	-	+	+

Tabelle 4.2.2: Entscheidungshilfe für das Skript-Monitoring

Bei *SNMP*, *NRPE* und *HTTP* liegen auszuführende Skripte lokal auf jedem zu überwachendem Server. Die dezentrale Ablage muss mit entsprechenden Standards für ein Roll-Out in großen Umgebungen besonders berücksichtigt werden.

Bei *SSH* besteht zusätzlich die Möglichkeit per *scp* Skripte auf den Servern zu verteilen. Das kann ein Roll-Out erleichtern.

SNMP bietet die höchste Skalierbarkeit. Durch das schlanke und verbindungslose Protokoll *UDP*, werden sehr wenig Ressourcen beansprucht. Auf dem *OpenNMS Server* selbst kön-

nen bei *SNMP*, *NRPE* und *HTTP* die Monitor als leichtgewichtige *Java-Threads* ausgeführt werden. Bei *SSH* und dem *General Purpose Monitor* werden für jeden Test mit *fork()* ein Systemprozess erzeugt, was sich mit höherem Ressourcenverbrauch auf dem *OpenNMS Server* bermerkbar macht. Zusätzlich werden bei Protokollen die auf *TCP* basieren, eine verbindungsorientierte und bestätigte Kommunikation verwendet, die mehr Ressourcen verwendet. Bei *SSH* kommt als zusätzlicher Aufwand noch die Verschlüsselung hinzu. In Umgebungen die durch *Firewalls* abgesichert sind, lassen sich jedoch häufig *TCP-basierende* Protokolle einfacher einsetzen.

Bei *SNMP* (nur Version 3) kann eine Authentifizierung und Verschlüsselung verwendet werden. Bei *NRPE*, *SSH* und *HTTP* ist lediglich eine Authentifizierung auf IP-Ebene Möglich. Jeder der *root-Zugriff* auf den *OpenNMS Server* hat kann die entsprechenden Kommandos oder *Plugins* auf dem entfernten Server ausführen.

4.3 Automatisches Service-Recovery

Der Einsatz von OpenNMS wird hauptsächlich zu reinen Überwachungszwecken von IT-Systemen verwendet. In manchen Umgebungen kann es hilfreich sein, wenn einfache Wiederherstellungsmassnahemn automatisch ausgeführt werden. Erst wenn diese zu keinem Erfolg führen, wird Benachrichtigung an einen Bereitschaftsmitarbeiter ausgelöst. In diesem Beispiel wird beschrieben wie eine OpenNMS Eskalations-Hierarchie verwendet werden kann um einen solchen Anwendungsfall abzubilden.

WARNUNG: Bei der Umsetzung sollte klar sein, was es bedeutet automatisch ins System einzugreifen und Systemkommandos über entfernte Rechner auszuführen. Zusätzlich sollten Seiteneffekte bezüglich Sicherheit und kennwortloser SSH-authentifizierung berücksichtigt werden. Die Funktionsweise der Eskalation und Benachrichtigung in OpenNMS sollte bekannt sein. Bitte anwenden, wenn Sie wissen was Sie tun.

4.3.1 Beispielszenario mit Web-Servern

Im folgenden wird das Konfigurationsszenario kurz beschrieben. In dem gezeigten Beispiel gibt es eine Reihe wichtiger und weniger wichtiger Webserver auf denen Apache2 ausgeführt wird. Die wichtigen Server sind in einer *Surveillance Category* mit der Bezeichnung *VIP-HTTP* zusammenfasst. Für einen Ausfall des Dienstes *HTTP* versucht *OpenNMS* drei mal den *apache2* Dienst neu zu starten bevor eine *SMS* an den Administrator versendet wird. die folgende Abbildung stellt das Szenario kurz dar.

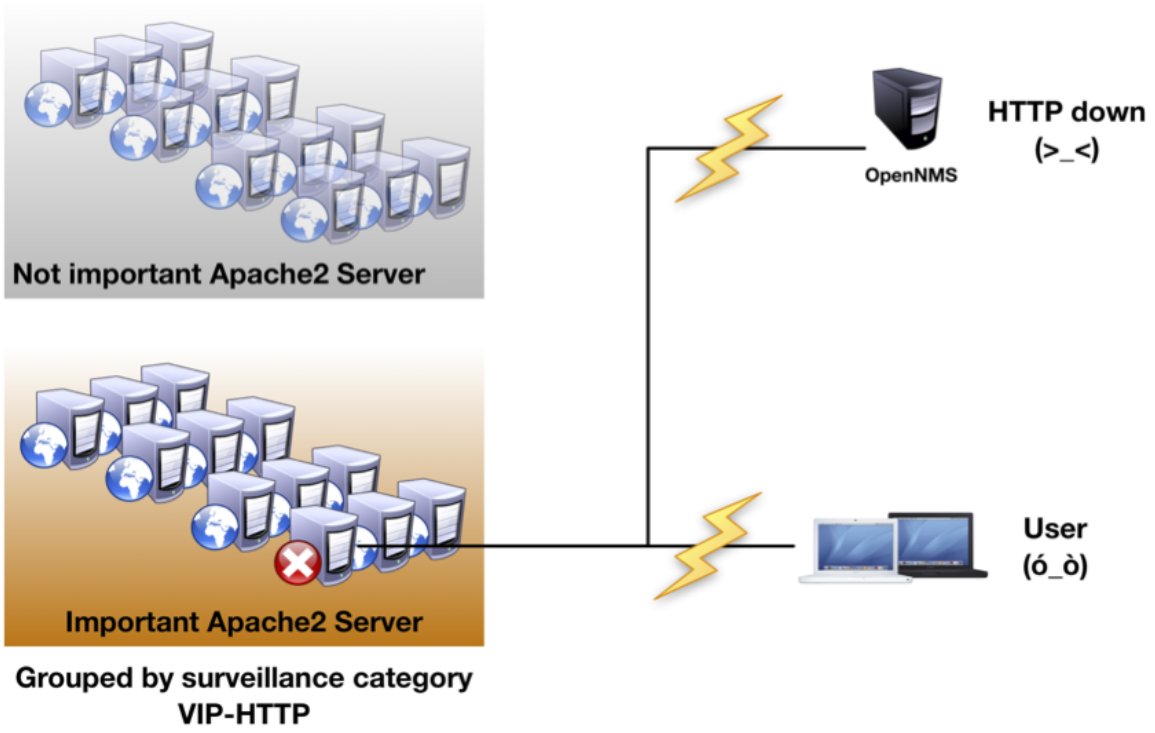

Abbildung 4.3.1: Wiederherstellung am Beispiel von Apache Web-Servern

Für die Einrichtung sind die im folgenden beschriebenen Voraussetzung notwendig. Der Ablauf stellt sich wie folgt dar:

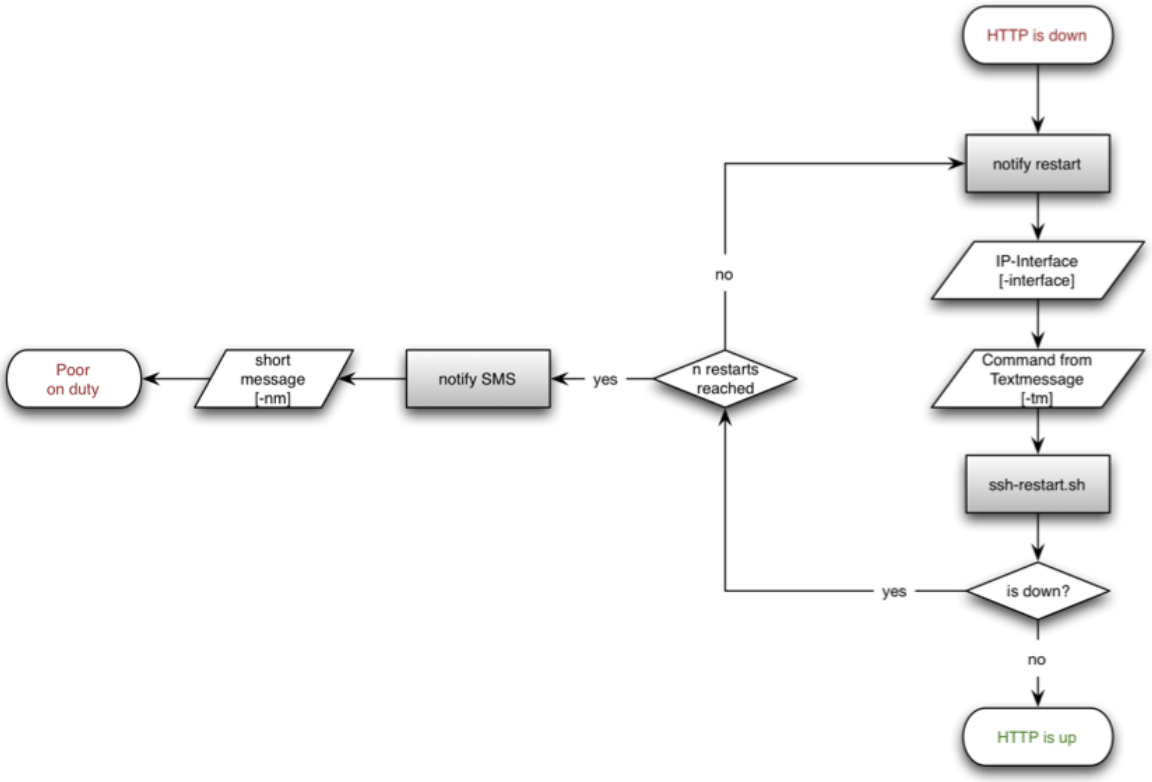

Abbildung 4.3.2: Ablaufdiagram der Benachrichtigung mit Eskalation zur Wiederherstellung des Apache Web-Servers in OpenNMS

4.3.2 Voraussetzungen

Der *OpenNMS* Server muss in der Lage sein, entfernte Kommandos per *SSH* ohne Kennworteingabe ausführen können. Dazu kann eine *pubkey-Authentifizierung* eingerichtet werden. Wie *pubkey-Authentifizierung* mit *ssh-keygen* unter Linuxdistributionen eingesetzt wird hier nicht weiter behandelt und auf die Internetseite `https://help.ubuntu.com/community/SSH/OpenSSH/Keys` verwiesen. Es ist wichtig, dass ***KEINE*** *passphrase* verwendet werden darf.

Um einen Serverdienst unter *Linux* neu starten zu können wird übelicherweise eine init-Skript verwendet. In unserem Beispiel wird *Apache2* verwendet. Das init-Skript welches beim Systemstart verwendet wird liegt in

```
    /etc/init.d/apache2 start|stop|restart
```

Der Parameter *restart* kann genutzt werden um einen bereits gestoppten oder fehlerhaft laufenden *apache2* Prozess neu zu starten. Ein *Wrapper-Script* übernimmt diese Aufgabe und kann wie folgt aussehen:

```
1  #!/bin/bash
2  # Wrapper to restart services automatically
3  # with SSH
4  # Use pubkey authentication for executing remote commands
5  #
```

```
 6  # Example usage:
 7  #   ssh-restart.sh 192.168.0.10 /etc/init.d/apache2 restart
 8  #
 9  # TODO: For productivity use, implement better error handling
10  #
11  #   - created: ronny@opennms.org
12  #
13  # HINT: For a little bit more security. Create a user
14  #       monitoring instead of root and configure sudo
15  #       correctly.
16
17  IP=${1}
18  SSH_USER=root
19  SSH_IDENTITY=/root/.ssh/id_rsa
20  SERVICE=${2}
21  RESTART_CMD=${3}
22  # SUDO=/usr/bin/sudo
23  LOG=/tmp/ssh-restart.log
24
25  CMD="ssh -i ${SSH_IDENTITY} ${SSH_USER}@${IP} ${SERVICE} ${RESTART_CMD}"
26  RESULT=$(${CMD} > /dev/null)
27  if [ $? -eq 0 ]; then
28    exit 0
29  else
30    echo "Error during executing ${CMD}" >> ${LOG}
31    exit 1
32  fi
```

Listing 4.21: *Wrapper-Skript* um einen service per *SSH* entfernt neu starten zu können

Das Skript erwartet drei Parameter und kann wie folgt ausgeführt werden:

```
ssh-restart.sh <ip-interface> <init-skript> <restart-command>
```

Wenn das Kommando vom *OpenNMS* Server auf den zu überwachenden Webserver funktioniert kann mit der Konfiguration der Benachrichtigung in OpenNMS fortgefahren werden.

4.3.3 Benachrichtigung erweitern

Das oben gezeigt Skript kann jetzt als neues Kommando für die Benachrichtigung eingerichtet werden. Dazu wird die Datei

```
$OPENNMS_HOME/etc/notificationCommands.xml
```

bearbeitet. ***Wichtig!*** Es ist darauf zu achten, dass die Pfadangabe in *<execute>...</execute>* korrekt ist.

```
1  <command binary="true">
2      <name>restart-service</name>
3      <execute>/opt/scripts/opennms/ssh-restart.sh</execute>
4      <comment>restart service</comment>
5      <argument streamed="false">
6          <switch>-interface</switch>
7      </argument>
8      <argument streamed="false">
9          <switch>-tm</switch>
```

```
10      </argument>
11  </command>
```

Listing 4.22: Benachrichtigung in OpenNMS mit *ssh-restart.sh* erweitern

Über die *-interface* und *-tm* können über die Benachrichtigung die beiden wichtigen Parameter für unser *SSH-Skript* mit übergeben werden. Nach einem Neustart kann unser *restart-service* im *Destination Path* ausgewählt werden.

4.3.4 Eskalation einrichten

Wir erzeugen in der Weboberfläche einen neuen *Destination Path* mit der Bezeichnung *restart-service*. Über die Eskalation wird festgelegt, wie oft versucht werden soll den Dienst neu zu starten. In der folgenden Abbildung ist die Konfiguration dargestellt. Über einen Benutzer *remote* wurde kenntlich gemacht, dass hier ein remote-Kommando drei mal in einem 5-minütigem Abstand ausgeführt werden soll. Falls der *HTTP-Dienst* nach dem dritten mal noch nicht wiederhergestellt ist, wird eine *SMS* an den Benutzer *indigo* gesendet.

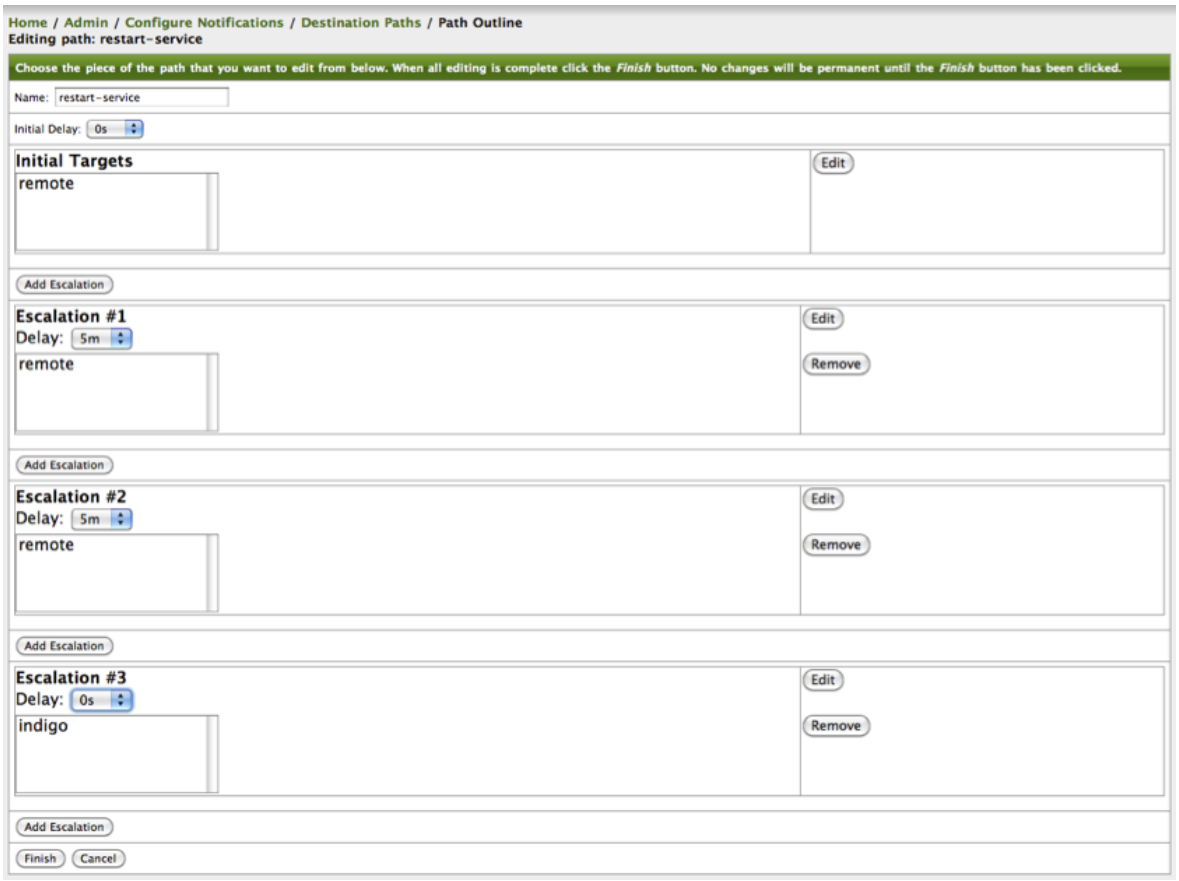

Abbildung 4.3.3: Eskalation für *service-restart* Benachrichtigung

Wichtig! Für den Benutzer *remote* ist zu beachten, dass kein Kommando ausgeführt wird, wenn der Dienst wieder verfügbar ist. Daher hier den Schalter auf *off* setzen. Damit wird

verhindert, dass Benachrichtigungen für *RESOLVED-Events* gesendet werden.

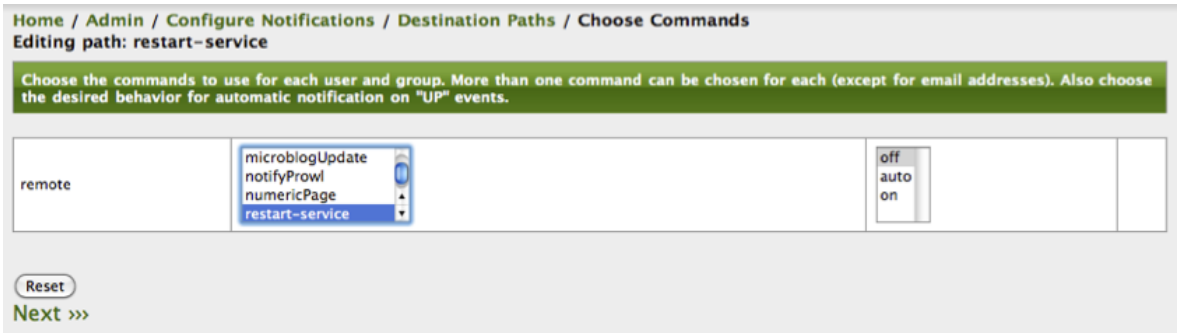

Abbildung 4.3.4: Konfiguration eines *Destination Path* für *Service-Recovery*

Die Benachrichtigung für *SMS* kann wie gewohnt eingerichtet werden. Im nächsten Schritt wird die Konfiguration für das *nodeLostService Event* beschrieben.

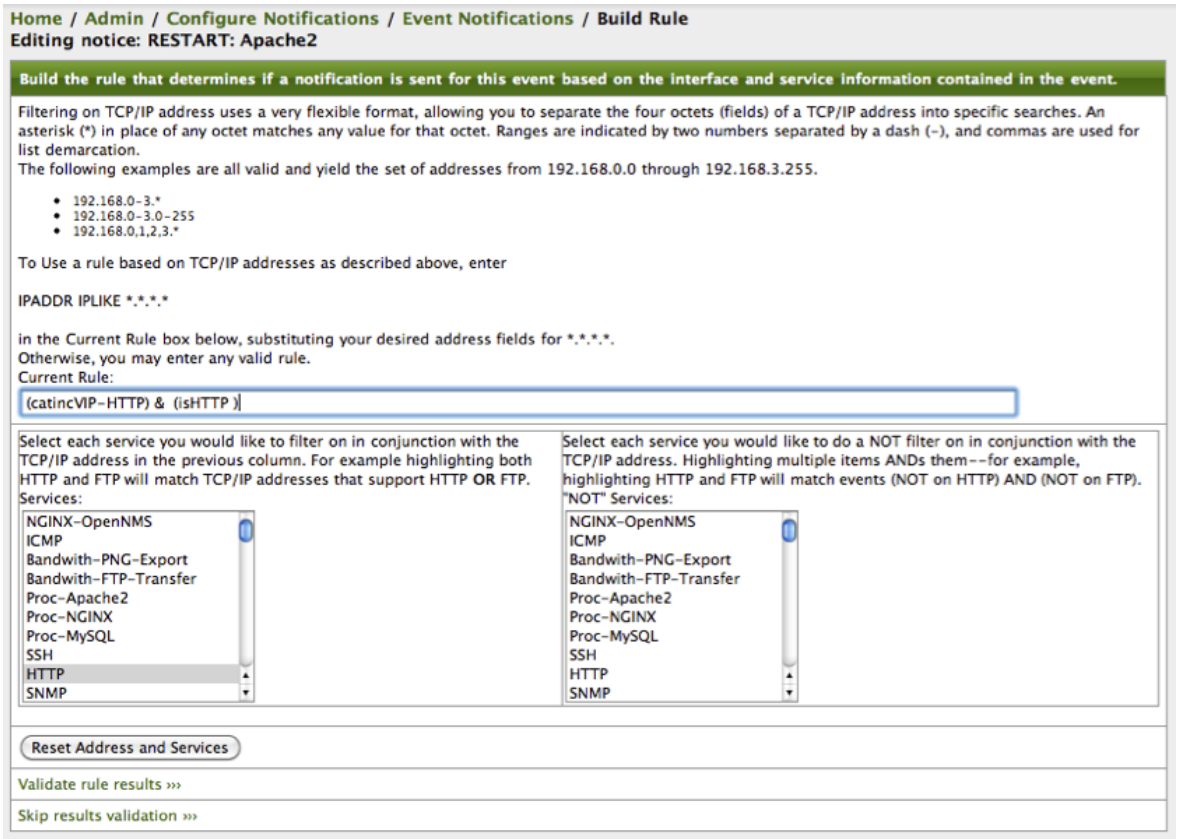

Abbildung 4.3.5: Regel für Benachrichtigung für relevante *VIP-HTTP Server*

Der *Apache2* soll nur für *Nodes* neu gestartet werden die den Service *HTTP* und in der Gruppe *VIP-HTTP* enthalten sind.

```
    (catincVIP-HTTP) & (isHTTP)
```

Die Regel kann über *Validate rule* geprüft und dann bestätigt werden. Im nächsten Schritt wird die eigentliche Konfiguration dargestellt. Das Feld *Text Message* hat hier eine besondere Bedeutung. Es enthält den Parameter dem Skript *ssh-restart.sh* mit übergeben wird.

Schlägt das neu starten des *Apache2-Service* fehl, wird eine *SMS* versendet. Diese Benachrichtigung verwendet das *Short Message* Textfeld als Nachrichtentext.

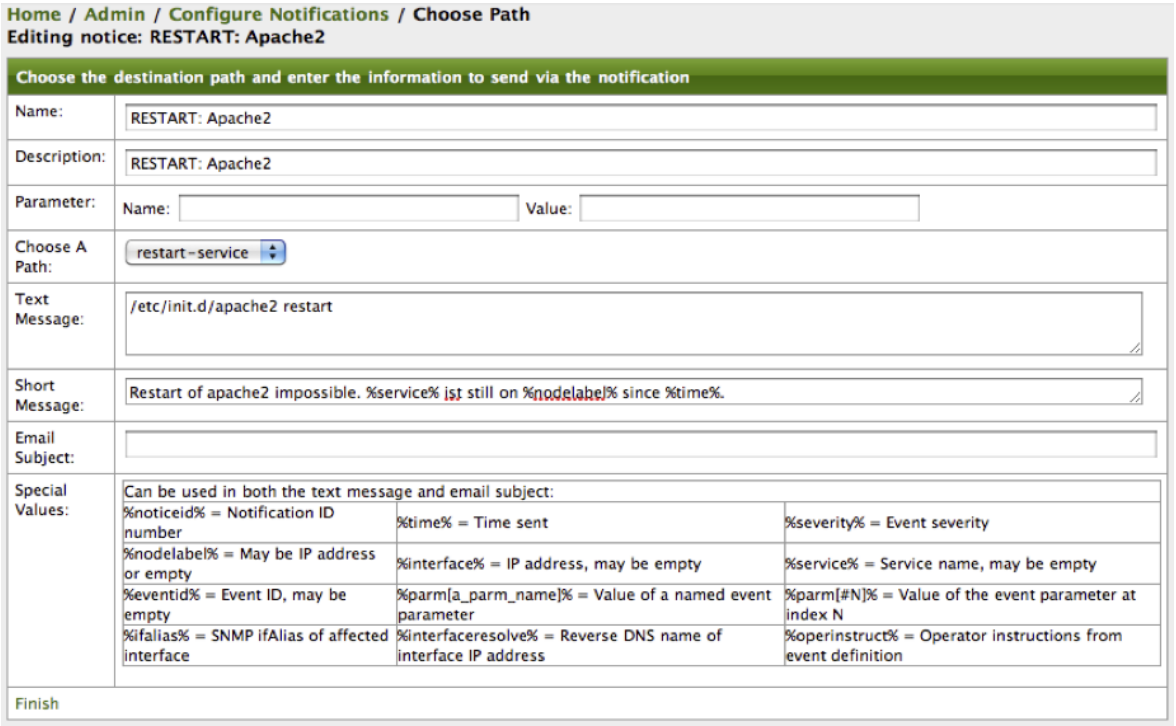

Abbildung 4.3.6: Text für die Benachrichtigung und *restart service* Kommando.

Es wird das Kommando */etc/init.d/apache2 restart* an das *notification command* übergeben. Mit dem angelegten *Destination path* können beliebige Dienste neu gestartet werden. Die Meldung in *Short Message* wird nach dem dritten Versuch des Neustarts als *SMS-Text* versendet.

Listings