Eigene Check Bilbliothek

Hier landen eigen gebaute Checks

- Endian VPN Überwachung
- MongoDB Größen überwachung
- Process Speicherüberwachung

Endian VPN Überwachung

Beschreibung:

Ein Custom Check für ein python Script zur endian VPN überwachung.
Dieses script gibt Graphen mit Anzahl der Verbindung für jede VPN Instanzen aus.
und eine Gesamtverbindungsübersicht für alle VPN Instanzen.

Das Script:

Endian RVS5

```
#!/bin/bash

STRING=$(python /root/checkmk_setup/check_metrics4_rvs5.py openvpn_users | grep
openvpn)
IFS=' ' read -ra vpn_array <<< "$STRING"
LEN=${#vpn_array[@]}
printf "0 \"OPENVPN Users\" "
for (( i=5; i<$LEN; i++))
do
    IFS=' ' read -ra part <<< ${vpn_array[$i]}
    printf "VPNCONF-${part[0]}"
    if [ $i -lt $(expr $LEN - 1) ]
       then
           printf "|"
    fi
done
printf "\n"



#echo "0 \"OpenVPN Users\" metric=10|metric1=20|metric3=40"
```

Endian RVS6

```
#!/bin/bash

STRING=$(python /root/checkmk_setup/check_metrics4.py openvpn_users | grep openvpn)
IFS=' ' read -ra vpn_array <<< "$STRING"
LEN=${#vpn_array[@]}
printf "0 \"OPENVPN Users\" "
for (( i=5; i<$LEN; i++))
do
    IFS='=' read -ra part <<< ${vpn_array[$i]}
    printf "VPNCONF-${part[0]}=${part[1]}"
    if [ $i -lt $(expr $LEN - 1) ]
       then
          printf "|"
    fi
done
printf "\n"



#echo "0 \"OpenVPN Users\" metric=10|metric1=20|metric3=40"
```

Es ruft

```
python /root/checkmk_setup/check_metrics4.py openvpn_users | grep openvpn
```

Ausgabe:

```
openvpn_users_total: 7 users (OK) | openvpn.2.conf_rvs6test_UDP_tecmata=6
openvpn.1.conf_rvs6test_TCP_tecmata=1 total_vpn_users=7
```

Diese Ausgabe muss dann nur noch in das Checkmk Format gebracht werden. Das macht das Script.
Die Ausgabe vom Script sieht dann so aus.
Befehl

```
. /usr/lib/check_mk_agent/local/vpn


Ausgabe:
0 "OPENVPN Users" VPNCONF-openvpn.2.conf_rvs6test_UDP_tecmata=6|VPNCONF-
```

```
openvpn.1.conf_rvs6test_TCP_tecmata=1|VPNCONF-total_vpn_users=7
```

Überarbeitet Script mit Metriken für Schwellwerte. Der Doppelpunkt hinter einem Wert heißt darunter und nicht darüber.

Endian RVS5

```
#!/bin/bash
WARN=8
CRIT=1
STRING=$(python /root/checkmk_setup/check_metrics4_rvs5.py openvpn_users | grep openvpn)
IFS=' ' read -ra vpn_array <<< "$STRING"
LEN=${#vpn_array[@]}
printf "P \"OPENVPN Users Metrik\" "
for (( i=5; i<$LEN; i++))
do
        IFS=' ' read -ra part <<< ${vpn_array[$i]}
        printf "VPNCONF-${part[0]}"
        if [ $i -lt $(expr $LEN - 1) ]
        then
                printf "|"
        else
                printf ";$WARN:;$CRIT:"
        fi
done
printf "\n"



#echo "0 \"OpenVPN Users\" metric=10|metric1=20|metric3=40;8:;1:"
```

Endian RVS6

```
#!/bin/bash
WARN=8
CRIT=1
STRING=$(python /root/checkmk_setup/check_metrics4.py openvpn_users | grep openvpn)
```

```
IFS=' ' read -ra vpn_array <<< "$STRING"
LEN=${#vpn_array[@]}
printf "P \"OPENVPN Users Metrik\" "
for (( i=5; i<$LEN; i++))
do
        IFS='=' read -ra part <<< ${vpn_array[$i]}
        printf "VPNCONF-${part[0]}=${part[1]}"
        if [ $i -lt $(expr $LEN - 1) ]
        then
                printf "|"
        else
                printf ";$WARN:;$CRIT:"
        fi
done
printf "\n"



#echo "0 \"OpenVPN Users\" metric=10|metric1=20|metric3=40;8:;1:"
```

Installation des VPN Python Script im System

In endian RVS5

Der unterschied in RVS5 zu 6 sind die Pfade für die OPENVPN Log

In RVS5 liegen diese unter os.chdir("/var/volatile/tmp")
Somit muss im Script der Pfad angepasst werden.

Nach os.chdir im Abschnitt openvpnuser suchen und anpassen nach os.chdir("/var/volatile/tmp")

Wurde aber schon gemacht in der Datei : check_metrics4_rvs5.py

Die check_metrics4_rvs5.py auf den RVS kopieren.
Zum Beispiel nach ~/checkmk_setup/

Das Script erstellen mit

```
nano /usr/lib/check_mk_agent/local/vpn
```

Das das Script rein, mit oder Ohne Metrik. Halt was gewünscht ist.
Wenn mit Metrik, nicht vergessen die Metrik Werte anzupassen.
Und den Pfad zur Checkmetric4 Python File anpassen, da es ja RVS5 oder 6 sein kann

nun das Script noch ausführbar machen

```
chmod +x /usr/lib/check_mk_agent/local/vpn
```

Nun erscheint der Service in den Services vom Host

In endian RVS6

Der unterschied in RVS5 zu 6 sind die Pfade für die OPENVPN Log

In RVS6 liegen diese unter os.chdir("/run/openvpn")
Somit muss im Script der Pfad angepasst werden.
Nach os.chdir im Abschnitt openvpn suchen und anpassen nach os.chdir("/run/openvpn")

Wurde aber schon gemacht in der Datei : check_metrics4.py

Die check_metrics4.py auf den RVS kopieren.
Zum Beispiel nach ~/checkmk_setup/

Das Script erstellen mit

```
nano /usr/lib/check_mk_agent/local/vpn
```

Das das Script rein, mit oder Ohne Metrik. Halt was gewünscht ist.
Wenn mit Metrik, nicht vergessen die Metrik Werte anzupassen.
Und den Pfad zur Checkmetric4 Python File anpassen, da es ja RVS5 oder 6 sein kann

nun das Script noch ausführbar machen

```
chmod +x /usr/lib/check_mk_agent/local/vpn
```

Nun erscheint der Service in den Services vom Host

MongoDB Größen überwachung

Beschreibung:

Ein lokal check, der alle collections durchgeht und ausgibt.
Der Check muss auf dem MongoDB Server eingefügt werden, da er eijne Localhostverbindung zum MongoDB Server aufbaut.

Das Script

```
#!/usr/bin/python3
from pymongo import MongoClient

# Konstanten für Schwellwerte
WARNUNGSGROESSE_GB = 0.5
KRITISCHGROESSE_GB = 1.0

# Verbindung zur MongoDB herstellen
client = MongoClient('mongodb://localhost:27017/')

# Alle Datenbanknamen abrufen
db_names = client.list_database_names()

# Durch jede Datenbank iterieren und die Statistiken abrufen
for db_name in db_names:
    db = client[db_name]
    stats = db.command('dbstats')

    # Größe in Gigabytes umrechnen
    groesse_gb = stats['dataSize'] / (1024**3)

    # Prüfe die Größe gegen Schwellwerte und gebe entsprechenden Status aus
    if groesse_gb > KRITISCHGROESSE_GB:
```

```
        status = 2  # Kritisch
        status_message = "CRITICAL"
    elif groesse_gb > WARNUNGSGROESSE_GB:
        status = 1  # Warnung
        status_message = "WARNING"
    else:
        status = 0  # OK
        status_message = "OK"
    #Zeile um den Status ausgeben, wenn checkmk nicht benutzt wird, hier auskommentiert da es ja ein checkmk Check wird
    #print(f"{status} MongoDB_{db_name}_Groesse - {status_message} - Groesse: {groesse_gb:.2f} GB")
    #Zeile die Augegeben wird für Check_MK
    print(f"P \"MongoDB_{db_name}_Groesse\" size={groesse_gb:.2f};{WARNUNGSGROESSE_GB};{KRITISCHGROESSE_GB}")

# MongoDB-Verbindung schließen
client.close()
```

Hier auchnochmals als download: mongodbsizes.py

Installation:

Das script in folgendem Pfad anlegen bzw. hin kopieren

```
/usr/lib/check_mk_agent/local/mongodbsizes.py
```

Nun noch ausführbar machen.

```
chmod +x /usr/lib/check_mk_agent/local/mongodbsizes.py
```

Fertig

Process Speicherüberwachung

Beschreibung:

Ein lokal check der alle Prozesses nach Speichergröße sortiert.
Auf Grund der Prozessmenge auf 20 Stück Limitiert.
Bei diesem Check gehts auch nur darum die Speicherfresser zu finden.
Die Schwellwerte WARN und CRIT können über die Variablen WARN und CRIT gesetzt werden. Die Einheit ist MB

Das Script

Einmal als Einzelauflistung:

```
#!/usr/bin/python3
import subprocess
import re

# Schwellenwerte in Megabyte
WARN = 1000  # Beispiel: 1000 MB
CRIT = 2000  # Beispiel: 2000 MB
COUNT_PROC = 20  # Anzahl der Prozesse, die ausgeben werden sollen - maximal 20

# Programmvariablen, bitte nicht verändern
output = "P \"Speicherverbrauch der Prozesse in MB\" "

# Führe den `ps`-Befehl aus, um Prozessname und Speicherverbrauch (RSS) zu bekommen
process = subprocess.Popen(['ps', '-eo', 'rss,comm', '--sort=-rss'], stdout=subprocess.PIPE)
stdout = process.communicate()[0]

# Verarbeite jede Zeile der Ausgabe
for i, line in enumerate(stdout.decode('utf-8').strip().split('\n')[1:]):  # Überspringe die Kopfzeile
    if i >= COUNT_PROC:  # Limit auf COUNT_PROC Prozesse
```

```
        break

    parts = line.split(None, 1)
    memory = int(parts[0]) // 1000  # Konvertiere RSS von KB zu MB
    process = re.sub('[^a-zA-Z0-9]', '_', parts[1])  # Ersetze alle nicht alphanumerischen Zeichen durch
Unterstriche

    # Hinzufügen der Performance-Daten für jeden Prozess zum Output
    output += f"NR-{i:02d}-{process}={memory};{WARN};{CRIT}|"

# Entferne das letzte Semikolon
output = output.rstrip('|')

# Entferne Kommas und Unterstriche (wenn nötig)
output = output.replace(',', '').replace('_', '')

# Ausgabe des gesammelten Outputs
print(output)
```

Ausgabe:

```
P "Speicherverbrauch der Prozesse in MB" NR-00-kvm=67338;1000;2000|NR-01-kvm=4382;1000;2000|NR-02-
kvm=4307;1000;2000|NR-03-cephosd=3305;1000;2000|NR-04-cephosd=3099;1000;2000|NR-05-
cephosd=2875;1000;2000|NR-06-cephosd=2679;1000;2000|NR-07-kvm=1879;1000;2000|NR-08-
kvm=1264;1000;2000|NR-09-kvm=916;1000;2000|NR-10-java=576;1000;2000|NR-11-
cephmon=491;1000;2000|NR-12-kvm=436;1000;2000|NR-13-kvm=420;1000;2000|NR-14-
cephmgr=325;1000;2000|NR-15-mongod=167;1000;2000|NR-16-corosync=165;1000;2000|NR-17-
pveproxy=157;1000;2000|NR-18-launcher=154;1000;2000|NR-19-pveproxy=144;1000;2000
```

Als Gesamtspeicherzusammenfassung:

```
#!/usr/bin/python3
import subprocess
import re

# Schwellenwerte in Megabyte
WARN = 1000  # Beispiel: 1000 MB
CRIT = 2000  # Beispiel: 2000 MB
COUNT_PROC = 20  # Anzahl der Prozesse, die ausgegeben werden sollen - maximal 20
```

```
# Programmvariablen, bitte nicht verändern
output = "P \"Speicherverbrauch der Prozesse in MB\" "

# Führe den `ps`-Befehl aus, um Prozessname und Speicherverbrauch (RSS) zu bekommen
process = subprocess.Popen(['ps', '-eo', 'rss,comm', '--sort=-rss'], stdout=subprocess.PIPE)
stdout = process.communicate()[0]

# Speichere Speicherverbrauch pro Prozessname in einem Wörterbuch
memory_usage = {}

# Verarbeite jede Zeile der Ausgabe
for line in stdout.decode('utf-8').strip().split('\n')[1:]:  # Überspringe die Kopfzeile
    parts = line.split(None, 1)
    memory = int(parts[0]) // 1000  # Konvertiere RSS von KB zu MB
    process_name = re.sub('[^a-zA-Z0-9]', '_', parts[1])  # Ersetze alle nicht alphanumerischen Zeichen durch
Unterstriche

    # Addiere den Speicherverbrauch für gleiche Prozessnamen
    if process_name in memory_usage:
        memory_usage[process_name] += memory
    else:
        memory_usage[process_name] = memory

# Sortiere Prozesse nach Speicherverbrauch, beschränke auf COUNT_PROC Einträge
sorted_processes = sorted(memory_usage.items(), key=lambda x: x[1], reverse=True)[:COUNT_PROC]

# Generiere die Ausgabezeile für die Prozesse
for i, (process, mem) in enumerate(sorted_processes):
    output += f"{process}={mem};{WARN};{CRIT}|"

# Entferne das letzte Semikolon
output = output.rstrip('|')

# Ausgabe des gesammelten Outputs
print(output)
```

Ausgabe:

```
P "Speicherverbrauch der Prozesse in MB"
kvm=80987;1000;2000|ceph_osd=12535;1000;2000|java=588;1000;2000|ceph_mon=485;1000;2000|pvedaem
```

```
on_worke=423;1000;2000|pveproxy_worker=423;1000;2000|ceph_mgr=325;1000;2000|corosync=165;1000;2000|mongod=165;1000;2000|pveproxy=157;1000;2000|launcher=154;1000;2000|pvedaemon=136;1000;2000|pvescheduler=113;1000;2000|pve_ha_crm=111;1000;2000
```

Installation

Auf dem zu überwachenden Server nach

```
nano /usr/lib/check_mk_agent/local/process.py
chmod +x /usr/lib/check_mk_agent/local/process.py
```

kopieren oder editieren über einfügen