4 Grundkonfiguration von Icinga 2

Die zentrale Konfigurationsdatei ist *icinga2.conf*. Alle anderen verwendeten Dateien werden direkt oder indirekt von dort im Stil von C/C++ aus eingebunden.

```
include "constants.conf"
```

In diesem Auszug wird *constants.conf* eingebunden. Benutzt man Double Quotes und keinen absoluten Pfad, wird im selben Verzeichnis als Ausgangspunkt gesucht, hier */etc/icinga2*.

```
include <itl>
include <plugins>

include "features-enabled/*.conf"
```

Hingegen wird bei der Verwendung von <...> im vorgegebenen Pfad */usr/share/icinga2/include* nach der entsprechenden Datei gesucht. Dieser Pfad kann sich je nach Installationspaket bzw. Plattform unterscheiden.


```
include_recursive "book.d"
```

Eine weitere Direktive ist *include_recursive*, bei der keine Datei, sondern ein Verzeichnis angegeben wird. Hierbei werden alle Dateien mit der Endung *.conf* in diesem Verzeichnis und rekursiv absteigend auch alle Dateien mit diesem Suffix in Unterverzeichnissen mit eingelesen.

4.1 Konstanten

Wie in *icinga2.conf* zu sehen, ist *constants.conf* die erste der Konfigurationsdateien, die gelesen wird. In ihr werden Konstanten gesetzt, die somit in den nachfolgenden Konfigurationsdateien zur weiteren Benutzung zur Verfügung stehen.

```
const PluginDir = "/usr/lib64/nagios/plugins"
//const NodeName = "localhost"
```

Codebeispiel 4.1: */etc/icinga2/constants.conf*

Standardmäßig ist *PluginDir* gesetzt und enthält als Wert den Pfad zu den Plugins. Diese Konstante wird dann z. B. in der Icinga Template Library (ITL) innerhalb der Definitionen von Check Commands verwendet. Wird *NodeName* nicht angegeben, ist automatisch der Fully Qualified Domain Name (FQDN) für diese Konstante gesetzt.

4.2 Icinga Template Library

Bei der ITL handelt es sich um eine Sammlung von Templates und Check Commands. Check Commands regeln in Icinga 2 den Aufruf der Plugins. Zur Erinnerung: Ein Plugin ist ein Programm, das durch eigene Logik den Status eines Service oder Hosts ermittelt. Das heißt, in einer Service-Definition wird das anzuwendende Check Command angegeben und damit wird bestimmt, welches Plugin für einen Check dieses Service ausgeführt wird.

Die ITL befindet sich in `/usr/share/icinga2/include` und besteht aus mehreren unterschiedlichen Dateien, die sich ggf. gegenseitig einbinden. So zieht *include <plugins>* unter anderem das Laden von `command-plugins.conf` nach sich. In dieser Datei befinden sich Check-Command-Definitionen zu den Plugins aus dem Monitoring-Plugins-Projekt[1].

```
include "command-plugins.conf"
```

Codebeispiel 4.2: `/usr/share/icinga2/include/plugins`

Die Datei `itl` selbst lädt Templates und Methoden nach, wie z. B. *plugin-check-command*, das den Aufruf von Plugins steuert.

Auf diese Weise werden Check-Command-Definitionen für häufig verwendete Plugins mitgeliefert. Im Gegensatz zu anderen Teilen der Konfiguration gibt es nicht viel Wahlmöglichkeit in diesen Definitionen, außerdem sind sie oft lang und eintönig. Durch die fertig ausformulierten Check-Command-Definitionen in der ITL nimmt das Icinga-Team den Anwendern viel Arbeit im Alltag ab. Es lohnt sich, immer wieder einen Blick in die ITL zu werfen, da sie mit neuen Icinga 2 Versionen regelmäßig erweitert wird. Eigene Check Commands sollte man übrigens keinesfalls in der ITL hinzufügen, da diese von den Paketen nicht als Konfigurationsdatei angesehen und bei Updates einfach überschrieben wird. Besser aufgehoben sind sie stattdessen in einer eigenen Datei im gleichen Verzeichnis wie die Service-Definitionen, egal ob sie in `/etc/icinga2/book.d` oder in einer globalen Zone, wie in Abschnitt 13.3 ab Seite 204 beschrieben, liegt.

[1] https://www.monitoring-plugins.org

4.3 Features

Viel der Funktionalität von Icinga 2 ist in sogenannte Features ausgelagert, die bei Bedarf hinzugeladen werden. Alle zur Auswahl stehenden Features können über einen CLI-Aufruf angezeigt werden. Die Ausgabe zeigt zusätzlich an, ob diese zurzeit aktiviert oder deaktiviert sind.

```
$ icinga2 feature list
Disabled features: command debuglog icingastatus compatlog ...
Enabled features: api checker mainlog notification
```

In der Datei *icinga2.conf* werden mittels »include« alle Dateien mit der Endung *.conf* im Verzeichnis */etc/icinga2/features-enabled* eingebunden. Die dort liegenden Dateien enthalten die Konfiguration des jeweiligen Features mit demselben Namen.

```
$ ls -l /etc/icinga2/features-enabled/
insgesamt 0
lwrx... api.conf     -> ../features-available/api.conf
lrwx... checker.conf -> ../features-available/checker.conf
lwrx... mainlog.conf -> ../features-available/mainlog.conf
lwrx... notification.conf -> ../features-a.../notification.conf
```

Ist ein Feature aktiviert, zeigt ein Link aus diesem Verzeichnis auf die entsprechende Datei in */etc/icinga2/features-available*.

```
$ ls /etc/icinga2/features-available/
api.conf          command.conf        gelf.conf
ido-mysql.conf    mainlog.conf        statusdata.conf
compatlog.conf    perfdata.conf       graphite.conf
ido-pgsql.conf    notification.conf   syslog.conf
checker.conf      debuglog.conf       icingastatus.conf
livestatus.conf
```


Aktiviert man beispielsweise das Feature *api* mit `icinga2 feature enable api` oder deaktiviert es entsprechend mit `icinga2 feature disable api`, muss Icinga 2 danach jeweils neu gestartet werden.

Hier folgten eine Auflistung und eine Kurzbeschreibung der Features und ihrer Funktion. Die Features sind jeweils eigene Objekte und werden entsprechend über Attribute konfiguriert. Standardmäßig sollte die Konfiguration über oben stehende Dateien erfolgen. Ausführlich werden sie in den entsprechenden Kapiteln, in denen sie Anwendung finden, erläutert.

ApiListener

Mit *api* lässt sich der API Listener steuern. Dieser öffnet einen TCP-Socket, welcher sowohl HTTPS für die REST-API als auch JSON-RPC für die Cluster-Kommunikation versteht. Letzteres ist Voraussetzung für den

Datenaustausch mit anderen Icinga 2 Instanzen. Andere Instanzen können dabei Satellitensysteme oder auch Icinga 2 Agenten sein.

CheckerComponent

Das Feature *checker* regelt das Ausführen der aktiven Checks, also den Aufruf der Plugins und das Entgegennehmen der Ergebnisse, d. h. den Exit Code, den Output und Performance-Daten. Zusätzlich obliegt ihm die Verantwortung, wann was zu geschehen hat, und dieses einzuplanen (Scheduling), z. B. auch das Handling von Downtimes.

CheckResultReader

Bildet zur Kompatibilität das Verzeichnis `checkresults` von Icinga 1 nach. Dies erleichtert die Migration von verteilten Monitorumgebungen. Es gibt hierfür kein Feature, das beschriebene Verhalten kann aber leicht in einer beliebigen Konfigurationsdatei aktiviert werden:

```
object CheckResultReader "reader" {
  spool_dir = "/var/cache/icinga2/checkresults"
}
```

CompatLogger

Das *compatlog* kümmert sich um ein zu Icinga 1 kompatibles Logfile `icinga.log` und dessen Rotation. Der Ablageort für das aktuelle Logfile ist `/var/log/icinga2/compat` und für die archivierten Dateien unterhalb von `archives`. Das Feature wird z. B. für die Icinga Classic UI benötigt, um historische Daten anzuzeigen oder auszuwerten.

GelfWriter

Bietet mit *gelf* eine Schnittstelle (Graylog Extended Log Format) zu Graylog[2].

ExternalCommandListener

Mit dem Aktivieren des Features *command* wird die Commandpipe `/var/run/icinga2/cmd/icinga2.cmd` zur Verfügung gestellt. Dabei handelt es sich um einen Unix-Socket, in den Anweisungen in einer von Icinga 1 übernommenen Syntax direkt an Icinga 2 übergeben werden können. Über diese kann Icinga 2 weitreichend gesteuert werden.

[2] http://www.graylog.org

So können Checkergebnisse eingereicht, Downtimes geplant, Probleme bestätigt und einiges mehr über diesen Weg in den Icinga 2 Kern gebracht werden.

Dieses Feature ist noch aus Kompatibilitätsgründen enthalten und sollte mit Icinga 2 und Icinga Web 2 jeweils ab Version 2.4 durch die Verwendung der REST-API ersetzt werden.

FileLogger

Mit *mainlog* wird das Logfile `/var/log/icinga2/icinga2.log` geschrieben. Das Logging lässt sich in den Abstufungen *error*, *warning*, *notice*, *information* und *debug* konfigurieren.

Über das Feature *debuglog* kann das Logfile `debug.log` für Debug-Meldungen angeschaltet werden. Es befindet sich ebenfalls im Verzeichnis `/var/log/icinga`.

Die Logfiles werden nicht automatisch durch Icinga 2 rotiert, hier muss auf einen externen Mechanismus zurückgegriffen werden. Bei der Installation aus Paketen ist `logrotate` hierfür vorkonfiguriert.

GraphiteWriter

Das Objekt *GraphiteWriter* im Feature *graphite* schreibt Performance-Daten als Metriken in den TCP-Socket, den der Carbon-Cache des Graphite-Projekts[3] oder einer alternativen Implementierung bereitstellt.

InfluxdbWriter

Stellt im Feature *influxdb* eine Schnittstelle zu InfluxDB[4] zur Verfügung zur Weiterverarbeitung von Metriken und Performance-Daten.

ElasticsearchWriter

Eine weitere, in Icinga 2 Version 2.8 neue hinzugekommene Schnittstelle im Feature *elasticsearch*, die Metriken weiterreicht, im geforderten Format für Elasticsearch[5].

OpenTsdbWriter

Schreibt Metriken und Perfomance-Daten an eine OpenTSDB[6].

[3] http://graphite.wikidot.com
[4] https://www.influxdata.com
[5] https://www.elastic.co
[6] http://opentsdb.net

PerfdataWriter

Schreibt mit `perfdata` rotiert Dateien mit Performance-Daten für die Weiterverarbeitung mit PNP4Nagios[7] nach `/var/spool/icinga2/perfdata`.

IdoMysqlConnection

Dieses Feature ist aus dem Paket `icinga2-ido-mysql` gesondert zu installieren. Mit *ido-mysql* werden die anfallenden Daten, wie aktueller Status, Statuswechsel und vieles mehr, in eine MySQL-Datenbank geschrieben. Von dort können sie dann von Add-ons wie Icinga Web 2 oder NagVis[8] ausgelesen werden.

IdoPgsqlConnection

Feature *ido-pgsql* zum Ablegen der selben Daten wie bei IdoPgsqlConnection in eine PostgreSQL-Datenbank. Für dieses Feature ist das Paket `icinga2-ido-pgsql` zusätzlich zu installieren.

LivestatusWriter

Aktiviert mit *livestatus* die Schnittstelle selbigen Namens. Verfügbar als TCP- oder UNIX-Socket. Um historische Daten über Livestatus abfragen zu können, wird außerdem der *CompatLogger* benötigt.

NotificationComponent

An- oder Abschalten von *notification* aktiviert bzw. deaktiviert das Benachrichtigungssystem von Icinga 2.

StatusDataWriter

Feature *statusdata* zur Kompatibilität schreibt die Dateien *`status.dat`*, *`object.cache`* in das Verzeichnis */var/cache/icinga2*, die erforderlich sind, um aktuelle Daten in zu Icinga Web 2 alternativen Frontends anzuzeigen.

SyslogLogger

Leitet mit der Aktivierung von *syslogq* die Logeinträge in ein `syslog`-Target um. Die Verwendung von *syslog* anstelle von *mainlog* reduziert den

[7] http://www.pnp4nagios.org
[8] http://www.nagvis.org

Plattenzugriff. Syslog schreibt nicht unmittelbar auf die Platte, sondern puffert Einträge erst, um sie dann gesammelt zu schreiben, und kann auch genutzt werden, um die Logs nicht »nur« am Icinga 2 Host abzulegen, sondern auch an einen zentralen Loghost zu senden.