Hochschule für Telekommunikation Leipzig (FH)
Studiengang Kommunikations- und Medieninformatik

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Engineering

Thema: Analyse der Migration auf das Monitoring System ICINGA 2 im OSS Bereich der Deutschen Telekom Technik

Vorgelegt von: Benjamin Bloßfeld

geboren am: 24.05.1988
in: Sangerhausen

Themensteller: Deutsche Telekom AG
Kärrnerstr. 66
04288 Leipzig

Erstprüfer: Dipl.-Ing. Michael Flegl

Zweitprüfer: M. Eng. Henrik Burmeister

Vorwort

Einordnung der Arbeit

Diese Arbeit wurde im Rahmen des 6. Semesters des Bachelor-Studiengangs Kommunikations- und Medieninformatik an der Hochschule für Telekommunikation erstellt. Die Bachelorarbeit wurde von der Deutschen Telekom AG in Auftrag gegeben.

Auftraggeber

Die Deutsche Telekom AG ist das größte deutsche Telekommunikationsunternehmen. Weltweit werden über 228.000 Mitarbeiter in über 50 Ländern beschäftigt (Stand Dezember 2014). Durch die Deutsche Telekom AG werden verschiedenste Informations- und Kommunikations-Dienste betrieben.
Das Unternehmen ging 1995 durch Inkrafttreten der zweiten Postreform aus dem Bereich „Telekommunikation und Fernmeldewesen“ der im Staatsbesitz befindlichen Deutschen Bundespost hervor. Es existieren verschiedene Tochtergesellschaften, zu denen auch die Telekom Deutschland GmbH gehört. Im Laufe der Jahre gab es diverse Umstrukturierungen, welche letztendlich zu der aktuellen Unterteilung der Telekom Deutschland GmbH zu drei Tochtergesellschaften geführt hat: Deutsche Telekom Technik GmbH (DTT), Deutsche Telekom Technischer Service GmbH (DTTS), Deutsche Telekom Technischer Kundenservice GmbH (DTKS). Die Bachelorarbeit wird im Bereich der DTTS durchgeführt. Zu den Aufgaben der DTTS gehören das Planen, Bauen und Betreiben von Systemen und Einrichtungen der technischen Infrastruktur sowie der Einführung neuer Technologien innerhalb Deutschlands.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AES:	Advanced Encryption Standard
API:	Application Programming Interface
CA:	Certificate Authority
CPU:	Central Processing Unit
CRD:	Controlled Remote Diagnosis
DCN:	Data Communication Network
DES:	Data Encryption Standard
DRBD:	Distributed Replicated Block Device
DTAG:	Deutsche Telekom Aktiengesellschaft
FNO:	Fibre Network Operations
HTTP:	Hypertext Transfer Protocol
IDODB:	Icinga Data Out Database
I/O:	Input/Output
iLO:	integrated Lights Out
IMS:	IP Multimedia Subsystem
IP:	Internet Protocol
IT:	Informationstechnik
JSON-RPC:	Javascript Object Notation-Remote Procedure Call
LAN:	Local Area Network
mDCN:	mobiles Data Communication Network
MIB:	Management Information Base
NFS:	Network File System
NMS:	Network Management System
NRPE:	Nagios Remote Plugin Executor
NSCA:	Nagios Service Check Acceptor
OAM SA:	Operation and Maintenance Service Area
OID:	Object Identifier
OMD:	Open Monitoring Distribution
OSPS:	Operations Support Systems Platform Services
OSS:	Operations Support Systems
P & I:	Products and Innovations
POP3:	Post Office Protocol Version 3
RAM:	Random Access Memory
RegEx:	Regular Expressions
RFC:	Request for Comment
SMTP:	Simple Mail Transfer Protocol
SNMP:	Simple Network Management Protocol
SQL:	Structured Query Language
SSH:	Secure Shell
SSL:	Secure Sockets Layer
TACN:	Test and Acceptance Comunnication Network
TCP:	Transport Control Protocol
TDCN:	Telekom Data Communication Network
TMDC:	Telekom Managed Data Center
TSI:	T-Systems
TLS:	Transport Layer Security
VHD:	Virtual Hard Disk
ZDCN:	Zentrales Data Communication Network

Abbildungsverzeichnis

Tabellenverzeichnis

2 Einleitung

Da die Komplexität von Systemen stetig zunimmt, wird es immer wichtiger, ein angemessenes Netzwerkmanagement zu betreiben. Als Netzwerk- oder auch Systemmanagement wird dabei die Gesamtheit aller Handlungen, Vorgehensweisen und Modelle bezeichnet, die zum Betrieb, der Verwaltung und der Überwachung von Systemen erforderlich sind. Das große Ziel stellt die Aufrechterhaltung des Regelbetriebes und somit der Produktivität dar.
Ohne entsprechende Managementsysteme kann im Fehlerfall ein enormer Aufwand anfallen, um die Funktionalität des eigentlichen Systems wiederherzustellen. Dies kann zu hohen Einnahmeverlusten führen und im Extremfall für Firmen sogar existenzbedrohend sein. Somit wird auch die Managementleistung selbst zunehmend zum Produkt, um Geld zu verdienen.
Besonders der Bereich des Monitorings ist hierbei von großer Bedeutung. Monitoring bedeutet in diesem Zusammenhang die systematische Überwachung und Darstellung von Qualitätsparametern, wie beispielsweise Verfügbarkeit oder Kapazität einer Ressource. Damit ist es möglich, die Zustände der Ressourcen übersichtlich darzustellen. Durch das wiederholte, regelmäßige Überwachen können die Qualitätsparameter unter anderem in Liniendiagrammen oder anderen Visualisierungsformen dargestellt werden, wodurch Langzeittrends gut erkennbar werden, um so Fehler schon frühzeitig abschätzen und entsprechende Gegenmaßnahmen ergreifen zu können. Auch verfügen viele Monitoring-Systeme über Benachrichtigungsfunktionen, die zusätzlich zu einer schnellstmöglichen Problembehebung beitragen.
All dies ist natürlich besonders in einem großen Telekommunikationsunternehmen wie der Deutschen Telekom AG essenziell. Hier ist der Bereich Operations Support Systems Platform Services (OSPS) für das Monitoring einer Vielzahl an Servern und Services verantwortlich. Generell befasst sich OSPS mit der Bereitstellung von Plattformen und Middleware[1]. So werden physikalische und virtualisierte Infrastruktursysteme betrieben. Sowohl Desktop-, als auch Servervirtualisierungen und Datacenter-Solutions[2] werden von OSPS für interne Kunden bereitgestellt. Des Weiteren gehören Life Cycle Management[3], Qualitäts-, Kosten- und Risikomanagement, sowie die Harmonisierung der verschiedenen Systeme und entsprechendes Projektmanagement zu den erweiterten Aufgaben. Außerdem ist OSPS verantwortlich für alle Cluster der Platform Services Operations Support Systems (OSS) und die Beschaffung von Hard- und Softwarekomponenten für OSS-Technik.
Ein Teilbereich von OSPS bildet das Team Fibre Network Operations (FNO). Hier werden visualisierte Applikationen und Desktops, physikalische und virtuelle Serversysteme und Plattformen sowie physikalische Clients zur Verfügung gestellt und konfiguriert. Für alle bereitgestellten Instanzen sowie mehrere telekominterne Systeme übernimmt FNO Change-, Incident- und Problemmanagement als auch die Useradministration. Außerdem bietet und betreibt OSPS diverse Tools für Alarm-Monitoring, Datenbanken, Backup-Services und weitere Dienste. Im Team FNO wird aktuell das Monitoringsystem Icinga verwendet, um verschiedenste Hardwarekomponenten oder Services in ganz Deutschland zu überwachen. Mittlerweile ist mit Icinga 2 allerdings schon ein Nachfolgesystem verfügbar. Die Migration von Icinga 1 auf Icinga 2 soll im weiteren Verlauf dieser Arbeit untersucht werden.

[1]Middleware – zusätzliche Schicht in einer Software-Architektur, die den Zugriff auf untergeordnete Schichten vereinfachen soll

[2]Datacenter-Solutions – Maßnahmen zur Bereitstellung und Optimierung von Rechenzentren

[3]Life Cycle Management - Konzept zur Rückführung aller Einflüsse und Ergebnisse aller Phasen im Lebenszyklus eines Produkts zur Verbesserung des selbigen

1.1 Zielstellung

Bereits mit dem aktuellen Icinga 1-System werden mehrere tausend Hosts und Services überwacht. Im Laufe der Jahre stieg die Anzahl der intern zu überwachenden Kunden immer weiter an und auch in naher Zukunft ist davon auszugehen, dass sich dieser Trend fortsetzt. Da im momentan aktiven System zeitweise schon Performance-Probleme auftreten, wurde sich dazu entschieden eine Migration auf ein neues Monitoring-System zu vollziehen. Man fürchtete innerhalb der Telekom auch, dass die Icinga-Entwickler ein Hauptaugenmerk auf die Entwicklung der Nachfolgeversion legen würden, wodurch Updates für das ursprüngliche Icinga ausblieben und die Aktualität nicht mehr gegeben wäre. Außerdem wurde von offizieller Seite der Icinga-Entwickler Icinga 2 gegenüber Icinga 1 mit einem großen Performance-Gewinn angepriesen.
Da telekom-intern zudem ein Umstieg auf eine barrierefreie Version der Web-Oberfläche Icinga Web 2 in Planung ist, wurde entschieden im Zuge dieser Umstellung ein Projekt in Auftrag zu geben, um die Migration von Icinga 1 auf Icinga 2 zu untersuchen.
Dabei ist zu untersuchen, welche Unterschiede sich bei den Icinga-Versionen ergeben und inwiefern diese Auswirkungen auf den Betrieb in der Telekom-Infrastruktur haben. Demnach gilt es zu analysieren, ob alle wichtigen Funktionalitäten auch nach einer Migration zur Verfügung stehen würden oder ob eventuelle Anpassungen nötig wären. Außerdem soll dargestellt werden, auf welche Arten diese Migration geschehen kann, welchen Aufwand diese verursachen würde und inwiefern die Migration als sinnvoll zu erachten ist. Die unbedingt benötigten Funktionalitäten, welche im Laufe der Arbeit untersucht werden sollen, werden nach der Darstellung der aktuellen System-Struktur im Punkt 2.2.3 näher erläutert.

1.2 Abgrenzung

Für ein vollständiges Verständnis der Arbeit sind verschiedene Grundlagen und auch gewisse Vertiefungen in bestimmten Themengebieten notwendig. Damit sich auf die wesentlichen Aspekte des Themas konzentriert werden kann und um die Arbeit nicht über Gebühr anwachsen zu lassen, wird auf einige dieser Grundlagen und Vertiefungen im Verlauf dieses Dokuments nicht weiter eingegangen. Folgende Themen sind darin eingeschlossen:

- Linux-Betriebssystem
- Prozess-Scheduling
- Rechnerarchitekturen
- TCP/IP
- Servervirtualisierung

2 Grundlagen

Auf den folgenden Seiten werden allgemeine Erläuterungen gegeben, die für das Verständnis der Arbeit benötigt werden. Es wird auf das aktuell vorhandene System in Zusammenhang mit der verwendeten Netzstruktur eingegangen und eine Präzisierung der Zielstellung erfolgen.

2.1 Icinga allgemein

Icinga ist eine Open Source Anwendung zur Netzwerk- und Systemüberwachung. Es ging im Jahr 2009 als Fork (Abspaltung) von der Monitoring-Software Nagios hervor. Die Entwicklung wird durch die Community vorangetrieben. Es gibt fünf Entwicklerteams; der Großteil besteht aus Entwicklern der Firma Netways GmbH. Allerdings ist es jedem möglich, sich bei der Entwicklung von Icinga einzubringen.
Mit Icinga ist es möglich, Netzwerkdienste (z.B. SNMP, HTTP, POP3), Host-Ressourcen (z.B. Speicherauslastungen, laufende Prozesse, Log-Einträge) oder Hardwarekomponenten (z.B. Router, Switches, Server) zu überwachen. Eine Anzeige der gesammelten Daten erfolgt über eine Web-Schnittstelle. Es bietet außerdem viele Möglichkeiten zur Alarmierung per Mail, SMS oder über andere Benachrichtigungskanäle bei kritischen Zuständen oder festgelegten Ereignissen, so dass eine relativ schnelle Erkennung und Behebung der Probleme möglich ist. Auf der Software-Seite wird dies durch die Erzeugung von verschiedenen Objekten gelöst. So werden durch Konfigurationsdateien Service- oder Host-Objekte angelegt. Die Abfrage der verschiedenen Objekte führt Icinga nicht selbst durch. Es sorgt lediglich für die Initialisierung eines Plugins, welches die eigentlichen Aufgaben zum Sammeln der Daten übernimmt.
Hauptbestandteile eines Icinga-Systems sind ein Monitoring-Core, eine Icinga Data Out Database (IDODB) sowie die klassische Web-Oberfläche. Der Monitoring-Core ist dabei für das Management des gesamten Systems verantwortlich; hier werden Abfragen initialisiert und deren Ergebnisse ausgewertet. Mittels IDODB werden diese Ergebnisse an eine externe Datenbank ausgelagert, damit diese beispielsweise durch verschiedene Web-Module wie Icinga Web abgefragt werden können. Alternativ ist die Anzeige der Daten auch durch die klassische Web-Oberfläche möglich. Diese erhält die Daten direkt vom Core und liest sie nicht aus der Datenbank aus.

Icinga 1 lieferte einige Verbesserungen im Vergleich zum ursprünglichen Nagios-Code. So konnten diverse Probleme gelöst und neue Features eingebunden werden. Dennoch gab es Schwierigkeiten, die mit dem an Nagios angelehnten Code nicht gelöst werden konnten. Unter anderem war die Skalierbarkeit in großen Monitoring-Umgebungen nur bedingt gegeben. Außerdem bot der Quellcode keine zufriedenstellende Qualität, da dieser beispielsweise nicht erlaubte, Änderungen zu implementieren ohne die Funktionalität der Addons zu beeinflussen.
All dies führte dazu, Icinga 2 von Grund auf neu zu entwickeln, wenngleich auch viele Einflüsse und Erfahrungen bezüglich der Entwicklung von Icinga 1 mit eingebracht wurden. Aufgrund dessen legten die Entwickler auch großen Wert auf eine Abwärtskompatibilität zu Nagios und Icinga 1. Alle in Nagios und Icinga verwendeten Plugins, Addons und Datenbankschemata können deshalb auch in Icinga 2 wiederverwendet werden. Offiziell wurde Icinga 2 am 16.06.2014 veröffentlicht und befindet sich seitdem in stetiger Weiterentwicklung.
Icinga 2 sollte mehr als eine einfache Monitoring-Instanz sein und eine Gesamtlösung für ein Monitoring mit einer Anwendung bieten. So wurden zusätzlich zum neuen Syntax neue Funktionalitäten eingeführt, die bisher nur von externen Programmen bewältigt werden konnten. Insgesamt wurde auch viel Wert darauf gelegt, ein modulares Monitoring-System bereitzustellen, was auch in einer einfacheren Handhabung für den Nutzer resultieren sollte.

2.2 Aktuelle Systemstruktur

In den nachfolgenden Kapiteln wird eine Beleuchtung der aktuellen Struktur des Telekom-Netzes erfolgen. Es werden außerdem Prinzipien erläutert, die in diesem Zusammenhang eine Rolle spielen. So wird auch eine Darstellung des bisherigen Systems stattfinden.

2.2.1 Netzstruktur

Das Netz der Telekom setzt sich aus einer Vielzahl an Betriebs- und Managementnetzen zusammen. Eine Übersicht der Gesamtstruktur ist der Abbildung 1: „Netze der Telekom“ zu entnehmen.

[Gesperrter Absatz]

2.2.2 Worker-Konzept

Alle durch Icinga überwachten Server empfangen die Anfragen für Checks nicht direkt vom Icinga-Core. Zwischen den abgefragten Management-Objekten und Icinga befinden sich verschiedene sogenannte Worker. Diese sind in der Telekom-Infrastruktur mittels Gearman implementiert. Gearman ist ein Programm, durch welches Aufgaben weitergegeben oder verteilt werden können. Die generelle Funktionsweise in Zusammenhang mit der Icinga-Struktur ist in Abbildung 2: „Gearman-Struktur“ zu erkennen:

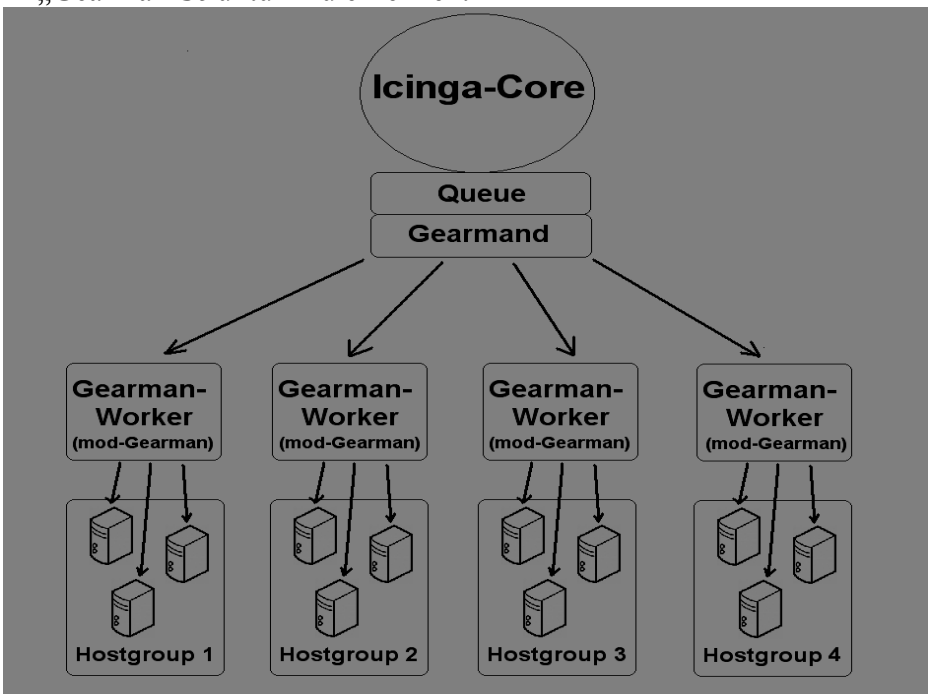


Abbildung 2: Gearman-Struktur

Der Icinga-Core ist in der Grundkonfiguration für die Ausführung von Checks verantwortlich. Mit der dargestellten Konstellation, welche in analoger Weise auch in der Telekom-Infrastruktur wiederzufinden ist, ist es möglich die eigentliche Ausführung der Checks durch die Gearman-Worker zu vollziehen. Auf den jeweiligen Maschinen befindet sich dafür eine mod-Gearman-Installation. Damit diese Aufgabenweiterleitung überhaupt möglich ist, schreibt der Icinga-Core die auszuführenden Aufgaben in eine Queue[4]. Ein Gearman-Daemon (Gearmand) kann aus dieser Queue die Check-Anforderungen entnehmen und dem entsprechenden Worker zuweisen. Die Zuweisung erfolgt anhand der jeweiligen Hostgroup. Der Gearman-Worker kann dann alle der Hostgroup zugeordneten Service-Abfragen bei den einzelnen Hosts durchführen. Die Ergebnisse der Abfragen leiten alle Worker an den Gearman-Daemon[5] zurück. Dieser wiederum schreibt die erhaltenen Ergebnisse in eine Queue, aus welcher der Icinga-Core diese entnehmen und verarbeiten kann.
Durch diese Infrastruktur ist es zum einen möglich, den Icinga-Core zu entlasten und somit mehr Checks durchführen zu können. Zum anderen befinden sich, wie in Abbildung 8 ersichtlich zwischen den jeweiligen Netzsegmenten, die abgefragt werden müssen, Firewalls. Würde man die Hosts einzeln abfragen lassen, bräuchte man etliche Firewallfreischaltungen. Durch diese Konfiguration werden lediglich eine geringe Anzahl an Freischaltungen zu den jeweiligen Workern benötigt, wodurch der Konfigurationsaufwand für die Firewall sowie der Traffic minimiert werden können.

[4]Queue: Warteschlangen-Datenstruktur zur Zwischenspeicherung von Objekten
[5]Daemon: im Hintergrund ablaufender Prozess bei Unix-Systemen; stellt Dienste bereit

2.2.3 Plugins

Im Netz der Telekom müssen eine Vielzahl von Komponenten überwacht werden. Dabei sind abhängig von der Komponente verschiedenste Eigenschaften abzufragen. Damit dies gewährleistet werden kann, setzt die Telekom unterschiedliche Plugins ein. Plugins sind in diesem Zusammenhang die Programme, die die eigentlichen Berechnungen bzw. das Ermitteln der Ergebnisse durchführen. Icinga selbst initiiert die Plugins nur und stellt die Ergebnisse der Pluginausgabe dar. Die Ausgabe setzt sich dabei immer aus einem Returncode und den jeweils ermittelten Werten zusammen. Dabei können grundsätzlich vier verschiedene Returncodes entstehen:
„0“ bedeutet OK; der gewünschte Zustand des Systems oder der Eigenschaft. „1“ steht für WARNING und signalisiert dem Anwender, dass die jeweilige Ressource eventuell bald einen kritischen Zustand erreicht. Der Returncode „2“ meldet, dass die Funktionalität nicht mehr in dem gewünschten Ausmaß gewährleistet werden kann, der Zustand CRITICAL ist erreicht. Der Rückgabewert „3“ bedeutet UNKNOWN und signalisiert, dass die Antwort nicht interpretiert werden kann.
Mittels übergegebener Parameter können dem Plugin Schwellwerte für die verschiedenen Zustände zugeordnet werden. Durch weitere Parameter können bei vielen Plugins noch zusätzliche Informationen für eine genauere Einschätzung des jeweiligen Status ausgegeben werden.

Die im aktuell aktiven Icinga 1-System für Abfragen verwendeten Plugins sind in Tabelle 1: „Check-Plugins“ abgebildet.

Plugin	Beschreibung
check_by_ssh	Remote-Überwachung per SSH (Aufbau SSH-Verbindung; Ausführen auf dem Host gespeicherter Plugins)
check_disk	Prüft Festplattenspeicherplatz
check_drbd	DRBD dient zur Echtzeitspiegelung in einem Cluster; Überwacht wird der Status des Dienstes, des Clusters und Prüfung auf Datenunterschiede
check_ftp	Portüberwachung (Port 21)
check_http	Portüberwachung (Port 80)
check_hpasm	Überwacht Serverhardware der HP Proliant Maschinen
check_juniperalarm.sh	Prüft Juniper-Firewalls auf Hardwarealarme
check_load	Überwacht CPU-Auslastung
check_netapp-1.6	Prüft Netapp auf Hardwarealarme
check_nrpe	Mit zusätzlichem Parameter „-c“ werden durch den NRPE-Daemon auf dem Remote-Host lokale Checks ausgeführt (alle auf Host gespeicherten Abfragen möglich)
check_nt	Abfragen verschiedener Systemwerte auf Windows-Rechnern
check_ntp_peer	NTP = Network Time Protocol; überwacht Unterschied zwischen Server- und Clientzeit
check_ping	Wertet Ping-Antwortzeit aus
check_procs	Prüft Anzahl laufender Prozesse
check_snmp_data_multi_oid.sh	Wertet gesammelte SNMP-Get-Anfragen aus
check_snmp_load.pl	CPU-Auslastung über SNMP abfragen
check_snmp_mem.pl	Speicherauslastung über SNMP abfragen
check_snmp_storage.pl	Storageüberwachung per SNMP
check_ssh	Portüberwachung (Port 22)
check_swap	Überwacht Auslastung des Swap-Speichers
check_tcp	Überwacht zu definierende Ports
check_users	Fragt Anzahl angemeldeter User ab

Tabelle 1: Check-Plugins

All diese Plugins sollen auch nach einer Migration weiterhin genutzt werden können. In dieser Tabelle sind auch sogenannte Remote-Plugins enthalten wie „check_nrpe“. Derartige Plugins ermöglichen die Ausführung von lokalen Plugins auf einem entfernten System. Da Icinga 2 laut Entwicklerinformationen hierfür eigene Funktionalitäten beinhaltet, wird im weiteren Verlauf dieser Arbeit auch eine Untersuchung der Möglichkeiten zur Abfrage entfernter Systeme erfolgen, um auf eventuelle Optimierungen des aktuellen Systems hinzuweisen.

2.2.4 Anschaltkonzept

Das aktuelle System wurde mit Hilfe von virtualisierten Servern aufgesetzt. Hierfür wurde folgende Hardware verwendet, welche mitsamt der Verkabelung auch in Abbildung 3: „Anschaltkonzept“ ersichtlich ist:

- 2 Cisco Catalyst 4500-X Series Switches (Redundante 10 GBit Storage Switches)
- 2 HP ProLiant DL380 Gen8 Server (2 Sockel, 2,9 GHz, 256GB RAM, 8 physikalische Cores)
- 1 Netapp FAS2240 Storage-System mit 24x 600GB SAS-Festplatten
- 1 Netapp FAS2220 mit 12x 2TB SATA-Festplatten
- 2 Cisco Catalyst 4510R-E Switches (TMDC Switches).

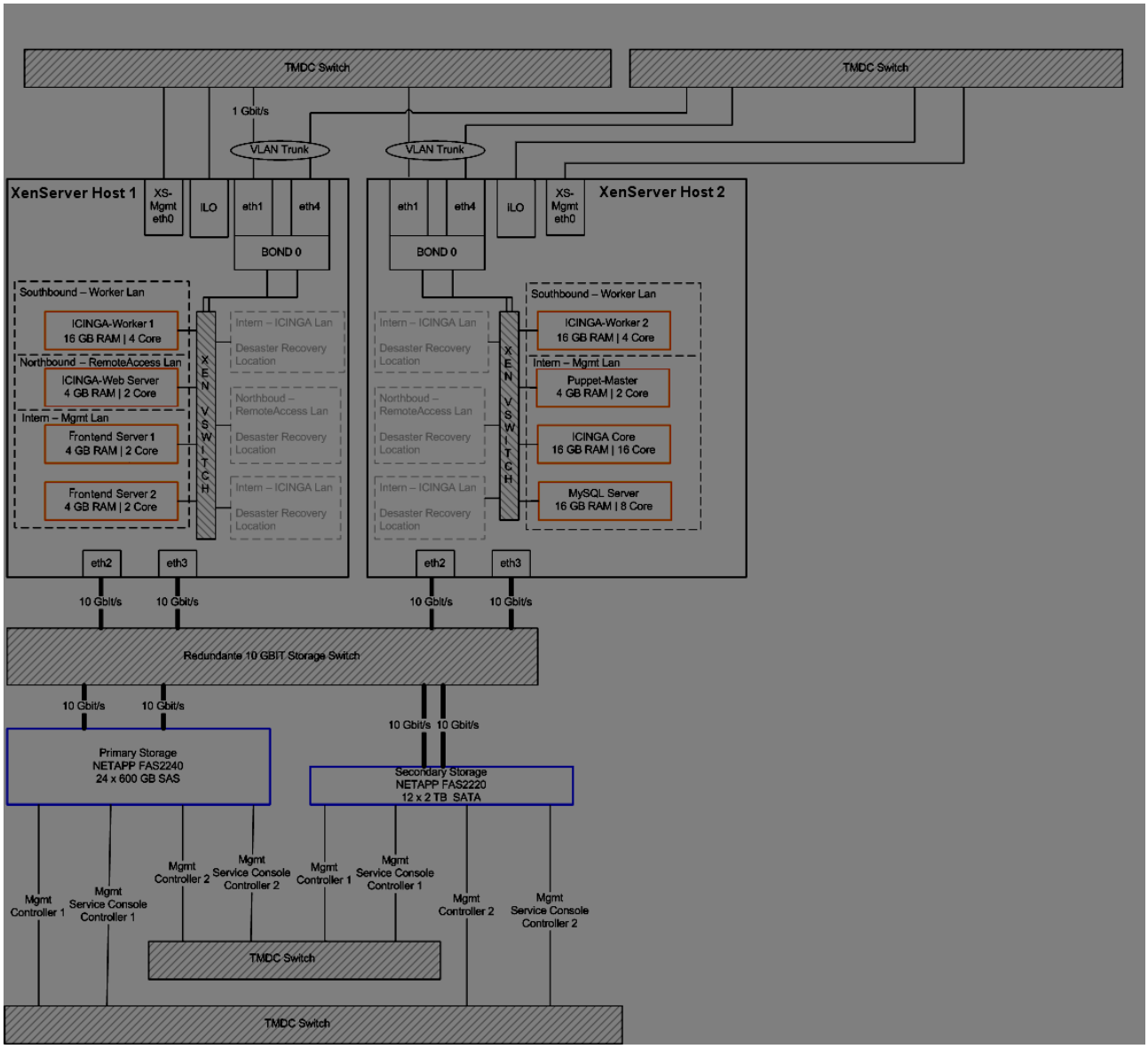


Abbildung 3: Anschaltkonzept

Die oben dargestellten TMDC Switches wurden aus Übersichtlichkeitsgründen unter den Netapp Storages noch einmal dargestellt. Die beiden Server sind über die Ethernet 0-, 1- und 4-Schnittstelle sowie das iLO[6]-Interface an die TMDC Switches angeschlossen, damit eine redundante Anbindung na die TMDC gegeben ist und die darin enthaltenen Netzelemente abgefragt werden können. Über

[6]ILO: integrated Lights Out – Management-System zur Administration und Fernwartung von Servern

die Ethernet 2- und 3-Schnittstelle sind die Server an die beiden Storage Switches angeschlossen, damit eine schnelle Verbindung zu den Netapp-Systemen besteht. Dies ist notwendig, damit die entsprechende Hochverfügbarkeit gewährleistet werden kann. Diese Thematik wird im Abschnitt „Hochverfügbarkeit“ näher erläutert.
Auf den beiden physikalischen Servern werden mit Hilfe der Virtualisierungsplattform XenServer diverse virtuelle Hosts bzw. Server erstellt. So laufen auf den Servern virtuelle Maschinen, die verschiedene Aufgaben bewältigen sollen. Es gibt zwei Worker-Maschinen, die aus der Icinga-Queue die für sie zur Ausführung abgelegten Abfragen entnehmen und an die jeweiligen Hosts weiterleiten, um die Ergebnisse letztendlich wieder in der Queue zu speichern, um somit die Last vom Icinga Core zu verringern, welcher eine weitere virtuelle Maschine darstellt. Getrennt vom Core werden auf zwei weiteren Maschinen der Icinga-Web-Server und die für Icinga zuständige SQL-Datenbank realisiert. Außerdem wurde ein Puppet-Master virtualisiert, welcher für die automatisierte Einspielung von Konfigurationen auf Hosts verantwortlich ist. Zudem gibt es noch zwei virtualisierte Frontend-Server. Diese haben die Aufgabe der Proxyweiterleitung auf die jeweilig angeforderte Anwendung.
Die Konfiguration des physikalischen Servers ist über die iLO-Schnittstelle vorgesehen. Das für das XenServer-Management zuständige Interface ist die Ethernet 0-Schnittstelle. Die beiden anderen mit den TMDC-Switches verbundenen Interfaces sind tagged Trunks[7]. Hiermit ist ein Zugriff auf die verschiedenen VLANs Southbound – Worker LAN, Northbound RemoteAccess LAN und das Intern Management LAN möglich. Über das Southbound Worker LAN wird die Verbindung zu den abzufragenden Netzelementen realisiert. Das Northbound RemoteAccess LAN dient als Schnittstelle zum internen Telekom-Netz und das Intern Management LAN für die Konfiguration des Icinga-Systems.
Gespeichert werden die virtuellen Maschinen auf den beiden Netapp Storages, durch welche auch eine gewisse Ausfallsicherheit gegeben ist. Sollte einer der Server oder eines der Netapp-Geräte ausfallen, erfolgt ein automatischer Schwenk der virtuellen Maschinen auf das jeweilige andere Gerät.

[7]tagged Trunks – Ports, die mehreren VLANs zugeordnet werden können anhand der Kennzeichnung (Tag) der ankommenden Pakete

2.3 Präzisierung der Zielstellung

Die Struktur des Telekom-Netzes bedingt einige Funktionalitäten, die für ein angemessenes Monitoring unverzichtbar sind. So ist das bereits erwähnte Worker-Konzept ein elementarer Bestandteil, der unter allen Umständen auch nach einer Migration gewährleistet sein muss. Es gilt deshalb zu untersuchen, welche Strukturen für ein verteiltes Monitoring genutzt werden können, ob eventuelle Verbesserungen durch die bei Icinga 2 neu eingeführten Funktionalitäten möglich sind und wie die Nutzung einer verteilten Monitoring-Struktur bestmöglich geschehen kann.
Viele der abgefragten Netzelemente müssen auch auf lokale Ressourcen geprüft werden. Dies kann nicht direkt durch eine externe Instanz geschehen. Eine Monitoring-Instanz kann hierfür nur Abfragen initialisieren, die durch ein Remote-Plugin dann auf dem jeweiligen Client durchgeführt werden. Diese Abfragen werden im momentanen System durch verschiedenste Remote-Plugins durchgeführt. Hier soll geprüft werden, inwiefern diese notwendig sind oder nicht zugunsten einer homogeneren Struktur ersetzt werden können. Dabei spielt der Aufwand, den eine solche Umstellung mit sich bringen würde, eine große Rolle und soll dementsprechend analysiert und abgeschätzt werden. Weiterhin soll anhand dessen auch beurteilt werden, wie eine Migration im günstigsten Fall geschehen kann. Dafür wird es nötig sein, die Konfigurationen des aktuellen Systems sowie die bei Icinga 2 verwendete Konfiguration zu untersuchen.
Ein großes Augenmerk besteht für den Auftraggeber aufgrund der aktuellen Performance-Probleme natürlich auch in der Analyse der Leistung des Systems. Dabei sollen Betrachtungen bezüglich der Dimensionierung des Systems geschehen mit Bezug zu der dadurch erreichbaren Leistung. Es gilt insbesondere eine Untersuchung bei Last-Szenarien durchzuführen, damit eine Beurteilung für das Verhalten bei weiterhin wachsendem System erfolgen kann.
In der Vergangenheit wurden zudem verschiedene Hochverfügbarkeitslösungen für das Monitoring genutzt, die jedoch auch gewisse Schwierigkeiten bereiteten. Icinga 2 bietet ein integriertes Cluster-Feature. Deshalb soll eine Untersuchung der verschiedenen Möglichkeiten durchgeführt werden. Es gilt zu analysieren, ob eine Ablösung der momentanen Hochverfügbarkeitsvariante durch das Icinga 2-Cluster möglich und sinnvoll wäre.
Letztendlich besteht das Ziel anhand all dieser Punkte darin eine Einschätzung zu geben, ob Icinga 2 für die Strukturen im Telekom-Netz geeignet ist und ob eine Migration erfolgen soll.

3 Analyse

In diesem Teil werden alle für eine Migration gemäß der vorgegebenen Ziele notwendigen Untersuchungen durchgeführt. Hierbei wird detailliert auf die jeweiligen Teilgebiete eingegangen und am Ende des jeweiligen Bereiches ein entsprechendes Fazit gezogen.

3.1 Vergleich der Konfigurationen

Da Icinga 2 von grundauf neu entwickelt wurde, ergeben sich im Syntax einige Änderungen. Aufgrunddessen bieten sich teilweise neue Optionen und flexiblere Strukturen. Um die Migration beurteilen zu können, ist es von großer Bedeutung die Unterschiede zu untersuchen und die Vor- und Nachteile der neuen Konfigurationen herauszustellen.

3.1.1 Allgemeine Konfiguration

Generell gibt es in Icinga zwei Arten von Konfigurationen: Objektbasiert und auf dem Key-Value-Prinzip aufbauend. Diese können sowohl in der Hauptkonfigurationsdatei *icinga.cfg* bzw. *icinga2.cfg* vorkommen, als auch in anderen Konfigurationsdateien, die durch die Hauptkonfigurationsdatei eingebunden wurden. Die Inklusion von anderen Dateien funktioniert bei Icinga 1 über die Befehle „*cfg_dir*“, welcher rekursiv alle in einem Ordner befindlichen Konfigurationsdateien einbindet, und „*cfg_file*“, der eine bestimmte Konfigurationsdatei integriert. Hier dürfen nur absolute Dateipfade[8] verwendet werden. Bei Icinga 2 hingegen wird die Einbindung von Dateien durch „include“ und die Einbindung von Ordnern durch „*include_recursive*“ gewährleistet. Hier können auch Platzhalter verwendet werden, um beispielsweise mehrere Dateien zu integrieren, wodurch Programmzeilen gespart und somit auch mehr Übersichtlichkeit bewahrt werden kann:
include "conf.d/*.conf"
Wie an diesem Beispiel erkennbar werden hier relative Dateipfade9 verwendet.
In der Hauptkonfigurationsdatei befinden sich bei Icinga 1 Einbindungen von Konfigurationsdateien, Verweise auf Log-Dateien, Cache-Dateien und andere Dateien für die Ablage von programmrelevanten Daten. Des Weiteren werden verschiedene Key-Value-Paare definiert, wie beispielsweise für die Aktivierung des Loggings, Festlegung des Icinga-Nutzers oder verschiedene Optionen des Loggings. Diverse Variablen, bzw. Makros, wie die Uservariablen für die Plugin-Verzeichnisse, werden zusätzlich in der *resource.cfg* definiert.
Icinga 2 hingegen legt in der Hauptkonfigurationsdatei lediglich die Integrationspfade für die anderen Konfigurationsdateien fest und bindet auch die Icinga Template Library (ITL)[10] ein. Der Icinga-User ist über die *init.conf* definiert, welche noch vor der Hauptkonfigurationsdatei geladen wird. Weiterhin werden Variablen für Plugin-Verzeichnisse oder Festlegung des Node-Namens und Zonen-Namens festgelegt. Dies geschieht in der *constants.conf*. Zudem wird hier auch das Ticketsalt für die Konfiguration der Zonen abgelegt. Insgesamt sind diese Dateien sehr klein gehalten, was zu mehr Übersichtlichkeit und einer leichteren Fehleranalyse bei Problemen führt.

[8]absoluter Dateipfad – kompletter Zielpfad für eine Datei
[9]relativer Dateipfad – Zielpfad in Abhänigkeit vom Verzeichnis (in diesem Fall Icinga-Verzeichnis)
[10]Icinga Template Library – Sammlung von definierten Vorlagen für Objekte (z.B. Commands, Notifications)

3.1.2 Host-Objekte

Icinga 1.x:

```
define host{
        use             generic_host
        host_name       linux-host1
        alias           li1
        display_name    linux1
        address         10.0.0.1
        hostgroups      linux-servers
}
```

Icinga 2.x:

```
object Host "linux-host1"{
        import "generic-host"
        display_name = "linux1"
        address ="10.0.0.1"
        vars.os = "Linux"
        if (!host.vars.users_cgreater) {
        vars.users_cgreater = 10}
}
```

Bei der Definition der Host-Objekte ergaben sich verhältnismäßig wenig bedeutsame Änderungen. Geringfügige Syntax-Änderungen wurden eingeführt; so ist beispielsweise der Host-Name direkt in der Definition angegeben und nicht bei den Objekteigenschaften. Außerdem geschehen die Zuweisungen der Objekteigenschaften mittels „=“ und es werden andere Bezeichnungen für diverse Definitionen genutzt:

Schlüsselwort in Icinga 1.x	Schlüsselwort in Icinga 2.x
host	Host
define	object
use	import
alias / display_name	display_name

Tabelle 2: Gegenüberstellung der Schlüsselwörter

Das Schlüsselwort „*alias*“ kann bei Icinga 1 zusammen mit dem „*display_name*“ genutzt werden, um eine kurze und ausführliche Beschreibung des Objekts zu ermöglichen. Da das gleiche Ziel auch nur mit der jeweiligen Objektbezeichnung und dem Attribut „*display-name*“ erreicht werden kann, wurde in Icinga 2 auf das Schlüsselwort „alias“ verzichtet.
Bei Icinga 2 ist darauf zu achten, die zuzuweisenden Eigenschaften in Anführungszeichen zu schreiben, da Icinga sonst von vorher definierten Variablen ausgeht und deren Werte der jeweiligen Eigenschaft zuordnen würde.
Durch diese Möglichkeit der Variablennutzung können Eigenschaften relativ dynamisch zugewiesen werden, was in diversen Szenarien eine sehr vorteilhafte Option sein kann.
Die bedeutsamste Änderung stellt die Einführung von benutzerdefinierten Variablen dar. Mit dem „*vars.*“-Schlüsselwort können Variablen für verschiedenste Anwendungsfälle erstellt werden. Große Vorteile bieten sich dabei in der Anpassung von hostabhängigen Check-Parametern. Genauere Erläuterungen sind im Abschnitt 3.1.5 „Service-Objekte“ beschrieben.
Zudem ist es auch möglich, Bedingungen zu prüfen. Im gewählten Beispiel wird anhand der if-Kondition geprüft, ob eine Variable „users_cgreater“ einen Wert besitzt. Ist dies nicht der Fall, wird die Variable mit dem Wert „10“ befüllt. Derartige Konditionen können verschiedene Szenarien

abfangen und somit auch insgesamt eine dynamischere Konfiguration gewährleisten.

3.1.3 Hostgroup-Objekte

Icinga 1.x:

```
define hostgroup {
        hostgroup_name          linux-servers
        members                 linux-host1, linux-host2
}

define hostgroup {
        hostgroup_name          servers
        members                 test-host
        hostgroup_members       linux-servers, windows-servers
}
```

Icinga 2.x:

```
object HostGroup "linux-servers" {
        assign where host.name in [ "linux-host1", "linux-host2" ]
}

object HostGroup "servers" {
        groups = [ "linux-servers" ]
        assign where host.name == "test-host"
}
```

Wie auch bei den Host-Objekten zu sehen, ersetzt „*object*“ das Schlüsselwort „*define*“. Diese Festlegung wurde für alle in Icinga 2 verwendeten Objekte erstellt. Die weitere Definition der Hostgruppen ist im Gegensatz zu den Host-Objekten nicht mehr so stark an Icinga 1.x orientiert. Statt einer starren Zuordnung der Hosts zu den Hostgruppen, geschieht diese Zuweisung mit Hilfe von Assign-Regeln. Sie agieren als eine Art Filter auf verschiedene Objekt-Attribute. Bei einem Host können dies beispielsweise die Adresse, der Name oder benutzerdefinierte Variablen sein.
Diese Regeln können auf den kompletten Wert der jeweiligen Eigenschaft oder mit Hilfe von Regular Expressions11 (RegEx) auf bestimmte Teilübereinstimmungen angewendet werden. Des Weiteren ist auch eine Verkettung von mehreren Regeln durch die Nutzung von logischen UND- und ODER-Verknüpfungen (&& entspricht UND; || entspricht ODER) möglich, sowie die zusätzliche Anwendung von ignore-Statements, um bestimmte Hosts auszuschließen.
Möchte man nun Hosts zu einer Hostgruppe hinzufügen, ist durch diese Zuweisungen eine dynamische Struktur geboten, welche verschiedene Implementationen für das gleiche Ziel erlaubt. Ein großer Vorteil ergibt sich dadurch, dass Hosts nicht unbedingt manuell hinzugefügt werden müssen, sondern dies durch gut gewählte Regeln automatisch geschehen kann.
Das oben gewählte Beispiel ist in dieser Hinsicht noch an Icinga 1.x angelehnt. Hier werden die Filter auf den Hostnamen gesetzt, welcher sich in dem benutzerdefinierten Array *["linux-host1", "linux-host2"]* befinden muss, damit der entsprechende Host der Hostgruppe „*linux-servers*“ hinzugefügt werden kann. Dadurch muss jeder neue Host auch in dieses Array eingetragen werden.
Dynamisch wäre diese Zuordnung unter anderem auf diese Weise realisierbar:

```
object HostGroup "linux-servers" {
        assign where match ("linux*", host.name) || (host.vars.os == "Linux")
        ignore where host.name == "linux-host3"
}
```

[11]Regular Expressions – Reguläre Ausdrücke; Zeichenkette mit syntaktischen Regeln zur Filterung von bestimmten Ausdrücken

Bei dieser Definition der Regeln werden der Hostgruppe „*linux-servers*“ automatisch alle Hosts zugeordnet, welche mit dem wort „*linux*“ beginnen. Zusätzlich werden alle Hosts, für die eine Variable „*os*“ mit dem Wert „*Linux*“ gesetzt wurde, ebenfalls zu dieser Gruppe hinzugefügt. Ausgeschlossen von dieser Zuweisung wird der Host mit dem Namen „*linux-host3*“.

3.1.4 Command-Objekte

Icinga 1.x:

```
define command {
        command_name                    check_users
        command_line                    $USER1$/check_users -w $ARG1$ -c $ARG2$
}
```

Icinga 2.x:

```
object CheckCommand "users" {
        import "plugin-check-command"
        command = [ PluginDir + "/check_users" ]
        arguments = {
                "-w" = "$users_wgreater$"
                "-c" = "$users_cgreater$"
        }

        vars.users_wgreater = 20
        vars.users_cgreater = 50
}
```

Command- oder Befehls-Objekte sind in Icinga 1 durch wenige Zeilen festgelegt. So werden in dem Objekt lediglich die Attribute „*command_name*“ und „*command_line*“ mit entsprechenden Werten befüllt. Über den „*command_name*“ kann der Befehl eindeutig identifiziert und in der Service-Definition dementsprechend zugeordnet werden. Die „*command_line*“ beinhaltet den Pfad zum eigentlich auszuführenden Plugin sowie die zur Anwendung kommenden Parameter des Befehls. Dabei setzt sich der Pfad in diesem Fall standardmäßig aus dem „*$USER1$*“-Makro, welches dem Pfad zu den Plugins entspricht, und dem Namen des Plugins zusammen. Die zum Befehl gehörigen Parameter werden danach angehängt und vorerst durch die *$ARGn$*-Makros (n ist eine natürliche Zahl, von 1 aufwärts die Argumente zählend) bestimmt, welche später durch die in den Service-Definitionen festgelegten Parameter überschrieben werden können.
In Icinga 2 sind die Command-Objekte durch einige Zeilen mehr definiert. Der Name und damit die Identifikation des Befehls wird wie bei allen Objekten durch „*object*“ und den jeweiligen Typ festgelegt. Generell werden bei Icinga 2 die gleichen Objekt-Attribute verwendet; lediglich der Syntax ist unterschiedlich und ein Import des Templates „*plugin-check-command*“ wird benötigt. Der erste Teil der „*command_line*“, welcher die Pfadangabe des jeweiligen Plugins bestimmt, ist im Attribut „*command*“ wiederzufinden. Dieses setzt sich analog zu Icinga 1 aus einer Variablen für den Pfad der Plugins („*PluginDir*“) und dem Dateinamen des Plugins zusammen. Die Befehlsparameter werden hier nicht direkt an den Pfad angehängt, sondern über ein weiteres Attribut („*arguments*“) definiert. Dort werden den entsprechenden Parametern Variablen zugeordnet. Diese können allgemeingültig für den Befehl in demselbigen befüllt werden und auch hostspezifisch bei den Host-Attributen geändert werden. Bei dem gewählten Beispiel würde der Zustand „Warning“ bei mehr als 20 Usern und „Critical“ bei mehr als 50 eingeloggten Nutzern eintreten. Durch diese Konstruktion der Eigenschaften ergeben sich große Vorteile in Bezug auf die generische Nutzung von Services.

Der nächste Abschnitt beschäftigt sich mit den Service-Objekten und wird diese Vorteile genauer beleuchten.

3.1.5 Service-Objekte

Icinga 1.x:

```
define service {
        use                     generic-service
        service_description     Users
        host_name               linux-host1
#       hostgroup_name          linux-servers
        check_command           check_users!20!50
        check_interval          5
        retry_interval          1
}
```

Icinga 2.x:

```
object Service "Users" {
        import "generic-service"
        host_name = "linux-host1"
#       hostgroup_name = "linux-servers"
        check_command = "users"
        check_interval = 5m
        retry_interval = 1m
}
```

Die Definition der Service-Objekte ist vom Aufbau denen der anderen Objekte gleichzusetzen. Es gibt verschiedene Attribute, denen die jeweiligen Werte zugewiesen werden. Obligatorisch sind bei den Services die Angabe des Namens, welcher bei Icinga 1 unter dem Schlüsselwort „*service_description*“ aufgeführt und in Icinga 2 bereits bei der Objektdefinition bestimmt ist, sowie das „*check_command*“ und die Zuordnung zu einem oder mehreren Hosts. Zusätzlich können unter anderem Templates, wie der „*generic-service*“ mithilfe von „*use*“ bzw. „*import*“ bei Icinga 2 hinzugefügt werden. Diese Templates enthalten weitere Attribute. Beispielsweise werden im „*generic-service*“ standardmäßig „*max_check_attempts*“, „*retry_interval*“ und „*check_interval*“ festgelegt. Diese können auch ohne Template bei den Services mit entsprechenden Werten versehen werden. Hier wurde eine kleine Änderung vorgenommen bei der Entwicklung von Icinga 2. So muss hinter alle Zeitintervalle die Angabe der Einheit hinzugefügt werden (s = Sekunde, m = Minute, h = Stunde). Bei Icinga 1 gab es für verschiedene Intervalle verschiedene Zeiteinheiten, die nicht explizit erwähnt wurden. Durch diese Änderung ist es für den Benutzer einfacher nachzuvollziehen und verhindert falsche Intervalle.
Die Zuordnung der Services zu Hosts geschieht bei Icinga 1 über die Attribute „*host_name*“ oder „*hostgroup_name*“. Dadurch müssen die Hosts oder Hostgruppen direkt benannt werden. Dies ist auch bei Icinga 2 möglich. Dort besteht allerdings noch die Möglichkeit diese Zuweisung mit Hilfe von Apply-/Assign-Regeln zu lösen:

```
apply Service "Users" {
        import "generic-service"
        check_command = "users"
        assign where "linux-servers" in host.groups
}
```

Analog der Zuordnung von Hostgruppen kann dadurch auf unterschiedlichste Host-Attribute gefiltert werden. Dies gewährleistet eine flexible und leicht anpassbare Struktur.
Wie bereits im vorherigen Abschnitt bezüglich der Command-Objekte erwähnt, gibt es auch durch die geänderte Nutzung der Befehlsparameter weitere Vorteile, welche auch mehr Flexibilität bieten. So werden die in Icinga 1 im Command-Objekt definierten *$ARGn$*-Makros durch das „*check_command*“ des jeweiligen Service-Objekts befüllt. Die Parameter folgen dem eigentlichen Befehl getrennt durch Ausrufezeichen. Der erste Wert entspricht damit dem *$ARG1$*-Makro und der zweite Wert dem *$ARG2$*-Makro. In diesem Beispiel wird der Service letztendlich mit folgender Programmzeile ausgeführt:
[Pfad zu den Plugins]/check_users -w 20 -c 50
Das bedeutet allerdings, dass diese Werte nur im Service-Objekt festgelegt sind und nicht für einzelne Hosts bestimmt werden können. Sollte nun für einzelne Hosts eine Prüfung der Abfrage mit anderen Parametern stattfinden, weil diese beispielsweise weniger leistungsstarke Hardware besitzen, so müsste jeweils ein neuer Service mit angepassten Parametern erstellt werden.

In Icinga 1 würde dies folgendermaßen aussehen:

```
define service {
        use                     generic-service
        service_description     Users_low
        host_name               linux-host3
        check_command           check_users!10!20
}
```

Bei Icinga 2 können dem jeweiligen Service auch entsprechende Werte zugeordnet werden, welche die in dem Command-Objekt definierten Parameter überschreiben. Realisiert wird dies im aktuellen Beispiel durch die Zuweisung von Werten für die Parameter „*vars.users_wgreater*“ und „*vars.users_cgreater*“. Dies würde das gleiche Ergebnis wie die Konfiguration in Icinga 1 hervorrufen. Jedoch sind diese Variablen in Icinga 2 auch bei der Hostdefinition zuweisbar. Dadurch können spezielle Abfrageparameter je nach Leistung oder Bedarf des jeweiligen Hosts zugeordnet werden, ohne dass dafür ein komplett neues Service-Objekt erstellt werden muss:

```
object Host "linux-host3"{
        import "generic-host"
        display_name = "linux3"
        address ="10.0.0.3"
        vars.users_wgreater = 10
        vars.users_cgreater = 20
}
```

Diese Option ist als große Verbesserung einzuschätzen, da zum einen der Konfigurationsaufwand deutlich geringer ausfällt und zum anderen weniger Service-Objekte erstellt werden, was sich wiederum positiv auf die Gesamt-Performance des Systems äußert.
Dennoch besteht weiterhin Verbesserungspotential, da die im Host festgelegten Variablen von denen, die im Service definiert wurden, überschrieben werden. Es dürfen demnach im Service keine Werte festgelegt werden, wenn einzelne Hosts mit anderen Parametern abgefragt werden sollen. Es gibt also immer nur zwei Stellen, bei denen die Konfiguration eine Rolle spielt: entweder Command-Definition

und Service-Definition oder Command-Definition und Host-Definition. Es ist nicht möglich sowohl die Werte aus Service- und Host-Definition zu beachten. Dennoch ist diese Neuerung als deutlicher Mehrgewinn anzusehen, da mehr Granularität in der Konfiguration mit weniger Programmcode realisierbar ist.
Analog zu den Hostgruppen können auch Servicegruppen-Objekte erstellt werden. Der Syntax ist hierbei identisch; das Schlüsselwort „HostGroup“ wird jedoch durch „ServiceGroup“ ersetzt.

3.1.6 Contact-/User-Objekte

Icinga 1.x:

```
define contact{
        contact_name                    Icinga-User1
        use                             generic-user
        alias                           User1
        service_notification_options    c,f,u
        service_notification_commands   service-notify-by-email
        notification_interval           20
        email                           user1@test.de
}
```

Icinga 2.x:

```
object User "Icinga-User1" {
        import "generic-user"
        display_name = "User1"
        email = "user1@test.de"
        types = [ Problem, Recovery, FlappingStart, FlappingEnd ]
        states = [Critical, Unknown]
}
```

Contact- bzw. User-Objekte werden von Icinga genutzt um Benachrichtigungen bei definierten System- oder Servicezuständen zu versenden. Die Konfiguration ist vergleichbar mit denen der anderen Objekte. Hier ergaben sich bei Icinga 2 keine größeren Änderungen. Lediglich die bei den Host-Objekten bereits erwähnten Veränderungen bei den Schlüsselwörtern sowie die Veränderung der Typbezeichnung von „*contact*“ zu „*User*“ sind hier zu erkennen. Die „*notification_options*“ werden in Icinga 2 in „*types*“ und „states“ aufgeteilt. Genauere Erläuterungen diesbezüglich sind im nächsten Abschnitt ersichtlich.
Ähnlich den Host- und Servicegruppen gibt es auch Usergruppen. Hier können mehrere User zusammengefasst werden, um die Zuordnungen zu Notifications einfacher zu gestalten. Die Usergruppen-Objekte fordern den gleichen Syntax; lediglich das Schlüsselwort „HostGroup“ wird durch „UserGroup“ ersetzt.

3.1.7 Notifications und Eskalationen

Icinga 1.x:
Notifications sind Benachrichtigungen, die an festgelegte User auf verschiedenen Wegen gesendet werden, um diese über einen bestimmten Status eines Services oder eines Hosts zu informieren.
In Icinga 1 geschieht dies über die Zuweisung von einem Contact- oder einem Contactgroup-Objekt zu einem Host oder Service. Dabei wird dem jeweiligen Host oder Service ein Attribut „*contact_name*“ oder „*contact_groups*“ zugefügt. Dadurch besteht eine sehr starre Verbindung zwischen Host-/Service-Objekt und den Contact-Objekten.
Damit die eigentliche Notification ausgeführt werden kann, müssen im jeweiligen Contact-Objekt die „*notification_options*“ und das „*notification_command*“ zugewiesen sein. Diese können sich auf Hosts oder Services oder beide Möglichkeiten beziehen. Im Beispiel unseres Contact-Objekts bezieht sich die Notification auf einen Service. Die dort angegeben Optionen „*c, f, u*“ bewirken das Versenden der Benachrichtigungen bezüglich des Services bei Critical- oder Unknown-Zustand und bei Beginn oder Ende des Flatterns[12] des Services. Alle möglichen Optionen sowie deren Bedeutung können der folgenden Tabelle entnommen werden:

Option	Bedeutung
w	Benachrichtigungen werden bei WARNING-Zustand des Services versendet
c	Benachrichtigungen werden bei CRITICAL-Zustand des Services versendet
u	Benachrichtigungen werden bei UNKNOWN-Zustand des Services versendet
r	Benachrichtigungen werden bei erholtem OK-Zustand des Services versendet
f	Benachrichtigungen werden bei Flattern (Anfang und Ende) des Services versendet
s	Benachrichtigungen werden bei geplanter Ausfallzeit (Anfang und Ende) des Services versendet
n	Es werden keine Benachrichtigungen versendet (andere Optionen nicht beachtet)

Tabelle 3: Service-Notification-Options

Die entsprechenden Host-Benachrichtigungsoptionen sind aus der nachstehenden Tabelle ersichtlich:

Option	Bedeutung
d	Benachrichtigungen werden bei DOWN-Zustand des Hosts versendet
u	Benachrichtigungen werden bei UNREACHABLE-Zustand des Hosts versendet
r	Benachrichtigungen werden bei erholtem OK-Zustand des Hosts versendet
f	Benachrichtigungen werden bei Flattern (Anfang und Ende) des Hosts versendet
s	Benachrichtigungen werden bei geplanter Ausfallzeit (Anfang und Ende) des Hosts versendet
n	Es werden keine Benachrichtigungen versendet (andere Optionen nicht beachtet)

Tabelle 4: Host-Benachrichtigungsoptionen

Mit den „*notification_commands*“ wird der jeweils definierte Befehl zugewiesen. Dieser besteht, wie

[12]Flattern/Flapping – bezeichnet in einem relativ kleinen Intervall oft auftretende Zustands-Wechsel von Host oder Services

bei den Command-Objekten üblich, aus Name und Command-Line. In der Command-Line werden diverse Makros übergeben, um die Benachrichtigung zu versenden:

```
define command{
        command_name    service-notify-by-email
        command_line    /usr/bin/printf "%b" "Notification Type:
                        $NOTIFICATIONTYPE$\n\nService:
                        $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress:
                        $HOSTADDRESS$\nState:
                        $SERVICESTATE$\n\nDate/Time:
                        $DATETIME$\n\nAdditional Info:\n\n$OUTPUT$" |
                        /bin/mail -s "** $NOTIFICATIONTYPE$ alert -
                        $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$
                        **"$CONTACTEMAIL$
}
```

So wird der entsprechende Nutzer über die durch das Makro erfasste Mailadresse kontaktiert und erhält alle nötigen Informationen, die ebenfalls über die Makros ermittelt wurden (beispielsweise Service-Name, Zeit, Service-Status). Sollte ein Service von besonderer Bedeutung sein, bzw. eine weitere Benachrichtigungsart oder ein zusätzlicher Benachrichtigungsempfänger bei längerer Ausfallzeit erwünscht sein, können Service-Eskalationen eingesetzt werden:

```
define serviceescalation{
        host_name               linux-server1
        service_description     Users
        first_notification      3
        last_notification       8
        notification_interval   10
        contact_groups          admins
}
```

Hier wird anhand von „*first_notification*“, „*last_notification*“ und „*notification_interval*“ der Zeitraum festgelegt, in welchem die Eskalationsbenachrichtigungen versendet werden sollen. In diesem Beispiel tritt nach den ersten beiden Benachrichtigungen, welche direkt beim Service definiert sind (dort ist auch das „*notification_interval*“ festgelegt), die Service-Eskalation in Kraft. Das bedeutet, dass die vorher festgelegten Benachrichtigungen unterdrückt werden und stattdessen die in der Service-Eskalation festgelegten Notifications verschickt werden. Die nächsten Benachrichtigungen erfolgen im Abstand von 10 Minuten bis die achte Mitteilung gesendet wurde. Danach werden weitere Benachrichtigungen wieder anhand der ursprünglich definierten Notification-Intervalle an die zugewiesenen User geleitet.

Icinga 2.x:
Bei Icinga 2 werden für Benachrichtigungen extra Objekte angelegt. Wie bei anderen Objekten ist es auch hier möglich eine starre Zuordnung ähnlich Icinga 1 herzustellen oder aber eine Lösung mit Assign-Regeln zu realisieren.

```
object Notification "users-notification" {
        service_name = "Users"
        import "mail-service-notification"
        users = [ "Icinga-User1", "Icinga-User2" ]
        interval = 20m
        types = [ Problem, Recovery, FlappingStart, FlappingEnd ]
        states = [Critical, Unknown]
}
```

Bei der starren Zuordnung muss der Service- oder Host-Name als Attribut definiert werden. Die Konfiguration mit Hilfe der Apply-/Assign-Regeln lässt abermals eine dynamische Filterung auf benutzerdefinierte Variablen oder andere Attribute zu.

```
apply Notification "users-notification" to Service {
        import "mail-service-notification"
        users = [ "Icinga-User1", "Icinga-User2" ]
        types = [ Problem, Recovery, FlappingStart, FlappingEnd ]
        states = [Critical, Unknown]
        assign where service.name == "Users"
}
```

Wie bereits erwähnt, unterteilt Icinga 2 die in Icinga 1 verwendeten „*notification_options*“ in „*types*“ und „*states*“. Da hier spezielle Objekte für die Benachrichtigungen definiert werden, ist es nicht unbedingt notwendig, diese Optionen beim User-Objekt festzulegen. Eine Definition im Notification-Objekt bewirkt die gleiche Reaktion. Jedoch kann mit dieser doppelten Möglichkeit der Festlegung eine feinere Struktur bewirkt werden. So können damit allgemeine Vorgaben für den Service bestimmt und für einzelne Kontakte abweichende Konfigurationen angewendet werden.
Die Unterteilung der „*notification_options*“ ist in der folgenden Tabelle aufgeführt:

Icinga 1 - notification_option	Icinga 2 - state	Icinga 2 - type
w	WARNING	Problem
c	CRITICAL	Problem
u	UNKNOWN	Problem
d	DOWN	Problem
s	-	DowntimeStart / DowntimeEnd / DowntimeRemoved
r	OK (nach Erholung)	Recovery
f	-	FlappingStart / FlappingEnd
n	0	0
-	-	Custom

Tabelle 5: Vergleich Notifications

Diese Umgestaltung bietet vor allem eine bessere Nachvollziehbarkeit für den Nutzer, da durch die eindeutigen Bezeichnungen der Zweck der Variablen sofort ersichtlich ist und die Aufteilung in „*state*“ und „*type*“ eine günstigere Struktur bietet. Zudem wurde der „Custom-type“ eingeführt, durch welchen über externe Kommandos Benachrichtigungen gesendet werden können (beispielsweise aufgrund von Wartungsarbeiten).
Obligatorisch für die Definition der Notifications sind die über die Attribute „*users*“ oder „*user_group*“ zugeordneten Kontakte, sowie das „*command*“-Attribut. Zudem ist die Zuweisung

eines Services oder Hosts notwendig. Das „*command*“-Attribut wird in unserem Beispiel durch den import der „*mail-service-notification*“ bereitgestellt:

```
object NotificationCommand "mail-service-notification" {
        import "plugin-notification-command"
        command = [ SysconfDir + "/icinga2/scripts/mail-service-notification.sh" ]
        env = {
                NOTIFICATIONTYPE = "$notification.type$"
                SERVICEDESC = "$service.name$"
                HOSTALIAS = "$host.display_name$"
                HOSTADDRESS = "$address$"
                SERVICESTATE = "$service.state$"
                LONGDATETIME = "$icinga.long_date_time$"
                SERVICEOUTPUT = "$service.output$"
                HOSTDISPLAYNAME = "$host.display_name$"
                SERVICEDISPLAYNAME = "$service.display_name$"
                USEREMAIL = "$user.email$"
        }
}
```

Hier werden durch „*env*“ verschiedene Makros zu Umgebungsvariablen gesetzt. Diese werden durch das im „*command*“ aufgerufene Script „*mail-service-notification.sh*“ abgerufen und alle benötigten Informationen mit Hilfe des Scripts an die zugewiesenen User übertragen.
Benachrichtigungseskalationen sind in Icinga 2 keine gesonderten Objekte mehr. Sie werden als Notifications deklariert und die Eskalation findet über diverse Optionen statt:

```
apply Notification "email-escalation" to Service {
        import "mail-service-notification"
        interval = 10m
        user_groups = [ "admins" ]
        times = {
                begin = 1h
                end = 2h
        }
        assign where service.name == "Users"
}
```

Hier wird ein neues Intervall gesetzt für die Benachrichtigungen und ein Start- und Endzeitpunkt für diese definiert. Im Beispiel würden eine Stunde nach der ersten Benachrichtigung die Eskalationsnachrichten im Abstand von zehn Minuten gesendet werden. Die letzte Benachrichtigung für diese Eskalation erfolgt letztendlich zwei Stunden nach der ersten Notification.
Bei Icinga 1 wurden bei Eskalationen die ursprünglichen Notifications für den jeweiligen Zeitpunkt ausgesetzt. Dies hatte zur Folge, dass die in der anfänglich deklarierten Kontakte in der Eskalation nicht mehr benachrichtigt wurden (es sei denn sie wurden dort zusätzlich definiert). In Icinga 2 werden die zu Beginn festgelegten User weiterhin benachrichtigt. Die Eskalation ist eine zusätzliche Notification, weshalb bei dieser die Kontakte nicht doppelt definiert werden müssen.

3.1.8 Fazit

Insgesamt ergibt sich eine Konfiguration, die stark an die bisherige Konfiguration angelehnt ist. Die Neuerungen, wie die Apply-/Assign-Regeln sowie die neuen Notification-Objekte sind als sehr positiv einzuschätzen, da hierdurch eine gewisse Dynamik gegeben ist. So können Services oder andere Objekte aufgrund des Namens oder eines anderen Attributs automatisch zugeordnet werden, ohne dass eine Änderung des Services stattfinden muss. Da unterschiedliche Definitionen der Objekte möglich sind, so auch die Nutzung der starren Zuordnung von Objekten, ist ein Umstieg auf Icinga 2 für mit Icinga 1 vertrauten Nutzern als unkritisch zu beurteilen. Der ähnliche Syntax sollte eine schnelle Einarbeitung ermöglichen.

3.2 Migrationsarten

Da im Netz der Telekom eine Vielzahl von Hosts und Services überwacht werden müssen, kann eine manuelle Migration sehr viel Zeit in Anspruch nehmen. Von Seiten der Icinga 2-Entwickler wurde deshalb ein Migrations-Tool[13] für die Automatisierung der Migration zur Verfügung gestellt. Es ist als Tool für Icinga Web 2 gedacht und dient bis zur Veröffentlichung der finalen Version als eigenständiges Programm für die Konvertierung der Konfiguration. Allerdings gilt das Programm als weitestgehend unausgereift und die entstehende Konfiguration wird als unsauber bzw. ineffektiv angesehen.
Im Folgenden soll die Nutzbarkeit des Programms untersucht werden und eine Beurteilung der möglichen Anwendung für eine Migration der telekominternen Konfiguration geschehen.
Dem Programm wird der Pfad zur Icinga-Hauptkonfigurationsdatei *„icinga.cfg“* übergeben. Da in dieser Datei Pfade zu allen anderen Konfigurationsdateien enthalten sind, kann auf diese im Ablauf des Programmes zugegriffen werden.
Beim Start des Tools mit Übergabe der aktuellen Konfiguration im System der Telekom ergab sich jedoch folgende Fehlermeldung:

PHP Notice: Undefined offset: 1 in /usr/src/icinga2-migration/modules/conftool/library/Conftool/Icinga/IcingaConfig.php on line 153

Nach Prüfung der genannten Datei und einer schrittweisen Ausführung mit zusätzlichen Ausgaben stellte sich heraus, dass in der folgenden for-Schleife die Variable „line“ teilweise aufgrund von Leerzeilen ohne Werte befüllt wurde und deshalb ein Offset-Fehler angezeigt wurde:

foreach (preg_split('~\n~', $content, -1, PREG_SPLIT_NO_EMPTY) as $line) {...}

Dies ließ sich durch das Einfügen der folgenden Zeile zwar beheben:

if(empty($line) or ctype_space($line)) continue;

Jedoch ergaben sich bei einer erneuten Ausführung des angepassten Programms weitere Fehler, die nicht korrigiert werden konnten:

ERROR: Trying to add invalid definition dir: /opt/icinga/etc/conf.d

Auch das Einbinden einzelner Konfigurationsdateien anstatt des ganzen Ordners konnte keine Abhilfe schaffen. Das von den Icinga-Entwicklern bereitgestellte Skript kann demnach für die

[13]Migrations-Tool – Umwandlung Icinga 1-Konfiguration in Icinga 2-Konfiguration: https://github.com/Icinga/icinga2-migration

vorhandene Systemkonfiguration nicht genutzt werden. Alternativ könnte ein eigenes Tool für die Migration geschrieben werden, in welchem die jeweiligen Schlüsselwörter durch den neuen Syntax ersetzt werden. Generell lassen sich Hosts und Services auf diese Weise einfach migrieren. Allerdings würde dies eine starre Zuordnung der Objekte voraussetzen. Die Nutzung der dynamischen Apply-/Assign-Regeln würde auf diese Weise kaum oder nur sehr schwer umsetzbar sein. Zudem ist auch die Einführung der neuen Notification-Objekte für die Umsetzung eines automatischen Migrationsskripts als ungünstig einzuschätzen. Der Aufwand für die Erstellung eines derartigen Programms würde nicht mehr in einem passenden Verhältnis zum eigentlichen Nutzen stehen. Da die Host- und Service-Objekte den größten Teil der Konfiguration ausmachen, könnte auch ein Tool, welches nur die Migration dieser Objekte durchführt, erstellt werden. Die Umsetzung wäre einfacher als bei einem vollständigen Migrations-Tool und könnte dennoch eine große Zeitersparnis hervorbringen. Die Untersuchung, ob die Entwicklung eines solchen Programms sinnvoll und somit der manuellen Migration zu bevorzugen wäre, wird im Rahmen der Aufwandsanalyse geschehen. Ohne Betrachtung des Zeitfaktors wäre die manuelle Migration zu bevorzugen, da dadurch eine sauberere und dynamischere Konfiguration geschaffen werden könnte und somit die Vorteile des neuen Syntax zur Verwendung kämen.

3.3 Verteiltes Monitoring

Da das Netz der Telekom aus vielen Segmenten besteht, ist ein verteiltes Monitoring auch aufgrund der in Abschnitt 2.2.2 erwähnten Firewall-Freischaltungen die praktikabelste Lösung. Die Untersuchung der Realisierung dieses Prinzips wird auf den folgenden Seiten geschehen.

3.3.1 Worker

Im aktuellen Wirkbetrieb sind im Netz der DTAG wie bereits erläutert diverse Worker für Abfragen in abgesetzten Netzen aktiv. Diese Funktionalität wird durch das Gearman-Tool realisiert. Aufgrund der Struktur des Netzes ist es unabdingbar auch bei einer eventuell anstehenden Migration dies gewährleisten zu können. Jedoch bietet Icinga 2 keine Gearman-Kompatibilität und sieht diese auch in weiteren Versionen nicht vor.
Icinga 2 ist als Out-of-the-box-Monitoring-Lösung[14] konzipiert wurden und soll damit sämtliche Aufgaben und Funktionalitäten ohne zusätzliche Programminstallationen bereitstellen können. Hierfür gibt es den Icinga 2-Agent. Dadurch können verschiedene Rollen durch Icinga 2 eingenommen werden: Es stehen die Master-, Satelliten- oder Client- Installation zur Verfügung.
Die Satelliten-Implementation soll dabei die Möglichkeiten des Gearman realisieren können. Um die Migration besser beurteilen zu können, ist es daher notwendig, den Satelliten mit dem Gearman zu vergleichen.

3.3.1.1 Installations- und Konfigurationsaufwand

Bei den folgenden Erläuterungen wird davon ausgegangen, dass bereits eine Icinga 1- bzw. Icinga 2-Instanz als Core installiert ist.

Damit in Icinga 1 verteiltes Monitoring über Gearman möglich ist, müssen auf dem Core ein Gearmand und auf jedem der Worker ein mod-Gearman kompiliert und installiert werden. Danach müssen manuell eine Konfigurationsdatei für den Gearmand und jeweils eine weitere für jeden Worker erstellt werden. Zudem wird die Verteilung der Schlüssel für die Sicherung der Verbindung auf die Worker vorausgesetzt. Um Icinga zu signalisieren, dass die auszuführenden Checks in eine

[14]Out-of-the-box-Monitoring – alle Funktionalitäten sollen durch ein Programm abgedeckt werden

Queue des Gearmand geschrieben, statt direkt ausgeführt zu werden, muss zusätzlich noch ein Modul als Eventhandler angelegt werden.
Zusammengefasst ergeben sich folgende Arbeitsschritte:
- Gearmand auf Core kompilieren und installieren
- mod-Gearman auf Workern kompilieren und installieren
- Gearmand-Konfiguration erstellen
- mod-Gearman-Konfiguration auf den Workern erstellen
- Schlüssel für Sicherung der Verbindung auf Worker verteilen
- in Icinga Eventhandler für Gearmand anlegen

Für die Nutzung von verteiltem Monitoring bei Icinga 2 muss auf jedem Satelliten eine Icinga 2-Instanz installiert und gestartet werden. Die Installation erfolgt hierbei im besten Fall über die vorhandenen Pakete in den jeweiligen Repositories um eine gleichmäßige Ordnerstruktur zu gewährleisten und damit keine weiteren Anpassungen der Pfade notwendig zu machen. Daraufhin müssen die Icinga 2-Instanzen konfiguriert werden, wodurch auch eine Zuteilung der Rollen als Master und Satellit geschieht. Dies ist über die Funktion des Node Wizards möglich, welcher durch die Beantwortung einiger weniger Fragen die Konfiguration automatisch erstellt. Letztendlich werden durch diesen Node Wizard die nachfolgenden Aktionen durchgeführt:

Auf dem Master:
- Erzeugen von Backups für alle betreffenden Dateien
- Erstellung von für die Verschlüsselung benötigten Dateien (Master wird zur CA[15])
- Updaten von Zonen- und Endpoint-Konfiguration (Master-Zone mit Endpunkt)
- Anpassen der Konstanten „TicketSalt“ und „NodeName“
- Aktivierung API Feature, optional Anpassung von „bind_host“ und „bind_port“

Auf den Satelliten:
- Erzeugen von Backups für alle betreffenden Dateien
- Festlegung des zugehörigen Masters
- Erstellung von für die Verschlüsselung benötigten Dateien mit Hilfe des Masters
- Herstellung der Verbindung zum Master
- Updaten der Zonen- und Endpoint-Konfiguration (Master- und Satelliten-Zone mit jeweiligen Endpunkten)
- Anpassen der „NodeName“-Konstante
- Aktivierung des API-Features, Deaktivierung des Notification-Features

All diese Schritte lassen sich alternativ auch manuell durchführen, was jedoch in einem größeren Zeitaufwand resultiert.

[15]CA – Certificate Authority; Zertifizierungsstelle zum Abgleich von Zertifkaten für Authentizität eines Nutzers

3.3.1.2 Funktionsweise

Bezüglich der Funktion gibt es diverse Unterschiede in der Verteilung und Festlegung der Check-Abfragen sowie bei der Sicherung der Datenverbindung.

Gearman:

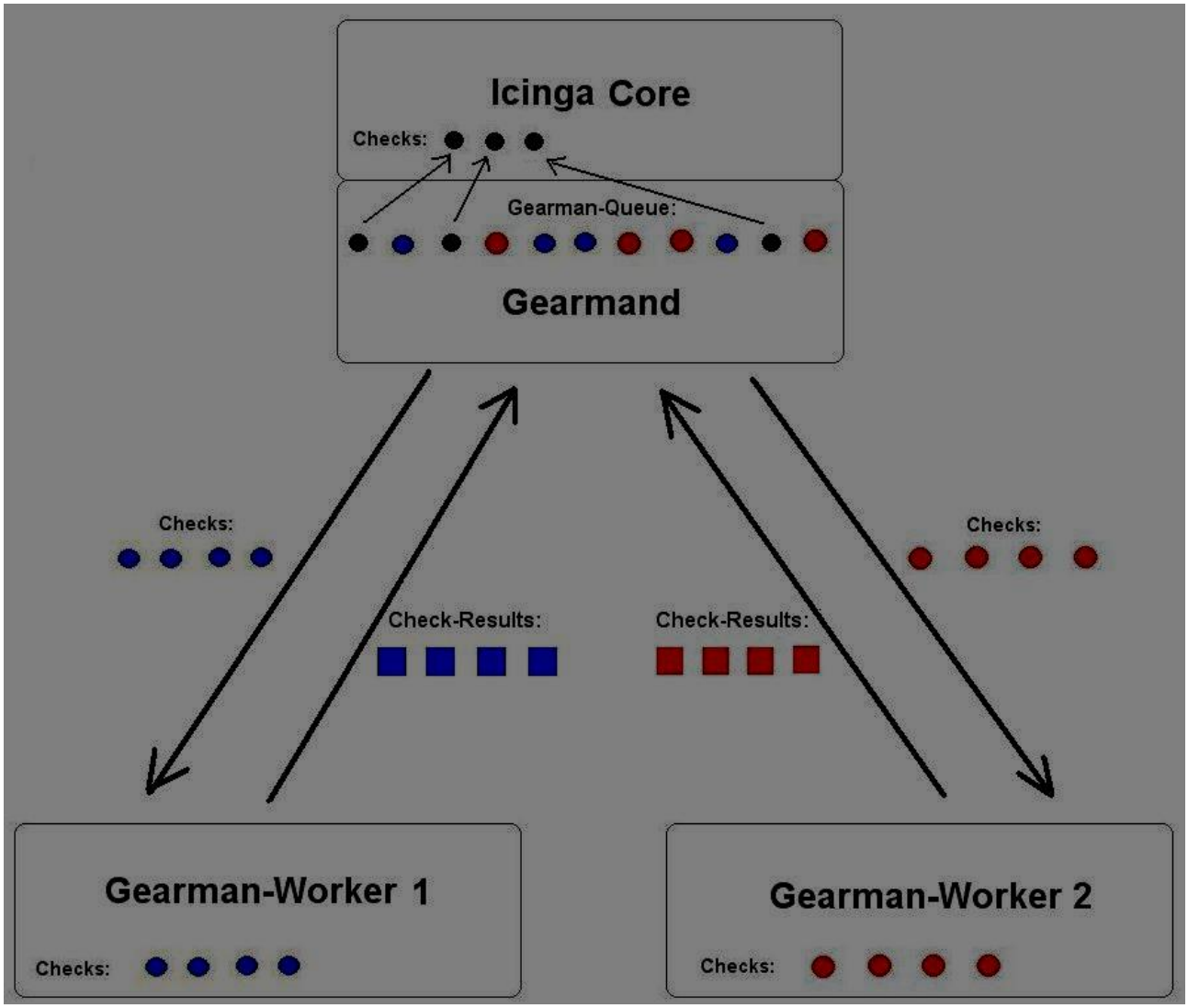


Abbildung 4: Gearman-Funktionsweise

Der Gearmand fängt alle auszuführenden Checks ab und erstellt verschiedene Queues (für Service-Checks, Host-Checks, Eventhandler und benutzerdefinierte Host- oder Servicegruppen), in welche er die Check-Abfragen einordnet. In Abbildung 4: „Gearman-Funktionalität“ wurden diese vereinfacht als einzelne Queue dargestellt. Durch die Einordnung in die unterschiedlichen Queues kann der Gearmand die Abfragen den jeweiligen Workern zuordnen. Sollten sich unter den Checks lokale Abfragen befinden, so werden diese zurück an den Core gegeben, damit dieser eigenständig die Prüfung durchführen kann.
Für alle anderen Check-Abfragen versendet er mittels Gearman-Protokoll binäre Pakete an die entsprechend zugewiesenen Worker. Die Verschlüsselung der Pakete geschieht durch eine symmetrische AES[16]-Blockverschlüsselung mit einem 256 Bit langen Schlüssel.
Die Worker führen die angewiesenen Checks durch und übermitteln die Resultate an eine Check-Results-Queue des Gearmand. Aus dieser Queue werden die Ergebnisse letztendlich an den Icinga-Core transferiert, um dort an ein Frontend für die Visualisierung weitergeleitet werden zu können.

[16]AES - Advanced Encryption Standard; definiert in RFC 3602 der Internet Engineering Task Force

Die Verbindung zwischen den Workern und dem Gearmand bleibt dabei ständig bestehend.

Icinga 2-Satellit:

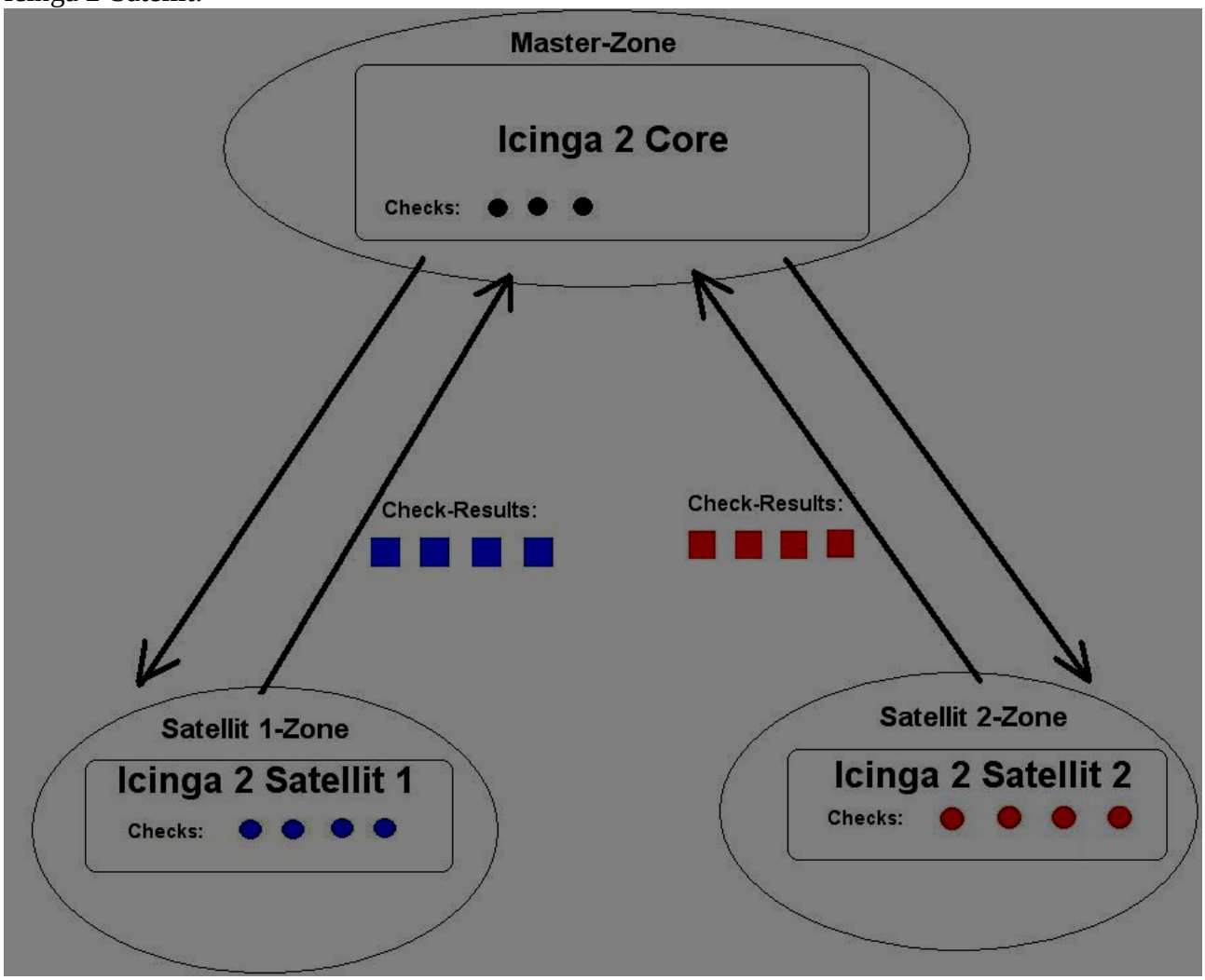


Abbildung 5: Funktionsweise Icinga 2-Satellit

Damit verteiltes Monitoring bei Icinga 2 funktioniert, wird die Definition von Zonen und Endpunkten vorausgesetzt. Dabei muss es eine Master-Zone geben, in der sich die Core-Instanz befindet. Für die Satelliten können je nach Verwendungszweck unterschiedliche oder gleiche Zonen benutzt werden. In Abbildung 5: „Funktionsweise Icinga 2-Satellit“ wurden unterschiedliche Zonen genutzt, damit Abfragen in unterschiedlichen Netzbereichen durchführbar sind. Werden mehrere Satelliten in einer Zone definiert, so ist ein Load-Balancing möglich.
In jedem Fall müssen die Satelliten-Zonen die Master-Zone als Parent zugewiesen bekommen.

Eine Zonenkonfiguration könnte dann folgendermaßen aussehen:

```
object Endpoint "Icinga2Core" {
        host = "10.0.0.1"
}
object Zone "Master-Zone" {
        endpoints = [ "Icinga2Core" ]
}
object Endpoint "Icinga2Satellit1" {
        host = "10.0.0.2"
}
object Zone "Satellit1-Zone" {
        endpoints = [ "Icinga2Satellit1" ]
        parent = "Master-Zone"
}
object Endpoint "Icinga2Satellit2" {
      host = "10.0.0.3"
}
object Zone "Satellit2-Zone" {
        endpoints = [ "Icinga2Satellit2" ]
        parent = "Master-Zone"
}
```

In diesem Fall könnten die Satelliten mit einer zweiten Netzwerkkarte in einem anderen Netzsegment Abfragen ausführen.
Durch die Parent-Beziehungen wird die Hierarchie für die Instanzen festgelegt. Sollten noch Clients an die Satelliten angeschlossen werden, erhalten diese die jeweilige Satelliten-Zone als Parent.
Bei Icinga 2 Satelliten müssen die Host- und Service-Checks sowohl auf dem Core als auch auf den Satelliten definiert sein. Hierbei müssen diese Definitionen identisch sein. Diese Konfiguration kann durch Synchronisation der Icinga 2-Instanzen hergestellt werden. Dabei können entweder die Definitionen auf den Satelliten festgelegt und vom Core aus gepullt werden oder eine zentrale Konfiguration auf dem Core angelegt und an alle Satelliten verteilt werden. Die Abfragen werden dann von jeder Instanz eigenständig durchgeführt. Aufgrund der festgelegten Beziehungen zwischen den Satelliten und dem Core senden diese ihre Check-Ergebnisse an die Master-Zone. Der Core sammelt die Resultate und zeigt sie über ein mögliches Frontend an. Die Kommunikation zwischen den Systemen geschieht über ein Icinga 2-Protokoll, welches auf JSON-RPC[17] aufbaut und über TLS[18] verschlüsselt wird. Der Verbindungsaufbau geschieht über einen API[19]-Listener auf beiden Instanzen. Die Verbindung wird dauerhaft aufrecht erhalten und nur bei einem Neustart eines Icinga2-Systems abgebaut und anschließend direkt wieder aufgebaut.

[17]JSON-RPC – Javascript Object Notation-Remote Procedure Call; JSON definiert in RFC 4627 und RFC 7159
[18]TLS – Transport Layer Security; definiert in RFC 5246 der Internet Engineering Task Force
[19]API – Application Programming Interface

3.3.1.3 Vergleich

Generell ergeben sich durch diese Herangehensweisen einige Differenzen, die Auswirkungen auf den Betrieb der jeweiligen Systeme haben.
Insbesondere beim Verbindungsabbruch ergeben sich signifikante Unterschiede im Verhalten. Trennt man die Verbindung zwischen Icinga und dem Gearmand oder zwischen dem Gearmand und den Workern, so werden keine Abfragen mehr von den Workern durchgeführt, da alle Checks vom Core initialisiert werden. Bei Icinga 2 sind die ausführenden Instanzen vollständige Icinga 2-Installationen mit teilweise deaktivierten Funktionalitäten. Dieses Prinzip resultiert in einem eigenen lokalen Scheduler[20], der die Planung und Durchführung der Checks realisiert. Somit werden auch bei Verbindungsabbruch weiterhin Abfragen durchgeführt. Da es zu jenem Zeitpunkt nicht möglich ist, die Ergebnisse an den Core zu senden, werden sie lokal in einem Replay-Log auf dem Satelliten gespeichert. Beim Wiederherstellen der Verbindung erfolgt eine Synchronisation, so dass auch auf der Master-Instanz die historischen Daten zu den jeweiligen Checks zur Verfügung stehen. Allerdings bewirkt diese Fähigkeit auch, dass in dem Zeitraum der Trennung der Instanzen auf dem Core weiterhin der gleiche Status angezeigt wird. Somit würde ein OK-Zustand angezeigt werden, wenn der Service oder Host diesen Status vor der Trennung der Verbindung aufweisen konnte. Ein Beispiel für dieses Verhalten ist auf Abbildung 6: „Fehlerhafter Status bei Verbindungsverlust“ zu erkennen (links der Status auf dem Master, rechts der Status auf dem betroffenen Satelliten).

Abbildung 6: Fehlerhafter Status bei Verbindungsverlust

Der Nutzer wäre ahnungslos, dass der Satellit bzw. der Service nicht mehr erreichbar sind. Dieses Verhalten verbirgt somit eventuelle Fehler und ist nur durch die Definition einer weiteren Abfrage der Verbindung auf dem Core zu unterbinden. Bei Gearman werden die Services oder Hosts bei so einem Fall als UNKNOWN angezeigt. Die Icinga-Entwickler sind sich diesem Problem bewusst. Da die Ergebnisse auf dem Core dann jedoch mit denen aus dem Replay-Log des Satelliten zu Widersprüchen führen würden, kann dieses Problem nicht einfach behoben werden. Allerdings

[20]Scheduler – Programmteil für die Steuerung der zeitlichen Ausführung von Prozessen

existiert das Feature-Request 8378[21] für das Icinga Web 2 – Modul, in welchem eine Prüfung auf Aktualität der Service-Checks vorgesehen ist, wodurch man diese Schwierigkeiten umgehen könnte. Weitere Schwierigkeiten könnten beim Nutzen von Load-Balancing mittels Icinga 2-Satelliten auftreten. Diverse Funktionalitätstests ergaben ein Verhalten, welches zu irritierenden Ergebnissen führen könnte. So greift jede Icinga 2-Instanz, die für die Ausführung von Checks verantwortlich ist, auf die systemeigene *„command-plugins.conf“* bzw. *„command-plugins-windows.conf“* zu. In diesen befinden sich die Befehlsdefinitionen für die auszuführenden Abfragen. Hier können auch Standardwerte, wie beispielsweise Schwellwerte für die Warning- und Critical-Zustände, für diese Befehle zugeordnet werden. Diese Dateien können nicht ohne zusätzliche Tools synchron gehalten werden. Sollte nun eine der Dateien verändert werden, können unterschiedliche Service-Zustände die Folge sein. Fällt ein Satellit aus, schwenkt die Ausführung des Service-Checks automatisch auf einen anderen. Sind hier andere Schwellwerte eingetragen, kann dementsprechend der Zustand wechseln, obwohl dem Service selbst keine Änderung widerfahren ist. Abhilfe schafft hier die Zuweisung der Werte in den Service-Definitionen, da diese über die Zonenkonfiguration synchronisiert werden können.
Außerdem ist bei der Verwendung von Load-Balancing ein ungünstiges Verhalten bei der Prüfung der lokalen Checks auf den Satelliten selbst aufgetreten. Soll beispielsweise Satellit 1 verschiedene lokale Checks wie Festplattenbelegung, Anzahl aktiver Prozesse oder CPU-Auslastung prüfen, werden diese Checks auf alle Satelliten der Zone verteilt. Es geschieht beispielsweise eine Zuordnung zu Satellit 1. Da es sich jedoch um lokale Checks handelt, sollten diese auf auf den jeweiligen Satelliten lokal durchgeführt werden. Es kann allerdings passieren, dass trotz der Zuweisung eines Services zu Satellit 1 die Prüfung aufgrund des Load-Balancings durch Satellit 2 durchgeführt wird. Dadurch können unter Umständen Ergebnisse von den lokalen Kapazitäten des zweiten Satelliten angezeigt werden. Umgehen lässt sich dieses Verhalten nur durch die Definition des „command-endpoint“-Attributs in den Service-Deklarationen. Dadurch wird vorgeschrieben, dass die Ausführung des jeweiligen Plugins auf exakt diesem einen Server stattfinden muss. Diese Methode erweist sich als umständlich, da auf diese Weise die Checks von einem anderen Satelliten initialisiert werden und Satellit 1 dann auch wieder an diesen das Ergebnis zurücksenden muss. Dadurch werden, wenn auch nur in kleinen Mengen, unnötig Traffic und CPU-Auslastung verursacht. Dieses Problem ist den Icinga 2-Entwicklern bekannt, da auch ein Bug[22] diesbezüglich gemeldet wurde. Eine Anpassung der Struktur bzw. Konfiguration wird demnach zu gegebener Zeit geschehen.
Bei der Sicherung der Verbindung setzt Gearman auf AES-256, welches ein reines Verschlüsselungsverfahren ist. Icinga 2 bietet mit TLS zusätzlich zu der Verschlüsselung noch eine Authentifizierung der jeweiligen Instanzen an.
Jedoch ergeben sich durchaus auch Gemeinsamkeiten zwischen den beiden Programmen. So wird bei beiden eine dauerhafte Verbindung aufgebaut. Auch die Verarbeitung der Check-Ergebnisse geschieht auf eine sehr ähnliche Weise. Sowohl bei Icinga 2, als auch bei Gearman werden die Check-Ergebnisse im Speicher verarbeitet und weiter gesendet (im Gegensatz dazu schreibt Icinga 1 die Ergebnisse in eine Datei auf der Festplatte und liest sie von dort wieder ein). Durch diese Verarbeitung im Speicher sowie die Nutzung von Multi-Threading lässt sich auch der angepriesene Geschwindigkeitsvorteil gegenüber Icinga 1 erklären. Es ist jedoch zu erwähnen, dass Gearman weniger Speicher nutzt als Icinga 2, was in der Tatsache begründet liegt, dass die Icinga-Instanzen weitaus mehr Funktionen liefern.

[21]Feature Request 8378 - „Indicate when check results are being late“ - https://dev.icinga.org/issues/8378
[22]Bug – Programmfehler; bezüglich Check-Ausführung auf festem Endpoint: https://dev.icinga.org/issues/10040

Zusammengefasst zeigt sich vergleichend folgendes Ergebnis:

	Icinga 2-Satellit	Gearman
Verschlüsselung	TLS	AES-256
Übertragungsprotokoll	Icinga 2-spezifisch (auf JSON-RPC basierend)	Gearman-spezifisch (binäre Pakete)
Verbindung zwischen Instanzen	dauerhaft	dauerhaft
Verarbeitung der Ergebnisse	im Speicher	im Speicher
Übertragene Daten vom Core/Gearmand	Nur Verbindungsdaten	Verbindungsdaten und Aufgaben (Checks)
Übertragene Daten vom Satelliten/Worker	Check-Ergebnisse und Verbindungsdaten	Check-Ergebnisse und Verbindungsdaten
Verhalten bei Verbindungsverlust	Weiterhin Ausführung von Abfragen	Keine Aktionen
Unterstützte Betriebssysteme	Windows- und Linux-Distributionen	Linux-Distributionen
Zuordnung der Abfragen	Über Zonen- und Endpunkt-Definitionen	Über Host- und Service-Gruppen-Queues

Tabelle 6: Vergleich Icinga 2-Satellit und Gearman

3.3.1.4 Fazit

Bei einer Umstellung von Icinga 1 auf Icinga 2 bleibt letztendlich keine andere Wahl als auf Gearman zu verzichten und den Icinga 2-Satelliten zu nutzen. Es stellte sich bei der Verwendung von Satelliten heraus, dass die Implementation teilweise noch gewisse problematische Verhaltensweisen, wie die ungünstige Ausführung der lokalen Checks bei Load-Balancing, bietet. Vor allem aber die irreführenden Ergebnisse bei Verbindungsabbruch könnten unter Umständen zu Schwierigkeiten führen. Da beide Problematiken bekannt sind, entsprechende Workarounds existieren und auch Icinga 2 ständig weiterentwickelt wird, sind diese Verhaltensweisen für die Nutzung im Netz der Telekom als unkritisch zu beurteilen. Natürlich ist die größere Speichernutzung als negativ anzusehen, jedoch bieten die dadurch möglichen Vorteile wie die trotz Verbindungsverlust stattfindende Prüfung der Clients und das damit verbundene Replay-Log auch einen erheblichen Mehrgewinn. Des Weiteren wird durch die Nutzung der Satelliten eine homogene Monitoring-Umgebung geschaffen. Es fällt damit eine Abhängigkeit von einem weiteren Programm weg, wodurch Wartbarkeit und Software-Pflege deutlich vereinfacht werden. In Bezug auf Wartbarkeit bzw. das Korrigieren von Fehlverhalten ist Gearman von einzelnen Entwicklern abhängig und hält eine wesentlich kleinere Community bereit, die bei Fragestellungen helfen könnte.
Alles in Allem kann der Icinga 2-Satellit alle Anforderungen erfüllen, die für das Monitoring im Bereich OSS der Deutschen Telekom AG von Bedeutung sind. Ein Umstieg ist aufgrund der genannten Vorteile deshalb zu empfehlen.

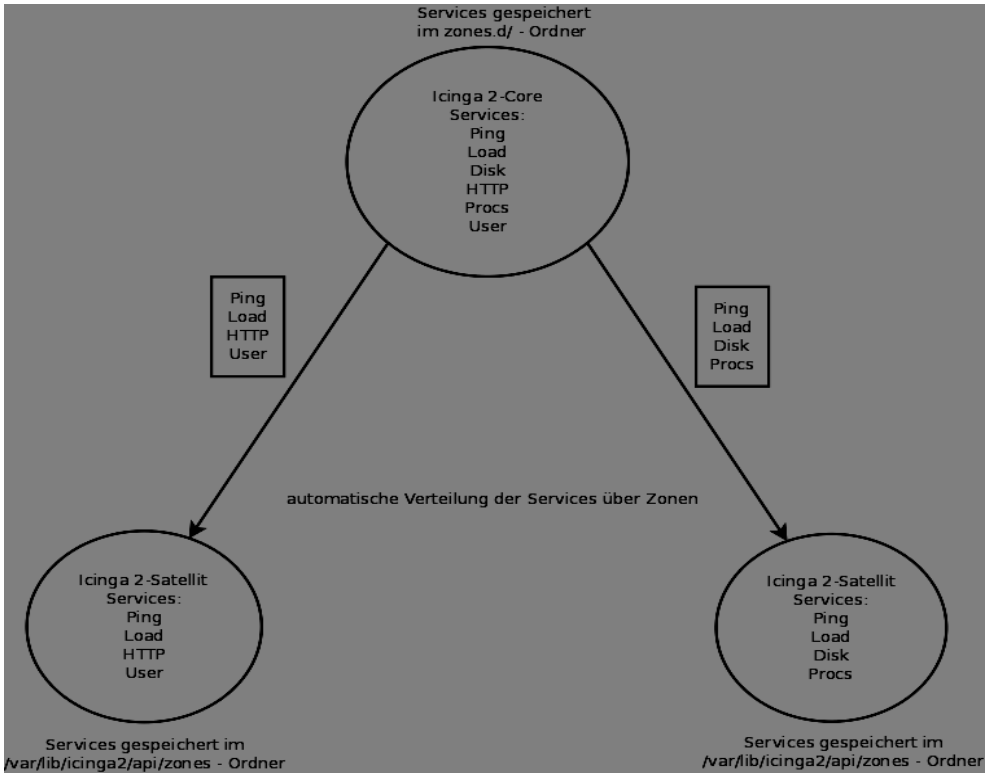
3.3.2 Struktur

Sollte eine Migration auf ein Icinga 2-System geschehen, stellt sich die Frage, ob eine zentrale Konfiguration auf dem Core oder eine Definition der Objekte auf den Satelliten sinnvoller wäre. Im bisherigen System wurde eine zentrale Verwaltung genutzt. Allerdings gab es bei Icinga 1 aufgrund der beschränkten Funktionen des Gearmans auch nicht die Möglichkeit dies anders zu gestalten. Icinga 2 bietet die beiden Ansätze der automatischen Konfigurationsverteilung vom Core aus oder das Aktualisieren der Konfiguration auf dem Core durch einen Befehl, der alle verbundenen Instanzen zum Senden der Konfigurationen bewegt. In jedem Fall ist Icinga 2 so konzipiert, dass die Objekte auf dem Core in gleicher Art und Weise definiert sein müssen wie auf den Satelliten. Bei verschiedenen Definitionen verbleiben die Services und Hosts im „Pending“-Status.

3.3.2.1 Top-Down

Bei der Festlegung der Objekte auf dem Core, müssen im *„zones.d“*-Ordner des Icinga 2-Verzeichnisses Unterordner mit den Namen der entsprechenden Zonen erstellt werden. Die in diesen Ordnern definierten Konfigurationsdateien werden dann automatisch an die Satelliten der jeweiligen Zonen verteilt. Die Speicherung erfolgt im Verzeichnis *„var/lib/icinga2/api“*. Das Verteilen der Konfiguration geschieht bei jedem Neustart bzw. Wiederherstellen einer Verbindung zwischen Icinga 2-Instanzen. Dieses Verfahren wird als „Push“-Methode oder Top-Down-Design bezeichnet.

Abbildung 7: Top-Down



3.3.2.2 Bottom-Up

Das Definieren der Objekte auf den Satelliten hingegen geschieht per „Pull“-Methode bzw. Bottom-Up-Design. Hierfür muss auf dem Core das Kommando „icinga2 node update-config“ ausgeführt werden. Dadurch holt sich dieser die Konfigurationen der verbundenen Icinga 2-Instanzen. Die Speicherung erfolgt im *„/etc/icinga2/repository.d/“*-Verzeichnis des Cores. Ein Automatisches Aktualisieren der Konfiguration ist bei dieser Methode bisher nicht implementiert.

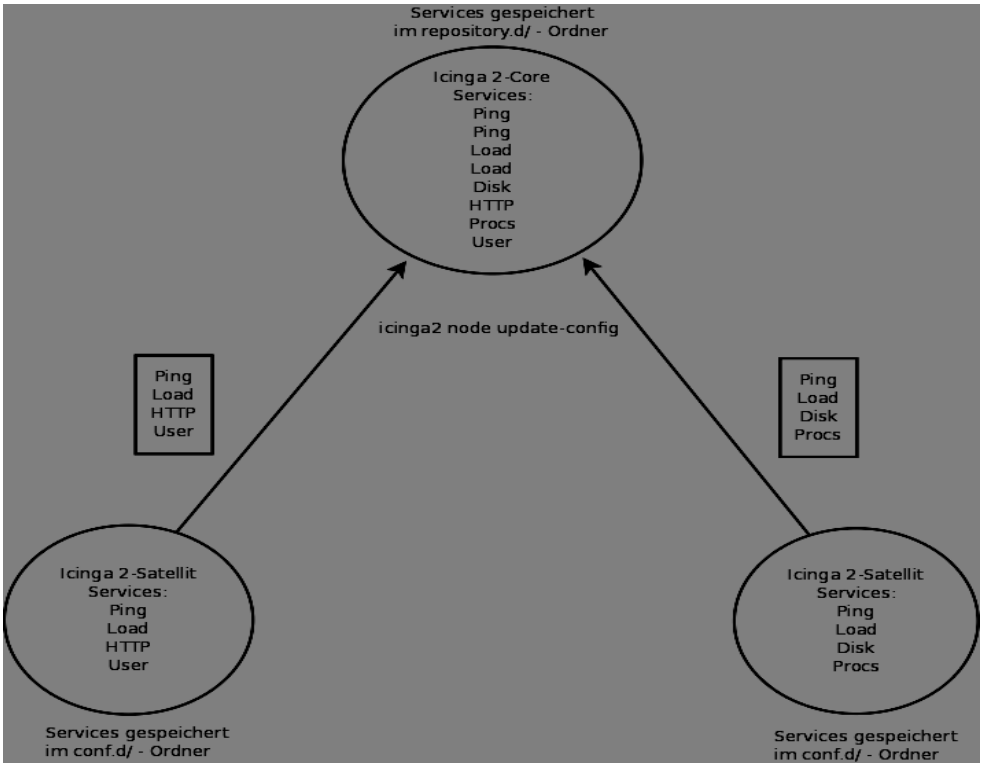


Abbildung 8: Bottom-Up

3.3.2.3 Vergleich

Generell ist ein Top-Down-Design in Bezug auf den Konfigurationsaufwand zu präferieren, da alle Services auf einem Server nur über die Zonen-Ordner zugeordnet und dort angelegt werden müssen. Bei der Verwendung von Bottom-Up müssen die Services auf jedem Satelliten definiert werden und dann per Kommando auf den Core übertragen werden. Das bedeutet auch, dass die Zuordnung der Services unter Umständen nicht so effektiv ist wie bei einer zentralen Konfiguration. Wenn beispielsweise auf zwei Satelliten teilweise gleiche Services vorhanden sind, wie das in Abbildung 8: „Bottom-Up“ der Fall ist mit den Ping- und Load-Services, so können diese beim Top-Down-Design mittels Assign-Regeln mehreren Instanzen zugeordnet werden. Es gibt somit jeweils nur eine Objektdefinition. Die Nutzung von Bottom-Up hingegen bedingt jeweils eine Service-Definition auf jedem Satelliten, die dem Host dann starr zugeordnet werden. Demnach muss der Service mehr als nur einmal definiert werden, was einen höheren Konfigurationsaufwand zur Folge hat. Führt der Nutzer dann den Befehl zur Übertragung der Konfiguration an den Core aus, werden im repository.d/-Verzeichnis des Cores für jeden Host ein Ordner und in diesem für jeden Service eine Datei angelegt. Positiv ist in diesem Fall, dass sehr schnell ersichtlich ist, welche Services einem Host zugeordnet sind. Allerdings ergibt sich dadurch auch eine aufgeblähte Dateistruktur. Ein Beispiel für die unterschiedlichen Dateistrukturen bei gleich definierten Services und Hosts ist in Anhang A: „Dateistruktur Bottom-Up und Top-Down“ dargestellt.
Den Vorteilen des Top-Down-Designs wie beispielsweise der automatischen

Konfigurationsverteilung und dem geringeren Aufwand, sowie der damit einhergehenden kompakteren Dateistruktur, steht unter anderem ein erhöhter Traffic gegenüber. Dieser kommt zu Stande durch die Synchronisation der Konfigurationsdateien bei jedem Aufbau bzw. jeder Wiederherstellung einer Verbindung zwischen Icinga 2-Instanzen. Zudem können bei unpassender Wahl der Assign-Regeln, so dass diese auf Hosts in verschiedenen Zonen zutreffen, doppelte Objekte auftreten, die zu Konfigurationsfehlern führen. Services, die einer falschen Zone zugeordnet wurden, bleiben jedoch im „Pending"-Status. Derartigem Verhalten könnte allerdings relativ einfach vorgebeugt werden, indem beispielsweise zusätzlich auf eine benutzerdefinierte Variable „Zone" gefiltert wird. Allerdings weist diese zonenübergreifende Zuordnung auch Vorteile auf, da hiermit Notifications dementsprechend zugewiesen werden können. Da die Konfiguration beim Top-Down-Design an zentraler Stelle geschieht, sind einheitliche Strukturen auch einfacher zu realisieren. Die Bottom-Up-Methode zeichnet sich vor allem durch die Übersichtlichkeit in Bezug auf die zugewiesenen Services und die wesentlich geringere Netzwerkbelastung aus (nur einmal während der Ausführung des Befehls zum Senden der Konfiguration). Allerdings stellte sich bei dieser Design-Variante heraus, dass die Verwendung von Bottom-Up nicht für eine Struktur mit mehreren Core-Instanzen geeignet ist. Führt man hier ein „icinga2 node update-config" aus, so wird ein Core als Haupt-Instanz ausgewählt und alle anderen untergeordnet, wodurch kein Load-Balancing und auch keine Ausfallsicherheit mehr gegeben ist. Trotz entsprechender Zonen-Definition wird nur eine Master-Instanz anerkannt.

Alle Vor- und Nachteile sind noch einmal in der nachfolgenden Tabelle zusammengefasst:

	Top-Down	Bottom-Up
Pro	+ zentrale Verwaltung (einheitliche Strukturen) + automatische Konfigurationsverteilung + geringerer Konfigurationsaufwand + kompakte Dateistruktur	+ Übersichtlickeit, welchem Host welche Services zugeordnet sind + weniger Traffic
Contra	- mehr Traffic (Sync bei Reconnect) - Services werden nicht nur innerhalb der Zone zugeordnet	- Konfiguration wird nicht automatisch übertragen - aufgeblähte Dateistruktur - Änderungen an mehreren Hosts sehr aufwändig - Cluster mit mehreren Core-Instanzen wird nicht erkannt

Tabelle 7: Vergleich Top-Down und Bottom-Up

3.3.2.4 Fazit

Sowohl Bottom-Up als auch Top-Down bieten Vorteile, die im Netz der Deutschen Telekom AG einen großen Einfluss haben. Die Übersichtlichkeit, die das Bottom-Up-Design mit sich bringt, ist insbesondere für der Fehlersuche bei einer Vielzahl von Hosts und Services sehr hilfreich. Dem gegenüber steht die zentrale Verwaltung mittels Top-Down-Methode. Da im Bereich OSS der Telekom viele tausend Hosts überwacht werden, bietet eine zentrale Konfigurationsverwaltung eine erhebliche Zeitersparnis für das Erstellen beziehungsweise Ändern der verschiedenen Konfigurationen. Auch die Tatsache, dass viele Hosts ähnliche oder gleiche Services besitzen, kommt dem Top-Down-Design entgegen. Nur damit ist eine optimale Nutzung der neuen Assign-Regeln möglich, wodurch Services mehreren Hosts zugeordnet werden können. Dadurch ergibt sich eine Einsparung vieler Konfigurationszeilen. Darüber hinaus ist die Einschränkung der Cluster-Nutzung bei Verwendung des Bottom-Up-Designs ein weiterer ausschlaggebender Faktor, welcher für den Einsatz des Top-Down-Designs spricht. Es ist deshalb zu empfehlen beim Umstieg von Icinga 1 auf Icinga 2 für den OSS-Bereich der Deutschen Telekom AG eine zentrale Konfiguration mittels Top-Down zu verwenden.

3.4 Remote-Abfragen

Beim Monitoring werden, wie in Abschnitt 2.2.3 „Plugins“ beschrieben, verschiedenste Ressourcen abgefragt. Dafür stehen unterschiedlichste Plugins zur Verfügung. Der Hauptanteil der Abfragen besteht in der Prüfung von lokalen Ressourcen auf den diversen Clients. Diese können jedoch nicht ohne zusätzliche Hilfsmittel abgefragt werden, wenn auf dem jeweiligen System keine Monitoring-Instanz läuft.
Lediglich bestimmte Netzwerkdienste lassen sich von einem Satelliten oder Core an einem Client prüfen. Hierzu zählen beispielsweise Dienste wie HTTP, SSH oder SMTP. Die Prüfung erfolgt dabei anhand eines spezifischen Ports, der im jeweiligen Check-Plugin definiert ist. Für die genannten Dienste wäre das Port 80 (HTTP), 22 (SSH) und 25 (SMTP; ggf. auch Port 587).
Jedoch gibt es auch viele Services, die nicht netzwerkfähig sind und nur lokal abgefragt werden können. Hierzu zählen zum Beispiel die Auslastung des Prozessors oder der Festplatte sowie die Abfrage der Anzahl laufender Prozesse. Um diese Service-Checks durchzuführen, werden zusätzliche Hilfsprogramme auf den Clients benötigt, welche die Abfragen durchführen und die Resultate an die jeweilige Monitoring-Instanz senden.
Da Icinga 2 generell alle Check-Plugins unterstützt, die auch in Icinga 1 genutzt werden können, werden auf den nachfolgenden Seiten einige der am meisten verbreiteten und auch im Umfeld des aktuellen Systems verwendeten Möglichkeiten für Remote-Abfragen untersucht und verglichen, um eventuelle Optimierungen diesbezüglich im Monitoring-Bereich OSS der Telekom aufzuzeigen und letztendlich die bestmögliche Variante auszuwählen. Untersucht werden hierbei der Nagios Remote Plugin Executor (NRPE), das check_by_ssh-Plugin, der NS-Client++, der Nagios Service Check Acceptor (NSCA) sowie die Abfragen mittels Simple Network Management Protocol (SNMP). Außerdem wird in diesem Zusammenhang eine Prüfung von Remote-Abfragen mittels Icinga 2 Client als Command Execution Bridge vorgenommen.

3.4.1 NRPE – Nagios Remote Plugin Executor

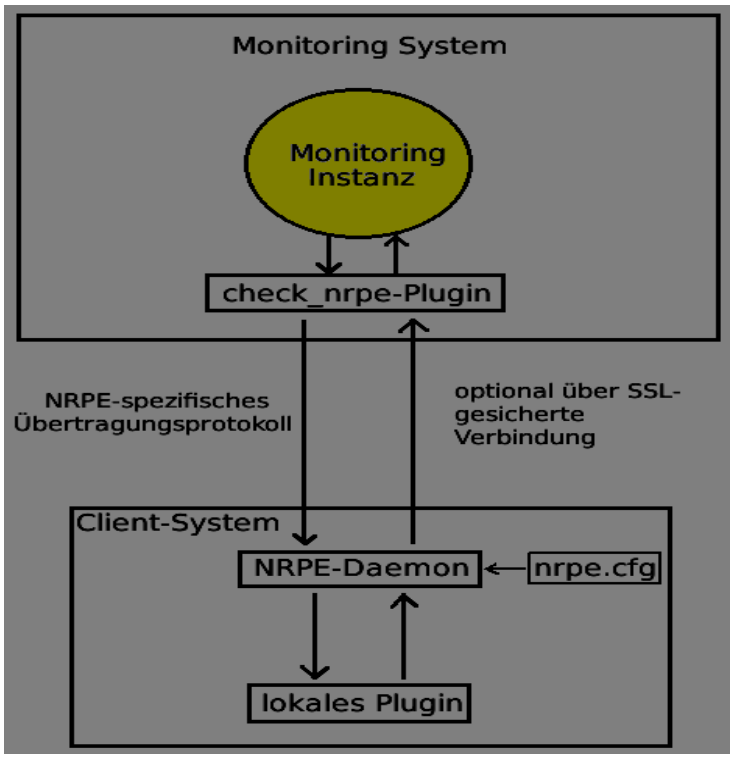


Abbildung 9: Funktionsweise NRPE

Der Nagios Remote Plugin Executor besteht auf der Seite der abfragenden Instanz aus dem check_nrpe-Plugin und auf der Client-Seite aus einem zu installierenden NRPE-Agent/-Daemon. Das Monitoring von Windows-Systemen über NRPE ist allerdings nur durch die Installation zusätzlicher Software möglich. Bei der Ausführung des check_nrpe-Plugins werden verschiedene Argumente verwendet, wie auch der eigentliche Befehl zur Abfrage der lokalen Ressource mit Hilfe des Parameters „-c“. Die Befehlsdefinition ergibt sich bei Icinga 2 gemäß Abbildung 10:

```
object CheckCommand "nrpe" {
        import "plugin-check-command"
        import "ipv4-or-ipv6"

        command = [ PluginDir + "/check_nrpe" ]

        arguments = {
                "-H" = "$nrpe_address$"
                "-p" = "$nrpe_port$"
                "-c" = "$nrpe_command$"
                "-n" = {
                        set_if = "$nrpe_no_ssl$"
                        description = "Do not use SSL."
                }
                "-u" = {
                        set_if = "$nrpe_timeout_unknown$"
                        description = "Make socket timeouts return an UNKNOWN state instead of CRITICAL"
                }
                "-t" = "$nrpe_timeout$"
                "-a" = {
                        value = "$nrpe_arguments$"
                        repeat_key = false
                        order = 1
                }
        }

        vars.nrpe_address = "$check_address$"
        vars.nrpe_no_ssl = false
        vars.nrpe_timeout_unknown = false

        timeout = 5m
}
```

Abbildung 10: Befehlskonfiguration check_nrpe

Daraus folgt beispielsweise eine solche Service-Definition zum Prüfen der Festplatte des Clients:

```
apply Service "NRPE-load" {
        import "generic-service"
        check_command = "nrpe"
        vars.nrpe_command = "check_load"
        assign where host.name == remoteclient1
}
```

Wenn nun der Service-Check durchgeführt werden soll, greift dieser auf das Check_Command „nrpe“ zurück, welches zur Ausführung des check_nrpe führt. Diesem werden anhand der in Abbildung 10 definierten Variablen verschiedene Parameter übergeben. So wird beispielsweise die benutzerdefinierte Variable nrpe_address mit der IP-Adresse des für den Service zugewiesenen Hosts befüllt und dann über den „-H“-Parameter an das Plugin übergeben.
Das check_nrpe-Plugin spricht dann mit Hilfe der jeweiligen Adresse einen auf dem entsprechenden Host laufenden Agenten an. Dafür wird eine TCP-Verbindung über den standardmäßig definierten Port 5666 aufgebaut. Die Übermittlung der Anfrage geschieht dann durch ein proprietäres Protokoll. Durch den in der benutzerdefinierten Variable nrpe.command festgelegten Befehl kann der Agent das entsprechende lokale Plugin auf dem Client ausführen. Voraussetzung hierfür ist das Vorhandensein des jeweiligen Plugins, sowie die Definition des jeweiligen Befehls und der Hosts, welche sich mit dem NRPE-Agent verbinden dürfen. Diese Festlegungen werden in der *nrpe.cfg* auf dem Client eingestellt:

```
allowed_hosts=10.0.0.1
command[check_load]=/usr/local/nagios/libexec/check_load
```

Die Ergebnisse wiederum sendet der Agent an das check_nrpe-Plugin der ausführenden Instanz zurück. Alle Verbindungen sind nach der Installation unverschlüsselt, wodurch eine sehr schnelle Übertragung aufgrund des geringen Overheads gewährleistet wird. Es besteht dennoch die Möglichkeit eine Verschlüsselung per SSL[23] zu aktivieren. Dafür wird der anonymous Diffie-Hellmann-Algorhythmus verwendet. Der zur Verwendung kommende Schlüssel wird allerdings zur Kompilierungszeit in Abhängigkeit von OpenSSL- und Betriebssystem-Konfiguration erstellt. Dadurch steht eine begrenzte Anzahl von Schlüsseln zur Verfügung.
Eine weitere Schwachstelle, welche auch in die Common Vulnerabilities and Exposures[24] aufgenommen wurde, ist die Möglichkeit der Ausführung von nicht vordefinierten Kommandos. Aufgrund des unsauber programmierten Programmcodes kann durch diese Schwachstelle jegliche Art von Befehlen ausgeführt werden. Die Nutzereingaben werden nicht geprüft, weshalb somit auch schädlicher Code an die Command Shell übergeben werden kann.
Zusammen mit der nicht vorhandenen Authentifizierung ergibt sich ein als nicht sehr sicher einzuschätzendes Verfahren.

[23]SSL – Secure Sockets Layer
[24]Common Vulnerabilities and Exposures (CVE) – standardisierte Liste von bekannten Schwachstellen von Systemen

3.4.2 NSClient++

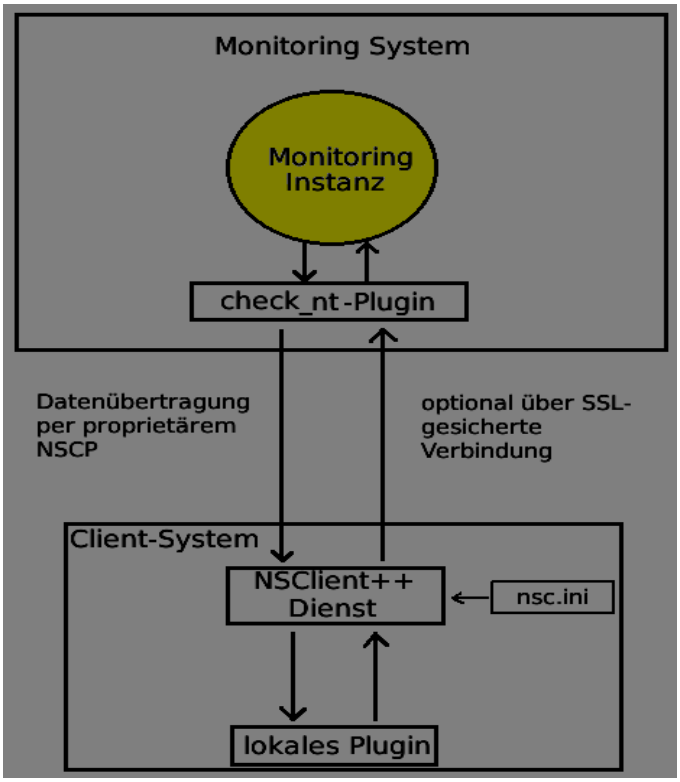


Abbildung 11: Funktionsweise NSClient++

Für das Monitoring von Windows-Systemen ist das reine NRPE-Plugin nicht geeignet. Hier gilt der NSClient++ als Alternative. Dieser funktioniert ähnlich dem NRPE-Dienst und kann diesen bei Bedarf auch nutzen. Anstelle des check_nrpe-Plugins baut dabei das check_nt-Plugin die Verbindung zum NSClient++ Dienst des Windows-Hosts auf. Die Konfiguration geschieht über die *„nsc.ini“* (in neueren Versionen *„nsclient.ini“*) und Befehle werden ebenfalls als Parameter übergeben. Auch die Verschlüsselung ist identisch mit NRPE: standardmäßig werden die Daten im Klartext übertragen; Es existiert aber die Möglichkeit eine SSL-Verschlüsselung zu aktivieren. Die Datenübertragung passiert mittels proprietärem NSCP-Protokoll. Die eigentliche Funktionsweise ist analog dem NRPE-Dienst: der NSClient++ Dienst nimmt die Abfragen entgegen, führt die lokalen Plugins aus und sendet die Ergebnisse an das check_nt-Plugin.
Theoretisch ist eine Nutzung von NSClient++ auch auf Unix-Systemen möglich, jedoch sind diese Implementationen nicht sonderlich ausgereift.

3.4.3 Check_by_ssh

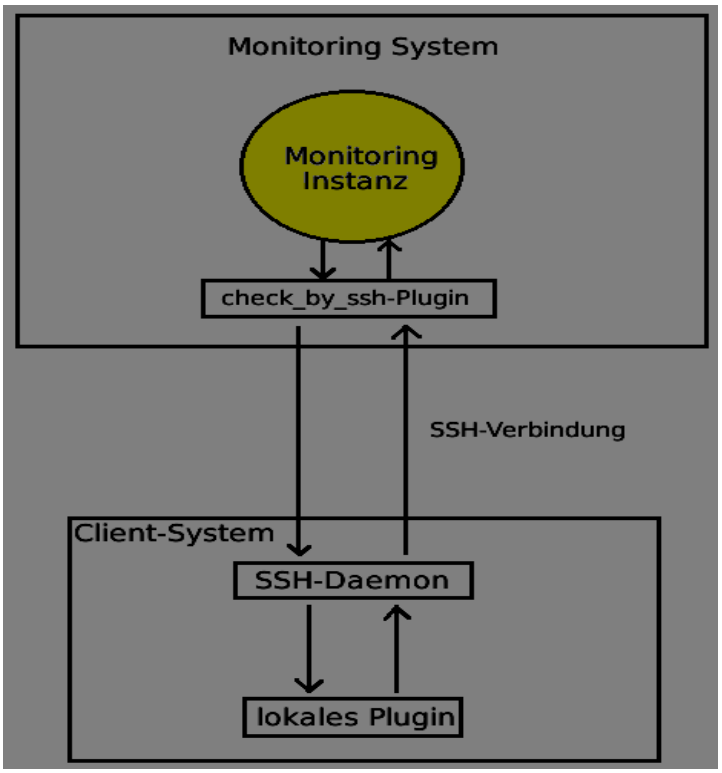


Abbildung 12: Funktionsweise check_by_ssh

Eine oft benutzte Alternative zum NRPE-Dienst stellt das check_by_ssh-Plugin dar. Hier wird durch das Plugin eine SSH-Verbindung zum SSH-Daemon des entfernten Hosts hergestellt. Da SSH auf Unix-Systemen standardmäßig installiert ist, besteht hier kein zusätzlicher Installationsaufwand. Lediglich bei Windows-Systemen muss der Dienst über ein externes Programm noch integriert werden. Es ist allerdings sowohl auf Windows- als auch auf Unix-Systemen nötig, einen entsprechenden User für die Ausführung der Befehle anzulegen. Damit nicht bei jeder Abfrage die Login-Daten eingegeben werden müssen, sollte ein Schlüsselpaar auf Monitoring-Server und Client hinterlegt sein, über welches sich beide authentifizieren können.
Wie auch bei NRPE wird mit Hilfe des Parameters „-c“ der eigentliche Befehl übermittelt. Der auf dem Host eingerichtete Nutzer führt diese mit entsprechend übergebenen Parametern aus, damit das Check-Resultat per SSH zurück an das check_by_ssh geliefert werden kann. Von dort werden die Ergebnisse an den Monitoring-Core übergeben, um letztendlich im Frontend angezeigt werden zu können.
Generell ist check_by_ssh eine sichere Variante für Remote-Abfragen. Jedoch kann ein Angreifer bei Kompromittierung des Monitoring-Systems nicht nur die Ausführung von Plugins auf dem entfernten Host veranlassen; auch die Ausführung von unterschiedlichen Programmen ist durch die dem Nutzer zwangsläufig zugewiesenen Rechte möglich.
Da für jede Abfrage eine separate Verbindung aufgebaut und nach dem Übermitteln des Ergebnisses geschlossen wird, ist der Overhead auch aufgrund der zusätzlichen Authentifizierung größer als beim NRPE, weshalb NRPE geringfügig performanter ist als das check_by_ssh-Plugin. Dieser Vorteil spielt jedoch erst bei einer sehr großen Anzahl an Services eine Rolle, da für die Überwachung Bandbreiten im Gigabit-Bereich zur Verfügung stehen.

3.4.4 NSCA – Nagios Service Check Acceptor

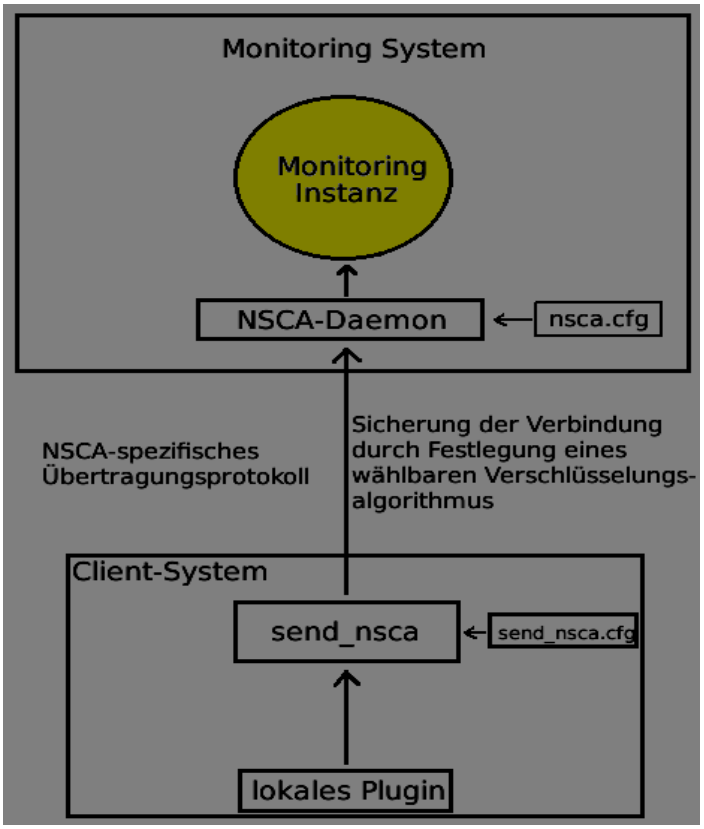


Abbildung 13: Funktionsweise NSCA

Im Gegensatz zu den bisher genannten aktiven Remote-Checks, geht die Initiative beim Nagios Service Check Acceptor nicht vom Monitoring-Core aus. Die Abfrage wird hier durch den auf dem Host laufenden NSCA-Dienst gestartet. Das heißt, die Monitoring-Instanz muss nicht erst einen Check initialisieren, damit dieser ausgeführt wird. Man spricht deshalb von einem passiven Check.
Auf dem Client läuft dafür ein NSCA-Daemon, der die Ergebnisse der jeweiligen Checks entgegen nimmt und diese mittels „send_nsca“ an einen NSCA-Daemon auf der Monitoring-Instanz überträgt. Hierbei ist die „send_nsca.cfg“ für die Konfiguration der Client-Seite und die „nsca.conf“ für die Konfiguration auf der Monitoring-Seite zuständig. In diesen Dateien können diverse Einstellungen wie Debugging, der Nutzer, das Passwort oder die Verschlüsselung angepasst werden. Das Passwort ist im Klartext gespeichert und regelt, wer Befehle an den Monitoring-Core senden darf. Aufgrund dieser Klartext-Speicherung ist sehr darauf zu achten, dass die Rechte für die entsprechenden Dateien passend gesetzt sind. Für die Verschlüsselung der Verbindung stehen verschiedene Verfahren zur Verfügung, welche teils sehr unterschiedliche Sicherheit bieten. Bekannte mögliche Verschlüsselungsverfahren sind beispielsweise DES[25], AES oder LOKI97.
Da bei dieser Variante der Abfrage von Hosts der Monitoring-Core selbst keine Checks initialisiert, ist NSCA eine verhältnismäßig performante Lösung für das Gesamtsystem. Jedoch besteht aufgrund der passiven Checks die Möglichkeit Kommandos an den Core zu senden. Bei Kompromittierung des Clients ist keine Sicherheit mehr gegeben.
NSCA ist als reine Linux-Anwendung entwickelt wurden, kann aber mit Hilfe von zusätzlichen Programmen wie NSClient++ auch auf Windows-Systemen genutzt werden.

[25]DES – Data Encryption Standard

3.4.5 SNMP – Simple Network Management Protocol

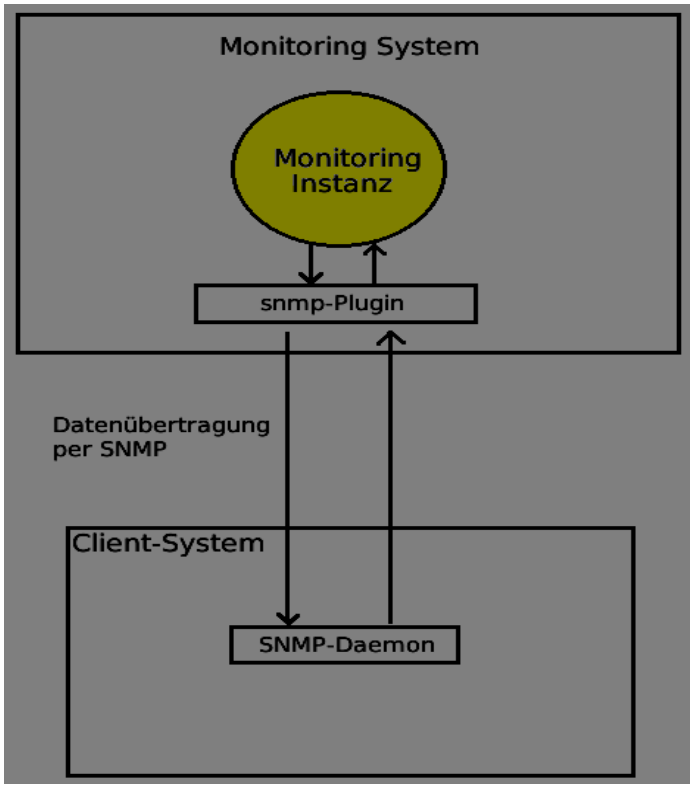


Abbildung 14: Funktionsweise SNMP-Remote-Abfragen

Ein weitere Möglichkeit zur Abfrage lokaler Ressourcen auf einem Remote-System ist die Nutzung des Simple Network Management Protocols. Dieses eignet sich im Gegensatz zu den anderen genannten Varianten nicht nur zur Abfrage von Unix- oder Windows-Systemen; mit SNMP können fast alle netzwerkfähigen Geräte, wie beispielsweise Router oder Switche abgefragt werden.
Für Windows- und Unix-Systeme steht auch hier wieder ein Daemon auf dem Remote-System zur Verfügung, welcher die eigentlichen Abfragen durchführt. Die Initialisierung geht auch bei dieser Variante vom Core aus, welcher dafür das jeweilige SNMP-Plugin aufruft. Auf hardwarenahen Geräten ohne Unix- oder Windows-Betriebssystem kommen als Daemon vom Hersteller implementierte SNMP-Engines zum Einsatz. Es besteht somit kein zusätzlicher Installationsaufwand. Lediglich die Abfragen müssen konfiguriert werden.
Bei Abfragen greift SNMP auf eine Management Information Base (MIB) zurück. Diese Informationsstruktur ist hierarchisch aufgebaut. Die Management-Objekte können hierbei eindeutig durch einen Object Identifier (OID) bestimmt werden. Die jeweilige OID wird in dem Plugin angegeben und signalisiert dem SNMP-Daemon, welche Ressource er abfragen soll.
Per SNMP können generell auch Management-Befehle ausgeführt werden. In der Version 1 und 2 besteht hier lediglich die Möglichkeit über ein Passwort (Community-String) eine Zugriffskontrolle zu setzen (nur Lesen oder Lesen und Schreiben). Da das Passwort außerdem im Klartext übertragen wird, sind diese Versionen als sehr unsicher einzuschätzen. Mit SNMPv3 wurden zusätzliche Authentifizierungs- und Verschlüsselungsmöglichkeiten implementiert, wodurch die Nutzung auch in sensibleren Umgebungen denkbar ist.
Zusätzlich zu diesen aktiven Abfragen, hält SNMP auch die Nutzung von sogenannten Traps bereit. Traps sind Ereignisse, die auf dem jeweiligen Host geschehen und eine Datenübermittlung an eine Monitoring-Instanz zur Folge haben. Mit Hilfe eines Trap-Receivers auf dieser Instanz können sie empfangen und die Ergebnisse von der Monitoring-Instanz dargestellt werden. Somit besteht bei SNMP auch die Möglichkeit passive Checks zu verwenden.
Da SNMP-Abfragen nicht viel Rechenleistung benötigen und für den Transport der Daten UDP-Packete genutzt werden, ist die Abfrage per SNMP sowohl in Bezug auf die CPU- als auch die

Netzwerkauslastung eine verhältnismäßig schnelle Alternative.

3.4.6 Icinga 2-Client

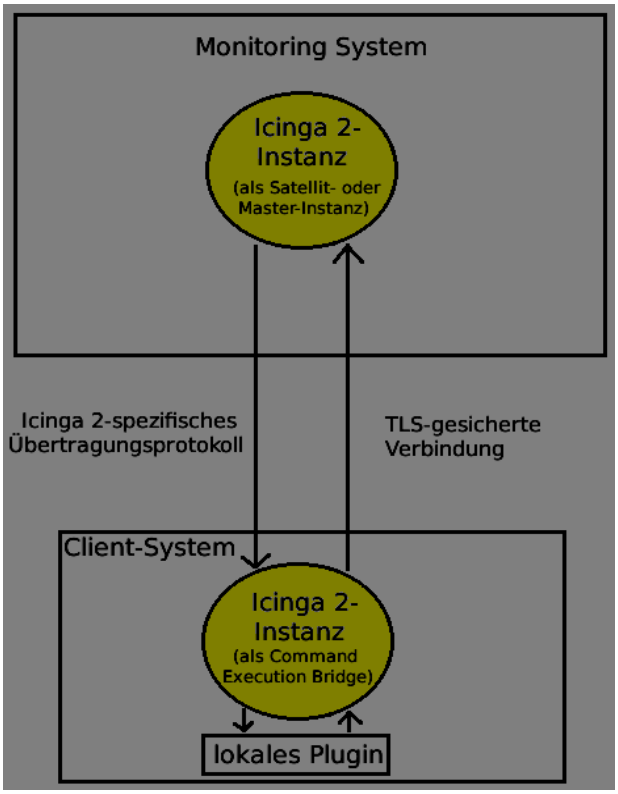


Abbildung 15: Funktionsweise Icinga 2-Client als Command Execution Bridge

Wie bereits im Laufe dieser Arbeit erwähnt, gibt es bei Icinga 2 die Möglichkeit unterschiedliche Installationsarten umzusetzen. Die Variante des Icinga 2-Clients als Command Execution Bridge soll dabei die Abfrage von Remote-Systemen ermöglichen. Im Gegensatz zum Icinga 2-Satelliten arbeitet der Icinga 2-Client nur als Befehlsempfänger und benutzt keinen eigenen Scheduler um Checks zu planen. Im Vergleich zum Satelliten werden somit weniger Ressourcen beansprucht.
Die Einrichtung selbst erfolgt wie beim Satelliten über die Pakete bei Linux und über einen Installer bei Windows-Systemen. Um die Verbindung zwischen den Instanzen herzustellen, müssen die Zonen-Dateien und die benötigten Zertifikate angepasst bzw. erstellt werden. Bei direkter Verbindung zum Core, kann dies über den Node Wizard geschehen, wodurch der Installationsaufwand sehr gering ausfällt. Da im Netz der Telekom Abfragen jedoch nur von Satelliten durchgeführt werden und die Clients somit keine direkte Verbindung zum Core besitzen, ist die Nutzung des Node Wizards nicht möglich. In diesem Fall müssen die Zertifikate über Konsolenbefehle erstellt und auf den entsprechenden Host verteilt werden. Außerdem müssen die Zonen-Dateien auf dem Host selbst, auf dem für den Host zuständigen Satelliten, sowie dem Core angepasst werden. Zudem muss das API-Feature aktiviert und die Option „accept_commands“ entsprechend gesetzt werden, damit Befehle von anderen Icinga 2-Instanzen angenommen werden. Damit die Befehle auf dem jeweiligen Remote-Rechner ausgeführt werden, muss in der Service-Definition noch der jeweilige Zonen-Endpunkt mit Hilfe des Attributs „command_endpoint“ definiert werden. Insgesamt ergibt sich ein relativ großer Installationsaufwand.
Im Normalfall besteht zwischen den Instanzen eine dauerhafte TLS-gesicherte Verbindung, durch welche die übergeordnete Instanz Befehle zur Ausführung eines lokalen Plugins an den Icinga 2-Client sendet. Dieser veranlasst die Ausführung und übermittelt das Ergebnis über die gleiche Verbindung zurück. Bezüglich der Sicherheit ergeben sich hier keinerlei Bedenken, da nur vordefinierte Abfragen durchgeführt werden können und bei einer Kompromittierung des Clients

oder des Satelliten das jeweils andere System nicht dadurch gefährdet ist.
Beim Testen des Icinga 2-Clients in einem Setup mit Core und Satelliten ergaben sich bezüglich des Loggings Probleme. So lief auf dem Core das für das Replay-Log zuständige Verzeichnis voll, obwohl alle Verbindungen ordnungsgemäß hergestellt waren (Vgl. Issue 9730[26]). Es werden in diesem Fall Events vorgehalten, weil der Client für den Core nicht direkt sichtbar ist. Somit können die gespeicherten Replays auch nie an die Instanz gesendet werden. Selbst in einem sehr kleinen Setup (2 Core-Instanzen, 2 Satelliten, 2 Clients) wurden so innerhalb von 10 Minuten bis zu 400 MB an Log-Dateien generiert. Umgehen lässt sich dieses Problem bisher nur durch das Festlegen des „log_duration"-Attributs bei der Definition der Endpoints. Diese muss allerdings aufgrund des rapiden Anstiegs der Log-Dateien so gering gewählt werden, dass das Replay-Log seinen Nutzen verliert. Bei einem Verbindungsverlust laufen auf beiden Icinga 2-Instanzen alle 5 Sekunden diesbezüglich Meldungen in den Log-Dateien auf. Da im Netz der Telekom mehrere tausend Hosts überwacht werden, ist die Wahrscheinlichkeit gegeben, dass auch eine Vielzahl an Host zu einem Zeitpunkt nicht erreichbar ist. Dadurch würde ein schneller Anstieg der Log-Dateien aufgrund der 5-sekündigen Schreibintervalle stattfinden.

3.4.7 Fazit

Um die Wahl einer entsprechenden Methode zur Abfrage von Remote-Hosts durchführen zu können, müssen verschiedenste Eigenschaften der jeweiligen Alternativen berücksichtigt werden. Die nach Meinung des Autors wichtigsten Faktoren werden deshalb vergleichend in Tabelle 8: „Vergleich der Remote-Abfragen" dargestellt.

	NRPE	check_by_ssh	NSClient++	NSCA	SNMP	Icinga 2-Client
Installations-aufwand	+	++	+	+	++	-
Komplexität Konfiguration für Abfragen	+	+	+	+	-	++
Aktiv-Abfragen	+	+	+	-	+	+
Passiv-Abfragen	-	-	-	+	+	-
Verschlüsselung	+	+	+	+	+/-	+
Authentifizierung	-	++	-	+	+/-	++
Sicherheit	+-	+-	+	-	+/-	++
Performance Gesamtsystem	+	+-	+	++	+	+
Plattform-unabhängigkeit	-	-	+-	-	+++	+
Gesamt	++	+++++	++++	++++	+++++++	++++++++

Tabelle 8: Vergleich der Remote-Abfragen

Beim Installationsaufwand sind NRPE, NSCA und der NSClient++ verhältnismäßig schnell einzurichten: Die Installation eines Daemons auf dem entfernten System sowie gegebenenfalls das Setzen von diversen Optionen wie Verschlüsselung müssen realisiert werden. Der Icinga 2-Client hingegen bedarf, zumindest bezüglich der Telekom-Netzstruktur, einen relativ hohen Aufwand, da hier zusätzlich noch Änderungen auf den verschiedenen Instanzen (Client, alle beteiligten Satelliten, Core) in den Zonen-Dateien vorzunehmen sind. Die besten Methoden sind bezüglich dieses

[26]Issue 9730 - „Split replay logs on Zone / Endpoint" https://dev.icinga.org/issues/9730

Gesichtspunktes SNMP und check_by_ssh, da diese meist standardmäßig installiert sind und somit keinen weiteren Aufwand darstellen.
Bei der Konfiguration der Abfragen sind die Remote-Abfragen relativ ähnlich. Die Konfiguration erfolgt jeweils über die Definition der Befehle in der entsprechend angewandten Programmiersprache und erfordert teilweise noch die Übergabe von diversen Parametern. SNMP wird dahingehend etwas komplexer eingeschätzt, da die OIDs als Zahlenfolge für den Menschen nicht unmittelbar verständlich sind. Icinga 2 ist in diesem Punkt die beste Wahl, da nur der Command-Endpoint beim jeweiligen Service eingetragen werden muss. Vorteilhaft ist in diesem Fall vor allem, dass kein zusätzlicher Syntax verstanden werden muss und so beispielsweise neue Mitarbeiter auch schneller angelernt werden können.
Die Mehrheit der untersuchten Programme setzt bei den Abfragen auf aktive Checks. Lediglich der Nagios Service Check Acceptor eignet sich nicht für derartige Prüfungen. Die einzige Methode, welche aktuell sowohl aktive als auch passive Checks ermöglicht, ist die Abfrage mittels SNMP (eine Möglichkeit zur Nutzung passiver Checks ist jedoch auch für Icinga 2.4 geplant).
Bezüglich der Verschlüsselung der Daten bieten alle genannten Remote-Abfragen zumindest eine dementsprechende Option an. Letztendlich bewegt sich diese bei den verschiedenen Abfragen auch in einem die Sicherheit betreffend angemessenen Bereich. Lediglich bei SNMP ist hier eine Einschränkung zu treffen, da diese Funktionalität erst in SNMPv3 integriert ist und die Daten bei den Vorgängerversionen im Klartext übertragen werden.
Große Unterschiede beim Vergleich ergeben sich hinsichtlich der verwendeten Authentifizierungsmechanismen. Eine akzeptable Authentifizierung bieten hier nur SNMP, NSCA, check_by_ssh und der Icinga 2-Client. Bei allen anderen Varianten ist keine Authentifizierung vorgesehen. Beim NSCA und SNMP geschieht diese über Passwort bzw. Benutzerkonten, was keine optimale Lösung ist. Außerdem ist bei SNMP die Authentifizierung erst in der Version 3 implementiert. Mehr Sicherheit bieten dahingehend der Icinga 2-Client und Check-Abfragen per SSH aufgrund der Nutzung von Zertifikaten.
Für die Beurteilung der Sicherheit im Ganzen wurde zusätzlich zur Authentifizierung und Verschlüsselung das Verhalten bei Kompromittierung eines Systems und die möglichen Auswirkungen auf das jeweilige andere System untersucht. Am kritischsten ist die Kompromittierung eines NSCA-Systems einzuschätzen, da hier Befehle vom Host an den Core gesendet werden können, die das Monitoring-System massiv beeinflussen können. Die anderen Abfrage-Methoden besitzen diverse Methoden um derartiges Verhalten zu unterbinden. Bei den meisten, wie beispielsweise NRPE oder die Prüfung per SSH ist nur die Kompromittierung des Monitoring-Systems kritisch. NRPE lässt zwar nur die Ausführung vordefinierter Befehle auf dem Host zu, besitzt jedoch eine Sicherheitslücke, die durch fachkundige Angreifer ausgenutzt werden kann, um auch andere Befehle auszuführen. Das gleiche Problem bietet sich bei SSH, da der Nutzer auf dem Host Rechte zum Ausführen benötigt und bei einer Kompromittierung des Cores aufgrund der ausgetauschten Schlüssel auch hier eine Gefahr besteht. SNMP ist in dieser Hinsicht auch vorsichtig zu behandeln, da hier Management-Befehle gesendet werden können. Auch da SNMP Version 3 noch nicht überall unterstützt wird, ist die Nutzung unter Umständen als kritisch anzusehen. Die größte Sicherheit ist mit dem Icinga 2-Client geboten. Zusätzlich zu der angemessenen Verschlüsselung und Authentifizierung ist es bei Kompromittierung der Monitoring-Instanz oder des Clients nicht möglich Manipulationen am jeweiligen Gegenpart durchzuführen, da generell nur Befehle zur Ausführung von Plugins und Check-Resultate übertragen werden können bzw. vor deren Ausführung geprüft werden.
Bezüglich der Performance des Gesamtsystems ergeben sich nur relativ geringe Unterschiede zwischen den getesteten Programmen. Bis auf den SSH-Check und den NSCA bewegen sich die Abfrage-Methoden annähernd im gleichen Bereich. Die Prüfungen mittels SSH weisen ein gewisses Defizit auf, da für jede Abfrage eine neue SSH-Verbindung aufgebaut werden muss. Dadurch kommt es zu einer geringfügig höheren CPU- und Netzwerkauslastung, welche mit zunehmenden Services mehr an Bedeutung gewinnt. Der NSCA ist in dieser Hinsicht als positiv herauszustellen, da hier Performance gespart wird, weil das jeweilige Monitoring-System den Check nicht initialisieren muss.

Lediglich die Auswertung der Prüfungen geschieht durch das Monitoring-System.
Hinsichtlich der Plattformunabhängigkeit erzielte SNMP das beste Ergebnis, da der Einsatz auf nahezu jedem Gerät möglich ist. Der Icinga 2-Client kann zumindest auf Windows- und Unix-Systemen betrieben werden, jedoch nicht auf hardwarenahen Geräten wie Routern oder Switchen. Gleiches gilt für den NSClient++, wobei die Implementation der Linux-Version noch verhältnismäßig unausgereift ist und dementsprechend selten eingesetzt wird. Alle anderen Programme sind jeweils nur auf Windows- oder Unix-Betriebssystemen einsetzbar.

Zusammenfassend ergibt sich nach Beurteilung der genannten Faktoren für den Icinga2-Client das beste Resultat. Nicht beachtet bei dieser Beurteilung wurde das übermäßige Logging-Verhalten, welches das Gesamtbild negativ beeinflusst. Diese Problematiken wurden allerdings als geringfügig eingeschätzt, da beispielsweise mittels Syslog-ng[27] bestimmte Meldungen gefiltert werden können, um diese in andere Dateien zu schreiben. Mit Hilfe eines Log-Rotates könnte diese Datei je nach Bedarf auf eine bestimmte Größe beschränkt und bei Überschreitung auslagert werden. Da durch den Icinga 2-Client jedoch Router oder ähnliche Hardware nicht ohne weiteres abzufragen ist, sollte im Netz der Telekom bezüglich der Remote-Abfragen eine Mischung aus Icinga 2-Client- und SNMP-Checks verwendet werden. Auf andere Abfrage-Methoden kann verzichtet werden, da durch diese Kombination alle gewünschten Funktionalitäten erfüllt werden können. Auch hinsichtlich der Wartbarkeit ist eine homogene Monitoring-Umgebung zu bevorzugen, weshalb auf die Verwendung weiterer Abfrage-Mechanismen verzichtet werden sollte.

3.5 Performance-Analyse

Einer der Hauptgründe für die Untersuchung der Migration liegt in der unzureichenden Performance von Icinga 1 im Netz der Telekom aufgrund der zahlreichen Abfragen. Hier wurden sehr hohe CPU-Auslastungen erreicht, wodurch die Grenzen für die Erweiterbarkeit des Systems schon bald erreicht sein werden.
Ob bei Icinga 2 mehr Abfragen durchgeführt werden können bzw. die CPU-Last geringer ausfällt, soll auf den folgenden Seiten untersucht werden.
Für die Untersuchung wurde das von Sven Nierlein entwickelte Programm „omd_utils“[28] verwendet. Vorraussetzung für die Nutzung des Programms ist eine vorhandene Installation der Open Monitoring Distribution.
Mit der Open Monitoring Distribution werden durch eine Installation mehrere Nagios-ähnliche Monitoring-Systeme bereitgestellt. Zudem besteht noch die Möglichkeit, Gearman für die jeweiligen Instanzen zu aktivieren.
Standardmäßig werden mit Hilfe des Makefiles der „omd_utils“ die Instanzen Nagios 3, Naemon, Icinga 1, Icinga 2, Nagios 3 mit Gearman und Naemon mit Gearman installiert. Für die Untersuchung der Migration wurde das Skript angepasst, da das Augenmerk auf Icinga 1 und 2 liegen sollte. Es wurden drei Instanzen erstellt: Icinga 1, Icinga 2, Icinga 1 mit Gearman. Die Untersuchung wird anhand dieser Instanzen stattfinden.
Die „omd_utils“ können sowohl einen Test der Core-Leistung erbringen als auch für graphische Nutzeroberflächen. Da sich diese Arbeit auf die zugrunde liegenden Systeme konzentriert, wird im Verlauf nur auf den Test der Monitoring-Instanzen und nicht auf verschiedene Weboberflächen eingegangen.
Für den Test werden die entsprechenden Cores aktiviert und verschiedenen simulierten Belastungsszenarien ausgesetzt. Dafür werden temporär Hosts und Services erstellt. Die folgenden Skripte sollen unterschiedliche Arten von Plugins für die Durchführung von Abfragen simulieren:

[27]Syslog-ng – Open Source Tool zum Verwalten von Log-Meldungen
[28]omd_utils – Benchmarktest für OMD-Installationen: https://github.com/sni/omd_utils

- „simple“ soll die Ausführung eines in C geschriebenen Plugins darstellen
- „simple.sh“ simuliert die Abarbeitung eines Shell-Skripts
- „simple.pl“ ist die Simulation eines kurzen Perl-Skripts
- „simple_epn.pl“ stellt die Ausführung eines kurzen Skripts mit embedded Perl dar
- „big.pl“ simuliert die Abarbeitung eines großen Perl-Plugins
- „big_epn.pl“ soll die Ausführung eines großen Skripts mit embedded Perl darstellen

Die Skripte wurden dabei mit Hilfe verschiedener „includes“ künstlich vergrößert, um die eigentliche Messung bzw. Berechnung der Check-Ergebnisse zu simulieren.
Durchgeführt wurden die Tests auf zwei verschiedenen Systemen: Der erste Test geschah auf einem virtualisierten Server mit 8 Cores und 16 GB RAM-Speicher. Der zweite Test wurde auf einem Server mit 32 Cores und 256 GB RAM-Speicher durchgeführt. Als Betriebssystem kam auf beiden Maschinen ein Debian 8 zum Einsatz.

3.5.1 Programmablauf

Beim Starten des Programms mit Hilfe des Befehls „make testall“ wird eine Datei „index.html“ erstellt, welche die im Test erzeugten Werte vorzugsweise mit einem Browser in Grafiken anzeigen lassen kann. Im entsprechenden Makefile sind sowohl die zu verwendenden Instanzen als auch die jeweiligen Plugins definiert.
Daraufhin wird das Benchmarkskript aufgerufen. Eine schematische Darstellung der wichtigsten Abarbeitungsschritte dieses Skripts ist in Anhang B: „Programmablaufplan“ ersichtlich.
Zu Beginn wird ein Plugin ausgewählt und die erste Monitoring-Instanz aktiviert. Für die Instanz und das jeweilige Plugin wird eine csv-Datei zur Speicherung der Resultate angelegt. Danach beginnt der Test mit den definierten Variablen (z.B. Serviceanzahl: 10). Daraufhin werden verschiedene Systemparameter, wie bspw. CPU-Idle, Check-Rate, Anzahl der Services mit Status „Pending“ und „OK“ ausgelesen und ausgegeben. Die für die Darstellung benötigten Werte werden der anfangs angelegten csv-Datei hinzugefügt.
Danach folgt eine Prüfung auf Abbruchbedingungen. Zu diesen zählen nachstehende Bestimmungen:

- mehr als 50% der Services besitzen den Status „Pending“
- weniger als 70% der Services befinden sich im Status „Pending“ oder „OK“
- CPU-Idle ist kleiner als 10%
- die Latenz[29] beträgt mehr als 10s
- 85% der Maximalrate wurden nicht erreicht
- 80% der erwarteten Rate wurden nicht erreicht

Mit diesen Bedingungen soll gewährleistet werden, dass der Test beendet wird, wenn das System seine Leistungsgrenze erreicht. Beispielsweise wird bei den ersten beiden Kriterien garantiert, dass der Core die Prüfungen im definierten Intervall bewältigen kann. Je mehr Services getestet werden, desto länger dauert es bis alle Services vom Status „Pending“ einer Prüfung unterzogen werden. Eine Fortführung des Tests mit zu vielen Services, die nicht geprüft werden können, wäre dementsprechend nicht sinnvoll. Die Abbruchbedingungen prüfen allerdings nicht nur den Core, sondern auch die Auslastung des Gesamtsystems. So ist das Kriterium für die CPU-Idle wichtig, da bei einer CPU-Auslastung von mehr als 90% die Grenze für die maximale Belastbarkeit fast erreicht ist und somit keine weiteren Services hinzugefügt werden müssen.
Ist keines der genannten Kriterien zu diesem Zeitpunkt erreicht, wird die Anzahl der durchzuführenden Service-Checks erhöht. Diese Erhöhung geschieht abhängig von der aktuellen Anzahl an Services. Je mehr Services vorhanden sind, umso mehr Services werden auch hinzugefügt, damit die Leistungsgrenze schnellstmöglich erreicht werden kann.

[29]Latenz – bezeichnet bei diesem Programm die Differenz zwischen geplanter und tatsächlicher Ausführungszeit der Services

Es folgt ein weiterer Testlauf mit den dazugehörigen Ausgaben und Aktualisierungen der csv-Datei bis zur Prüfung der Abbruchbedingungen.
Sollte eines der Kriterien erreicht werden, wird die genutzte Monitoring-Instanz beendet. Es wird daraufhin die nächste Instanz aktiviert und die Abarbeitung aller Arbeitsschritte auch für diese vollzogen. Sind alle Instanzen auf diese Weise beendet worden, folgt die Auswahl des nächsten Plugins und eine Wiederholung der Prozedur. Dies geschieht so oft bis auch das letzte Plugin mit allen Instanzen durchlaufen wurde.

3.5.2 Auswertung

Im Folgenden werden die durch das Programm erzeugten Diagramme für die jeweiligen Plugins analysiert und ausgewertet. Hierfür wird jeweils die Grafik abgebildet, welche den Vergleich zwischen den drei Instanzen darstellt. Die spezifischen Diagramme, die jeweils nur eine Instanz mit dem entsprechenden Plugin wiedergeben, sind in Anhang C: „Performancetest – Diagramme“ ersichtlich. Die Auswertung dieser spezifischen Grafiken wird in Zusammenhang mit den Vergleichs-Bildern geschehen.

Simple-Plugin

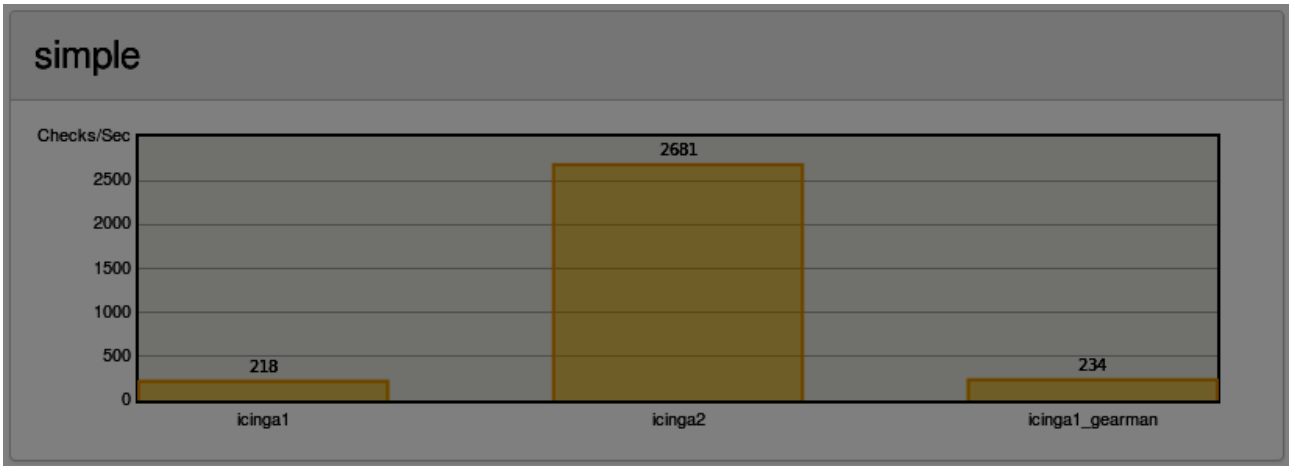


Abbildung 16: Vergleich der Instanzen bei 8 Cores mit simple-Plugin

Beim Test der Virtuellen Maschine mit 8 Cores durch Abarbeitung des simple-Plugins ergaben sich ungefähr gleiche Werte für Icinga 1 mit und ohne Gearman. Hier wurden etwas mehr als 200 Checks pro Sekunde ausgeführt. Icinga 2 hingegen erreichte mit einem Wert von über 2600 mehr als das 10-fache an Checks pro Sekunde. Diese enormen Unterschiede sind vor allem auf das bei Icinga 2 verwendete Multi-Threading zurückzuführen. Dadurch können mehrere Checks parallel initialisiert und auch ausgewertet werden. Icinga 1 arbeitet demgegenüber single-threaded und kann die Initialisierung und Auswertung nur nacheinander durchführen. Dies erklärt unter anderem auch den plötzlichen Einbruch der Check-Rate bei den Einzelgrafiken C.1 und C.2. Hier kommt es zu einem „Blocking“, bei dem zu viele Checks durch den Core verarbeitet werden müssen und dieser dadurch blockiert wird, weshalb weniger Checks pro Sekunde ausgeführt werden können. Ein weiterer Grund für die bessere Performance bei Icinga 2 ist die Tatsache, dass die Erzeugung von Kindprozessen standardmäßig durch „vfork()“ und nicht „fork()“ stattfindet (Vgl. [3], S.360).
Dass zwischen Icinga 1 mit und ohne Gearman kaum Unterschiede bestehen, liegt in der Tatsache, dass C-Plugins als Shell-Fork ausgeführt werden. Da kein Overhead durch eine zusätzliche Skriptsprache interpretiert werden muss, können die Ergebnisse sehr schnell zurückgegeben werden. Damit ist auch eine Aufteilung der Checks auf Gearman-Worker nicht viel schneller als die Abarbeitung durch den Icinga 1-Core selbst.

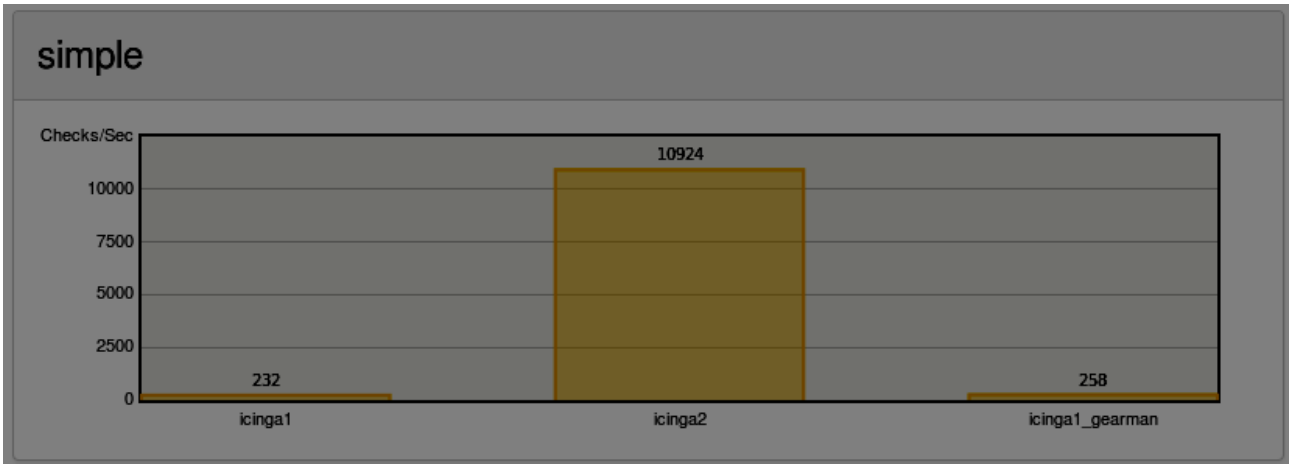


Abbildung 17: Vergleich der Instanzen bei 32 Cores mit simple-Plugin

Die Ausführung des Tests mit dem simple-Plugin auf einem System mit 32 Cores ergab bezüglich der Check-Rate keine Veränderung bei Icinga 1 mit und ohne Gearman. Lediglich die Anzahl der geprüften Services stieg von ca. 20.000 bei Icinga 1 ohne Gearman und knapp 30.000 bei Icinga 1 mit Gearman auf über 40.000 Services (Vgl. C.4 und C.5). Da die Instanz mit Gearman die Check-Ausführung an Worker abgibt, die eigene Prozesse für die Prüfung starten, müsste die Anzahl der möglichen Checks höher sein als bei der Instanz ohne Gearman. Da dies bei 32 Cores nicht mehr der Fall ist, ist davon auszugehen, dass die Obergrenze für die maximale Zahl an Service-Checks, die ein Icinga 1-Core verarbeiten kann, fast erreicht ist. Je nach in Kraft tretendem Abbruchkriterium bzw. Systemparametern sind noch geringe Abweichungen möglich. Bei Icinga 2 hingegen lässt sich sowohl bei der Service-Check-Rate als auch bei der Anzahl der Service-Checks eine Steigerung auf das 4-Fache verzeichnen (siehe C.3 und C.6). So stieg die Anzahl der Services von über 150.000 auf mehr als 600.000 und die Check-Rate auf fast 11.000 Checks pro Sekunde. Ursache für diese Steigerung ist ebenfalls das verwendete Multi-Threading. Der Icinga 2-Core nimmt sich so viele Ressourcen, wie verfügbar sind und kann deshalb mit 32 Cores 4 mal so viele Checks wie mit 8 Cores durchführen. Es ergibt sich somit eine direkte Abhängigkeit von Core-Anzahl und Service-Check-Rate sowie Service-Anzahl.

simple.sh-Plugin

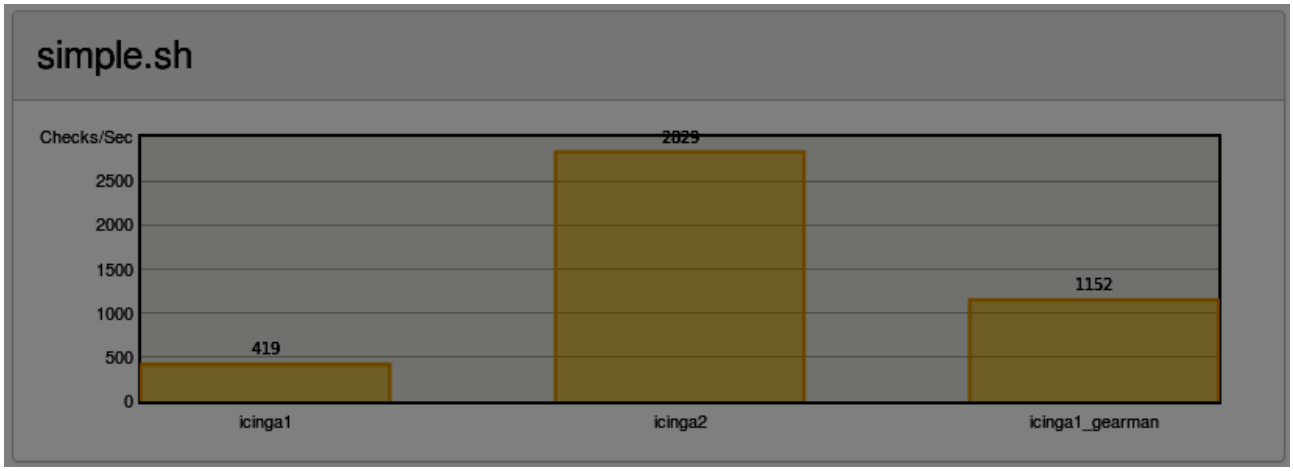


Abbildung 18: Vergleich der Instanzen bei 8 Cores mit simple.sh-Plugin

Das simple.sh-Plugin ergab für die Service-Check-Raten von Icinga 1 mit und ohne Gearman sehr unterschiedliche Steigerungen in Bezug auf das simple-Plugin. So besserte sich die Check-Rate noch relativ gering um 100 auf knapp über 400 Checks pro Sekunde. Bei der Icinga 1-Instanz mit Gearman hingegen ergab sich eine wesentliche Steigerung auf über 1100 Checks pro Sekunde. Hier zeigen sich die Vorteile der Nutzung von Gearman statt einer einfachen Icinga 1-Core-Installation. Durch die Auslagerung der Service-Checks an die Worker-Prozesse müssen die Berechnungen nicht vom Core durchgeführt werden. Da die Checks aufwendiger sind als beim C-basierten simple-Plugin, werden die Ergebnisse nicht so schnell zurückgegeben und der Core kann in dieser Zeit weitere Prozesse an die Worker geben. Dadurch können mehr Checks pro Sekunde vollzogen werden, was eine Steigerung der Service-Check-Rate bedeutet. Dass die Check-Rate für Icinga 1 ohne Gearman höher ist als bei der Ausführung des simple-Plugins kann nicht nachvollzogen werden. Eine Untersuchung der beiden Plugins mittels „strace“ ergab 27 Operationszeilen für das simple-Plugin und 101 Operationszeilen für das simple.sh-Plugin. Demnach sollte das in C geschriebene simple-Plugin zu einer größeren Check-Rate führen. Eine weiterführende Untersuchung dieses Sachverhalts soll jedoch nicht Bestandteil dieser Arbeit sein.
Icinga 2 weist bei diesem Plugin keine wirklichen Unterschiede zum simple-Plugin auf. Da jeder Service-Check per Multi-Threading ausgeführt wird, sind bei derart einfachen Plugins keinerlei Laufzeitunterschiede zu erkennen, weshalb auch die Anzahl der geprüften Services in etwa gleich bleibt (C.9 und C.12).

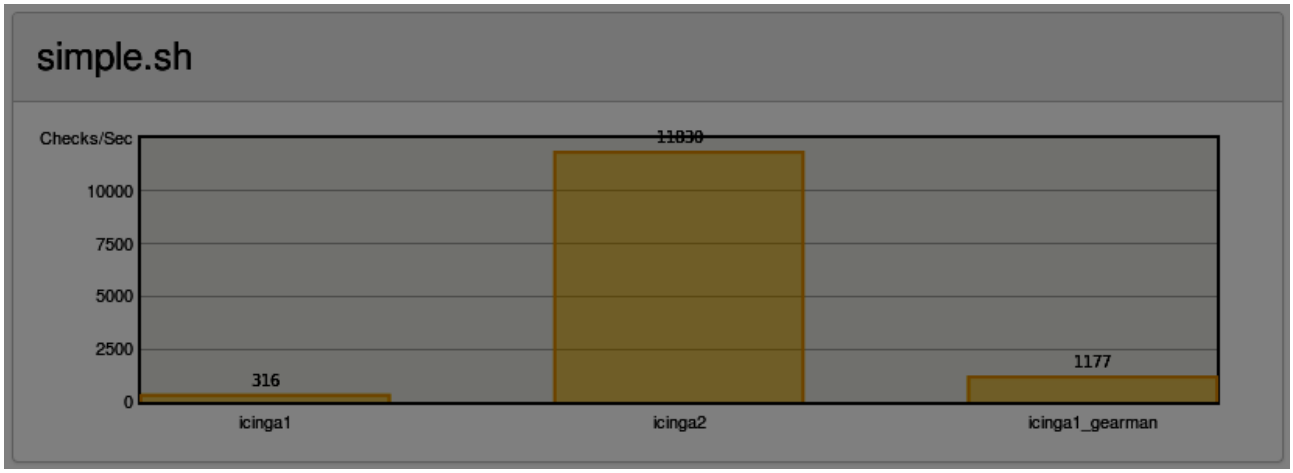


Abbildung 19: Vergleich der Instanzen bei 32 Cores mit simple.sh-Plugin

Das Ausführen des simple.sh-Plugins mit 32 Cores führte bei Icinga 1 mit und ohne Gearman abermals zur gleichen Check-Rate wie beim 8-Kern-System. Lediglich die Anzahl der Services

konnte minimal erhöht werden. Von jeweils 40.000 Services konnte die Anzahl auf knapp 60.000 bei Icinga 1 ohne Gearman und ca. 45.000 für die Instanz mit Gearman gesteigert werden. Dass die Zahl bei Icinga 1 mit Gearman geringer ist, hängt in diesem Fall nur von den Abbruchbedingungen ab, die von verschiedenen Systemfaktoren abhängig sind, auf die kein Einfluss genommen werden kann. Werden gleiche Zeitpunkte beim Test betrachtet, so ergeben sich ähnliche Werte für die beiden Instanzen (C.10 und C.11). Für Icinga 2 ergibt sich bei diesem Test das gleiche Verhalten wie beim simple-Plugin: 4-fache Anzahl an Kernen führt zu 4-facher Check-Rate und Service-Anzahl.

simple.pl-Plugin

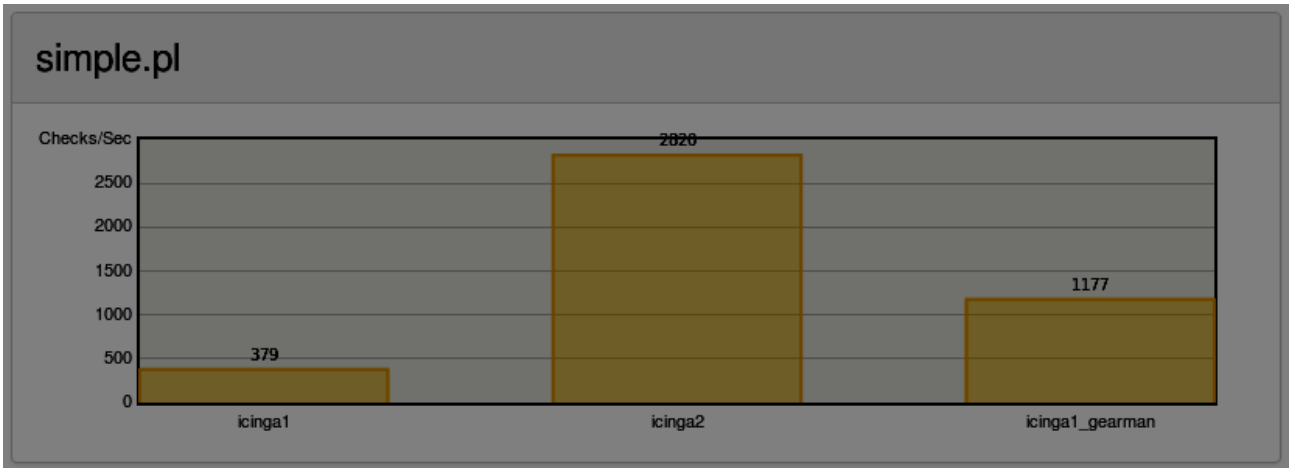


Abbildung 20: Vergleich der Instanzen bei 8 Cores mit simple.pl-Plugin

Beim simple.pl-Plugin ergibt sich für alle Instanzen bezüglich der Service-Check-Rate ein nahezu identisches Bild wie bei der Ausführung des simple.sh-Plugins. Lediglich bei Icinga 1 ohne Gearman ist der Wert geringfügig kleiner. Dies liegt an der zusätzlichen Berechnung für die Interpretierung des Perl-Codes. Icinga 1 erzeugt für Service-Checks Kind-Prozesse und kann erst nach der Verarbeitung der Ergebnisse fortfahren. Bei Icinga 1 mit Gearman und Icinga 2 ergeben sich keine Änderungen, da die Service-Checks in neuen Prozessen abgearbeitet werden. Während dies geschieht, kann der Monitoring-Core weitere Prozesse für Service-Checks erzeugen. Aufgrund der zusätzlichen Interpretation des Codes war die Service-Anzahl allerdings bei den beiden Icinga 1-Instanzen etwas geringer. Die Abbruchkriterien wurden demnach schon eher erreicht.

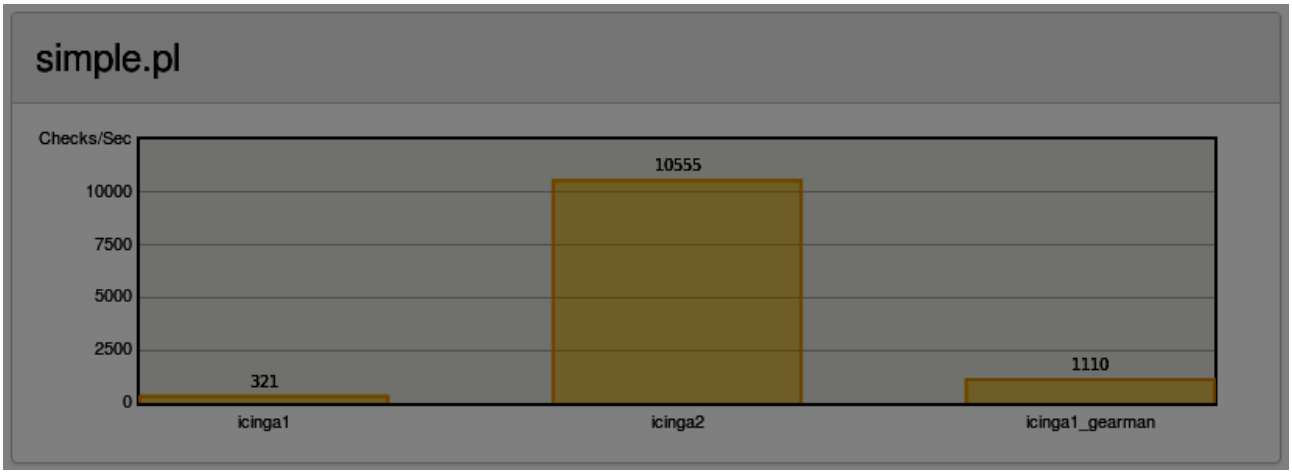


Abbildung 21: Vergleich der Instanzen bei 32 Cores mit simple.pl-Plugin

Generell bietet sich bei den Check-Raten wieder das gleiche Bild wie auch beim simple.sh-Plugin: Icinga 1 mit über 300 Checks pro Sekunde, Icinga 2 mit mehr als 10.000 und Icinga 1 mit Gearman mit über 1.000 Checks pro Sekunde. Die Service-Anzahl konnte durch die größere Zahl an Cores bei den Icinga 1-Instanzen wieder auf ca. 40.000 Services erhöht werden. Mehr Kerne konnten somit den größeren Overhead kompensieren, so dass die Abbruchbedingungen erst später eintraten.

simple_epn.pl-Plugin

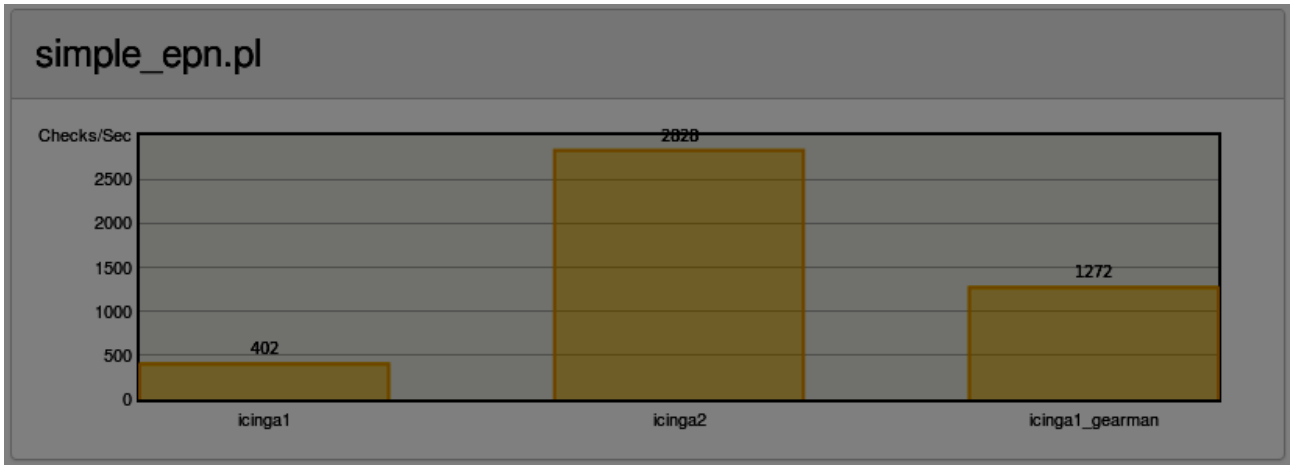


Abbildung 22: Vergleich der Instanzen bei 8 Cores mit simple_epn.pl-Plugin

Die Abarbeitung des Tests mit dem simple_epn.pl-Plugin führte zu einem minimal veränderten Bild im Vergleich zum simple.pl-Plugin. Beim embedded Perl wird bei jedem Aufruf des Plugins die C-API von Perl angesprochen, um dann das entsprechende Perl-Programm auszuführen. Dafür wird bei Icinga 2 bei jedem Aufruf ein Perl-Interpreter aus den zur Verfügung stehenden Bibliotheken erstellt und bei Beendigung des Plugins wieder beendet. Es wird demzufolge jedes mal Speicher allokiert, das Programm ausgeführt und Speicher wieder freigegeben. Icinga 1 sowie auch Gearman hingegen besitzen einen embedded Perl Interpreter, der dauerhaft zur Verfügung steht und nicht bei jedem Aufruf erstellt werden muss. Es wird nicht nur Speicher für die einmalige Ausführung des Plugins allokiert und wieder freigegeben, sondern ständig von Gearman bzw. Icinga 1 für den Interpreter reserviert. Dadurch können Plugins, die auf embedded Perl zugreifen, schneller abgearbeitet werden. Der Performance-Gewinn ist bei diesem Programm nur minimal, da es sich um ein sehr kurzes Programm handelt („strace“ ergab 225 Operationszeilen) und nur eine Bibliothek geladen werden muss.

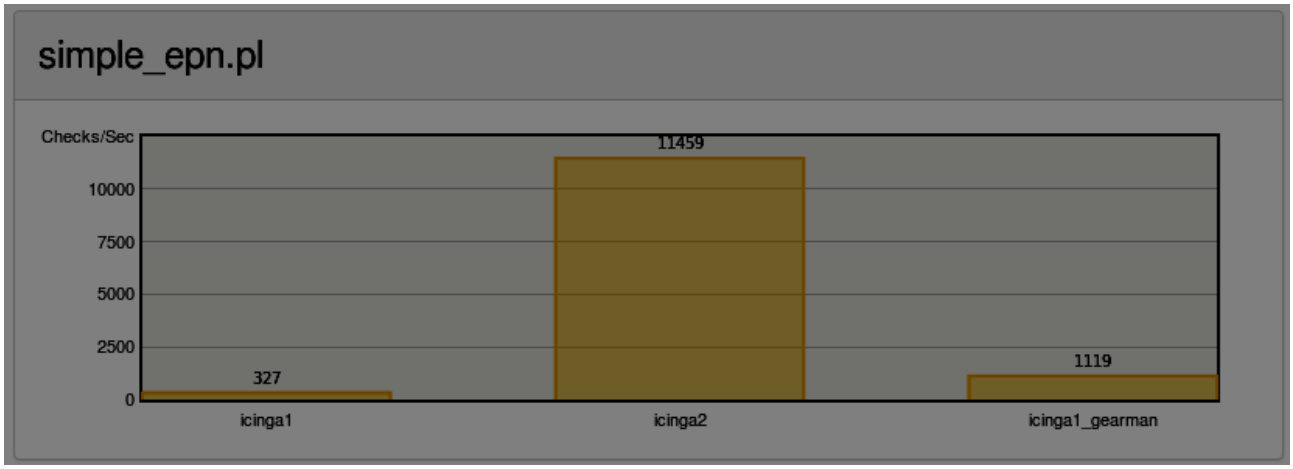


Abbildung 23: Vergleich der Instanzen bei 32 Cores mit simple_epn.pl-Plugin

Beim Ausführen des Plugins durch die verschiedenen Instanzen mit 32 Cores ergeben sich analoge Ergebnisse zu den vorhergehenden Tests. Sowohl bei Service-Check-Rate und Service-Anzahl sind die Werte für alle Instanzen annähernd auf dem gleichen Niveau wie beim simple.pl- Plugin. Bei der Service-Anzahl ist wieder ein geringer Anstieg bei Icinga 2 zu erkennen, was durch das bereits kompilierte Skript und gegebenenfalls durch Momentanwerte des Systems zu erklären ist.

big.pl-Plugin

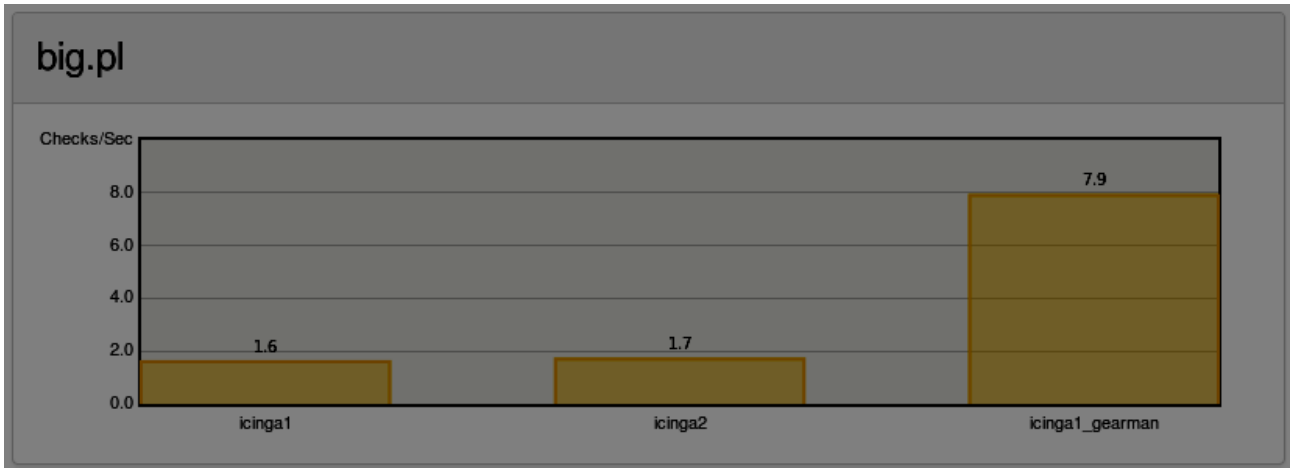


Abbildung 24: Vergleich der Instanzen bei 8 Cores mit big.pl-Plugin

Die Abarbeitung des big.pl-Plugins lieferte für Icinga 1 ohne Gearman und Icinga 2 mit 1,6 und 1,7 Checks pro Sekunde in etwa gleiche Ergebnisse. Hier konnte die Icinga 1-Instanz mit Gearman ein wesentlich besseres Ergebnis erzielen. Es konnten hier bis zu 7,9 Checks pro Sekunde erreicht werden. Auch bei der Service-Anzahl ergaben sich sehr große Unterschiede. Während bei Icinga 1 ohne und mit Gearman 200 bzw. über 400 Services geprüft wurden, konnten bei Icinga 2 nur knapp 150 Services geprüft werden. Insgesamt entspricht dieses Ergebnis nicht den Erwartungen. Auch eine mehrfache Wiederholung des Tests führte jedoch zu ähnlichen Resultaten. Aufgrund des Multi-Threadings sollte Icinga 2 ähnliche Werte wie die Icinga 1-Instanz mit Gearman liefern. Dass dies nicht der Fall ist, könnte am big.pl-Skript selbst liegen. Hier werden komplette Perl-Frameworks (Moose und Catalyst) geladen und viele externe Abhängigkeiten geschaffen, die massiv In- und Output erzeugen. Es ergibt sich ein riesiges Skript, weshalb auch die CPU-Auslastung in den Einzelgrafiken (C.25, C.26, C.27) sehr hoch ist. Eine weitere Untersuchung dieser Anomalie dieses Diagramms wird nicht geschehen, da das Analysieren von Skripten nicht im Fokus dieser Arbeit steht.

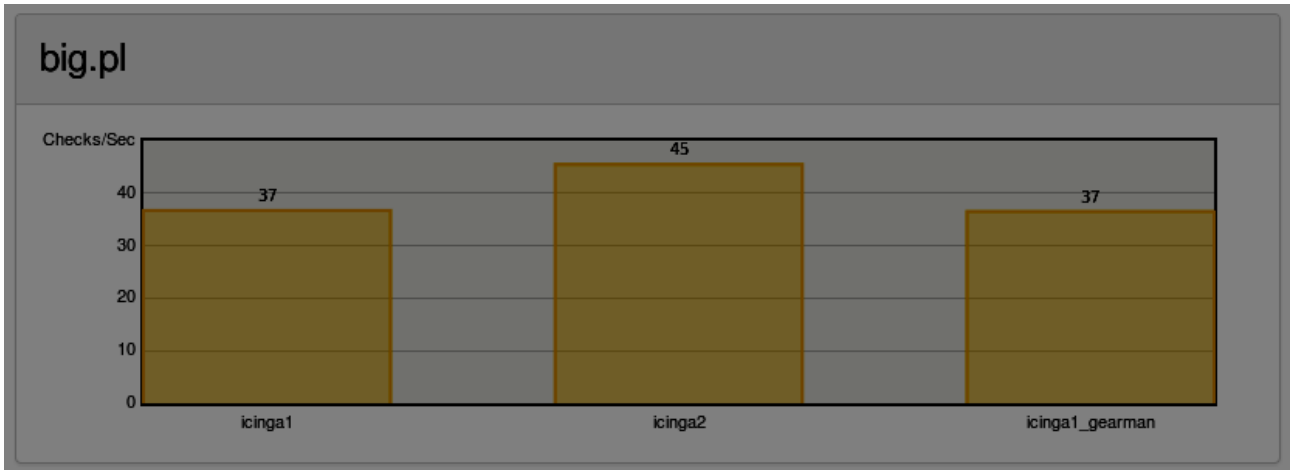


Abbildung 25: Vergleich der Instanzen bei 32 Cores mit big.pl-Plugin

Der Test des big.pl-Plugins mit dem 32 Core-System ergab ein sehr verändertes Bild. Hier liefert Icinga 2 mit einer Check-Rate von 45 Checks pro Sekunde den besten Wert. Icinga 1 ohne und mit Gearman erreichten beide bis zu 37 Checks pro Sekunde. Wie in den Einzelgrafiken zu erkennen (C.28, C.29, C.30), lagen Icinga 2 und Icinga 1 mit Gearman bei der Anzahl der getesten Services etwa gleichauf mit ca. 3000 Services. Icinga 1 ohne Gearman erreichte lediglich 2000 Services. Es zeigte sich bei diesem Test, dass bei einem extrem I/O[30]-lastigen Plugin Icinga 2 nur noch geringfügig

[30]I/O – Input/Output; Ein- und Ausgabe-Operationen

schneller ist als Icinga 1. Bei einem solchen Plugin dauert die Ausführung so lange, dass die Weitergabe und das Verarbeiten der Checks von den Gearman-Workern so viel Zeit beansprucht, dass der Gearman gegenüber der Icinga 1-Instanz ohne Worker keinerlei Vorteile bezüglich der Geschwindigkeit liefert.

big_epn.pl-Plugin

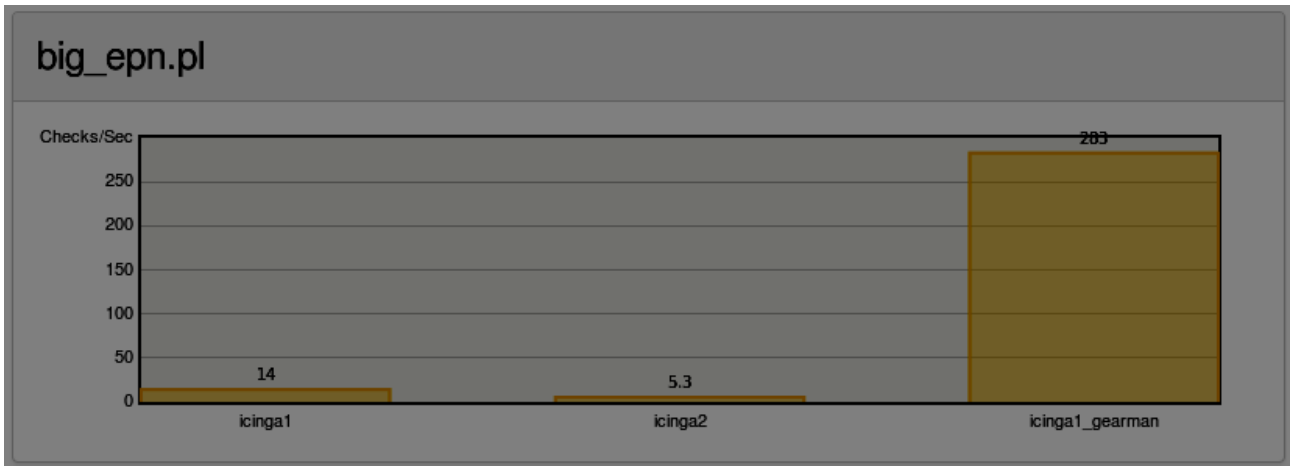


Abbildung 26: Vergleich der Instanzen bei 8 Cores mit big_epn.pl-Plugin

Das big_epn.pl-Plugin ist ein umfangreiches Plugin mit embedded Perl-Code. Der Test mit dem 8 Core-System ergab für Icinga 1 eine Check-Rate von 14 Checks pro Sekunde. Icinga 2 erreichte bei diesem Test lediglich 5,3 Checks pro Sekunde. Den mit Abstand besten Wert erzielte Icinga 1 mit Gearman. Hier konnten über 280 Checks pro Sekunde ausgeführt werden. Hier zeigt sich deutlich, dass dank des embedded Perl Interpreters das Skript von Icinga 1 mit und ohne Gearman deutlich schneller abgearbeitet werden kann. Die Instanz mit Gearman liefert mit Abstand die beste Check-Rate, da zusätzlich zum embedded Perl Interpreter durch das Worker-Design eine Art Multi-Threading verwirklicht wird. Icinga 2 kann trotzdessen keine höheren Check-Raten erreichen, da das big_epn.pl-Plugin sehr groß und die Abarbeitung ohne zusätzlichen Interpreter sehr zeitaufwändig ist.

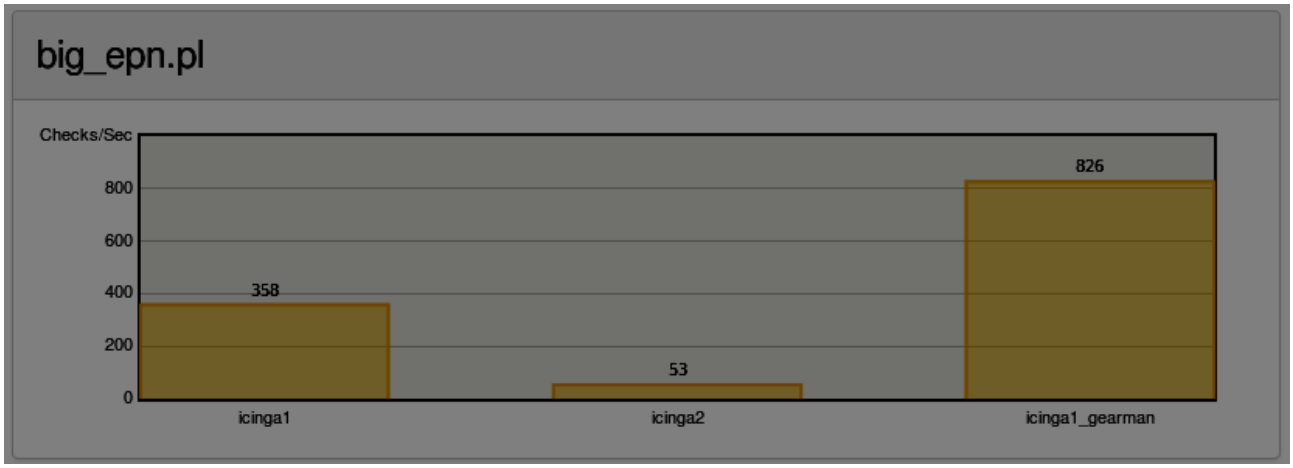


Abbildung 27: Vergleich der Instanzen bei 32 Cores mit big_epn.pl-Plugin

Das Abarbeiten des big_epn.pl-Plugins mit 32 Cores führte bei Icinga 1 ohne Gearman und Icinga 2 zu enormen Steigerungen der Check-Raten. Hier zeigte sich eine überproportionale Erhöhung der Geschwindigkeit. Bei Icinga 1 mit Gearman erfolgte auch ein Anstieg der Check-Rate. Dieser fiel jedoch in Relation zu den anderen Instanzen wesentlich geringer aus. Hier ist davon auszugehen, dass

eine systeminterne Grenze erreicht wurde bzw. bald erreicht wird. Eine weitere Erhöhung der Core-Anzahl würde in diesem Fall kaum oder keine weiteren Geschwindigkeitsgewinne erzeugen.

3.5.3 Fehlerbetrachtung und Fazit

Um die Relevanz dieses Tests beurteilen zu können, gilt es zu prüfen, welche Faktoren Einfluss auf das Ergebnis haben können und welche Fehler durch die Struktur des Programms bzw. des Ablaufs möglich sind.
Eine Einschränkung ist natürlich, dass alle Tests auf einem System durchgeführt werden. In der Wirkumgebung werden Hosts in anderen Netzen abgefragt. Auch kommen dort Satelliten bzw. Worker auf entfernten Maschinen zum Einsatz, welche bei diesem Test auf dem gleichen System laufen. Somit werden einerseits der Netzwerkverkehr sowie dadurch anfallende Verzögerungen nicht berücksichtigt. Zum anderen übernimmt der Core im Netz der Telekom kaum Abfragen; diese werden durch verschiedene Server mit Workern ausgeführt. Durch die Verteilung der Abfragen kann letztendlich mit mehrerer Server mehr Rechenleistung zur Verfügung gestellt werden. Da bei Icinga 2 bei den meisten Plugins eine direkte Abhängigkeit zwischen der Check-Rate und Rechenleistung erkennbar war, kann davon ausgegangen werden, dass unter realen Bedingungen noch höhere Werte erreicht werden können. Bei den Icinga 1-Instanzen ist davon nicht auszugehen, da die Core-interne Verarbeitung bei den meisten Tests scheinbar als Leistungsgrenze für das jeweilige System erreicht wurde.
Generell sind viele Werte, welche durch das Programm erzeugt werden, auch mit einer gewissen Ungenauigkeit behaftet. Diese ist auf verschiedene Faktoren zurückzuführen. Zum einen sind hier die systemspezifischen Parameter zu nennen: CPU-Auslastung oder andere Systemwerte sind nicht nur vom Programm selbst abhängig; so können auch betriebssysteminterne Prozesse beispielsweise kurzzeitige Auslastungen verursachen, welche zu kleinen Abweichungen führen. Zum anderen bieten verschiedene Werte, die im Performancetest definiert sind, Möglichkeiten zu gewissen Differenzen bei den Ergebnissen. Zum Beispiel werden die Services um vorgegebene Zahlen erhöht (beginnend von 10 bis hin zu 10.000). Somit ist nicht mit Sicherheit gegeben, dass die Leistungsgrenze des Systems exakt getroffen wird. Gleiches gilt für die Abbruchbedingungen: Sie sind willkürlich gewählt und bieten keinerlei Sicherheit, dass die maximale Leistung für das System erreicht werden kann und der Test nicht minimal zu früh abgebrochen wird. Die für die Abbruchkriterien gewählten Werte sind dennoch als sinnvoll zu erachten, geben jedoch keine Garantie für große Genauigkeiten. Da letztendlich alle Tests mit den gleichen Werten durchgeführt werden, ist eine Vergleichbarkeit gegeben. So zeigte sich auch bei Wiederholung der Tests, dass die Abweichungen unter 1% lagen und somit ein akzeptables Ergebnis aufweisen.
Zu relativieren sind die Ergebnisse bezüglich des big.pl-Plugins. Hier werden zwei komplette Perl-Frameworks geladen, wodurch viele Abhängigkeiten erzeugt werden. Dies ist in realen Umgebungen eher selten vorzufinden. Des Weiteren ist auch die Verwendung von embedded Perl aufgrund der dort benutzten C-API kaum noch in Systemen aufzufinden. Diese C-API ist mit Bugs und Memory Leaks[31] behaftet (Vgl. [14] Abschnitt „embedded Perl; [19]). Speziell auf den OSS-Bereich der Telekom bezogen wird kein derartiges Plugin in der Wirkumgebung verwendet.
Insgesamt legt das Programm auch einen großen Fokus auf Perl-Programme. Wie aus Tabelle 1: „Check-Plugins“ ersichtlich, sind aktuell ledliglich drei SNMP-Plugins im Einsatz, die auf Perl basieren. Eine weitere Einschränkung ist aufgrund der homogenen Umgebung zu treffen. Es wird jeweils nur ein Plugin simuliert. Heterogene Umgebungen reagieren teilweise auf eine andere Art. Optimal wäre eine Nutzung von vielen Plugins (Python, Perl, C, etc.) auf unterschiedlichen Diensten (SNMP, SSH, lokale Dienste, etc.). Dies würde ein sehr wirklichkeitsnahes Ergebnis liefern, wodurch eine höhere Relevanz der Resultate gegeben wäre.
Dennoch ist dem Programm ein ausreichender Informationsgewinn bezüglich der Performanz der

[31]Memory Leak – Fehler in der Speicherverwaltung: Arbeitsspeicher wird belegt, aber nicht genutzt oder freigegeben

verschiedenen Systeme zu entnehmen. Die von der Telekom für Abfragen verwendeten Plugins enthalten eine geringe Anzahl an Programmzeilen, weshalb die Tests mit den ersten Plugins dahingehend eine akzeptable Simulation darstellen. Insbesondere in Anbetracht der eindeutigen Ergebnisse lassen sich Tendenzen für wirkliche Systeme ableiten. Demnach ist Icinga 2 in dieser Beziehung deutlich performanter und erreichte im Test mit kleinen Plugins bisher noch kein Limit. Bei Icinga 1 war dies bereits mit 8 Cores und dementsprechend wenigen Services erreicht. Auch trotz Nutzung von Gearman waren die erlangten Ergebnisse weitaus schlechter als bei Icinga 2. Zudem ist bei Icinga 2 die Performanz von der Core-Anzahl abhängig. Eine entsprechend große Dimensionierung wird deshalb zu sehr guten Ergebnisse führen. Bei einer virtualisierten Lösung können so auch bei in ferner Zukunft auftretenden Engpässen in dieser Hinsicht gegebenenfalls noch weitere Cores hinzugefügt werden, um die Leistung wieder auf ein für den Nutzer angemessenes Niveau zu bringen. Hinsichtlich der bereits momentan kritischen Auslastung des in der Wirkumgebung verwendeten Icinga 1-Systems ist eine Migration auf Icinga 2 demnach als sinnvoll zu erachten.

3.6 Hochverfügbarkeit

Das Monitoring erhält aufgrund immer größer werdender Netzwerke zunehmend an Bedeutung, da hierdurch Fehler schnellstmöglich bemerkt und die Produktivität schneller wiederhergestellt werden kann. Entsprechend wichtig ist auch das Zusichern der Verfügbarkeit bzw. Ausfallsicherheit des jeweiligen Monitoring-Systems. Um diese Hochverfügbarkeit sicherzustellen, können verschiedene Ansätze genutzt werden. Dabei gibt es sowohl kommerziell lizensierte Programme als auch Open-Source-Produkte. Kommerzielle Programme, die derartige Funktionen bereitstellen, sind beispielsweise Oracle Clusterware oder Veritas Cluster Server. Da die Telekom für derartige Software aktuell keinerlei finanzielle Mittel bereitstellen möchte, werden im Folgenden nur Open-Source-Produkte untersucht. Hierbei wurden aufgrund der Empfehlungen des Linux-HA-Projects sowie der Tatsache, dass diese Varianten schon in verschiedenen Monitoring-Systemen der Telekom in Verwendung sind, folgende Möglichkeiten untersucht: Hochverfügbarkeit mittels Distributed Replicated Block Device (DRBD) Version 8 mit Pacemaker und Hochverfügbarkeit durch Xen-Hypervisor mit einem Netapp Storage. Zudem wurde auch die Verwendung des Icinga 2-Clusters getestet, um zu prüfen, ob dadurch auf externe Programme verzichtet werden kann.

3.6.1 DRBD mit Pacemaker

Distributed Replicated Block Device (DRBD) ist eine Software, um Dateien auf verschiedenen Servern synchron zu halten. Hierzu wird auf jedem der Rechner ein spezielles Blockgerät[32] erstellt, welches eingebunden und auf einem der Server aktiviert werden muss. Alle auf diesem primären Server geschriebenen Daten werden dann automatisch über das Netzwerk auch auf den sekundären Server transferiert. Erst wenn dieser die Daten erfolgreich geschrieben hat, meldet er dies an den primären Server zurück. Danach kann dieser mit weiteren Schreibvorgängen fortfahren. In Abbildung 28: „DRBD-Funktionsweise“ wird diese Verfahrensweise dargestellt.

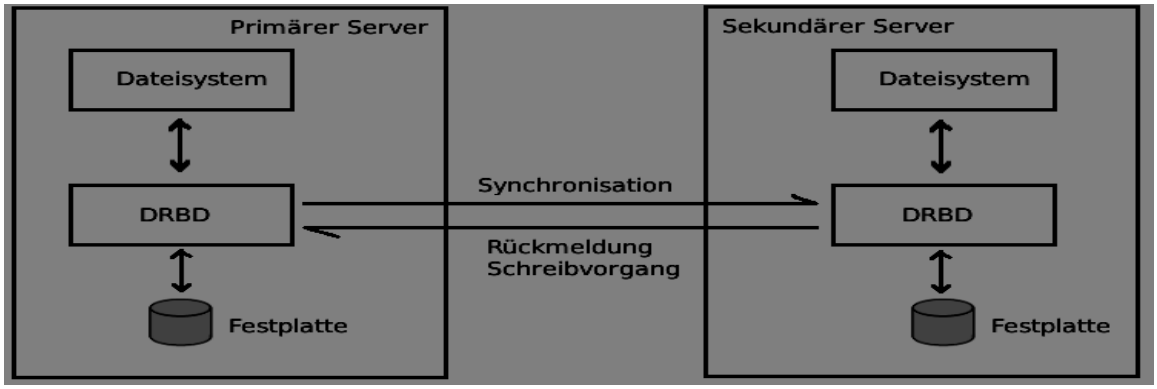


Abbildung 28: DRBD-Funktionsweise

Es ist jedoch auch möglich DRBD asynchron zu betreiben. Das heißt, es wird nicht auf eine Rückmeldung des Schreibvorgangs auf dem zweiten Server gewartet bis der nächste Schreibvorgang gestartet wird.
Damit eine Hochverfügbarkeit gegeben ist, wird weitere Software zur Überwachung der Kommunikation zwischen den Servern und dem Managen der verschiedenen Ressourcen benötigt. Die Überwachung der Kommunikation erfolgt über Corosync, einem sogenannten Cluster Communication Manager. Für die Überwachung und Steuerung der verschiedenen Dienste wird hierbei ein Cluster Ressource Manager genutzt. Dafür wird in einem solchen Setup im Bereich der Telekom Pacemaker genutzt.
Die Nutzung von DRBD / Pacemaker mit dem Icinga 2– Dienst bedingt ein Aktiv-/Passiv-Cluster.

[32]Blockgerät – Geräte, die Daten in Datenblöcken übertragen (z.B. Festplatten)

Das bedeutet, der primäre Server ist für die Ausführung von Aufgaben verantwortlich. Der sekundäre Server hingegen übernimmt keinerlei Funktionalitäten; er befindet sich im Hot Standby[33] und dient als Backup-Server. Fällt der primäre Server aus, so erkennt Corosync dies. Infolgedessen kann durch Pacemaker ein automatischer Failover[34] stattfinden: Der zweite Server wird durch Pacemaker als primärer Server festgelegt. Danach werden alle überwachten Prozesse auf diesem wieder gestartet. In Abbildung 29: „DRBD Failover“ wird dieser Vorgang beispielhaft veranschaulicht.

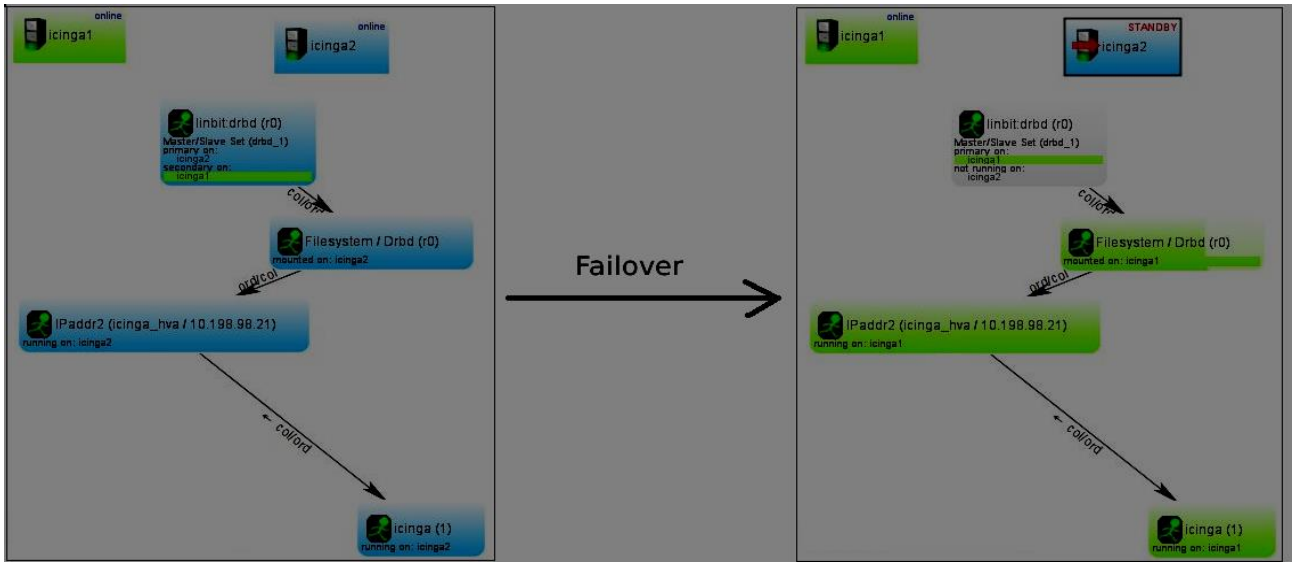


Abbildung 29: DRBD Failover

Hier ist zuerst der Server „icinga2“ als primäre Instanz aktiv und stellt verschiedene Ressourcen wie beispielsweise IP-Adresse und Icinga-Dienst zur Verfügung. Simuliert man nun einen Ausfall von „icinga2“ indem man diesen auf Standby setzt, so schwenken die Dienste, welche über Pacemaker überwacht und deren Daten durch DRBD synchronisiert werden, automatisch auf „icinga1“. Bei einem derartigen manuellen Wechsel, auch als „Switchover“ bezeichnet, werden auf dem primären Server alle Prozesse erst beendet und dann auf dem sekundären Server wieder gestartet. Deshalb gibt es einen minimalen Ausfall der jeweiligen Ressourcen (im Sekundenbereich). Ein Failover führt bis auf das Beenden der Prozesse durch Pacemaker zu einem identischen Verhalten. Sind die beiden Instanzen wieder miteinander verbunden, werden alle auf dem aktiven System geschehenen Änderungen per DRBD auf das andere System synchronisiert. In diesem Szenario müssen auf beiden Systemen die jeweiligen Dienste installiert sein. Es schwenken in diesem Fall ledlich konfigurationsspezifische Dateien.

3.6.2 Xen-Hypervisor mit Netapp Storage

Diese Variante zur Sicherstellung der Hochverfügbarkeit entspricht dem Aufbau in Abbildung 3: „Anschaltkonzept“.
Ein Netapp Storage besteht aus vielen physikalischen Festplatten. Diese werden zu RAID-Gruppen zusammengefasst. Verwendet wird hierfür ein RAID DP (näher beschrieben in Anlage D: „RAID DP“). Diese RAID-Gruppen wiederum werden zu einem „Aggregate“ zusammengefasst. Aus diesem können nun mehrere logische Festplatten, so genannte Volumes, erstellt werden. Das Netapp Storage besitzt zwei Controller, die für die Steuerung zuständig sind. Fällt einer der beiden Controller aus, so erfolgt ein Übertragen der Daten auf den anderen Controller.
Auf den beiden Xen-Servern läuft jeweils ein CentOS 5.5 als Betriebssystem. Durch das

[33]Hot Standby – Beim Ausfall eines Servers übernimmt der andere automatisch die Aufgaben (Anstoß durch externes Programm oder Administrator nicht nötig)
[34]Failover – ungeplanter Wechsel von Netzwerkdiensten bei Ausfall einer Instanz

Betriebssystem wird auf dem Netapp Storage mittels NFS[35] ein Volume gemounted (in Abbildung 30: „Xen-Hypervisor“ als „V1“ bezeichnet). Auf diesem gemounteten Volume werden die verschiedenen VMs in Form von VHD[36]-Dateien gespeichert. Dafür sorgt der auf den Servern installierte Xen-Hypervisor[37]. Dieser verwaltet die verschiedenen VMs. Auf den Xen-Servern liegen lediglich die Metadaten (Name, Beschreibung, ID, Konfiguration, etc.) der diversen VMs. Diese Metadaten werden durch den Hypervisor ständig synchronisiert. Jeder Server weiß somit von allen Virtuellen Maschinen. Diese Synchronisation ist gleichzeitig eine Art Heartbeat[38]. Zusätzlich zu dieser Prüfung der Verbindung wird durch das Betriebssystem ein zweites Volume (V2 in Abbildung 30) gemounted, auf dem sich Statusdateien für die Server befinden. Es wird regelmäßig ein Heartbeat in die Statusdateien geschrieben.

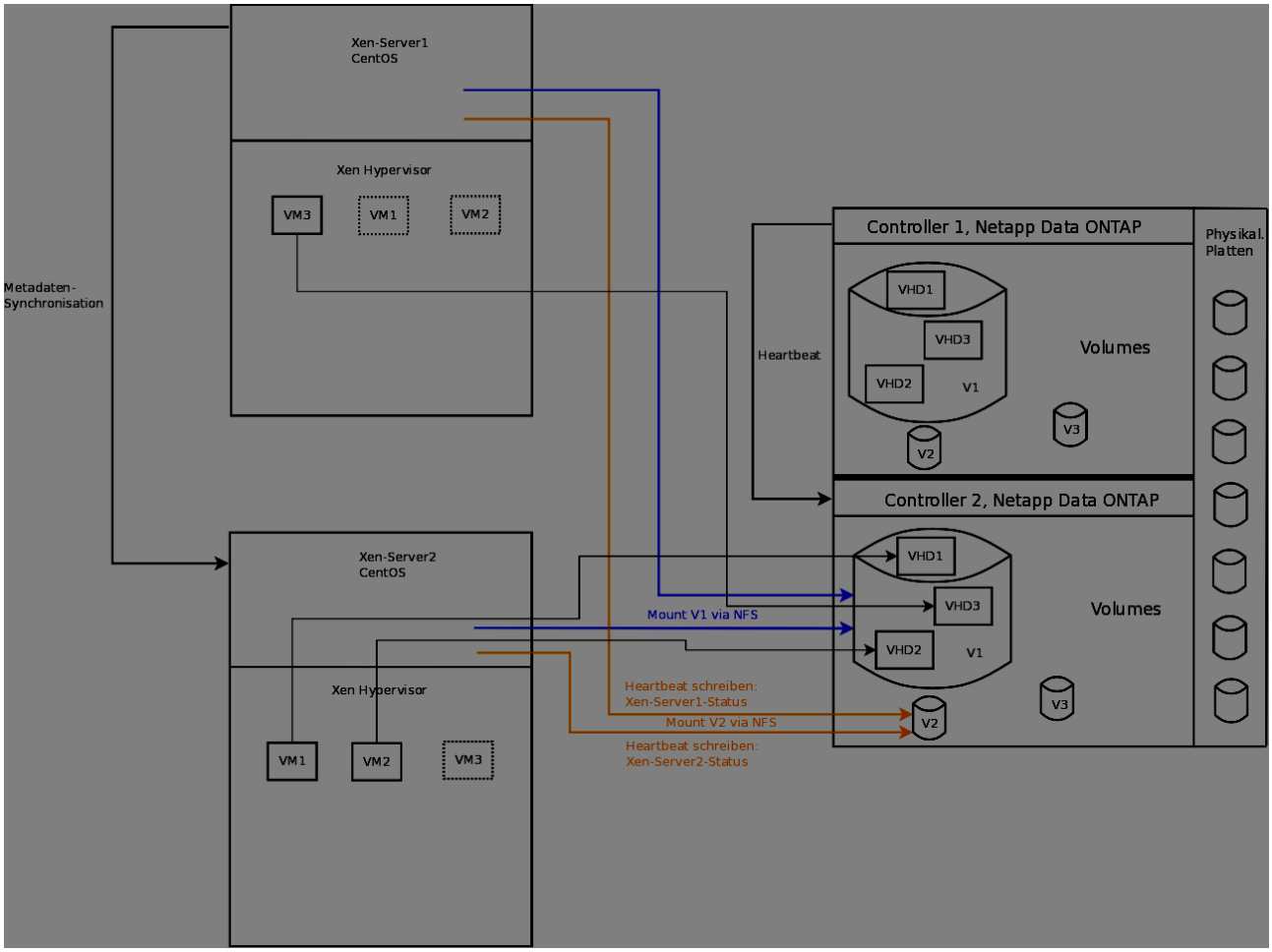


Abbildung 30: Xen-Hypervisor

Fällt nun einer der beiden Server aus, bemerkt dies der andere aufgrund der ausbleibenden Metadaten-Synchronisation. Damit sichergestellt werden kann, dass nicht nur die Verbindung der Server untereinander gestört ist, erfolgt ein Abgleich der Statusdateien auf dem zweiten Volume. Wird dort festgestellt, dass der Server nicht mehr in seine Statusdatei schreibt, beginnt der Failover-Mechanismus. Sollte der Server Master gewesen sein, so wird ein neuer Master festgelegt. Danach wird bestimmt, auf welchem Xen-Server die VMs des ausgefallenen Servers wieder gestartet werden sollen. Da im dargestellten Setup sich nur zwei Server im Cluster befinden, allen sowohl die Wahl des neuen Masters als auch der Ort zum Starten der VMs auf den jeweils anderen Server. Im nächsten

[35]NFS – Network File System
[36]VHD – Virtual Hard Disk (Virtuelle Festplatte)
[37]Hypervisor – Software zum Managen von Virtuellen Maschinen
[38]Heartbeat – ständiger Austausch von Nachrichten, um zu prüfen, ob die Verbindungspartner verfügbar sind

Schritt werden die benötigten VHD-Dateien gemounted und daraufhin die Virtuelle Maschine gestartet. Diese Abfolge wird für alle ausgefallenen VMs wiederholt bis diese auf einem anderen Server gestartet wurden.

3.6.3 Icinga 2-Cluster

Bei der Entwicklung von Icinga 2 wurde viel Wert darauf gelegt, eine Monitoring-Lösung zu erstellen, die alle benötigten Funktionalitäten in einem Programm vereint. Deshalb wurde auch das Cluster-Feature eingeführt, um eine gewisse Hochverfügbarkeit zu gewährleisten. Die Konfiguration geschieht hierbei über die Zonen-Definition. Werden mehrere Instanzen zur Master-Zone hinzugefügt, können diese bei Aktivierung der gewünschten Features die Aufgaben einer ausgefallenen Instanz übernehmen. Die zwischen den Instanzen aktive Verbindung wird mit einem Heartbeat im 5-Sekunden-Intervall geprüft. Fällt eine Instanz aus, erfolgt ein automatischer Failover für die Datenbankverbindung, das Versenden von Notifications sowie die Ausführung von Check-Abfragen. Sowohl Notifications als auch Check-Abfragen können bei diesem Setup load-balanced betrieben werden. Es handelt sich deshalb um ein Aktiv-/Aktiv-Cluster. Da alle Prozesse bereits auf den involvierten Maschinen laufen, ist für den Nutzer keine Ausfallzeit feststellbar.

3.6.4 Vergleich

Damit eine Bewertung der verschiedenen Hochverfügbarkeitslösungen gegeben werden kann, wurden Kriterien ausgewählt, die für derartige Systeme von Bedeutung sind. Diese werden in Tabelle 9: „Vergleich Hochverfügbarkeit“ dargestellt. Hier wurde eine Wichtung der Kriterien vorgenommen. Je höher der angegebene Wert, als umso wichtiger wurde das entsprechende Kriterium angesehen. Die Failover-Zeit und die Stabilität der verwendeten Soft- und Hardware wurden als besonders bedeutungsvoll erachtet, da das Hauptaugenmerk immer der Regelbetrieb sein muss. Haben Systeme lange Failover-Zeiten oder laufen instabil, ist dies nicht mehr gegeben. Ebenfalls wichtig, aber nicht so kritisch wie die beiden genannten Faktoren, sind Konfigurationsaufwand bzw. -komplexität sowie Datenverluste. Die Konfiguration ist in dem Sinne von Bedeutung, dass bei einer simplen Konfiguration sich die Fehleranalyse und Fehlerbehebung einfacher gestaltet und somit der Regelbetrieb schneller wieder aufgenommen werden kann. Datenverluste wurden ebenfalls als wichtig eingeschätzt, da ein Verlust von Daten immer mit Aufwand zur Wiederherstellung verbunden ist. Als weniger kritisch wurden die Faktoren Ressourcenauslastung und Kosten erachtet. Um eine gewisse Ausfallsicherheit zu erreichen, muss in Hardware und eventuell auch Software investiert werden. Natürlich sollte eine Wirtschaftlichkeit gewährleistet werden. Da Sicherheit jedoch zwangsweise Kosten verursacht, wurden diese Faktoren als unkritisch angesehen. Das Gesamtergebnis setzt sich aus der Summe der mit der jeweiligen Wichtung multiplizierten Beurteilung des entsprechenden Kriteriums zusammen. Je größer das Resultat, umso positiver ist das Ergebnis einzuschätzen.

	DRBD/Pacemaker	Xen-Hypervisor/ Netapp Storage	Icinga 2-Cluster	Wichtung
Failover-Zeit	+	-	+	3
Stabilität	-	+	+	3
Konfiguration	-	+	+	2
Datenverlust	+	-	-	2
Kosten	+	-	+	1
Ressourcenauslastung	-	+	+	1
Gesamtergebnis	0	0	8	

Tabelle 9: Vergleich Hochverfügbarkeit

Die Failover-Zeit ist sowohl bei DRBD mit Pacemaker als auch beim Icinga 2-Cluster als sehr gut zu beurteilen. Da auf dem sekundären Server alle Dienste installiert sind, muss Pacemaker lediglich die Prozesse dort starten. Der Failover passiert dabei im Sekundenbereich. Bei Icinga 2 laufen alle benötigten Prozesse bereits auf der anderen Instanz, weshalb nach dem Feststellen des Verbindungsausfalls keinerlei Zeit benötigt wird, um Prozesse zu starten. Es geschieht lediglich eine Festlegung, dass die noch aktiven Server im Cluster die Aufgaben des ausgefallenen Servers übernehmen. Beim Xen-Hypervisor dauert der Failover-Prozess etwas länger. Die längere Ausfallzeit ist durch zwei Ursachen zu begründen. Der Verbindungsverlust wird durch zwei verschiedene Mechanismen nacheinander geprüft: Erst wird die direkte Verbindung der Xen-Server untereinander getestet und danach die Abfrage über eine neutrale Instanz (Netapp Storage) durch die in Dateien geschriebenen Heartbeats. Der andere Faktor für den langsameren Schwenk ist die Tatsache, dass auf dem noch aktiven Server nicht wie bei Pacemaker nur Prozesse gestartet werden müssen, sondern komplette Virtuelle Maschinen. Dadurch entsteht eine Ausfallzeit von mehreren Minuten.
Die Konfiguration von DRBD mit Pacemaker ist im Vergleich zum Xen Hypervisor und dem Icinga 2-Cluster deutlich komplexer. Hier müssen sowohl bei Corosync als auch bei Pacemaker Interfaces konfiguriert und Schlüssel für die Authentifizierung erzeugt werden. Zudem müssen Konfigurationsdateien für DRBD, Corosync und Pacemaker angepasst werden. Eine zusätzliche Definition der Ressourcen über die CRM[39]-Konsole des Pacemakers ist ebenfalls notwendig. Einfacher gestaltet sich dies beim Xen-Hypervisor, der eine Konfiguration über grafisches Interface durch wenige Schritte ermöglicht (Vgl. [13]). Die Nutzung eines Icinga 2-Clusters ist ebenso durch wenige Konsolenbefehle möglich. Lediglich die Aktivierung der gewünschten Features sowie die Anpassung der Zonen ist hier notwendig.
Bezüglich des Datenverlustes erwies sich DRDB mit Pacemaker als optimale Lösung. Da der Schwenk innerhalb von Sekunden abläuft, kommt es praktisch zu kaum Datenverlust. Beim Xen-Hypervisor gehen die Daten der letzten Minuten verloren, begründet durch die Failover-Zeit. In Bezug auf den Datenverlust ist bei Icinga 2 eine Unterscheidung vorzunehmen. Es gibt letztendlich einen Konfigurations-Master, der die Konfigurationen an die anderen Instanzen verteilt oder sich diese per Kommando von den Instanzen holt. Fällt eine der anderen Instanzen aus, ist kein Datenverlust feststellbar. Fällt der Konfigurations-Master aus, ergibt sich für den aktuellen Betrieb auch keine Änderung; alles funktioniert weiter wie bisher. Allerdings ist es nötig für weitere Konfigurationen, im Sinne der Erstellung diverser Objekte, dass ein Administrator Verzeichnisse verschiebt, so dass ein noch aktiver Server für die Verteilung der Konfigurationen zuständig sein kann. Problematisch ist in diesem Falle die fehlende Synchronisation für den Fall, dass der eigentliche Konfigurations-Master wieder vorhanden sein sollte. Geschehene Änderungen im Zeitraum des

[39]CRM – Cluster Ressource Manager

Ausfalls werden diesem nicht übertragen. So wird ein weiterer administrativer Eingriff benötigt, um die Daten wieder konsistent zu halten.
Beim Kosten-Kriterium ergeben sich für DRBD mit Pacemaker und Icinga 2 keine Unterschiede. Bei beiden Varianten ist lediglich mindestens ein weiterer Server von Nöten um eine Hochverfügbarkeit zu gewährleisten. Die Möglichkeit zur Hochverfügbarkeit mittels Xen-Hypervisor hingegen benötigt ein externes Speichermedium, in diesem Fall ein Netapp Storage. Durch diesen zusätzlichen Kostenfaktor sind DRBD mit Pacemaker und das Icinga 2-Cluster in dieser Hinsicht als besser zu betrachten. Da alle Produkte frei verfügbar als Open-Source vorliegen, ist eine Betrachtung der Softwarekosten hinfällig.
Bezüglich der Ressourcenauslastung ist die Hochverfügbarkeitslösung mit DRBD und Pacemaker als unvorteilhaft einzuschätzen. Hier verharrt ein Server ständig im Hot Standby und übernimmt keinerlei Aufgaben. Dadurch ist nur eine 50%-ige Auslastung der Ressourcen gegeben. Demgegenüber sind das Icinga 2-Cluster und der Xen-Hypervisor für Aktiv-/Aktiv-Cluster vorgesehen. Das bedeutet, dass alle im Cluster vorhandenen Instanzen Aufgaben übernehmen, wodurch eine Lastverteilung und somit eine bessere Ressourcenauslastung erreicht wird.

3.6.5 Fazit

Zusammenfassend bietet sich in Bezug auf die Hochverfügbarkeit ein sehr eindeutiges Ergebnis, welches auch Tabelle 9: „Vergleich Hochverfügbarkeit“ darstellt. Viele der bewerteten Kriterien können durch das Icinga 2-Cluster gut realisiert werden, was bei den beiden Varianten Xen Hypervisor mit Netapp Storage und DRDB mit Pacemaker zumindest in einigen Punkten nicht so optimal funktioniert. Eine Nutzung des Icinga 2-Clusters ist deshalb sehr zu empfehlen. Allerdings ist die Hochverfügbarkeit lediglich auf den Icinga 2-Prozess bezogen. Für das Monitoring ist es zudem notwendig auch weitere Komponenten wie beispielsweise den Datenbank-Server oder den Web-Server hochverfügbar zu halten. Dies kann nur durch eine der beiden anderen Varianten geschehen. Die optimale Lösung würde eine Kombination vom Xen-Hypervisor mit Netapp Storage und Pacemaker darstellen. So könnte auf der einen Seite der Nachteil der 50%-igen Ressourcenauslastung beim DRBD-/Pacemaker-Setup ausgeglichen werden. Auf der anderen Seite wäre nicht nur die reine Überwachung der jeweiligen VM möglich, sondern durch Pacemaker auch die Überwachung der laufenden Prozesse, um diese gegebenenfalls wieder starten zu können. Es ist somit zu empfehlen ein gemischtes Setup zu nutzen, um eine angemessene Ausfallsicherheit für alle benötigten Komponenten gewährleisten zu können.

3.7 Aufwandsanalyse

Die Migration eines Monitoring-Systems nimmt je nach Größe des Systems unterschiedlich viel Zeit in Anspruch. Das momentan im Einsatz befindliche Icinga 1-System umfasst mittlerweile 19.577 Hosts und 158.751 Services (187 unterschiedliche Services). Bei solch einem großen System ist mit dementsprechend viel Zeit und demnach auch mit einer großen Arbeitsleistung zu rechnen, was natürlich mit Kosten für das Unternehmen verbunden ist.Deshalb ist die Abwägung der Kosten und des durch die Migration gewonnenen Nutzens von großer Bedeutung. Im Folgenden wird eine Abschätzung der benötigten Zeit für die Installation der Instanzen sowie für die Konfiguration der Objekte erfolgen. Als Installationszeit wird hierbei die Zeit definiert, die für die reine Installation der Pakete sowie der Herstellung der Verbindung zwischen den Instanzen inklusive Erstellung der Zertifikate, Anpassung der NodeNames und Zonenkonfigurationen benötigt wird. Die Konfigurationszeit wird als die Zeit definiert, die für die Erstellung aller im aktuellen System vorhandenen Objekte auf dem neuen System aufgewendet werden muss. Die Aufnahme der Zeiten erfolgte anhand der Installationen und Konfigurationen, die durch Mitarbeiter des Teams FNO durchgeführt wurden, da diese später auch für die Betreuung des Systems zuständig sein werden.

3.7.1 Installationszeit

Icinga 2-Satellit:

Inst.-Nr.	1	2	3	4	5	6	7	8	9	10
Zeit t_{Sat} in min	8min 56s	8min 53s	8min 33s	8min 15s	8min 24s	8min 07s	8min 29s	8min 25s	8min 18s	8min 22s

Tabelle 10: Installationszeit Icinga 2-Satellit

Durchschnittliche Zeit $\overline{t_{Sat}}$: 8min 28s
Anzahl Satelliten n_{Sat} im aktuellen System: 7

Installationsdauer Satelliten: $t_{Satelliten} = \overline{t_{Sat}} * n_{Sat}$
$t_{Satelliten} = 59\text{min}16\text{s}$

Icinga 2-Client:

Inst.-Nr.	1	2	3	4	5	6	7	8	9	10
Zeit t_C in min	12min 05s	11min 38s	11min 42s	10min 58s	11min 40s	11min 13s	11min 55s	11min 28s	11min 41s	11min 36s

Tabelle 11: Installationszeit Icinga 2-Client

Durchschnittliche Installationsdauer $\overline{t_C}$: 11min 36s
Anzahl der Clients n_C im aktuellen System: 19.577

Gesamtinstallationsdauer Clients: $t_{Clients} = \overline{t_C} * n_C$
$t_{Clients} = 3784\text{h}53\text{min}12\text{s}$

Die Installationsdauer eines einzelnen Satelliten und Clients unterscheidet sich aufgrund der manuellen Erzeugung der für die Verbindung benötigten Zertifikate und der nötigen Anpassung der Zonenkonfigurationen auf mehreren Instanzen bei der Installation der Clients.
Insgesamt ergibt sich mit knapp 3786h ein enormer Arbeitsaufwand. Rechnet man diesen Wert

bezogen auf eine bei der Telekom üblichen 38h-Woche, so müsste eine Arbeitskraft mehr als 99 Wochen sich jeden Arbeitstag durchgehend nur mit der Migration beschäftigen. Diese Werte sind vor allem bezüglich der Produktivität aus betrieblicher Hinsicht nicht vertretbar, weshalb eine manuelle Installation der Instanzen nicht durchgeführt werden kann. Alternativ können verschiedene Arbeitsschritte durch spezielle Programme automatisiert werden. Im OSS-Bereich der Telekom wird dafür unter anderem „Puppet“ verwendet. Dies könnte beispielsweise genutzt werden, um die Icinga 2-Installationen automatisch durchzuführen und die benötigten Zertifikate an die Clients und Satelliten zu verteilen. Nachfolgend werden die Zeitmessungen für die Einrichtung der Clients erneut durchgeführt. Die Einrichtung beschränkt sich in diesem Fall auf die Zonenkonfiguration und weitere für jeden Server individuelle Konfigurationen. Dafür wurden Server mit bereits vorhandenen Icinga 2-Installationen und Zertifikaten genutzt.

Inst.-Nr.	1	2	3	4	5	6	7	8	9	10
Zeit t_C in min	1min 45s	1min 35s	1min 34s	1min 26s	1min 30s	1min 33s	1min 30s	1min 19s	1min 29s	1min 32s

Tabelle 12: Installationszeit Icinga 2-Client mit Puppet

Durchschnittliche Installationsdauer $\bar{t_C}$: 1min 31s
Anzahl der Clients n_C im aktuellen System: 19.577

Gesamtinstallationsdauer Clients: $t_{Clients} = \bar{t_C} * n_C$
$t_{Clients} = 494\text{h}51\text{min}47\text{s}$

Durch Nutzung von Puppet oder ähnlichen Automatisierungs-Tools könnte die Installationsdauer der Clients drastisch gesenkt werden auf knapp 495h. Bezogen auf eine 38h-Woche würde eine Arbeitskraft mehr als 13 Wochen lang an der Migration arbeiten.

3.7.2 Konfigurationszeit

Da Hosts und Services eine annähernd gleiche Anzahl an Code-Zeilen aufweisen, abhängig von jedem speziellen Host und Service, wird im weiteren Verlauf von einer gleichen Konfigurationszeit ausgegangen und auf eine differenzierte Betrachtung verzichtet.

Host-/ Service-Nr.	1	2	3	4	5	6	7	8	9	10
Zeit t_{conf} in min	53s	48s	45s	48s	41s	51s	47s	46s	42s	45s

Tabelle 13: Konfigurationszeit

Durchschnittliche Konfigurationsdauer $\overline{t_{conf}}$: 47s
Anzahl der Host-Objekte n_H im aktuellen System: 19.577
Anzahl der unterschiedlichen Service-Objekte n_S im aktuellen System: 187
Anzahl der Host-und Service-Objekte n_O im aktuellen System: 19.577 + 187 = 19.764

Gesamtinstallationsdauer Clients: $t_{Konfiguration} = \overline{t_{conf}} * n_O$

$t_{Konfiguration} = 258\text{h}1\text{min}52\text{s}$

Allein für die Konfiguration der Host- und Service-Objekte wären bei einer manuellen Migration mehr als 258h Stunden notwendig. Bezogen auf eine 38h-Woche würde eine Arbeitskraft fast 7 Wochen dafür benötigen. Auf die Betrachtung weiterer spezifischer Konfigurationen, die für den Betrieb notwendig sind, wurde verzichtet, da diese im Vergleich zur Konfiguration der Host- und Service-Objekte einen minimalen Anteil einnehmen und deshalb bei der Aufwandsbetrachtung vernachlässigt werden können.
Wie im Abschnitt 3.2 „Migrationsarten“ bereits erwähnt, könnte für die Konfigurationsmigration ein neues Tool entwickelt werden. Die genaue Entwicklungszeit für ein derartiges Programm ist nur schwer einzuschätzen, sollte jedoch deutlich unter 258h liegen, weshalb diese Migrationsart zu bevorzugen ist.

3.7.3 Fazit

Alles in allem ergibt sich für die gesamte Migration des Systems ein sehr großer Zeitbedarf. Bei einer kompletten Umstellung aller Clients ergäbe sich eine mindestens benötigte Zeit von ca. 754h. Weitere Zeiteinsparungen sind wie bereits erwähnt durch die Erzeugung eines neuen Automatisierungs-Tools möglich. Dennoch bleibt mit fast 500h für die reine Umstellung der Clients noch ohne benötigte Konfigurationsmigration ein enormer Aufwand. Da auf allen Servern bereits Programme für Remote-Abfragen installiert und konfiguriert sind, wäre die Umstellung aufgrund der Funktionalitäten des Icinga 2-Clients zwar ein Gewinn für das Unternehmen; Jedoch steht der Arbeitsaufwand in keinerlei Relation zum Mehrgewinn, weshalb davon abgeraten werden muss, den Icinga 2-Client auf allen Remote-Instanzen zu installieren. Vielmehr sollten bereits vorhandene Strukturen genutzt werden. Die Umstellung der Worker auf die Icinga 2-Satelliten hingegen nimmt aufgrund der geringen Anzahl nur wenig Zeit in Anspruch. Für die reine Installation und die Herstellung der Verbindungen wird lediglich eine Stunde benötigt. Natürlich ist trotzdem noch die Konfigurationsmigration nötig. Da sich in der Performance-Analyse eine erhebliche Steigerung der Leistung gezeigt hat, ist hier trotz der 258h Konfigurationsaufwand, welche durch die Nutzung eines Automatisierungs-Tools noch deutlich gesenkt werden könnten, eine Migration als sinnvoll und vor allem auch wirtschaftlich zu erachten.

4 Auswertung

Im OSS-Bereich der Deutschen Telekom AG werden aktuell bereits über 158.000 Services und über 19.000 Hosts überwacht. Die Migration eines Monitoring-Systems von derart großem Ausmaß ist mit einem nicht unerheblichen Aufwand verbunden, weshalb vorher analysiert werden sollte, ob das neue System den Anforderungen auch entspricht. Mit Icinga 2 fiel die Wahl auf den direkten Nachfolger des aktuellen Monitoring-Systems. Bei der Entwicklung flossen viele Erfahrungen aus dem Umgang mit Icinga 1 und Nagios ein, was sich letztendlich auch für den Nutzer bemerkbar macht. Icinga 2 ist vom Syntax noch stark an Icinga 1 angelehnt. So haben sich zwar einige Schlüsselwörter für die Definition der Objekte geändert, der generelle Aufbau kann jedoch wie bei Icinga 1 vollzogen werden. Damit ist keine große Umgewöhnung für den Nutzer notwendig. Zusätzlich wurden weitere Optionen für die Konfiguration der Objekte hinzugefügt. Es kann vor allem durch die Nutzung von Apply- und Assign-Regeln, aber auch anderer Änderungen wie den neuen Notification-Objekten oder die Möglichkeit zur Nutzung von benutzerdefinierten Variablen sowie Bedingungen bei der Definition der Objekt-Attribute, eine sehr dynamische Konfiguration geschaffen werden. Dass auch die starre Zuordnung von Objekten noch immer möglich ist, begünstigt eine mögliche automatische Migration der Konfiguration. Das Testen des von den Entwicklern bereitgestellten Migrations-Tools war dennoch erfolglos. Es wurden verschiedene Fehlermeldungen erzeugt, die nicht alle beseitigt werden konnten, weshalb die Migration mit Hilfe des bereitgestellten Programms nicht vollzogen werden konnte. Bei der Analyse des Aufwands für die Migration der Konfiguration zeigte sich, dass die Entwicklung eines entsprechenden Tools für die Migration der Host- und Service-Objekte, welche den Großteil der Konfiguration ausmachen, eine große Zeitersparnis darstellen kann. Unter Umständen müsste dafür allerdings auf einige Vorteile der neuen Konfiguration verzichtet werden. Wäre der Zeitaufwand vernachlässigbar, würde die manuelle Migration die bessere Wahl darstellen.
Ein Punkt, der auch große Bedeutung für die Migration hatte, war die Untersuchung des Icinga 2-Clusters, um eine Einschätzung der verschiedenen Hochverfügbarkeitsvarianten im Telekom-Netz durchführen zu können. Hier ergab sich bezüglich des Icinga 2-Prozesses ein sehr eindeutiges Ergebnis zugunsten des Icinga 2-Clusters. Bei den wichtigsten Kriterien zeigten sowohl die Hochverfügbarkeitsvariante per DRBD mit Pacemaker als auch die Variante des Xen-Hypervisors mit Netapp Storage kleine Nachteile. Allerdings wird durch das Icinga 2-Cluster lediglich der Icinga 2-Prozess überwacht. Die optimale Lösung zur Sicherung der Hochverfügbarkeit wäre deshalb ein Icinga 2-Cluster und zusätzlich ein kombinierter Einsatz vom Xen-Hypervisor mit Netapp Storage und Pacemaker, um andere für das Monitoring benötigte Komponenten wie Datenbank und Webserver mit angemessener Ausfallsicherheit zu realisieren. Die Kombination der Varianten sichert eine optimale Ressourcenauslastung mit Überwachung der laufenden Prozesse.
Als Hauptgrund für das in Auftrag geben dieser Arbeit war die mangelnde Leistung des aktuellen Icinga-Systems. Eine Performance-Analyse sollte hier Aufschluss geben, ob durch eine Migration ein ausreichender Performance-Gewinn erzielt werden könne. Zu diesem Zweck wurde ein Open Source Tool genutzt, um künstlich Last-Situationen für Icinga 1- und Icinga 2-Instanzen zu generieren und anhand dessen Messwerte zu erzeugen, die eine Beurteilung der Leistung zulassen. Dafür wurden Testdurchläufe mit einem 8 Core-System sowie einem System mit 32 Cores vorgenommen. Es zeigte sich, dass Icinga 1 trotz Gearman-Implementation bei den meisten Tests nur ein Zehntel der Leistung von Icinga 2 erbringen konnte. Lediglich bei großen Perl-Plugins, die in realen Umgebungen selten eingesetzt und auch im Netz der Telekom nicht verwendet werden, zeigten sich Vorteile für die Icinga 1-Instanzen. Bei allen Plugins, die vergleichbar zu den für die Abfrage von Ressourcen im Telekom-Netz genutzten Plugins sind, zeigte sich Icinga 2 als sehr viel performanter. Zudem konnte bei Icinga 2 aufgrund des Multithreadings eine Abhängigkeit von Leistung und Core-Anzahl nachgewiesen werden. Das bedeutet, dass bei einer angemessenen Dimensionierung des zukünftigen Systems auch in naher Zukunft keine Performance-Probleme zu erwarten sind, weshalb aus dieser Hinsicht eine Migration sehr sinnvoll erscheint.
Als Bedingung dafür, dass eine Migration von Seiten der Telekom in Betracht gezogen wird, galt auch das verteilte Monitoring. Es zeigte sich, dass mit Icinga 2 keine Gearman-Kompatibilität mehr

gegeben war. Stattdessen lieferten die Entwickler die Möglichkeit Icinga 2-Instanzen selbst als eine Art Worker zu implementieren. Die sogenannten Icinga 2-Satelliten sollten den Gearman-Worker ersetzen können. Ein Vergleich der beiden Programme ergab, alle von Gearman gebotenen wichtigen Funktionen auch mittels Icinga 2-Satelliten umgesetzt werden konnten. Zudem ergaben sich weitere Vorteile wie der lokale Scheduler der Satelliten, durch den Abfragen auch trotz Verbindungsverlust zum Core weiterhin durchgeführt werden können. Mittels Replay-Log werden die Abfrageergebnisse bei Wiederherstellung der Verbindung an den Core übertragen. Allerdings zeigten sich auch kleine Probleme bei der Verwendung von Load-Balancing und der Anzeige von Ergebnissen auf dem Core bei einem Verbindungsverlust zum Satelliten. Diese konnten irreführende Check-Ergebnisse erzeugen. Da Maßnahmen zum Umgehen dieser Problematiken bekannt sind, können diese Schwierigkeiten als unkritisch angesehen werden.
Bei einem Setup mit Icinga 2-Satelliten muss eine Konfiguration auf jeder Icinga 2-Instanz vorhanden sein. Dafür bestehen zwei Möglichkeiten: Das Verteilen der Konfiguration vom Core oder das Senden der lokalen Konfiguration der Satelliten an den Core. Die erste Variante ist bei einem so großen System wie im Netz der Telekom aufgrund des deutlich geringeren Konfigurationsaufwandes und der automatischen Verteilung derselbigen die beste Lösung.
Eine weitere Implementationsmöglichkeit für eine Icinga 2-Instanz ist der Icinga 2-Client als Command Execution Bridge. Hiermit soll die Ausführung von Remote-Plugins möglich sein. Ein Vergleich der aktuell im Telekom-Netz eingesetzten Remote-Abfragen und dem Icinga 2-Client fiel zu Gunsten des Icinga 2-Clients aus. Dieser zeigte sich hinsichtlich Sicherheit und vorhandener Funktionalitäten als beste Möglichkeit für Remote-Abfragen. Dennoch wäre eine Kombination aus SNMP-Abfragen und Icinga 2-Client nötig, da hardwarenahe Geräte wie Router nicht durch den Icinga 2-Client abgefragt werden können. Gegenüber dieser funktionalen Betrachtungsweise musste allerdings in Bezug auf die Wirtschaftlichkeit festgestellt werden, dass eine Installation des Icinga 2-Clients auf allen abzufragenden Hosts als nicht sinnvoll zu erachten ist. Auch die Verwendung von Programmen zur automatisierten Installation bedeutet zwar eine große Zeitersparnis, ist aufgrund der dennoch nötigen individuellen Anpassungen aber nicht ausreichend, um den Zeitaufwand gegenüber dem Nutzen einer solchen Umstellung zu rechtfertigen.

Insgesamt erweist sich Icinga 2 als sehr umfassendes Monitoring-Tool mit vielen hilfreichen Funktionen, die die Einsparung von diversen externen Programmen ermöglichen. Nichtsdestotrotz zeigen sich auch hier noch gewisse Probleme, die ohne entsprechende Vorsichtsmaßnahmen zu unerwünschten Beeinträchtigungen des Monitorings führen können. Zusammenfassend ist eine Migration jedoch sehr zu empfehlen. Der deutliche Performance-Gewinn sowie neue Funktionalitäten, die eine homogenere Monitoring-Umgebung ermöglichen, sind dabei sehr wichtige Faktoren. Bezogen auf die Telekom ist dabei folgende Struktur ratsam: Mindestens zwei Icinga 2-Core Instanzen, um das integrierte Cluster nutzen zu können und mindestens zwei Satelliten für jedes Netzsegment, um Load-Balancing und eine gewisse Ausfallsicherheit gewährleisten zu können. Bezüglich der Hochverfügbarkeit sollte zusätzlich ein Xen-Hypervisor mit Netapp-Storage und Pacemaker zur Prozessüberwachung eingesetzt werden. Auf die Nutzung der Icinga 2-Clients sollte aufgrund des enormen Aufwands zur Umstellung und den damit verbundenen Kosten verzichtet werden. Die Migration der Konfiguration sollte aus selbigem Grund mittels eines neu entwickelten Tools erfolgen. Die Konfiguration sollte dann von den Core-Instanzen automatisch an die Satelliten verteilt werden.
Abschließend bleibt zu erwähnen, dass in Bezug auf die aufzuwendende Zeit und dem daraus resultierenden Ergebnis dieses Setup als angemessene Lösung für das Monitoring im OSS-Bereich der Deutschen Telekom AG genutzt werden kann.

Quellenverzeichnis

Bücher:

[1] Timo Kucza, Ralf Staudemeyer:
Das Nagios-/ Icinga-Kochbuch (August 2013)

[2] Wolfgang Barth:
Nagios System- und Netzwerk-Monitoring (Oktober 2005)

[3] Helmut Herold:
Linux/Unix Systemprogrammierung (2004)

[4] Oliver Liebel:
Linux Hochverfügbarkeit (2011)

[5] Klaus M. Rodewig:
Netzwerke mit Linux – Hochverfügbarkeit, Sicherheit und Performance (2006)

[6] Michael Schwarkopff:
Clusterbau: Hochverfügbarkeit mit Linux (Juni 2012)

[7] Douglas R. Mauro, Kevin J. Schmidt:
Essential SNMP (September 2005)

Zeitschriftenaufsätze und wissenschaftliche Arbeiten:

[8] Admin-Magazin:
Einführung in DRBD (Ausgabe 06/2010)
http://www.admin-magazin.de/Das-Heft/2010/06/Einfuehrung-in-DRBD

[9] Admin-Magazin:
HA-Serie, Teil 1: Grundlagen von Pacemaker und Co. (Ausgabe 04/2011)
http://www.admin-magazin.de/Das-Heft/2011/04/HA-Serie-Teil-1-Grundlagen-von-Pacemaker-und-Co

[10] Eric Day, Brian Aker:
Gearman (Juli 2009)
http://cdn.oreillystatic.com/en/assets/1/event/27/Gearman_%20Build%20Your%20Own%20Distributed%20Platform%20in%203%20Hours%20Presentation.pdf

[11] Heise Zeitschriften Verlag:
iX Kompakt Administration Open-Source 2014 (Ausgabe 03/2014)

[12] Heise online – c't-Magazin
Stille Helfer unter Beschuss (Mai 2002)

Internetquellen:

[13] Citrtix Support Knowledge Center:
http://support.citrix.com/article/CTX121708 (aufgerufen am 11.08.2015)

[14] Consol Labs – Gearman Dokumentation:

https://labs.consol.de/de/nagios/mod-gearman/index.html (aufgerufen am 30.07.2015)

[15] Gearman-Webseite:

http://gearman.org/ (aufgerufen am 30.07.2015)

[16] Icinga-Dokumentation:

http://docs.icinga.org/latest/de/ (aufgerufen am 04.06.2015)

[17] Icinga 2-Dokumentation:

http://docs.icinga.org/icinga2/latest/doc/module/icinga2/toc (aufgerufen am 04.06.2015)

[18] Icinga-Wiki:

https://wiki.icinga.org/dashboard.action

https://wiki.icinga.org/display/howtos/Monitoring+the+Icinga+Host (aufgerufen am 02.06.2015)

https://wiki.icinga.org/display/testing/Icinga+Plugin+Testing (aufgerufen am 02.06.2015)

https://wiki.icinga.org/display/testing/Icinga+Web+Testing (aufgerufen am 02.06.2015)

https://wiki.icinga.org/display/howtos/Performance+Tuning (aufgerufen am 02.06.2015)

[19] Nagios Bug Tracker

http://tracker.nagios.org/view.php?id=339 (aufgerufen am 27.08.2015)

[20] Netapp-Dokumentation RAID DP:

http://www.netapp.com/de/communities/tech-ontap/tot-back-to-basics-RAID-DP-1110 (aufgerufen am 11.08.2015)

[21] Netways Webinare:

https://www.netways.de/webinare/archiv/icinga_webinare/

Icinga 2: Neuheiten in 2.3 (aufgerufen am 09.06.2015)

Open Source Monitoring mit Icinga 2 (aufgerufen am 09.06.2015)

Warum Monitoring und warum Icinga 2 (aufgerufen am 09.06.2015)

Icinga Web 2: Icinga Web in neuem Design (aufgerufen am 09.06.2015)

Icinga 2: Entwicklungsstand 2014 (aufgerufen am 09.06.2015)

Icinga 2: Enterprise Monitoring der nächsten Generation (aufgerufen am 09.06.2015)

Icinga 2: Integrierte Hochverfügbarkeit (aufgerufen am 09.06.2015)

Icinga Web 2: Modernes Monitoring Interface (aufgerufen am 09.06.2015)

[22] Netways-Webseite mit Informationen zu Icinga:

https://www.netways.de/produkte/icinga/ (aufgerufen am 09.06.2015)

https://www.netways.de/produkte/icinga/icinga_2/ (sowie Unterpunkte Features, Architektur und Cluster aufgerufen am 09.06.2015)

[23] Offizielle Icinga-Webseite:

https://www.icinga.org/

https://www.icinga.org/icinga/icinga-1/architecture/ (aufgerufen am 02.06.2015)

https://www.icinga.org/icinga/icinga-1/features/ (aufgerufen am 02.06.2015)

https://www.icinga.org/icinga/icinga-2/architecture/ (aufgerufen am 02.06.2015)

https://www.icinga.org/icinga/icinga-2/features/ (aufgerufen am 02.06.2015)

https://www.icinga.org/icinga/icinga-2/why-icinga-2/ (aufgerufen am 02.06.2015)

https://www.icinga.org/icinga/icinga-2/distributed-monitoring/ (aufgerufen am 02.06.2015)

[24] Root.in:

http://www.slashroot.in/secure-shell-how-does-ssh-work (aufgerufen am 08.07.2015)

[25] Thomas Krenn AG: RAID DP

https://www.thomas-krenn.com/de/wiki/RAID-DP (aufgerufen am 11.08.2015)

Selbstständigkeitserklärung

Hiermit erkläre ich, Benjamin Bloßfeld, dass ich die von mir an der Hochschule für Telekommunikation Leipzig (FH) eingereichte Abschlussarbeit zum Thema

Analyse der Migration auf das Monitoring System ICINGA 2 im OSS Bereich der Deutschen Telekom Technik

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Leipzig, den 27.11.2015

Benjamin Bloßfeld

Anhang

Anhang A: Dateistruktur Bottom-Up und Top-Down

Top-Down:

```
[root@icinga2-core zones.d]# tree
.
├── global-templates
│   ├── templates.conf
│   └── user-not.conf
├── master
│   ├── hosts.conf
│   └── services.conf
├── README
└── satneu
    ├── hostgroups.conf
    ├── hosts.conf
    └── services.conf

3 directories, 8 files
```

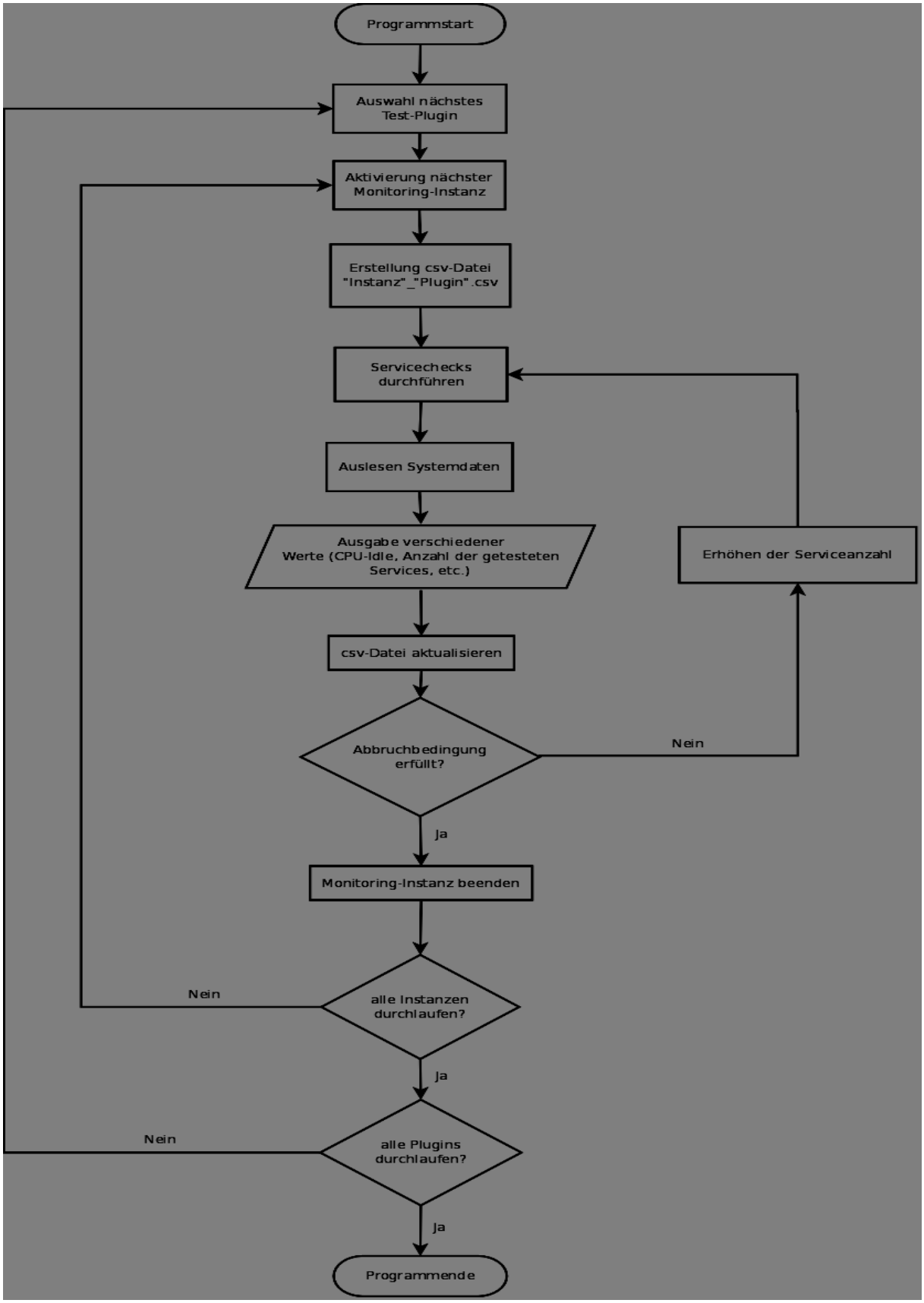
Den Abbildungen liegt das gleiche Monitoring-System zu Grunde:
- 2 Icinga 2-Core-Instanzen
- 2 Satelliten
- 2 Clients

Bottom-Up:

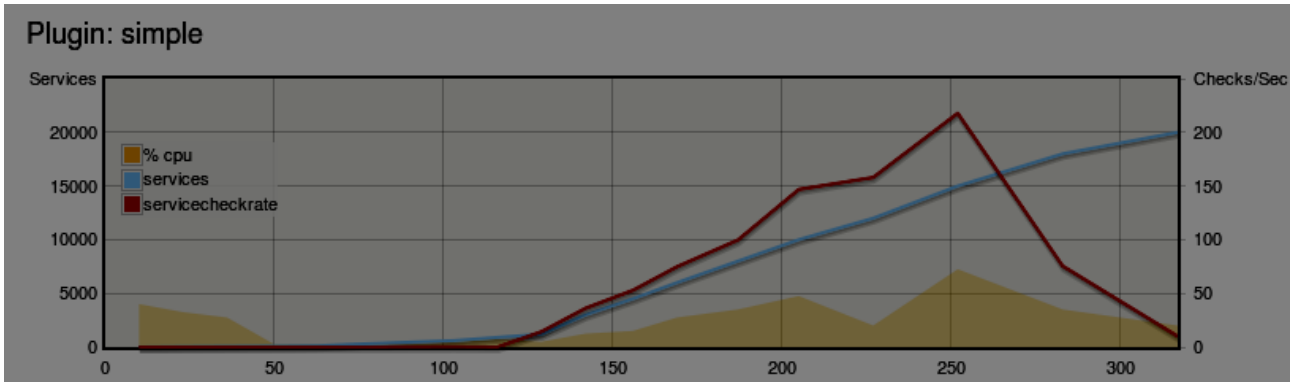
```
[root@icinga2-core repository.d]# tree
.
├── endpoints
│   ├── icinga2-core2.conf
│   ├── icinga-client.conf
│   ├── sat-2.conf
│   ├── satneu.conf
│   └── win-client.conf
├── hosts
│   ├── Client-172.16.0.3
│   │   ├── disk-remote.conf
│   │   ├── icinga.conf
│   │   ├── icinga-remote.conf
│   │   ├── load-remote.conf
│   │   ├── procs-remote.conf
│   │   ├── swap-remote.conf
│   │   ├── testping.conf
│   │   └── users-remote.conf
│   ├── Client-172.16.0.3.conf
│   ├── icinga2-core2
│   │   ├── cluster.conf
│   │   ├── disk-master.conf
│   │   ├── icinga.conf
│   │   ├── load-master.conf
│   │   ├── procs-master.conf
│   │   ├── swap-master.conf
│   │   └── users-master.conf
│   ├── icinga2-core2.conf
│   ├── icinga2-core.ospo
│   │   ├── cluster.conf
│   │   ├── disk-master.conf
│   │   ├── icinga.conf
│   │   ├── load-master.conf
│   │   ├── procs-master.conf
│   │   ├── swap-master.conf
│   │   └── users-master.conf
│   ├── icinga2-core.ospo.conf
│   ├── icinga-client.conf
│   ├── master.conf
│   ├── Satellit-10.0.0.2
│   │   ├── icinga.conf
│   │   └── testping.conf
│   ├── Satellit-10.0.0.2.conf
│   ├── Satellit-10.0.0.5
│   │   ├── icinga.conf
│   │   ├── if-0.conf
│   │   ├── if-1.conf
│   │   ├── if-2.conf
│   │   └── testping.conf
│   ├── Satellit-10.0.0.5.conf
│   ├── satneu.conf
│   ├── Win-Client-172.16.0.6
│   │   ├── icinga.conf
│   │   ├── load-remote.conf
│   │   ├── procs-remote.conf
│   │   ├── swap-remote.conf
│   │   ├── users-remote.conf
│   │   ├── win-disk-remote.conf
│   │   ├── win-load-remote.conf
│   │   ├── win-memory-remote.conf
│   │   ├── win-procs-remote.conf
│   │   ├── win-swap-remote.conf
│   │   ├── win-update-remote.conf
│   │   ├── win-uptime-remote.conf
│   │   └── win-users-remote.conf
│   └── Win-Client-172.16.0.6.conf
└── zones
    ├── icinga-client.conf
    ├── master.conf
    └── satneu.conf

9 directories, 59 files
```

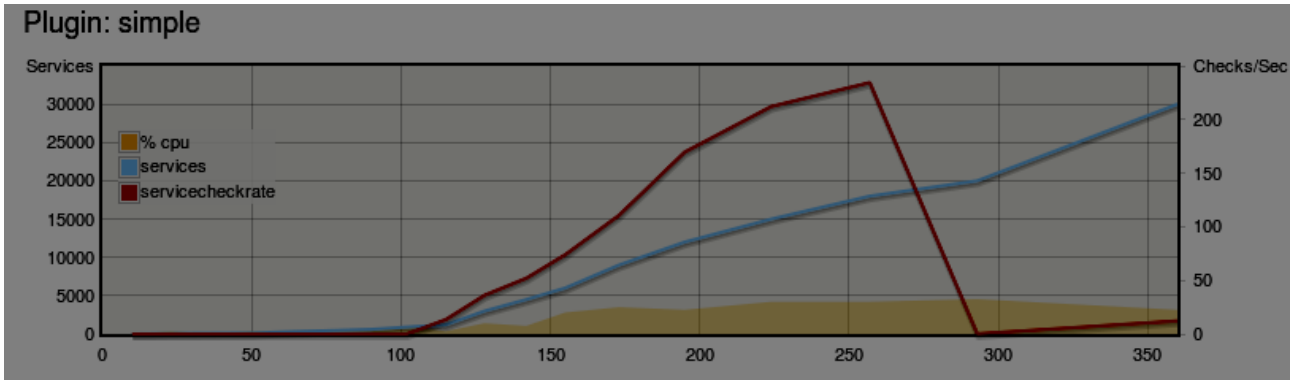
Anhang B: Programmablaufplan



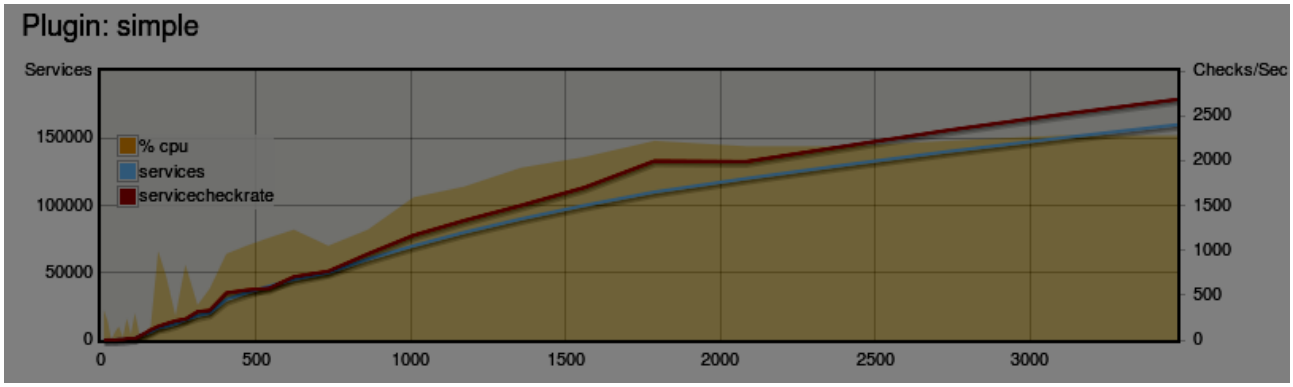
Anhang C: Performancetest – Diagramme



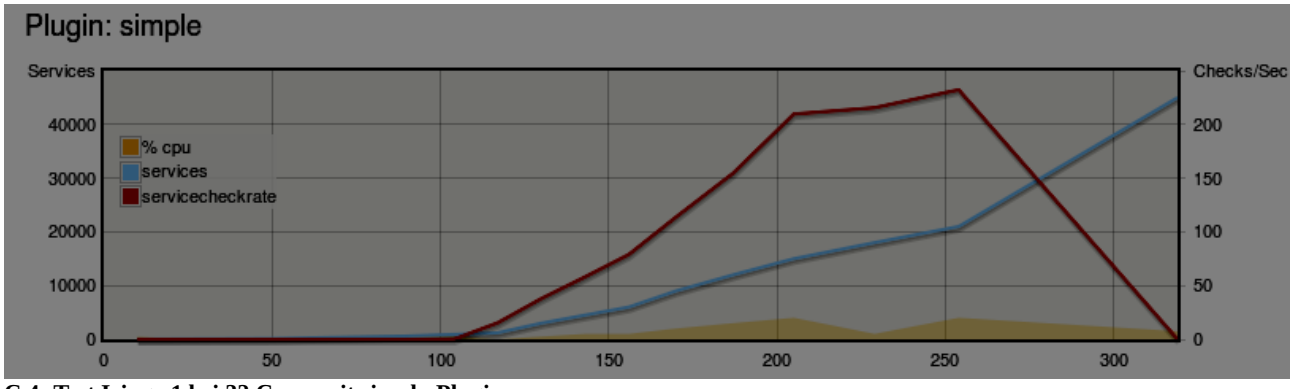
C.1: Test Icinga 1 bei 8 Cores mit simple-Plugin



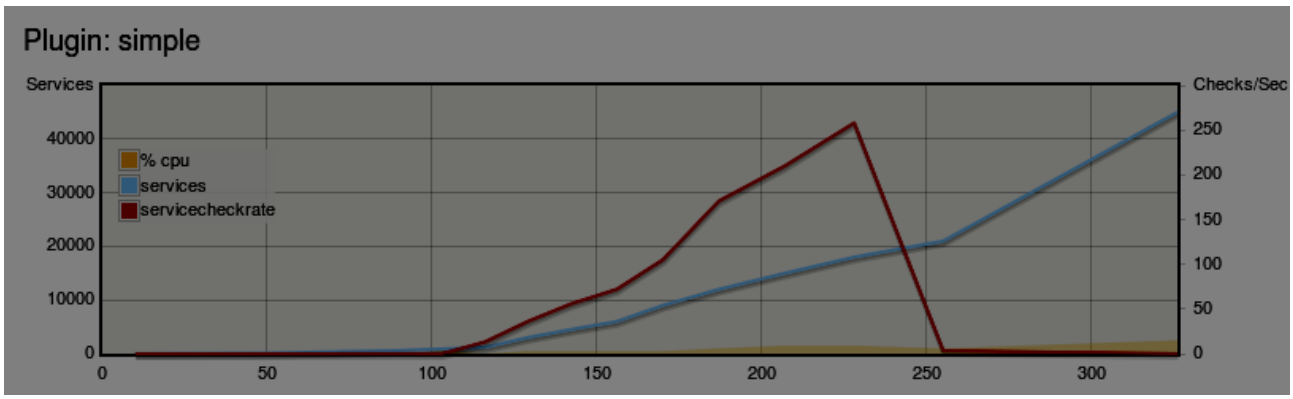
C.2: Test Icinga 1 mit Gearman bei 8 Cores mit simple-Plugin



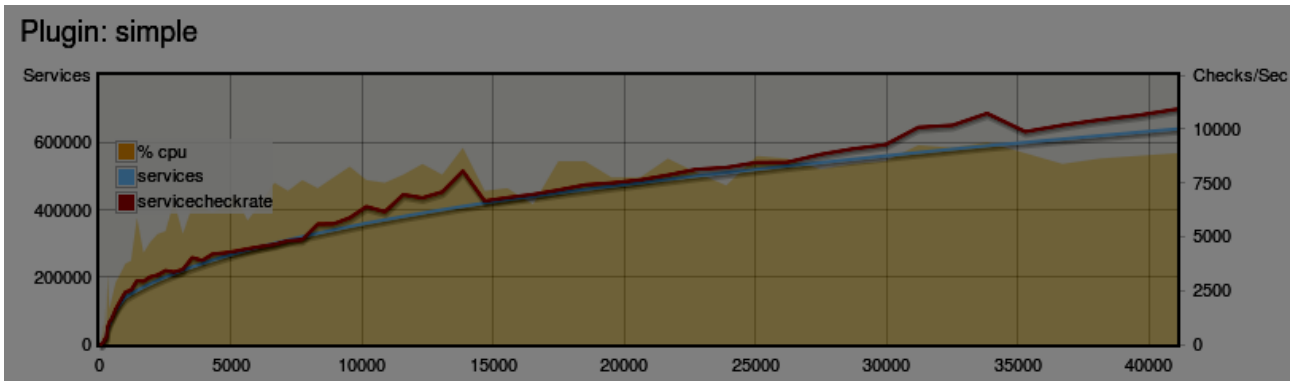
C.3: Test Icinga 2 bei 8 Cores mit simple-Plugin



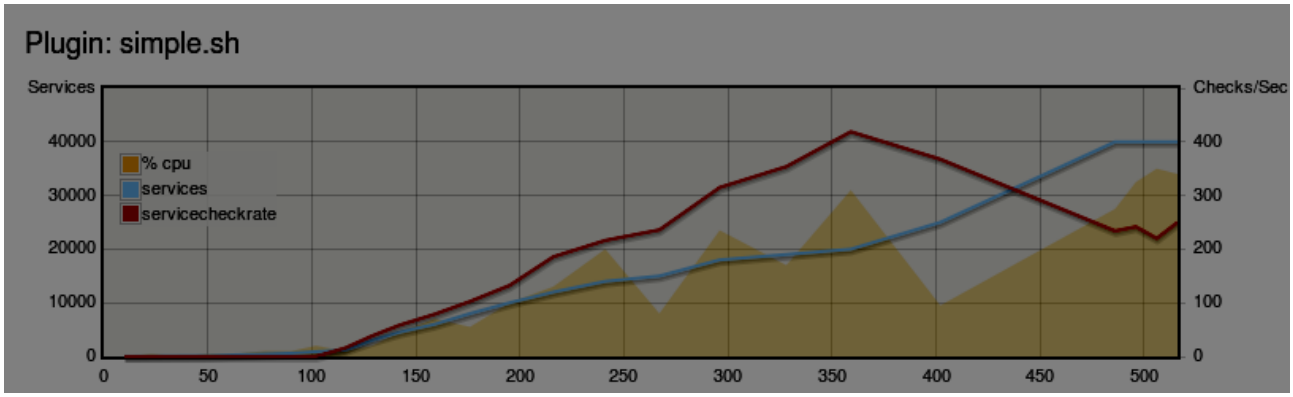
C.4: Test Icinga 1 bei 32 Cores mit simple-Plugin



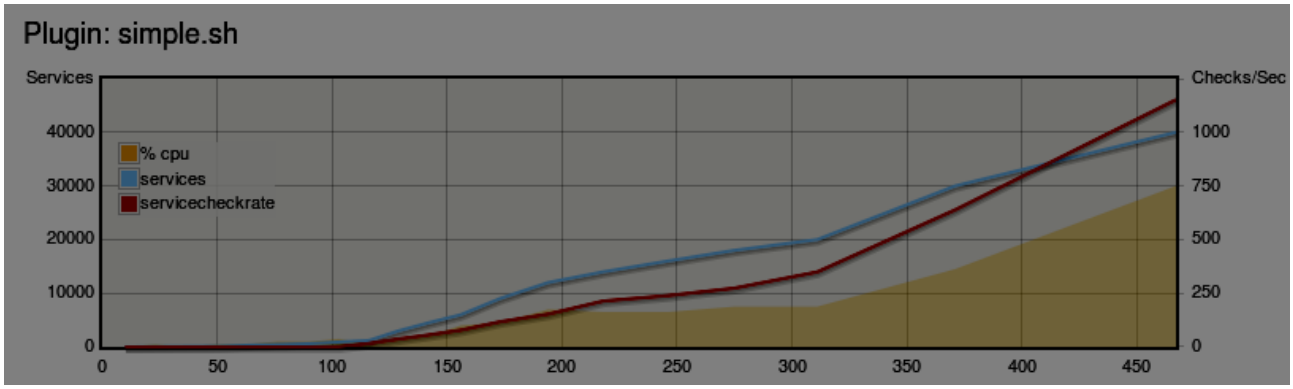
C.5: Test Icinga 1 mit Gearman bei 32 Cores mit simple-Plugin



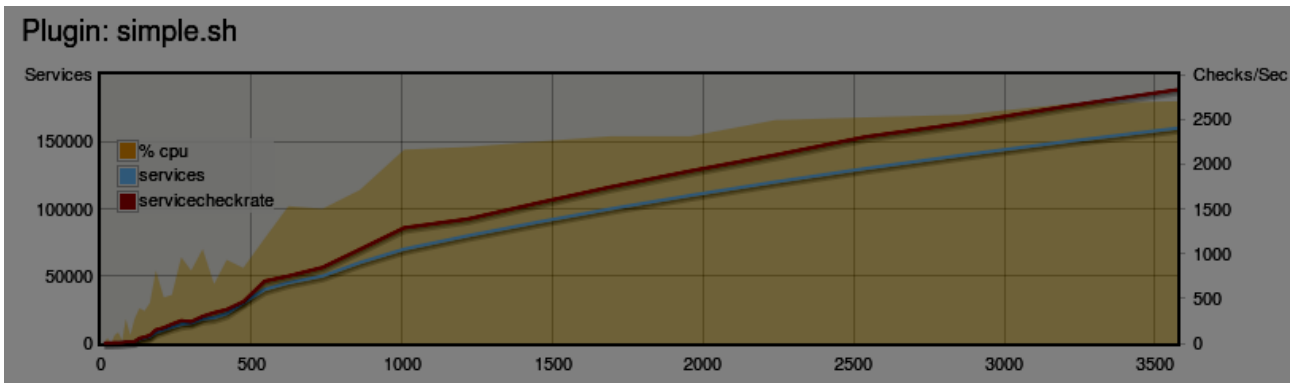
C.6: Test Icinga 2 bei 32 Cores mit simple-Plugin



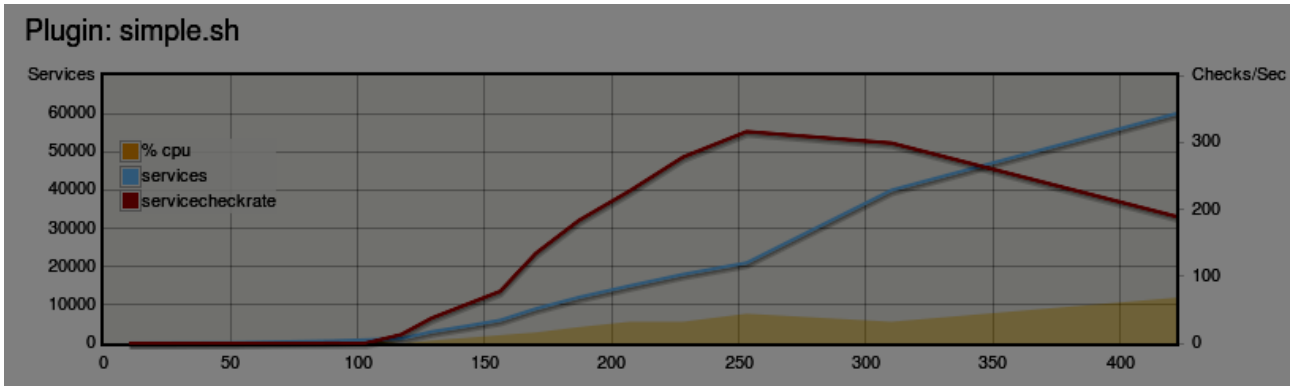
C.7: Test Icinga 1 bei 8 Cores mit simple.sh-Plugin



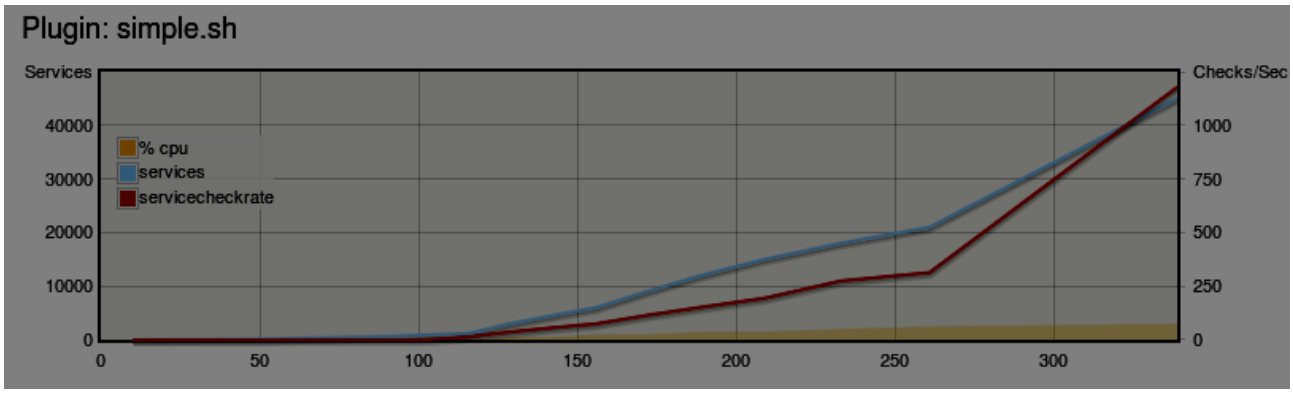
C.8: Test Icinga 1 mit Gearman bei 8 Cores mit simple.sh-Plugin



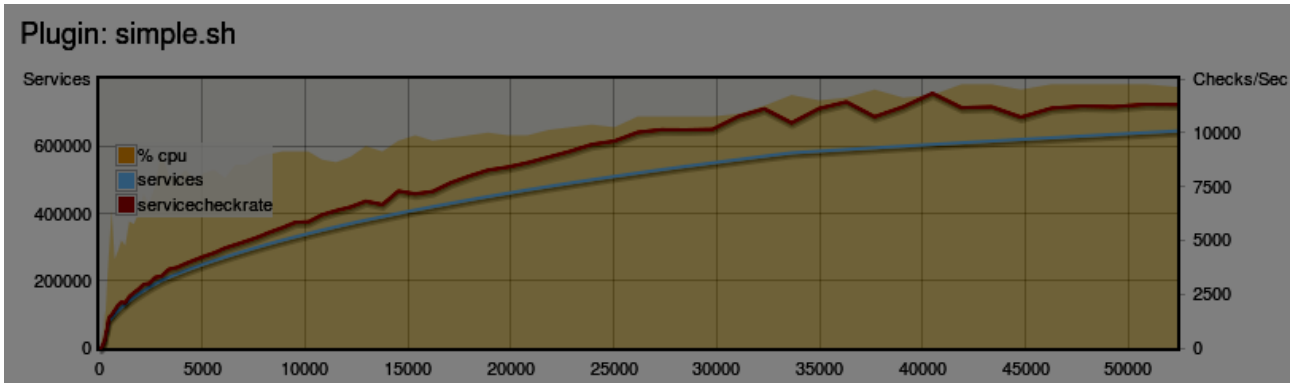
C.9: Test Icinga 2 bei 8 Cores mit simple.sh-Plugin



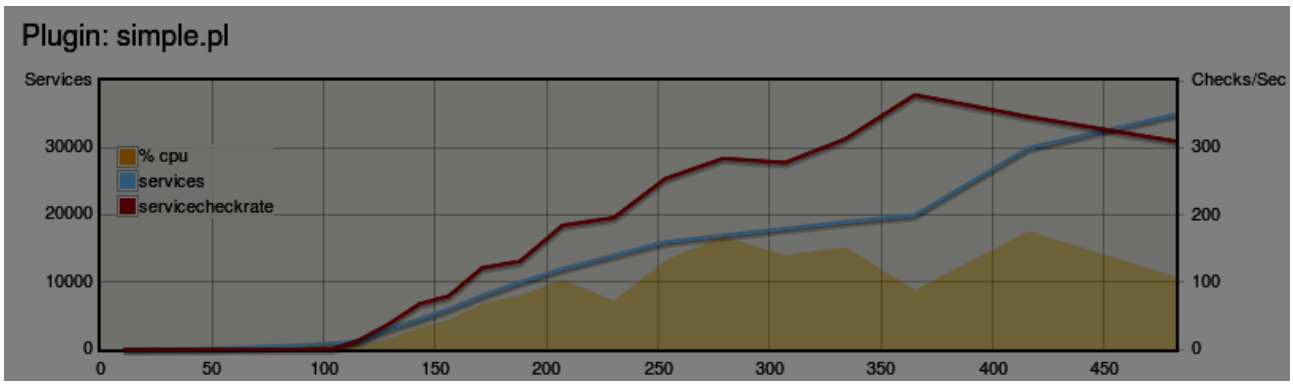
C.10: Test Icinga 1 bei 32 Cores mit simple.sh-Plugin



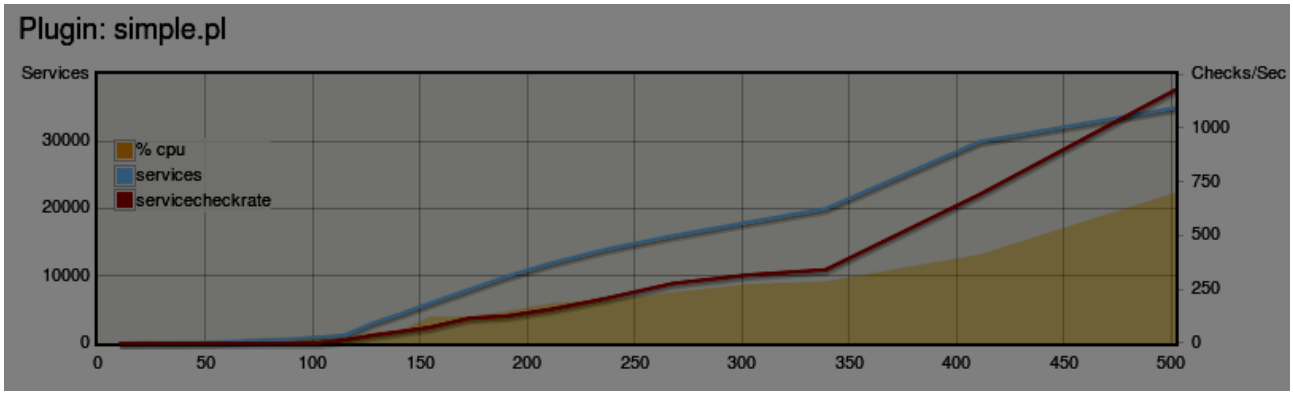
C.11: Test Icinga 1 mit Gearman bei 32 Cores mit simple.sh-Plugin



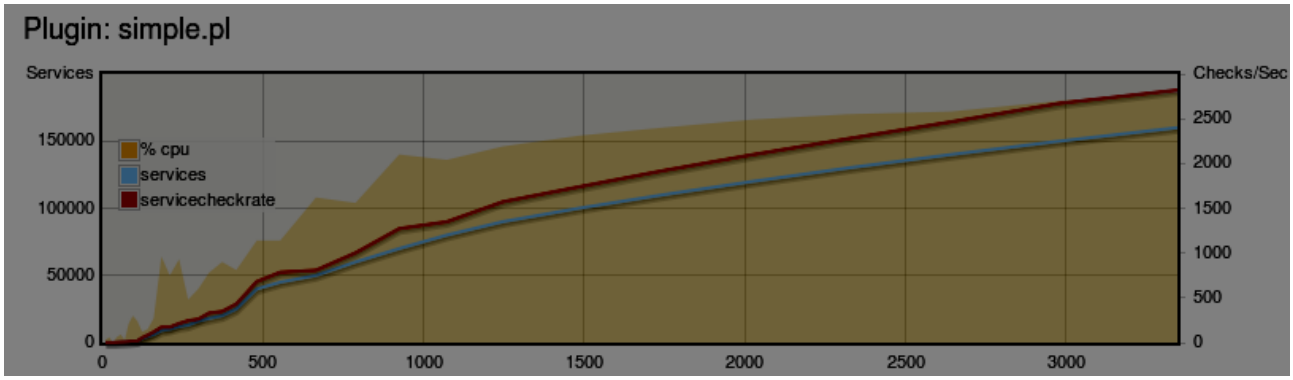
C.12: Test Icinga 2 bei 32 Cores mit simple.sh-Plugin



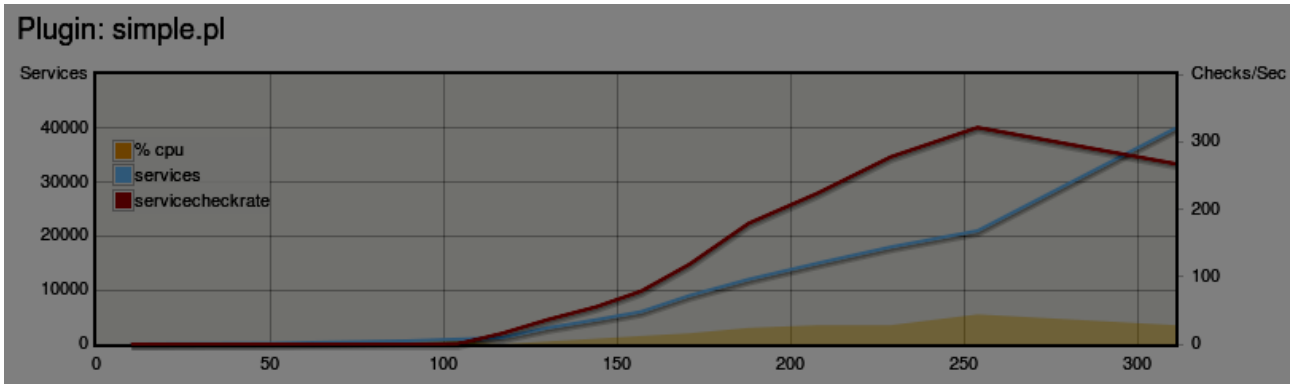
C.13: Test Icinga 1 bei 8 Cores mit simple.pl-Plugin



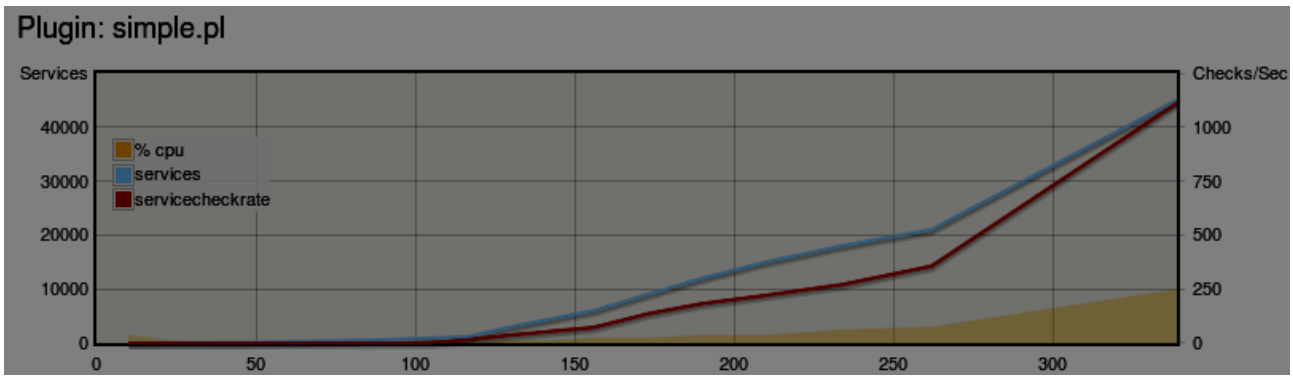
C.14: Test Icinga 1 mit Gearman bei 8 Cores mit simple.pl-Plugin



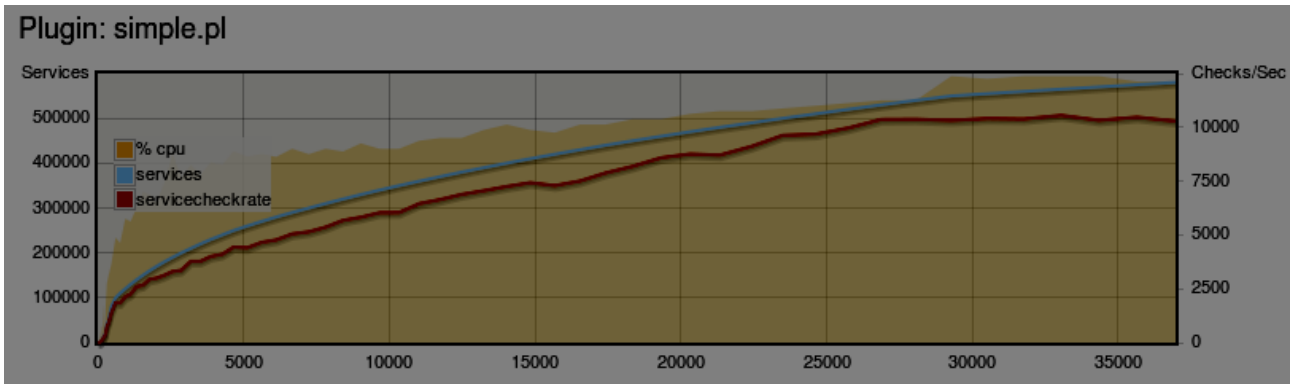
C.15: Test Icinga 2 bei 8 Cores mit simple.pl-Plugin



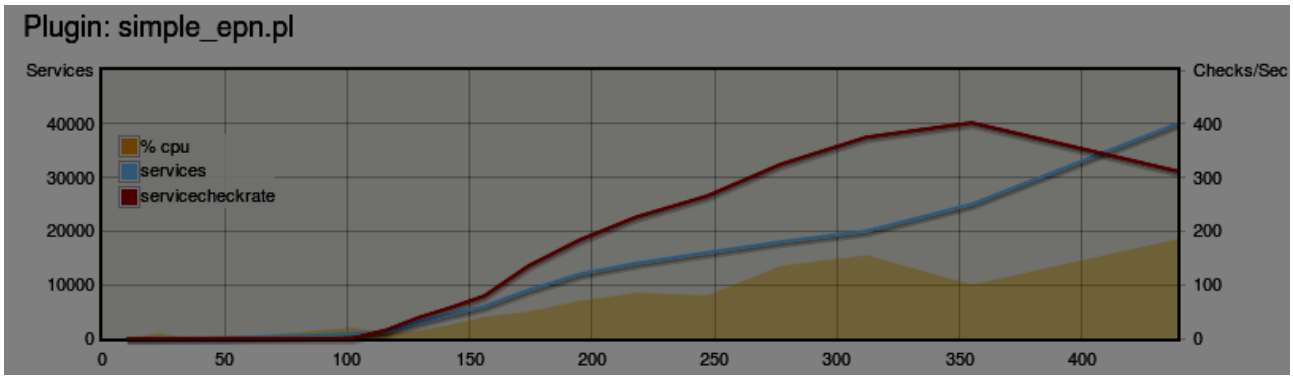
C.16: Test Icinga 1 bei 32 Cores mit simple.pl-Plugin



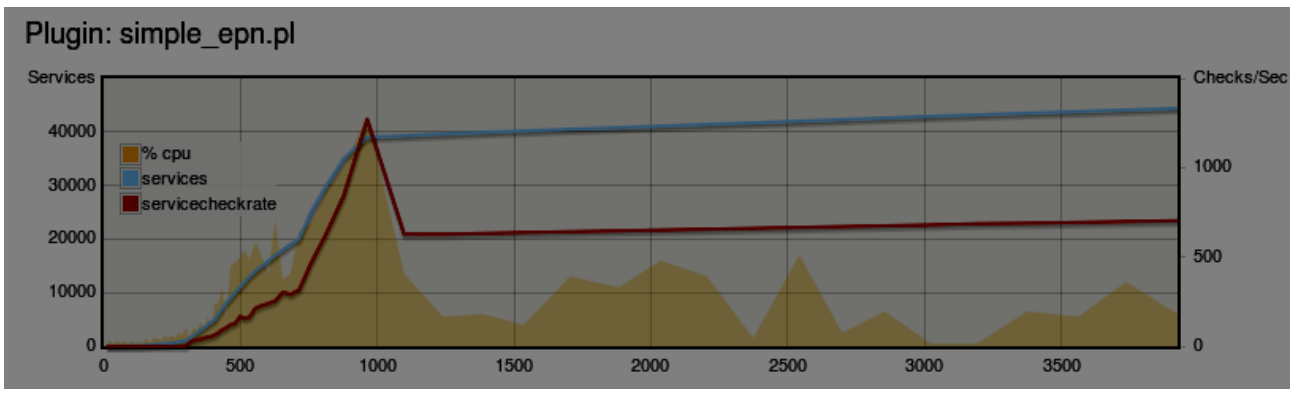
C.17: Test Icinga 1 mit Gearman bei 32 Cores mit simple.pl-Plugin



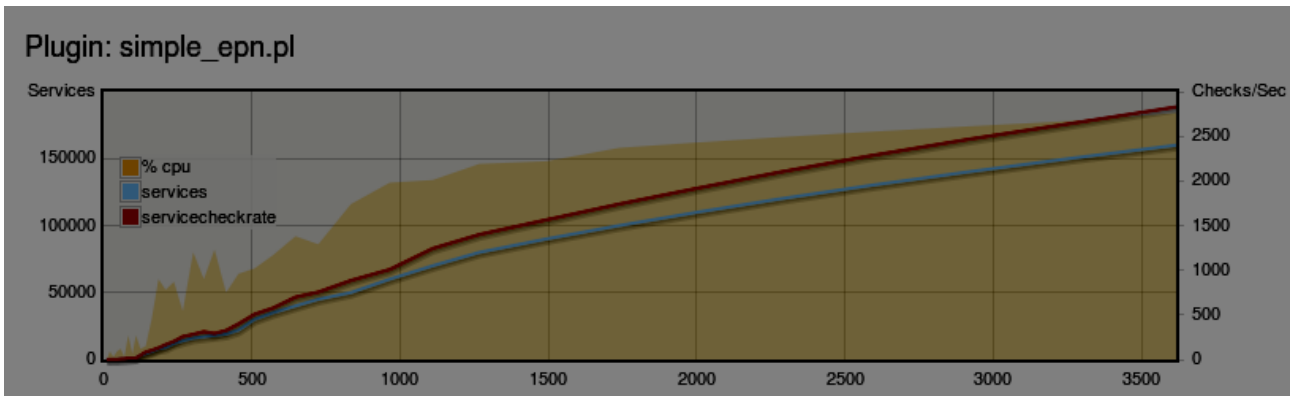
C.18: Test Icinga 2 bei 32 Cores mit simple.pl-Plugin



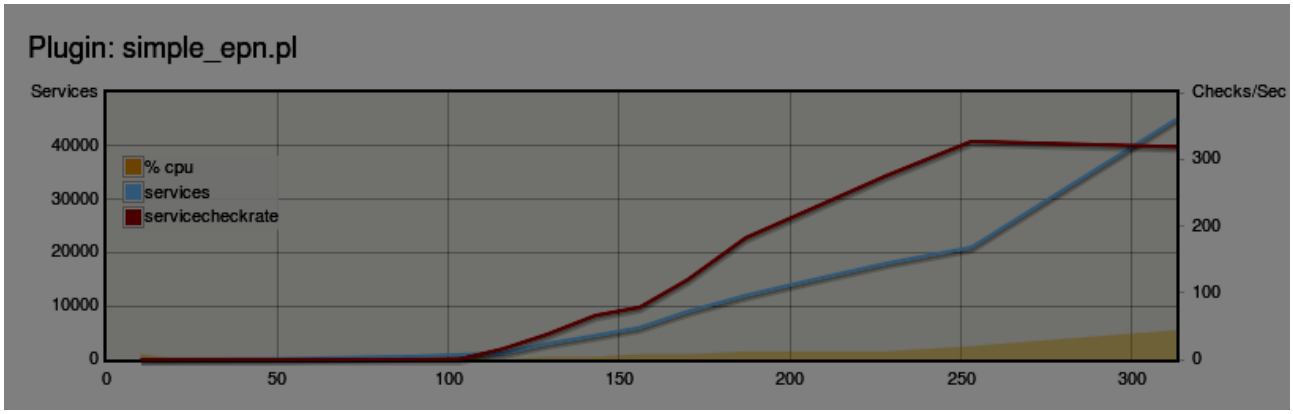
C.19: Test Icinga 1 bei 8 Cores mit simple_epn.pl-Plugin



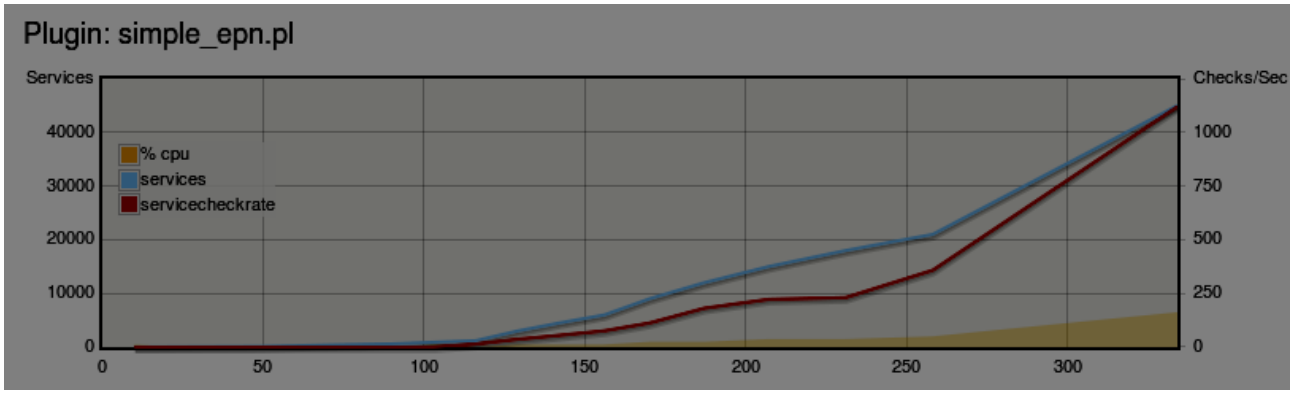
C.20: Test Icinga 1 mit Gearman bei 8 Cores mit simple_epn.pl-Plugin



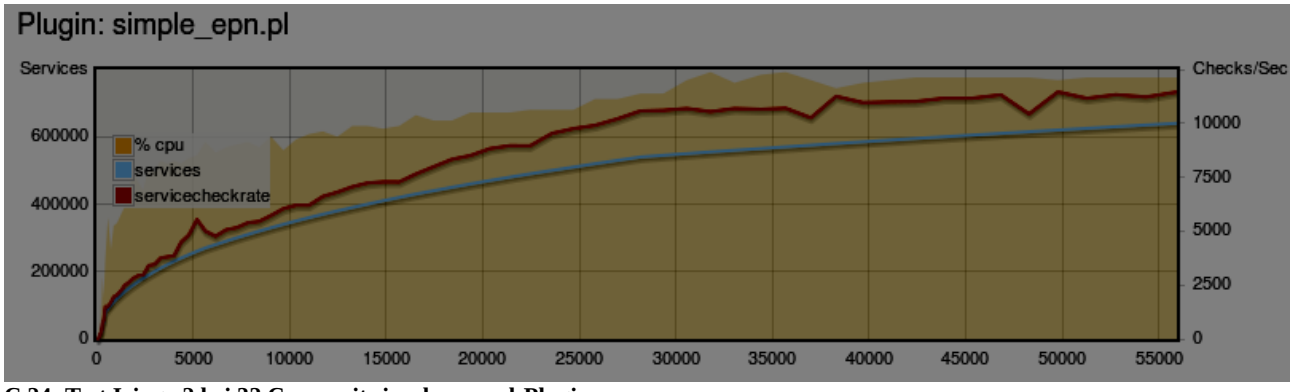
C.21: Test Icinga 2 bei 8 Cores mit simple_epn.pl-Plugin



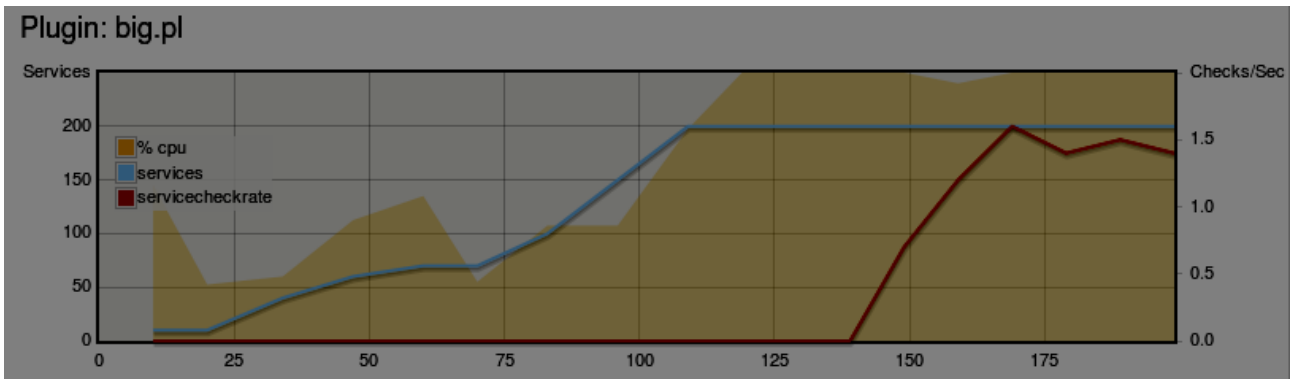
C.22: Test Icinga 1 bei 32 Cores mit simple_epn.pl-Plugin



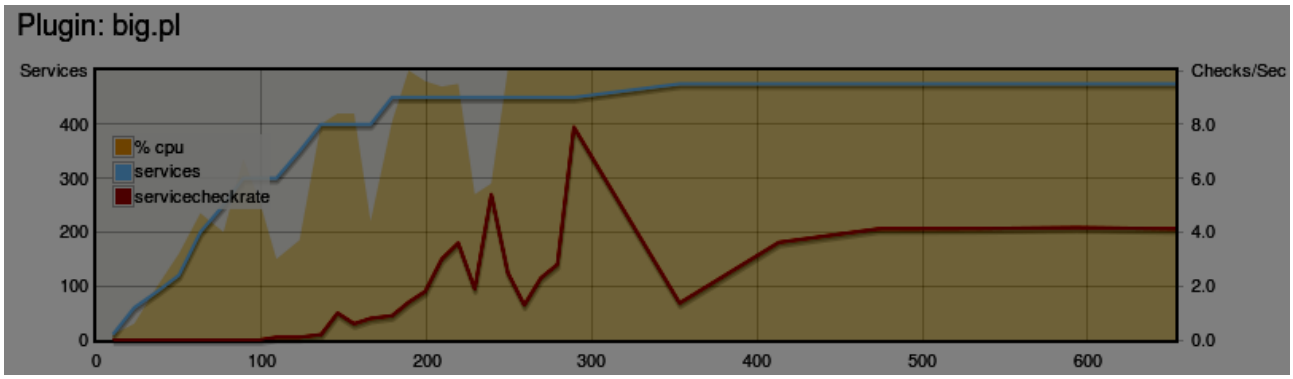
C.23: Test Icinga 1 mit Gearman bei 32 Cores mit simple_epn.pl-Plugin



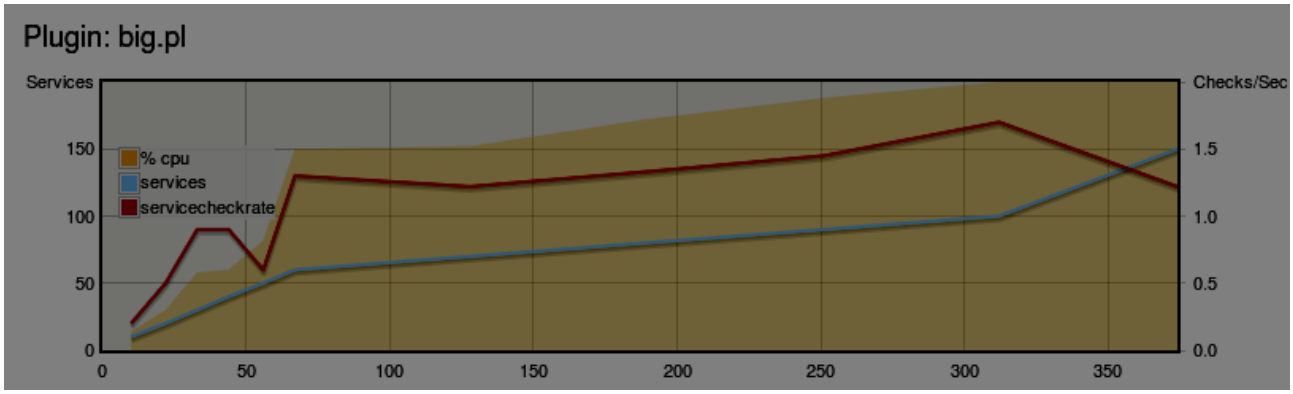
C.24: Test Icinga 2 bei 32 Cores mit simple_epn.pl-Plugin



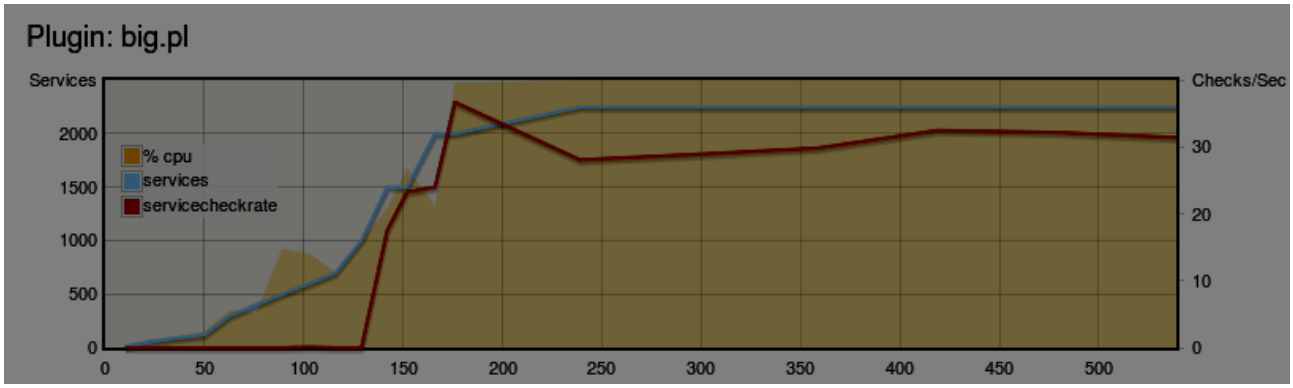
C.25: Test Icinga 1 bei 8 Cores mit big.pl-Plugin



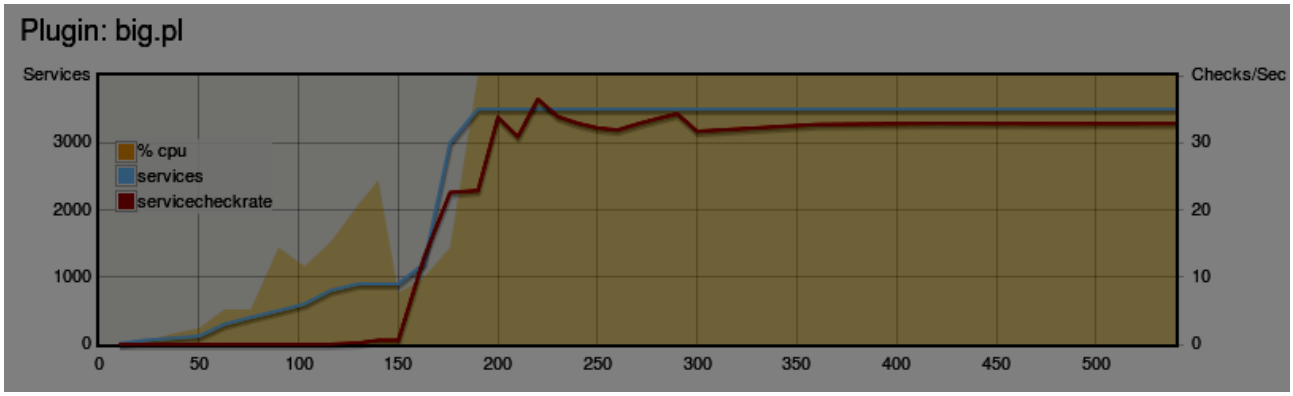
C.26: Test Icinga 1 mit Gearman bei 8 Cores mit big.pl-Plugin



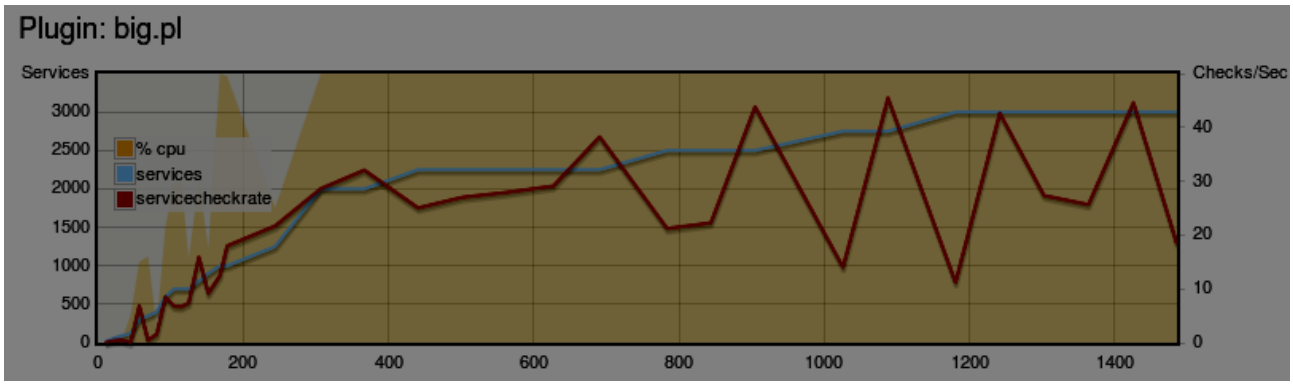
C.27: Test Icinga 2 bei 8 Cores mit big.pl-Plugin



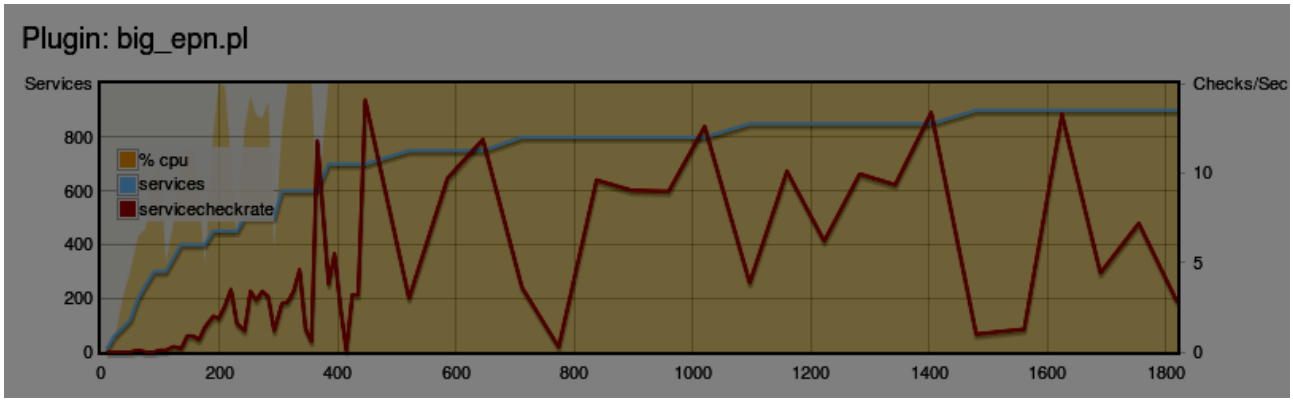
C.28: Test Icinga 1 bei 32 Cores mit big.pl-Plugin



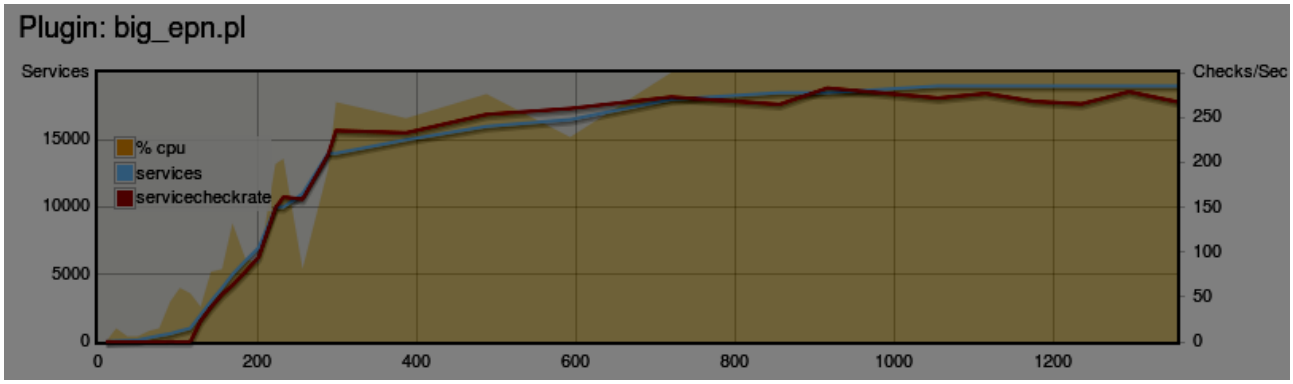
C.29: Test Icinga 1 mit Gearman bei 32 Cores mit big.pl-Plugin



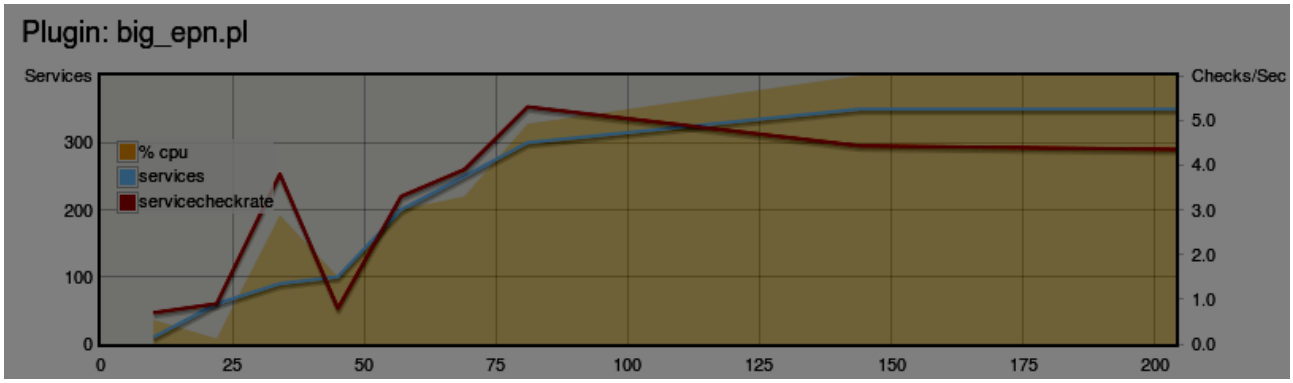
C.30: Test Icinga 2 bei 32 Cores mit big.pl-Plugin



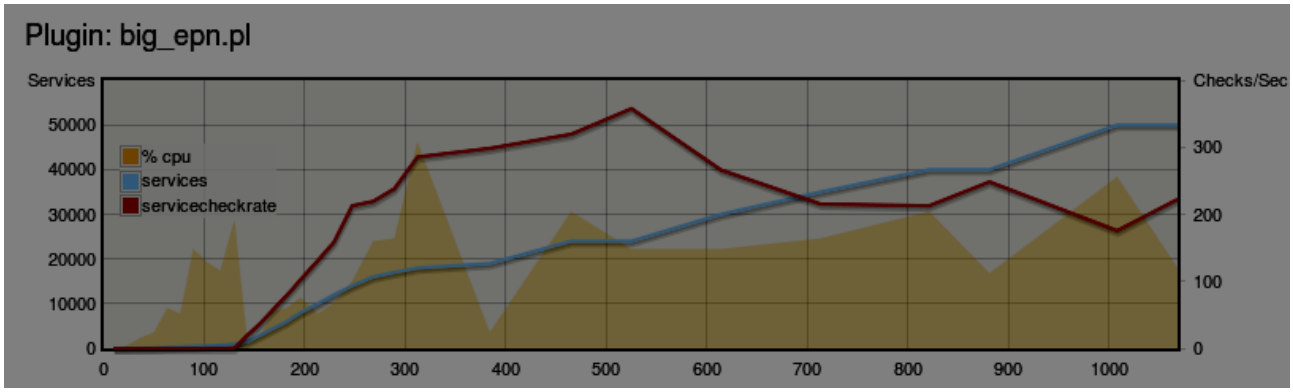
C.31: Test Icinga 1 bei 8 Cores mit big_epn.pl-Plugin



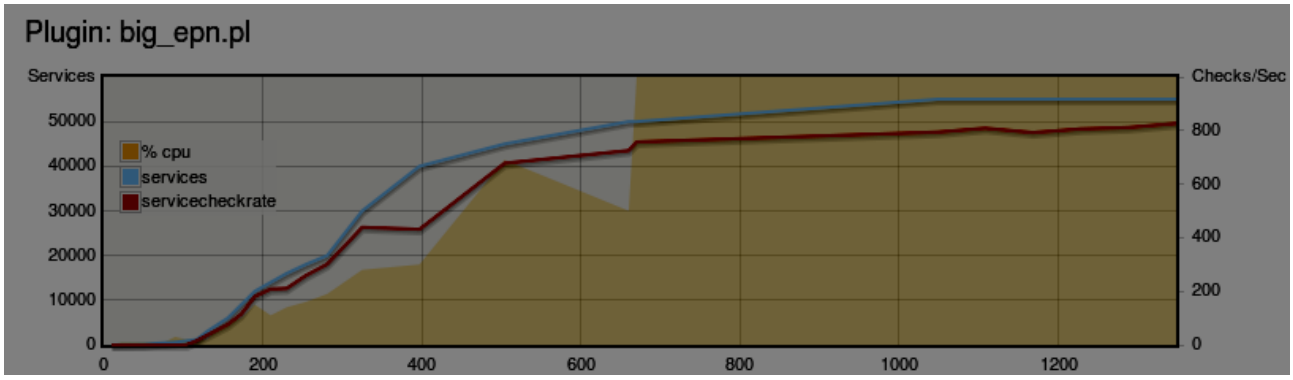
C.32: Test Icinga 1 mit Gearman bei 8 Cores mit big_epn.pl-Plugin



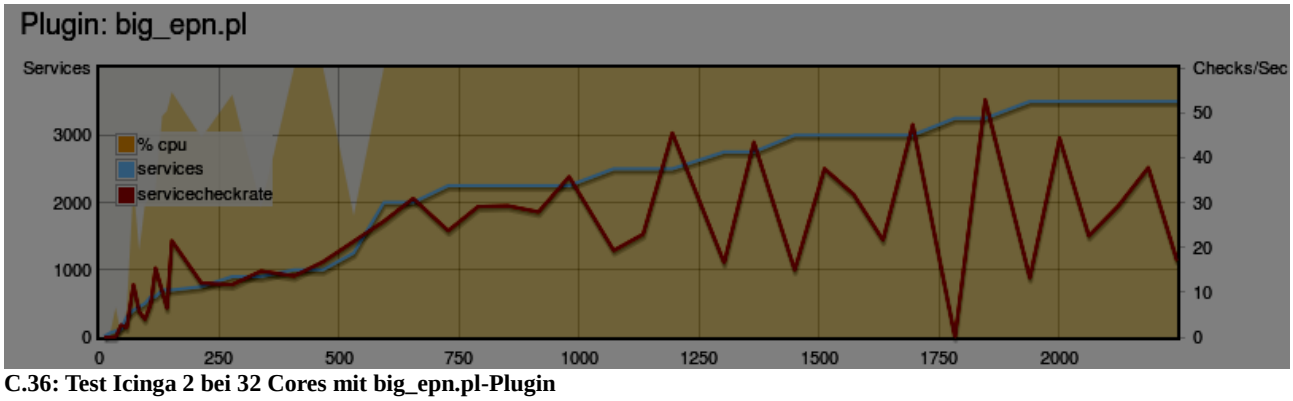
C.33: Test Icinga 2 bei 8 Cores mit big_epn.pl-Plugin



C.34: Test Icinga 1 bei 32 Cores mit big_epn.pl-Plugin



C.35: Test Icinga 1 mit Gearman bei 32 Cores mit big_epn.pl-Plugin



C.36: Test Icinga 2 bei 32 Cores mit big_epn.pl-Plugin

Anhang D: RAID DP

RAID DP (Redundant Array of Independent Disks Double Parity) ist ein Verfahren zur Erhöhung der Ausfallsicherheit von physikalischen Speichermedien. Physikalische Speicher werden zu einem logischen Laufwerk zusammengefasst. Bei RAID DP werden um die Sicherheit der Daten zu erhöhen zwei zusätzliche Platten für die Speicherung von Paritäten genutzt. Dabei wird die erste Parität horizontal über die Platten durchgeführt. Die zweite Platte verwendet eine diagonale Parität über die erste Paritätsfestplatte und alle weiteren Festplatten bis auf eine. Durch diesen Mechanismus können bis zu zwei Festplatten ausfallen ohne dass ein Datenverlust auftritt. Die Wiederherstellung erfolgt anhand der Paritätspartitionen. Im folgenden Bild wurde dies beispielhaft dargestellt: