Contents

1 About Icinga 2

1.1 What is Icinga 2?

Icinga 2 is an open source monitoring system which checks the availability of your network resources, notifies users of outages and generates performance data for reporting.

Scalable and extensible, Icinga 2 can monitor large, complex environments across multiple locations.

1.2 Licensing

Icinga 2 and the Icinga 2 documentation are licensed under the terms of the GNU General Public License Version 2. You will find a copy of this license in the LICENSE file included in the source package.

1.3 Support

Check the project website at https://www.icinga.com for status updates. Join the community channels for questions or ask an Icinga partner for professional support.

1.4 Contribute

There are many ways to contribute to Icinga – whether it be sending patches, testing, reporting bugs or reviewing and updating the documentation. Every contribution is appreciated!

Please continue reading in the Contributing chapter.

1.4.1 Icinga 2 Development

The Git repository is located on GitHub.

Icinga 2 is written in C++ and can be built on Linux/Unix and Windows. Read more about development builds in the INSTALL.md file.

1.5 What's New

The Icinga 2 Changelog is located here. Please follow our release announcements on icinga.com too.

2 Getting Started

This tutorial is a step-by-step introduction to installing Icinga 2 and Icinga Web 2. It assumes that you are familiar with the operating system you're using to install Icinga 2.

In case you are upgrading an existing setup, please ensure to follow the upgrade documentation.

2.1 Setting up Icinga 2

First off you have to install Icinga 2. The preferred way of doing this is to use the official package repositories depending on which operating system and distribution you are running.

Distribution	Repository
Debian	Icinga Repository
Ubuntu	Icinga Repository
RHEL/CentOS	Icinga Repository
openSUSE	Icinga Repository
SLES	Icinga Repository
Gentoo	Upstream
FreeBSD	Upstream
OpenBSD	Upstream
ArchLinux	Upstream
Alpine Linux	Upstream

Packages for distributions other than the ones listed above may also be available. Please contact your distribution packagers.

2.1.1 Package Repositories

You need to add the Icinga repository to your package management configuration. Below is a list with examples for the various distributions.

Debian:

```
# wget -O - https://packages.icinga.com/icinga.key | apt-key add -
# echo 'deb https://packages.icinga.com/debian icinga-stretch main' >/etc/apt/sources.list.d/i
# apt-get update
```

Ubuntu:

```
# wget -O - https://packages.icinga.com/icinga.key | apt-key add -
# echo 'deb https://packages.icinga.com/ubuntu icinga-xenial main' >/etc/apt/sources.list.d/ic
# apt-get update
```

RHEL/CentOS 7:

```
yum install https://packages.icinga.com/epel/icinga-rpm-release-7-latest.noarch.rpm
```

RHEL/CentOS 6:

```
yum install https://packages.icinga.com/epel/icinga-rpm-release-6-latest.noarch.rpm
```

Fedora 26:

```
dnf install https://packages.icinga.com/fedora/icinga-rpm-release-26-latest.noarch.rpm
```

Fedora 25:

```
dnf install https://packages.icinga.com/fedora/icinga-rpm-release-25-latest.noarch.rpm
```

SLES 11:

```
# zypper ar https://packages.icinga.com/SUSE/ICINGA-release-11.repo
# zypper ref
```

SLES 12:

```
# zypper ar https://packages.icinga.com/SUSE/ICINGA-release.repo
# zypper ref
```

openSUSE:

```
# zypper ar https://packages.icinga.com/openSUSE/ICINGA-release.repo
# zypper ref
```

Alpine Linux:

```
# echo "http://dl-cdn.alpinelinux.org/alpine/edge/community" >> /etc/apk/repositories
# apk update
```

2.1.1.1 RHEL/CentOS EPEL Repository

The packages for RHEL/CentOS depend on other packages which are distributed as part of the EPEL repository.

CentOS 7/6:

```
yum install epel-release
```

If you are using RHEL you need to enable the `optional` repository and then install the EPEL rpm package.

2.1.1.2 SLES Security Repository

The packages for SLES 11 depend on the `openssl1` package which is distributed as part of the SLES 11 Security Module.

2.1.1.3 SLES 12 SDK

Icinga 2 requires the `libboost_chrono1_54_0` package from the `SLES 12 SDK` repository. Refer to the SUSE Enterprise Linux documentation for further information.

2.1.1.4 Alpine Linux Notes

The example provided assumes that you are running Alpine edge, which is the -dev branch and is a rolling release. If you are using a stable version please "pin" the edge repository on the latest Icinga 2 package version. In order to correctly manage your repository, please follow these instructions

2.1.2 Installing Icinga 2

You can install Icinga 2 by using your distribution's package manager to install the `icinga2` package.

Debian/Ubuntu:

```
# apt-get install icinga2
```

RHEL/CentOS 6:

```
# yum install icinga2
# chkconfig icinga2 on
# service icinga2 start
```

RHEL/CentOS 7 and Fedora:

```
# yum install icinga2
# systemctl enable icinga2
# systemctl start icinga2
```

SLES/openSUSE:

```
# zypper install icinga2
```

FreeBSD:

```
# pkg install icinga2
```

Alpine Linux:

```
# apk add icinga2
```

2.1.3 Enabled Features during Installation

The default installation will enable three features required for a basic Icinga 2 installation:

- `checker` for executing checks
- `notification` for sending notifications
- `mainlog` for writing the `icinga2.log` file

You can verify that by calling `icinga2 feature list` CLI command to see which features are enabled and disabled.

```
# icinga2 feature list
Disabled features: api command compatlog debuglog gelf graphite icingastatus ido-mysql ido-pgsq
Enabled features: checker mainlog notification
```

2.1.4 Installation Paths

By default Icinga 2 uses the following files and directories:

Path	Description
/etc/icinga2	Contains Icinga 2 configuration files.
/usr/lib/systemd/system/icinga2.service	The Icinga 2 Systemd service file on systems using Systemd.
/etc/systemd/system/icinga2.service.d/limits.conf	On distributions with Systemd >227, additional service limits are required.
/etc/init.d/icinga2	The Icinga 2 init script on systems using SysVinit or OpenRC.
/usr/sbin/icinga2	Shell wrapper for the Icinga 2 binary.

Path	Description
/usr/lib*/icinga2	Libraries and the Icinga 2 binary (use `find /usr -type f -name icinga2` to locate the binary path).
/usr/share/doc/icinga2	Documentation files that come with Icinga 2.
/usr/share/icinga2/include	The Icinga Template Library and plugin command configuration.
/var/lib/icinga2	Icinga 2 state file, cluster log, master CA, node certificates and configuration files (cluster, api).
/var/run/icinga2	PID file.
/var/run/icinga2/cmd	Command pipe and Livestatus socket.
/var/cache/icinga2	status.dat/objects.cache, icinga2.debug files.
/var/spool/icinga2	Used for performance data spool files.
/var/log/icinga2	Log file location and compat/ directory for the CompatLogger feature.

FreeBSD uses slightly different paths:

By default Icinga 2 uses the following files and directories:

Path	Description
/usr/local/etc/icinga2	Contains Icinga 2 configuration files.
/usr/local/etc/rc.d/icinga2	The Icinga 2 init script.
/usr/local/sbin/icinga2	Shell wrapper for the Icinga 2 binary.
/usr/local/lib/icinga2	Libraries and the Icinga 2 binary.
/usr/local/share/doc/icinga2	Documentation files that come with Icinga 2.
/usr/local/share/icinga2/include	The Icinga Template Library and plugin command configuration.
/var/lib/icinga2	Icinga 2 state file, cluster log, master CA, node certificates and configuration files (cluster, api).
/var/run/icinga2	PID file.
/var/run/icinga2/cmd	Command pipe and Livestatus socket.

Path	Description
/var/cache/icinga2	status.dat/objects.cache, icinga2.debug files.
/var/spool/icinga2	Used for performance data spool files.
/var/log/icinga2	Log file location and compat/ directory for the CompatLogger feature.

2.2 Setting up Check Plugins

Without plugins Icinga 2 does not know how to check external services. The Monitoring Plugins Project provides an extensive set of plugins which can be used with Icinga 2 to check whether services are working properly.

These plugins are required to make the example configuration work out-of-the-box.

For your convenience here is a list of package names for some of the more popular operating systems/distributions:

OS/Distribution	Package Name	Repository	Installation Path
RHEL/CentOS	nagios-plugins-all	EPEL	/usr/lib/nagios/plugins or /usr/lib64/nagios/plugins
SLES/OpenSUSE	monitoring-plugins	server:monitoring	/usr/lib/nagios/plugins
Debian/Ubuntu	monitoring-plugins	-	/usr/lib/nagios/plugins
FreeBSD	monitoring-plugins	-	/usr/local/libexec/nagios
Alpine Linux	monitoring-plugins	-	/usr/lib/monitoring-plugins
OS X	nagios-plugins	MacPorts, Homebrew	/opt/local/libexec or /usr/local/sbin

The recommended way of installing these standard plugins is to use your distribution's package manager.

Debian/Ubuntu:

```
# apt-get install monitoring-plugins
```

RHEL/CentOS:

```
# yum install nagios-plugins-all
```

The packages for RHEL/CentOS depend on other packages which are distributed as part of the EPEL repository. Please make sure to enable this repository by following these instructions.

Fedora:

```
# dnf install nagios-plugins-all
```

SLES/openSUSE:

```
# zypper install monitoring-plugins
```

The packages for SLES/OpenSUSE depend on other packages which are distributed as part of the server:monitoring repository. Please make sure to enable this repository beforehand.

FreeBSD:

```
# pkg install monitoring-plugins
```

Alpine Linux:

```
# apk add monitoring-plugins
```

Note: For Alpine you don't need to explicitly add the `monitoring-plugins` package since it is a dependency of `icinga2` and is pulled automatically.

Depending on which directory your plugins are installed into you may need to update the global `PluginDir` constant in your Icinga 2 configuration. This constant is used by the check command definitions contained in the Icinga Template Library to determine where to find the plugin binaries.

Note

Please refer to the service monitoring chapter for details about how to integrate additional check plugins into your Icinga 2 setup.

2.3 Running Icinga 2

2.3.1 Init Script

Icinga 2's init script is installed in `/etc/init.d/icinga2` (`/usr/local/etc/rc.d/icinga2` on FreeBSD) by default:

```
# /etc/init.d/icinga2
Usage: /etc/init.d/icinga2 {start|stop|restart|reload|checkconfig|status}
```

The init script supports the following actions:

Command	Description
start	The `start` action starts the Icinga 2 daemon.
stop	The `stop` action stops the Icinga 2 daemon.
restart	The `restart` action is a shortcut for running the `stop` action followed by `start`.
reload	The `reload` action sends the `HUP` signal to Icinga 2 which causes it to restart. Unlike the `restart` action `reload` does not wait until Icinga 2 has restarted.
checkconfig	The `checkconfig` action checks if the `/etc/icinga2/icinga2.conf` configuration file contains any errors.
status	The `status` action checks if Icinga 2 is running.

By default, the Icinga 2 daemon is running as `icinga` user and group using the init script. Using Debian packages the user and group are set to `nagios` for historical reasons.

2.3.2 Systemd Service

Some distributions (e.g. Fedora, openSUSE and RHEL/CentOS 7) use Systemd. The Icinga 2 packages automatically install the necessary Systemd unit files.

The Icinga 2 Systemd service can be (re-)started, reloaded, stopped and also queried for its current status.

```
# systemctl status icinga2
icinga2.service - Icinga host/service/network monitoring system
  Loaded: loaded (/usr/lib/systemd/system/icinga2.service; disabled)
  Active: active (running) since Mi 2014-07-23 13:39:38 CEST; 15s ago
 Process: 21692 ExecStart=/usr/sbin/icinga2 -c ${ICINGA2_CONFIG_FILE} -d -e ${ICINGA2_ERROR_LO
 Process: 21674 ExecStartPre=/usr/sbin/icinga2-prepare-dirs /etc/sysconfig/icinga2 (code=exit
Main PID: 21727 (icinga2)
   CGroup: /system.slice/icinga2.service
```

```
    21727 /usr/sbin/icinga2 -c /etc/icinga2/icinga2.conf -d -e /var/log/icinga2/error.log -

Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif icinga2[21692]: [2014-07-23 13:39:38 +0200] information/ConfigItem: Check
Jul 23 13:39:38 nbmif systemd[1]: Started Icinga host/service/network monitoring system.
```

The `systemctl` command supports the following actions:

Command	Description
start	The `start` action starts the Icinga 2 daemon.
stop	The `stop` action stops the Icinga 2 daemon.
restart	The `restart` action is a shortcut for running the `stop` action followed by `start`.
reload	The `reload` action sends the `HUP` signal to Icinga 2 which causes it to restart. Unlike the `restart` action `reload` does not wait until Icinga 2 has restarted.
status	The `status` action checks if Icinga 2 is running.
enable	The `enable` action enables the service being started at system boot time (similar to `chkconfig`)

Examples:

```
# systemctl enable icinga2

# systemctl restart icinga2
Job for icinga2.service failed. See 'systemctl status icinga2.service' and 'journalctl -xn' for
```

If you're stuck with configuration errors, you can manually invoke the configuration validation.

Usually Icinga 2 is a mission critical part of infrastructure and should be online at all times. In case of a recoverable crash (e.g. OOM) you may want to restart Icinga 2 automatically. With Systemd it is as easy as overriding some settings of the Icinga 2 Systemd service by creating `/etc/systemd/system/icinga2.service.d/override.conf` with the following content:

```
[Service]
Restart=always
RestartSec=1
StartLimitInterval=10
StartLimitBurst=3
```

Using the watchdog can also help with monitoring Icinga 2, to activate and use it add the following to the override:

```
WatchdogSec=30s
```

This way Systemd will kill Icinga 2 if does not notify for over 30 seconds, a timout of less than 10 seconds is not recommended. When the watchdog is activated, `Restart=` can be set to `watchdog` to restart Icinga 2 in the case of a watchdog timeout.

Run `systemctl daemon-reload && systemctl restart icinga2` to apply the changes. Now Systemd will always try to restart Icinga 2 (except if you run `systemctl stop icinga2`). After three failures in ten seconds it will stop trying because you probably have a problem that requires manual intervention.

> **Tip**
>
> If you are running into fork errors with Systemd enabled distributions, please check the troubleshooting chapter.

2.3.3 FreeBSD

On FreeBSD you need to enable icinga2 in your rc.conf

```
# sysrc icinga2_enable=yes

# service icinga2 restart
```

2.3.4 SELinux

SELinux is a mandatory access control (MAC) system on Linux which adds a fine-grained permission system for access to all system resources such as files, devices, networks and inter-process communication.

Icinga 2 provides its own SELinux policy. `icinga2-selinux` is a policy package for Red Hat Enterprise Linux 7 and derivatives. The package runs the

targeted policy which confines Icinga 2 including enabled features and running commands.

RHEL/CentOS 7:

```
yum install icinga2-selinux
```

Fedora:

```
dnf install icinga2-selinux
```

Read more about SELinux in this chapter.

2.4 Configuration Syntax Highlighting

Icinga 2 ships configuration examples for syntax highlighting using the `vim` and `nano` editors. The RHEL and SUSE package `icinga2-common` installs these files into `/usr/share/doc/icinga2-common-[x.x.x]/syntax` (where `[x.x.x]` is the version number, e.g. `2.4.3` or `2.4.4`). Sources provide these files in `tools/syntax`. On Debian systems the `icinga2-common` package provides only the Nano configuration file (`/usr/share/nano/icinga2.nanorc`); to obtain the Vim configuration, please install the extra package `vim-icinga2`. The files are located in `/usr/share/vim/addons`.

2.4.1 Configuration Syntax Highlighting using Vim

Install the package `vim-icinga2` with your distribution's package manager.

Debian/Ubuntu:

```
# apt-get install vim-icinga2 vim-addon-manager
# vim-addon-manager -w install icinga2
Info: installing removed addon 'icinga2' to /var/lib/vim/addons
```

RHEL/CentOS/Fedora:

```
# yum install vim-icinga2
```

SLES/openSUSE:

```
# zypper install vim-icinga2
```

Alpine Linux:

```
# apk add icinga2-vim
```

Ensure that syntax highlighting is enabled e.g. by editing the user's `vimrc` configuration file:

```
# vim ~/.vimrc
syntax on
```

Test it:

```
# vim /etc/icinga2/conf.d/templates.conf
```

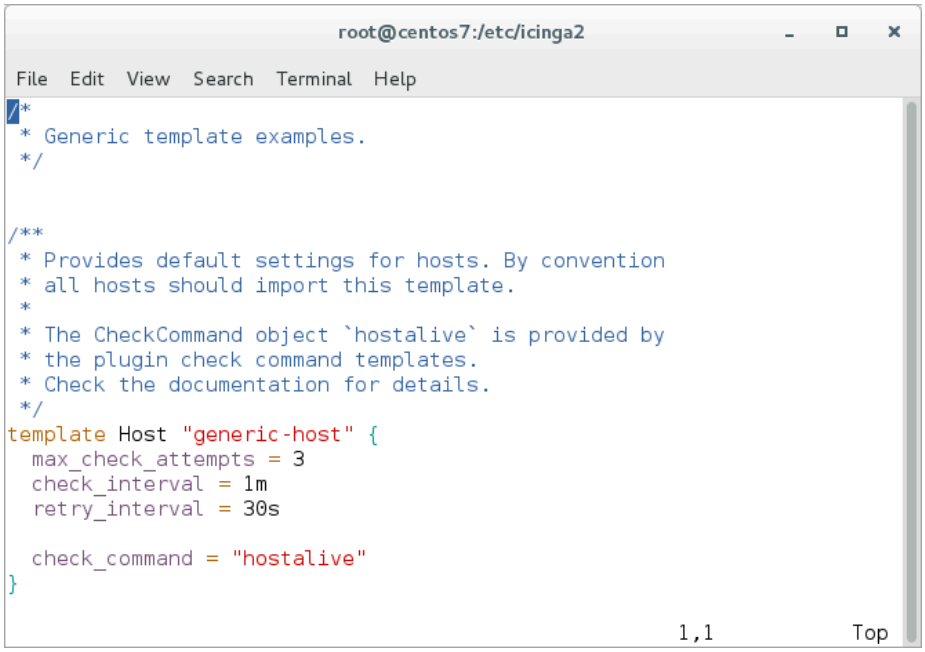


Figure 1: Vim with syntax highlighting

2.4.2 Configuration Syntax Highlighting using Nano

Install the package `nano-icinga2` with your distribution's package manager.

Debian/Ubuntu:

Note: The syntax files are installed with the `icinga2-common` package already.

RHEL/CentOS/Fedora:

```
# yum install nano-icinga2
```

SLES/openSUSE:

```
# zypper install nano-icinga2
```

Copy the `/etc/nanorc` sample file to your home directory.

```
$ cp /etc/nanorc ~/.nanorc
```

Include the `icinga2.nanorc` file.

```
$ vim ~/.nanorc

## Icinga 2
include "/usr/share/nano/icinga2.nanorc"
```

Test it:

```
$ nano /etc/icinga2/conf.d/templates.conf
```

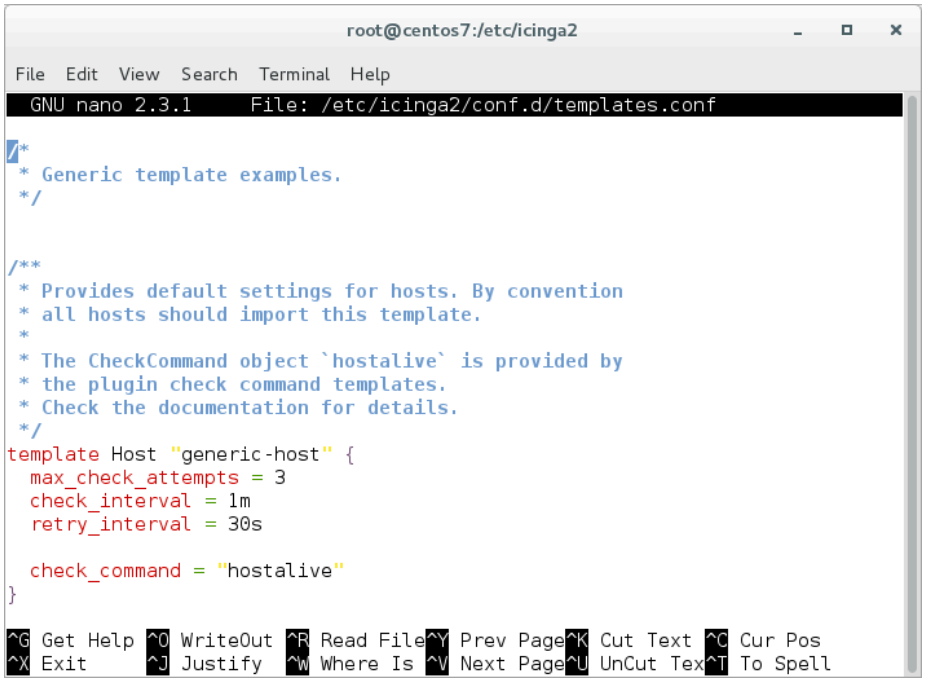


Figure 2: Nano with syntax highlighting

2.5 Setting up Icinga Web 2

Icinga 2 can be used with Icinga Web 2 and a number of other web interfaces. This chapter explains how to set up Icinga Web 2.

The DB IDO (Database Icinga Data Output) modules for Icinga 2 take care of exporting all configuration and status information into a database. The IDO database is used by a number of projects including Icinga Web 2, Icinga Reporting or Icinga Web 1.x.

There is a separate module for each database backend. At present support for both MySQL and PostgreSQL has been implemented.

Please choose whether to install MySQL or PostgreSQL.

2.5.1 Configuring DB IDO MySQL

2.5.1.1 Installing MySQL database server

Debian/Ubuntu:

```
# apt-get install mysql-server mysql-client
# mysql_secure_installation
```

RHEL/CentOS 6:

```
# yum install mysql-server mysql
# chkconfig mysqld on
# service mysqld start
# mysql_secure_installation
```

RHEL/CentOS 7 and Fedora:

```
# yum install mariadb-server mariadb
# systemctl enable mariadb
# systemctl start mariadb
# mysql_secure_installation
```

SUSE:

```
# zypper install mysql mysql-client
# chkconfig mysqld on
# service mysqld start
```

FreeBSD:

```
# pkg install mysql56-server
# sysrc mysql_enable=yes
# service mysql-server restart
# mysql_secure_installation
```

Alpine Linux:

```
# apk add mariadb
# rc-service mariadb setup
# rc-update add mariadb default
# rc-service mariadb start
```

2.5.1.2 Installing the IDO modules for MySQL

The next step is to install the `icinga2-ido-mysql` package using your distribution's package manager.

Debian/Ubuntu:

```
# apt-get install icinga2-ido-mysql
```

RHEL/CentOS:

```
# yum install icinga2-ido-mysql
```

SUSE:

```
# zypper install icinga2-ido-mysql
```

FreeBSD:

On FreeBSD the IDO modules for MySQL are included with the icinga2 package and located at /usr/local/share/icinga2-ido-mysql/schema/mysql.sql

Alpine Linux:

On Alpine Linux the IDO modules for MySQL are included with the `icinga2` package and located at /usr/share/icinga2-ido-mysql/schema/mysql.sql

> **Note**
>
> The Debian/Ubuntu packages provide a database configuration wizard by default. You can skip the automated setup and install/upgrade the database manually if you prefer that.

2.5.1.3 Setting up the MySQL database

Set up a MySQL database for Icinga 2:

```
# mysql -u root -p

mysql>  CREATE DATABASE icinga;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, DROP, CREATE VIEW, INDEX, EXECUTE ON icinga.* TO 'i
mysql> quit
```

After creating the database you can import the Icinga 2 IDO schema using the following command:

```
# mysql -u root -p icinga < /usr/share/icinga2-ido-mysql/schema/mysql.sql
```

2.5.1.4 Enabling the IDO MySQL module

The package provides a new configuration file that is installed in `/etc/icinga2/features-available/ido-mys`
You will need to update the database credentials in this file.

All available attributes are explained in the IdoMysqlConnection object chapter.

You can enable the `ido-mysql` feature configuration file using `icinga2 feature enable`:

```
# icinga2 feature enable ido-mysql
Module 'ido-mysql' was enabled.
Make sure to restart Icinga 2 for these changes to take effect.
```

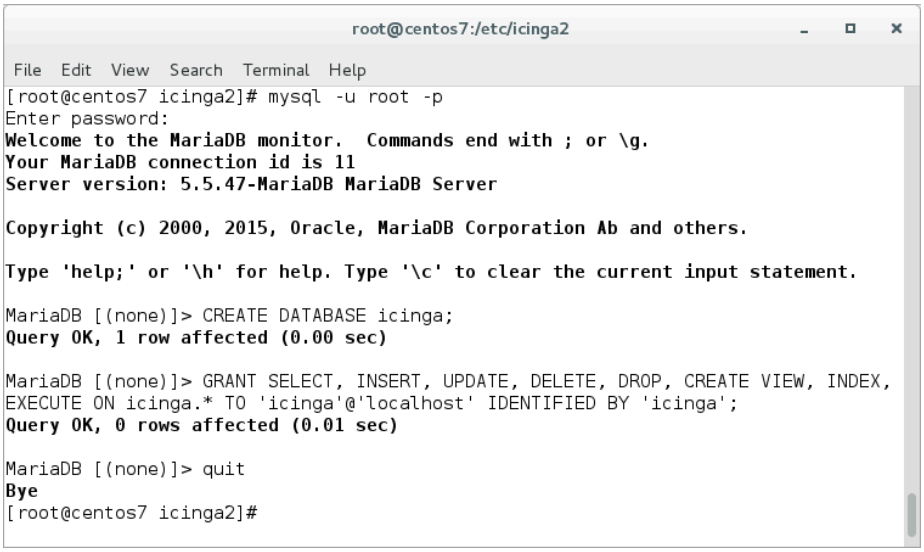


Figure 3: setting up the database on CentOS 7

After enabling the ido-mysql feature you have to restart Icinga 2:

RHEL/CentOS 7/Fedora, SLES 12/openSUSE > 12.2, Debian Jessie/Stretch, Ubuntu Xenial:

```
# systemctl restart icinga2
```

Debian/Ubuntu, RHEL/CentOS 6, SLES 11/openSUSE < 12.3 and FreeBSD:

```
# service icinga2 restart
```

Alpine Linux:

rc-service icinga2 restart

Continue with the webserver setup.

2.5.2 Configuring DB IDO PostgreSQL

2.5.2.1 Installing PostgreSQL database server

Debian/Ubuntu:

```
# apt-get install postgresql
```

RHEL/CentOS 6:

```
# yum install postgresql-server postgresql
# chkconfig postgresql on
# service postgresql initdb
```

```
# service postgresql start
```

RHEL/CentOS 7:

```
# yum install postgresql-server postgresql
# postgresql-setup initdb
# systemctl enable postgresql
# systemctl start postgresql
```

SUSE:

```
# zypper install postgresql postgresql-server
# chkconfig postgresql on
# service postgresql initdb
# service postgresql start
```

FreeBSD:

```
# pkg install postgresql93-server
# sysrc postgresql_enable=yes
# service postgresql initdb
# service postgresql start
```

Alpine Linux:

```
# apk add postgresql
# rc-update add postgresql default
# rc-service postgresql setup
# rc-service postgresql start
```

2.5.2.2 Installing the IDO modules for PostgreSQL

The next step is to install the `icinga2-ido-pgsql` package using your distribution's package manager.

Debian/Ubuntu:

```
# apt-get install icinga2-ido-pgsql
```

RHEL/CentOS:

```
# yum install icinga2-ido-pgsql
```

SUSE:

```
# zypper install icinga2-ido-pgsql
```

FreeBSD:

On FreeBSD the IDO modules for PostgreSQL are included with the icinga2 package and located at /usr/local/share/icinga2-ido-pgsql/schema/pgsql.sql

Alpine Linux:

On Alpine Linux the IDO modules for PostgreSQL are included with the `icinga2` package and located at /usr/share/icinga2-ido-pgsql/schema/pgsql.sql

Note

Upstream Debian packages provide a database configuration wizard by default. You can skip the automated setup and install/upgrade the database manually if you prefer that.

2.5.2.3 Setting up the PostgreSQL database

Set up a PostgreSQL database for Icinga 2:

```
# cd /tmp
# sudo -u postgres psql -c "CREATE ROLE icinga WITH LOGIN PASSWORD 'icinga'"
# sudo -u postgres createdb -O icinga -E UTF8 icinga
# sudo -u postgres createlang plpgsql icinga
```

Note

When using PostgreSQL 9.x you can omit the `createlang` command. Also it is assumed here that your locale is set to utf-8, you may run into problems otherwise.

Locate your pg_hba.conf (Debian: `/etc/postgresql/*/main/pg_hba.conf`, RHEL/SUSE: `/var/lib/pgsql/data/pg_hba.conf`), add the icinga user with md5 authentication method and restart the postgresql server.

```
# icinga
local   icinga      icinga                          md5
host    icinga      icinga      127.0.0.1/32        md5
host    icinga      icinga      ::1/128             md5

# "local" is for Unix domain socket connections only
local   all         all                             ident
# IPv4 local connections:
host    all         all         127.0.0.1/32        ident
# IPv6 local connections:
host    all         all         ::1/128             ident

# service postgresql restart
```

After creating the database and permissions you can import the Icinga 2 IDO schema using the following command:

```
# export PGPASSWORD=icinga
# psql -U icinga -d icinga < /usr/share/icinga2-ido-pgsql/schema/pgsql.sql
```

Figure 4: importing the Icinga 2 IDO schema

2.5.2.4 Enabling the IDO PostgreSQL module

The package provides a new configuration file that is installed in `/etc/icinga2/features-available/ido-pgs`
You will need to update the database credentials in this file.

All available attributes are explained in the IdoPgsqlConnection object chapter.

You can enable the `ido-pgsql` feature configuration file using `icinga2 feature enable`:

```
# icinga2 feature enable ido-pgsql
Module 'ido-pgsql' was enabled.
Make sure to restart Icinga 2 for these changes to take effect.
```

After enabling the ido-pgsql feature you have to restart Icinga 2:

RHEL/CentOS 7/Fedora, SLES 12/openSUSE > 12.2, Debian Jessie/Stretch, Ubuntu Xenial:

```
# systemctl restart icinga2
```

Debian/Ubuntu, RHEL/CentOS 6, SLES 11/openSUSE < 12.3 and FreeBSD:

```
# service icinga2 restart
```

Alpine Linux:

```
# rc-service icinga2 restart
```

Continue with the webserver setup.

2.5.3 Webserver

Debian/Ubuntu:

```
# apt-get install apache2
```

RHEL/CentOS 6:

```
# yum install httpd
# chkconfig httpd on
# service httpd start
```

RHEL/CentOS 7, Fedora:

```
# yum install httpd
# systemctl enable httpd
# systemctl start httpd
```

SUSE:

```
# zypper install apache2
# chkconfig on
# service apache2 start
```

FreeBSD (nginx, but you could also use the apache24 package):

```
# pkg install nginx php56-gettext php56-ldap php56-openssl php56-mysql php56-pdo_mysql php56-pg
# sysrc php_fpm_enable=yes
# sysrc nginx_enable=yes
# sed -i '' "s/listen\ =\ 127.0.0.1:9000/listen\ =\ \/var\/run\/php5-fpm.sock/" /usr/local/etc/
# sed -i '' "s/;listen.owner/listen.owner/" /usr/local/etc/php-fpm.conf
# sed -i '' "s/;listen.group/listen.group/" /usr/local/etc/php-fpm.conf
# sed -i '' "s/;listen.mode/listen.mode/" /usr/local/etc/php-fpm.conf
# service php-fpm start
# service nginx start
```

Alpine Linux:

```
# apk add apache2 php7-apache2
# sed -i -e "s/^#LoadModule rewrite_module/LoadModule rewrite_module/" /etc/apache2/httpd.conf
# rc-update add apache2 default
# rc-service apache2 start
```

2.5.4 Firewall Rules

Example:

```
# iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
# service iptables save
```

RHEL/CentOS 7 specific:

```
# firewall-cmd --add-service=http
# firewall-cmd --permanent --add-service=http
```

FreeBSD: Please consult the FreeBSD Handbook how to configure one of FreeBSD's firewalls.

2.5.5 Setting Up Icinga 2 REST API

Icinga Web 2 and other web interfaces require the REST API to send actions (reschedule check, etc.) and query object details.

You can run the CLI command `icinga2 api setup` to enable the `api` feature and set up certificates as well as a new API user `root` with an auto-generated password in the `/etc/icinga2/conf.d/api-users.conf` configuration file:

```
# icinga2 api setup
```

Edit the `api-users.conf` file and add a new ApiUser object. Specify the permissions attribute with minimal permissions required by Icinga Web 2.

```
# vim /etc/icinga2/conf.d/api-users.conf

object ApiUser "icingaweb2" {
  password = "Wijsn8Z9eRs5E25d"
 permissions = [ "status/query", "actions/*", "objects/modify/*", "objects/query/*" ]
}
```

Make sure to restart Icinga 2 to activate the configuration.

RHEL/CentOS 7/Fedora, SLES 12/openSUSE > 12.2, Debian Jessie/Stretch, Ubuntu Xenial:

```
# systemctl restart icinga2
```

Debian/Ubuntu, RHEL/CentOS 6, SLES 11/openSUSE < 12.3 and FreeBSD:

```
# service icinga2 restart
```

Alpine Linux:

```
# rc-service icinga2 restart
```

2.5.6 Installing Icinga Web 2

Please consult the installation documentation for further instructions on how to install Icinga Web 2.

The Icinga 2 API can be defined as command transport in Icinga Web 2 >= 2.4.

2.6 Addons

A number of additional features are available in the form of addons. A list of popular addons is available in the Addons and Plugins chapter.

2.7 Backup

Ensure to include the following in your backups:

- Configuration files in `/etc/icinga2`
- Certificate files in `/var/lib/icinga2/ca` (Master CA key pair) and `/var/lib/icinga2/certs` (node certificates)
- Runtime files in `/var/lib/icinga2`
- Optional: IDO database backup

3 Monitoring Basics

This part of the Icinga 2 documentation provides an overview of all the basic monitoring concepts you need to know to run Icinga 2. Keep in mind these examples are made with a Linux server. If you are using Windows, you will need to change the services accordingly. See the ITL reference for further information.

3.1 Attribute Value Types

The Icinga 2 configuration uses different value types for attributes.

Type	Example
Number	`5`
Duration	`1m`
String	`"These are notes"`
Boolean	`true`
Array	`[ "value1", "value2" ]`
Dictionary	`{ "key1" = "value1", "key2" = false }`

It is important to use the correct value type for object attributes as otherwise the configuration validation will fail.

3.2 Hosts and Services

Icinga 2 can be used to monitor the availability of hosts and services. Hosts and services can be virtually anything which can be checked in some way:

- Network services (HTTP, SMTP, SNMP, SSH, etc.)
- Printers
- Switches or routers
- Temperature sensors
- Other local or network-accessible services

Host objects provide a mechanism to group services that are running on the same physical device.

Here is an example of a host object which defines two child services:

```
object Host "my-server1" {
  address = "10.0.0.1"
  check_command = "hostalive"
}

object Service "ping4" {
  host_name = "my-server1"
  check_command = "ping4"
}

object Service "http" {
  host_name = "my-server1"
  check_command = "http"
}
```

The example creates two services `ping4` and `http` which belong to the host `my-server1`.

It also specifies that the host should perform its own check using the `hostalive` check command.

The `address` attribute is used by check commands to determine which network address is associated with the host object.

Details on troubleshooting check problems can be found here.

3.2.1 Host States

Hosts can be in any one of the following states:

Name	Description
UP	The host is available.
DOWN	The host is unavailable.

3.2.2 Service States

Services can be in any one of the following states:

Name	Description
OK	The service is working properly.
WARNING	The service is experiencing some problems but is still considered to be in working condition.
CRITICAL	The service is in a critical state.
UNKNOWN	The check could not determine the service's state.

3.2.3 Check Result State Mapping

Check plugins return with an exit code which is converted into a state number. Services map the states directly while hosts will treat `0` or `1` as `UP` for example.

Value	Host State	Service State
0	Up	OK
1	Up	Warning
2	Down	Critical
3	Down	Unknown

3.2.4 Hard and Soft States

When detecting a problem with a host/service, Icinga re-checks the object a number of times (based on the `max_check_attempts` and `retry_interval` settings) before sending notifications. This ensures that no unnecessary notifications are sent for transient failures. During this time the object is in a `SOFT` state.

After all re-checks have been executed and the object is still in a non-OK state,

the host/service switches to a `HARD` state and notifications are sent.

Name	Description
HARD	The host/service's state hasn't recently changed. `check_interval` applies here.
SOFT	The host/service has recently changed state and is being re-checked with `retry_interval`.

3.2.5 Host and Service Checks

Hosts and services determine their state by running checks in a regular interval.

```
object Host "router" {
  check_command = "hostalive"
  address = "10.0.0.1"
}
```

The `hostalive` command is one of several built-in check commands. It sends ICMP echo requests to the IP address specified in the `address` attribute to determine whether a host is online.

> **Tip**
>
> `hostalive` is the same as `ping` but with different default thresholds. Both use the `ping` CLI command to execute sequential checks.
>
> If you need faster ICMP checks, look into the icmp CheckCommand.

A number of other built-in check commands are also available. In addition to these commands the next few chapters will explain in detail how to set up your own check commands.

3.2.5.1 Host Check Alternatives

If the host is not reachable with ICMP, HTTP, etc. you can also use the dummy CheckCommand to set a default state.

```
object Host "dummy-host" {
  check_command = "dummy"
  vars.dummy_state = 0 //Up
  vars.dummy_text = "Everything OK."
}
```

This method is also used when you send in external check results.

A more advanced technique is to calculate an overall state based on all services. This is described here.

3.3 Templates

Templates may be used to apply a set of identical attributes to more than one object:

```
template Service "generic-service" {
  max_check_attempts = 3
  check_interval = 5m
  retry_interval = 1m
  enable_perfdata = true
}

apply Service "ping4" {
  import "generic-service"

  check_command = "ping4"

  assign where host.address
}

apply Service "ping6" {
  import "generic-service"

  check_command = "ping6"

  assign where host.address6
}
```

In this example the `ping4` and `ping6` services inherit properties from the template `generic-service`.

Objects as well as templates themselves can import an arbitrary number of other templates. Attributes inherited from a template can be overridden in the object if necessary.

You can also import existing non-template objects.

Note

Templates and objects share the same namespace, i.e. you can't define a template that has the same name like an object.

3.3.1 Multiple Templates

The following example uses custom attributes which are provided in each template. The `web-server` template is used as the base template for any host providing web services. In addition to that it specifies the custom attribute `webserver_type`, e.g. `apache`. Since this template is also the base template, we import the `generic-host` template here. This provides the `check_command` attribute by default and we don't need to set it anywhere later on.

```
template Host "web-server" {
  import "generic-host"
  vars = {
    webserver_type = "apache"
  }
}
```

The `wp-server` host template specifies a Wordpress instance and sets the `application_type` custom attribute. Please note the `+=` operator which adds dictionary items, but does not override any previous `vars` attribute.

```
template Host "wp-server" {
  vars += {
    application_type = "wordpress"
  }
}
```

The final host object imports both templates. The order is important here: First the base template `web-server` is added to the object, then additional attributes are imported from the `wp-server` object.

```
object Host "wp.example.com" {
  import "web-server"
  import "wp-server"

  address = "192.168.56.200"
}
```

If you want to override specific attributes inherited from templates, you can specify them on the host object.

```
object Host "wp1.example.com" {
  import "web-server"
  import "wp-server"
```

```
  vars.webserver_type = "nginx" //overrides attribute from base template

  address = "192.168.56.201"
}
```

3.4 Custom Attributes

In addition to built-in attributes you can define your own attributes inside the `vars` attribute:

```
object Host "localhost" {
  check_command = "ssh"
  vars.ssh_port = 2222
}
```

`vars` is a dictionary where you can set specific keys to values. The example above uses the shorter indexer syntax.

An alternative representation can be written like this:

```
  vars = {
    ssh_port = 2222
  }
```

or

```
  vars["ssh_port"] = 2222
```

3.4.1 Custom Attribute Values

Valid values for custom attributes include:

- Strings, numbers and booleans
- Arrays and dictionaries
- Functions

You can also define nested values such as dictionaries in dictionaries.

This example defines the custom attribute `disks` as dictionary. The first key is set to `disk /` is itself set to a dictionary with one key-value pair.

```
  vars.disks["disk /"] = {
    disk_partitions = "/"
  }
```

This can be written as resolved structure like this:

```
  vars = {
    disks = {
      "disk /" = {
```

```
      disk_partitions = "/"
    }
  }
}
```

Keep this in mind when trying to access specific sub-keys in apply rules or functions.

Another example which is shown in the example configuration:

```
vars.notification["mail"] = {
  groups = [ "icingaadmins" ]
}
```

This defines the `notification` custom attribute as dictionary with the key `mail`. Its value is a dictionary with the key `groups` which itself has an array as value. Note: This array is the exact same as the `user_groups` attribute for notification apply rules expects.

```
vars.notification = {
  mail = {
    groups = [
      "icingaadmins"
    ]
  }
}
```

3.4.2 Functions as Custom Attributes

Icinga 2 lets you specify functions for custom attributes. The special case here is that whenever Icinga 2 needs the value for such a custom attribute it runs the function and uses whatever value the function returns:

```
object CheckCommand "random-value" {
  command = [ PluginDir + "/check_dummy", "0", "$text$" ]

  vars.text = {{ Math.random() * 100 }}
}
```

This example uses the abbreviated lambda syntax.

These functions have access to a number of variables:

Variable	Description
user	The User object (for notifications).

Variable	Description
service	The Service object (for service checks/notifications/event handlers).
host	The Host object.
command	The command object (e.g. a CheckCommand object for checks).

Here's an example:

```
vars.text = {{ host.check_interval }}
```

In addition to these variables the macro function can be used to retrieve the value of arbitrary macro expressions:

```
vars.text = {{
  if (macro("$address$") == "127.0.0.1") {
    log("Running a check for localhost!")
  }

  return "Some text"
}}
```

The `resolve_arguments` function can be used to resolve a command and its arguments much in the same fashion Icinga does this for the `command` and `arguments` attributes for commands. The `by_ssh` command uses this functionality to let users specify a command and arguments that should be executed via SSH:

```
arguments = {
  "-C" = {{
    var command = macro("$by_ssh_command$")
    var arguments = macro("$by_ssh_arguments$")

    if (typeof(command) == String && !arguments) {
      return command
    }

    var escaped_args = []
    for (arg in resolve_arguments(command, arguments)) {
      escaped_args.add(escape_shell_arg(arg))
```

```
    }
    return escaped_args.join(" ")
  }}
  ...
}
```

Accessing object attributes at runtime inside these functions is described in the advanced topics chapter.

3.5 Runtime Macros

Macros can be used to access other objects' attributes at runtime. For example they are used in command definitions to figure out which IP address a check should be run against:

```
object CheckCommand "my-ping" {
  command = [ PluginDir + "/check_ping", "-H", "$ping_address$" ]

  arguments = {
    "-w" = "$ping_wrta$,$ping_wpl$%"
    "-c" = "$ping_crta$,$ping_cpl$%"
    "-p" = "$ping_packets$"
  }

  vars.ping_address = "$address$"

  vars.ping_wrta = 100
  vars.ping_wpl = 5

  vars.ping_crta = 250
  vars.ping_cpl = 10

  vars.ping_packets = 5
}

object Host "router" {
  check_command = "my-ping"
  address = "10.0.0.1"
}
```

In this example we are using the `$address$` macro to refer to the host's `address` attribute.

We can also directly refer to custom attributes, e.g. by using `$ping_wrta$`. Icinga automatically tries to find the closest match for the attribute you specified. The exact rules for this are explained in the next section.

Note

When using the `$` sign as single character you must escape it with an additional dollar character (`$$`).

3.5.1 Evaluation Order

When executing commands Icinga 2 checks the following objects in this order to look up macros and their respective values:

1. User object (only for notifications)
2. Service object
3. Host object
4. Command object
5. Global custom attributes in the `Vars` constant

This execution order allows you to define default values for custom attributes in your command objects.

Here's how you can override the custom attribute `ping_packets` from the previous example:

```
object Service "ping" {
  host_name = "localhost"
  check_command = "my-ping"

  vars.ping_packets = 10 // Overrides the default value of 5 given in the command
}
```

If a custom attribute isn't defined anywhere, an empty value is used and a warning is written to the Icinga 2 log.

You can also directly refer to a specific attribute – thereby ignoring these evaluation rules – by specifying the full attribute name:

```
$service.vars.ping_wrta$
```

This retrieves the value of the `ping_wrta` custom attribute for the service. This returns an empty value if the service does not have such a custom attribute no matter whether another object such as the host has this attribute.

3.5.2 Host Runtime Macros

The following host custom attributes are available in all commands that are executed for hosts or services:

Name	Description
host.name	The name of the host object.

Name	Description
host.display_name	The value of the `display_name` attribute.
host.state	The host's current state. Can be one of `UNREACHABLE`, `UP` and `DOWN`.
host.state_id	The host's current state. Can be one of `0` (up), `1` (down) and `2` (unreachable).
host.state_type	The host's current state type. Can be one of `SOFT` and `HARD`.
host.check_attempt	The current check attempt number.
host.max_check_attempts	The maximum number of checks which are executed before changing to a hard state.
host.last_state	The host's previous state. Can be one of `UNREACHABLE`, `UP` and `DOWN`.
host.last_state_id	The host's previous state. Can be one of `0` (up), `1` (down) and `2` (unreachable).

Name	Description
host.last_state_type	The host's previous state type. Can be one of `SOFT` and `HARD`.
host.last_state_change	The last state change's timestamp.
host.downtime_depth	The number of active downtimes.
host.duration_sec	The time since the last state change.
host.latency	The host's check latency.
host.execution_time	The host's check execution time.
host.output	The last check's output.
host.perfdata	The last check's performance data.
host.last_check	The timestamp when the last check was executed.
host.check_source	The monitoring instance that performed the last check.
host.num_services	Number of services associated with the host.
host.num_services_ok	Number of services associated with the host which are in an `OK` state.

Name	Description
host.num_services_warning	Number of services associated with the host which are in a `WARNING` state.
host.num_services_unknown	Number of services associated with the host which are in an `UNKNOWN` state.
host.num_services_critical	Number of services associated with the host which are in a `CRITICAL` state.

In addition to these specific runtime macros host object attributes can be accessed too.

3.5.3 Service Runtime Macros

The following service macros are available in all commands that are executed for services:

Name	Description
service.name	The short name of the service object.
service.display_name	The value of the `display_name` attribute.
service.check_command	The short name of the command along with any arguments to be used for the check.

Name	Description
service.state	The service's current state. Can be one of `OK`, `WARNING`, `CRITICAL` and `UNKNOWN`.
service.state_id	The service's current state. Can be one of `0` (ok), `1` (warning), `2` (critical) and `3` (unknown).
service.state_type	The service's current state type. Can be one of `SOFT` and `HARD`.
service.check_attempt	The current check attempt number.
service.max_check_attempts	The maximum number of checks which are executed before changing to a hard state.
service.last_state	The service's previous state. Can be one of `OK`, `WARNING`, `CRITICAL` and `UNKNOWN`.
service.last_state_id	The service's previous state. Can be one of `0` (ok), `1` (warning), `2` (critical) and `3` (unknown).

Name	Description
service.last_state_type	The service's previous state type. Can be one of `SOFT` and `HARD`.
service.last_state_change	The last state change's timestamp.
service.downtime_depth	The number of active downtimes.
service.duration_sec	The time since the last state change.
service.latency	The service's check latency.
service.execution_time	The service's check execution time.
service.output	The last check's output.
service.perfdata	The last check's performance data.
service.last_check	The timestamp when the last check was executed.
service.check_source	The monitoring instance that performed the last check.

In addition to these specific runtime macros service object attributes can be accessed too.

3.5.4 Command Runtime Macros

The following custom attributes are available in all commands:

Name	Description
command.name	The name of the command object.

3.5.5 User Runtime Macros

The following custom attributes are available in all commands that are executed for users:

Name	Description
user.name	The name of the user object.
user.display_name	The value of the `display_name` attribute.

In addition to these specific runtime macros user object attributes can be accessed too.

3.5.6 Notification Runtime Macros

Name	Description
notification.type	The type of the notification.
notification.author	The author of the notification comment if existing.
notification.comment	The comment of the notification if existing.

In addition to these specific runtime macros notification object attributes can be accessed too.

3.5.7 Global Runtime Macros

The following macros are available in all executed commands:

Name	Description
icinga.timet	Current UNIX timestamp.

Name	Description
icinga.long_date_time	Current date and time including timezone information. Example: `2014-01-03 11:23:08 +0000`
icinga.short_date_time	Current date and time. Example: `2014-01-03 11:23:08`
icinga.date	Current date. Example: `2014-01-03`
icinga.time	Current time including timezone information. Example: `11:23:08 +0000`
icinga.uptime	Current uptime of the Icinga 2 process.

The following macros provide global statistics:

Name	Description
icinga.num_services_ok	Current number of services in state 'OK'.
icinga.num_services_warning	Current number of services in state 'Warning'.
icinga.num_services_critical	Current number of services in state 'Critical'.
icinga.num_services_unknown	Current number of services in state 'Unknown'.
icinga.num_services_pending	Current number of pending services.
icinga.num_services_unreachable	Current number of unreachable services.
icinga.num_services_flapping	Current number of flapping services.

Name	Description
icinga.num_services_in_downtime	Current number of services in downtime.
icinga.num_services_acknowledged	Current number of acknowledged service problems.
icinga.num_hosts_up	Current number of hosts in state 'Up'.
icinga.num_hosts_down	Current number of hosts in state 'Down'.
icinga.num_hosts_unreachable	Current number of unreachable hosts.
icinga.num_hosts_pending	Current number of pending hosts.
icinga.num_hosts_flapping	Current number of flapping hosts.
icinga.num_hosts_in_downtime	Current number of hosts in downtime.
icinga.num_hosts_acknowledged	Current number of acknowledged host problems.

3.6 Apply Rules

Several object types require an object relation, e.g. Service, Notification, Dependency, ScheduledDowntime objects. The object relations are documented in the linked chapters.

If you for example create a service object you have to specify the host_name attribute and reference an existing host attribute.

```
object Service "ping4" {
  check_command = "ping4"
  host_name = "icinga2-client1.localdomain"
}
```

This isn't comfortable when managing a huge set of configuration objects which could match on a common pattern.

Instead you want to use **apply rules**.

If you want basic monitoring for all your hosts, add a `ping4` service apply rule for all hosts which have the `address` attribute specified. Just one rule for 1000 hosts instead of 1000 service objects. Apply rules will automatically generate them for you.

```
apply Service "ping4" {
  check_command = "ping4"
  assign where host.address
}
```

More explanations on assign where expressions can be found here.

3.6.1 Apply Rules: Prerequisites

Before you start with apply rules keep the following in mind:

- Define the best match.
  - A set of unique custom attributes for these hosts/services?
  - Or group memberships, e.g. a host being a member of a hostgroup which should have a service set?
  - A generic pattern match on the host/service name?
  - Multiple expressions combined with `&&` or `||` operators
- All expressions must return a boolean value (an empty string is equal to `false` e.g.)

More specific object type requirements are described in these chapters:

- Apply services to hosts
- Apply notifications to hosts and services
- Apply dependencies to hosts and services
- Apply scheduled downtimes to hosts and services

3.6.2 Apply Rules: Usage Examples

You can set/override object attributes in apply rules using the respectively available objects in that scope (host and/or service objects).

```
vars.application_type = host.vars.application_type
```

Custom attributes can also store nested dictionaries and arrays. That way you can use them for not only matching for their existence or values in apply expressions, but also assign ("inherit") their values into the generated objected from apply rules.

Remember the examples shown for custom attribute values:

```
  vars.notification["mail"] = {
    groups = [ "icingaadmins" ]
  }
```

You can do two things here:

- Check for the existence of the `notification` custom attribute and its nested dictionary key `mail`. If this is boolean true, the notification object will be generated.
- Assign the value of the `groups` key to the `user_groups` attribute.

```
apply Notification "mail-icingaadmin" to Host {
  [...]
```

```
  user_groups = host.vars.notification.mail.groups

  assign where host.vars.notification.mail
}
```

A more advanced example is to use apply rules with for loops on arrays or dictionaries provided by custom atttributes or groups.

Remember the examples shown for custom attribute values:

```
  vars.disks["disk /"] = {
    disk_partitions = "/"
  }
```

You can iterate over all dictionary keys defined in `disks`. You can optionally use the value to specify additional object attributes.

```
apply Service for (disk => config in host.vars.disks) {
  [...]

  vars.disk_partitions = config.disk_partitions
}
```

Please read the apply for chapter for more specific insights.

> **Tip**
>
> Building configuration in that dynamic way requires detailed information of the generated objects. Use the `object list` CLI command after successful configuration validation.

3.6.3 Apply Rules Expressions

You can use simple or advanced combinations of apply rule expressions. Each expression must evaluate into the boolean `true` value. An empty string will be for instance interpreted as `false`. In a similar fashion undefined attributes will return `false`.

Returns `false`:

```
assign where host.vars.attribute_does_not_exist
```

Multiple `assign where` condition rows are evaluated as `OR` condition.

You can combine multiple expressions for matching only a subset of objects. In some cases, you want to be able to add more than one assign/ignore where expression which matches a specific condition. To achieve this you can use the logical `and` and `or` operators.

3.6.3.1 Apply Rules Expressions Examples

Assign a service to a specific host in a host group array using the in operator:

```
assign where "hostgroup-dev" in host.groups
```

Assign an object when a custom attribute is equal to a value:

```
assign where host.vars.application_type == "database"
```

```
assign where service.vars.sms_notify == true
```

Assign an object if a dictionary contains a given key:

```
assign where host.vars.app_dict.contains("app")
```

Match the host name by either using a case insensitive match:

```
assign where match("webserver*", host.name)
```

Match the host name by using a regular expression. Please note the escaped backslash character:

```
assign where regex("^webserver-[\\d+]", host.name)
```

Match all `*mysql*` patterns in the host name and (`&&`) custom attribute `prod_mysql_db` matches the `db-*` pattern. All hosts with the custom attribute `test_server` set to `true` should be ignored, or any host name ending with `*internal` pattern.

```
object HostGroup "mysql-server" {
  display_name = "MySQL Server"

 assign where match("*mysql*", host.name) && match("db-*", host.vars.prod_mysql_db)
  ignore where host.vars.test_server == true
  ignore where match("*internal", host.name)
}
```

Similar example for advanced notification apply rule filters: If the service attribute `notes` matches the `has gold support 24x7` string `AND` one of the two condition passes, either the `customer` host custom attribute is set to `customer-xy` `OR` the host custom attribute `always_notify` is set to `true`.

The notification is ignored for services whose host name ends with `*internal` `OR` the `priority` custom attribute is less than 2.

```
template Notification "cust-xy-notification" {
  users = [ "noc-xy", "mgmt-xy" ]
  command = "mail-service-notification"
}

apply Notification "notify-cust-xy-mysql" to Service {
  import "cust-xy-notification"
```

```
  assign where match("*has gold support 24x7*", service.notes) && (host.vars.customer == "custom
  ignore where match("*internal", host.name) || (service.vars.priority < 2 && host.vars.is_clus
}
```

More advanced examples are covered here.

3.6.4 Apply Services to Hosts

The sample configuration already includes a detailed example in hosts.conf and services.conf for this use case.

The example for `ssh` applies a service object to all hosts with the `address` attribute being defined and the custom attribute `os` set to the string `Linux` in `vars`.

```
apply Service "ssh" {
  import "generic-service"

  check_command = "ssh"

  assign where host.address && host.vars.os == "Linux"
}
```

Other detailed examples are used in their respective chapters, for example apply services with custom command arguments.

3.6.5 Apply Notifications to Hosts and Services

Notifications are applied to specific targets (`Host` or `Service`) and work in a similar manner:

```
apply Notification "mail-noc" to Service {
  import "mail-service-notification"

  user_groups = [ "noc" ]

  assign where host.vars.notification.mail
}
```

In this example the `mail-noc` notification will be created as object for all services having the `notification.mail` custom attribute defined. The notification command is set to `mail-service-notification` and all members of the user group `noc` will get notified.

It is also possible to generally apply a notification template and dynamically overwrite values from the template by checking for custom attributes. This can be achieved by using conditional statements:

```
apply Notification "host-mail-noc" to Host {
  import "mail-host-notification"

 // replace interval inherited from `mail-host-notification` template with new notfication inte
  if (host.vars.notification_interval) {
    interval = host.vars.notification_interval
  }

  // same with notification period
  if (host.vars.notification_period) {
    period = host.vars.notification_period
  }

 // Send SMS instead of email if the host's custom attribute `notification_type` is set to `sms`
  if (host.vars.notification_type == "sms") {
    command = "sms-host-notification"
  } else {
    command = "mail-host-notification"
  }

  user_groups = [ "noc" ]

  assign where host.address
}
```

In the example above the notification template `mail-host-notification` contains all relevant notification settings. The apply rule is applied on all host objects where the `host.address` is defined.

If the host object as a specific custom attributed set, its value is inherited into the local notification object scope, e.g. `host.vars.notification_interval`, `host.vars.notification_period` and `host.vars.notification_type`. This overwrites attributes already specified in the imported `mail-host-notification` template.

The corresponding host object could look like this:

```
object Host "host1" {
  import "host-linux-prod"
  display_name = "host1"
  address = "192.168.1.50"
  vars.notification_interval = 1h
  vars.notification_period = "24x7"
  vars.notification_type = "sms"
}
```

3.6.6 Apply Dependencies to Hosts and Services

Detailed examples can be found in the dependencies chapter.

3.6.7 Apply Recurring Downtimes to Hosts and Services

The sample configuration includes an example in downtimes.conf.

Detailed examples can be found in the recurring downtimes chapter.

3.6.8 Using Apply For Rules

Next to the standard way of using apply rules there is the requirement of applying objects based on a set (array or dictionary) using apply for expressions.

The sample configuration already includes a detailed example in hosts.conf and services.conf for this use case.

Take the following example: A host provides the snmp oids for different service check types. This could look like the following example:

```
object Host "router-v6" {
  check_command = "hostalive"
  address6 = "::1"

  vars.oids["if01"] = "1.1.1.1.1"
  vars.oids["temp"] = "1.1.1.1.2"
  vars.oids["bgp"] = "1.1.1.1.5"
}
```

The idea is to create service objects for `if01` and `temp` but not `bgp`. The oid value should also be used as service custom attribute `snmp_oid`. This is the command argument required by the snmp check command. The service's `display_name` should be set to the identifier inside the dictionary, e.g. `if01`.

```
apply Service for (identifier => oid in host.vars.oids) {
  check_command = "snmp"
  display_name = identifier
  vars.snmp_oid = oid

 ignore where identifier == "bgp" //don't generate service for bgp checks
}
```

Icinga 2 evaluates the `apply for` rule for all objects with the custom attribute `oids` set. It iterates over all dictionary items inside the `for` loop and evaluates the `assign/ignore where` expressions. You can access the loop variable in these expressions, e.g. to ignore specific values.

In this example the `bgp` identifier is ignored. This avoids to generate unwanted services. A different approach would be to match the `oid` value with a regex/wildcard match pattern for example.

```
ignore where regex("^\d.\d.\d.\d.5$", oid)
```

Note

You don't need an `assign where` expression which checks for the existence of the `oids` custom attribute.

This method saves you from creating multiple apply rules. It also moves the attribute specification logic from the service to the host.

3.6.8.1 Apply For and Custom Attribute Override

Imagine a different more advanced example: You are monitoring your network device (host) with many interfaces (services). The following requirements/problems apply:

- Each interface service should be named with a prefix and a name defined in your host object (which could be generated from your CMDB, etc.)
- Each interface has its own VLAN tag
- Some interfaces have QoS enabled
- Additional attributes such as `display_name` or `notes`, `notes_url` and `action_url` must be dynamically generated.

Tip

Define the SNMP community as global constant in your constants.conf file.

```
const IftrafficSnmpCommunity = "public"
```

Define the `interfaces` custom attribute on the `cisco-catalyst-6509-34` host object and add three example interfaces as dictionary keys.

Specify additional attributes inside the nested dictionary as learned with custom attribute values:

```
object Host "cisco-catalyst-6509-34" {
  import "generic-host"
  display_name = "Catalyst 6509 #34 VIE21"
  address = "127.0.1.4"

  /* "GigabitEthernet0/2" is the interface name,
   * and key name in service apply for later on
   */
  vars.interfaces["GigabitEthernet0/2"] = {
     /* define all custom attributes with the
      * same name required for command parameters/arguments
```

```
     * in service apply (look into your CheckCommand definition)
     */
    iftraffic_units = "g"
    iftraffic_community = IftrafficSnmpCommunity
    iftraffic_bandwidth = 1
    vlan = "internal"
    qos = "disabled"
  }
  vars.interfaces["GigabitEthernet0/4"] = {
    iftraffic_units = "g"
    //iftraffic_community = IftrafficSnmpCommunity
    iftraffic_bandwidth = 1
    vlan = "remote"
    qos = "enabled"
  }
  vars.interfaces["MgmtInterface1"] = {
    iftraffic_community = IftrafficSnmpCommunity
    vlan = "mgmt"
    interface_address = "127.99.0.100" #special management ip
  }
}
```

Start with the apply for definition and iterate over `host.vars.interfaces`. This is a dictionary and should use the variables `interface_name` as key and `interface_config` as value for each generated object scope.

`"if-"` specifies the object name prefix for each service which results in `if-<interface_name>` for each iteration.

```
/* loop over the host.vars.interfaces dictionary
 * for (key => value in dict) means `interface_name` as key
 * and `interface_config` as value. Access config attributes
 * with the indexer (`.`) character.
 */
apply Service "if-" for (interface_name => interface_config in host.vars.interfaces) {
```

Import the `generic-service` template, assign the iftraffic `check_command`. Use the dictionary key `interface_name` to set a proper `display_name` string for external interfaces.

```
  import "generic-service"
  check_command = "iftraffic"
  display_name = "IF-" + interface_name
```

The `interface_name` key's value is the same string used as command parameter for `iftraffic`:

```
 /* use the key as command argument (no duplication of values in host.vars.interfaces) */
  vars.iftraffic_interface = interface_name
```

Remember that `interface_config` is a nested dictionary. In the first iteration it looks like this:

```
interface_config = {
  iftraffic_units = "g"
  iftraffic_community = IftrafficSnmpCommunity
  iftraffic_bandwidth = 1
  vlan = "internal"
  qos = "disabled"
}
```

Access the dictionary keys with the indexer syntax and assign them to custom attributes used as command parameters for the `iftraffic` check command.

```
  /* map the custom attributes as command arguments */
  vars.iftraffic_units = interface_config.iftraffic_units
  vars.iftraffic_community = interface_config.iftraffic_community
```

If you just want to inherit all attributes specified inside the `interface_config` dictionary, add it to the generated service custom attributes like this:

```
 /* the above can be achieved in a shorter fashion if the names inside host.vars.interfaces
  * are the _exact_ same as required as command parameter by the check command
   * definition.
   */
  vars += interface_config
```

If the user did not specify default values for required service custom attributes, add them here. This also helps to avoid unwanted configuration validation errors or runtime failures. Please read more about conditional statements here.

```
  /* set a default value for units and bandwidth */
  if (interface_config.iftraffic_units == "") {
    vars.iftraffic_units = "m"
  }
  if (interface_config.iftraffic_bandwidth == "") {
    vars.iftraffic_bandwidth = 1
  }
  if (interface_config.vlan == "") {
    vars.vlan = "not set"
  }
  if (interface_config.qos == "") {
    vars.qos = "not set"
  }
```

If the host object did not specify a custom SNMP community, set a default value specified by the global constant `IftrafficSnmpCommunity`.

```
  /* set the global constant if not explicitely
   * not provided by the `interfaces` dictionary on the host
```

```
   */
  if (len(interface_config.iftraffic_community) == 0 || len(vars.iftraffic_community) == 0) {
    vars.iftraffic_community = IftrafficSnmpCommunity
  }
```

Use the provided values to calculate more object attributes which can be e.g. seen in external interfaces.

```
 /* Calculate some additional object attributes after populating the `vars` dictionary */
 notes = "Interface check for " + interface_name + " (units: '" + interface_config.iftraffic_uni
  notes_url = "https://foreman.company.com/hosts/" + host.name
 action_url = "http://snmp.checker.company.com/" + host.name + "/if-" + interface_name
}
```

Tip

Building configuration in that dynamic way requires detailed information of the generated objects. Use the `object list` CLI command after successful configuration validation.

Verify that the apply-for-rule successfully created the service objects with the inherited custom attributes:

```
# icinga2 daemon -C
# icinga2 object list --type Service --name *catalyst*

Object 'cisco-catalyst-6509-34!if-GigabitEthernet0/2' of type 'Service':
......
  * vars
   % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 59:3-59:26
    * iftraffic_bandwidth = 1
    * iftraffic_community = "public"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 53:3-53:65
    * iftraffic_interface = "GigabitEthernet0/2"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 49:3-49:43
    * iftraffic_units = "g"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 52:3-52:57
    * qos = "disabled"
    * vlan = "internal"


Object 'cisco-catalyst-6509-34!if-GigabitEthernet0/4' of type 'Service':
...
  * vars
   % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 59:3-59:26
    * iftraffic_bandwidth = 1
    * iftraffic_community = "public"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 53:3-53:65
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 79:5-79:53
```

```
    * iftraffic_interface = "GigabitEthernet0/4"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 49:3-49:43
    * iftraffic_units = "g"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 52:3-52:57
    * qos = "enabled"
    * vlan = "remote"

Object 'cisco-catalyst-6509-34!if-MgmtInterface1' of type 'Service':
...
  * vars
   % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 59:3-59:26
    * iftraffic_bandwidth = 1
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 66:5-66:32
    * iftraffic_community = "public"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 53:3-53:65
    * iftraffic_interface = "MgmtInterface1"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 49:3-49:43
    * iftraffic_units = "m"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 52:3-52:57
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 63:5-63:30
    * interface_address = "127.99.0.100"
    * qos = "not set"
    % = modified in '/etc/icinga2/conf.d/iftraffic.conf', lines 72:5-72:24
    * vlan = "mgmt"
```

3.6.9 Use Object Attributes in Apply Rules

Since apply rules are evaluated after the generic objects, you can reference existing host and/or service object attributes as values for any object attribute specified in that apply rule.

```
object Host "opennebula-host" {
  import "generic-host"
  address = "10.1.1.2"

  vars.hosting["cust1"] = {
    http_uri = "/shop"
    customer_name = "Customer 1"
    customer_id = "7568"
    support_contract = "gold"
  }
  vars.hosting["cust2"] = {
    http_uri = "/"
    customer_name = "Customer 2"
    customer_id = "7569"
```

```
    support_contract = "silver"
  }
}
```

`hosting` is a custom attribute with the Dictionary value type. This is mandatory to iterate with the `key => value` notation in the below apply for rule.

```
apply Service for (customer => config in host.vars.hosting) {
  import "generic-service"
  check_command = "ping4"

  vars.qos = "disabled"

  vars += config

  vars.http_uri = "/" + customer + "/" + config.http_uri

 display_name = "Shop Check for " + vars.customer_name + "-" + vars.customer_id

 notes = "Support contract: " + vars.support_contract + " for Customer " + vars.customer_name +

  notes_url = "https://foreman.company.com/hosts/" + host.name
 action_url = "http://snmp.checker.company.com/" + host.name + "/" + vars.customer_id
}
```

Each loop iteration has different values for `customer` and config‘ in the local scope.

1.

```
customer = "cust 1"
config = {
  http_uri = "/shop"
  customer_name = "Customer 1"
  customer_id = "7568"
  support_contract = "gold"
}
```

2.

```
customer = "cust2"
config = {
  http_uri = "/"
  customer_name = "Customer 2"
  customer_id = "7569"
  support_contract = "silver"
}
```

You can now add the `config` dictionary into `vars`.

```
vars += config
```

Now it looks like the following in the first iteration:

```
customer = "cust 1"
vars = {
  http_uri = "/shop"
  customer_name = "Customer 1"
  customer_id = "7568"
  support_contract = "gold"
}
```

Remember, you know this structure already. Custom attributes can also be accessed by using the indexer syntax.

```
  vars.http_uri = ... + config.http_uri
```

can also be written as

```
  vars += config
  vars.http_uri = ... + vars.http_uri
```

3.7 Groups

A group is a collection of similar objects. Groups are primarily used as a visualization aid in web interfaces.

Group membership is defined at the respective object itself. If you have a hostgroup name `windows` for example, and want to assign specific hosts to this group for later viewing the group on your alert dashboard, first create a HostGroup object:

```
object HostGroup "windows" {
  display_name = "Windows Servers"
}
```

Then add your hosts to this group:

```
template Host "windows-server" {
  groups += [ "windows" ]
}

object Host "mssql-srv1" {
  import "windows-server"

  vars.mssql_port = 1433
}

object Host "mssql-srv2" {
  import "windows-server"
```

```
  vars.mssql_port = 1433
}
```

This can be done for service and user groups the same way:

```
object UserGroup "windows-mssql-admins" {
  display_name = "Windows MSSQL Admins"
}

template User "generic-windows-mssql-users" {
  groups += [ "windows-mssql-admins" ]
}

object User "win-mssql-noc" {
  import "generic-windows-mssql-users"

  email = "noc@example.com"
}

object User "win-mssql-ops" {
  import "generic-windows-mssql-users"

  email = "ops@example.com"
}
```

3.7.1 Group Membership Assign

Instead of manually assigning each object to a group you can also assign objects to a group based on their attributes:

```
object HostGroup "prod-mssql" {
  display_name = "Production MSSQL Servers"

  assign where host.vars.mssql_port && host.vars.prod_mysql_db
  ignore where host.vars.test_server == true
  ignore where match("*internal", host.name)
}
```

In this example all hosts with the `vars` attribute `mssql_port` will be added as members to the host group `mssql`. However, all hosts matching the string `\*internal` or with the `test_server` attribute set to `true` are **not** added to this group.

Details on the `assign where` syntax can be found in the Language Reference.

3.8 Notifications

Notifications for service and host problems are an integral part of your monitoring setup.

When a host or service is in a downtime, a problem has been acknowledged or the dependency logic determined that the host/service is unreachable, no notifications are sent. You can configure additional type and state filters refining the notifications being actually sent.

There are many ways of sending notifications, e.g. by email, XMPP, IRC, Twitter, etc. On its own Icinga 2 does not know how to send notifications. Instead it relies on external mechanisms such as shell scripts to notify users. More notification methods are listed in the addons and plugins chapter.

A notification specification requires one or more users (and/or user groups) who will be notified in case of problems. These users must have all custom attributes defined which will be used in the `NotificationCommand` on execution.

The user `icingaadmin` in the example below will get notified only on `Warning` and `Critical` problems. In addition to that `Recovery` notifications are sent (they require the `OK` state).

```
object User "icingaadmin" {
  display_name = "Icinga 2 Admin"
  enable_notifications = true
  states = [ OK, Warning, Critical ]
  types = [ Problem, Recovery ]
  email = "icinga@localhost"
}
```

If you don't set the `states` and `types` configuration attributes for the `User` object, notifications for all states and types will be sent.

Details on troubleshooting notification problems can be found here.

> **Note**
>
> Make sure that the notification feature is enabled in order to execute notification commands.

You should choose which information you (and your notified users) are interested in case of emergency, and also which information does not provide any value to you and your environment.

An example notification command is explained here.

You can add all shared attributes to a `Notification` template which is inherited to the defined notifications. That way you'll save duplicated attributes in each `Notification` object. Attributes can be overridden locally.

```
template Notification "generic-notification" {
```

```
  interval = 15m

  command = "mail-service-notification"

  states = [ Warning, Critical, Unknown ]
 types = [ Problem, Acknowledgement, Recovery, Custom, FlappingStart,
        FlappingEnd, DowntimeStart, DowntimeEnd, DowntimeRemoved ]

  period = "24x7"
}
```

The time period `24x7` is included as example configuration with Icinga 2.

Use the `apply` keyword to create `Notification` objects for your services:

```
apply Notification "notify-cust-xy-mysql" to Service {
  import "generic-notification"

  users = [ "noc-xy", "mgmt-xy" ]

 assign where match("*has gold support 24x7*", service.notes) && (host.vars.customer == "custom
 ignore where match("*internal", host.name) || (service.vars.priority < 2 && host.vars.is_clus
}
```

Instead of assigning users to notifications, you can also add the `user_groups` attribute with a list of user groups to the `Notification` object. Icinga 2 will send notifications to all group members.

Note

Only users who have been notified of a problem before (`Warning`, `Critical`, `Unknown` states for services, `Down` for hosts) will receive `Recovery` notifications.

3.8.1 Notifications: Users from Host/Service

A common pattern is to store the users and user groups on the host or service objects instead of the notification object itself.

The sample configuration provided in hosts.conf and notifications.conf already provides an example for this question.

Tip

Please make sure to read the apply and custom attribute values chapter to fully understand these examples.

Specify the user and groups as nested custom attribute on the host object:

```
object Host "icinga2-client1.localdomain" {
```

```
  [...]

  vars.notification["mail"] = {
    groups = [ "icingaadmins" ]
    users = [ "icingaadmin" ]
  }
  vars.notification["sms"] = {
    users = [ "icingaadmin" ]
  }
}
```

As you can see, there is the option to use two different notification apply rules here: One for `mail` and one for `sms`.

This example assigns the `users` and `groups` nested keys from the `notification` custom attribute to the actual notification object attributes.

Since errors are hard to debug if host objects don't specify the required configuration attributes, you can add a safety condition which logs which host object is affected.

```
critical/config: Host 'icinga2-client3.localdomain' does not specify required user/user_groups
```

You can also use the script debugger for more advanced insights.

```
apply Notification "mail-host-notification" to Host {
  [...]

 /* Log which host does not specify required user/user_groups attributes. This will fail immedia
 if (len(host.vars.notification.mail.users) == 0 && len(host.vars.notification.mail.user_grou
  log(LogCritical, "config", "Host '" + host.name + "' does not specify required user/user_grou
 }

 users = host.vars.notification.mail.users
 user_groups = host.vars.notification.mail.groups

 assign where host.vars.notification.mail && typeof(host.vars.notification.mail) == Dictionar
}

apply Notification "sms-host-notification" to Host {
  [...]

 /* Log which host does not specify required user/user_groups attributes. This will fail immedia
 if (len(host.vars.notification.sms.users) == 0 && len(host.vars.notification.sms.user_groups
  log(LogCritical, "config", "Host '" + host.name + "' does not specify required user/user_grou
 }

 users = host.vars.notification.sms.users
 user_groups = host.vars.notification.sms.groups
```

```
  assign where host.vars.notification.sms && typeof(host.vars.notification.sms) == Dictionary
}
```

The example above uses typeof as safety function to ensure that the `mail` key really provides a dictionary as value. Otherwise the configuration validation could fail if an admin adds something like this on another host:

```
  vars.notification.mail = "yes"
```

You can also do a more fine granular assignment on the service object:

```
apply Service "http" {
  [...]

  vars.notification["mail"] = {
    groups = [ "icingaadmins" ]
    users = [ "icingaadmin" ]
  }

  [...]
}
```

This notification apply rule is different to the one above. The service notification users and groups are inherited from the service and if not set, from the host object. A default user is set too.

```
apply Notification "mail-host-notification" to Service {
  [...]

  if (service.vars.notification.mail.users) {
    users = service.vars.notification.mail.users
  } else if (host.vars.notification.mail.users) {
    users = host.vars.notification.mail.users
  } else {
    /* Default user who receives everything. */
    users = [ "icingaadmin" ]
  }

  if (service.vars.notification.mail.groups) {
    user_groups = service.vars.notification.mail.groups
  } else (host.vars.notification.mail.groups) {
    user_groups = host.vars.notification.mail.groups
  }

  assign where host.vars.notification.mail && typeof(host.vars.notification.mail) == Dictionar
}
```

3.8.2 Notification Escalations

When a problem notification is sent and a problem still exists at the time of re-notification you may want to escalate the problem to the next support level. A different approach is to configure the default notification by email, and escalate the problem via SMS if not already solved.

You can define notification start and end times as additional configuration attributes making the `Notification` object a so-called `notification escalation`. Using templates you can share the basic notification attributes such as users or the `interval` (and override them for the escalation then).

Using the example from above, you can define additional users being escalated for SMS notifications between start and end time.

```
object User "icinga-oncall-2nd-level" {
  display_name = "Icinga 2nd Level"

  vars.mobile = "+1 555 424642"
}

object User "icinga-oncall-1st-level" {
  display_name = "Icinga 1st Level"

  vars.mobile = "+1 555 424642"
}
```

Define an additional NotificationCommand for SMS notifications.

> **Note**
>
> The example is not complete as there are many different SMS providers. Please note that sending SMS notifications will require an SMS provider or local hardware with an active SIM card.

```
object NotificationCommand "sms-notification" {
   command = [
     PluginDir + "/send_sms_notification",
     "$mobile$",
     "..."
}
```

The two new notification escalations are added onto the local host and its service `ping4` using the `generic-notification` template. The user `icinga-oncall-2nd-level` will get notified by SMS (`sms-notification` command) after `30m` until `1h`.

> **Note**
>
> The `interval` was set to 15m in the `generic-notification` tem-

plate example. Lower that value in your escalations by using a secondary template or by overriding the attribute directly in the `notifications` array position for `escalation-sms-2nd-level`.

If the problem does not get resolved nor acknowledged preventing further notifications, the `escalation-sms-1st-level` user will be escalated `1h` after the initial problem was notified, but only for one hour (`2h` as `end` key for the `times` dictionary).

```
apply Notification "mail" to Service {
  import "generic-notification"

  command = "mail-notification"
  users = [ "icingaadmin" ]

  assign where service.name == "ping4"
}

apply Notification "escalation-sms-2nd-level" to Service {
  import "generic-notification"

  command = "sms-notification"
  users = [ "icinga-oncall-2nd-level" ]

  times = {
    begin = 30m
    end = 1h
  }

  assign where service.name == "ping4"
}

apply Notification "escalation-sms-1st-level" to Service {
  import "generic-notification"

  command = "sms-notification"
  users = [ "icinga-oncall-1st-level" ]

  times = {
    begin = 1h
    end = 2h
  }

  assign where service.name == "ping4"
}
```

3.8.3 Notification Delay

Sometimes the problem in question should not be announced when the notification is due (the object reaching the `HARD` state), but after a certain period. In Icinga 2 you can use the `times` dictionary and set `begin = 15m` as key and value if you want to postpone the notification window for 15 minutes. Leave out the `end` key – if not set, Icinga 2 will not check against any end time for this notification. Make sure to specify a relatively low notification `interval` to get notified soon enough again.

```
apply Notification "mail" to Service {
  import "generic-notification"

  command = "mail-notification"
  users = [ "icingaadmin" ]

  interval = 5m

  times.begin = 15m // delay notification window

  assign where service.name == "ping4"
}
```

3.8.4 Disable Re-notifications

If you prefer to be notified only once, you can disable re-notifications by setting the `interval` attribute to `0`.

```
apply Notification "notify-once" to Service {
  import "generic-notification"

  command = "mail-notification"
  users = [ "icingaadmin" ]

  interval = 0 // disable re-notification

  assign where service.name == "ping4"
}
```

3.8.5 Notification Filters by State and Type

If there are no notification state and type filter attributes defined at the `Notification` or `User` object, Icinga 2 assumes that all states and types are being notified.

Available state and type filters for notifications are:

```
template Notification "generic-notification" {

  states = [ OK, Warning, Critical, Unknown ]
 types = [ Problem, Acknowledgement, Recovery, Custom, FlappingStart,
        FlappingEnd, DowntimeStart, DowntimeEnd, DowntimeRemoved ]
}
```

3.9 Commands

Icinga 2 uses three different command object types to specify how checks should be performed, notifications should be sent, and events should be handled.

3.9.1 Check Commands

CheckCommand objects define the command line how a check is called.

CheckCommand objects are referenced by Host and Service objects using the `check_command` attribute.

> **Note**
>
> Make sure that the checker feature is enabled in order to execute checks.

3.9.1.1 Integrate the Plugin with a CheckCommand Definition

Unless you have done so already, download your check plugin and put it into the PluginDir directory. The following example uses the `check_mysql` plugin contained in the Monitoring Plugins package.

The plugin path and all command arguments are made a list of double-quoted string arguments for proper shell escaping.

Call the `check_disk` plugin with the `--help` parameter to see all available options. Our example defines warning (`-w`) and critical (`-c`) thresholds for the disk usage. Without any partition defined (`-p`) it will check all local partitions.

```
icinga@icinga2 $ /usr/lib64/nagios/plugins/check_mysql --help
...
This program tests connections to a MySQL server

Usage:
check_mysql [-d database] [-H host] [-P port] [-s socket]
[-u user] [-p password] [-S] [-l] [-a cert] [-k key]
[-C ca-cert] [-D ca-dir] [-L ciphers] [-f optfile] [-g group]
```

Next step is to understand how command parameters are being passed from a host or service object, and add a CheckCommand definition based on these required parameters and/or default values.

Please continue reading in the plugins section for additional integration examples.

3.9.1.2 Passing Check Command Parameters from Host or Service

Check command parameters are defined as custom attributes which can be accessed as runtime macros by the executed check command.

The check command parameters for ITL provided plugin check command definitions are documented here, for example disk.

In order to practice passing command parameters you should integrate your own plugin.

The following example will use `check_mysql` provided by the Monitoring Plugins installation.

Define the default check command custom attributes, for example `mysql_user` and `mysql_password` (freely definable naming schema) and optional their default threshold values. You can then use these custom attributes as runtime macros for command arguments on the command line.

> **Tip**
>
> Use a common command type as prefix for your command arguments to increase readability. `mysql_user` helps understanding the context better than just `user` as argument.

The default custom attributes can be overridden by the custom attributes defined in the host or service using the check command `my-mysql`. The custom attributes can also be inherited from a parent template using additive inheritance (`+=`).

```
# vim /etc/icinga2/conf.d/commands.conf

object CheckCommand "my-mysql" {
 command = [ PluginDir + "/check_mysql" ] //constants.conf -> const PluginDir

  arguments = {
    "-H" = "$mysql_host$"
    "-u" = {
      required = true
      value = "$mysql_user$"
    }
    "-p" = "$mysql_password$"
```

```
    "-P" = "$mysql_port$"
    "-s" = "$mysql_socket$"
    "-a" = "$mysql_cert$"
    "-d" = "$mysql_database$"
    "-k" = "$mysql_key$"
    "-C" = "$mysql_ca_cert$"
    "-D" = "$mysql_ca_dir$"
    "-L" = "$mysql_ciphers$"
    "-f" = "$mysql_optfile$"
    "-g" = "$mysql_group$"
    "-S" = {
      set_if = "$mysql_check_slave$"
     description = "Check if the slave thread is running properly."
    }
    "-l" = {
      set_if = "$mysql_ssl$"
      description = "Use ssl encryption"
    }
  }

  vars.mysql_check_slave = false
  vars.mysql_ssl = false
  vars.mysql_host = "$address$"
}
```

The check command definition also sets `mysql_host` to the `$address$` default value. You can override this command parameter if for example your MySQL host is not running on the same server's ip address.

Make sure pass all required command parameters, such as `mysql_user`, `mysql_password` and `mysql_database`. `MysqlUsername` and `MysqlPassword` are specified as global constants in this example.

```
# vim /etc/icinga2/conf.d/services.conf

apply Service "mysql-icinga-db-health" {
  import "generic-service"

  check_command = "my-mysql"

  vars.mysql_user = MysqlUsername
  vars.mysql_password = MysqlPassword

  vars.mysql_database = "icinga"
  vars.mysql_host = "192.168.33.11"

  assign where match("icinga2*", host.name)
```

```
  ignore where host.vars.no_health_check == true
}
```

Take a different example: The example host configuration in hosts.conf also applies an `ssh` service check. Your host's ssh port is not the default `22`, but set to `2022`. You can pass the command parameter as custom attribute `ssh_port` directly inside the service apply rule inside services.conf:

```
apply Service "ssh" {
  import "generic-service"

  check_command = "ssh"
  vars.ssh_port = 2022 //custom command parameter

 assign where (host.address || host.address6) && host.vars.os == "Linux"
}
```

If you prefer this being configured at the host instead of the service, modify the host configuration object instead. The runtime macro resolving order is described here.

```
object Host "icinga2-client1.localdomain {
...
  vars.ssh_port = 2022
}
```

3.9.1.3 Passing Check Command Parameters Using Apply For

The host `localhost` with the generated services from the `basic-partitions` dictionary (see apply for for details) checks a basic set of disk partitions with modified custom attributes (warning thresholds at `10%`, critical thresholds at `5%` free disk space).

The custom attribute `disk_partition` can either hold a single string or an array of string values for passing multiple partitions to the `check_disk` check plugin.

```
object Host "my-server" {
  import "generic-host"
  address = "127.0.0.1"
  address6 = "::1"

  vars.local_disks["basic-partitions"] = {
    disk_partitions = [ "/", "/tmp", "/var", "/home" ]
  }
}

apply Service for (disk => config in host.vars.local_disks) {
  import "generic-service"
```

```
  check_command = "my-disk"

  vars += config

  vars.disk_wfree = "10%"
  vars.disk_cfree = "5%"
}
```

More details on using arrays in custom attributes can be found in this chapter.

3.9.1.4 Command Arguments

By defining a check command line using the `command` attribute Icinga 2 will resolve all macros in the static string or array. Sometimes it is required to extend the arguments list based on a met condition evaluated at command execution. Or making arguments optional – only set if the macro value can be resolved by Icinga 2.

```
object CheckCommand "http" {
  command = [ PluginDir + "/check_http" ]

  arguments = {
    "-H" = "$http_vhost$"
    "-I" = "$http_address$"
    "-u" = "$http_uri$"
    "-p" = "$http_port$"
    "-S" = {
      set_if = "$http_ssl$"
    }
    "--sni" = {
      set_if = "$http_sni$"
    }
    "-a" = {
      value = "$http_auth_pair$"
    description = "Username:password on sites with basic authentication"
    }
    "--no-body" = {
      set_if = "$http_ignore_body$"
    }
    "-r" = "$http_expect_body_regex$"
    "-w" = "$http_warn_time$"
    "-c" = "$http_critical_time$"
    "-e" = "$http_expect$"
  }

  vars.http_address = "$address$"
```

```
  vars.http_ssl = false
  vars.http_sni = false
}
```

The example shows the `check_http` check command defining the most common arguments. Each of them is optional by default and is omitted if the value is not set. For example, if the service calling the check command does not have `vars.http_port` set, it won't get added to the command line.

If the `vars.http_ssl` custom attribute is set in the service, host or command object definition, Icinga 2 will add the `-S` argument based on the `set_if` numeric value to the command line. String values are not supported.

If the macro value cannot be resolved, Icinga 2 will not add the defined argument to the final command argument array. Empty strings for macro values won't omit the argument.

That way you can use the `check_http` command definition for both, with and without SSL enabled checks saving you duplicated command definitions.

Details on all available options can be found in the CheckCommand object definition.

3.9.1.4.1 Command Arguments: set_if

The `set_if` attribute in command arguments can be used to only add this parameter if the runtime macro value is boolean `true`.

Best practice is to define and pass only boolean values here. Numeric values are allowed too.

Examples:

```
vars.test_b = true
vars.test_n = 3.0

arguments = {
  "-x" = {
    set_if = "$test_b$"
  }
  "-y" = {
    set_if = "$test_n$"
  }
}
```

If you accidentally used a String value, this could lead into an undefined behaviour.

If you still want to work with String values and other variants, you can also use runtime evaluated functions for `set_if`.

```
vars.test_s = "1.1.2.1"
arguments = {
  "-z" = {
    set_if = {{
      var str = macro("$test_s$")

      return regex("^\d.\d.\d.\d$", str)
    }}
  }
```

References: abbreviated lambda syntax, macro, regex.

3.9.1.5 Environment Variables

The `env` command object attribute specifies a list of environment variables with values calculated from either runtime macros or custom attributes which should be exported as environment variables prior to executing the command.

This is useful for example for hiding sensitive information on the command line output when passing credentials to database checks:

```
object CheckCommand "mysql-health" {
  command = [
    PluginDir + "/check_mysql"
  ]

  arguments = {
    "-H" = "$mysql_address$"
    "-d" = "$mysql_database$"
  }

  vars.mysql_address = "$address$"
  vars.mysql_database = "icinga"
  vars.mysql_user = "icinga_check"
  vars.mysql_pass = "password"

  env.MYSQLUSER = "$mysql_user$"
  env.MYSQLPASS = "$mysql_pass$"
}
```

3.9.2 Notification Commands

NotificationCommand objects define how notifications are delivered to external interfaces (email, XMPP, IRC, Twitter, etc.). NotificationCommand objects are referenced by Notification objects using the `command` attribute.

Note

Make sure that the notification feature is enabled in order to execute notification commands.

While it's possible to specify an entire notification command right in the NotificationCommand object it is generally advisable to create a shell script in the `/etc/icinga2/scripts` directory and have the NotificationCommand object refer to that.

A fresh Icinga 2 install comes with with two example scripts for host and service notifications by email. Based on the Icinga 2 runtime macros (such as `$service.output$` for the current check output) it's possible to send email to the user(s) associated with the notification itself (`$user.email$`). Feel free to take these scripts as a starting point for your own individual notification solution - and keep in mind that nearly everything is technically possible.

Information needed to generate notifications is passed to the scripts as arguments. The NotificationCommand objects `mail-host-notification` and `mail-service-notification` correspond to the shell scripts `mail-host-notification.sh` and `mail-service-notification.sh` in `/etc/icinga2/scripts` and define default values for arguments. These defaults can always be overwritten locally.

Note

This example requires the `mail` binary installed on the Icinga 2 master.

3.9.2.1 Notification Commands in 2.7

Icinga 2 v2.7.0 introduced new notification scripts which support both environment variables and command line parameters.

Therefore the `NotificationCommand` objects inside the commands.conf and `Notification` apply rules inside the notifications.conf configuration files have been updated. Your configuration needs to be updated next to the notification scripts themselves.

Note

Several parameters have been changed. Please review the notification script parameters and configuration objects before updating your production environment.

The safest way is to incorporate the configuration updates from v2.7.0 inside the commands.conf and notifications.conf configuration files.

A quick-fix is shown below:

```
@@ -5,7 +5,8 @@ object NotificationCommand "mail-host-notification" {

   env = {
     NOTIFICATIONTYPE = "$notification.type$"
```

```
-    HOSTALIAS = "$host.display_name$"
+    HOSTNAME = "$host.name$"
+    HOSTDISPLAYNAME = "$host.display_name$"
     HOSTADDRESS = "$address$"
     HOSTSTATE = "$host.state$"
     LONGDATETIME = "$icinga.long_date_time$"
@@ -22,8 +23,9 @@ object NotificationCommand "mail-service-notification" {

   env = {
     NOTIFICATIONTYPE = "$notification.type$"
-    SERVICEDESC = "$service.name$"
-    HOSTALIAS = "$host.display_name$"
+    SERVICENAME = "$service.name$"
+    HOSTNAME = "$host.name$"
+    HOSTDISPLAYNAME = "$host.display_name$"
     HOSTADDRESS = "$address$"
     SERVICESTATE = "$service.state$"
     LONGDATETIME = "$icinga.long_date_time$"
```

3.9.2.2 mail-host-notification

The `mail-host-notification` NotificationCommand object uses the example notification script located in `/etc/icinga2/scripts/mail-host-notification.sh`.

Here is a quick overview of the arguments that can be used. See also host runtime macros for further information.

Name	Description
`notification_date`	**Required.** Date and time. Defaults to `$icinga.long_date_time$`.
`notification_hostname`	**Required.** The host's `FQDN`. Defaults to `$host.name$`.
`notification_hostdisplayname`	**Required.** The host's display name. Defaults to `$host.display_name$`.
`notification_hostoutput`	**Required.** Output from host check. Defaults to `$host.output$`.
`notification_useremail`	**Required.** The notification's recipient(s). Defaults to `$user.email$`.
`notification_hoststate`	**Required.** Current state of host. Defaults to `$host.state$`.
`notification_type`	**Required.** Type of notification. Defaults to `$notification.type$`.
`notification_address`	**Optional.** The host's IPv4 address. Defaults to `$address$`.
`notification_address6`	**Optional.** The host's IPv6 address. Defaults to `$address6$`.

Name	Description
notification_author	**Optional.** Comment author. Defaults to $notification.author$.
notification_comment	**Optional.** Comment text. Defaults to $notification.comment$.
notification_from	**Optional.** Define a valid From: string (e.g. "Icinga 2 Host Monitoring <icinga@example.com>"). Requires GNU mailutils (Debian/Ubuntu) or mailx (RHEL/SUSE).
notification_icingaweb2url	**Optional.** Define URL to your Icinga Web 2 (e.g. "https://www.example.com/icingaweb2")
notification_logtosyslog	**Optional.** Set true to log notification events to syslog; useful for debugging. Defaults to false.

3.9.2.3 mail-service-notification

The mail-service-notification NotificationCommand object uses the example notification script located in /etc/icinga2/scripts/mail-service-notification.sh.

Here is a quick overview of the arguments that can be used. See also service runtime macros for further information.

Name	Description
notification_date	**Required.** Date and time. Defaults to $icinga.long_date_time$.
notification_hostname	**Required.** The host's FQDN. Defaults to $host.name$.
notification_servicename	**Required.** The service name. Defaults to $service.name$.
notification_hostdisplayname	**Required.** Host display name. Defaults to $host.display_name$.
notification_servicedisplayname	**Required.** Service display name. Defaults to $service.display_name$.
notification_serviceoutput	**Required.** Output from service check. Defaults to $service.output$.
notification_useremail	**Required.** The notification's recipient(s). Defaults to $user.email$.
notification_servicestate	**Required.** Current state of host. Defaults to $service.state$.
notification_type	**Required.** Type of notification. Defaults to $notification.type$.

Name	Description
notification_address	**Optional.** The host's IPv4 address. Defaults to $address$.
notification_address6	**Optional.** The host's IPv6 address. Defaults to $address6$.
notification_author	**Optional.** Comment author. Defaults to $notification.author$.
notification_comment	**Optional.** Comment text. Defaults to $notification.comment$.
notification_from	**Optional.** Define a valid From: string (e.g. "Icinga 2 Host Monitoring <icinga@example.com>"). Requires GNU mailutils (Debian/Ubuntu) or mailx (RHEL/SUSE).
notification_icingaweb2url	**Optional.** Define URL to your Icinga Web 2 (e.g. "https://www.example.com/icingaweb2")
notification_logtosyslog	**Optional.** Set true to log notification events to syslog; useful for debugging. Defaults to false.

3.10 Dependencies

Icinga 2 uses host and service Dependency objects for determining their network reachability.

A service can depend on a host, and vice versa. A service has an implicit dependency (parent) to its host. A host to host dependency acts implicitly as host parent relation. When dependencies are calculated, not only the immediate parent is taken into account but all parents are inherited.

The parent_host_name and parent_service_name attributes are mandatory for service dependencies, parent_host_name is required for host dependencies. Apply rules will allow you to determine these attributes in a more dynamic fashion if required.

```
parent_host_name = "core-router"
parent_service_name = "uplink-port"
```

Notifications are suppressed by default if a host or service becomes unreachable. You can control that option by defining the disable_notifications attribute.

```
disable_notifications = false
```

If the dependency should be triggered in the parent object's soft state, you need to set ignore_soft_states to false.

The dependency state filter must be defined based on the parent object being either a host (`Up`, `Down`) or a service (`OK`, `Warning`, `Critical`, `Unknown`).

The following example will make the dependency fail and trigger it if the parent object is **not** in one of these states:

```
states = [ OK, Critical, Unknown ]
```

> **In other words**
>
> If the parent service object changes into the `Warning` state, this dependency will fail and render all child objects (hosts or services) unreachable.

You can determine the child's reachability by querying the `is_reachable` attribute in for example DB IDO.

3.10.1 Implicit Dependencies for Services on Host

Icinga 2 automatically adds an implicit dependency for services on their host. That way service notifications are suppressed when a host is `DOWN` or `UNREACHABLE`. This dependency does not overwrite other dependencies and implicitely sets `disable_notifications = true` and `states = [ Up ]` for all service objects.

Service checks are still executed. If you want to prevent them from happening, you can apply the following dependency to all services setting their host as `parent_host_name` and disabling the checks. `assign where true` matches on all `Service` objects.

```
apply Dependency "disable-host-service-checks" to Service {
  disable_checks = true
  assign where true
}
```

3.10.2 Dependencies for Network Reachability

A common scenario is the Icinga 2 server behind a router. Checking internet access by pinging the Google DNS server `google-dns` is a common method, but will fail in case the `dsl-router` host is down. Therefore the example below defines a host dependency which acts implicitly as parent relation too.

Furthermore the host may be reachable but ping probes are dropped by the router's firewall. In case the `dsl-router`'s `ping4` service check fails, all further checks for the `ping4` service on host `google-dns` service should be suppressed. This is achieved by setting the `disable_checks` attribute to `true`.

```
object Host "dsl-router" {
  import "generic-host"
```

```
  address = "192.168.1.1"
}

object Host "google-dns" {
  import "generic-host"
  address = "8.8.8.8"
}

apply Service "ping4" {
  import "generic-service"

  check_command = "ping4"

  assign where host.address
}

apply Dependency "internet" to Host {
  parent_host_name = "dsl-router"
  disable_checks = true
  disable_notifications = true

  assign where host.name != "dsl-router"
}

apply Dependency "internet" to Service {
  parent_host_name = "dsl-router"
  parent_service_name = "ping4"
  disable_checks = true

  assign where host.name != "dsl-router"
}
```

3.10.3 Apply Dependencies based on Custom Attributes

You can use apply rules to set parent or child attributes, e.g. `parent_host_name` to other objects' attributes.

A common example are virtual machines hosted on a master. The object name of that master is auto-generated from your CMDB or VMWare inventory into the host's custom attributes (or a generic template for your cloud).

Define your master host object:

```
/* your master */
object Host "master.example.com" {
  import "generic-host"
```

```
}
```

Add a generic template defining all common host attributes:

```
/* generic template for your virtual machines */
template Host "generic-vm" {
  import "generic-host"
}
```

Add a template for all hosts on your example.com cloud setting custom attribute `vm_parent` to `master.example.com`:

```
template Host "generic-vm-example.com" {
  import "generic-vm"
  vars.vm_parent = "master.example.com"
}
```

Define your guest hosts:

```
object Host "www.example1.com" {
  import "generic-vm-master.example.com"
}

object Host "www.example2.com" {
  import "generic-vm-master.example.com"
}
```

Apply the host dependency to all child hosts importing the `generic-vm` template and set the `parent_host_name` to the previously defined custom attribute `host.vars.vm_parent`.

```
apply Dependency "vm-host-to-parent-master" to Host {
  parent_host_name = host.vars.vm_parent
  assign where "generic-vm" in host.templates
}
```

You can extend this example, and make your services depend on the `master.example.com` host too. Their local scope allows you to use `host.vars.vm_parent` similar to the example above.

```
apply Dependency "vm-service-to-parent-master" to Service {
  parent_host_name = host.vars.vm_parent
  assign where "generic-vm" in host.templates
}
```

That way you don't need to wait for your guest hosts becoming unreachable when the master host goes down. Instead the services will detect their reachability immediately when executing checks.

Note

This method with setting locally scoped variables only works in apply rules, but not in object definitions.

3.10.4 Dependencies for Agent Checks

Another classic example are agent based checks. You would define a health check for the agent daemon responding to your requests, and make all other services querying that daemon depend on that health check.

The following configuration defines two nrpe based service checks `nrpe-load` and `nrpe-disk` applied to the host `nrpe-server` matched by its name. The health check is defined as `nrpe-health` service.

```
apply Service "nrpe-health" {
  import "generic-service"
  check_command = "nrpe"
  assign where match("nrpe-*", host.name)
}

apply Service "nrpe-load" {
  import "generic-service"
  check_command = "nrpe"
  vars.nrpe_command = "check_load"
  assign where match("nrpe-*", host.name)
}

apply Service "nrpe-disk" {
  import "generic-service"
  check_command = "nrpe"
  vars.nrpe_command = "check_disk"
  assign where match("nrpe-*", host.name)
}

object Host "nrpe-server" {
  import "generic-host"
  address = "192.168.1.5"
}

apply Dependency "disable-nrpe-checks" to Service {
  parent_service_name = "nrpe-health"

  states = [ OK ]
  disable_checks = true
  disable_notifications = true
  assign where service.check_command == "nrpe"
  ignore where service.name == "nrpe-health"
```

```
}
```

The `disable-nrpe-checks` dependency is applied to all services on the `nrpe-service` host using the `nrpe` check_command attribute but not the `nrpe-health` service itself.

3.10.5 Event Commands

Unlike notifications, event commands for hosts/services are called on every check execution if one of these conditions matches:

- The host/service is in a soft state
- The host/service state changes into a hard state
- The host/service state recovers from a soft or hard state to OK/Up

EventCommand objects are referenced by Host and Service objects with the `event_command` attribute.

Therefore the `EventCommand` object should define a command line evaluating the current service state and other service runtime attributes available through runtime variables. Runtime macros such as `$service.state_type$` and `$service.state$` will be processed by Icinga 2 and help with fine-granular triggered events

If the host/service is located on a client as command endpoint the event command will be executed on the client itself (similar to the check command).

Common use case scenarios are a failing HTTP check which requires an immediate restart via event command. Another example would be an application that is not responding and therefore requires a restart. You can also use event handlers to forward more details on state changes and events than the typical notification alerts provide.

3.10.5.1 Use Event Commands to Send Information from the Master

This example sends a web request from the master node to an external tool for every event triggered on a `businessprocess` service.

Define an EventCommand object `send_to_businesstool` which sends state changes to the external tool.

```
object EventCommand "send_to_businesstool" {
  command = [
    "/usr/bin/curl",
    "-s",
    "-X PUT"
  ]
```

```
  arguments = {
    "-H" = {
      value ="$businesstool_url$"
      skip_key = true
    }
    "-d" = "$businesstool_message$"
  }

  vars.businesstool_url = "http://localhost:8080/businesstool"
 vars.businesstool_message = "$host.name$ $service.name$ $service.state$ $service.state_type$
}
```

Set the `event_command` attribute to `send_to_businesstool` on the Service.

```
object Service "businessprocess" {
  host_name = "businessprocess"

  check_command = "icingacli-businessprocess"
  vars.icingacli_businessprocess_process = "icinga"
  vars.icingacli_businessprocess_config = "training"

  event_command = "send_to_businesstool"
}
```

In order to test this scenario you can run:

```
nc -l 8080
```

This allows to catch the web request. You can also enable the debug log and search for the event command execution log message.

```
tail -f /var/log/icinga2/debug.log | grep EventCommand
```

Feed in a check result via REST API action process-check-result or via Icinga Web 2.

Expected Result:

```
# nc -l 8080
PUT /businesstool HTTP/1.1
User-Agent: curl/7.29.0
Host: localhost:8080
Accept: */*
Content-Length: 47
Content-Type: application/x-www-form-urlencoded

businessprocess businessprocess CRITICAL SOFT 1
```

3.10.5.2 Use Event Commands to Restart Service Daemon via Command Endpoint on Linux

This example triggers a restart of the `httpd` service on the local system when the `procs` service check executed via Command Endpoint fails. It only triggers if the service state is `Critical` and attempts to restart the service before a notification is sent.

Requirements:

- Icinga 2 as client on the remote node
- icinga user with sudo permissions to the httpd daemon

Example on CentOS 7:

```
# visudo
icinga  ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart httpd
```

Note: Distributions might use a different name. On Debian/Ubuntu the service is called `apache2`.

Define an EventCommand object `restart_service` which allows to trigger local service restarts. Put it into a global zone to sync its configuration to all clients.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/global-templates/eventcommands

object EventCommand "restart_service" {
  command = [ PluginDir + "/restart_service" ]

  arguments = {
    "-s" = "$service.state$"
    "-t" = "$service.state_type$"
    "-a" = "$service.check_attempt$"
    "-S" = "$restart_service$"
  }

  vars.restart_service = "$procs_command$"
}
```

This event command triggers the following script which restarts the service. The script only is executed if the service state is `CRITICAL`. Warning and Unknown states are ignored as they indicate not an immediate failure.

```
[root@icinga2-client1.localdomain /]# vim /usr/lib64/nagios/plugins/restart_service

#!/bin/bash

while getopts "s:t:a:S:" opt; do
  case $opt in
    s)
      servicestate=$OPTARG
```

```
      ;;
    t)
      servicestatetype=$OPTARG
      ;;
    a)
      serviceattempt=$OPTARG
      ;;
    S)
      service=$OPTARG
      ;;
  esac
done

if ( [ -z $servicestate ] || [ -z $servicestatetype ] || [ -z $serviceattempt ] || [ -z $service ]
 echo "USAGE: $0 -s servicestate -z servicestatetype -a serviceattempt -S service"
  exit 3;
else
  # Only restart on the third attempt of a critical event
 if ( [ $servicestate == "CRITICAL" ] && [ $servicestatetype == "SOFT" ] && [ $serviceattempt -ec
    sudo /usr/bin/systemctl restart $service
  fi
fi
```

```
[root@icinga2-client1.localdomain /]# chmod +x /usr/lib64/nagios/plugins/restart_service
```

Add a service on the master node which is executed via command endpoint on the client. Set the `event_command` attribute to `restart_service`, the name of the previously defined EventCommand object.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/icinga2-client1.localdo
```

```
object Service "Process httpd" {
  check_command = "procs"
  event_command = "restart_service"
  max_check_attempts = 4

  host_name = "icinga2-client1.localdomain"
  command_endpoint = "icinga2-client1.localdomain"

  vars.procs_command = "httpd"
  vars.procs_warning = "1:10"
  vars.procs_critical = "1:"
}
```

In order to test this configuration just stop the `httpd` on the remote host `icinga2-client1.localdomain`.

```
[root@icinga2-client1.localdomain /]# systemctl stop httpd
```

You can enable the debug log and search for the executed command line.

```
[root@icinga2-client1.localdomain /]# tail -f /var/log/icinga2/debug.log | grep restart_servic
```

3.10.5.3 Use Event Commands to Restart Service Daemon via Command Endpoint on Windows

This example triggers a restart of the `httpd` service on the remote system when the `service-windows` service check executed via Command Endpoint fails. It only triggers if the service state is `Critical` and attempts to restart the service before a notification is sent.

Requirements:

- Icinga 2 as client on the remote node
- Icinga 2 service with permissions to execute Powershell scripts (which is the default)

Define an EventCommand object `restart_service-windows` which allows to trigger local service restarts. Put it into a global zone to sync its configuration to all clients.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/global-templates/eventcommands
```

```
object EventCommand "restart_service-windows" {
  command = [
   "C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe",
    PluginDir + "/restart_service.ps1"
  ]

  arguments = {
    "-ServiceState" = "$service.state$"
    "-ServiceStateType" = "$service.state_type$"
    "-ServiceAttempt" = "$service.check_attempt$"
    "-Service" = "$restart_service$"
    "; exit" = {
        order = 99
        value = "$$LASTEXITCODE"
    }
  }

  vars.restart_service = "$service_win_service$"
}
```

This event command triggers the following script which restarts the service. The script only is executed if the service state is `CRITICAL`. Warning and Unknown states are ignored as they indicate not an immediate failure.

Add the `restart_service.ps1` Powershell script into `C:\Program Files\Icinga2\sbin`:

```
param(
        [string]$Service                = '',
        [string]$ServiceState           = '',
        [string]$ServiceStateType       = '',
        [int]$ServiceAttempt            = ''
    )

if (!$Service -Or !$ServiceState -Or !$ServiceStateType -Or !$ServiceAttempt) {
   $scriptName = GCI $MyInvocation.PSCommandPath | Select -Expand Name;
   Write-Host "USAGE: $scriptName -ServiceState servicestate -ServiceStateType servicestatetyp
    exit 3;
}

# Only restart on the third attempt of a critical event
if ($ServiceState -eq "CRITICAL" -And $ServiceStateType -eq "SOFT" -And $ServiceAttempt -eq 3) {
    Restart-Service $Service;
}

exit 0;
```

Add a service on the master node which is executed via command endpoint on the client. Set the `event_command` attribute to `restart_service-windows`, the name of the previously defined EventCommand object.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/icinga2-client2.localdo

object Service "Service httpd" {
  check_command = "service-windows"
  event_command = "restart_service-windows"
  max_check_attempts = 4

  host_name = "icinga2-client2.localdomain"
  command_endpoint = "icinga2-client2.localdomain"

  vars.service_win_service = "httpd"
}
```

In order to test this configuration just stop the `httpd` on the remote host `icinga2-client1.localdomain`.

```
C:> net stop httpd
```

You can enable the debug log and search for the executed command line in `C:\ProgramData\icinga2\var\log\icinga2\debug.log`.

3.10.5.4 Use Event Commands to Restart Service Daemon via SSH

This example triggers a restart of the `httpd` daemon via SSH when the `http` service check fails.

Requirements:

- SSH connection allowed (firewall, packet filters)
- icinga user with public key authentication
- icinga user with sudo permissions to restart the httpd daemon.

Example on Debian:

```
# ls /home/icinga/.ssh/
authorized_keys

# visudo
icinga  ALL=(ALL) NOPASSWD: /etc/init.d/apache2 restart
```

Define a generic EventCommand object `event_by_ssh` which can be used for all event commands triggered using SSH:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/local_eventcommands.co

/* pass event commands through ssh */
object EventCommand "event_by_ssh" {
  command = [ PluginDir + "/check_by_ssh" ]

  arguments = {
    "-H" = "$event_by_ssh_address$"
    "-p" = "$event_by_ssh_port$"
    "-C" = "$event_by_ssh_command$"
    "-l" = "$event_by_ssh_logname$"
    "-i" = "$event_by_ssh_identity$"
    "-q" = {
      set_if = "$event_by_ssh_quiet$"
    }
    "-w" = "$event_by_ssh_warn$"
    "-c" = "$event_by_ssh_crit$"
    "-t" = "$event_by_ssh_timeout$"
  }

  vars.event_by_ssh_address = "$address$"
  vars.event_by_ssh_quiet = false
}
```

The actual event command only passes the `event_by_ssh_command` attribute. The `event_by_ssh_service` custom attribute takes care of passing the correct daemon name, while `test $service.state_id$ -gt 0` makes sure that the daemon is only restarted when the service is not in an `OK` state.

```
object EventCommand "event_by_ssh_restart_service" {
```

```
  import "event_by_ssh"

  //only restart the daemon if state > 0 (not-ok)
  //requires sudo permissions for the icinga user
 vars.event_by_ssh_command = "test $service.state_id$ -gt 0 && sudo systemctl restart $event_by
}
```

Now set the `event_command` attribute to `event_by_ssh_restart_service` and tell it which service should be restarted using the `event_by_ssh_service` attribute.

```
apply Service "http" {
  import "generic-service"
  check_command = "http"

  event_command = "event_by_ssh_restart_service"
  vars.event_by_ssh_service = "$host.vars.httpd_name$"

  //vars.event_by_ssh_logname = "icinga"
  //vars.event_by_ssh_identity = "/home/icinga/.ssh/id_rsa.pub"

  assign where host.vars.httpd_name
}
```

Specify the `httpd_name` custom attribute on the host to assign the service and set the event handler service.

```
object Host "remote-http-host" {
  import "generic-host"
  address = "192.168.1.100"

  vars.httpd_name = "apache2"
}
```

In order to test this configuration just stop the `httpd` on the remote host `icinga2-client1.localdomain`.

```
[root@icinga2-client1.localdomain /]# systemctl stop httpd
```

You can enable the debug log and search for the executed command line.

```
[root@icinga2-client1.localdomain /]# tail -f /var/log/icinga2/debug.log | grep by_ssh
```

4 Configuring Icinga 2: First Steps

This chapter provides an introduction into best practices for your Icinga 2 configuration. The configuration files which are automatically created when installing the Icinga 2 packages are a good way to start with Icinga 2.

The Language Reference chapter explains details on value types (string, number, dictionaries, etc.) and the general configuration syntax.

4.1 Configuration Best Practice

If you are ready to configure additional hosts, services, notifications, dependencies, etc., you should think about the requirements first and then decide for a possible strategy.

There are many ways of creating Icinga 2 configuration objects:

- Manually with your preferred editor, for example vi(m), nano, notepad, etc.
- A configuration tool for Icinga 2 e.g. the Icinga Director
- Generated by a configuration management tool such as Puppet, Chef, Ansible, etc.
- A custom exporter script from your CMDB or inventory tool
- etc.

Find the best strategy for your own configuration and ask yourself the following questions:

- Do your hosts share a common group of services (for example linux hosts with disk, load, etc. checks)?
- Only a small set of users receives notifications and escalations for all hosts/services?

If you can at least answer one of these questions with yes, look for the apply rules logic instead of defining objects on a per host and service basis.

- You are required to define specific configuration for each host/service?
- Does your configuration generation tool already know about the host-service-relationship?

Then you should look for the object specific configuration setting `host_name` etc. accordingly.

You decide on the "best" layout for configuration files and directories. Ensure that the icinga2.conf configuration file includes them.

Consider these ideas:

- tree-based on locations, host groups, specific host attributes with sub levels of directories.
- flat `hosts.conf`, `services.conf`, etc. files for rule based configuration.
- generated configuration with one file per host and a global configuration for groups, users, etc.
- one big file generated from an external application (probably a bad idea for maintaining changes).
- your own.

In either way of choosing the right strategy you should additionally check the following:

- Are there any specific attributes describing the host/service you could set as `vars` custom attributes? You can later use them for applying assign/ignore rules, or export them into external interfaces.
- Put hosts into hostgroups, services into servicegroups and use these attributes for your apply rules.
- Use templates to store generic attributes for your objects and apply rules making your configuration more readable. Details can be found in the using templates chapter.
- Apply rules may overlap. Keep a central place (for example, services.conf or notifications.conf) storing the configuration instead of defining apply rules deep in your configuration tree.
- Every plugin used as check, notification or event command requires a `Command` definition. Further details can be looked up in the check commands chapter.

If you are planning to use a distributed monitoring setup with master, satellite and client installations take the configuration location into account too. Everything configured on the master, synced to all other nodes? Or any specific local configuration (e.g. health checks)?

There is a detailed chapter on distributed monitoring scenarios. Please ensure to have read the introduction at first glance.

If you happen to have further questions, do not hesitate to join the community support channels and ask community members for their experience and best practices.

4.2 Your Configuration

If you prefer to organize your own local object tree, you can also remove `include_recursive "conf.d"` from your icinga2.conf file.

Create a new configuration directory, e.g. `objects.d` and include it in your icinga2.conf file.

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/objects.d

[root@icinga2-master1.localdomain /]# vim /etc/icinga2/icinga2.conf

/* Local object configuration on our master instance. */
include_recursive "objects.d"
```

This approach is used by the Icinga 2 Puppet module.

If you plan to setup a distributed setup with HA clusters and clients, please refer to this chapter for examples with `zones.d` as configuration directory.

4.3 Configuration Overview

4.3.1 icinga2.conf

An example configuration file is installed for you in `/etc/icinga2/icinga2.conf`.

Here's a brief description of the example configuration:

```
/**
 * Icinga 2 configuration file
 * -- this is where you define settings for the Icinga application including
 * which hosts/services to check.
 *
 * For an overview of all available configuration options please refer
 * to the documentation that is distributed as part of Icinga 2.
 */
```

Icinga 2 supports C/C++-style comments.

```
/**
 * The constants.conf defines global constants.
 */
include "constants.conf"
```

The `include` directive can be used to include other files.

```
/**
 * The zones.conf defines zones for a cluster setup.
 * Not required for single instance setups.
 */
 include "zones.conf"
```

The Icinga Template Library provides a set of common templates and CheckCommand definitions.

```
/**
 * The Icinga Template Library (ITL) provides a number of useful templates
 * and command definitions.
 * Common monitoring plugin command definitions are included separately.
 */
include <itl>
include <plugins>
include <plugins-contrib>
include <manubulon>

/**
 * This includes the Icinga 2 Windows plugins. These command definitions
 * are required on a master node when a client is used as command endpoint.
 */
```

```
include <windows-plugins>

/**
 * This includes the NSClient++ check commands. These command definitions
 * are required on a master node when a client is used as command endpoint.
 */
include <nscp>

/**
 * The features-available directory contains a number of configuration
 * files for features which can be enabled and disabled using the
 * icinga2 feature enable / icinga2 feature disable CLI commands.
 * These commands work by creating and removing symbolic links in
 * the features-enabled directory.
 */
include "features-enabled/*.conf"
```

This `include` directive takes care of including the configuration files for all the features which have been enabled with `icinga2 feature enable`. See Enabling/Disabling Features for more details.

```
/**
 * Although in theory you could define all your objects in this file
 * the preferred way is to create separate directories and files in the conf.d
 * directory. Each of these files must have the file extension ".conf".
 */
include_recursive "conf.d"
```

You can put your own configuration files in the conf.d directory. This directive makes sure that all of your own configuration files are included.

4.3.2 constants.conf

The `constants.conf` configuration file can be used to define global constants.

By default, you need to make sure to set these constants:

- The `PluginDir` constant must be set to the path where the Monitoring Project plugins are installed. This constant is used by a number of built-in check command definitions.
- The `NodeName` constant defines your local node name. Should be set to FQDN which is the default if not set. This constant is required for local host configuration, monitoring remote clients and cluster setup.

Example:

```
/* The directory which contains the plugins from the Monitoring Plugins project. */
const PluginDir = "/usr/lib64/nagios/plugins"
```

```
/* The directory which contains the Manubulon plugins.
 * Check the documentation, chapter "SNMP Manubulon Plugin Check Commands", for details.
 */
const ManubulonPluginDir = "/usr/lib64/nagios/plugins"

/* Our local instance name. By default this is the server's hostname as returned by `hostname --f
 * This should be the common name from the API certificate.
 */
//const NodeName = "localhost"

/* Our local zone name. */
const ZoneName = NodeName

/* Secret key for remote node tickets */
const TicketSalt = ""
```

The `ZoneName` and `TicketSalt` constants are required for remote client and distributed setups only.

4.3.3 zones.conf

This file can be used to specify the required Zone and Endpoint configuration object for distributed monitoring.

By default the `NodeName` and `ZoneName` constants will be used.

It also contains several global zones for distributed monitoring environments.

Please ensure to modify this configuration with real names i.e. use the FQDN mentioned in this chapter for your `Zone` and `Endpoint` object names.

4.3.4 The conf.d Directory

This directory contains **example configuration** which should help you get started with monitoring the local host and its services. It is included in the icinga2.conf configuration file by default.

It can be used as reference example for your own configuration strategy. Just keep in mind to include the main directories in the icinga2.conf file.

> **Note**
>
> You can remove the include directive in icinga2.conf if you prefer your own way of deploying Icinga 2 configuration.

Further details on configuration best practice and how to build your own strategy is described in this chapter.

Available configuration files which are installed by default:

- hosts.conf
- services.conf
- users.conf
- notifications.conf
- commands.conf
- groups.conf
- templates.conf
- downtimes.conf
- timeperiods.conf
- api-users.conf
- app.conf

4.3.4.1 hosts.conf

The `hosts.conf` file contains an example host based on your `NodeName` setting in constants.conf. You can use global constants for your object names instead of string values.

The `import` keyword is used to import the `generic-host` template which takes care of setting up the host check command to `hostalive`. If you require a different check command, you can override it in the object definition.

The `vars` attribute can be used to define custom attributes which are available for check and notification commands. Most of the Plugin Check Commands in the Icinga Template Library require an `address` attribute.

The custom attribute `os` is evaluated by the `linux-servers` group in groups.conf making the local host a member.

The example host will show you how to:

- define http vhost attributes for the `http` service apply rule defined in services.conf.
- define disks (all, specific `/`) and their attributes for the `disk` service apply rule defined in services.conf.
- define notification types (`mail`) and set the groups attribute. This will be used by notification apply rules in notifications.conf.

If you've installed Icinga Web 2, you can uncomment the http vhost attributes and reload Icinga 2. The apply rules in services.conf will automatically generate a new service checking the `/icingaweb2` URI using the `http` check.

```
/*
 * Host definitions with object attributes
 * used for apply rules for Service, Notification,
 * Dependency and ScheduledDowntime objects.
 *
```

```
 * Tip: Use `icinga2 object list --type Host` to
 * list all host objects after running
 * configuration validation (`icinga2 daemon -C`).
 */

/*
 * This is an example host based on your
 * local host's FQDN. Specify the NodeName
 * constant in `constants.conf` or use your
 * own description, e.g. "db-host-1".
 */

object Host NodeName {
  /* Import the default host template defined in `templates.conf`. */
  import "generic-host"

  /* Specify the address attributes for checks e.g. `ssh` or `http`. */
  address = "127.0.0.1"
  address6 = "::1"

  /* Set custom attribute `os` for hostgroup assignment in `groups.conf`. */
  vars.os = "Linux"

  /* Define http vhost attributes for service apply rules in `services.conf`. */
  vars.http_vhosts["http"] = {
    http_uri = "/"
  }
  /* Uncomment if you've sucessfully installed Icinga Web 2. */
  //vars.http_vhosts["Icinga Web 2"] = {
  //  http_uri = "/icingaweb2"
  //}

  /* Define disks and attributes for service apply rules in `services.conf`. */
  vars.disks["disk"] = {
    /* No parameters. */
  }
  vars.disks["disk /"] = {
    disk_partitions = "/"
  }

  /* Define notification mail attributes for notification apply rules in `notifications.conf`. *
  vars.notification["mail"] = {
    /* The UserGroup `icingaadmins` is defined in `users.conf`. */
    groups = [ "icingaadmins" ]
  }
}
```

This is only the host object definition. Now we'll need to make sure that this host and your additional hosts are getting services applied.

Tip

If you don't understand all the attributes and how to use apply rules, don't worry – the monitoring basics chapter will explain that in detail.

4.3.4.2 services.conf

These service apply rules will show you how to monitor the local host, but also allow you to re-use or modify them for your own requirements.

You should define all your service apply rules in `services.conf` or any other central location keeping them organized.

By default, the local host will be monitored by the following services

Service(s)	Applied on host(s)
`load`, `procs`, `swap`, `users`, `icinga`	The `NodeName` host only.
`ping4`, `ping6`	All hosts with `address` resp. `address6` attribute.
`ssh`	All hosts with `address` and `vars.os` set to `Linux`
`http`, optional: `Icinga Web 2`	All hosts with custom attribute `http_vhosts` defined as dictionary.
`disk`, `disk /`	All hosts with custom attribute `disks` defined as dictionary.

The Debian packages also include an additional `apt` service check applied to the local host.

The command object `icinga` for the embedded health check is provided by the Icinga Template Library (ITL) while `http_ip`, `ssh`, `load`, `processes`, `users` and `disk` are all provided by the Plugin Check Commands which we enabled earlier by including the `itl` and `plugins` configuration file.

Example `load` service apply rule:

```
apply Service "load" {
  import "generic-service"

  check_command = "load"
```

```
  /* Used by the ScheduledDowntime apply rule in `downtimes.conf`. */
  vars.backup_downtime = "02:00-03:00"

  assign where host.name == NodeName
}
```

The `apply` keyword can be used to create new objects which are associated with another group of objects. You can `import` existing templates, define (custom) attributes.

The custom attribute `backup_downtime` is defined to a specific timerange string. This variable value will be used for applying a `ScheduledDowntime` object to these services in downtimes.conf.

In this example the `assign where` condition is a boolean expression which is evaluated for all objects of type `Host` and a new service with name "load" is created for each matching host. Expression operators may be used in `assign where` conditions.

Multiple `assign where` condition can be combined with `AND` using the `&&` operator as shown in the `ssh` example:

```
apply Service "ssh" {
  import "generic-service"

  check_command = "ssh"

  assign where host.address && host.vars.os == "Linux"
}
```

In this example, the service `ssh` is applied to all hosts having the `address` attribute defined `AND` having the custom attribute `os` set to the string `Linux`. You can modify this condition to match multiple expressions by combining `AND` and `OR` using `&&` and `||` operators, for example `assign where host.address && (vars.os == "Linux" || vars.os == "Windows")`.

A more advanced example is shown by the `http` and `disk` service apply rules. While one `apply` rule for `ssh` will only create a service for matching hosts, you can go one step further: Generate apply rules based on array items or dictionary key-value pairs.

The idea is simple: Your host in hosts.conf defines the `disks` dictionary as custom attribute in `vars`.

Remember the example from hosts.conf:

```
...
 /* Define disks and attributes for service apply rules in `services.conf`. */
  vars.disks["disk"] = {
    /* No parameters. */
  }
```

```
  vars.disks["disk /"] = {
    disk_partition = "/"
  }
...
```

This dictionary contains multiple service names we want to monitor. `disk` should just check all available disks, while `disk /` will pass an additional parameter `disk_partition` to the check command.

You'll recognize that the naming is important – that's the very same name as it is passed from a service to a check command argument. Read about services and passing check commands in this chapter.

Using `apply Service for` omits the service name, it will take the key stored in the `disk` variable in `key => config` as new service object name.

The `for` keyword expects a loop definition, for example `key => value in dictionary` as known from Perl and other scripting languages.

Once defined like this, the `apply` rule defined below will do the following:

- only match hosts with `host.vars.disks` defined through the `assign where` condition
- loop through all entries in the `host.vars.disks` dictionary. That's `disk` and `disk /` as keys.
- call `apply` on each, and set the service object name from the provided key
- inside apply, the `generic-service` template is imported
- defining the disk check command requiring command arguments like `disk_partition`
- adding the `config` dictionary items to `vars`. Simply said, there's now `vars.disk_partition` defined for the generated service

Configuration example:

```
apply Service for (disk => config in host.vars.disks) {
  import "generic-service"

  check_command = "disk"

  vars += config
}
```

A similar example is used for the `http` services. That way you can make your host the information provider for all apply rules. Define them once, and only manage your hosts.

Look into notifications.conf how this technique is used for applying notifications to hosts and services using their type and user attributes.

Don't forget to install the check plugins required by the hosts and services and their check commands.

Further details on the monitoring configuration can be found in the monitoring basics chapter.

4.3.4.3 users.conf

Defines the `icingaadmin` User and the `icingaadmins` UserGroup. The latter is used in hosts.conf for defining a custom host attribute later used in notifications.conf for notification apply rules.

```
object User "icingaadmin" {
  import "generic-user"

  display_name = "Icinga 2 Admin"
  groups = [ "icingaadmins" ]

  email = "icinga@localhost"
}

object UserGroup "icingaadmins" {
  display_name = "Icinga 2 Admin Group"
}
```

4.3.4.4 notifications.conf

Notifications for check alerts are an integral part or your Icinga 2 monitoring stack.

The examples in this file define two notification apply rules for hosts and services. Both `apply` rules match on the same condition: They are only applied if the nested dictionary attribute `notification.mail` is set.

Please note that the `to` keyword is important in notification apply rules defining whether these notifications are applies to hosts or services. The `import` keyword imports the specific mail templates defined in templates.conf.

The `interval` attribute is not explicitly set – it defaults to 30 minutes.

By setting the `user_groups` to the value provided by the respective host.vars.notification.mail attribute we'll implicitely use the `icingaadmins` UserGroup defined in users.conf.

```
apply Notification "mail-icingaadmin" to Host {
  import "mail-host-notification"

  user_groups = host.vars.notification.mail.groups
  users = host.vars.notification.mail.users

  assign where host.vars.notification.mail
```

```
}

apply Notification "mail-icingaadmin" to Service {
  import "mail-service-notification"

  user_groups = host.vars.notification.mail.groups
  users = host.vars.notification.mail.users

  assign where host.vars.notification.mail
}
```

More details on defining notifications and their additional attributes such as filters can be read in this chapter.

4.3.4.5 commands.conf

This is the place where your own command configuration can be defined. By default only the notification commands used by the notification templates defined in templates.conf.

You can freely customize these notification commands, and adapt them for your needs. Read more on that topic here.

4.3.4.6 groups.conf

The example host defined in hosts.conf already has the custom attribute `os` set to `Linux` and is therefore automatically a member of the host group `linux-servers`.

This is done by using the group assign expressions similar to previously seen apply rules.

```
object HostGroup "linux-servers" {
  display_name = "Linux Servers"

  assign where host.vars.os == "Linux"
}

object HostGroup "windows-servers" {
  display_name = "Windows Servers"

  assign where host.vars.os == "Windows"
}
```

Service groups can be grouped together by similar pattern matches. The match function expects a wildcard match string and the attribute string to match with.

```
object ServiceGroup "ping" {
```

```
  display_name = "Ping Checks"

  assign where match("ping*", service.name)
}

object ServiceGroup "http" {
  display_name = "HTTP Checks"

  assign where match("http*", service.check_command)
}

object ServiceGroup "disk" {
  display_name = "Disk Checks"

  assign where match("disk*", service.check_command)
}
```

4.3.4.7 templates.conf

Most of the example configuration objects use generic global templates by default:

```
template Host "generic-host" {
  max_check_attempts = 5
  check_interval = 1m
  retry_interval = 30s

  check_command = "hostalive"
}

template Service "generic-service" {
  max_check_attempts = 3
  check_interval = 1m
  retry_interval = 30s
}
```

The `hostalive` check command is part of the Plugin Check Commands.

```
template Notification "mail-host-notification" {
  command = "mail-host-notification"

  states = [ Up, Down ]
  types = [ Problem, Acknowledgement, Recovery, Custom,
            FlappingStart, FlappingEnd,
            DowntimeStart, DowntimeEnd, DowntimeRemoved ]

  period = "24x7"
```

```
}

template Notification "mail-service-notification" {
  command = "mail-service-notification"

  states = [ OK, Warning, Critical, Unknown ]
  types = [ Problem, Acknowledgement, Recovery, Custom,
            FlappingStart, FlappingEnd,
            DowntimeStart, DowntimeEnd, DowntimeRemoved ]

  period = "24x7"
}
```

More details on `Notification` object attributes can be found here.

4.3.4.8 downtimes.conf

The `load` service apply rule defined in services.conf defines the `backup_downtime` custom attribute.

The ScheduledDowntime apply rule uses this attribute to define the default value for the time ranges required for recurring downtime slots.

```
apply ScheduledDowntime "backup-downtime" to Service {
  author = "icingaadmin"
  comment = "Scheduled downtime for backup"

  ranges = {
    monday = service.vars.backup_downtime
    tuesday = service.vars.backup_downtime
    wednesday = service.vars.backup_downtime
    thursday = service.vars.backup_downtime
    friday = service.vars.backup_downtime
    saturday = service.vars.backup_downtime
    sunday = service.vars.backup_downtime
  }

  assign where service.vars.backup_downtime != ""
}
```

4.3.4.9 timeperiods.conf

This file contains the default timeperiod definitions for `24x7`, `9to5` and `never`. TimePeriod objects are referenced by `*period` objects such as hosts, services or notifications.

4.3.4.10 api-users.conf

Provides the default ApiUser object named "root" for the API authentication.

4.3.4.11 app.conf

Provides the default IcingaApplication object named "app" for additional settings such as disabling notifications globally, etc.

5 Service Monitoring

The power of Icinga 2 lies in its modularity. There are thousands of community plugins available next to the standard plugins provided by the Monitoring Plugins project.

5.1 Requirements

5.1.1 Plugins

All existing Nagios or Icinga 1.x plugins work with Icinga 2. Community plugins can be found for example on Icinga Exchange.

The recommended way of setting up these plugins is to copy them to a common directory and create a new global constant, e.g. `CustomPluginDir` in your constants.conf configuration file:

```
# cp check_snmp_int.pl /opt/monitoring/plugins
# chmod +x /opt/plugins/check_snmp_int.pl

# cat /etc/icinga2/constants.conf
/**
 * This file defines global constants which can be used in
 * the other configuration files. At a minimum the
 * PluginDir constant should be defined.
 */

const PluginDir = "/usr/lib/nagios/plugins"
const CustomPluginDir = "/opt/monitoring/plugins"
```

Prior to using the check plugin with Icinga 2 you should ensure that it is working properly by trying to run it on the console using whichever user Icinga 2 is running as:

```
# su - icinga -s /bin/bash
$ /opt/monitoring/plugins/check_snmp_int.pl --help
```

Additional libraries may be required for some plugins. Please consult the plugin documentation and/or the included README file for installation instructions. Sometimes plugins contain hard-coded paths to other components. Instead of changing the plugin it might be easier to create a symbolic link to make sure it doesn't get overwritten during the next update.

Sometimes there are plugins which do not exactly fit your requirements. In that case you can modify an existing plugin or just write your own.

5.1.2 CheckCommand Definition

Each plugin requires a CheckCommand object in your configuration which can be used in the Service or Host object definition.

Please check if the Icinga 2 package already provides an existing CheckCommand definition. If that's the case, throroughly check the required parameters and integrate the check command into your host and service objects.

Please make sure to follow these conventions when adding a new command object definition:

- Use command arguments whenever possible. The `command` attribute must be an array in `[ ... ]` for shell escaping.
- Define a unique `prefix` for the command's specific arguments. That way you can safely set them on host/service level and you'll always know which command they control.
- Use command argument default values, e.g. for thresholds.
- Use advanced conditions like `set_if` definitions.

This is an example for a custom `my-snmp-int` check command:

```
object CheckCommand "my-snmp-int" {
  command = [ CustomPluginDir + "/check_snmp_int.pl" ]

  arguments = {
    "-H" = "$snmp_address$"
    "-C" = "$snmp_community$"
    "-p" = "$snmp_port$"
    "-2" = {
      set_if = "$snmp_v2$"
    }
    "-n" = "$snmp_interface$"
    "-f" = {
      set_if = "$snmp_perf$"
    }
    "-w" = "$snmp_warn$"
    "-c" = "$snmp_crit$"
  }
```

```
  vars.snmp_v2 = true
  vars.snmp_perf = true
  vars.snmp_warn = "300,400"
  vars.snmp_crit = "0,600"
}
```

For further information on your monitoring configuration read the Monitoring Basics chapter.

If you have created your own `CheckCommand` definition, please kindly send it upstream.

5.1.3 Plugin API

Currently Icinga 2 supports the native plugin API specification from the Monitoring Plugins project. It is defined in the Monitoring Plugins Development Guidelines.

5.1.4 Create a new Plugin

Sometimes an existing plugin does not satisfy your requirements. You can either kindly contact the original author about plans to add changes and/or create a patch.

If you just want to format the output and state of an existing plugin it might also be helpful to write a wrapper script. This script could pass all configured parameters, call the plugin script, parse its output/exit code and return your specified output/exit code.

On the other hand plugins for specific services and hardware might not yet exist.

Common best practices when creating a new plugin are for example:

- Choose the programming language wisely
- Scripting languages (Bash, Python, Perl, Ruby, PHP, etc.) are easier to write and setup but their check execution might take longer (invoking the script interpreter as overhead, etc.).
- Plugins written in C/C++, Go, etc. improve check execution time but may generate an overhead with installation and packaging.
- Use a modern VCS such as Git for developing the plugin (e.g. share your plugin on GitHub).
- Add parameters with key-value pairs to your plugin. They should allow long names (e.g. `--host localhost`) and also short parameters (e.g. `-H localhost`)
- `-h|--help` should print the version and all details about parameters and runtime invocation.

- Add a verbose/debug output functionality for detailed on-demand logging.
- Respect the exit codes required by the Plugin API.
- Always add performance data to your plugin output

Example skeleton:

```
# 1. include optional libraries
# 2. global variables
# 3. helper functions and/or classes
# 4. define timeout condition

if (<timeout_reached>) then
  print "UNKNOWN - Timeout (...) reached | 'time'=30.0
endif

# 5. main method

<execute and fetch data>

if (<threshold_critical_condition>) then
  print "CRITICAL - ... | 'time'=0.1 'myperfdatavalue'=5.0
  exit(2)
else if (<threshold_warning_condition>) then
  print "WARNING - ... | 'time'=0.1 'myperfdatavalue'=3.0
  exit(1)
else
  print "OK - ... | 'time'=0.2 'myperfdatavalue'=1.0
endif
```

There are various plugin libraries available which will help with plugin execution and output formatting too, for example nagiosplugin from Python.

> **Note**
>
> Ensure to test your plugin properly with special cases before putting it into production!

Once you've finished your plugin please upload/sync it to Icinga Exchange. Thanks in advance!

5.2 Service Monitoring Overview

The following examples should help you to start implementing your own ideas. There is a variety of plugins available. This collection is not complete – if you have any updates, please send a documentation patch upstream.

5.2.1 General Monitoring

If the remote service is available (via a network protocol and port), and if a check plugin is also available, you don't necessarily need a local client. Instead, choose a plugin and configure its parameters and thresholds. The following examples are included in the Icinga 2 Template Library:

- ping4, ping6, fping4, fping6, hostalive
- tcp, udp, ssl
- ntp_time

5.2.2 Linux Monitoring

- disk
- mem, swap
- procs
- users
- running_kernel
- package management: apt, yum, etc.
- ssh
- performance: iostat, check_sar_perf

5.2.3 Windows Monitoring

- check_wmi_plus
- NSClient++ (in combination with the Icinga 2 client and either check_nscp_api or nscp-local check commands)
- Icinga 2 Windows Plugins (disk, load, memory, network, performance counters, ping, procs, service, swap, updates, uptime, users
- vbs and Powershell scripts

5.2.4 Database Monitoring

- MySQL/MariaDB: mysql_health, mysql, mysql_query
- PostgreSQL: postgres
- Oracle: oracle_health
- MSSQL: mssql_health
- DB2: db2_health
- MongoDB: mongodb
- Elasticsearch: elasticsearch
- Redis: redis

5.2.5 SNMP Monitoring

- Manubulon plugins (interface, storage, load, memory, process)
- snmp, snmpv3

5.2.6 Network Monitoring

- nwc_health
- interfaces
- interfacetable
- iftraffic, iftraffic64

5.2.7 Web Monitoring

- http
- ftp
- webinject
- squid
- apache_status
- nginx_status
- kdc
- rbl

5.2.8 Java Monitoring

- jmx4perl

5.2.9 DNS Monitoring

- dns
- dig
- dhcp

5.2.10 Backup Monitoring

- check_bareos

5.2.11 Log Monitoring

- check_logfiles
- check_logstash

- check_graylog2_stream

5.2.12 Virtualization Monitoring

5.2.13 VMware Monitoring

- esxi_hardware
- VMware

Tip: If you are encountering timeouts using the VMware Perl SDK, check this blog entry.

5.2.14 SAP Monitoring

- check_sap_health
- SAP CCMS

5.2.15 Mail Monitoring

- smtp, ssmtp
- imap, simap
- pop, spop
- mailq

5.2.16 Hardware Monitoring

- hpasm
- ipmi-sensor

5.2.17 Metrics Monitoring

- graphite

6 Distributed Monitoring with Master, Satellites, and Clients

This chapter will guide you through the setup of a distributed monitoring environment, including high-availability clustering and setup details for the Icinga 2 client.

6.1 Roles: Master, Satellites, and Clients

Icinga 2 nodes can be given names for easier understanding:

- A `master` node which is on top of the hierarchy.
- A `satellite` node which is a child of a `satellite` or `master` node.
- A `client` node which works as an `agent` connected to `master` and/or `satellite` nodes.

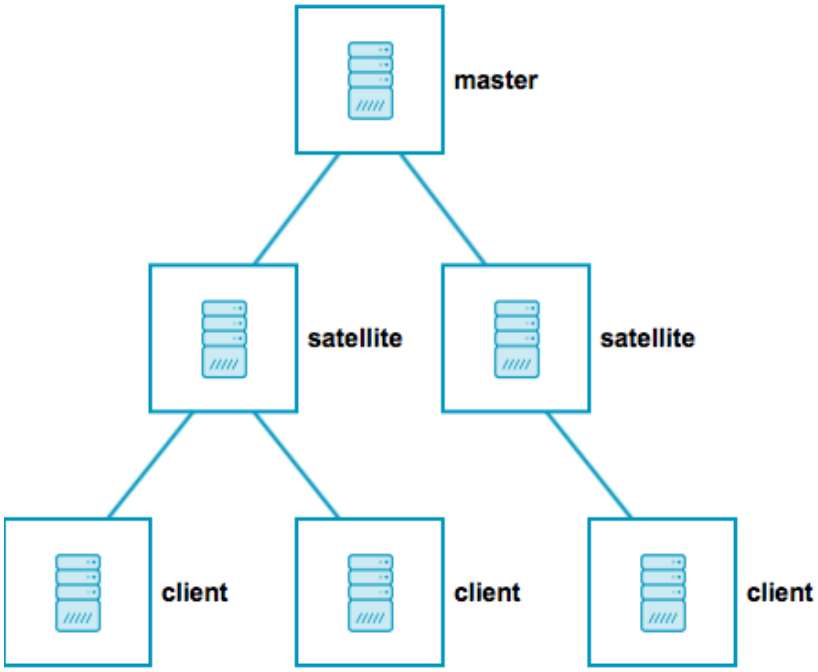


Figure 5: Icinga 2 Distributed Roles

Rephrasing this picture into more details:

- A `master` node has no parent node.
- A `master`node is where you usually install Icinga Web 2.
- A `master` node can combine executed checks from child nodes into backends and notifications.
- A `satellite` node has a parent and a child node.
- A `satellite` node may execute checks on its own or delegate check execution to child nodes.
- A `satellite` node can receive configuration for hosts/services, etc. from the parent node.

- A `satellite` node continues to run even if the master node is temporarily unavailable.
- A `client` node only has a parent node.
- A `client` node will either run its own configured checks or receive command execution events from the parent node.

The following sections will refer to these roles and explain the differences and the possibilities this kind of setup offers.

Tip: If you just want to install a single master node that monitors several hosts (i.e. Icinga 2 clients), continue reading – we'll start with simple examples. In case you are planning a huge cluster setup with multiple levels and lots of clients, read on – we'll deal with these cases later on.

The installation on each system is the same: You need to install the Icinga 2 package and the required plugins.

The required configuration steps are mostly happening on the command line. You can also automate the setup.

The first thing you need learn about a distributed setup is the hierarchy of the single components.

6.2 Zones

The Icinga 2 hierarchy consists of so-called zone objects. Zones depend on a parent-child relationship in order to trust each other.

Have a look at this example for the `satellite` zones which have the `master` zone as a parent zone:

```
object Zone "master" {
   //...
}

object Zone "satellite region 1" {
  parent = "master"
  //...
}

object Zone "satellite region 2" {
  parent = "master"
  //...
}
```

There are certain limitations for child zones, e.g. their members are not allowed to send configuration commands to the parent zone members. Vice versa, the trust hierarchy allows for example the `master` zone to send configuration files to the `satellite` zone. Read more about this in the security section.

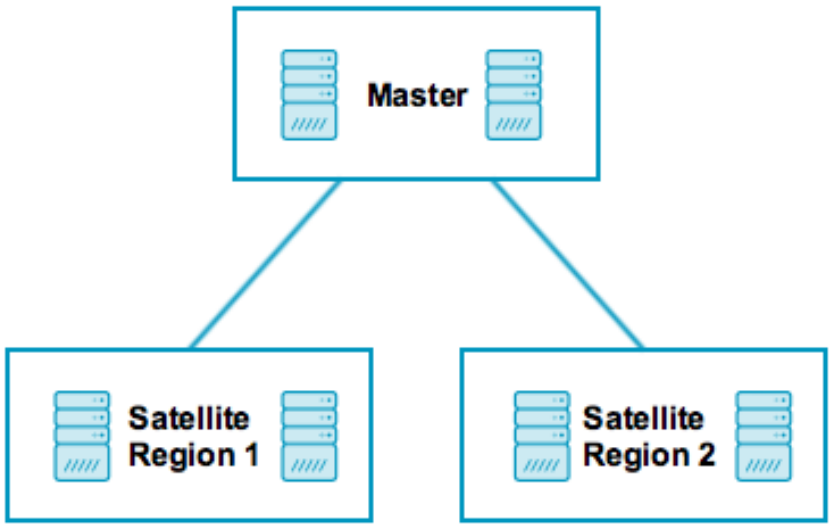


Figure 6: Icinga 2 Distributed Zones

`client` nodes also have their own unique zone. By convention you can use the FQDN for the zone name.

6.3 Endpoints

Nodes which are a member of a zone are so-called Endpoint objects.

Here is an example configuration for two endpoints in different zones:

```
object Endpoint "icinga2-master1.localdomain" {
  host = "192.168.56.101"
}

object Endpoint "icinga2-satellite1.localdomain" {
  host = "192.168.56.105"
}

object Zone "master" {
  endpoints = [ "icinga2-master1.localdomain" ]
}

object Zone "satellite" {
  endpoints = [ "icinga2-satellite1.localdomain" ]
```

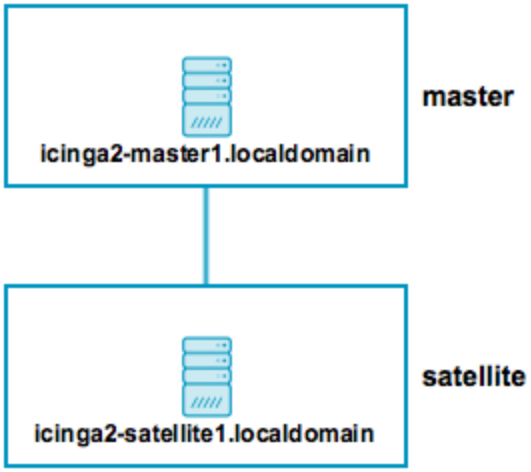


Figure 7: Icinga 2 Distributed Endpoints

```
  parent = "master"
}
```

All endpoints in the same zone work as high-availability setup. For example, if you have two nodes in the `master` zone, they will load-balance the check execution.

Endpoint objects are important for specifying the connection information, e.g. if the master should actively try to connect to a client.

The zone membership is defined inside the `Zone` object definition using the `endpoints` attribute with an array of `Endpoint` names.

If you want to check the availability (e.g. ping checks) of the node you still need a Host object.

6.4 ApiListener

In case you are using the CLI commands later, you don't have to write this configuration from scratch in a text editor. The ApiListener object is used to load the SSL certificates and specify restrictions, e.g. for accepting configuration commands.

It is also used for the Icinga 2 REST API which shares the same host and port with the Icinga 2 Cluster protocol.

The object configuration is stored in the `/etc/icinga2/features-enabled/api.conf` file. Depending on the configuration mode the attributes `accept_commands` and `accept_config` can be configured here.

In order to use the `api` feature you need to enable it and restart Icinga 2.

```
icinga2 feature enable api
```

6.5 Conventions

By convention all nodes should be configured using their FQDN.

Furthermore, you must ensure that the following names are exactly the same in all configuration files:

- Host certificate common name (CN).
- Endpoint configuration object for the host.
- NodeName constant for the local host.

Setting this up on the command line will help you to minimize the effort. Just keep in mind that you need to use the FQDN for endpoints and for common names when asked.

6.6 Security

While there are certain mechanisms to ensure a secure communication between all nodes (firewalls, policies, software hardening, etc.), Icinga 2 also provides additional security:

- SSL certificates are mandatory for communication between nodes. The CLI commands help you create those certificates.
- Child zones only receive updates (check results, commands, etc.) for their configured objects.
- Child zones are not allowed to push configuration updates to parent zones.
- Zones cannot interfere with other zones and influence each other. Each checkable host or service object is assigned to **one zone** only.
- All nodes in a zone trust each other.
- Config sync and remote command endpoint execution is disabled by default.

The underlying protocol uses JSON-RPC event notifications exchanged by nodes. The connection is secured by TLS. The message protocol uses an internal API, and as such message types and names may change internally and are not documented.

Zones build the trust relationship in a distributed environment. If you do not specify a zone for a client and specify the parent zone, its zone members e.g. the master instance won't trust the client.

Building this trust is key in your distributed environment. That way the parent node knows that it is able to send messages to the child zone, e.g. configuration objects, configuration in global zones, commands to be executed in this zone/for this endpoint. It also receives check results from the child zone for checkable objects (host/service).

Vice versa, the client trusts the master and accepts configuration and commands if enabled in the api feature. If the client would send configuration to the parent zone, the parent nodes will deny it. The parent zone is the configuration entity, and does not trust clients in this matter. A client could attempt to modify a different client for example, or inject a check command with malicious code.

While it may sound complicated for client setups, it removes the problem with different roles and configurations for a master and a client. Both of them work the same way, are configured in the same way (Zone, Endpoint, ApiListener), and you can troubleshoot and debug them in just one go.

6.7 Master Setup

This section explains how to install a central single master node using the `node wizard` command. If you prefer to do an automated installation, please refer to the automated setup section.

Install the Icinga 2 package and setup the required plugins if you haven't done so already.

Note: Windows is not supported for a master node setup.

The next step is to run the `node wizard` CLI command. Prior to that ensure to collect the required information:

Parameter	Description
Common name (CN)	**Required.** By convention this should be the host's FQDN. Defaults to the FQDN.
Global zones	**Optional.** Allows to specify more global zones in addition to `global-templates` and `director-global`. Defaults to `n`.

Parameter	Description
API bind host	**Optional.** Allows to specify the address the ApiListener is bound to. For advanced usage only.
API bind port	**Optional.** Allows to specify the port the ApiListener is bound to. For advanced usage only (requires changing the default port 5665 everywhere).

The setup wizard will ensure that the following steps are taken:

- Enable the `api` feature.
- Generate a new certificate authority (CA) in `/var/lib/icinga2/ca` if it doesn't exist.
- Create a certificate for this node signed by the CA key.
- Update the zones.conf file with the new zone hierarchy.
- Update the ApiListener and constants configuration.

Here is an example of a master setup for the `icinga2-master1.localdomain` node on CentOS 7:

```
[root@icinga2-master1.localdomain /]# icinga2 node wizard

Welcome to the Icinga 2 Setup Wizard!

We will guide you through all required configuration details.

Please specify if this is a satellite/client setup ('n' installs a master setup) [Y/n]: n

Starting the Master setup routine...

Please specify the common name (CN) [icinga2-master1.localdomain]: icinga2-master1.localdomain
Reconfiguring Icinga...
Checking for existing certificates for common name 'icinga2-master1.localdomain'...
Certificates not yet generated. Running 'api setup' now.
Generating master configuration for Icinga 2.
Enabling feature api. Make sure to restart Icinga 2 for these changes to take effect.

Do you want to specify additional global zones? [y/N]: N
Please specify the API bind host/port (optional):
```

```
Bind Host []:
Bind Port []:

Done.

Now restart your Icinga 2 daemon to finish the installation!
```

You can verify that the CA public and private keys are stored in the `/var/lib/icinga2/ca` directory. Keep this path secure and include it in your backups.

In case you lose the CA private key you have to generate a new CA for signing new client certificate requests. You then have to also re-create new signed certificates for all existing nodes.

Once the master setup is complete, you can also use this node as primary CSR auto-signing master. The following section will explain how to use the CLI commands in order to fetch their signed certificate from this master node.

6.8 Signing Certificates on the Master

All certificates must be signed by the same certificate authority (CA). This ensures that all nodes trust each other in a distributed monitoring environment.

This CA is generated during the master setup and should be the same on all master instances.

You can avoid signing and deploying certificates manually by using built-in methods for auto-signing certificate signing requests (CSR):

- CSR Auto-Signing which uses a client ticket generated on the master as trust identifier.
- On-Demand CSR Signing which allows to sign pending certificate requests on the master.

Both methods are described in detail below.

Note

On-Demand CSR Signing is available in Icinga 2 v2.8+.

6.8.1 CSR Auto-Signing

A client which sends a certificate signing request (CSR) must authenticate itself in a trusted way. The master generates a client ticket which is included in this request. That way the master can verify that the request matches the previously trusted ticket and sign the request.

Note

Icinga 2 v2.8 adds the possibility to forward signing requests on a satellite to the master node. This helps with the setup of three level clusters and more.

Advantages:

- Nodes can be installed by different users who have received the client ticket.
- No manual interaction necessary on the master node.
- Automation tools like Puppet, Ansible, etc. can retrieve the pre-generated ticket in their client catalog and run the node setup directly.

Disadvantages:

- Tickets need to be generated on the master and copied to client setup wizards.
- No central signing management.

Setup wizards for satellite/client nodes will ask you for this specific client ticket.

There are two possible ways to retrieve the ticket:

- CLI command executed on the master node.
- REST API request against the master node.

Required information:

Parameter	Description
Common name (CN)	**Required.** The common name for the satellite/client. By convention this should be the FQDN.

The following example shows how to generate a ticket on the master node `icinga2-master1.localdomain` for the client `icinga2-client1.localdomain`:

```
[root@icinga2-master1.localdomain /]# icinga2 pki ticket --cn icinga2-client1.localdomain
```

Querying the Icinga 2 API on the master requires an ApiUser object with at least the `actions/generate-ticket` permission.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/conf.d/api-users.conf

object ApiUser "client-pki-ticket" {
  password = "bea11beb7b810ea9ce6ea" //change this
  permissions = [ "actions/generate-ticket" ]
```

```
}
```

```
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Retrieve the ticket on the master node `icinga2-master1.localdomain` with `curl`, for example:

```
[root@icinga2-master1.localdomain /]# curl -k -s -u client-pki-ticket:bea11beb7b810ea9ce6ea -
-X POST 'https://icinga2-master1.localdomain:5665/v1/actions/generate-ticket' -d '{ "cn": "ic
```

Store that ticket number for the satellite/client setup below.

Note: Never expose the ticket salt and/or ApiUser credentials to your client nodes. Example: Retrieve the ticket on the Puppet master node and send the compiled catalog to the authorized Puppet agent node which will invoke the automated setup steps.

6.8.2 On-Demand CSR Signing

Icinga 2 v2.8 adds the possibility to sign certificates from clients without requiring a client ticket for auto-signing.

Instead, the client sends a certificate signing request to specified parent node. This could either be directly the master, or a satellite which forwards the request to the signing master.

Advantages:

- Central certificate request signing management.
- No pre-generated ticket is required for client setups.

Disadvantages:

- Asynchronous step for automated deployments.
- Needs client verification on the master.

You can list certificate requests by using the `ca list` CLI command. This also shows which requests already have been signed.

```
[root@icinga2-master1.localdomain /]# icinga2 ca list
Fingerprint                                          | Timestamp        | Signed | Subject
-----------------------------------------------------------------------|---------------------|----
403da5b228df384f07f980f45ba50202529cded7c8182abf96740660caa09727 | 2017/09/06 17:02:40 | *
71700c28445109416dd7102038962ac3fd421fbb349a6e7303b6033ec1772850 | 2017/09/06 17:20:02 |
```

Tip: Add `--json` to the CLI command to retrieve the details in JSON format.

If you want to sign a specific request, you need to use the `ca sign` CLI command and pass its fingerprint as argument.

```
[root@icinga2-master1.localdomain /]# icinga2 ca sign 71700c28445109416dd7102038962ac3fd421fb
information/cli: Signed certificate for 'CN = icinga2-client2.localdomain'.
```

6.9 Client/Satellite Setup

This section describes the setup of a satellite and/or client connected to an existing master node setup. If you haven't done so already, please run the master setup.

Icinga 2 on the master node must be running and accepting connections on port `5665`.

6.9.1 Client/Satellite Setup on Linux

Please ensure that you've run all the steps mentioned in the client/satellite section.

Install the Icinga 2 package and setup the required plugins if you haven't done so already.

The next step is to run the `node wizard` CLI command.

In this example we're generating a ticket on the master node `icinga2-master1.localdomain` for the client `icinga2-client1.localdomain`:

```
[root@icinga2-master1.localdomain /]# icinga2 pki ticket --cn icinga2-client1.localdomain
4f75d2ecd253575fe9180938ebff7cbca262f96e
```

Note: You don't need this step if you have chosen to use On-Demand CSR Signing.

Start the wizard on the client `icinga2-client1.localdomain`:

```
[root@icinga2-client1.localdomain /]# icinga2 node wizard

Welcome to the Icinga 2 Setup Wizard!

We will guide you through all required configuration details.
```

Press `Enter` or add `y` to start a satellite or client setup.

```
Please specify if this is a satellite/client setup ('n' installs a master setup) [Y/n]:
```

Press `Enter` to use the proposed name in brackets, or add a specific common name (CN). By convention this should be the FQDN.

```
Starting the Client/Satellite setup routine...

Please specify the common name (CN) [icinga2-client1.localdomain]: icinga2-client1.localdomain
```

Specify the direct parent for this node. This could be your primary master `icinga2-master1.localdomain` or a satellite node in a multi level cluster scenario.

```
Please specify the parent endpoint(s) (master or satellite) where this node should connect to:
Master/Satellite Common Name (CN from your master/satellite node): icinga2-master1.localdomain
```

Press `Enter` or choose `y` to establish a connection to the parent node.

```
Do you want to establish a connection to the parent node from this node? [Y/n]:
```

> **Note:**
>
> If this node cannot connect to the parent node, choose `n`. The setup wizard will provide instructions for this scenario – signing questions are disabled then.

Add the connection details for `icinga2-master1.localdomain`.

```
Please specify the master/satellite connection information:
Master/Satellite endpoint host (IP address or FQDN): 192.168.56.101
Master/Satellite endpoint port [5665]: 5665
```

You can add more parent nodes if necessary. Press `Enter` or choose `n` if you don't want to add any. This comes in handy if you have more than one parent node, e.g. two masters or two satellites.

```
Add more master/satellite endpoints? [y/N]:
```

Verify the parent node's certificate:

```
Parent certificate information:

 Subject:     CN = icinga2-master1.localdomain
 Issuer:      CN = Icinga CA
 Valid From:  Sep  7 13:41:24 2017 GMT
 Valid Until: Sep  3 13:41:24 2032 GMT
 Fingerprint: AC 99 8B 2B 3D B0 01 00 E5 21 FA 05 2E EC D5 A9 EF 9E AA E3

Is this information correct? [y/N]: y
```

The setup wizard fetches the parent node's certificate and ask you to verify this information. This is to prevent MITM attacks or any kind of untrusted parent relationship.

Note: The certificate is not fetched if you have chosen not to connect to the parent node.

Proceed with adding the optional client ticket for CSR auto-signing:

```
Please specify the request ticket generated on your Icinga 2 master (optional).
 (Hint: # icinga2 pki ticket --cn 'icinga2-client1.localdomain'):
4f75d2ecd253575fe9180938ebff7cbca262f96e
```

In case you've chosen to use On-Demand CSR Signing you can leave the ticket question blank.

Instead, Icinga 2 tells you to approve the request later on the master node.

```
No ticket was specified. Please approve the certificate signing request manually
on the master (see 'icinga2 ca list' and 'icinga2 ca sign --help' for details).
```

You can optionally specify a different bind host and/or port.

```
Please specify the API bind host/port (optional):
Bind Host []:
Bind Port []:
```

The next step asks you to accept configuration (required for config sync mode) and commands (required for command endpoint mode).

```
Accept config from parent node? [y/N]: y
Accept commands from parent node? [y/N]: y
```

You can add more global zones in addition to `global-templates` and `director-global` if necessary. Press `Enter` or choose `n`, if you don't want to add any additional.

```
Reconfiguring Icinga...

Do you want to specify additional global zones? [y/N]: N
```

The wizard proceeds and you are good to go.

```
Done.

Now restart your Icinga 2 daemon to finish the installation!
```

Note

If you have chosen not to connect to the parent node, you cannot start Icinga 2 yet. The wizard asked you to manually copy the master's public CA certificate file into `/var/lib/icinga2/certs/ca.crt`.

You need to manually sign the CSR on the master node.

Restart Icinga 2 as requested.

```
[root@icinga2-client1.localdomain /]# systemctl restart icinga2
```

Here is an overview of all parameters in detail:

Parameter	Description
Common name (CN)	**Required.** By convention this should be the host's FQDN. Defaults to the FQDN.

Parameter	Description
Master common name	**Required.** Use the common name you've specified for your master node before.
Establish connection to the parent node	**Optional.** Whether the node should attempt to connect to the parent node or not. Defaults to `y`.
Master/Satellite endpoint host	**Required if the the client needs to connect to the master/satellite.** The parent endpoint's IP address or FQDN. This information is included in the `Endpoint` object configuration in the `zones.conf` file.
Master/Satellite endpoint port	**Optional if the the client needs to connect to the master/satellite.** The parent endpoints's listening port. This information is included in the `Endpoint` object configuration.
Add more master/satellite endpoints	**Optional.** If you have multiple master/satellite nodes configured, add them here.
Parent Certificate information	**Required.** Verify that the connecting host really is the requested master node.

Parameter	Description
Request ticket	**Optional.** Add the ticket generated on the master.
API bind host	**Optional.** Allows to specify the address the ApiListener is bound to. For advanced usage only.
API bind port	**Optional.** Allows to specify the port the ApiListener is bound to. For advanced usage only (requires changing the default port 5665 everywhere).
Accept config	**Optional.** Whether this node accepts configuration sync from the master node (required for config sync mode). For security reasons this defaults to `n`.
Accept commands	**Optional.** Whether this node accepts command execution messages from the master node (required for command endpoint mode). For security reasons this defaults to `n`.
Global zones	**Optional.** Allows to specify more global zones in addition to `global-templates` and `director-global`. Defaults to `n`.

The setup wizard will ensure that the following steps are taken:

- Enable the `api` feature.
- Create a certificate signing request (CSR) for the local node.
- Request a signed certificate i(optional with the provided ticket number) on the master node.
- Allow to verify the parent node's certificate.
- Store the signed client certificate and ca.crt in `/var/lib/icinga2/certs`.
- Update the `zones.conf` file with the new zone hierarchy.
- Update `/etc/icinga2/features-enabled/api.conf` (`accept_config`, `accept_commands`) and `constants.conf`.

You can verify that the certificate files are stored in the `/var/lib/icinga2/certs` directory.

Note

The certificate location changed in v2.8 to `/var/lib/icinga2/certs`. Please read the upgrading chapter for more details.

Note

If the client is not directly connected to the certificate signing master, signing requests and responses might need some minutes to fully update the client certificates.

If you have chosen to use On-Demand CSR Signing certificates need to be signed on the master first. Ticket-less setups require at least Icinga 2 v2.8+ on all involved instances.

Now that you've successfully installed a Linux/Unix satellite/client instance, please proceed to the configuration modes.

6.9.2 Client Setup on Windows

Download the MSI-Installer package from https://packages.icinga.com/windows/.

Requirements:

- Windows Vista/Server 2008 or higher
- Versions older than Windows 10/Server 2016 require the Universal C Runtime for Windows
- Microsoft .NET Framework 2.0 for the setup wizard

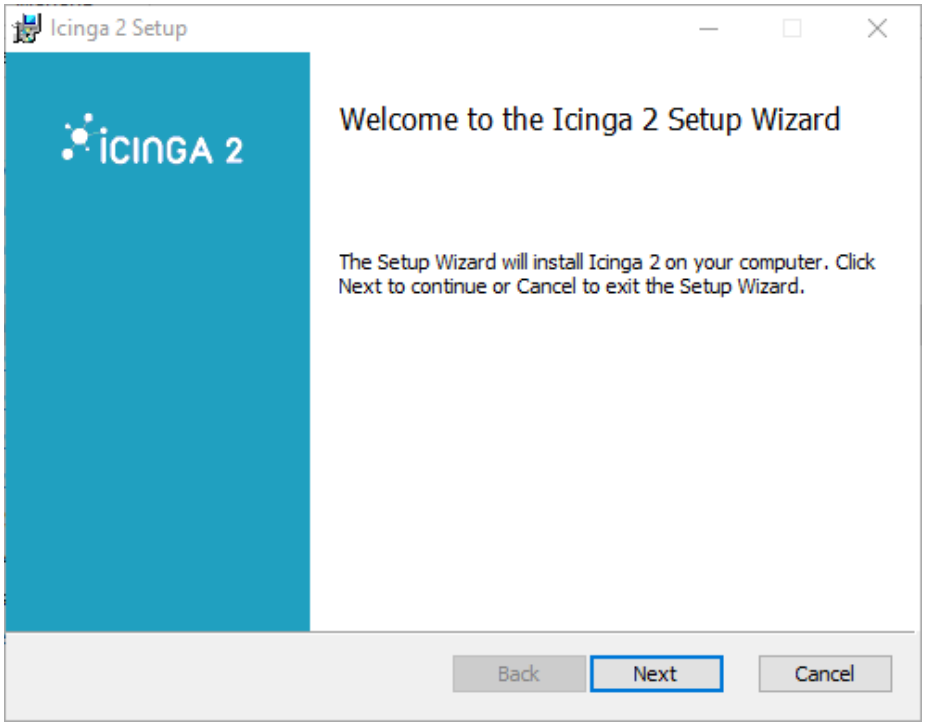
The installer package includes the NSClient++ package so that Icinga 2 can use its built-in plugins. You can find more details in this chapter. The Windows package also installs native monitoring plugin binaries to get you started more easily.

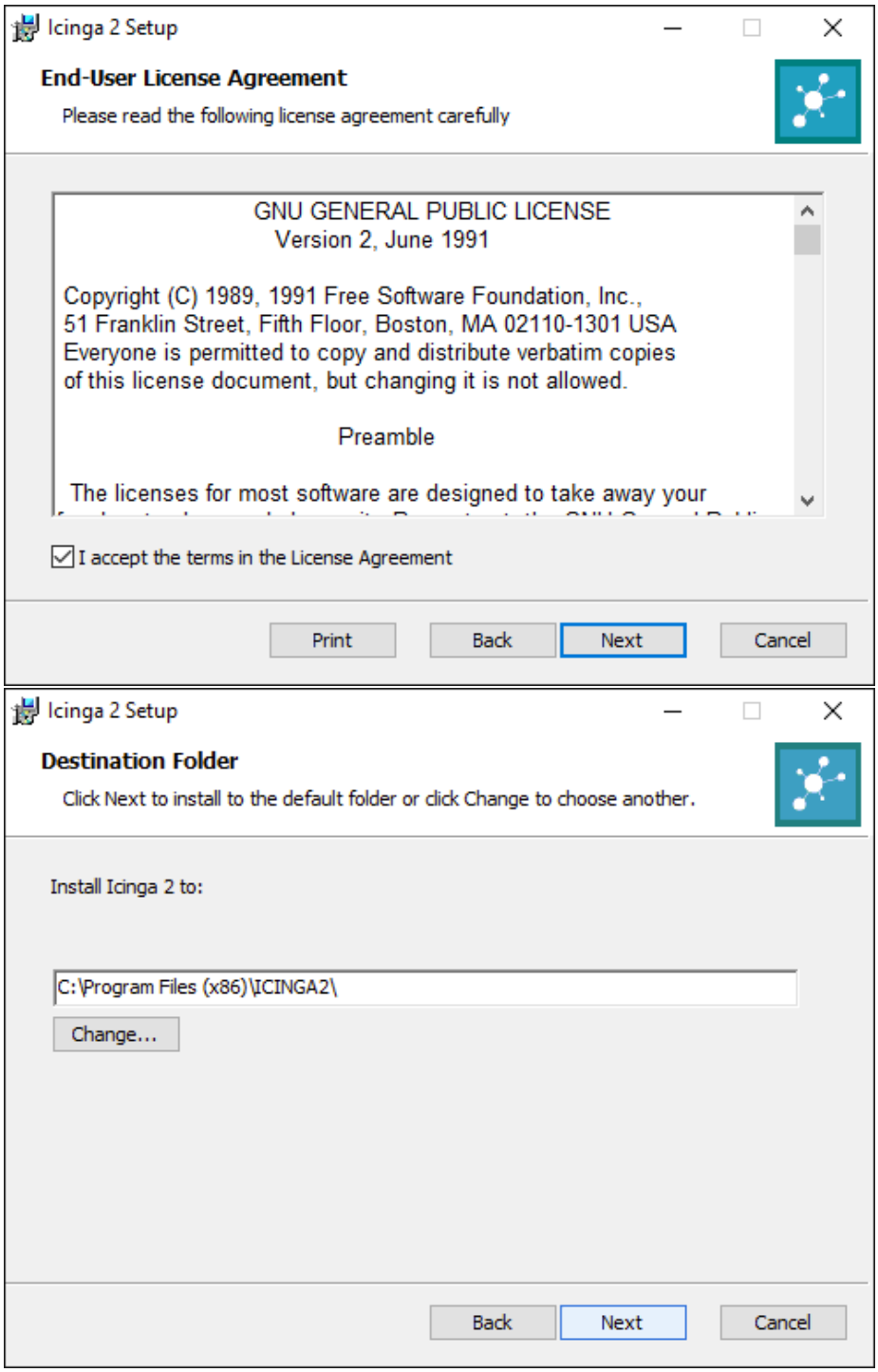
Note

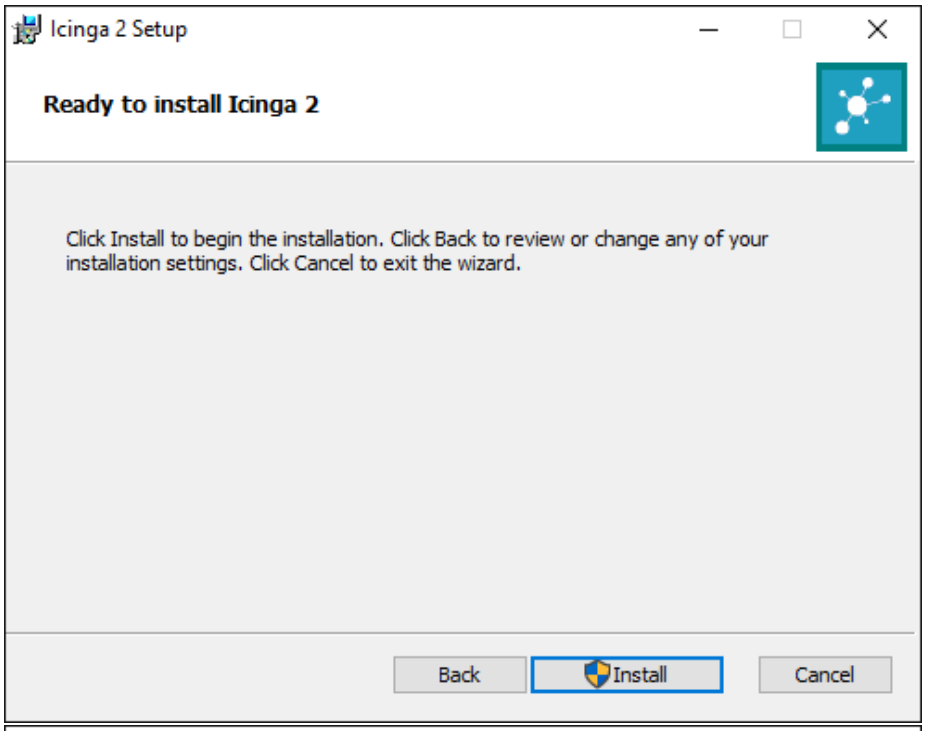
Please note that Icinga 2 was designed to run as light-weight client on Windows. There is no support for satellite instances.

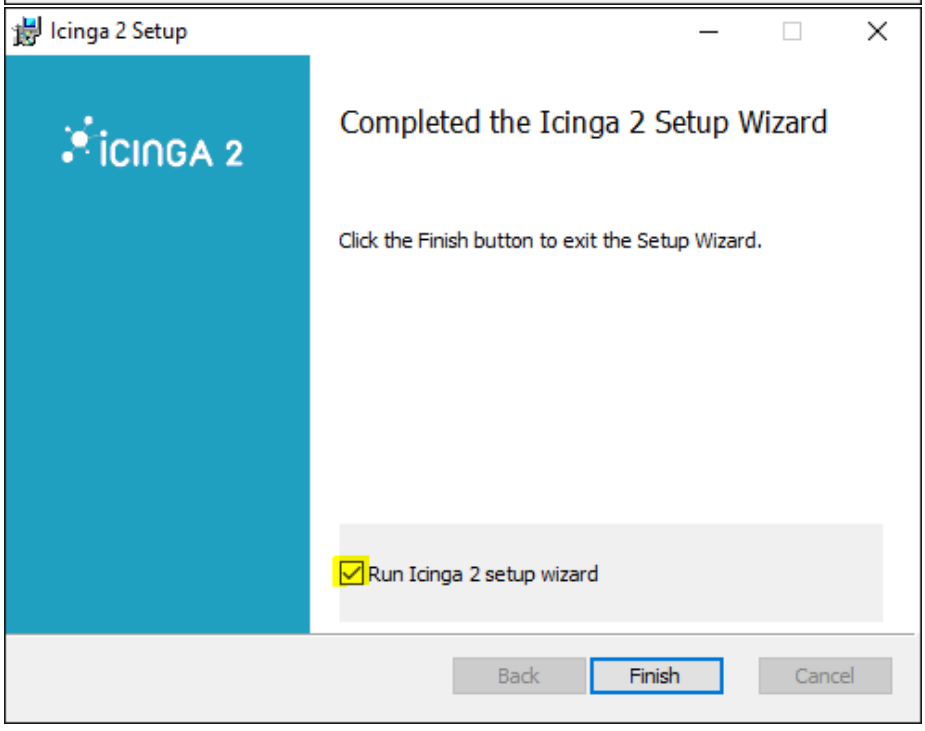
6.9.2.1 Windows Client Setup Start

Run the MSI-Installer package and follow the instructions shown in the screenshots.




Icinga 2 Setup
End-User License Agreement
Please read the following license agreement carefully
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your
I accept the terms in the License Agreement
Print
Back
Next
Cancel
Icinga 2 Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another.
Install Icinga 2 to:
C:\Program Files (x86)\ICINGA2\
Change...
Back
Next
Cancel


Icinga 2 Setup
Ready to install Icinga 2
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


Icinga 2 Setup
ICINGA 2
Completed the Icinga 2 Setup Wizard
Click the Finish button to exit the Setup Wizard.
Run Icinga 2 setup wizard
Back
Finish
Cancel

The graphical installer offers to run the Icinga 2 setup wizard after the installation. Select the check box to proceed.

Tip

You can also run the Icinga 2 setup wizard from the Start menu later.

On a fresh installation the setup wizard guides you through the initial configuration. It also provides a mechanism to send a certificate request to the CSR signing master.

The following configuration details are required:

Parameter	Description
Instance name	**Required.** By convention this should be the host's FQDN. Defaults to the FQDN.
Setup ticket	**Optional.** Paste the previously generated ticket number. If left blank, the certificate request must be signed on the master node.

Fill in the required information and click `Add` to add a new master connection.

Add the following details:

Parameter	Description
Instance name	**Required.** The master/satellite endpoint name where this client is a direct child of.
Master/Satellite endpoint host	**Required.** The master or satellite's IP address or FQDN. This information is included in the `Endpoint` object configuration in the `zones.conf` file.
Master/Satellite endpoint port	**Optional.** The master or satellite's listening port. This information is included in the `Endpoint` object configuration.

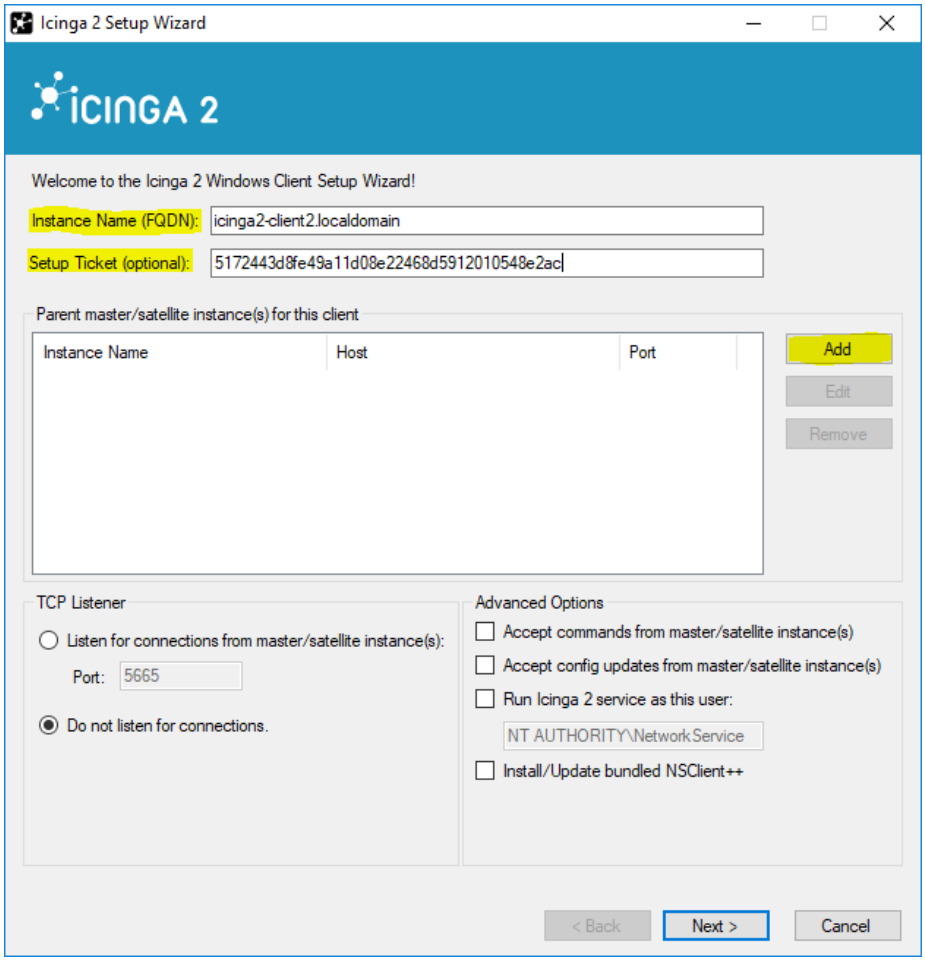


Figure 8: Icinga 2 Windows Setup

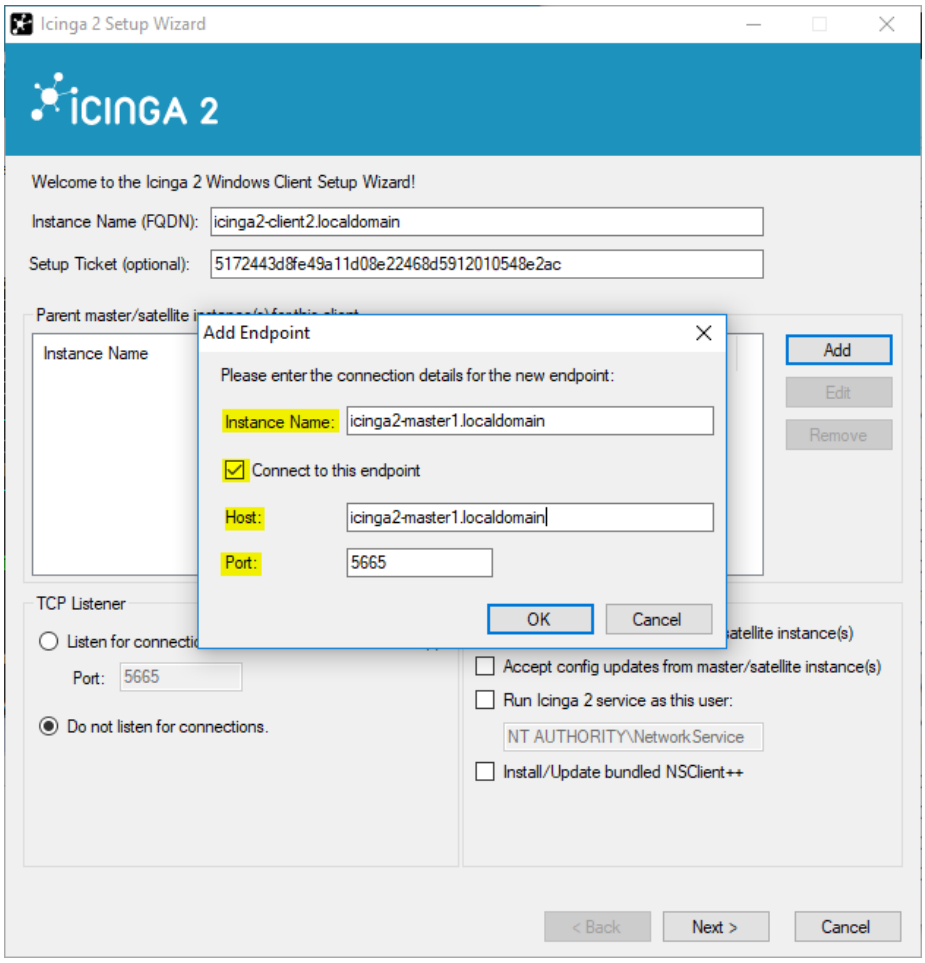


Figure 9: Icinga 2 Windows Setup

Optionally enable the following settings:

Parameter	Description
Accept config	**Optional.** Whether this node accepts configuration sync from the master node (required for config sync mode). For security reasons this is disabled by default.

Parameter	Description
Accept commands	**Optional.** Whether this node accepts command execution messages from the master node (required for command endpoint mode). For security reasons this is disabled by default.
Run Icinga 2 service as this user	**Optional.** Specify a different Windows user. This defaults to `NT AUTHORITY\Network Service` and is required for more privileged service checks.
Install NSClient++	**Optional.** The Windows installer bundles the NSClient++ installer for additional plugin checks.

Verify the certificate from the master/satellite instance where this node should connect to.

6.9.2.2 Bundled NSClient++ Setup

If you have chosen to install/update the NSClient++ package, the Icinga 2 setup wizard asks you to do so.

Choose the `Generic` setup.

Choose the `Custom` setup type.

NSClient++ does not install a sample configuration by default. Change this as shown in the screenshot.

Generate a secure password and enable the web server module. **Note**: The webserver module is available starting with NSClient++ 0.5.0. Icinga 2 v2.6+ is required which includes this version.

Finish the installation.

Open a web browser and navigate to `https://localhost:8443`. Enter the password you've configured during the setup. In case you lost it, look into the `C:\Program Files\NSClient++\nsclient.ini` configuration file.

The NSClient++ REST API can be used to query metrics. check_nscp_api uses this transport method.

6.9.2.3 Finish Windows Client Setup

Finish the Windows setup wizard.

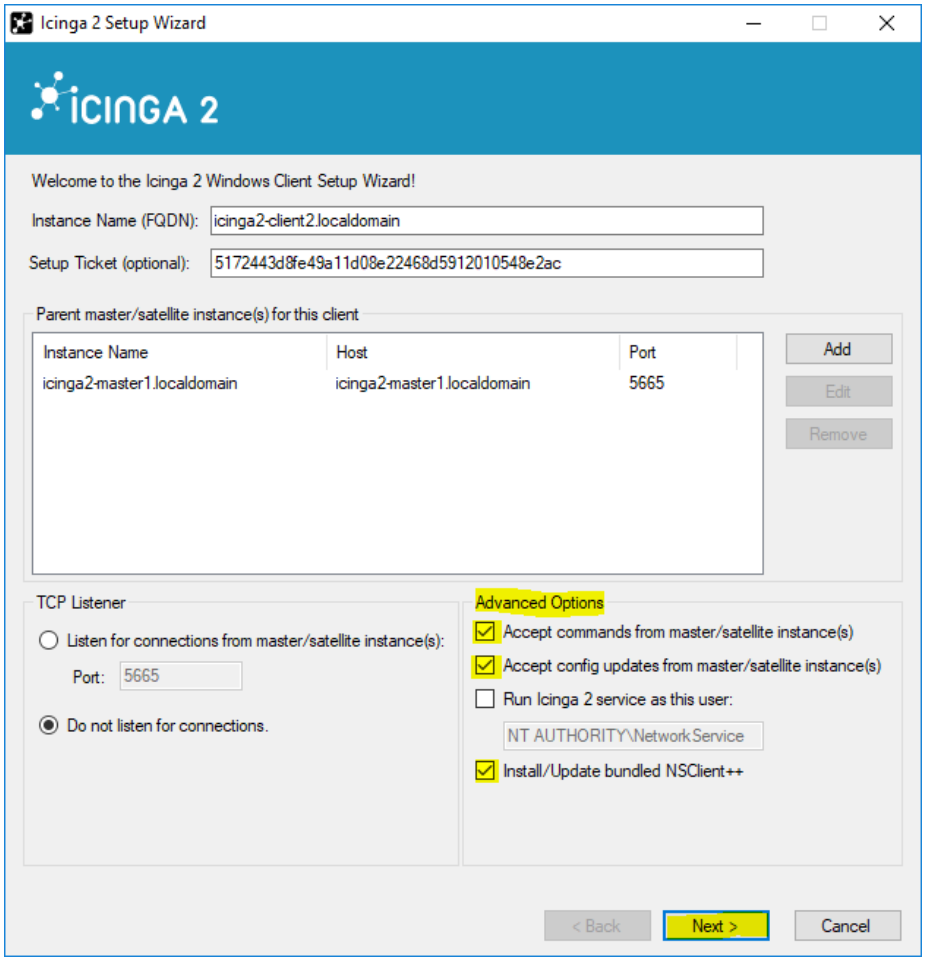


Figure 10: Icinga 2 Windows Setup

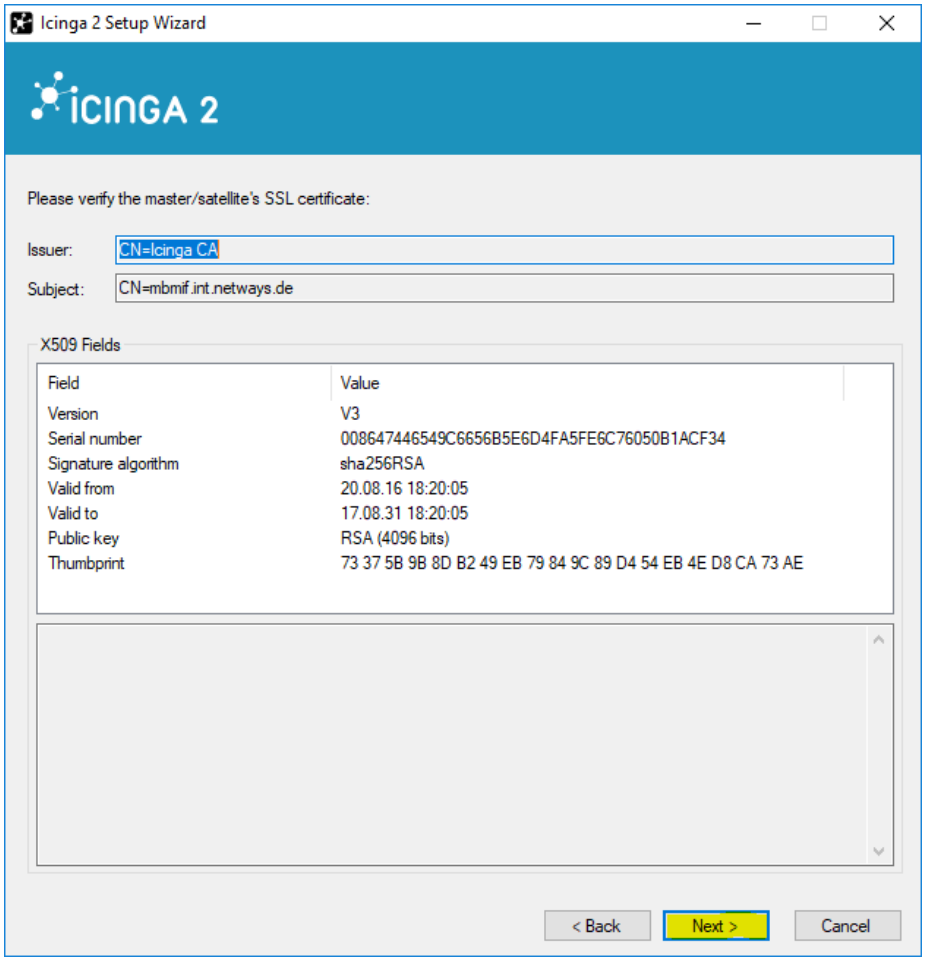


Figure 11: Icinga 2 Windows Setup

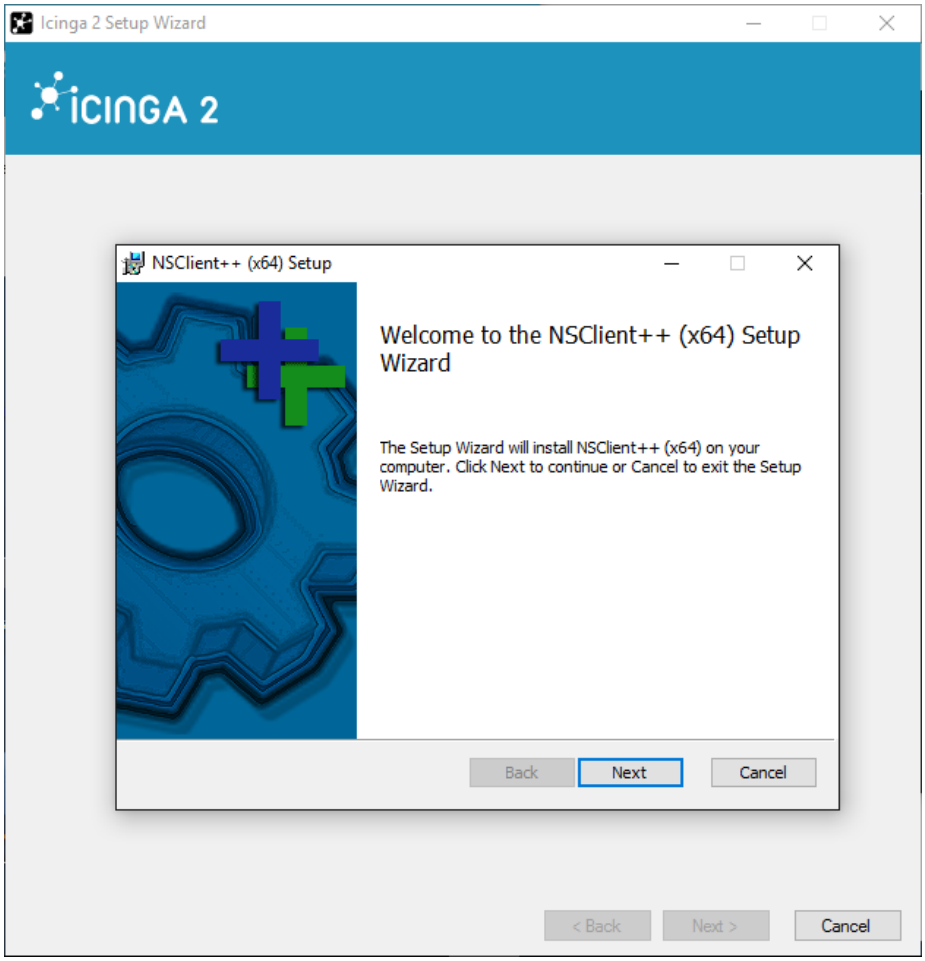


Figure 12: Icinga 2 Windows Setup NSClient++

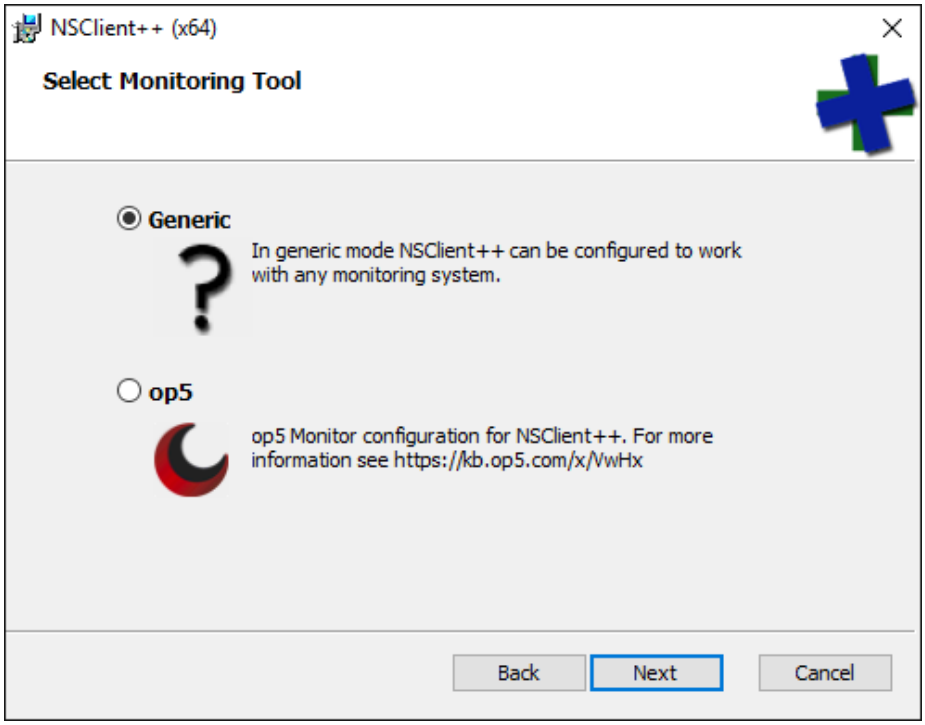


Figure 13: Icinga 2 Windows Setup NSClient++

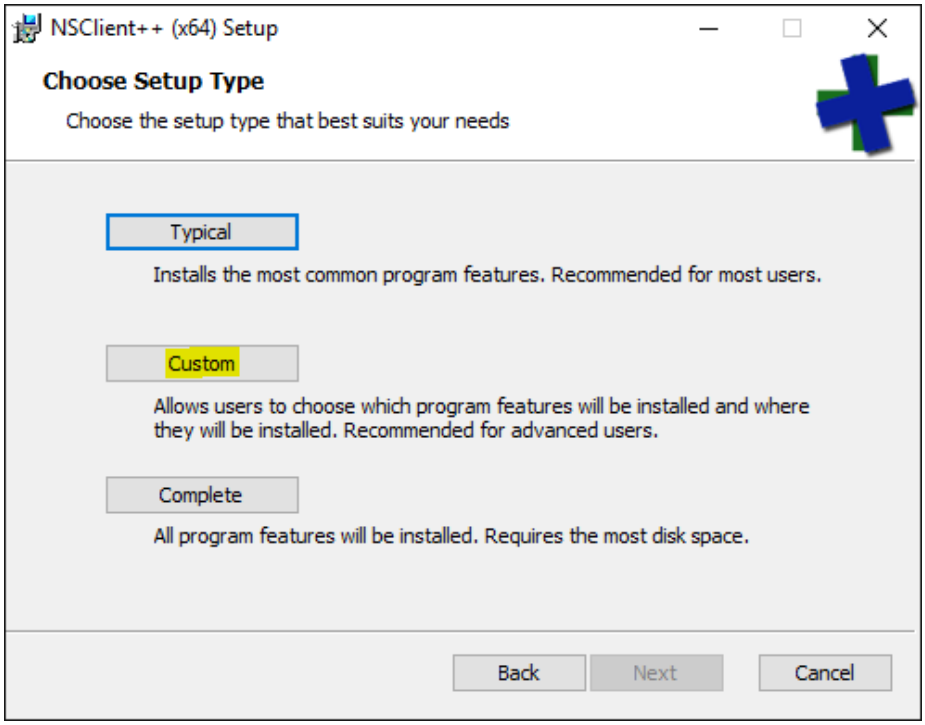


Figure 14: Icinga 2 Windows Setup NSClient++

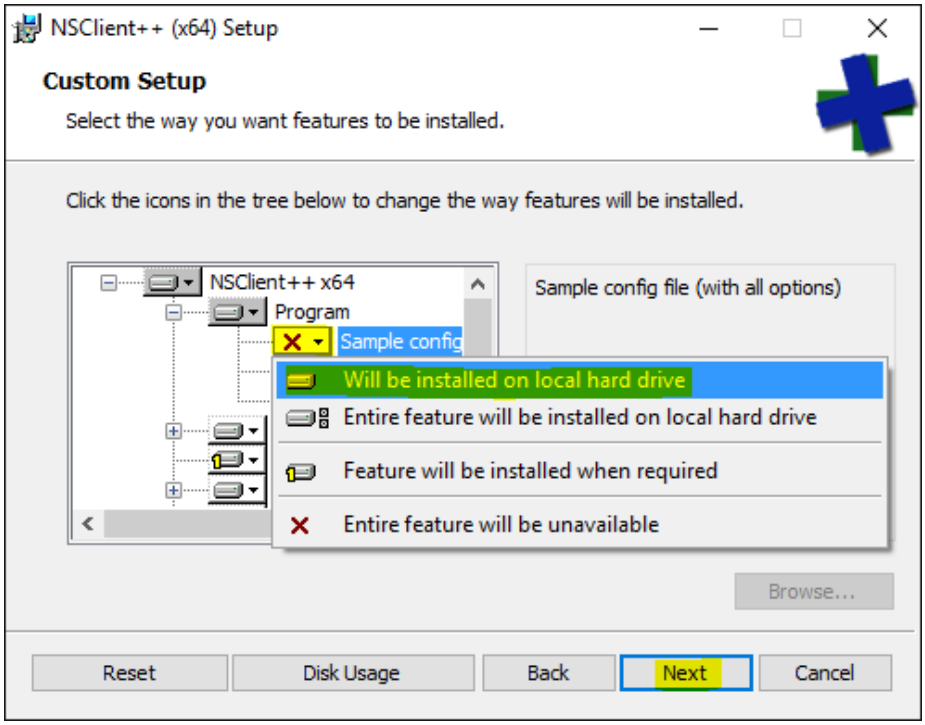


Figure 15: Icinga 2 Windows Setup NSClient++

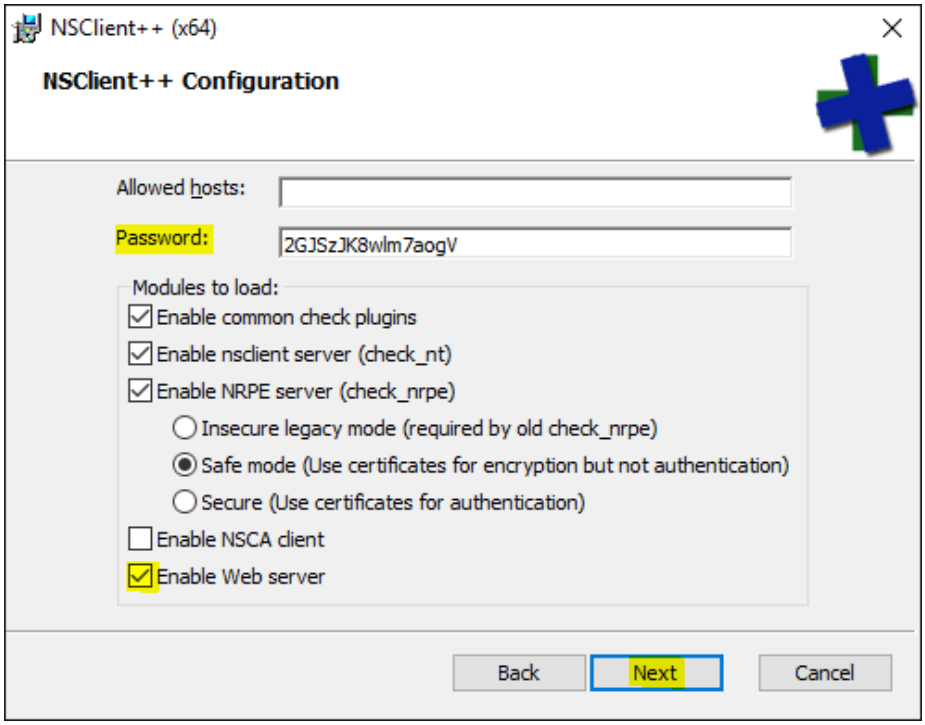


Figure 16: Icinga 2 Windows Setup NSClient++

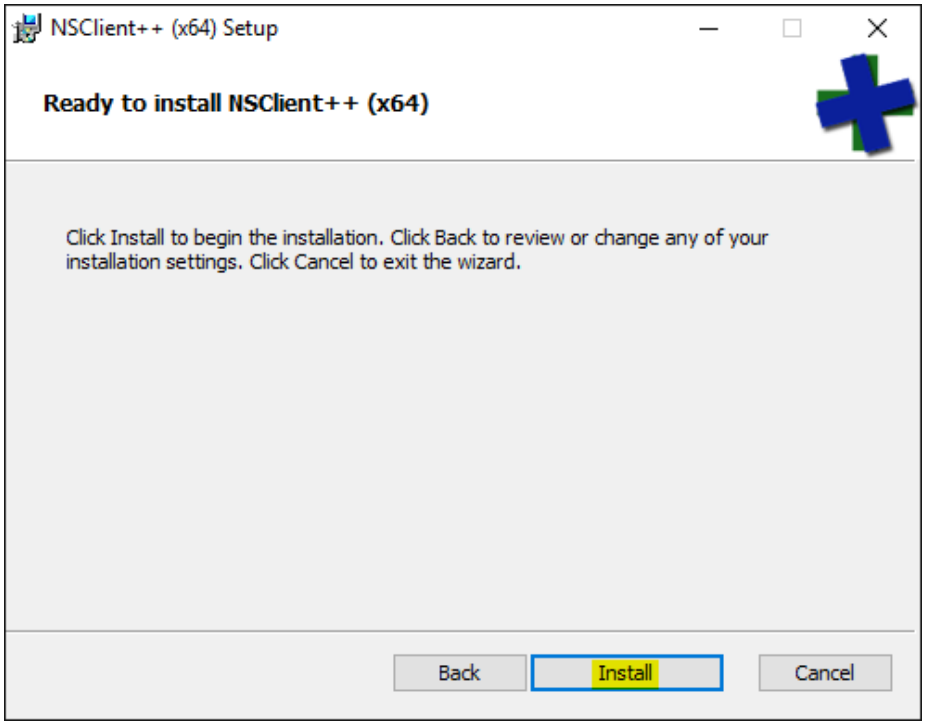


Figure 17: Icinga 2 Windows Setup NSClient++

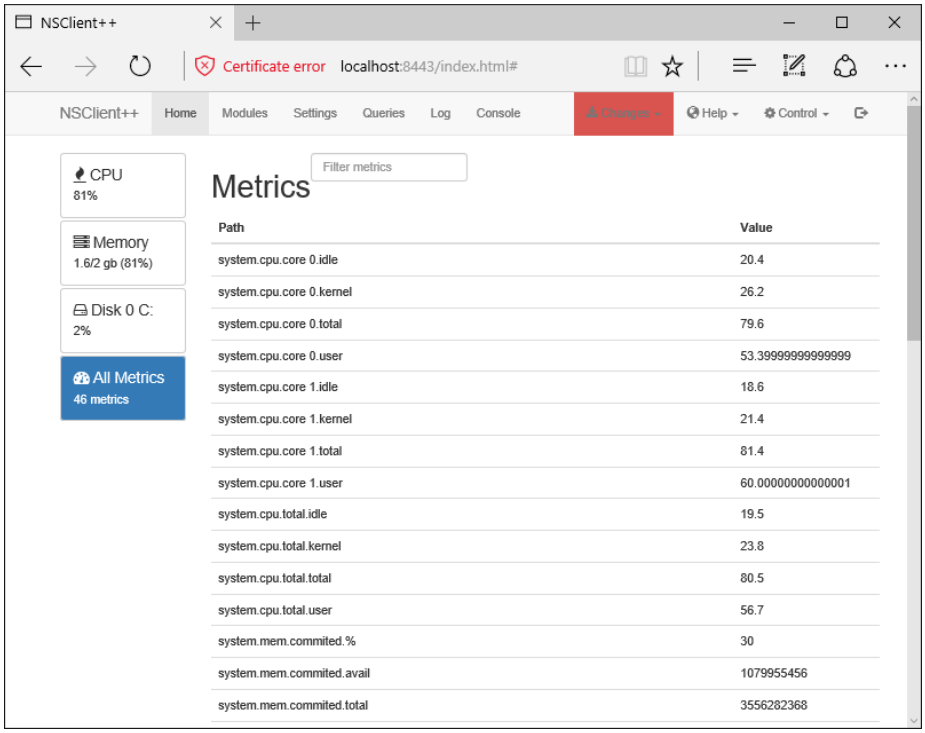


Figure 18: Icinga 2 Windows Setup NSClient++

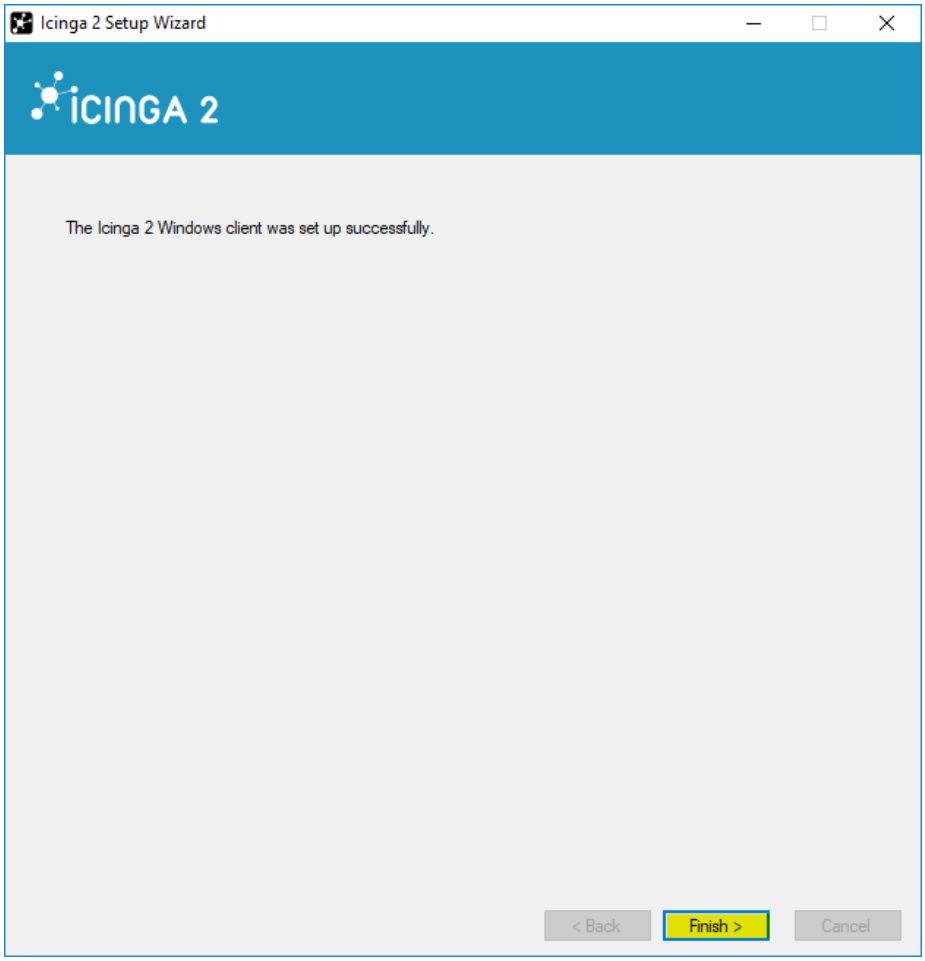


Figure 19: Icinga 2 Windows Setup

If you did not provide a setup ticket, you need to sign the certificate request on the master. The setup wizards tells you to do so. The Icinga 2 service is running at this point already and will automatically receive and update a signed client certificate.

Note

Ticket-less setups require at least Icinga 2 v2.8+ on all involved instances.

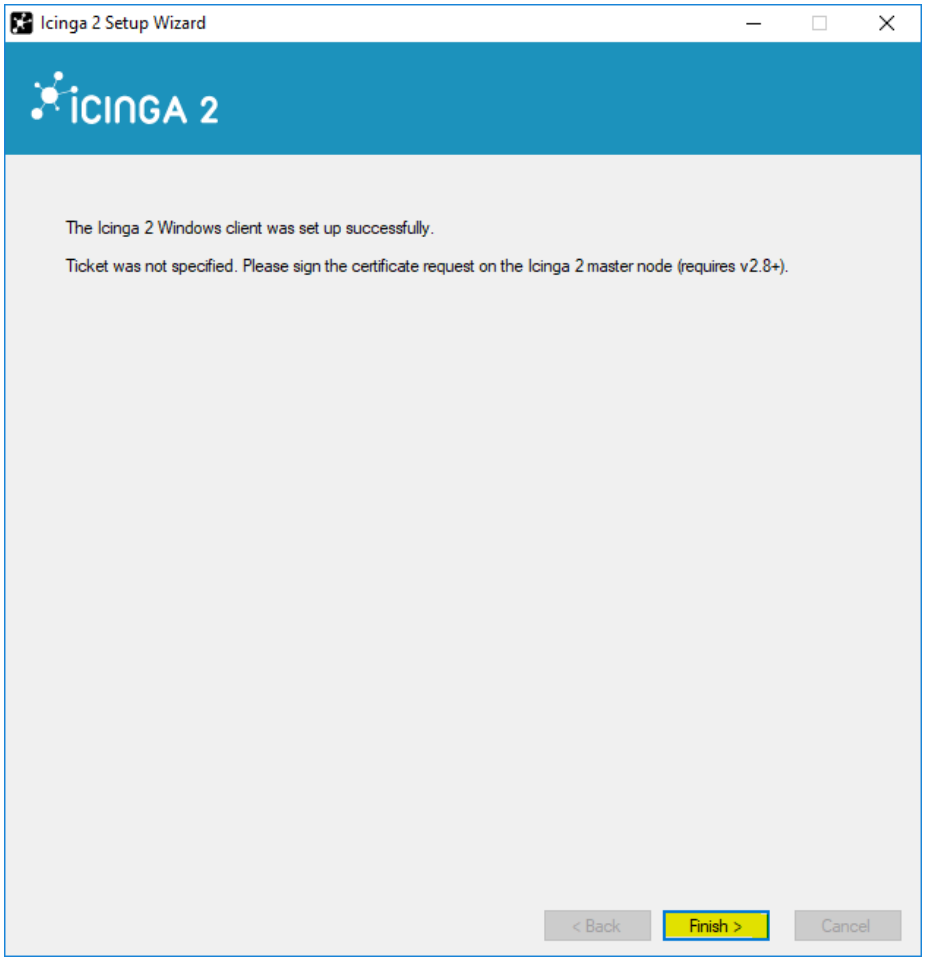


Figure 20: Icinga 2 Windows Setup

Icinga 2 is automatically started as a Windows service.

The Icinga 2 configuration is stored inside the `C:\ProgramData\icinga2` directory. Click `Examine Config` in the setup wizard to open a new Explorer

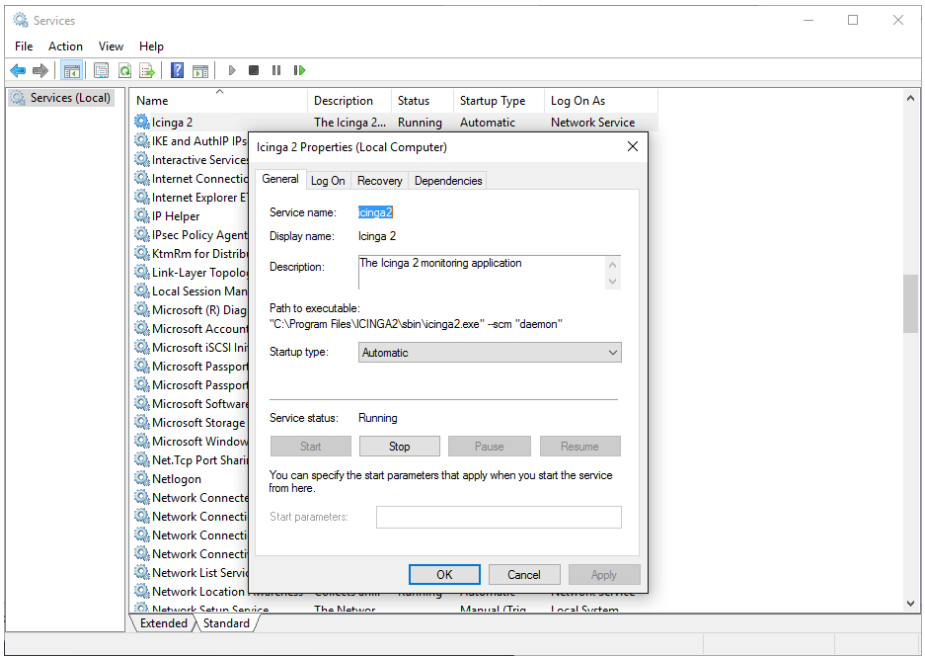


Figure 21: Icinga 2 Windows Setup

window.

The configuration files can be modified with your favorite editor e.g. Notepad.

In order to use the top down client configuration prepare the following steps.

Add a global zone for syncing check commands later. Navigate to `C:\ProgramData\icinga2\etc\icinga2` and open the `zones.conf` file in your preferred editor. Add the following lines if not existing already:

```
object Zone "global-templates" {
  global = true
}
```

Note: Packages >= 2.8 provide this configuration by default.

You don't need any local configuration on the client except for CheckCommand definitions which can be synced using the global zone above. Therefore disable the inclusion of the `conf.d` directory in the `icinga2.conf` file. Navigate to `C:\ProgramData\icinga2\etc\icinga2` and open the `icinga2.conf` file in your preferred editor. Remove or comment (`//`) the following line:

```
// Commented out, not required on a client with top down mode
//include_recursive "conf.d"
```

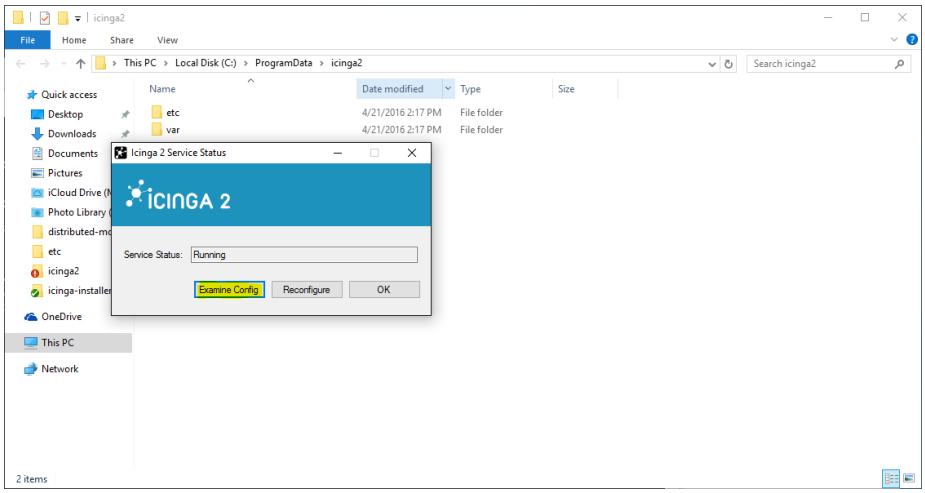


Figure 22: Icinga 2 Windows Setup

Validate the configuration on Windows open an administrator terminal and run the following command:

```
C:\WINDOWS\system32>cd "C:\Program Files\ICINGA2\sbin"
C:\Program Files\ICINGA2\sbin>icinga2.exe daemon -C
```

Note: You have to run this command in a shell with `administrator` privileges.

Now you need to restart the Icinga 2 service. Run `services.msc` from the start menu and restart the `icinga2` service. Alternatively, you can use the `net {start,stop}` CLI commands.

Now that you've successfully installed a Windows client, please proceed to the detailed configuration modes.

Note

The certificate location changed in v2.8 to `%ProgramData%\var\lib\icinga2\certs`. Please read the upgrading chapter for more details.

6.10 Configuration Modes

There are different ways to ensure that the Icinga 2 cluster nodes execute checks, send notifications, etc.

The preferred method is to configure monitoring objects on the master and distribute the configuration to satellites and clients.

The following chapters will explain this in detail with hands-on manual configuration examples. You should test and implement this once to fully understand

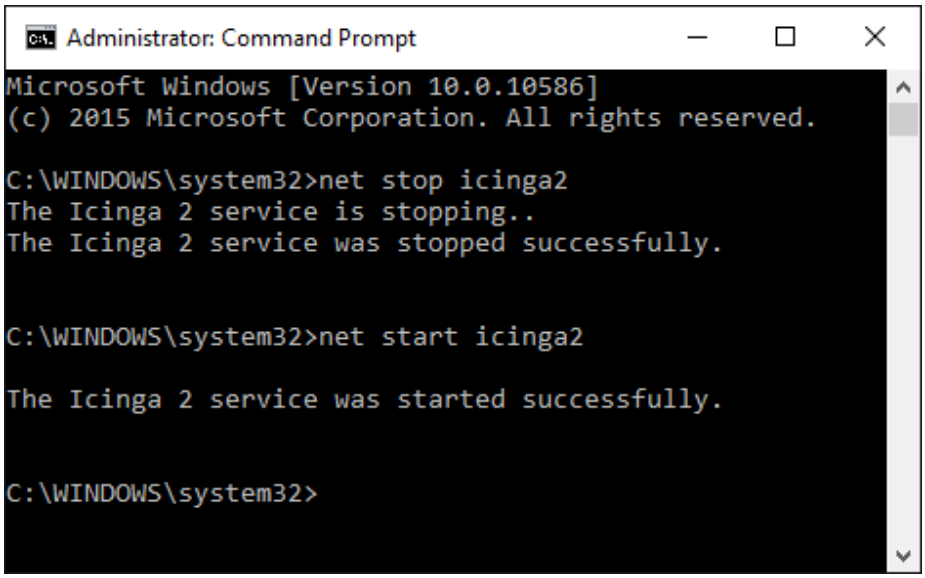


Figure 23: Icinga 2 Windows Service Start/Stop

how it works.

Once you are familiar with Icinga 2 and distributed monitoring, you can start with additional integrations to manage and deploy your configuration:

- Icinga Director provides a web interface to manage configuration and also allows to sync imported resources (CMDB, PuppetDB, etc.)
- Ansible Roles
- Puppet Module
- Chef Cookbook

More details can be found here.

6.10.1 Top Down

There are two different behaviors with check execution:

- Send a command execution event remotely: The scheduler still runs on the parent node.
- Sync the host/service objects directly to the child node: Checks are executed locally.

Again, technically it does not matter whether this is a `client` or a `satellite` which is receiving configuration or command execution events.

6.10.2 Top Down Command Endpoint

This mode will force the Icinga 2 node to execute commands remotely on a specified endpoint. The host/service object configuration is located on the master/satellite and the client only needs the CheckCommand object definitions being used there.

Every endpoint has its own remote check queue. The amount of checks executed simultaneously can be limited on the endpoint with the `MaxConcurrentChecks` constant defined in constants.conf. Icinga 2 may discard check requests, if the remote check queue is full.

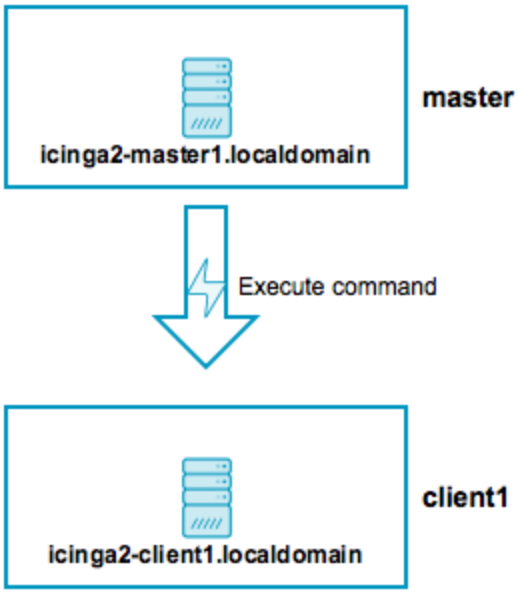


Figure 24: Icinga 2 Distributed Top Down Command Endpoint

Advantages:

- No local checks need to be defined on the child node (client).
- Light-weight remote check execution (asynchronous events).
- No replay log is necessary for the child node.
- Pin checks to specific endpoints (if the child zone consists of 2 endpoints).

Disadvantages:

- If the child node is not connected, no more checks are executed.

- Requires additional configuration attribute specified in host/service objects.
- Requires local `CheckCommand` object configuration. Best practice is to use a global config zone.

To make sure that all nodes involved will accept configuration and/or commands, you need to configure the `Zone` and `Endpoint` hierarchy on all nodes.

- `icinga2-master1.localdomain` is the configuration master in this scenario.
- `icinga2-client1.localdomain` acts as client which receives command execution messages via command endpoint from the master. In addition, it receives the global check command configuration from the master.

Include the endpoint and zone configuration on **both** nodes in the file `/etc/icinga2/zones.conf`.

The endpoint configuration could look like this, for example:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
  host = "192.168.56.101"
}

object Endpoint "icinga2-client1.localdomain" {
  host = "192.168.56.111"
}
```

Next, you need to define two zones. There is no naming convention, best practice is to either use `master`, `satellite/client-fqdn` or to choose region names for example `Europe`, `USA` and `Asia`, though.

Note: Each client requires its own zone and endpoint configuration. Best practice is to use the client's FQDN for all object names.

The `master` zone is a parent of the `icinga2-client1.localdomain` zone:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

object Zone "master" {
 endpoints = [ "icinga2-master1.localdomain" ] //array with endpoint names
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "master" //establish zone hierarchy
}
```

In addition, add a global zone for syncing check commands later:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

object Zone "global-templates" {
  global = true
}
```

Note: Packages >= 2.8 provide this configuration by default.

You don't need any local configuration on the client except for CheckCommand definitions which can be synced using the global zone above. Therefore disable the inclusion of the `conf.d` directory in `/etc/icinga2/icinga2.conf`.

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/icinga2.conf

// Commented out, not required on a client as command endpoint
//include_recursive "conf.d"
```

Edit the `api` feature on the client `icinga2-client1.localdomain` in the `/etc/icinga2/features-enabled/api.conf` file and make sure to set `accept_commands` and `accept_config` to `true`:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/features-enabled/api.conf

object ApiListener "api" {
   //...
   accept_commands = true
   accept_config = true
}
```

Now it is time to validate the configuration and to restart the Icinga 2 daemon on both nodes.

Example on CentOS 7:

```
[root@icinga2-client1.localdomain /]# icinga2 daemon -C
[root@icinga2-client1.localdomain /]# systemctl restart icinga2

[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Once the clients have successfully connected, you are ready for the next step: **execute a remote check on the client using the command endpoint**.

Include the host and service object configuration in the `master` zone – this will help adding a secondary master for high-availability later.

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/master
```

Add the host and service objects you want to monitor. There is no limitation for files and directories – best practice is to sort things by type.

By convention a master/satellite/client host object should use the same name as the endpoint object. You can also add multiple hosts which execute checks against remote services/clients.

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client1.localdomain" {
  check_command = "hostalive" //check is executed on the master
  address = "192.168.56.111"

 vars.client_endpoint = name //follows the convention that host name == endpoint name
}
```

Given that you are monitoring a Linux client, we'll add a remote disk check.

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "disk" {
  check_command = "disk"

  //specify where the check is executed
  command_endpoint = host.vars.client_endpoint

  assign where host.vars.client_endpoint
}
```

If you have your own custom `CheckCommand` definition, add it to the global zone:

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/global-templates
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/global-templates/commands.conf

object CheckCommand "my-cmd" {
  //...
}
```

Save the changes and validate the configuration on the master node:

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
```

Restart the Icinga 2 daemon (example for CentOS 7):

```
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

The following steps will happen:

- Icinga 2 validates the configuration on `icinga2-master1.localdomain` and restarts.
- The `icinga2-master1.localdomain` node schedules and executes the checks.

- The `icinga2-client1.localdomain` node receives the execute command event with additional command parameters.
- The `icinga2-client1.localdomain` node maps the command parameters to the local check command, executes the check locally, and sends back the check result message.

As you can see, no interaction from your side is required on the client itself, and it's not necessary to reload the Icinga 2 service on the client.

You have learned the basics about command endpoint checks. Proceed with the scenarios section where you can find detailed information on extending the setup.

6.10.3 Top Down Config Sync

This mode syncs the object configuration files within specified zones. It comes in handy if you want to configure everything on the master node and sync the satellite checks (disk, memory, etc.). The satellites run their own local scheduler and will send the check result messages back to the master.

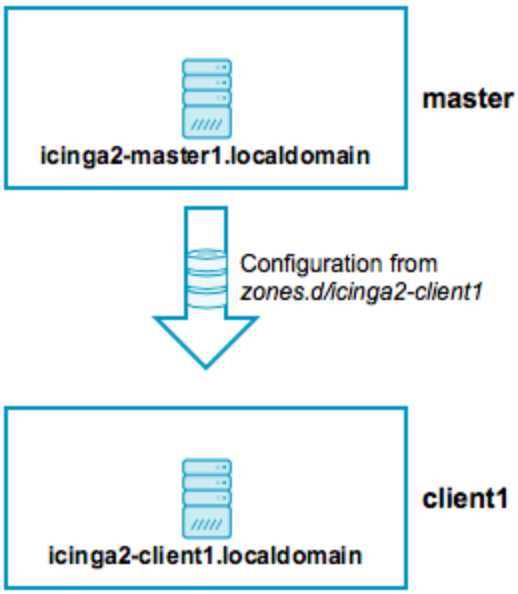


Figure 25: Icinga 2 Distributed Top Down Config Sync

Advantages:

- Sync the configuration files from the parent zone to the child zones.
- No manual restart is required on the child nodes, as syncing, validation, and restarts happen automatically.
- Execute checks directly on the child node's scheduler.
- Replay log if the connection drops (important for keeping the check history in sync, e.g. for SLA reports).
- Use a global zone for syncing templates, groups, etc.

Disadvantages:

- Requires a config directory on the master node with the zone name underneath `/etc/icinga2/zones.d`.
- Additional zone and endpoint configuration needed.
- Replay log is replicated on reconnect after connection loss. This might increase the data transfer and create an overload on the connection.

To make sure that all involved nodes accept configuration and/or commands, you need to configure the `Zone` and `Endpoint` hierarchy on all nodes.

- `icinga2-master1.localdomain` is the configuration master in this scenario.
- `icinga2-client2.localdomain` acts as client which receives configuration from the master. Checks are scheduled locally.

Include the endpoint and zone configuration on **both** nodes in the file `/etc/icinga2/zones.conf`.

The endpoint configuration could look like this:

```
[root@icinga2-client2.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
  host = "192.168.56.101"
}

object Endpoint "icinga2-client2.localdomain" {
  host = "192.168.56.112"
}
```

Next, you need to define two zones. There is no naming convention, best practice is to either use `master`, `satellite/client-fqdn` or to choose region names for example `Europe`, `USA` and `Asia`, though.

Note: Each client requires its own zone and endpoint configuration. Best practice is to use the client's FQDN for all object names.

The `master` zone is a parent of the `icinga2-client2.localdomain` zone:

```
[root@icinga2-client2.localdomain /]# vim /etc/icinga2/zones.conf

object Zone "master" {
```

```
 endpoints = [ "icinga2-master1.localdomain" ] //array with endpoint names
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]

  parent = "master" //establish zone hierarchy
}
```

Edit the `api` feature on the client `icinga2-client2.localdomain` in the `/etc/icinga2/features-enabled/api.conf` file and set `accept_config` to `true`.

```
[root@icinga2-client2.localdomain /]# vim /etc/icinga2/features-enabled/api.conf

object ApiListener "api" {
   //...
   accept_config = true
}
```

Now it is time to validate the configuration and to restart the Icinga 2 daemon on both nodes.

Example on CentOS 7:

```
[root@icinga2-client2.localdomain /]# icinga2 daemon -C
[root@icinga2-client2.localdomain /]# systemctl restart icinga2

[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Tip: Best practice is to use a global zone for common configuration items (check commands, templates, groups, etc.).

Once the clients have connected successfully, it's time for the next step: **execute a local check on the client using the configuration sync**.

Navigate to `/etc/icinga2/zones.d` on your master node `icinga2-master1.localdomain` and create a new directory with the same name as your satellite/client zone name:

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/icinga2-client2.localdoma
```

Add the host and service objects you want to monitor. There is no limitation for files and directories – best practice is to sort things by type.

By convention a master/satellite/client host object should use the same name as the endpoint object. You can also add multiple hosts which execute checks against remote services/clients.

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/icinga2-client2.localdomain
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/icinga2-client2.localdomain]# vim host
```

```
object Host "icinga2-client2.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.112"
 zone = "master" //optional trick: sync the required host object to the client, but enforce the '
}
```

Given that you are monitoring a Linux client we'll just add a local disk check.

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/icinga2-client2.localdomain]# vim serv

object Service "disk" {
  host_name = "icinga2-client2.localdomain"

  check_command = "disk"
}
```

Save the changes and validate the configuration on the master node:

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
```

Restart the Icinga 2 daemon (example for CentOS 7):

```
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

The following steps will happen:

- Icinga 2 validates the configuration on `icinga2-master1.localdomain`.
- Icinga 2 copies the configuration into its zone config store in `/var/lib/icinga2/api/zones`.
- The `icinga2-master1.localdomain` node sends a config update event to all endpoints in the same or direct child zones.
- The `icinga2-client2.localdomain` node accepts config and populates the local zone config store with the received config files.
- The `icinga2-client2.localdomain` node validates the configuration and automatically restarts.

Again, there is no interaction required on the client itself.

You can also use the config sync inside a high-availability zone to ensure that all config objects are synced among zone members.

Note: You can only have one so-called "config master" in a zone which stores the configuration in the `zones.d` directory. Multiple nodes with configuration files in the `zones.d` directory are **not supported**.

Now that you've learned the basics about the configuration sync, proceed with the scenarios section where you can find detailed information on extending the setup.

If you are eager to start fresh instead you might take a look into the Icinga Director.

6.11 Scenarios

The following examples should give you an idea on how to build your own distributed monitoring environment. We've seen them all in production environments and received feedback from our community and partner support channels:

- Single master with clients.
- HA master with clients as command endpoint.
- Three level cluster with config HA masters, satellites receiving config sync, and clients checked using command endpoint.

6.11.1 Master with Clients

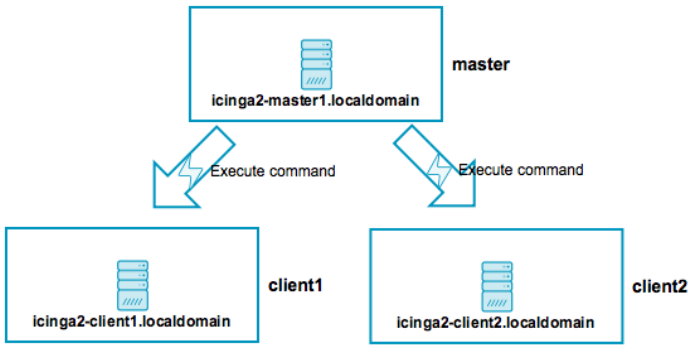


Figure 26: Icinga 2 Distributed Master with Clients

- `icinga2-master1.localdomain` is the primary master node.
- `icinga2-client1.localdomain` and `icinga2-client2.localdomain` are two child nodes as clients.

Setup requirements:

- Set up `icinga2-master1.localdomain` as master.
- Set up `icinga2-client1.localdomain` and `icinga2-client2.localdomain` as client.

Edit the `zones.conf` configuration file on the master:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
}

object Endpoint "icinga2-client1.localdomain" {
 host = "192.168.56.111" //the master actively tries to connect to the client
```

```
}

object Endpoint "icinga2-client2.localdomain" {
 host = "192.168.56.112" //the master actively tries to connect to the client
}

object Zone "master" {
  endpoints = [ "icinga2-master1.localdomain" ]
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "master"
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]

  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

The two client nodes do not necessarily need to know about each other. The only important thing is that they know about the parent zone and their endpoint members (and optionally the global zone).

If you specify the `host` attribute in the `icinga2-master1.localdomain` endpoint object, the client will actively try to connect to the master node. Since we've specified the client endpoint's attribute on the master node already, we don't want the clients to connect to the master. **Choose one connection direction.**

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
}

object Endpoint "icinga2-client1.localdomain" {
}

object Zone "master" {
```

```
  endpoints = [ "icinga2-master1.localdomain" ]
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}

[root@icinga2-client2.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
}

object Endpoint "icinga2-client2.localdomain" {
}

object Zone "master" {
  endpoints = [ "icinga2-master1.localdomain" ]
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]

  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

Now it is time to define the two client hosts and apply service checks using the command endpoint execution method on them. Note: You can also use the config sync mode here.

Create a new configuration directory on the master node:

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/master
```

Add the two client nodes as host objects:

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client1.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.111"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
}

object Host "icinga2-client2.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.112"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
}
```

Add services using command endpoint checks:

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "ping4" {
  check_command = "ping4"
  //check is executed on the master node
  assign where host.address
}

apply Service "disk" {
  check_command = "disk"

  //specify where the check is executed
  command_endpoint = host.vars.client_endpoint

  assign where host.vars.client_endpoint
}
```

Validate the configuration and restart Icinga 2 on the master node `icinga2-master1.localdomain`.

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Open Icinga Web 2 and check the two newly created client hosts with two new services – one executed locally (`ping4`) and one using command endpoint (`disk`).

6.11.2 High-Availability Master with Clients

This scenario is similar to the one in the previous section. The only difference is that we will now set up two master nodes in a high-availability setup. These

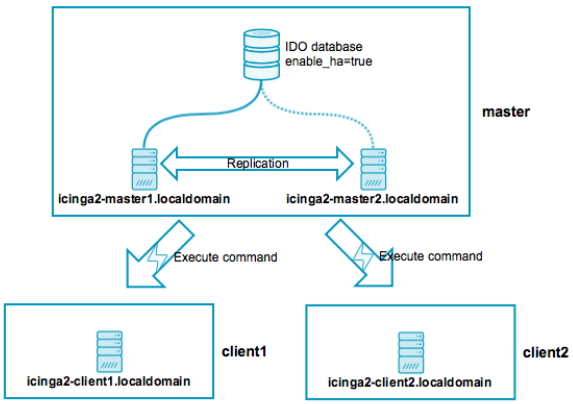


Figure 27: Icinga 2 Distributed High Availability Master with Clients

nodes must be configured as zone and endpoints objects.

The setup uses the capabilities of the Icinga 2 cluster. All zone members replicate cluster events amongst each other. In addition to that, several Icinga 2 features can enable HA functionality.

Note: All nodes in the same zone require that you enable the same features for high-availability (HA).

Overview:

- `icinga2-master1.localdomain` is the config master master node.
- `icinga2-master2.localdomain` is the secondary master master node without config in `zones.d`.
- `icinga2-client1.localdomain` and `icinga2-client2.localdomain` are two child nodes as clients.

Setup requirements:

- Set up `icinga2-master1.localdomain` as master.
- Set up `icinga2-master2.localdomain` as client (we will modify the generated configuration).
- Set up `icinga2-client1.localdomain` and `icinga2-client2.localdomain` as clients (when asked for adding multiple masters, set to `y` and add the secondary master `icinga2-master2.localdomain`).

In case you don't want to use the CLI commands, you can also manually create and sync the required SSL certificates. We will modify and discuss all the details of the automatically generated configuration here.

Since there are now two nodes in the same zone, we must consider the high-availability features.

- Checks and notifications are balanced between the two master nodes. That's fine, but it requires check plugins and notification scripts to exist on both nodes.
- The IDO feature will only be active on one node by default. Since all events are replicated between both nodes, it is easier to just have one central database.

One possibility is to use a dedicated MySQL cluster VIP (external application cluster) and leave the IDO feature with enabled HA capabilities. Alternatively, you can disable the HA feature and write to a local database on each node. Both methods require that you configure Icinga Web 2 accordingly (monitoring backend, IDO database, used transports, etc.).

The zone hierarchy could look like this. It involves putting the two master nodes `icinga2-master1.localdomain` and `icinga2-master2.localdomain` into the `master` zone.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
  host = "192.168.56.101"
}

object Endpoint "icinga2-master2.localdomain" {
  host = "192.168.56.102"
}

object Endpoint "icinga2-client1.localdomain" {
 host = "192.168.56.111" //the master actively tries to connect to the client
}

object Endpoint "icinga2-client2.localdomain" {
 host = "192.168.56.112" //the master actively tries to connect to the client
}

object Zone "master" {
 endpoints = [ "icinga2-master1.localdomain", "icinga2-master2.localdomain" ]
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "master"
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]
```

```
  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

The two client nodes do not necessarily need to know about each other. The only important thing is that they know about the parent zone and their endpoint members (and optionally about the global zone).

If you specify the `host` attribute in the `icinga2-master1.localdomain` and `icinga2-master2.localdomain` endpoint objects, the client will actively try to connect to the master node. Since we've specified the client endpoint's attribute on the master node already, we don't want the clients to connect to the master nodes. **Choose one connection direction.**

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
}

object Endpoint "icinga2-master2.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
}

object Endpoint "icinga2-client1.localdomain" {
}

object Zone "master" {
 endpoints = [ "icinga2-master1.localdomain", "icinga2-master2.localdomain" ]
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

```
[root@icinga2-client2.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
}

object Endpoint "icinga2-master2.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
}

object Endpoint "icinga2-client2.localdomain" {
}

object Zone "master" {
 endpoints = [ "icinga2-master1.localdomain", "icinga2-master2.localdomain" ]
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]

  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

Now it is time to define the two client hosts and apply service checks using the command endpoint execution method. Note: You can also use the config sync mode here.

Create a new configuration directory on the master node `icinga2-master1.localdomain`. **Note**: The secondary master node `icinga2-master2.localdomain` receives the configuration using the config sync mode.

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/master
```

Add the two client nodes as host objects:

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client1.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.111"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
}
```

```
object Host "icinga2-client2.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.112"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
}
```

Add services using command endpoint checks:

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "ping4" {
  check_command = "ping4"
  //check is executed on the master node
  assign where host.address
}

apply Service "disk" {
  check_command = "disk"

  //specify where the check is executed
  command_endpoint = host.vars.client_endpoint

  assign where host.vars.client_endpoint
}
```

Validate the configuration and restart Icinga 2 on the master node `icinga2-master1.localdomain`.

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Open Icinga Web 2 and check the two newly created client hosts with two new services – one executed locally (`ping4`) and one using command endpoint (`disk`).

Tip: It's a good idea to add health checks to make sure that your cluster notifies you in case of failure.

6.11.3 Three Levels with Master, Satellites, and Clients

This scenario combines everything you've learned so far: High-availability masters, satellites receiving their configuration from the master zone, and clients checked via command endpoint from the satellite zones.

Tip: It can get complicated, so grab a pen and paper and bring your thoughts to life. Play around with a test setup before using it in a production environment!

Overview:

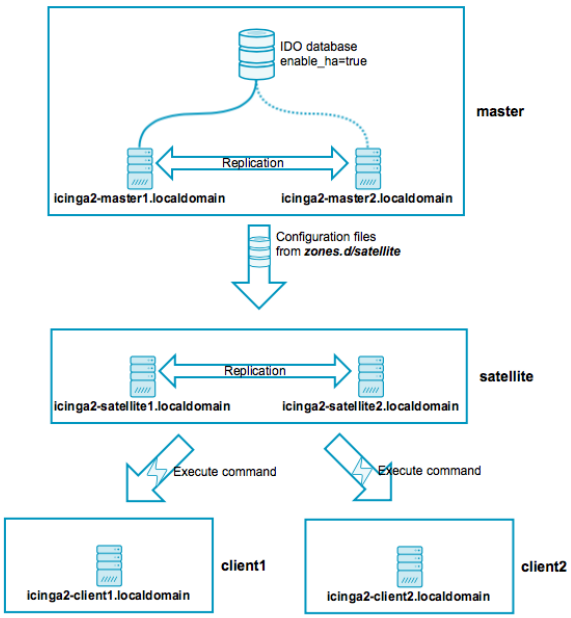


Figure 28: Icinga 2 Distributed Master and Satellites with Clients

- `icinga2-master1.localdomain` is the configuration master master node.
- `icinga2-master2.localdomain` is the secondary master master node without configuration in `zones.d`.
- `icinga2-satellite1.localdomain` and `icinga2-satellite2.localdomain` are satellite nodes in a `master` child zone.
- `icinga2-client1.localdomain` and `icinga2-client2.localdomain` are two child nodes as clients.

Setup requirements:

- Set up `icinga2-master1.localdomain` as master.
- Set up `icinga2-master2.localdomain`, `icinga2-satellite1.localdomain` and `icinga2-satellite2.localdomain` as clients (we will modify the generated configuration).
- Set up `icinga2-client1.localdomain` and `icinga2-client2.localdomain` as clients.

When being asked for the master endpoint providing CSR auto-signing capabilities, please add the master node which holds the CA and has the `ApiListener` feature configured and enabled. The parent endpoint must still remain the satellite endpoint name.

Example for `icinga2-client1.localdomain`:

```
Please specify the master endpoint(s) this node should connect to:
```

Master is the first satellite `icinga2-satellite1.localdomain`:

```
Master Common Name (CN from your master setup): icinga2-satellite1.localdomain
Do you want to establish a connection to the master from this node? [Y/n]: y
Please fill out the master connection information:
Master endpoint host (Your master's IP address or FQDN): 192.168.56.105
Master endpoint port [5665]:
```

Add the second satellite `icinga2-satellite2.localdomain` as master:

```
Add more master endpoints? [y/N]: y
Master Common Name (CN from your master setup): icinga2-satellite2.localdomain
Do you want to establish a connection to the master from this node? [Y/n]: y
Please fill out the master connection information:
Master endpoint host (Your master's IP address or FQDN): 192.168.56.106
Master endpoint port [5665]:
Add more master endpoints? [y/N]: n
```

Specify the master node `icinga2-master2.localdomain` with the CA private key and ticket salt:

```
Please specify the master connection for CSR auto-signing (defaults to master endpoint host):
Host [192.168.56.106]: icinga2-master1.localdomain
Port [5665]:
```

In case you cannot connect to the master node from your clients, you'll manually need to generate the SSL certificates and modify the configuration accordingly.

We'll discuss the details of the required configuration below.

The zone hierarchy can look like this. We'll define only the directly connected zones here.

You can safely deploy this configuration onto all master and satellite zone members. You should keep in mind to control the endpoint connection direction using the `host` attribute.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-master1.localdomain" {
  host = "192.168.56.101"
}

object Endpoint "icinga2-master2.localdomain" {
  host = "192.168.56.102"
}

object Endpoint "icinga2-satellite1.localdomain" {
  host = "192.168.56.105"
```

```
}

object Endpoint "icinga2-satellite2.localdomain" {
  host = "192.168.56.106"
}

object Zone "master" {
 endpoints = [ "icinga2-master1.localdomain", "icinga2-master2.localdomain" ]
}

object Zone "satellite" {
 endpoints = [ "icinga2-satellite1.localdomain", "icinga2-satellite2.localdomain" ]

  parent = "master"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

Repeat the configuration step for `icinga2-master2.localdomain`, `icinga2-satellite1.localdomain` and `icinga2-satellite2.localdomain`.

Since we want to use top down command endpoint checks, we must configure the client endpoint and zone objects. In order to minimize the effort, we'll sync the client zone and endpoint configuration to the satellites where the connection information is needed as well.

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/{master,satellite,global-
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/satellite

[root@icinga2-master1.localdomain /etc/icinga2/zones.d/satellite]# vim icinga2-client1.localc

object Endpoint "icinga2-client1.localdomain" {
 host = "192.168.56.111" //the satellite actively tries to connect to the client
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "satellite"
}

[root@icinga2-master1.localdomain /etc/icinga2/zones.d/satellite]# vim icinga2-client2.localc

object Endpoint "icinga2-client2.localdomain" {
```

```
 host = "192.168.56.112" //the satellite actively tries to connect to the client
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]

  parent = "satellite"
}
```

The two client nodes do not necessarily need to know about each other, either. The only important thing is that they know about the parent zone (the satellite) and their endpoint members (and optionally the global zone).

If you specify the `host` attribute in the `icinga2-satellite1.localdomain` and `icinga2-satellite2.localdomain` endpoint objects, the client node will actively try to connect to the satellite node. Since we've specified the client endpoint's attribute on the satellite node already, we don't want the client node to connect to the satellite nodes. **Choose one connection direction.**

Example for `icinga2-client1.localdomain`:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-satellite1.localdomain" {
 //do not actively connect to the satellite by leaving out the 'host' attribute
}

object Endpoint "icinga2-satellite2.localdomain" {
 //do not actively connect to the satellite by leaving out the 'host' attribute
}

object Endpoint "icinga2-client1.localdomain" {
  //that's us
}

object Zone "satellite" {
 endpoints = [ "icinga2-satellite1.localdomain", "icinga2-satellite2.localdomain" ]
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]

  parent = "satellite"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
```

```
}
```

Example for `icinga2-client2.localdomain`:

```
[root@icinga2-client2.localdomain /]# vim /etc/icinga2/zones.conf

object Endpoint "icinga2-satellite1.localdomain" {
 //do not actively connect to the satellite by leaving out the 'host' attribute
}

object Endpoint "icinga2-satellite2.localdomain" {
 //do not actively connect to the satellite by leaving out the 'host' attribute
}

object Endpoint "icinga2-client2.localdomain" {
  //that's us
}

object Zone "satellite" {
 endpoints = [ "icinga2-satellite1.localdomain", "icinga2-satellite2.localdomain" ]
}

object Zone "icinga2-client2.localdomain" {
  endpoints = [ "icinga2-client2.localdomain" ]

  parent = "satellite"
}

/* sync global commands */
object Zone "global-templates" {
  global = true
}
```

Now it is time to define the two client hosts on the master, sync them to the satellites and apply service checks using the command endpoint execution method to them. Add the two client nodes as host objects to the `satellite` zone.

We've already created the directories in `/etc/icinga2/zones.d` including the files for the zone and endpoint configuration for the clients.

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/satellite
```

Add the host object configuration for the `icinga2-client1.localdomain` client. You should have created the configuration file in the previous steps and it should contain the endpoint and zone object configuration already.

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/satellite]# vim icinga2-client1.localdo

object Host "icinga2-client1.localdomain" {
```

```
  check_command = "hostalive"
  address = "192.168.56.111"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
}
```

Add the host object configuration for the `icinga2-client2.localdomain` client configuration file:

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/satellite]# vim icinga2-client2.localdomain.conf

object Host "icinga2-client2.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.112"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
}
```

Add a service object which is executed on the satellite nodes (e.g. `ping4`). Pin the apply rule to the `satellite` zone only.

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/satellite]# vim services.conf

apply Service "ping4" {
  check_command = "ping4"
  //check is executed on the satellite node
  assign where host.zone == "satellite" && host.address
}
```

Add services using command endpoint checks. Pin the apply rules to the `satellite` zone only.

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/satellite]# vim services.conf

apply Service "disk" {
  check_command = "disk"

  //specify where the check is executed
  command_endpoint = host.vars.client_endpoint

 assign where host.zone == "satellite" && host.vars.client_endpoint
}
```

Validate the configuration and restart Icinga 2 on the master node `icinga2-master1.localdomain`.

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Open Icinga Web 2 and check the two newly created client hosts with two new services – one executed locally (`ping4`) and one using command endpoint (`disk`).

Tip: It's a good idea to add health checks to make sure that your cluster notifies you in case of failure.

6.12 Best Practice

We've put together a collection of configuration examples from community feedback. If you like to share your tips and tricks with us, please join the community channels!

6.12.1 Global Zone for Config Sync

Global zones can be used to sync generic configuration objects to all nodes depending on them. Common examples are:

- Templates which are imported into zone specific objects.
- Command objects referenced by Host, Service, Notification objects.
- Apply rules for services, notifications, dependencies and scheduled downtimes.
- User objects referenced in notifications.
- Group objects.
- TimePeriod objects.

Plugin scripts and binaries cannot be synced, this is for Icinga 2 configuration files only. Use your preferred package repository and/or configuration management tool (Puppet, Ansible, Chef, etc.) for that.

Note: Checkable objects (hosts and services) cannot be put into a global zone. The configuration validation will terminate with an error.

The zone object configuration must be deployed on all nodes which should receive the global configuration files:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.conf

object Zone "global-templates" {
  global = true
}
```

Note: Packages >= 2.8 provide this configuration by default.

Similar to the zone configuration sync you'll need to create a new directory in `/etc/icinga2/zones.d`:

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/global-templates
```

Next, add a new check command, for example:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/global-templates/commands.conf
```

```
object CheckCommand "my-cmd" {
  //...
}
```

Restart the client(s) which should receive the global zone before before restarting the parent master/satellite nodes.

Then validate the configuration on the master node and restart Icinga 2.

Tip: You can copy the example configuration files located in `/etc/icinga2/conf.d` into your global zone.

Example:

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/conf.d
[root@icinga2-master1.localdomain /etc/icinga2/conf.d]# cp {commands,downtimes,groups,notifi
```

6.12.2 Health Checks

In case of network failures or other problems, your monitoring might either have late check results or just send out mass alarms for unknown checks.

In order to minimize the problems caused by this, you should configure additional health checks.

The `cluster` check, for example, will check if all endpoints in the current zone and the directly connected zones are working properly:

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/icinga2-master1.localdo

object Host "icinga2-master1.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.101"
}

[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/cluster.conf

object Service "cluster" {
  check_command = "cluster"
  check_interval = 5s
  retry_interval = 1s

  host_name = "icinga2-master1.localdomain"
}
```

The `cluster-zone` check will test whether the configured target zone is currently connected or not. This example adds a health check for the ha master with clients scenario.

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/services.conf

apply Service "cluster-health" {
  check_command = "cluster-zone"

  display_name = "cluster-health-" + host.name

 /* This follows the convention that the client zone name is the FQDN which is the same as the hos
  vars.cluster_zone = host.name

  assign where host.vars.client_endpoint
}
```

In case you cannot assign the `cluster_zone` attribute, add specific checks to your cluster:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/cluster.conf

object Service "cluster-zone-satellite" {
  check_command = "cluster-zone"
  check_interval = 5s
  retry_interval = 1s
  vars.cluster_zone = "satellite"

  host_name = "icinga2-master1.localdomain"
}
```

If you are using top down checks with command endpoint configuration, you can add a dependency which prevents notifications for all other failing services:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/dependencies.conf

apply Dependency "health-check" to Service {
  parent_service_name = "child-health"

  states = [ OK ]
  disable_notifications = true

  assign where host.vars.client_endpoint
  ignore where service.name == "child-health"
}
```

6.12.3 Pin Checks in a Zone

In case you want to pin specific checks to their endpoints in a given zone you'll need to use the `command_endpoint` attribute. This is reasonable if you want to execute a local disk check in the `master` Zone on a specific endpoint then.

```
[root@icinga2-master1.localdomain /]# mkdir -p /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/icinga2-master1.localdo

object Host "icinga2-master1.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.101"
}

[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.d/master/services.conf

apply Service "disk" {
  check_command = "disk"

 command_endpoint = host.name //requires a host object matching the endpoint object name e.g. ic

 assign where host.zone == "master" && match("icinga2-master*", host.name)
}
```

The `host.zone` attribute check inside the expression ensures that the service object is only created for host objects inside the `master` zone. In addition to that the match function ensures to only create services for the master nodes.

6.12.4 Windows Firewall

6.12.4.1 ICMP Requests

By default ICMP requests are disabled in the Windows firewall. You can change that by adding a new rule.

```
C:\WINDOWS\system32>netsh advfirewall firewall add rule name="ICMP Allow incoming V4 echo reque
```

6.12.4.2 Icinga 2

If your master/satellite nodes should actively connect to the Windows client you'll also need to ensure that port `5665` is enabled.

```
C:\WINDOWS\system32>netsh advfirewall firewall add rule name="Open port 5665 (Icinga 2)" dir=in
```

6.12.4.3 NSClient++ API

If the check_nscp_api plugin is used to query NSClient++ remotely, you need to ensure that its port is enabled.

```
C:\WINDOWS\system32>netsh advfirewall firewall add rule name="Open port 8443 (NSClient++ API)"
```

6.12.5 Windows Client and Plugins

The Icinga 2 package on Windows already provides several plugins. Detailed documentation is available for all check command definitions.

Add the following `include` statement on all your nodes (master, satellite, client):

```
vim /etc/icinga2/icinga2.conf

include <windows-plugins>
```

Based on the master with clients scenario we'll now add a local disk check.

First, add the client node as host object:

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client2.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.112"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
  vars.os_type = "windows"
}
```

Next, add the disk check using command endpoint checks (details in the disk-windows documentation):

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "disk C:" {
  check_command = "disk-windows"

  vars.disk_win_path = "C:"

  //specify where the check is executed
  command_endpoint = host.vars.client_endpoint

 assign where host.vars.os_type == "windows" && host.vars.client_endpoint
}
```

Validate the configuration and restart Icinga 2.

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Open Icinga Web 2 and check your newly added Windows disk check :)

If you want to add your own plugins please check this chapter for the requirements.

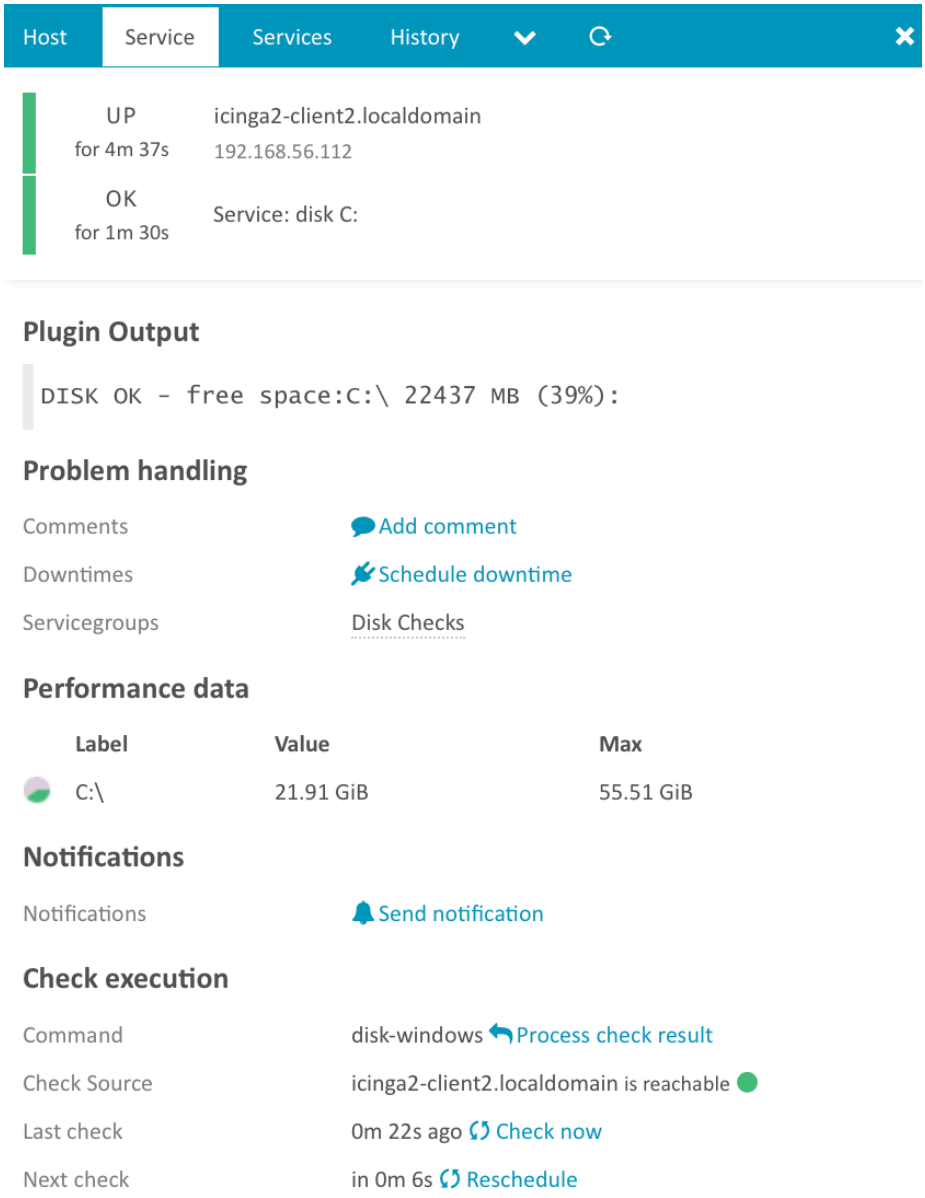


Figure 29: Icinga 2 Client Windows

6.12.6 Windows Client and NSClient++

There are two methods available for querying NSClient++:

- Query the HTTP API locally or remotely (requires a running NSClient++ service)
- Run a local CLI check (does not require NSClient++ as a service)

Both methods have their advantages and disadvantages. One thing to note: If you rely on performance counter delta calculations such as CPU utilization, please use the HTTP API instead of the CLI sample call.

6.12.6.1 NSCLient++ with check_nscp_api

The Windows setup already allows you to install the NSClient++ package. In addition to the Windows plugins you can use the nscp_api command provided by the Icinga Template Library (ITL).

The initial setup for the NSClient++ API and the required arguments is the described in the ITL chapter for the nscp_api CheckCommand.

Based on the master with clients scenario we'll now add a local nscp check which queries the NSClient++ API to check the free disk space.

Define a host object called `icinga2-client2.localdomain` on the master. Add the `nscp_api_password` custom attribute and specify the drives to check.

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client1.localdomain" {
    check_command = "hostalive"
    address = "192.168.56.111"
   vars.client_endpoint = name //follows the convention that host name == endpoint name
    vars.os_type = "Windows"
    vars.nscp_api_password = "icinga"
    vars.drives = [ "C:", "D:" ]
}
```

The service checks are generated using an apply for rule based on `host.vars.drives`:

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "nscp-api-" for (drive in host.vars.drives) {
  import "generic-service"

  check_command = "nscp_api"
  command_endpoint = host.vars.client_endpoint
```

```
  //display_name = "nscp-drive-" + drive

  vars.nscp_api_host = "localhost"
  vars.nscp_api_query = "check_drivesize"
  vars.nscp_api_password = host.vars.nscp_api_password
  vars.nscp_api_arguments = [ "drive=" +  drive ]

  ignore where host.vars.os_type != "Windows"
}
```

Validate the configuration and restart Icinga 2.

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Two new services ("nscp-drive-D:" and "nscp-drive-C:") will be visible in Icinga Web 2.

Note: You can also omit the `command_endpoint` configuration to execute the command on the master. This also requires a different value for `nscp_api_host` which defaults to `host.address`.

```
  //command_endpoint = host.vars.client_endpoint

  //vars.nscp_api_host = "localhost"
```

You can verify the check execution by looking at the `Check Source` attribute in Icinga Web 2 or the REST API.

If you want to monitor specific Windows services, you could use the following example:

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client1.localdomain" {
    check_command = "hostalive"
    address = "192.168.56.111"
   vars.client_endpoint = name //follows the convention that host name == endpoint name
    vars.os_type = "Windows"
    vars.nscp_api_password = "icinga"
    vars.services = [ "Windows Update", "wscsvc" ]
}

[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "nscp-api-" for (svc in host.vars.services) {
  import "generic-service"
```

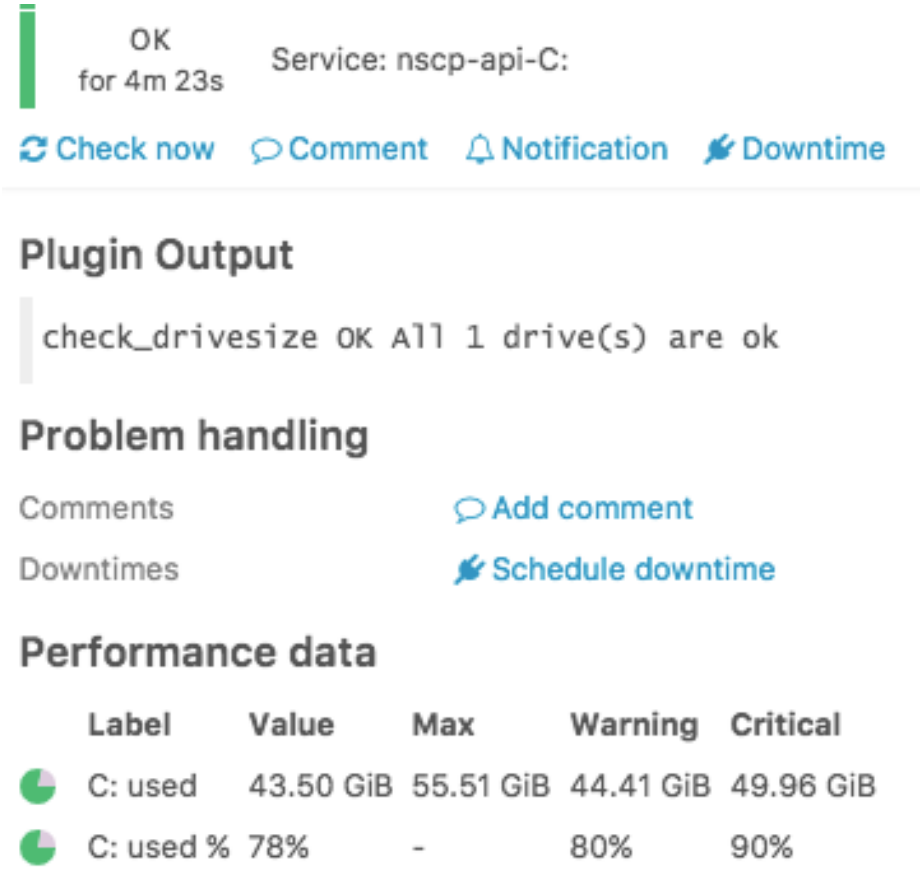


Figure 30: Icinga 2 Distributed Monitoring Windows Client with NSClient++ nscp-api

```
  check_command = "nscp_api"
  command_endpoint = host.vars.client_endpoint

  //display_name = "nscp-service-" + svc

  vars.nscp_api_host = "localhost"
  vars.nscp_api_query = "check_service"
  vars.nscp_api_password = host.vars.nscp_api_password
  vars.nscp_api_arguments = [ "service=" + svc ]

  ignore where host.vars.os_type != "Windows"
}
```

6.12.6.2 NSCLient++ with nscp-local

The Windows setup already allows you to install the NSClient++ package. In addition to the Windows plugins you can use the nscp-local commands provided by the Icinga Template Library (ITL).

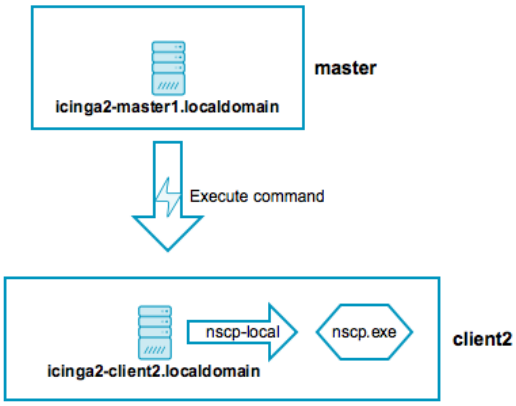


Figure 31: Icinga 2 Distributed Monitoring Windows Client with NSClient++

Add the following `include` statement on all your nodes (master, satellite, client):

`vim /etc/icinga2/icinga2.conf`

```
include <nscp>
```

The CheckCommand definitions will automatically determine the installed path to the `nscp.exe` binary.

Based on the master with clients scenario we'll now add a local nscp check querying a given performance counter.

First, add the client node as host object:

```
[root@icinga2-master1.localdomain /]# cd /etc/icinga2/zones.d/master
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim hosts.conf

object Host "icinga2-client1.localdomain" {
  check_command = "hostalive"
  address = "192.168.56.111"
 vars.client_endpoint = name //follows the convention that host name == endpoint name
  vars.os_type = "windows"
}
```

Next, add a performance counter check using command endpoint checks (details in the nscp-local-counter documentation):

```
[root@icinga2-master1.localdomain /etc/icinga2/zones.d/master]# vim services.conf

apply Service "nscp-local-counter-cpu" {
  check_command = "nscp-local-counter"
  command_endpoint = host.vars.client_endpoint

  vars.nscp_counter_name = "\\Processor(_total)\\% Processor Time"
  vars.nscp_counter_perfsyntax = "Total Processor Time"
  vars.nscp_counter_warning = 1
  vars.nscp_counter_critical = 5

  vars.nscp_counter_showall = true

 assign where host.vars.os_type == "windows" && host.vars.client_endpoint
}
```

Validate the configuration and restart Icinga 2.

```
[root@icinga2-master1.localdomain /]# icinga2 daemon -C
[root@icinga2-master1.localdomain /]# systemctl restart icinga2
```

Open Icinga Web 2 and check your newly added Windows NSClient++ check :)

6.13 Advanced Hints

You can find additional hints in this section if you prefer to go your own route with automating setups (setup, certificates, configuration).

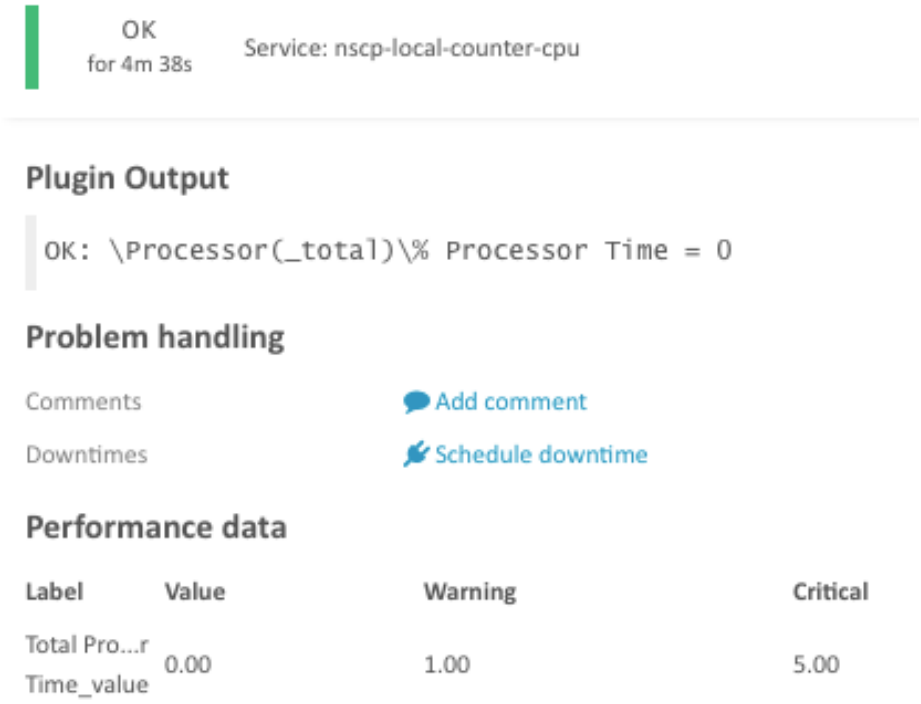


Figure 32: Icinga 2 Distributed Monitoring Windows Client with NSClient++ nscp-local

6.13.1 Certificate Auto-Renewal

Icinga 2 v2.8+ adds the possibility that nodes request certificate updates on their own. If their expiration date is soon enough, they automatically renew their already signed certificate by sending a signing request to the parent node.

6.13.2 High-Availability for Icinga 2 Features

All nodes in the same zone require that you enable the same features for high-availability (HA).

By default, the following features provide advanced HA functionality:

- Checks (load balanced, automated failover).
- Notifications (load balanced, automated failover).
- DB IDO (Run-Once, automated failover).

6.13.2.1 High-Availability with Checks

All instances within the same zone (e.g. the `master` zone as HA cluster) must have the `checker` feature enabled.

Example:

```
# icinga2 feature enable checker
```

All nodes in the same zone load-balance the check execution. If one instance shuts down, the other nodes will automatically take over the remaining checks.

6.13.2.2 High-Availability with Notifications

All instances within the same zone (e.g. the `master` zone as HA cluster) must have the `notification` feature enabled.

Example:

```
# icinga2 feature enable notification
```

Notifications are load-balanced amongst all nodes in a zone. By default this functionality is enabled. If your nodes should send out notifications independently from any other nodes (this will cause duplicated notifications if not properly handled!), you can set `enable_ha = false` in the NotificationComponent feature.

6.13.2.3 High-Availability with DB IDO

All instances within the same zone (e.g. the `master` zone as HA cluster) must have the DB IDO feature enabled.

Example DB IDO MySQL:

```
# icinga2 feature enable ido-mysql
```

By default the DB IDO feature only runs on one node. All other nodes in the same zone disable the active IDO database connection at runtime. The node with the active DB IDO connection is not necessarily the zone master.

Note: The DB IDO HA feature can be disabled by setting the `enable_ha` attribute to `false` for the IdoMysqlConnection or IdoPgsqlConnection object on **all** nodes in the **same** zone.

All endpoints will enable the DB IDO feature and connect to the configured database and dump configuration, status and historical data on their own.

If the instance with the active DB IDO connection dies, the HA functionality will automatically elect a new DB IDO master.

The DB IDO feature will try to determine which cluster endpoint is currently writing to the database and bail out if another endpoint is active. You can manually verify that by running the following query command:

```
icinga=> SELECT status_update_time, endpoint_name FROM icinga_programstatus;
   status_update_time    | endpoint_name
-------------------------+---------------
 2016-08-15 15:52:26+02 | icinga2-master1.localdomain
(1 Zeile)
```

This is useful when the cluster connection between endpoints breaks, and prevents data duplication in split-brain-scenarios. The failover timeout can be set for the `failover_timeout` attribute, but not lower than 60 seconds.

6.13.3 Endpoint Connection Direction

Nodes will attempt to connect to another node when its local Endpoint object configuration specifies a valid `host` attribute (FQDN or IP address).

Example for the master node `icinga2-master1.localdomain` actively connecting to the client node `icinga2-client1.localdomain`:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.conf

//...

object Endpoint "icinga2-client1.localdomain" {
 host = "192.168.56.111" //the master actively tries to connect to the client
  log_duration = 0
}
```

Example for the client node `icinga2-client1.localdomain` not actively connecting to the master node `icinga2-master1.localdomain`:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

//...

object Endpoint "icinga2-master1.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
  log_duration = 0
}
```

It is not necessary that both the master and the client node establish two connections to each other. Icinga 2 will only use one connection and close the second connection if established.

Tip: Choose either to let master/satellite nodes connect to client nodes or vice versa.

6.13.4 Disable Log Duration for Command Endpoints

The replay log is a built-in mechanism to ensure that nodes in a distributed setup keep the same history (check results, notifications, etc.) when nodes are temporarily disconnected and then reconnect.

This functionality is not needed when a master/satellite node is sending check execution events to a client which is purely configured for command endpoint checks only.

The Endpoint object attribute `log_duration` can be lower or set to 0 to fully disable any log replay updates when the client is not connected.

Configuration on the master node `icinga2-master1.localdomain`:

```
[root@icinga2-master1.localdomain /]# vim /etc/icinga2/zones.conf

//...

object Endpoint "icinga2-client1.localdomain" {
 host = "192.168.56.111" //the master actively tries to connect to the client
  log_duration = 0
}

object Endpoint "icinga2-client2.localdomain" {
 host = "192.168.56.112" //the master actively tries to connect to the client
  log_duration = 0
}
```

Configuration on the client `icinga2-client1.localdomain`:

```
[root@icinga2-client1.localdomain /]# vim /etc/icinga2/zones.conf

//...

object Endpoint "icinga2-master1.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
  log_duration = 0
}

object Endpoint "icinga2-master2.localdomain" {
 //do not actively connect to the master by leaving out the 'host' attribute
  log_duration = 0
}
```

6.13.5 CSR auto-signing with HA and multiple Level Cluster

If you are using two masters in a High-Availability setup it can be necessary to allow both to sign requested certificates. Ensure to safely sync the following details in private:

- `TicketSalt` constant in `constants.conf`.
- `var/lib/icinga2/ca` directory.

This also helps if you are using a three level cluster and your client nodes are not able to reach the CSR auto-signing master node(s). Make sure that the directory permissions for `/var/lib/icinga2/ca` are secure (not world readable).

Do not expose these private keys to anywhere else. This is a matter of security.

6.13.6 Manual Certificate Creation

6.13.6.1 Create CA on the Master

Choose the host which should store the certificate authority (one of the master nodes).

The first step is the creation of the certificate authority (CA) by running the following command as root user:

```
[root@icinga2-master1.localdomain /root]# icinga2 pki new-ca
```

6.13.6.2 Create CSR and Certificate

Create a certificate signing request (CSR) for the local instance:

```
[root@icinga2-master1.localdomain /root]# icinga2 pki new-cert --cn icinga2-master1.localdomai
  --key icinga2-master1.localdomain.key \
  --csr icinga2-master1.localdomain.csr
```

Sign the CSR with the previously created CA:

```
[root@icinga2-master1.localdomain /root]# icinga2 pki sign-csr --csr icinga2-master1.localdoma
```

Repeat the steps for all instances in your setup.

Note

The certificate location changed in v2.8 to `/var/lib/icinga2/certs`.
Please read the upgrading chapter for more details.

6.13.6.3 Copy Certificates

Copy the host's certificate files and the public CA certificate to `/var/lib/icinga2/certs`:

```
[root@icinga2-master1.localdomain /root]# mkdir -p /var/lib/icinga2/certs
[root@icinga2-master1.localdomain /root]# cp icinga2-master1.localdomain.{crt,key} /var/lib/i
[root@icinga2-master1.localdomain /root]# cp /var/lib/icinga2/ca/ca.crt /var/lib/icinga2/cert
```

Ensure that proper permissions are set (replace `icinga` with the Icinga 2 daemon user):

```
[root@icinga2-master1.localdomain /root]# chown -R icinga:icinga /var/lib/icinga2/certs
[root@icinga2-master1.localdomain /root]# chmod 600 /var/lib/icinga2/certs/*.key
[root@icinga2-master1.localdomain /root]# chmod 644 /var/lib/icinga2/certs/*.crt
```

The CA public and private key are stored in the `/var/lib/icinga2/ca` directory. Keep this path secure and include it in your backups.

6.13.6.4 Create Multiple Certificates

Use your preferred method to automate the certificate generation process.

```
[root@icinga2-master1.localdomain /var/lib/icinga2/certs]# for node in icinga2-master1.locald
information/base: Writing private key to 'icinga2-master1.localdomain.key'.
information/base: Writing certificate signing request to 'icinga2-master1.localdomain.csr'.
information/base: Writing private key to 'icinga2-master2.localdomain.key'.
information/base: Writing certificate signing request to 'icinga2-master2.localdomain.csr'.
information/base: Writing private key to 'icinga2-satellite1.localdomain.key'.
information/base: Writing certificate signing request to 'icinga2-satellite1.localdomain.csr'.

[root@icinga2-master1.localdomain /var/lib/icinga2/certs]# for node in icinga2-master1.locald
information/pki: Writing certificate to file 'icinga2-master1.localdomain.crt'.
information/pki: Writing certificate to file 'icinga2-master2.localdomain.crt'.
information/pki: Writing certificate to file 'icinga2-satellite1.localdomain.crt'.
```

Copy and move these certificates to the respective instances e.g. with SSH/SCP.

6.14 Automation

These hints should get you started with your own automation tools (Puppet, Ansible, Chef, Salt, etc.) or custom scripts for automated setup.

These are collected best practices from various community channels.

- Silent Windows setup
- Node Setup CLI command with parameters

If you prefer an alternate method, we still recommend leaving all the Icinga 2 features intact (e.g. `icinga2 feature enable api`). You should also use well known and documented default configuration file locations (e.g. `zones.conf`). This will tremendously help when someone is trying to help in the community channels.

6.14.1 Silent Windows Setup

If you want to install the client silently/unattended, use the `/qn` modifier. The installation should not trigger a restart, but if you want to be completely sure, you can use the `/norestart` modifier.

```
C:> msiexec /i C:\Icinga2-v2.5.0-x86.msi /qn /norestart
```

Once the setup is completed you can use the `node setup` cli command too.

6.14.2 Node Setup using CLI Parameters

Instead of using the `node wizard` CLI command, there is an alternative `node setup` command available which has some prerequisites.

Note: The CLI command can be used on Linux/Unix and Windows operating systems. The graphical Windows setup wizard actively uses these CLI commands.

6.14.2.1 Node Setup on the Master Node

In case you want to setup a master node you must add the `--master` parameter to the `node setup` CLI command. In addition to that the `--cn` can optionally be passed (defaults to the FQDN).

Parameter	Description
Common name (CN)	**Optional.** Specified with the `--cn` parameter. By convention this should be the host's FQDN. Defaults to the FQDN.
Listen on	**Optional.** Specified with the `--listen` parameter. Syntax is `host,port`.

Example:

```
[root@icinga2-master1.localdomain /]# icinga2 node setup --master
```

In case you want to bind the `ApiListener` object to a specific host/port you can specify it like this:

```
--listen 192.68.56.101,5665
```

6.14.2.2 Node Setup with Satellites/Clients

Note

The certificate location changed in v2.8 to `/var/lib/icinga2/certs`. Please read the upgrading chapter for more details.

Make sure that the `/var/lib/icinga2/certs` directory exists and is owned by the `icinga` user (or the user Icinga 2 is running as).

```
[root@icinga2-client1.localdomain /]# mkdir -p /var/lib/icinga2/certs
[root@icinga2-client1.localdomain /]# chown -R icinga:icinga /var/lib/icinga2/certs
```

First you'll need to generate a new local self-signed certificate. Pass the following details to the `pki new-cert` CLI command:

Parameter	Description
Common name (CN)	**Required.** By convention this should be the host's FQDN. Defaults to the FQDN.

Parameter	Description
Client certificate files	**Required.** These generated files will be put into the specified location (–key and –file). By convention this should be using `/var/lib/icinga2/certs` as directory.

Example:

```
[root@icinga2-client1.localdomain /]# icinga2 pki new-cert --cn icinga2-client1.localdomain \
--key /var/lib/icinga2/certs/icinga2-client1.localdomain.key \
--cert /var/lib/icinga2/certs/icinga2-client1.localdomain.crt
```

Request the master certificate from the master host (`icinga2-master1.localdomain`) and store it as `trusted-master.crt`. Review it and continue.

Pass the following details to the `pki save-cert` CLI command:

Parameter	Description
Client certificate files	**Required.** Pass the previously generated files using the `--key` and `--cert` parameters.
Trusted master certificate	**Required.** Store the master's certificate file. Manually verify that you're trusting it.
Master host	**Required.** FQDN or IP address of the master host.

Example:

```
[root@icinga2-client1.localdomain /]# icinga2 pki save-cert --key /var/lib/icinga2/certs/icing
--cert /var/lib/icinga2/certs/icinga2-client1.localdomain.crt \
--trustedcert /var/lib/icinga2/certs/trusted-master.crt \
--host icinga2-master1.localdomain
```

Continue with the additional node setup step. Specify a local endpoint

and zone name (`icinga2-client1.localdomain`) and set the master host (`icinga2-master1.localdomain`) as parent zone configuration. Specify the path to the previously stored trusted master certificate.

Pass the following details to the `node setup` CLI command:

Parameter	Description
Common name (CN)	**Optional.** Specified with the `--cn` parameter. By convention this should be the host's FQDN.
Request ticket	**Required.** Add the previously generated ticket number.
Trusted master certificate	**Required.** Add the previously fetched trusted master certificate (this step means that you've verified its origin).
Master endpoint	**Required.** Specify the master's endpoint name.
Client zone name	**Required.** Specify the client's zone name.
Master host	**Required.** FQDN or IP address of the master host.
Accept config	**Optional.** Whether this node accepts configuration sync from the master node (required for config sync mode).
Accept commands	**Optional.** Whether this node accepts command execution messages from the master node (required for command endpoint mode).

Parameter	Description
Global zones	**Optional.** Allows to specify more global zones in addition to `global-templates` and `director-global`.

Example for Icinga 2 v2.8:

```
[root@icinga2-client1.localdomain /]# icinga2 node setup --ticket ead2d570e18c78abf285d6b8552
--cn icinga2-client1.localdomain \
--endpoint icinga2-master1.localdomain \
--zone icinga2-client1.localdomain \
--master_host icinga2-master1.localdomain \
--trustedcert /var/lib/icinga2/certs/trusted-master.crt \
--accept-commands --accept-config
```

In case the client should connect to the master node, you'll need to modify the `--endpoint` parameter using the format `cn,host,port`:

```
--endpoint icinga2-master1.localdomain,192.168.56.101,5665
```

In case the client should know the additional global zone `linux-templates`, you'll need to set the `--global-zones` parameter.

```
--global_zones linux-templates
```

Restart Icinga 2 afterwards:

```
# service icinga2 restart
```

You can find additional best practices below.

Add an additional global zone. Please note the `>>` append mode.

```
[root@icinga2-client1.localdomain /]# cat <<EOF >>/etc/icinga2/zones.conf
object Zone "global-templates" {
  global = true
}
EOF
```

Note: Packages >= 2.8 provide this configuration by default.

If this client node is configured as remote command endpoint execution you can safely disable the `checker` feature. The `node setup` CLI command already disabled the `notification` feature.

```
[root@icinga2-client1.localdomain /]# icinga2 feature disable checker
```

Disable "conf.d" inclusion if this is a top down configured client.

```
[root@icinga2-client1.localdomain /]# sed -i 's/include_recursive "conf.d"/\/\/include_recurs
```

Optional: Add an ApiUser object configuration for remote troubleshooting.

```
[root@icinga2-client1.localdomain /]# cat <<EOF >/etc/icinga2/conf.d/api-users.conf
object ApiUser "root" {
  password = "clientsupersecretpassword"
  permissions = ["*"]
}
EOF
```

In case you've previously disabled the "conf.d" directory only add the file file `conf.d/api-users.conf`:

```
[root@icinga2-client1.localdomain /]# echo 'include "conf.d/api-users.conf"' >> /etc/icinga2/i
```

Finally restart Icinga 2.

```
[root@icinga2-client1.localdomain /]# systemctl restart icinga2
```

Your automation tool must then configure master node in the meantime. Add the global zone `global-templates` in case it did not exist.

```
# cat <<EOF >>/etc/icinga2/zones.conf
object Endpoint "icinga2-client1.localdomain" {
  //client connects itself
}

object Zone "icinga2-client1.localdomain" {
  endpoints = [ "icinga2-client1.localdomain" ]
  parent = "master"
}

object Zone "global-templates" {
  global = true
}
EOF
```

7 Additional Agent-based Checks

If the remote services are not directly accessible through the network, a local agent installation exposing the results to check queries can become handy.

7.1 SNMP

The SNMP daemon runs on the remote system and answers SNMP queries by plugin binaries. The Monitoring Plugins package ships the `check_snmp` plugin

binary, but there are plenty of existing plugins for specific use cases already around, for example monitoring Cisco routers.

The following example uses the SNMP ITL `CheckCommand` and just overrides the `snmp_oid` custom attribute. A service is created for all hosts which have the `snmp-community` custom attribute.

```
apply Service "uptime" {
  import "generic-service"

  check_command = "snmp"
  vars.snmp_oid = "1.3.6.1.2.1.1.3.0"
  vars.snmp_miblist = "DISMAN-EVENT-MIB"

  assign where host.vars.snmp_community != ""
}
```

Additional SNMP plugins are available using the Manubulon SNMP Plugins.

If no `snmp_miblist` is specified, the plugin will default to `ALL`. As the number of available MIB files on the system increases so will the load generated by this plugin if no `MIB` is specified. As such, it is recommended to always specify at least one `MIB`.

7.2 SSH

Calling a plugin using the SSH protocol to execute a plugin on the remote server fetching its return code and output. The `by_ssh` command object is part of the built-in templates and requires the `check_by_ssh` check plugin which is available in the Monitoring Plugins package.

```
object CheckCommand "by_ssh_swap" {
  import "by_ssh"

 vars.by_ssh_command = "/usr/lib/nagios/plugins/check_swap -w $by_ssh_swap_warn$ -c $by_ssh_s
  vars.by_ssh_swap_warn = "75%"
  vars.by_ssh_swap_crit = "50%"
}

object Service "swap" {
  import "generic-service"

  host_name = "remote-ssh-host"

  check_command = "by_ssh_swap"

  vars.by_ssh_logname = "icinga"
```

```
}
```

7.3 NSClient++

NSClient++ works on both Windows and Linux platforms and is well known for its magnificent Windows support. There are alternatives like the WMI interface, but using `NSClient++` will allow you to run local scripts similar to check plugins fetching the required output and performance counters.

You can use the `check_nt` plugin from the Monitoring Plugins project to query NSClient++. Icinga 2 provides the nscp check command for this:

Example:

```
object Service "disk" {
  import "generic-service"

  host_name = "remote-windows-host"

  check_command = "nscp"

  vars.nscp_variable = "USEDDISKSPACE"
  vars.nscp_params = "c"
  vars.nscp_warn = 70
  vars.nscp_crit = 80
}
```

For details on the `NSClient++` configuration please refer to the official documentation.

7.4 NSCA-NG

NSCA-ng provides a client-server pair that allows the remote sender to push check results into the Icinga 2 `ExternalCommandListener` feature.

Note

This addon works in a similar fashion like the Icinga 1.x distributed model. If you are looking for a real distributed architecture with Icinga 2, scroll down.

7.5 NRPE

NRPE runs as daemon on the remote client including the required plugins and command definitions. Icinga 2 calls the `check_nrpe` plugin binary in order to query the configured command on the remote client.

Note

The NRPE protocol is considered insecure and has multiple flaws in its design. Upstream is not willing to fix these issues.

In order to stay safe, please use the native Icinga 2 client instead.

The NRPE daemon uses its own configuration format in nrpe.cfg while `check_nrpe` can be embedded into the Icinga 2 `CheckCommand` configuration syntax.

You can use the `check_nrpe` plugin from the NRPE project to query the NRPE daemon. Icinga 2 provides the nrpe check command for this:

Example:

```
object Service "users" {
  import "generic-service"

  host_name = "remote-nrpe-host"

  check_command = "nrpe"
  vars.nrpe_command = "check_users"
}
```

nrpe.cfg:

```
command[check_users]=/usr/local/icinga/libexec/check_users -w 5 -c 10
```

If you are planning to pass arguments to NRPE using the `-a` command line parameter, make sure that your NRPE daemon has them supported and enabled.

Note

Enabling command arguments in NRPE is considered harmful and exposes a security risk allowing attackers to execute commands remotely. Details at seclists.org.

The plugin check command `nrpe` provides the `nrpe_arguments` custom attribute which expects either a single value or an array of values.

Example:

```
object Service "nrpe-disk-/" {
  import "generic-service"

  host_name = "remote-nrpe-host"

  check_command = "nrpe"
  vars.nrpe_command = "check_disk"
  vars.nrpe_arguments = [ "20%", "10%", "/" ]
}
```

Icinga 2 will execute the nrpe plugin like this:

```
/usr/lib/nagios/plugins/check_nrpe -H <remote-nrpe-host> -c 'check_disk' -a '20%' '10%' '/'
```

NRPE expects all additional arguments in an ordered fashion and interprets the first value as `$ARG1$` macro, the second value as `$ARG2$`, and so on.

nrpe.cfg:

```
command[check_disk]=/usr/local/icinga/libexec/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
```

Using the above example with `nrpe_arguments` the command executed by the NRPE daemon looks similar to that:

```
/usr/local/icinga/libexec/check_disk -w 20% -c 10% -p /
```

You can pass arguments in a similar manner to NSClient++ when using its NRPE supported check method.

7.6 Passive Check Results and SNMP Traps

SNMP Traps can be received and filtered by using SNMPTT and specific trap handlers passing the check results to Icinga 2.

Following the SNMPTT Format documentation and the Icinga external command syntax found here we can create generic services that can accommodate any number of hosts for a given scenario.

7.6.1 Simple SNMP Traps

A simple example might be monitoring host reboots indicated by an SNMP agent reset. Building the event to auto reset after dispatching a notification is important. Setup the manual check parameters to reset the event from an initial unhandled state or from a missed reset event.

Add a directive in `snmptt.conf`

```
EVENT coldStart .1.3.6.1.6.3.1.1.5.1 "Status Events" Normal
FORMAT Device reinitialized (coldStart)
EXEC echo "[$@] PROCESS_SERVICE_CHECK_RESULT;$A;Coldstart;2;The snmp agent has reinitialized."
SDESC
A coldStart trap signifies that the SNMPv2 entity, acting
in an agent role, is reinitializing itself and that its
configuration may have been altered.
EDESC
```

1. Define the `EVENT` as per your need.
2. Construct the `EXEC` statement with the service name matching your template applied to your *n* hosts. The host address inferred by SNMPTT will

be the correlating factor. You can have snmptt provide host names or ip addresses to match your Icinga convention.

Add an `EventCommand` configuration object for the passive service auto reset event.

```
object EventCommand "coldstart-reset-event" {
 command = [ SysconfDir + "/icinga2/conf.d/custom/scripts/coldstart_reset_event.sh" ]

  arguments = {
    "-i" = "$service.state_id$"
    "-n" = "$host.name$"
    "-s" = "$service.name$"
  }
}
```

Create the `coldstart_reset_event.sh` shell script to pass the expanded variable data in. The `$service.state_id$` is important in order to prevent an endless loop of event firing after the service has been reset.

```
#!/bin/bash

SERVICE_STATE_ID=""
HOST_NAME=""
SERVICE_NAME=""

show_help()
{
cat <<-EOF
    Usage: ${0##*/} [-h] -n HOST_NAME -s SERVICE_NAME
    Writes a coldstart reset event to the Icinga command pipe.

      -h                  Display this help and exit.
      -i SERVICE_STATE_ID The associated service state id.
      -n HOST_NAME        The associated host name.
      -s SERVICE_NAME     The associated service name.
EOF
}

while getopts "hi:n:s:" opt; do
    case "$opt" in
      h)
          show_help
          exit 0
          ;;
      i)
          SERVICE_STATE_ID=$OPTARG
          ;;
```

```
      n)
          HOST_NAME=$OPTARG
          ;;
      s)
          SERVICE_NAME=$OPTARG
          ;;
      '?')
          show_help
          exit 0
          ;;
      esac
done

if [ -z "$SERVICE_STATE_ID" ]; then
    show_help
    printf "\n  Error: -i required.\n"
    exit 1
fi

if [ -z "$HOST_NAME" ]; then
    show_help
    printf "\n  Error: -n required.\n"
    exit 1
fi

if [ -z "$SERVICE_NAME" ]; then
    show_help
    printf "\n  Error: -s required.\n"
    exit 1
fi

if [ "$SERVICE_STATE_ID" -gt 0 ]; then
   echo "[`date +%s`] PROCESS_SERVICE_CHECK_RESULT;$HOST_NAME;$SERVICE_NAME;0;Auto-reset (`da
fi
```

Finally create the `Service` and assign it:

```
apply Service "Coldstart" {
  import "generic-service-custom"

  check_command         = "dummy"
  event_command         = "coldstart-reset-event"

  enable_notifications  = 1
  enable_active_checks  = 0
  enable_passive_checks = 1
  enable_flapping       = 0
```

```
  volatile              = 1
  enable_perfdata       = 0

  vars.dummy_state      = 0
  vars.dummy_text       = "Manual reset."

  vars.sla              = "24x7"

 assign where (host.vars.os == "Linux" || host.vars.os == "Windows")
}
```

7.6.2 Complex SNMP Traps

A more complex example might be passing dynamic data from a traps varbind list for a backup scenario where the backup software dispatches status updates. By utilizing active and passive checks, the older freshness concept can be leveraged.

By defining the active check as a hard failed state, a missed backup can be reported. As long as the most recent passive update has occurred, the active check is bypassed.

Add a directive in `snmptt.conf`

```
EVENT enterpriseSpecific <YOUR OID> "Status Events" Normal
FORMAT Enterprise specific trap
EXEC echo "[$@] PROCESS_SERVICE_CHECK_RESULT;$A;$1;$2;$3" >> /var/run/icinga2/cmd/icinga2.cmd
SDESC
An enterprise specific trap.
The varbinds in order denote the Icinga service name, state and text.
EDESC
```

1. Define the `EVENT` as per your need using your actual oid.
2. The service name, state and text are extracted from the first three varbinds. This has the advantage of accommodating an unlimited set of use cases.

Create a `Service` for the specific use case associated to the host. If the host matches and the first varbind value is `Backup`, SNMPTT will submit the corresponding passive update with the state and text from the second and third varbind:

```
object Service "Backup" {
  import "generic-service-custom"

  host_name             = "host.domain.com"
  check_command         = "dummy"
```

```
  enable_notifications  = 1
  enable_active_checks  = 1
  enable_passive_checks = 1
  enable_flapping       = 0
  volatile              = 1
  max_check_attempts    = 1
  check_interval        = 87000
  enable_perfdata       = 0

  vars.sla              = "24x7"
  vars.dummy_state      = 2
  vars.dummy_text       = "No passive check result received."
}
```

8 Advanced Topics

This chapter covers a number of advanced topics. If you're new to Icinga, you can safely skip over things you're not interested in.

8.1 Downtimes

Downtimes can be scheduled for planned server maintenance or any other targeted service outage you are aware of in advance.

Downtimes will suppress any notifications, and may trigger other downtimes too. If the downtime was set by accident, or the duration exceeds the maintenance, you can manually cancel the downtime. Planned downtimes will also be taken into account for SLA reporting tools calculating the SLAs based on the state and downtime history.

Multiple downtimes for a single object may overlap. This is useful when you want to extend your maintenance window taking longer than expected. If there are multiple downtimes triggered for one object, the overall downtime depth will be greater than `1`.

If the downtime was scheduled after the problem changed to a critical hard state triggering a problem notification, and the service recovers during the downtime window, the recovery notification won't be suppressed.

8.1.1 Fixed and Flexible Downtimes

A `fixed` downtime will be activated at the defined start time, and removed at the end time. During this time window the service state will change to `NOT-OK`

and then actually trigger the downtime. Notifications are suppressed and the downtime depth is incremented.

Common scenarios are a planned distribution upgrade on your linux servers, or database updates in your warehouse. The customer knows about a fixed downtime window between 23:00 and 24:00. After 24:00 all problems should be alerted again. Solution is simple - schedule a `fixed` downtime starting at 23:00 and ending at 24:00.

Unlike a `fixed` downtime, a `flexible` downtime will be triggered by the state change in the time span defined by start and end time, and then last for the specified duration in minutes.

Imagine the following scenario: Your service is frequently polled by users trying to grab free deleted domains for immediate registration. Between 07:30 and 08:00 the impact will hit for 15 minutes and generate a network outage visible to the monitoring. The service is still alive, but answering too slow to Icinga 2 service checks. For that reason, you may want to schedule a downtime between 07:30 and 08:00 with a duration of 15 minutes. The downtime will then last from its trigger time until the duration is over. After that, the downtime is removed (may happen before or after the actual end time!).

8.1.2 Scheduling a downtime

You can schedule a downtime either by using the Icinga 2 API action schedule-downtime or by sending an external command.

8.1.2.1 Fixed Downtime

If the host/service changes into a NOT-OK state between the start and end time window, the downtime will be marked as `in effect` and increases the downtime depth counter.

```
   |       |         |
start      |        end
       trigger time
```

8.1.2.2 Flexible Downtime

A flexible downtime defines a time window where the downtime may be triggered from a host/service NOT-OK state change. It will then last until the specified time duration is reached. That way it can happen that the downtime end time is already gone, but the downtime ends at `trigger time + duration`.

```
   |       |         |
start      |        end               actual end time
           |--------------duration--------|
```

```
trigger time
```

8.1.3 Triggered Downtimes

This is optional when scheduling a downtime. If there is already a downtime scheduled for a future maintenance, the current downtime can be triggered by that downtime. This renders useful if you have scheduled a host downtime and are now scheduling a child host's downtime getting triggered by the parent downtime on `NOT-OK` state change.

8.1.4 Recurring Downtimes

ScheduledDowntime objects can be used to set up recurring downtimes for services.

Example:

```
apply ScheduledDowntime "backup-downtime" to Service {
  author = "icingaadmin"
  comment = "Scheduled downtime for backup"

  ranges = {
    monday = "02:00-03:00"
    tuesday = "02:00-03:00"
    wednesday = "02:00-03:00"
    thursday = "02:00-03:00"
    friday = "02:00-03:00"
    saturday = "02:00-03:00"
    sunday = "02:00-03:00"
  }

  assign where "backup" in service.groups
}
```

8.2 Comments

Comments can be added at runtime and are persistent over restarts. You can add useful information for others on repeating incidents (for example "last time syslog at 100% cpu on 17.10.2013 due to stale nfs mount") which is primarily accessible using web interfaces.

You can add a comment either by using the Icinga 2 API action add-comment or by sending an external command.

8.3 Acknowledgements

If a problem persists and notifications have been sent, you can acknowledge the problem. That way other users will get a notification that you're aware of the issue and probably are already working on a fix.

Note: Acknowledgements also add a new comment which contains the author and text fields.

You can send an acknowledgement either by using the Icinga 2 API action acknowledge-problem or by sending an external command.

8.3.1 Sticky Acknowledgements

The acknowledgement is removed if a state change occurs or if the host/service recovers (OK/Up state).

If you acknowledge a problem once you've received a `Critical` notification, the acknowledgement will be removed if there is a state transition to `Warning`.

```
OK -> WARNING -> CRITICAL -> WARNING -> OK
```

If you prefer to keep the acknowledgement until the problem is resolved (`OK` recovery) you need to enable the `sticky` parameter.

8.3.2 Expiring Acknowledgements

Once a problem is acknowledged it may disappear from your `handled problems` dashboard and no-one ever looks at it again since it will suppress notifications too.

This `fire-and-forget` action is quite common. If you're sure that a current problem should be resolved in the future at a defined time, you can define an expiration time when acknowledging the problem.

Icinga 2 will clear the acknowledgement when expired and start to re-notify, if the problem persists.

8.4 Time Periods

Time Periods define time ranges in Icinga where event actions are triggered, for example whether a service check is executed or not within the `check_period` attribute. Or a notification should be sent to users or not, filtered by the `period` and `notification_period` configuration attributes for `Notification` and `User` objects.

Note

If you are familiar with Icinga 1.x, these time period definitions are called `legacy timeperiods` in Icinga 2.

An Icinga 2 legacy timeperiod requires the `ITL` provided template `legacy-timeperiod`.

The `TimePeriod` attribute `ranges` may contain multiple directives, including weekdays, days of the month, and calendar dates. These types may override other types in your ranges dictionary.

The descending order of precedence is as follows:

- Calendar date (2008-01-01)
- Specific month date (January 1st)
- Generic month date (Day 15)
- Offset weekday of specific month (2nd Tuesday in December)
- Offset weekday (3rd Monday)
- Normal weekday (Tuesday)

If you don't set any `check_period` or `notification_period` attribute on your configuration objects, Icinga 2 assumes `24x7` as time period as shown below.

```
object TimePeriod "24x7" {
  import "legacy-timeperiod"

  display_name = "Icinga 2 24x7 TimePeriod"
  ranges = {
    "monday"    = "00:00-24:00"
    "tuesday"   = "00:00-24:00"
    "wednesday" = "00:00-24:00"
    "thursday"  = "00:00-24:00"
    "friday"    = "00:00-24:00"
    "saturday"  = "00:00-24:00"
    "sunday"    = "00:00-24:00"
  }
}
```

If your operation staff should only be notified during workhours, create a new timeperiod named `workhours` defining a work day from 09:00 to 17:00.

```
object TimePeriod "workhours" {
  import "legacy-timeperiod"

  display_name = "Icinga 2 8x5 TimePeriod"
  ranges = {
    "monday"    = "09:00-17:00"
    "tuesday"   = "09:00-17:00"
    "wednesday" = "09:00-17:00"
```

```
    "thursday"  = "09:00-17:00"
    "friday"    = "09:00-17:00"
  }
}
```

Furthermore if you wish to specify a notification period across midnight, you can define it the following way:

```
object Timeperiod "across-midnight" {
  import "legacy-timeperiod"

  display_name = "Nightly Notification"
  ranges = {
    "saturday" = "22:00-24:00"
    "sunday" = "00:00-03:00"
  }
}
```

Below you can see another example for configuring timeperiods across several days, weeks or months. This can be useful when taking components offline for a distinct period of time.

```
object Timeperiod "standby" {
  import "legacy-timeperiod"

  display_name = "Standby"
  ranges = {
    "2016-09-30 - 2016-10-30" = "00:00-24:00"
  }
}
```

Please note that the spaces before and after the dash are mandatory.

Once your time period is configured you can Use the `period` attribute to assign time periods to `Notification` and `Dependency` objects:

```
object Notification "mail" {
  import "generic-notification"

  host_name = "localhost"

  command = "mail-notification"
  users = [ "icingaadmin" ]
  period = "workhours"
}
```

8.4.1 Time Periods Inclusion and Exclusion

Sometimes it is necessary to exclude certain time ranges from your default time period definitions, for example, if you don't want to send out any notification during the holiday season, or if you only want to allow small time windows for executed checks.

The TimePeriod object provides the `includes` and `excludes` attributes to solve this issue. `prefer_includes` defines whether included or excluded time periods are preferred.

The following example defines a time period called `holidays` where notifications should be suppressed:

```
object TimePeriod "holidays" {
  import "legacy-timeperiod"

  ranges = {
    "january 1" = "00:00-24:00"                 //new year's day
    "july 4" = "00:00-24:00"                    //independence day
    "december 25" = "00:00-24:00"               //christmas
   "december 31" = "18:00-24:00"              //new year's eve (6pm+)
    "2017-04-16" = "00:00-24:00"                //easter 2017
   "monday -1 may" = "00:00-24:00"           //memorial day (last monday in may)
   "monday 1 september" = "00:00-24:00"       //labor day (1st monday in september)
   "thursday 4 november" = "00:00-24:00"      //thanksgiving (4th thursday in november)
  }
}
```

In addition to that the time period `weekends` defines an additional time window which should be excluded from notifications:

```
object TimePeriod "weekends-excluded" {
  import "legacy-timeperiod"

  ranges = {
    "saturday"  = "00:00-09:00,18:00-24:00"
    "sunday"    = "00:00-09:00,18:00-24:00"
  }
}
```

The time period `prod-notification` defines the default time ranges and adds the excluded time period names as an array.

```
object TimePeriod "prod-notification" {
  import "legacy-timeperiod"

  excludes = [ "holidays", "weekends-excluded" ]
```

```
  ranges = {
    "monday"    = "00:00-24:00"
    "tuesday"   = "00:00-24:00"
    "wednesday" = "00:00-24:00"
    "thursday"  = "00:00-24:00"
    "friday"    = "00:00-24:00"
    "saturday"  = "00:00-24:00"
    "sunday"    = "00:00-24:00"
  }
}
```

8.5 External Check Results

Hosts or services which do not actively execute a check plugin to receive the state and output are called "passive checks" or "external check results". In this scenario an external client or script is sending in check results.

You can feed check results into Icinga 2 with the following transport methods:

- process-check-result action available with the REST API (remote and local)
- External command sent via command pipe (local only)

Each time a new check result is received, the next expected check time is updated. This means that if there are no check result received from the external source, Icinga 2 will execute freshness checks.

> **Note**
>
> The REST API action allows to specify the `check_source` attribute which helps identifying the external sender. This is also visible in Icinga Web 2 and the REST API queries.

8.6 Check Result Freshness

In Icinga 2 active check freshness is enabled by default. It is determined by the `check_interval` attribute and no incoming check results in that period of time.

The threshold is calculated based on the last check execution time for actively executed checks:

```
(last check execution time + check interval) > current time
```

If this host/service receives check results from an external source, the threshold is based on the last time a check result was received:

```
(last check result time + check interval) > current time
```

Tip

The process-check-result REST API action allows to overrule the pre-defined check interval with a specified TTL in Icinga 2 v2.9+.

If the freshness checks fail, Icinga 2 will execute the defined check command.

Best practice is to define a dummy `check_command` which gets executed when freshness checks fail.

```
apply Service "external-check" {
  check_command = "dummy"
  check_interval = 1m

  /* Set the state to UNKNOWN (3) if freshness checks fail. */
  vars.dummy_state = 3

 /* Use a runtime function to retrieve the last check time and more details. */
  vars.dummy_text = {{
   var service = get_service(macro("$host.name$"), macro("$service.name$"))
    var lastCheck = DateTime(service.last_check).to_string()

   return "No check results received. Last result time: " + lastCheck
  }}

  assign where "external" in host.vars.services
}
```

References: get_service, macro, DateTime.

Example output in Icinga Web 2:

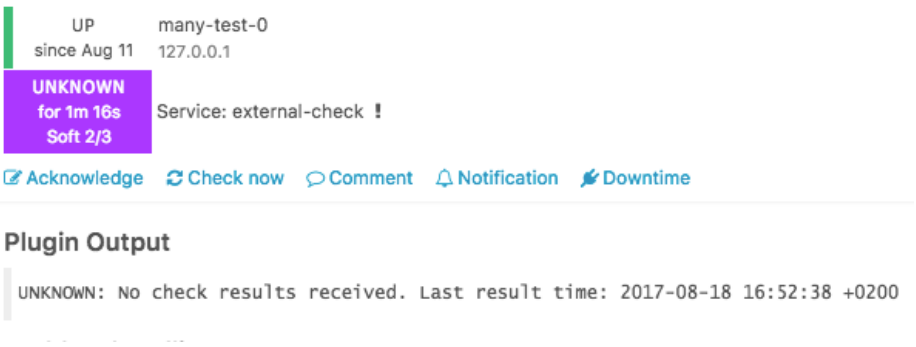


Figure 33: Icinga 2 Freshness Checks

8.7 Check Flapping

Icinga 2 supports optional detection of hosts and services that are "flapping".

Flapping occurs when a service or host changes state too frequently, which would result in a storm of problem and recovery notifications. With flapping detection enabled a flapping notification will be sent while other notifications are suppresed until it calms down after receiving the same status from checks a few times. Flapping detection can help detect

configuration problems (wrong thresholds), troublesome services, or network problems.

Flapping detection can be enabled or disabled using the `enable_flapping` attribute. The `flapping_threshold_high` and `flapping_threshold_low` attributes allows to specify the thresholds that control when a host or service is considered to be flapping.

The default thresholds are 30% for high and 25% for low. If the computed flapping value exceeds the high threshold a host or service is considered flapping until it drops below the low flapping threshold.

`FlappingStart` and `FlappingEnd` notifications will be sent out accordingly, if configured. See the chapter on notifications for details

> Note: There is no distinctions between hard and soft states with flapping. All state changes count and notifications will be sent out regardless of the objects state.

8.7.1 How it works

Icinga 2 saves the last 20 state changes for every host and service. See the graphic below:

All the states ware weighted, with the most recent one being worth the most (1.15) and the 20th the least (0.8). The states in between are fairly distributed. The final flapping value are the weighted state changes divided by the total count of 20.

In the example above, the added states would have a total value of 7.82 (`0.84 + 0.86 + 0.88 + 0.9 + 0.98 + 1.06 + 1.12 + 1.18`). This yields a flapping percentage of 39.1% (`7.82 / 20 * 100`). As the default upper flapping threshold is 30%, it would be considered flapping.

If the next seven check results then would not be state changes, the flapping percentage would fall below the lower threshold of 25% and therefore the host or service would recover from flapping.

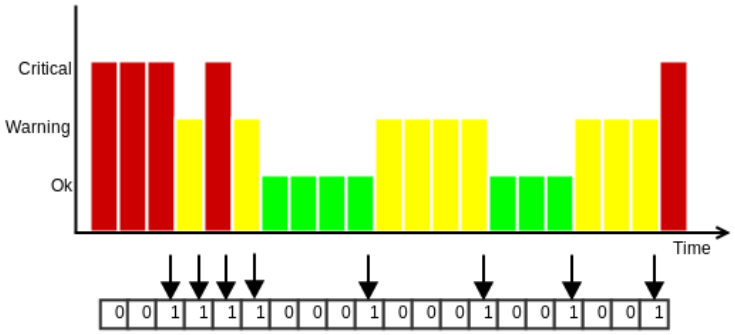


Figure 34: Icinga 2 Flapping State Timeline

8.8 Volatile Services

By default all services remain in a non-volatile state. When a problem occurs, the `SOFT` state applies and once `max_check_attempts` attribute is reached with the check counter, a `HARD` state transition happens. Notifications are only triggered by `HARD` state changes and are then re-sent defined by the `interval` attribute.

It may be reasonable to have a volatile service which stays in a `HARD` state type if the service stays in a `NOT-OK` state. That way each service recheck will automatically trigger a notification unless the service is acknowledged or in a scheduled downtime.

8.9 Monitoring Icinga 2

Why should you do that? Icinga and its components run like any other service application on your server. There are predictable issues such as "disk space is running low" and your monitoring suffers from just that.

You would also like to ensure that features and backends are running and storing required data. Be it the database backend where Icinga Web 2 presents fancy dashboards, forwarded metrics to Graphite or InfluxDB or the entire distributed setup.

This list isn't complete but should help with your own setup. Windows client specific checks are highlighted.

Type	Description	Plugins and CheckCommands
System	Filesystem	disk, disk-windows (Windows Client)

Type	Description	Plugins and CheckCommands
System	Memory, Swap	mem, swap, memory (Windows Client)
System	Hardware	hpasm, ipmi-sensor
System	Virtualization	VMware, esxi_hardware
System	Processes	procs, service-windows (Windows Client)
System	System Activity Reports	check_sar_perf
System	I/O	iostat
System	Network interfaces	nwc_health, interfaces
System	Users	users, users-windows (Windows Client)
System	Logs	Forward them to Elastic Stack or Graylog and add your own alerts.
System	NTP	ntp_time
System	Updates	apt, yum
Icinga	Status & Stats	icinga (more below)
Icinga	Cluster & Clients	health checks
Database	MySQL	mysql_health
Database	PostgreSQL	postgres
Database	Housekeeping	Check the database size and growth and analyse metrics to examine trends.
Database	DB IDO	ido (more below)
Webserver	Apache2, Nginx, etc.	http, apache_status, nginx_status
Webserver	Certificates	http
Webserver	Authorization	http
Notifications	Mail (queue)	smtp, mailq
Notifications	SMS (GSM modem)	check_sms3_status
Notifications	Messengers, Cloud services	XMPP, Twitter, IRC, Telegram, PagerDuty, VictorOps, etc.
Metrics	PNP, RRDTool	check_pnp_rrds checks for stale RRD files.
Metrics	Graphite	graphite
Metrics	InfluxDB	check_influxdb
Metrics	Elastic Stack	elasticsearch, Elastic Stack integration
Metrics	Graylog	Graylog integration

The icinga CheckCommand provides metrics for the runtime stats of Icinga 2. You can forward them to your preferred graphing solution. If you require more metrics you can also query the REST API and write your own custom check plugin. Or you keep using the built-in object accessor functions to calculate

stats in-memory.

There is a built-in ido check available for DB IDO MySQL/PostgreSQL which provides additional metrics for the IDO database.

```
apply Service "ido-mysql" {
  check_command = "ido"

  vars.ido_type = "IdoMysqlConnection"
 vars.ido_name = "ido-mysql" //the name defined in /etc/icinga2/features-enabled/ido-mysql.co

  assign where match("master*.localdomain", host.name)
}
```

More specific database queries can be found in the DB IDO chapter.

Distributed setups should include specific health checks. You might also want to add additional checks for SSL certificate expiration.

8.10 Advanced Configuration Hints

8.10.1 Advanced Use of Apply Rules

Apply rules can be used to create a rule set which is entirely based on host objects and their attributes. In addition to that apply for and custom attribute override extend the possibilities.

The following example defines a dictionary on the host object which contains configuration attributes for multiple web servers. This then used to add three checks:

- A `ping4` check using the local IP `address` of the web server.
- A `tcp` check querying the TCP port where the HTTP service is running on.
- If the `url` key is defined, the third apply for rule will create service objects using the `http` CheckCommand. In addition to that you can optionally define the `ssl` attribute which enables HTTPS checks.

Host definition:

```
object Host "webserver01" {
  import "generic-host"
  address = "192.168.56.200"
  vars.os = "Linux"

  vars.webserver = {
    instance["status"] = {
      address = "192.168.56.201"
      port = "80"
```

```
      url = "/status"
    }
    instance["tomcat"] = {
      address = "192.168.56.202"
      port = "8080"
    }
    instance["icingaweb2"] = {
      address = "192.168.56.210"
      port = "443"
      url = "/icingaweb2"
      ssl = true
    }
  }
}
```

Service apply for definitions:

```
apply Service "webserver_ping" for (instance => config in host.vars.webserver.instance) {
  display_name = "webserver_" + instance
  check_command = "ping4"

  vars.ping_address = config.address

  assign where host.vars.webserver.instance
}

apply Service "webserver_port" for (instance => config in host.vars.webserver.instance) {
  display_name = "webserver_" + instance + "_" + config.port
  check_command = "tcp"

  vars.tcp_address = config.address
  vars.tcp_port = config.port

  assign where host.vars.webserver.instance
}

apply Service "webserver_url" for (instance => config in host.vars.webserver.instance) {
  display_name = "webserver_" + instance + "_" + config.url
  check_command = "http"

  vars.http_address = config.address
  vars.http_port = config.port
  vars.http_uri = config.url

  if (config.ssl) {
    vars.http_ssl = config.ssl
  }
```

```
  assign where config.url != ""
}
```

The variables defined in the host dictionary are not using the typical custom attribute prefix recommended for CheckCommand parameters. Instead they are re-used for multiple service checks in this example. In addition to defining check parameters this way, you can also enrich the `display_name` attribute with more details. This will be shown in in Icinga Web 2 for example.

8.10.2 Use Functions in Object Configuration

There is a limited scope where functions can be used as object attributes such as:

- As value for Custom Attributes
- Returning boolean expressions for set_if inside command arguments
- Returning a command array inside command objects

The other way around you can create objects dynamically using your own global functions.

> **Note**
>
> Functions called inside command objects share the same global scope as runtime macros. Therefore you can access host custom attributes like `host.vars.os`, or any other object attribute from inside the function definition used for set_if or command.

Tips when implementing functions:

- Use log() to dump variables. You can see the output inside the `icinga2.log` file depending in your log severity
- Use the `icinga2 console` to test basic functionality (e.g. iterating over a dictionary)
- Build them step-by-step. You can always refactor your code later on.

8.10.2.1 Register and Use Global Functions

Functions can be registered into the global scope. This allows custom functions being available in objects and other functions. Keep in mind that these functions are not marked as side-effect-free and as such are not available via the REST API.

Add a new configuration file `functions.conf` and include it into the icinga2.conf configuration file in the very beginning, e.g. after `constants.conf`. You can also manage global functions inside `constants.conf` if you prefer.

The following function converts a given state parameter into a returned string value. The important bits for registering it into the global scope are:

- `globals.<unique_function_name>` adds a new globals entry.
- `function()` specifies that a call to `state_to_string()` executes a function.
- Function parameters are defined inside the `function()` definition.

```
globals.state_to_string = function(state) {
  if (state == 2) {
    return "Critical"
  } else if (state == 1) {
    return "Warning"
  } else if (state == 0) {
    return "OK"
  } else if (state == 3) {
    return "Unknown"
  } else {
   log(LogWarning, "state_to_string", "Unknown state " + state + " provided.")
  }
}
```

The else-condition allows for better error handling. This warning will be shown in the Icinga 2 log file once the function is called.

> **Note**
>
> If these functions are used in a distributed environment, you must ensure to deploy them everywhere needed.

In order to test-drive the newly created function, restart Icinga 2 and use the debug console to connect to the REST API.

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/'
Icinga 2 (version: v2.8.1-373-g4bea6d25c)
<1> => globals.state_to_string(1)
"Warning"
<2> => state_to_string(2)
"Critical"
```

You can see that this function is now registered into the global scope. The function call `state_to_string()` can be used in any object at static config compile time or inside runtime lambda functions.

The following service object example uses the service state and converts it to string output. The function definition is not optimized and is enrolled for better readability including a log message.

```
object Service "state-test" {
  check_command = "dummy"
  host_name = NodeName
```

```
  vars.dummy_state = 2

  vars.dummy_text = {{
    var h = macro("$host.name$")
    var s = macro("$service.name$")

    var state = get_service(h, s).state

   log(LogInformation, "dummy_state", "Host: " + h + " Service: " + s + " State: " + state)

    return state_to_string(state)
  }}
}
```

8.10.2.2 Use Custom Functions as Attribute

To use custom functions as attributes, the function must be defined in a slightly unexpected way. The following example shows how to assign values depending on group membership. All hosts in the `slow-lan` host group use 300 as value for `ping_wrta`, all other hosts use 100.

```
globals.group_specific_value = function(group, group_value, non_group_value) {
    return function() use (group, group_value, non_group_value) {
        if (group in host.groups) {
            return group_value
        } else {
            return non_group_value
        }
    }
}

apply Service "ping4" {
    import "generic-service"
    check_command = "ping4"

    vars.ping_wrta = group_specific_value("slow-lan", 300, 100)
    vars.ping_crta = group_specific_value("slow-lan", 500, 200)

    assign where true
}
```

8.10.2.3 Use Functions in Assign Where Expressions

If a simple expression for matching a name or checking if an item exists in an array or dictionary does not fit, you should consider writing your own global

functions. You can call them inside `assign where` and `ignore where` expressions for apply rules or group assignments just like any other global functions for example match.

The following example requires the host `myprinter` being added to the host group `printers-lexmark` but only if the host uses a template matching the name `lexmark*`.

```
template Host "lexmark-printer-host" {
  vars.printer_type = "Lexmark"
}

object Host "myprinter" {
  import "generic-host"
  import "lexmark-printer-host"

  address = "192.168.1.1"
}

/* register a global function for the assign where call */
globals.check_host_templates = function(host, search) {
 /* iterate over all host templates and check if the search matches */
  for (tmpl in host.templates) {
    if (match(search, tmpl)) {
      return true
    }
  }

  /* nothing matched */
  return false
}

object HostGroup "printers-lexmark" {
  display_name = "Lexmark Printers"
  /* call the global function and pass the arguments */
  assign where check_host_templates(host, "lexmark*")
}
```

Take a different more complex example: All hosts with the custom attribute `vars_app` as nested dictionary should be added to the host group `ABAP-app-server`. But only if the `app_type` for all entries is set to `ABAP`.

It could read as wildcard match for nested dictionaries:

```
where host.vars.vars_app["*"].app_type == "ABAP"
```

The solution for this problem is to register a global function which checks the `app_type` for all hosts with the `vars_app` dictionary.

```
object Host "appserver01" {
  check_command = "dummy"
  vars.vars_app["ABC"] = { app_type = "ABAP" }
}
object Host "appserver02" {
  check_command = "dummy"
  vars.vars_app["DEF"] = { app_type = "ABAP" }
}

globals.check_app_type = function(host, type) {
 /* ensure that other hosts without the custom attribute do not match */
  if (typeof(host.vars.vars_app) != Dictionary) {
    return false
  }

  /* iterate over the vars_app dictionary */
  for (key => val in host.vars.vars_app) {
   /* if the value is a dictionary and if contains the app_type being the requested type */
    if (typeof(val) == Dictionary && val.app_type == type) {
      return true
    }
  }

  /* nothing matched */
  return false
}

object HostGroup "ABAP-app-server" {
  assign where check_app_type(host, "ABAP")
}
```

8.10.2.4 Use Functions in Command Arguments set_if

The `set_if` attribute inside the command arguments definition in the CheckCommand object definition is primarily used to evaluate whether the command parameter should be set or not.

By default you can evaluate runtime macros for their existence. If the result is not an empty string, the command parameter is passed. This becomes fairly complicated when want to evaluate multiple conditions and attributes.

The following example was found on the community support channels. The user had defined a host dictionary named `compellent` with the key `disks`. This was then used inside service apply for rules.

```
object Host "dict-host" {
  check_command = "check_compellent"
```

```
  vars.compellent["disks"] = {
    file = "/var/lib/check_compellent/san_disks.0.json",
    checks = ["disks"]
  }
}
```

The more significant problem was to only add the command parameter `--disk` to the plugin call when the dictionary `compellent` contains the key `disks`, and omit it if not found.

By defining `set_if` as abbreviated lambda function and evaluating the host custom attribute `compellent` containing the `disks` this problem was solved like this:

```
object CheckCommand "check_compellent" {
  command   = [ "/usr/bin/check_compellent" ]
  arguments   = {
    "--disks"  = {
      set_if = {{
        var host_vars = host.vars
        log(host_vars)
        var compel = host_vars.compellent
        log(compel)
        compel.contains("disks")
      }}
    }
  }
}
```

This implementation uses the dictionary type method contains and will fail if `host.vars.compellent` is not of the type `Dictionary`. Therefore you can extend the checks using the typeof function.

You can test the types using the `icinga2 console`:

```
# icinga2 console
Icinga (version: v2.3.0-193-g3eb55ad)
<1> => srv_vars.compellent["check_a"] = { file="outfile_a.json", checks = [ "disks", "fans" ] }
null
<2> => srv_vars.compellent["check_b"] = { file="outfile_b.json", checks = [ "power", "voltages"
null
<3> => typeof(srv_vars.compellent)
type 'Dictionary'
<4> =>
```

The more programmatic approach for `set_if` could look like this:

```
    "--disks" = {
      set_if = {{
        var srv_vars = service.vars
```

```
        if(len(srv_vars) > 0) {
          if (typeof(srv_vars.compellent) == Dictionary) {
            return srv_vars.compellent.contains("disks")
          } else {
        log(LogInformationen, "checkcommand set_if", "custom attribute compellent_checks is no
            return false
          }
        } else {
       log(LogWarning, "checkcommand set_if", "empty custom attributes")
          return false
        }
      }}
    }
```

8.10.2.5 Use Functions as Command Attribute

This comes in handy for NotificationCommands or EventCommands which does not require a returned checkresult including state/output.

The following example was taken from the community support channels. The requirement was to specify a custom attribute inside the notification apply rule and decide which notification script to call based on that.

```
object User "short-dummy" {
}

object UserGroup "short-dummy-group" {
  assign where user.name == "short-dummy"
}

apply Notification "mail-admins-short" to Host {
   import "mail-host-notification"
   command = "mail-host-notification-test"
   user_groups = [ "short-dummy-group" ]
   vars.short = true
   assign where host.vars.notification.mail
}
```

The solution is fairly simple: The `command` attribute is implemented as function returning an array required by the caller Icinga 2. The local variable `mailscript` sets the default value for the notification scrip location. If the notification custom attribute `short` is set, it will override the local variable `mailscript` with a new value. The `mailscript` variable is then used to compute the final notification command array being returned.

You can omit the `log()` calls, they only help debugging.

```
object NotificationCommand "mail-host-notification-test" {
```

```
  command = {{
    log("command as function")
    var mailscript = "mail-host-notification-long.sh"
    if (notification.vars.short) {
       mailscript = "mail-host-notification-short.sh"
    }
    log("Running command")
    log(mailscript)

    var cmd = [ SysconfDir + "/icinga2/scripts/" + mailscript ]
    log(LogCritical, "me", cmd)
    return cmd
  }}

  env = {
  }
}
```

8.10.3 Access Object Attributes at Runtime

The Object Accessor Functions can be used to retrieve references to other objects by name.

This allows you to access configuration and runtime object attributes. A detailed list can be found here.

8.10.3.1 Access Object Attributes at Runtime: Cluster Check

This is a simple cluster example for accessing two host object states and calculating a virtual cluster state and output:

```
object Host "cluster-host-01" {
  check_command = "dummy"
  vars.dummy_state = 2
  vars.dummy_text = "This host is down."
}

object Host "cluster-host-02" {
  check_command = "dummy"
  vars.dummy_state = 0
  vars.dummy_text = "This host is up."
}

object Host "cluster" {
  check_command = "dummy"
  vars.cluster_nodes = [ "cluster-host-01", "cluster-host-02" ]
```

```
  vars.dummy_state = {{
    var up_count = 0
    var down_count = 0
    var cluster_nodes = macro("$cluster_nodes$")

    for (node in cluster_nodes) {
      if (get_host(node).state > 0) {
        down_count += 1
      } else {
        up_count += 1
      }
    }

    if (up_count >= down_count) {
      return 0 //same up as down -> UP
    } else {
      return 2 //something is broken
    }
  }}

  vars.dummy_text = {{
    var output = "Cluster hosts:\n"
    var cluster_nodes = macro("$cluster_nodes$")

    for (node in cluster_nodes) {
    output += node + ": " + get_host(node).last_check_result.output + "\n"
    }

    return output
  }}
}
```

8.10.3.2 Time Dependent Thresholds

The following example sets time dependent thresholds for the load check based on the current time of the day compared to the defined time period.

```
object TimePeriod "backup" {
  import "legacy-timeperiod"

  ranges = {
    monday = "02:00-03:00"
    tuesday = "02:00-03:00"
    wednesday = "02:00-03:00"
    thursday = "02:00-03:00"
```

```
    friday = "02:00-03:00"
    saturday = "02:00-03:00"
    sunday = "02:00-03:00"
  }
}

object Host "webserver-with-backup" {
  check_command = "hostalive"
  address = "127.0.0.1"
}

object Service "webserver-backup-load" {
  check_command = "load"
  host_name = "webserver-with-backup"

  vars.load_wload1 = {{
    if (get_time_period("backup").is_inside) {
      return 20
    } else {
      return 5
    }
  }}
  vars.load_cload1 = {{
    if (get_time_period("backup").is_inside) {
      return 40
    } else {
      return 10
    }
  }}
}
```

8.11 Advanced Value Types

In addition to the default value types Icinga 2 also uses a few other types to represent its internal state. The following types are exposed via the API.

8.11.1 CheckResult

Name	Type	Description
exit_status	Number	The exit status returned by the check execution.
output	String	The check output.
performance_data	Array	Array of performance data values.

Name	Type	Description
check_source	String	Name of the node executing the check.
state	Number	The current state (0 = OK, 1 = WARNING, 2 = CRITICAL, 3 = UNKNOWN).
command	Value	Array of command with shell-escaped arguments or command line string.
execution_start	Timestamp	Check execution start time (as a UNIX timestamp).
execution_end	Timestamp	Check execution end time (as a UNIX timestamp).
schedule_start	Timestamp	Scheduled check execution start time (as a UNIX timestamp).
schedule_end	Timestamp	Scheduled check execution end time (as a UNIX timestamp).
active	Boolean	Whether the result is from an active or passive check.
vars_before	Dictionary	Internal attribute used for calculations.
vars_after	Dictionary	Internal attribute used for calculations.
ttl	Number	Time-to-live duration in seconds for this check result. The next expected check result is `now + ttl` where freshness checks are executed.

8.11.2 PerfdataValue

Icinga 2 parses performance data strings returned by check plugins and makes the information available to external interfaces (e.g. GraphiteWriter or the Icinga 2 API).

Name	Type	Description
label	String	Performance data label.
value	Number	Normalized performance data value without unit.

Name	Type	Description
counter	Boolean	Enabled if the original value contains `c` as unit. Defaults to `false`.
unit	String	Unit of measurement (`seconds`, `bytes`. `percent`) according to the plugin API.
crit	Value	Critical threshold value.
warn	Value	Warning threshold value.
min	Value	Minimum value returned by the check.
max	Value	Maximum value returned by the check.

9 Config Object Types

This chapter provides an overview of all available config object types which can be instantiated using the `object` keyword.

Additional details on configuration and runtime attributes and their description are explained here too.

The attributes need to have a specific type value. Many of them are explained in this chapter already. You should note that the `Timestamp` type is a `Number`. In addition to that `Object name` is an object reference to an existing object name as `String` type.

Configuration objects share these runtime attributes which cannot be modified by the user. You can access these attributes using the Icinga 2 API.

Name	Type	Description
version	Number	Timestamp when the object was created or modified. Synced throughout cluster nodes.
type	String	Object type.
original_attributes	Dictionary	Original values of object attributes modified at runtime.
active	Boolean	Object is active (e.g. a service being checked).

Name	Type	Description
paused	Boolean	Object has been paused at runtime (e.g. IdoMysqlConnection. Defaults to `false`.
templates	Array	Templates imported on object compilation.
package	String	Configuration package name this object belongs to. Local configuration is set to `_etc`, runtime created objects use `_api`.
source_location	Dictionary	Location information where the configuration files are stored.

9.1 ApiListener

ApiListener objects are used for distributed monitoring setups and API usage specifying the certificate files used for ssl authorization and additional restrictions. This configuration object is available as api feature.

The `TicketSalt` constant must be defined in constants.conf.

Example:

```
object ApiListener "api" {
  accept_commands = true
  accept_config = true

  ticket_salt = TicketSalt
}
```

Configuration Attributes:

Name	Type	Description
cert_path	String	**Deprecated.** Path to the public key.
key_path	String	**Deprecated.** Path to the private key.
ca_path	String	**Deprecated.** Path to the CA certificate file.

Name	Type	Description
ticket_salt	String	**Optional.** Private key for CSR auto-signing. **Required** for a signing master instance.
crl_path	String	**Optional.** Path to the CRL file.
bind_host	String	**Optional.** The IP address the api listener should be bound to. Defaults to `0.0.0.0`.
bind_port	Number	**Optional.** The port the api listener should be bound to. Defaults to `5665`.
accept_config	Boolean	**Optional.** Accept zone configuration. Defaults to `false`.
accept_commands	Boolean	**Optional.** Accept remote commands. Defaults to `false`.
cipher_list	String	**Optional.** Cipher list that is allowed. For a list of available ciphers run `openssl ciphers`. Defaults to `ALL:!LOW:!WEAK:!MEDIUM:!EXP:!NULL`.
tls_protocolmin	String	**Optional.** Minimum TLS protocol version. Must be one of `TLSv1`, `TLSv1.1` or `TLSv1.2`. Defaults to `TLSv1`.
access_control_allow_origin	Array	**Optional.** Specifies an array of origin URLs that may access the API. (MDN docs)
access_control_allow_credentials	Boolean	**Optional.** Indicates whether or not the actual request can be made using credentials. Defaults to `true`. (MDN docs)

Name	Type	Description
access_control_allow_headers	String	**Optional.** Used in response to a preflight request to indicate which HTTP headers can be used when making the actual request. Defaults to `Authorization`. (MDN docs)
access_control_allow_methods	String	**Optional.** Used in response to a preflight request to indicate which HTTP methods can be used when making the actual request. Defaults to `GET, POST, PUT, DELETE`. (MDN docs)

The ApiListener type expects its certificate files to be in the following locations:

Type	Location
Private key	`LocalStateDir + "/lib/icinga2/certs/" + NodeName + ".key"`
Certificate file	`LocalStateDir + "/lib/icinga2/certs/" + NodeName + ".crt"`
CA certificate file	`LocalStateDir + "/lib/icinga2/certs/ca.crt"`

If the deprecated attributes `cert_path`, `key_path` and/or `ca_path` are specified Icinga 2 copies those files to the new location in `LocalStateDir + "/lib/icinga2/certs"` unless the file(s) there are newer.

Please check the upgrading chapter for more details.

While Icinga 2 and the underlying OpenSSL library use sane and secure defaults, the attributes `cipher_list` and `tls_protocolmin` can be used to increase communication security. A good source for a more secure configuration is provided by the Mozilla Wiki. Ensure to use the same configuration for both attributes on **all** endpoints to avoid communication problems which requires to use `cipher_list` compatible with the endpoint using the oldest version of the OpenSSL library. If using other tools to connect to the API ensure also compatibility with them as this setting affects not only inter-cluster communcation

but also the REST API.

9.2 ApiUser

ApiUser objects are used for authentication against the Icinga 2 API.

Example:

```
object ApiUser "root" {
  password = "mysecretapipassword"
  permissions = [ "*" ]
}
```

Configuration Attributes:

Name	Type	Description
password	String	**Optional.** Password string. Note: This attribute is hidden in API responses.
hashed_password	String	**Optional.** A hashed password string in the form of /etc/shadow. Note: This attribute is hidden in API responses.
client_cn	String	**Optional.** Client Common Name (CN).
permissions	Array	**Required.** Array of permissions. Either as string or dictionary with the keys `permission` and `filter`. The latter must be specified as function.

Available permissions are explained in the API permissions chapter.

9.3 CheckCommand

A check command definition. Additional default command custom attributes can be defined here.

> **Note**
>
> Icinga 2 versions < 2.6.0 require the import of the plugin-check-command template.

Example:

```
object CheckCommand "http" {
  command = [ PluginDir + "/check_http" ]

  arguments = {
    "-H" = "$http_vhost$"
    "-I" = "$http_address$"
    "-u" = "$http_uri$"
    "-p" = "$http_port$"
    "-S" = {
      set_if = "$http_ssl$"
    }
    "--sni" = {
      set_if = "$http_sni$"
    }
    "-a" = {
      value = "$http_auth_pair$"
    description = "Username:password on sites with basic authentication"
    }
    "--no-body" = {
      set_if = "$http_ignore_body$"
    }
    "-r" = "$http_expect_body_regex$"
    "-w" = "$http_warn_time$"
    "-c" = "$http_critical_time$"
    "-e" = "$http_expect$"
  }

  vars.http_address = "$address$"
  vars.http_ssl = false
  vars.http_sni = false
}
```

Configuration Attributes:

Name	Type	Description
command	Array	**Required.** The command. This can either be an array of individual command arguments. Alternatively a string can be specified in which case the shell interpreter (usually /bin/sh) takes care of parsing the command. When using the "arguments" attribute this must be an array. Can be specified as function for advanced implementations.
env	Dictionary	**Optional.** A dictionary of macros which should be exported as environment variables prior to executing the command.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this command.
timeout	Duration	**Optional.** The command timeout in seconds. Defaults to `1m`.
arguments	Dictionary	**Optional.** A dictionary of command arguments.

9.3.1 CheckCommand Arguments

Command arguments can be defined as key-value-pairs in the `arguments` dictionary. If the argument requires additional configuration, for example a `description` attribute or an optional condition, the value can be defined as dictionary specifying additional options.

Service:

```
vars.x_val = "My command argument value."
vars.have_x = "true"
```

CheckCommand:

```
arguments = {
  "-X" = {
    value = "$x_val$"
```

```
    key = "-Xnew"       /* optional, set a new key identifier */
    description = "My plugin requires this argument for doing X."
    required = false    /* optional, no error if not set */
    skip_key = false    /* always use "-X <value>" */
   set_if = "$have_x$" /* only set if variable defined and resolves to a numeric value. String va
    order = -1          /* first position */
   repeat_key = true  /* if `value` is an array, repeat the key as parameter: ... 'key' 'value[0]
  }
  "-Y" = {
    value = "$y_val$"
    description = "My plugin requires this argument for doing Y."
    required = false    /* optional, no error if not set */
    skip_key = true     /* don't prefix "-Y" only use "<value>" */
   set_if = "$have_y$" /* only set if variable defined and resolves to a numeric value. String va
    order = 0           /* second position */
   repeat_key = false  /* if `value` is an array, do not repeat the key as parameter: ... 'key' 'v
  }
}
```

Name	Type	Description
value	String/Function	Optional argument value set by a runtime macro string or a function call.
key	String	Optional argument key overriding the key identifier.
description	String	Optional argument description.
required	Boolean	Required argument. Execution error if not set. Defaults to false (optional).
skip_key	Boolean	Use the value as argument and skip the key.
set_if	String/Function	Argument is added if the runtime macro string resolves to a defined numeric or boolean value. String values are not supported. Function calls returning a value are supported too.
order	Number	Set if multiple arguments require a defined argument order.

Name	Type	Description
repeat_key	Boolean	If the argument value is an array, repeat the argument key, or not. Defaults to true (repeat).

Argument order:

```
..., -3, -2, -1, <un-ordered keys>, 1, 2, 3, ...
```

Argument array with `repeat_key = true`:

```
'key' 'value[0]' 'key' 'value[1]' 'key' 'value[2]'
```

Argument array with `repeat_key = false`:

```
'key' 'value[0]' 'value[1]' 'value[2]'
```

9.4 CheckerComponent

The checker component is responsible for scheduling active checks. This configuration object is available as checker feature.

Example:

```
object CheckerComponent "checker" { }
```

Configuration Attributes:

Name	Type	Description
concurrent_checks	Number	**Optional and deprecated.** The maximum number of concurrent checks. Was replaced by global constant `MaxConcurrentChecks` which will be set if you still use `concurrent_checks`.

9.5 CheckResultReader

Reads Icinga 1.x check result files from a directory. This functionality is provided to help existing Icinga 1.x users and might be useful for migration scenarios.

Example:

```
object CheckResultReader "reader" {
```

```
  spool_dir = "/data/check-results"
}
```

Configuration Attributes:

Name	Type	Description
spool_dir	String	**Optional.** The directory which contains the check result files. Defaults to LocalStateDir + "/lib/icinga2/spool/checkresults/".

9.6 Comment

Comments created at runtime are represented as objects. Note: This is for reference only. You can create comments with the add-comment API action.

Example:

```
object Comment "localhost!my-comment" {
  host_name = "localhost"
  author = "icingaadmin"
  text = "This is a comment."
}
```

Configuration Attributes:

Name	Type	Description
host_name	Object name	**Required.** The name of the host this comment belongs to.
service_name	Object name	**Optional.** The short name of the service this comment belongs to. If omitted, this comment object is treated as host comment.
author	String	**Required.** The author's name.
text	String	**Required.** The comment text.
entry_time	Timestamp	**Optional.** The UNIX timestamp when this comment was added.

Name	Type	Description
entry_type	Number	**Optional.** The comment type (`User` = 1, `Downtime` = 2, `Flapping` = 3, `Acknowledgement` = 4).
expire_time	Timestamp	**Optional.** The comment's expire time as UNIX timestamp.
persistent	Boolean	**Optional.** Only evaluated for `entry_type` Acknowledgement. `true` does not remove the comment when the acknowledgement is removed.

9.7 CompatLogger

Writes log files in a format that's compatible with Icinga 1.x. This configuration object is available as compatlog feature.

Example:

```
object CompatLogger "compatlog" {
  log_dir = "/var/log/icinga2/compat"
  rotation_method = "DAILY"
}
```

Configuration Attributes:

Name	Type	Description
log_dir	String	**Optional.** Path to the compat log directory. Defaults to LocalStateDir + "/log/icinga2/compat".
rotation_method	String	**Optional.** Specifies when to rotate log files. Can be one of "HOURLY", "DAILY", "WEEKLY" or "MONTHLY". Defaults to "HOURLY".

9.8 Dependency

Dependency objects are used to specify dependencies between hosts and services. Dependencies can be defined as Host-to-Host, Service-to-Service, Service-to-Host, or Host-to-Service relations.

> **Best Practice**
>
> Rather than creating a `Dependency` object for a specific host or service it is usually easier to just create a `Dependency` template and use the `apply` keyword to assign the dependency to a number of hosts or services. Use the `to` keyword to set the specific target type for `Host` or `Service`. Check the dependencies chapter for detailed examples.

Service-to-Service Example:

```
object Dependency "webserver-internet" {
  parent_host_name = "internet"
  parent_service_name = "ping4"

  child_host_name = "webserver"
  child_service_name = "ping4"

  states = [ OK, Warning ]

  disable_checks = true
}
```

Host-to-Host Example:

```
object Dependency "webserver-internet" {
  parent_host_name = "internet"

  child_host_name = "webserver"

  states = [ Up ]

  disable_checks = true
}
```

Configuration Attributes:

Name	Type	Description
parent_host_name	Object name	**Required.** The parent host.
parent_service_name	Object name	**Optional.** The parent service. If omitted, this dependency object is treated as host dependency.
child_host_name	Object name	**Required.** The child host.

Name	Type	Description
child_service_name	Object name	**Optional.** The child service. If omitted, this dependency object is treated as host dependency.
disable_checks	Boolean	**Optional.** Whether to disable checks when this dependency fails. Defaults to false.
disable_notifications	Boolean	**Optional.** Whether to disable notifications when this dependency fails. Defaults to true.
ignore_soft_states	Boolean	**Optional.** Whether to ignore soft states for the reachability calculation. Defaults to true.
period	Object name	**Optional.** Time period object during which this dependency is enabled.
states	Array	**Optional.** A list of state filters when this dependency should be OK. Defaults to [OK, Warning] for services and [Up] for hosts.

Available state filters:

```
OK
Warning
Critical
Unknown
Up
Down
```

When using apply rules for dependencies, you can leave out certain attributes which will be automatically determined by Icinga 2.

Service-to-Host Dependency Example:

```
apply Dependency "internet" to Service {
  parent_host_name = "dsl-router"
  disable_checks = true

  assign where host.name != "dsl-router"
}
```

This example sets all service objects matching the assign condition into a dependency relation to the parent host object `dsl-router` as implicit child services.

Service-to-Service-on-the-same-Host Dependency Example:

```
apply Dependency "disable-agent-checks" to Service {
  parent_service_name = "agent-health"

  assign where service.check_command == "ssh"
  ignore where service.name == "agent-health"
}
```

This example omits the `parent_host_name` attribute and Icinga 2 automatically sets its value to the name of the host object matched by the apply rule condition. All services where apply matches are made implicit child services in this dependency relation.

Dependency objects have composite names, i.e. their names are based on the `child_host_name` and `child_service_name` attributes and the name you specified. This means you can define more than one object with the same (short) name as long as one of the `child_host_name` and `child_service_name` attributes has a different value.

9.9 Downtime

Downtimes created at runtime are represented as objects. You can create downtimes with the schedule-downtime API action.

Example:

```
object Downtime "my-downtime" {
  host_name = "localhost"
  author = "icingaadmin"
  comment = "This is a downtime."
  start_time = 1505312869
  end_time = 1505312924
}
```

Configuration Attributes:

Name	Type	Description
host_name	Object name	**Required.** The name of the host this comment belongs to.

Name	Type	Description
service_name	Object name	**Optional.** The short name of the service this comment belongs to. If omitted, this comment object is treated as host comment.
author	String	**Required.** The author's name.
comment	String	**Required.** The comment text.
start_time	Timestamp	**Required.** The start time as UNIX timestamp.
end_time	Timestamp	**Required.** The end time as UNIX timestamp.
duration	Number	**Optional.** The duration as number.
entry_time	Timestamp	**Optional.** The UNIX timestamp when this downtime was added.
fixed	Boolean	**Optional.** Whether the downtime is fixed (true) or flexible (false). Defaults to flexible. Details in the advanced topics chapter.
triggers	Array of object names	**Optional.** List of downtimes which should be triggered by this downtime.

Runtime Attributes:

Name	Type	Description
trigger_time	Timestamp	The UNIX timestamp when this downtime was triggered.
triggered_by	Object name	The name of the downtime this downtime was triggered by.

9.10 ElasticsearchWriter

Writes check result metrics and performance data to an Elasticsearch instance. This configuration object is available as elasticsearch feature.

Example:

```
object ElasticsearchWriter "elasticsearch" {
  host = "127.0.0.1"
  port = 9200
  index = "icinga2"

  enable_send_perfdata = true

  flush_threshold = 1024
  flush_interval = 10
}
```

The index is rotated daily, as is recommended by Elastic, meaning the index will be renamed to `$index-$d.$M.$y`.

Configuration Attributes:

Name	Type	Description
host	String	**Required.** Elasticsearch host address. Defaults to `127.0.0.1`.
port	Number	**Required.** Elasticsearch port. Defaults to `9200`.
index	String	**Required.** Elasticsearch index name. Defaults to `icinga2`.
enable_send_perfdata	Boolean	**Optional.** Send parsed performance data metrics for check results. Defaults to `false`.
flush_interval	Duration	**Optional.** How long to buffer data points before transferring to Elasticsearch. Defaults to `10s`.
flush_threshold	Number	**Optional.** How many data points to buffer before forcing a transfer to Elasticsearch. Defaults to `1024`.
username	String	**Optional.** Basic auth username if Elasticsearch is hidden behind an HTTP proxy.
password	String	**Optional.** Basic auth password if Elasticsearch is hidden behind an HTTP proxy.

Name	Type	Description
enable_tls	Boolean	**Optional.** Whether to use a TLS stream. Defaults to `false`. Requires an HTTP proxy.
ca_path	String	**Optional.** Path to CA certificate to validate the remote host. Requires `enable_tls` set to `true`.
cert_path	String	**Optional.** Path to host certificate to present to the remote host for mutual verification. Requires `enable_tls` set to `true`.
key_path	String	**Optional.** Path to host key to accompany the cert_path. Requires `enable_tls` set to `true`.

Note: If `flush_threshold` is set too low, this will force the feature to flush all data to Elasticsearch too often. Experiment with the setting, if you are processing more than 1024 metrics per second or similar.

Basic auth is supported with the `username` and `password` attributes. This requires an HTTP proxy (Nginx, etc.) in front of the Elasticsearch instance. Check this blogpost for an example.

TLS for the HTTP proxy can be enabled with `enable_tls`. In addition to that you can specify the certificates with the `ca_path`, `cert_path` and `cert_key` attributes.

9.11 Endpoint

Endpoint objects are used to specify connection information for remote Icinga 2 instances. More details can be found in the distributed monitoring chapter.

Example:

```
object Endpoint "icinga2-client1.localdomain" {
  host = "192.168.56.111"
  port = 5665
  log_duration = 1d
}
```

Example (disable replay log):

```
object Endpoint "icinga2-client1.localdomain" {
  host = "192.168.5.111"
  port = 5665
  log_duration = 0
}
```

Configuration Attributes:

Name	Type	Description
host	String	**Optional.** The hostname/IP address of the remote Icinga 2 instance.
port	Number	**Optional.** The service name/port of the remote Icinga 2 instance. Defaults to `5665`.
log_duration	Duration	**Optional.** Duration for keeping replay logs on connection loss. Defaults to `1d` (86400 seconds). Attribute is specified in seconds. If log_duration is set to 0, replaying logs is disabled. You could also specify the value in human readable format like `10m` for 10 minutes or `1h` for one hour.

Endpoint objects cannot currently be created with the API.

9.12 EventCommand

An event command definition.

> **Note**
>
> Icinga 2 versions < 2.6.0 require the import of the plugin-event-command template.

Example:

```
object EventCommand "restart-httpd-event" {
  command = "/opt/bin/restart-httpd.sh"
}
```

Configuration Attributes:

Name	Type	Description
command	Array	**Required.** The command. This can either be an array of individual command arguments. Alternatively a string can be specified in which case the shell interpreter (usually /bin/sh) takes care of parsing the command. When using the "arguments" attribute this must be an array. Can be specified as function for advanced implementations.
env	Dictionary	**Optional.** A dictionary of macros which should be exported as environment variables prior to executing the command.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this command.
timeout	Duration	**Optional.** The command timeout in seconds. Defaults to `1m`.
arguments	Dictionary	**Optional.** A dictionary of command arguments.

Command arguments can be used the same way as for CheckCommand objects.

More advanced examples for event command usage can be found here.

9.13 ExternalCommandListener

Implements the Icinga 1.x command pipe which can be used to send commands to Icinga. This configuration object is available as command feature.

Example:

```
object ExternalCommandListener "command" {
    command_path = "/var/run/icinga2/cmd/icinga2.cmd"
}
```

Configuration Attributes:

Name	Type	Description
command_path	String	**Optional.** Path to the command pipe. Defaults to RunDir + "/icinga2/cmd/icinga2.cmd".

9.14 FileLogger

Specifies Icinga 2 logging to a file. This configuration object is available as `mainlog` and `debuglog` logging feature.

Example:

```
object FileLogger "debug-file" {
  severity = "debug"
  path = "/var/log/icinga2/debug.log"
}
```

Configuration Attributes:

Name	Type	Description
path	String	**Required.** The log path.
severity	String	**Optional.** The minimum severity for this log. Can be "debug", "notice", "information", "warning" or "critical". Defaults to "information".

9.15 GelfWriter

Writes event log entries to a defined GELF receiver host (Graylog, Logstash). This configuration object is available as gelf feature.

Example:

```
object GelfWriter "gelf" {
  host = "127.0.0.1"
  port = 12201
}
```

Configuration Attributes:

Name	Type	Description
host	String	**Optional.** GELF receiver host address. Defaults to `127.0.0.1`.
port	Number	**Optional.** GELF receiver port. Defaults to `12201`.
source	String	**Optional.** Source name for this instance. Defaults to `icinga2`.
enable_send_perfdata	Boolean	**Optional.** Enable performance data for 'CHECK RESULT' events.

9.16 GraphiteWriter

Writes check result metrics and performance data to a defined Graphite Carbon host. This configuration object is available as graphite feature.

Example:

```
object GraphiteWriter "graphite" {
  host = "127.0.0.1"
  port = 2003
}
```

Configuration Attributes:

Name	Type	Description
host	String	**Optional.** Graphite Carbon host address. Defaults to `127.0.0.1`.
port	Number	**Optional.** Graphite Carbon port. Defaults to `2003`.
host_name_template	String	**Optional.** Metric prefix for host name. Defaults to `icinga2.$host.name$.host.$host.check_command$`.
service_name_template	String	**Optional.** Metric prefix for service name. Defaults to `icinga2.$host.name$.services.$service.name$.$servic`
enable_send_thresholds	Boolean	**Optional.** Send additional threshold metrics. Defaults to `false`.

Name	Type	Description
enable_send_metadata	Boolean	**Optional.** Send additional metadata metrics. Defaults to `false`.

Additional usage examples can be found here.

9.17 Host

A host.

Example:

```
object Host "icinga2-client1.localdomain" {
  display_name = "Linux Client 1"
  address = "192.168.56.111"
  address6 = "2a00:1450:4001:815::2003"

  groups = [ "linux-servers" ]

  check_command = "hostalive"
}
```

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the host (e.g. displayed by external interfaces instead of the name if set).
address	String	**Optional.** The host's IPv4 address. Available as command runtime macro `$address$` if set.
address6	String	**Optional.** The host's IPv6 address. Available as command runtime macro `$address6$` if set.
groups	Array of object names	**Optional.** A list of host groups this host belongs to.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this host.

Name	Type	Description
check_command	Object name	**Required.** The name of the check command.
max_check_attempts	Number	**Optional.** The number of times a host is re-checked before changing into a hard state. Defaults to 3.
check_period	Object name	**Optional.** The name of a time period which determines when this host should be checked. Not set by default.
check_timeout	Duration	**Optional.** Check command timeout in seconds. Overrides the CheckCommand's `timeout` attribute.
check_interval	Duration	**Optional.** The check interval (in seconds). This interval is used for checks when the host is in a `HARD` state. Defaults to `5m`.
retry_interval	Duration	**Optional.** The retry interval (in seconds). This interval is used for checks when the host is in a `SOFT` state. Defaults to `1m`.
enable_notifications	Boolean	**Optional.** Whether notifications are enabled. Defaults to true.
enable_active_checks	Boolean	**Optional.** Whether active checks are enabled. Defaults to true.
enable_passive_checks	Boolean	**Optional.** Whether passive checks are enabled. Defaults to true.
enable_event_handler	Boolean	**Optional.** Enables event handlers for this host. Defaults to true.
enable_flapping	Boolean	**Optional.** Whether flap detection is enabled. Defaults to false.
enable_perfdata	Boolean	**Optional.** Whether performance data processing is enabled. Defaults to true.

Name	Type	Description
event_command	Object name	**Optional.** The name of an event command that should be executed every time the host's state changes or the host is in a `SOFT` state.
flapping_threshold_high	Number	**Optional.** Flapping upper bound in percent for a host to be considered flapping. Default `30.0`
flapping_threshold_low	Number	**Optional.** Flapping lower bound in percent for a host to be considered not flapping. Default `25.0`
volatile	Boolean	**Optional.** The volatile setting enables always `HARD` state types if `NOT-OK` state changes occur. Defaults to false.
zone	Object name	**Optional.** The zone this object is a member of. Please read the distributed monitoring chapter for details.
command_endpoint	Object name	**Optional.** The endpoint where commands are executed on.
notes	String	**Optional.** Notes for the host.
notes_url	String	**Optional.** URL for notes for the host (for example, in notification commands).
action_url	String	**Optional.** URL for actions for the host (for example, an external graphing tool).
icon_image	String	**Optional.** Icon image for the host. Used by external interfaces only.
icon_image_alt	String	**Optional.** Icon image description for the host. Used by external interface only.

The actual check interval might deviate slightly from the configured values due

to the fact that Icinga tries to evenly distribute all checks over a certain period of time, i.e. to avoid load spikes.

Best Practice

The `address` and `address6` attributes are required for running commands using the `$address$` and `$address6$` runtime macros.

Runtime Attributes:

Name	Type	Description
next_check	Timestamp	When the next check occurs (as a UNIX timestamp).
last_check	Timestamp	When the last check occurred (as a UNIX timestamp).
check_attempt	Number	The current check attempt number.
state_type	Number	The current state type (0 = SOFT, 1 = HARD).
last_state_type	Number	The previous state type (0 = SOFT, 1 = HARD).
last_reachable	Boolean	Whether the host was reachable when the last check occurred.
last_check_result	CheckResult	The current check result.
last_state_change	Timestamp	When the last state change occurred (as a UNIX timestamp).
last_hard_state_change	Timestamp	When the last hard state change occurred (as a UNIX timestamp).
last_in_downtime	Boolean	Whether the host was in a downtime when the last check occurred.
acknowledgement	Number	The acknowledgement type (0 = NONE, 1 = NORMAL, 2 = STICKY).
acknowledgement_expiry	Timestamp	When the acknowledgement expires (as a UNIX timestamp; 0 = no expiry).
downtime_depth	Number	Whether the host has one or more active downtimes.
flapping_last_change	Timestamp	When the last flapping change occurred (as a UNIX timestamp).

Name	Type	Description
flapping	Boolean	Whether the host is flapping between states.
flapping_current	Number	Current flapping value in percent (see flapping_thresholds)
state	Number	The current state (0 = UP, 1 = DOWN).
last_state	Number	The previous state (0 = UP, 1 = DOWN).
last_hard_state	Number	The last hard state (0 = UP, 1 = DOWN).
last_state_up	Timestamp	When the last UP state occurred (as a UNIX timestamp).
last_state_down	Timestamp	When the last DOWN state occurred (as a UNIX timestamp).

9.18 HostGroup

A group of hosts.

Best Practice

Assign host group members using the group assign rules.

Example:

```
object HostGroup "linux-servers" {
  display_name = "Linux Servers"

  assign where host.vars.os == "Linux"
}
```

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the host group.
groups	Array of object names	**Optional.** An array of nested group names.

9.19 IcingaApplication

The IcingaApplication object is required to start Icinga 2. The object name must be `app`. If the object configuration is missing, Icinga 2 will automatically create an IcingaApplication object.

Example:

```
object IcingaApplication "app" {
  enable_perfdata = false
}
```

Configuration Attributes:

Name	Type	Description
enable_notifications	Boolean	**Optional.** Whether notifications are globally enabled. Defaults to true.
enable_event_handlers	Boolean	**Optional.** Whether event handlers are globally enabled. Defaults to true.
enable_flapping	Boolean	**Optional.** Whether flap detection is globally enabled. Defaults to true.
enable_host_checks	Boolean	**Optional.** Whether active host checks are globally enabled. Defaults to true.
enable_service_checks	Boolean	**Optional.** Whether active service checks are globally enabled. Defaults to true.
enable_perfdata	Boolean	**Optional.** Whether performance data processing is globally enabled. Defaults to true.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are available globally.

9.20 IdoMySqlConnection

IDO database adapter for MySQL. This configuration object is available as ido-mysql feature.

Example:

```
object IdoMysqlConnection "mysql-ido" {
```

```
  host = "127.0.0.1"
  port = 3306
  user = "icinga"
  password = "icinga"
  database = "icinga"

  cleanup = {
    downtimehistory_age = 48h
    contactnotifications_age = 31d
  }
}
```

Configuration Attributes:

Name	Type	Description
host	String	**Optional.** MySQL database host address. Defaults to `localhost`.
port	Number	**Optional.** MySQL database port. Defaults to `3306`.
socket_path	String	**Optional.** MySQL socket path.
user	String	**Optional.** MySQL database user with read/write permission to the icinga database. Defaults to `icinga`.
password	String	**Optional.** MySQL database user's password. Defaults to `icinga`.
database	String	**Optional.** MySQL database name. Defaults to `icinga`.
enable_ssl	Boolean	**Optional.** Use SSL. Defaults to false. Change to `true` in case you want to use any of the SSL options.
ssl_key	String	**Optional.** MySQL SSL client key file path.
ssl_cert	String	**Optional.** MySQL SSL certificate file path.
ssl_ca	String	**Optional.** MySQL SSL certificate authority certificate file path.

Name	Type	Description
ssl_capath	String	**Optional.** MySQL SSL trusted SSL CA certificates in PEM format directory path.
ssl_cipher	String	**Optional.** MySQL SSL list of allowed ciphers.
table_prefix	String	**Optional.** MySQL database table prefix. Defaults to `icinga_`.
instance_name	String	**Optional.** Unique identifier for the local Icinga 2 instance. Defaults to `default`.
instance_description	String	**Optional.** Description for the Icinga 2 instance.
enable_ha	Boolean	**Optional.** Enable the high availability functionality. Only valid in a cluster setup. Defaults to "true".
failover_timeout	Duration	**Optional.** Set the failover timeout in a HA cluster. Must not be lower than 60s. Defaults to `60s`.
cleanup	Dictionary	**Optional.** Dictionary with items for historical table cleanup.
categories	Array	**Optional.** Array of information types that should be written to the database.

Cleanup Items:

Name	Type	Description
acknowledgements_age	Duration	**Optional.** Max age for acknowledgements table rows (entry_time). Defaults to 0 (never).
commenthistory_age	Duration	**Optional.** Max age for commenthistory table rows (entry_time). Defaults to 0 (never).

Name	Type	Description
contactnotifications_age	Duration	**Optional.** Max age for contactnotifications table rows (start_time). Defaults to 0 (never).
contactnotificationmethods_age	Duration	**Optional.** Max age for contactnotificationmethods table rows (start_time). Defaults to 0 (never).
downtimehistory_age	Duration	**Optional.** Max age for downtimehistory table rows (entry_time). Defaults to 0 (never).
eventhandlers_age	Duration	**Optional.** Max age for eventhandlers table rows (start_time). Defaults to 0 (never).
externalcommands_age	Duration	**Optional.** Max age for externalcommands table rows (entry_time). Defaults to 0 (never).
flappinghistory_age	Duration	**Optional.** Max age for flappinghistory table rows (event_time). Defaults to 0 (never).
hostchecks_age	Duration	**Optional.** Max age for hostalives table rows (start_time). Defaults to 0 (never).
logentries_age	Duration	**Optional.** Max age for logentries table rows (logentry_time). Defaults to 0 (never).
notifications_age	Duration	**Optional.** Max age for notifications table rows (start_time). Defaults to 0 (never).
processevents_age	Duration	**Optional.** Max age for processevents table rows (event_time). Defaults to 0 (never).
statehistory_age	Duration	**Optional.** Max age for statehistory table rows (state_time). Defaults to 0 (never).

Name	Type	Description
servicechecks_age	Duration	**Optional.** Max age for servicechecks table rows (start_time). Defaults to 0 (never).
systemcommands_age	Duration	**Optional.** Max age for systemcommands table rows (start_time). Defaults to 0 (never).

Data Categories:

Name	Description	Required by
DbCatConfig	Configuration data	Icinga Web 2
DbCatState	Current state data	Icinga Web 2
DbCatAcknowledgement	Acknowledgements	Icinga Web 2
DbCatComment	Comments	Icinga Web 2
DbCatDowntime	Downtimes	Icinga Web 2
DbCatEventHandler	Event handler data	Icinga Web 2
DbCatExternalCommand	External commands	–
DbCatFlapping	Flap detection data	Icinga Web 2
DbCatCheck	Check results	–
DbCatLog	Log messages	–
DbCatNotification	Notifications	Icinga Web 2
DbCatProgramStatus	Program status data	Icinga Web 2
DbCatRetention	Retention data	Icinga Web 2
DbCatStateHistory	Historical state data	Icinga Web 2

The default value for `categories` includes everything required by Icinga Web 2 in the table above.

In addition to the category flags listed above the `DbCatEverything` flag may be used as a shortcut for listing all flags.

9.21 IdoPgsqlConnection

IDO database adapter for PostgreSQL. This configuration object is available as ido-pgsql feature.

Example:

```
object IdoPgsqlConnection "pgsql-ido" {
  host = "127.0.0.1"
```

```
  port = 5432
  user = "icinga"
  password = "icinga"
  database = "icinga"

  cleanup = {
    downtimehistory_age = 48h
    contactnotifications_age = 31d
  }
}
```

Configuration Attributes:

Name	Type	Description
host	String	**Optional.** PostgreSQL database host address. Defaults to `localhost`.
port	Number	**Optional.** PostgreSQL database port. Defaults to `5432`.
user	String	**Optional.** PostgreSQL database user with read/write permission to the icinga database. Defaults to `icinga`.
password	String	**Optional.** PostgreSQL database user's password. Defaults to `icinga`.
database	String	**Optional.** PostgreSQL database name. Defaults to `icinga`.
table_prefix	String	**Optional.** PostgreSQL database table prefix. Defaults to `icinga_`.
instance_name	String	**Optional.** Unique identifier for the local Icinga 2 instance. Defaults to `default`.
instance_description	String	**Optional.** Description for the Icinga 2 instance.
enable_ha	Boolean	**Optional.** Enable the high availability functionality. Only valid in a cluster setup. Defaults to "true".

Name	Type	Description
failover_timeout	Duration	**Optional.** Set the failover timeout in a HA cluster. Must not be lower than 60s. Defaults to `60s`.
cleanup	Dictionary	**Optional.** Dictionary with items for historical table cleanup.
categories	Array	**Optional.** Array of information types that should be written to the database.

Cleanup Items:

Name	Type	Description
acknowledgements_age	Duration	**Optional.** Max age for acknowledgements table rows (entry_time). Defaults to 0 (never).
commenthistory_age	Duration	**Optional.** Max age for commenthistory table rows (entry_time). Defaults to 0 (never).
contactnotifications_age	Duration	**Optional.** Max age for contactnotifications table rows (start_time). Defaults to 0 (never).
contactnotificationmethods_age	Duration	**Optional.** Max age for contactnotificationmethods table rows (start_time). Defaults to 0 (never).
downtimehistory_age	Duration	**Optional.** Max age for downtimehistory table rows (entry_time). Defaults to 0 (never).
eventhandlers_age	Duration	**Optional.** Max age for eventhandlers table rows (start_time). Defaults to 0 (never).
externalcommands_age	Duration	**Optional.** Max age for externalcommands table rows (entry_time). Defaults to 0 (never).

Name	Type	Description
flappinghistory_age	Duration	**Optional.** Max age for flappinghistory table rows (event_time). Defaults to 0 (never).
hostchecks_age	Duration	**Optional.** Max age for hostalives table rows (start_time). Defaults to 0 (never).
logentries_age	Duration	**Optional.** Max age for logentries table rows (logentry_time). Defaults to 0 (never).
notifications_age	Duration	**Optional.** Max age for notifications table rows (start_time). Defaults to 0 (never).
processevents_age	Duration	**Optional.** Max age for processevents table rows (event_time). Defaults to 0 (never).
statehistory_age	Duration	**Optional.** Max age for statehistory table rows (state_time). Defaults to 0 (never).
servicechecks_age	Duration	**Optional.** Max age for servicechecks table rows (start_time). Defaults to 0 (never).
systemcommands_age	Duration	**Optional.** Max age for systemcommands table rows (start_time). Defaults to 0 (never).

Data Categories:

Name	Description	Required by
DbCatConfig	Configuration data	Icinga Web 2
DbCatState	Current state data	Icinga Web 2
DbCatAcknowledgement	Acknowledgements	Icinga Web 2
DbCatComment	Comments	Icinga Web 2
DbCatDowntime	Downtimes	Icinga Web 2
DbCatEventHandler	Event handler data	Icinga Web 2
DbCatExternalCommand	External commands	–

Name	Description	Required by
DbCatFlapping	Flap detection data	Icinga Web 2
DbCatCheck	Check results	–
DbCatLog	Log messages	–
DbCatNotification	Notifications	Icinga Web 2
DbCatProgramStatus	Program status data	Icinga Web 2
DbCatRetention	Retention data	Icinga Web 2
DbCatStateHistory	Historical state data	Icinga Web 2

The default value for `categories` includes everything required by Icinga Web 2 in the table above.

In addition to the category flags listed above the `DbCatEverything` flag may be used as a shortcut for listing all flags.

9.22 InfluxdbWriter

Writes check result metrics and performance data to a defined InfluxDB host. This configuration object is available as influxdb feature.

Example:

```
object InfluxdbWriter "influxdb" {
  host = "127.0.0.1"
  port = 8086
  database = "icinga2"

  flush_threshold = 1024
  flush_interval = 10s

  host_template = {
    measurement = "$host.check_command$"
    tags = {
      hostname = "$host.name$"
    }
  }
  service_template = {
    measurement = "$service.check_command$"
    tags = {
      hostname = "$host.name$"
      service = "$service.name$"
    }
  }
}
```

Configuration Attributes:

Name	Type	Description
host	String	**Required.** InfluxDB host address. Defaults to `127.0.0.1`.
port	Number	**Required.** InfluxDB HTTP port. Defaults to `8086`.
database	String	**Required.** InfluxDB database name. Defaults to `icinga2`.
username	String	**Optional.** InfluxDB user name. Defaults to `none`.
password	String	**Optional.** InfluxDB user password. Defaults to `none`.
ssl_enable	Boolean	**Optional.** Whether to use a TLS stream. Defaults to `false`.
ssl_ca_cert	String	**Optional.** Path to CA certificate to validate the remote host.
ssl_cert	String	**Optional.** Path to host certificate to present to the remote host for mutual verification.
ssl_key	String	**Optional.** Path to host key to accompany the ssl_cert.
host_template	String	**Required.** Host template to define the InfluxDB line protocol.
service_template	String	**Required.** Service template to define the influxDB line protocol.
enable_send_thresholds	Boolean	**Optional.** Whether to send warn, crit, min & max tagged data.
enable_send_metadata	Boolean	**Optional.** Whether to send check metadata e.g. states, execution time, latency etc.
flush_interval	Duration	**Optional.** How long to buffer data points before transferring to InfluxDB. Defaults to `10s`.

Name	Type	Description
flush_threshold	Number	**Optional.** How many data points to buffer before forcing a transfer to InfluxDB. Defaults to `1024`.

Note: If `flush_threshold` is set too low, this will always force the feature to flush all data to InfluxDB. Experiment with the setting, if you are processing more than 1024 metrics per second or similar.

9.23 LiveStatusListener

Livestatus API interface available as TCP or UNIX socket. Historical table queries require the CompatLogger feature enabled pointing to the log files using the `compat_log_path` configuration attribute. This configuration object is available as livestatus feature.

Examples:

```
object LivestatusListener "livestatus-tcp" {
  socket_type = "tcp"
  bind_host = "127.0.0.1"
  bind_port = "6558"
}

object LivestatusListener "livestatus-unix" {
  socket_type = "unix"
  socket_path = "/var/run/icinga2/cmd/livestatus"
}
```

Configuration Attributes:

Name	Type	Description
socket_type	String	**Optional.** Specifies the socket type. Can be either `tcp` or `unix`. Defaults to `unix`.
bind_host	String	**Optional.** Only valid when `socket_type` is set to `tcp`. Host address to listen on for connections. Defaults to `127.0.0.1`.

Name	Type	Description
bind_port	Number	**Optional.** Only valid when `socket_type` is set to `tcp`. Port to listen on for connections. Defaults to `6558`.
socket_path	String	**Optional.** Only valid when `socket_type` is set to `unix`. Specifies the path to the UNIX socket file. Defaults to RunDir + "/icinga2/cmd/livestatus".
compat_log_path	String	**Optional.** Path to Icinga 1.x log files. Required for historical table queries. Requires `CompatLogger` feature enabled. Defaults to LocalStateDir + "/log/icinga2/compat"

Note

UNIX sockets are not supported on Windows.

9.24 Notification

Notification objects are used to specify how users should be notified in case of host and service state changes and other events.

Best Practice

Rather than creating a `Notification` object for a specific host or service it is usually easier to just create a `Notification` template and use the `apply` keyword to assign the notification to a number of hosts or services. Use the `to` keyword to set the specific target type for `Host` or `Service`. Check the notifications chapter for detailed examples.

Example:

```
object Notification "localhost-ping-notification" {
  host_name = "localhost"
  service_name = "ping4"

  command = "mail-notification"
```

```
  users = [ "user1", "user2" ]

  types = [ Problem, Recovery ]
}
```

Configuration Attributes:

Name	Type	Description
host_name	Object name	**Required.** The name of the host this notification belongs to.
service_name	Object name	**Optional.** The short name of the service this notification belongs to. If omitted, this notification object is treated as host notification.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this notification object.
users	Array of object names	**Optional.** A list of user names who should be notified.
user_groups	Array of object names	**Optional.** A list of user group names who should be notified.
times	Dictionary	**Optional.** A dictionary containing `begin` and `end` attributes for the notification.
command	Object name	**Required.** The name of the notification command which should be executed when the notification is triggered.
interval	Duration	**Optional.** The notification interval (in seconds). This interval is used for active notifications. Defaults to 30 minutes. If set to 0, re-notifications are disabled.
period	Object name	**Optional.** The name of a time period which determines when this notification should be triggered. Not set by default.

Name	Type	Description
zone	Object name	**Optional.** The zone this object is a member of. Please read the distributed monitoring chapter for details.
types	Array	**Optional.** A list of type filters when this notification should be triggered. By default everything is matched.
states	Array	**Optional.** A list of state filters when this notification should be triggered. By default everything is matched.

Available notification state filters for Service:

```
OK
Warning
Critical
Unknown
```

Available notification state filters for Host:

```
Up
Down
```

Available notification type filters:

```
DowntimeStart
DowntimeEnd
DowntimeRemoved
Custom
Acknowledgement
Problem
Recovery
FlappingStart
FlappingEnd
```

Runtime Attributes:

Name	Type	Description
last_notification	Timestamp	When the last notification was sent for this Notification object (as a UNIX timestamp).
next_notification	Timestamp	When the next notification is going to be sent for this assuming the associated host/service is still in a non-OK state (as a UNIX timestamp).
notification_number	Number	The notification number.
last_problem_notification	Timestamp	When the last notification was sent for a problem (as a UNIX timestamp).

9.25 NotificationCommand

A notification command definition.

> **Note**
>
> Icinga 2 versions < 2.6.0 require the import of the plugin-notification-command template.

Example:

```
object NotificationCommand "mail-service-notification" {
 command = [ SysconfDir + "/icinga2/scripts/mail-service-notification.sh" ]

  arguments += {
    "-4" = {
      required = true
      value = "$notification_address$"
    }
    "-6" = "$notification_address6$"
    "-b" = "$notification_author$"
    "-c" = "$notification_comment$"
```

```
  "-d" = {
    required = true
    value = "$notification_date$"
  }
  "-e" = {
    required = true
    value = "$notification_servicename$"
  }
  "-f" = {
    value = "$notification_from$"
  description = "Set from address. Requires GNU mailutils (Debian/Ubuntu) or mailx (RHEL/SUSE
  }
  "-i" = "$notification_icingaweb2url$"
  "-l" = {
    required = true
    value = "$notification_hostname$"
  }
  "-n" = {
    required = true
    value = "$notification_hostdisplayname$"
  }
  "-o" = {
    required = true
    value = "$notification_serviceoutput$"
  }
  "-r" = {
    required = true
    value = "$notification_useremail$"
  }
  "-s" = {
    required = true
    value = "$notification_servicestate$"
  }
  "-t" = {
    required = true
    value = "$notification_type$"
  }
  "-u" = {
    required = true
    value = "$notification_servicedisplayname$"
  }
  "-v" = "$notification_logtosyslog$"
}

vars += {
  notification_address = "$address$"
```

```
    notification_address6 = "$address6$"
    notification_author = "$notification.author$"
    notification_comment = "$notification.comment$"
    notification_type = "$notification.type$"
    notification_date = "$icinga.long_date_time$"
    notification_hostname = "$host.name$"
    notification_hostdisplayname = "$host.display_name$"
    notification_servicename = "$service.name$"
    notification_serviceoutput = "$service.output$"
    notification_servicestate = "$service.state$"
    notification_useremail = "$user.email$"
    notification_servicedisplayname = "$service.display_name$"
  }
}
```

Configuration Attributes:

Name	Type	Description
command	Array	**Required.** The command. This can either be an array of individual command arguments. Alternatively a string can be specified in which case the shell interpreter (usually /bin/sh) takes care of parsing the command. When using the "arguments" attribute this must be an array. Can be specified as function for advanced implementations.
env	Dictionary	**Optional.** A dictionary of macros which should be exported as environment variables prior to executing the command.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this command.
timeout	Duration	**Optional.** The command timeout in seconds. Defaults to `1m`.
arguments	Dictionary	**Optional.** A dictionary of command arguments.

Command arguments can be used the same way as for CheckCommand objects.

More details on specific attributes can be found in this chapter.

9.26 NotificationComponent

The notification component is responsible for sending notifications. This configuration object is available as notification feature.

Example:

```
object NotificationComponent "notification" { }
```

Configuration Attributes:

Name	Type	Description
enable_ha	Boolean	**Optional.** Enable the high availability functionality. Only valid in a cluster setup. Disabling this currently only affects reminder notifications. Defaults to "true".

9.27 OpenTsdbWriter

Writes check result metrics and performance data to OpenTSDB. This configuration object is available as opentsdb feature.

Example:

```
object OpenTsdbWriter "opentsdb" {
  host = "127.0.0.1"
  port = 4242
```

Configuration Attributes:

Name	Type	Description
host	String	**Optional.** OpenTSDB host address. Defaults to `127.0.0.1`.
port	Number	**Optional.** OpenTSDB port. Defaults to `4242`.

9.28 PerfdataWriter

Writes check result performance data to a defined path using macro pattern consisting of custom attributes and runtime macros. This configuration object

is available as perfdata feature.

Example:

```
object PerfdataWriter "perfdata" {
  host_perfdata_path = "/var/spool/icinga2/perfdata/host-perfdata"

  service_perfdata_path = "/var/spool/icinga2/perfdata/service-perfdata"

  host_format_template = "DATATYPE::HOSTPERFDATA\tTIMET::$icinga.timet$\tHOSTNAME::$host.name
  service_format_template = "DATATYPE::SERVICEPERFDATA\tTIMET::$icinga.timet$\tHOSTNAME::$hos

  rotation_interval = 15s
}
```

Configuration Attributes:

Name	Type	Description
host_perfdata_path	String	**Optional.** Path to the host performance data file. Defaults to LocalStateDir + "/spool/icinga2/perfdata/host-perfdata".
service_perfdata_path	String	**Optional.** Path to the service performance data file. Defaults to LocalStateDir + "/spool/icinga2/perfdata/service-perfdata".
host_temp_path	String	**Optional.** Path to the temporary host file. Defaults to LocalStateDir + "/spool/icinga2/tmp/host-perfdata".
service_temp_path	String	**Optional.** Path to the temporary service file. Defaults to LocalStateDir + "/spool/icinga2/tmp/service-perfdata".
host_format_template	String	**Optional.** Host Format template for the performance data file. Defaults to a template that's suitable for use with PNP4Nagios.

Name	Type	Description
service_format_template	String	**Optional.** Service Format template for the performance data file. Defaults to a template that's suitable for use with PNP4Nagios.
rotation_interval	Duration	**Optional.** Rotation interval for the files specified in `{host,service}_perfdata_path`. Defaults to `30s`.

When rotating the performance data file the current UNIX timestamp is appended to the path specified in `host_perfdata_path` and `service_perfdata_path` to generate a unique filename.

9.29 ScheduledDowntime

ScheduledDowntime objects can be used to set up recurring downtimes for hosts/services.

> **Best Practice**
>
> Rather than creating a `ScheduledDowntime` object for a specific host or service it is usually easier to just create a `ScheduledDowntime` template and use the `apply` keyword to assign the scheduled downtime to a number of hosts or services. Use the `to` keyword to set the specific target type for `Host` or `Service`. Check the recurring downtimes example for details.

Example:

```
object ScheduledDowntime "some-downtime" {
  host_name = "localhost"
  service_name = "ping4"

  author = "icingaadmin"
  comment = "Some comment"

  fixed = false
  duration = 30m

  ranges = {
    "sunday" = "02:00-03:00"
  }
}
```

Configuration Attributes:

Name	Type	Description
host_name	Object name	**Required.** The name of the host this scheduled downtime belongs to.
service_name	Object name	**Optional.** The short name of the service this scheduled downtime belongs to. If omitted, this downtime object is treated as host downtime.
author	String	**Required.** The author of the downtime.
comment	String	**Required.** A comment for the downtime.
fixed	Boolean	**Optional.** Whether this is a fixed downtime. Defaults to `true`.
duration	Duration	**Optional.** How long the downtime lasts. Only has an effect for flexible (non-fixed) downtimes.
ranges	Dictionary	**Required.** A dictionary containing information which days and durations apply to this timeperiod.

ScheduledDowntime objects have composite names, i.e. their names are based on the `host_name` and `service_name` attributes and the name you specified. This means you can define more than one object with the same (short) name as long as one of the `host_name` and `service_name` attributes has a different value.

9.30 Service

Service objects describe network services and how they should be checked by Icinga 2.

> **Best Practice**
>
> Rather than creating a `Service` object for a specific host it is usually easier to just create a `Service` template and use the `apply` keyword

to assign the service to a number of hosts. Check the apply chapter for details.

Example:

```
object Service "uptime" {
  host_name = "localhost"

  display_name = "localhost Uptime"

  check_command = "snmp"

  vars.snmp_community = "public"
  vars.snmp_oid = "DISMAN-EVENT-MIB::sysUpTimeInstance"

  check_interval = 60s
  retry_interval = 15s

  groups = [ "all-services", "snmp" ]
}
```

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the service.
host_name	Object name	**Required.** The host this service belongs to. There must be a `Host` object with that name.
groups	Array of object names	**Optional.** The service groups this service belongs to.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this service.
check_command	Object name	**Required.** The name of the check command.
max_check_attempts	Number	**Optional.** The number of times a service is re-checked before changing into a hard state. Defaults to 3.

Name	Type	Description
check_period	Object name	**Optional.** The name of a time period which determines when this service should be checked. Not set by default.
check_timeout	Duration	**Optional.** Check command timeout in seconds. Overrides the CheckCommand's `timeout` attribute.
check_interval	Duration	**Optional.** The check interval (in seconds). This interval is used for checks when the service is in a `HARD` state. Defaults to `5m`.
retry_interval	Duration	**Optional.** The retry interval (in seconds). This interval is used for checks when the service is in a `SOFT` state. Defaults to `1m`.
enable_notifications	Boolean	**Optional.** Whether notifications are enabled. Defaults to `true`.
enable_active_checks	Boolean	**Optional.** Whether active checks are enabled. Defaults to `true`.
enable_passive_checks	Boolean	**Optional.** Whether passive checks are enabled. Defaults to `true`.
enable_event_handler	Boolean	**Optional.** Enables event handlers for this host. Defaults to `true`.
enable_flapping	Boolean	**Optional.** Whether flap detection is enabled. Defaults to `false`.
flapping_threshold_high	Number	**Optional.** Flapping upper bound in percent for a service to be considered flapping. `30.0`
flapping_threshold_low	Number	**Optional.** Flapping lower bound in percent for a service to be considered not flapping. `25.0`

Name	Type	Description
enable_perfdata	Boolean	**Optional.** Whether performance data processing is enabled. Defaults to `true`.
event_command	Object name	**Optional.** The name of an event command that should be executed every time the service's state changes or the service is in a `SOFT` state.
volatile	Boolean	**Optional.** The volatile setting enables always `HARD` state types if `NOT-OK` state changes occur. Defaults to `false`.
zone	Object name	**Optional.** The zone this object is a member of. Please read the distributed monitoring chapter for details.
name	String	**Required.** The service name. Must be unique on a per-host basis. For advanced usage in apply rules only.
command_endpoint	Object name	**Optional.** The endpoint where commands are executed on.
notes	String	**Optional.** Notes for the service.
notes_url	String	**Optional.** URL for notes for the service (for example, in notification commands).
action_url	String	**Optional.** URL for actions for the service (for example, an external graphing tool).
icon_image	String	**Optional.** Icon image for the service. Used by external interfaces only.
icon_image_alt	String	**Optional.** Icon image description for the service. Used by external interface only.

Service objects have composite names, i.e. their names are based on the host_name attribute and the name you specified. This means you can define

more than one object with the same (short) name as long as the `host_name` attribute has a different value.

The actual check interval might deviate slightly from the configured values due to the fact that Icinga tries to evenly distribute all checks over a certain period of time, i.e. to avoid load spikes.

Runtime Attributes:

Name	Type	Description
next_check	Timestamp	When the next check occurs (as a UNIX timestamp).
last_check	Timestamp	When the last check occurred (as a UNIX timestamp).
check_attempt	Number	The current check attempt number.
state_type	Number	The current state type (0 = SOFT, 1 = HARD).
last_state_type	Number	The previous state type (0 = SOFT, 1 = HARD).
last_reachable	Boolean	Whether the service was reachable when the last check occurred.
last_check_result	CheckResult	The current check result.
last_state_change	Timestamp	When the last state change occurred (as a UNIX timestamp).
last_hard_state_change	Timestamp	When the last hard state change occurred (as a UNIX timestamp).
last_in_downtime	Boolean	Whether the service was in a downtime when the last check occurred.
acknowledgement	Number	The acknowledgement type (0 = NONE, 1 = NORMAL, 2 = STICKY).
acknowledgement_expiry	Timestamp	When the acknowledgement expires (as a UNIX timestamp; 0 = no expiry).
downtime_depth	Number	Whether the service has one or more active downtimes.
flapping_last_change	Timestamp	When the last flapping change occurred (as a UNIX timestamp).

Name	Type	Description
flapping_current	Number	Current flapping value in percent (see flapping_thresholds)
flapping	Boolean	Whether the host is flapping between states.
state	Number	The current state (0 = OK, 1 = WARNING, 2 = CRITICAL, 3 = UNKNOWN).
last_state	Number	The previous state (0 = OK, 1 = WARNING, 2 = CRITICAL, 3 = UNKNOWN).
last_hard_state	Number	The last hard state (0 = OK, 1 = WARNING, 2 = CRITICAL, 3 = UNKNOWN).
last_state_ok	Timestamp	When the last OK state occurred (as a UNIX timestamp).
last_state_warning	Timestamp	When the last WARNING state occurred (as a UNIX timestamp).
last_state_critical	Timestamp	When the last CRITICAL state occurred (as a UNIX timestamp).
last_state_unknown	Timestamp	When the last UNKNOWN state occurred (as a UNIX timestamp).

9.31 ServiceGroup

A group of services.

Best Practice

Assign service group members using the group assign rules.

Example:

```
object ServiceGroup "snmp" {
  display_name = "SNMP services"
}
```

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the service group.
groups	Array of object names	**Optional.** An array of nested group names.

9.32 StatusDataWriter

Periodically writes status and configuration data files which are used by third-party tools. This configuration object is available as statusdata feature.

Example:

```
object StatusDataWriter "status" {
    status_path = "/var/cache/icinga2/status.dat"
    objects_path = "/var/cache/icinga2/objects.cache"
    update_interval = 30s
}
```

Configuration Attributes:

Name	Type	Description
status_path	String	**Optional.** Path to the `status.dat` file. Defaults to LocalStateDir + "/cache/icinga2/status.dat".
objects_path	String	**Optional.** Path to the `objects.cache` file. Defaults to LocalStateDir + "/cache/icinga2/objects.cache".
update_interval	Duration	**Optional.** The interval in which the status files are updated. Defaults to `15s`.

9.33 SyslogLogger

Specifies Icinga 2 logging to syslog. This configuration object is available as `syslog` logging feature.

Example:

```
object SyslogLogger "syslog" {
  severity = "warning"
}
```

Configuration Attributes:

Name	Type	Description
severity	String	**Optional.** The minimum severity for this log. Can be “debug”, “notice”, “information”, “warning” or “critical”. Defaults to “warning”.
facility	String	**Optional.** Defines the facility to use for syslog entries. This can be a facility constant like `FacilityDaemon`. Defaults to `FacilityUser`.

Facility Constants:

Name	Facility	Description
FacilityAuth	LOG_AUTH	The authorization system.
FacilityAuthPriv	LOG_AUTHPRIV	The same as `FacilityAuth`, but logged to a file readable only by selected individuals.
FacilityCron	LOG_CRON	The cron daemon.
FacilityDaemon	LOG_DAEMON	System daemons that are not provided for explicitly by other facilities.
FacilityFtp	LOG_FTP	The file transfer protocol daemons.
FacilityKern	LOG_KERN	Messages generated by the kernel. These cannot be generated by any user processes.
FacilityLocal0	LOG_LOCAL0	Reserved for local use.

Name	Facility	Description
FacilityLocal1	LOG_LOCAL1	Reserved for local use.
FacilityLocal2	LOG_LOCAL2	Reserved for local use.
FacilityLocal3	LOG_LOCAL3	Reserved for local use.
FacilityLocal4	LOG_LOCAL4	Reserved for local use.
FacilityLocal5	LOG_LOCAL5	Reserved for local use.
FacilityLocal6	LOG_LOCAL6	Reserved for local use.
FacilityLocal7	LOG_LOCAL7	Reserved for local use.
FacilityLpr	LOG_LPR	The line printer spooling system.
FacilityMail	LOG_MAIL	The mail system.
FacilityNews	LOG_NEWS	The network news system.
FacilitySyslog	LOG_SYSLOG	Messages generated internally by syslogd.
FacilityUser	LOG_USER	Messages generated by user processes. This is the default facility identifier if none is specified.
FacilityUucp	LOG_UUCP	The UUCP system.

9.34 TimePeriod

Time periods can be used to specify when hosts/services should be checked or to limit when notifications should be sent out.

Note

Icinga 2 versions < 2.6.0 require the import of the legacy-timeperiod template.

Examples:

```
object TimePeriod "nonworkhours" {
  display_name = "Icinga 2 TimePeriod for non working hours"

  ranges = {
    monday = "00:00-8:00,17:00-24:00"
    tuesday = "00:00-8:00,17:00-24:00"
    wednesday = "00:00-8:00,17:00-24:00"
    thursday = "00:00-8:00,17:00-24:00"
    friday = "00:00-8:00,16:00-24:00"
    saturday = "00:00-24:00"
    sunday = "00:00-24:00"
  }
}

object TimePeriod "exampledays" {
    display_name = "Icinga 2 TimePeriod for random example days"

    ranges = {
        //We still believe in Santa, no peeking!
        //Applies every 25th of December every year
        "december 25" = "00:00-24:00"

        //Any point in time can be specified,
        //but you still have to use a range
        "2038-01-19" = "03:13-03:15"

        //Evey 3rd day from the second monday of February
        //to 8th of November
        "monday 2 february - november 8 / 3" = "00:00-24:00"
    }
}
```

Additional examples can be found here.

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the time period.
ranges	Dictionary	**Required.** A dictionary containing information which days and durations apply to this timeperiod.

Name	Type	Description
prefer_includes	Boolean	**Optional.** Whether to prefer timeperiods `includes` or `excludes`. Default to true.
excludes	Array of object names	**Optional.** An array of timeperiods, which should exclude from your timerange.
includes	Array of object names	**Optional.** An array of timeperiods, which should include into your timerange

Runtime Attributes:

Name	Type	Description
is_inside	Boolean	Whether we're currently inside this timeperiod.

9.35 User

A user.

Example:

```
object User "icingaadmin" {
  display_name = "Icinga 2 Admin"
  groups = [ "icingaadmins" ]
  email = "icinga@localhost"
  pager = "icingaadmin@localhost.localdomain"

  period = "24x7"

  states = [ OK, Warning, Critical, Unknown ]
  types = [ Problem, Recovery ]

  vars.additional_notes = "This is the Icinga 2 Admin account."
}
```

Available notification state filters:

```
OK
Warning
Critical
Unknown
Up
```

`Down`

Available notification type filters:

```
DowntimeStart
DowntimeEnd
DowntimeRemoved
Custom
Acknowledgement
Problem
Recovery
FlappingStart
FlappingEnd
```

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the user.
email	String	**Optional.** An email string for this user. Useful for notification commands.
pager	String	**Optional.** A pager string for this user. Useful for notification commands.
vars	Dictionary	**Optional.** A dictionary containing custom attributes that are specific to this user.
groups	Array of object names	**Optional.** An array of group names.
enable_notifications	Boolean	**Optional.** Whether notifications are enabled for this user.
period	Object name	**Optional.** The name of a time period which determines when a notification for this user should be triggered. Not set by default.
types	Array	**Optional.** A set of type filters when a notification for this user should be triggered. By default everything is matched.

Name	Type	Description
states	Array	**Optional.** A set of state filters when a notification for this should be triggered. By default everything is matched.

Runtime Attributes:

Name	Type	Description
last_notification	Timestamp	When the last notification was sent for this user (as a UNIX timestamp).

9.36 UserGroup

A user group.

Best Practice

Assign user group members using the group assign rules.

Example:

```
object UserGroup "icingaadmins" {
    display_name = "Icinga 2 Admin Group"
}
```

Configuration Attributes:

Name	Type	Description
display_name	String	**Optional.** A short description of the user group.
groups	Array of object names	**Optional.** An array of nested group names.

9.37 Zone

Zone objects are used to specify which Icinga 2 instances are located in a zone.
Please read the distributed monitoring chapter for additional details. Example:

```
object Zone "master" {
 endpoints = [ "icinga2-master1.localdomain", "icinga2-master2.localdomain" ]

}
```

```
object Zone "satellite" {
  endpoints = [ "icinga2-satellite1.localdomain" ]
  parent = "master"
}
```

Configuration Attributes:

Name	Type	Description
endpoints	Array of object names	**Optional.** Array of endpoint names located in this zone.
parent	Object name	**Optional.** The name of the parent zone.
global	Boolean	**Optional.** Whether configuration files for this zone should be synced to all endpoints. Defaults to `false`.

Zone objects cannot currently be created with the API.

10 Icinga Template Library

The Icinga Template Library (ITL) implements standard templates and object definitions.

There is a subset of templates and object definitions available:

- Generic ITL templates
- CheckCommand definitions for Icinga 2 (this includes icinga, cluster, cluster-zone, ido, etc.)
- CheckCommand definitions for Monitoring Plugins
- CheckCommand definitions for Icinga 2 Windows Plugins
- CheckCommand definitions for NSClient++
- CheckCommand definitions for Manubulon SNMP
- Contributed CheckCommand definitions

The ITL content is updated with new releases. Please do not modify templates and/or objects as changes will be overridden without further notice.

You are advised to create your own CheckCommand definitions in `/etc/icinga2`.

10.1 Generic Templates

By default the generic templates are included in the icinga2.conf configuration file:

```
include <itl>
```

These templates are imported by the provided example configuration.

> **Note**:
>
> These templates are built into the binaries. By convention all command and timeperiod objects should import these templates.

10.1.1 plugin-check-command

Command template for check plugins executed by Icinga 2.

The `plugin-check-command` command does not support any vars.

By default this template is automatically imported into all CheckCommand definitions.

10.1.2 plugin-notification-command

Command template for notification scripts executed by Icinga 2.

The `plugin-notification-command` command does not support any vars.

By default this template is automatically imported into all NotificationCommand definitions.

10.1.3 plugin-event-command

Command template for event handler scripts executed by Icinga 2.

The `plugin-event-command` command does not support any vars.

By default this template is automatically imported into all EventCommand definitions.

10.1.4 legacy-timeperiod

Timeperiod template for Timeperiod objects.

The `legacy-timeperiod` timeperiod does not support any vars.

By default this template is automatically imported into all TimePeriod definitions.

10.2 Check Commands

These check commands are embedded into Icinga 2 and do not require any external plugin scripts.

10.2.1 icinga

Check command for the built-in `icinga` check. This check returns performance data for the current Icinga instance and optionally allows for minimum version checks.

Custom attributes passed as command parameters:

Name	Description
icinga_min_version	**Optional.** Required minimum Icinga 2 version, e.g. `2.8.0`. If not satisfied, the state changes to `Critical`. Release packages only.

10.2.2 cluster

Check command for the built-in `cluster` check. This check returns performance data for the current Icinga instance and connected endpoints.

The `cluster` check command does not support any vars.

10.2.3 cluster-zone

Check command for the built-in `cluster-zone` check.

Custom attributes passed as command parameters:

Name	Description
cluster_zone	**Required.** The zone name. Defaults to `$host.name$`.

Name	Description
cluster_lag_warning	**Optional.** Warning threshold for log lag in seconds. Applies if the log lag is greater than the threshold.
cluster_lag_critical	**Optional.** Critical threshold for log lag in seconds. Applies if the log lag is greater than the threshold.

10.2.4 ido

Check command for the built-in `ido` check.

Custom attributes passed as command parameters:

Name	Description
ido_type	**Required.** The type of the IDO connection object. Can be either "IdoMysqlConnection" or "IdoPgsqlConnection".
ido_name	**Required.** The name of the IDO connection object.
ido_queries_warning	**Optional.** Warning threshold for queries/s. Applies if the rate is lower than the threshold.
ido_queries_critical	**Optional.** Critical threshold for queries/s. Applies if the rate is lower than the threshold.

Name	Description
ido_pending_queries_warning	**Optional.** Warning threshold for pending queries. Applies if pending queries are higher than the threshold. Supersedes the `ido_queries` thresholds above.
ido_pending_queries_critical	**Optional.** Critical threshold for pending queries. Applies if pending queries are higher than the threshold. Supersedes the `ido_queries` thresholds above.

10.2.5 dummy

Check command for the built-in `dummy` check. This allows to set a check result state and output and can be used in freshness checks or runtime object checks. In contrast to the check_dummy plugin, Icinga 2 implements a light-weight in memory check with 2.9+.

Custom attributes passed as command parameters:

Name	Description
dummy_state	**Optional.** The state. Can be one of 0 (ok), 1 (warning), 2 (critical) and 3 (unknown). Defaults to 0.
dummy_text	**Optional.** Plugin output. Defaults to "Check was successful.".

10.2.6 passive

Specialised check command object for passive checks which uses the functionality of the "dummy" check command with appropriate default values.

Custom attributes passed as command parameters:

Name	Description
dummy_state	**Optional.** The state. Can be one of 0 (ok), 1 (warning), 2 (critical) and 3 (unknown). Defaults to 3.
dummy_text	**Optional.** Plugin output. Defaults to "No Passive Check Result Received.".

10.2.7 random

Check command for the built-in `random` check. This check returns random states and adds the check source to the check output.

For test and demo purposes only. The `random` check command does not support any vars.

10.2.8 exception

Check command for the built-in `exception` check. This check throws an exception.

For test and demo purposes only. The `exception` check command does not support any vars.

10.3 Plugin Check Commands for Monitoring Plugins

The Plugin Check Commands provides example configuration for plugin check commands provided by the Monitoring Plugins project.

By default the Plugin Check Commands are included in the icinga2.conf configuration file:

```
include <plugins>
```

The plugin check commands assume that there's a global constant named `PluginDir` which contains the path of the plugins from the Monitoring Plugins project.

Note: If there are command parameters missing for the provided CheckCommand definitions please kindly send a patch upstream. This should include an update for the ITL CheckCommand itself and this documentation section.

10.3.1 apt

The plugin apt checks for software updates on systems that use package management systems based on the apt-get(8) command found in Debian based systems.

Custom attributes passed as command parameters:

Name	Description
apt_extra_opts	**Optional.** Read options from an ini file.
apt_upgrade	**Optional.** [Default] Perform an upgrade. If an optional OPTS argument is provided, apt-get will be run with these command line options instead of the default.
apt_dist_upgrade	**Optional.** Perform a dist-upgrade instead of normal upgrade. Like with -U OPTS can be provided to override the default options.
apt_include	**Optional.** Include only packages matching REGEXP. Can be specified multiple times the values will be combined together.
apt_exclude	**Optional.** Exclude packages matching REGEXP from the list of packages that would otherwise be included. Can be specified multiple times.
apt_critical	**Optional.** If the full package information of any of the upgradable packages match this REGEXP, the plugin will return CRITICAL status. Can be specified multiple times.
apt_timeout	**Optional.** Seconds before plugin times out (default: 10).
apt_only_critical	**Optional.** Only warn about critical upgrades.

10.3.2 breeze

The check_breeze plugin reports the signal strength of a Breezecom wireless equipment.

Custom attributes passed as command parameters:

Name	Description
breeze_hostname	**Required.** Name or IP address of host to check. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
breeze_community	**Optional.** SNMPv1 community. Defaults to "public".
breeze_warning	**Required.** Percentage strength below which a WARNING status will result. Defaults to 50.
breeze_critical	**Required.** Percentage strength below which a WARNING status will result. Defaults to 20.

10.3.3 by_ssh

The check_by_ssh plugin uses SSH to execute commands on a remote host.

Custom attributes passed as command parameters:

Name	Description
by_ssh_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
by_ssh_port	**Optional.** The SSH port. Defaults to 22.
by_ssh_command	**Required.** The command that should be executed. Can be an array if multiple arguments should be passed to `check_by_ssh`.

Name	Description
by_ssh_arguments	**Optional.** A dictionary with arguments for the command. This works exactly like the 'arguments' dictionary for ordinary CheckCommands.
by_ssh_logname	**Optional.** The SSH username.
by_ssh_identity	**Optional.** The SSH identity.
by_ssh_quiet	**Optional.** Whether to suppress SSH warnings. Defaults to false.
by_ssh_warn	**Optional.** The warning threshold.
by_ssh_crit	**Optional.** The critical threshold.
by_ssh_timeout	**Optional.** The timeout in seconds.
by_ssh_options	**Optional.** Call ssh with '-o OPTION' (multiple options may be specified as an array).
by_ssh_ipv4	**Optional.** Use IPv4 connection. Defaults to false.

Name	Description
by_ssh_ipv6	**Optional.** Use IPv6 connection. Defaults to false.
by_ssh_skip_stderr	**Optional.** Ignore all or (if specified) first n lines on STDERR.

10.3.4 clamd

The check_clamd plugin tests CLAMD connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
clamd_address	**Required.** The host's address or unix socket (must be an absolute path).
clamd_port	**Optional.** Port number (default: none).
clamd_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
clamd_all	**Optional.** All expect strings need to occur in server response. Defaults to false.

Name	Description
clamd_escape_send	**Optional.** Enable usage of \n, \r, \t or \\ in send string.
clamd_send	**Optional.** String to send to the server.
clamd_escape_quit	**Optional.** Enable usage of \n, \r, \t or \\ in quit string.
clamd_quit	**Optional.** String to send server to initiate a clean close of the connection.
clamd_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit. Defaults to crit.
clamd_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit. Defaults to warn.
clamd_jail	**Optional.** Hide output from TCP socket.
clamd_maxbytes	**Optional.** Close connection once more than this number of bytes are received.

Name	Description
clamd_delay	**Optional.** Seconds to wait between sending string and polling for response.
clamd_certificate	**Optional.** Minimum number of days a certificate has to be valid. 1st value is number of days for warning, 2nd is critical (if not specified: 0) – separated by comma.
clamd_ssl	**Optional.** Use SSL for the connection. Defaults to false.
clamd_wtime	**Optional.** Response time to result in warning status (seconds).
clamd_ctime	**Optional.** Response time to result in critical status (seconds).
clamd_timeout	**Optional.** Seconds before connection times out. Defaults to 10.
clamd_ipv4	**Optional.** Use IPv4 connection. Defaults to false.

Name	Description
clamd_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.5 dhcp

The check_dhcp plugin tests the availability of DHCP servers on a network.

Custom attributes passed as command parameters:

Name	Description
dhcp_serverip	**Optional.** The IP address of the DHCP server which we should get a response from.
dhcp_requestedip	**Optional.** The IP address which we should be offered by a DHCP server.
dhcp_timeout	**Optional.** The timeout in seconds.
dhcp_interface	**Optional.** The interface to use.
dhcp_mac	**Optional.** The MAC address to use in the DHCP request.
dhcp_unicast	**Optional.** Whether to use unicast requests. Defaults to false.

10.3.6 dig

The check_dig plugin test the DNS service on the specified host using dig.

Custom attributes passed as command parameters:

Name	Description
dig_server	**Optional.** The DNS server to query. Defaults to "127.0.0.1".
dig_port	**Optional.** Port number (default: 53).
dig_lookup	**Required.** The address that should be looked up.
dig_record_type	**Optional.** Record type to lookup (default: A).
dig_expected_address	**Optional.** An address expected to be in the answer section. If not set, uses whatever was in -l.
dig_arguments	**Optional.** Pass STRING as argument(s) to dig.
dig_retries	**Optional.** Number of retries passed to dig, timeout is divided by this value (Default: 3).
dig_warning	**Optional.** Response time to result in warning status (seconds).

Name	Description
dig_critical	**Optional.** Response time to result in critical status (seconds).
dig_timeout	**Optional.** Seconds before connection times out (default: 10).
dig_ipv4	**Optional.** Force dig to only use IPv4 query transport. Defaults to false.
dig_ipv6	**Optional.** Force dig to only use IPv6 query transport. Defaults to false.

10.3.7 disk

The check_disk plugin checks the amount of used disk space on a mounted file system and generates an alert if free space is less than one of the threshold values.

Custom attributes passed as command parameters:

Name	Description
disk_wfree	**Optional.** The free space warning threshold. Defaults to "20%". If the percent sign is omitted, units from `disk_units` are used.

Name	Description
disk_cfree	**Optional.** The free space critical threshold. Defaults to "10%". If the percent sign is omitted, units from `disk_units` are used.
disk_inode_wfree	**Optional.** The free inode warning threshold.
disk_inode_cfree	**Optional.** The free inode critical threshold.
disk_partition	**Optional.** The partition. **Deprecated in 2.3.**
disk_partition_excluded	**Optional.** The excluded partition. **Deprecated in 2.3.**
disk_partitions	**Optional.** The partition(s). Multiple partitions must be defined as array.
disk_partitions_excluded	**Optional.** The excluded partition(s). Multiple partitions must be defined as array.
disk_clear	**Optional.** Clear thresholds. May be true or false.
disk_exact_match	**Optional.** For paths or partitions specified with -p, only check for exact paths. May be true or false.
disk_errors_only	**Optional.** Display only devices/mountpoints with errors. May be true or false.
disk_ignore_reserved	**Optional.** If set, account root-reserved blocks are not accounted for freespace in perfdata. May be true or false.
disk_group	**Optional.** Group paths. Thresholds apply to (free-)space of all partitions together.

Name	Description
disk_kilobytes	**Optional.** Same as –units kB. May be true or false.
disk_local	**Optional.** Only check local filesystems. May be true or false.
disk_stat_remote_fs	**Optional.** Only check local filesystems against thresholds. Yet call stat on remote filesystems to test if they are accessible (e.g. to detect Stale NFS Handles). May be true or false.
disk_mountpoint	**Optional.** Display the mountpoint instead of the partition. May be true or false.
disk_megabytes	**Optional.** Same as –units MB. May be true or false.
disk_all	**Optional.** Explicitly select all paths. This is equivalent to -R '.*'. May be true or false.
disk_eregi_path	**Optional.** Case insensitive regular expression for path/partition. Multiple regular expression strings must be defined as array.
disk_ereg_path	**Optional.** Regular expression for path or partition. Multiple regular expression strings must be defined as array.
disk_ignore_eregi_path	**Optional.** Regular expression to ignore selected path/partition (case insensitive). Multiple regular expression strings must be defined as array.

Name	Description
disk_ignore_ereg_path	**Optional.** Regular expression to ignore selected path or partition. Multiple regular expression strings must be defined as array.
disk_timeout	**Optional.** Seconds before connection times out (default: 10).
disk_units	**Optional.** Choose bytes, kB, MB, GB, TB (default: MB).
disk_exclude_type	**Optional.** Ignore all filesystems of indicated type. Multiple regular expression strings must be defined as array. Defaults to “none”, “tmpfs”, “sysfs”, “proc”, “configfs”, “devtmpfs”, “devfs”, “mtmfs”, “tracefs”, “cgroup”, “fuse.gvfsd-fuse”, “fuse.gvfs-fuse-daemon”, “fdescfs”, “overlay”, “nsfs”.

10.3.8 disk_smb

The check_disk_smb plugin uses the `smbclient` binary to check SMB shares.

Custom attributes passed as command parameters:

Name	Description
disk_smb_hostname	**Required.** NetBIOS name of the server.
disk_smb_share	**Required.** Share name being queried.
disk_smb_workgroup	**Optional.** Workgroup or Domain used (defaults to ‘WORKGROUP’ if omitted).

Name	Description
disk_smb_address	**Optional.** IP address of the host (only necessary if host belongs to another network).
disk_smb_username	**Optional.** Username for server log-in (defaults to 'guest' if omitted).
disk_smb_password	**Optional.** Password for server log-in (defaults to an empty password if omitted).
disk_smb_wused	**Optional.** The used space warning threshold. Defaults to "85%". If the percent sign is omitted, use optional disk units.
disk_smb_cused	**Optional.** The used space critical threshold. Defaults to "95%". If the percent sign is omitted, use optional disk units.
disk_smb_port	**Optional.** Connection port, e.g. `139` or `445`. Defaults to `smbclient` default if omitted.

10.3.9 dns

The check_dns plugin uses the nslookup program to obtain the IP address for the given host/domain query. An optional DNS server to use may be specified. If no DNS server is specified, the default server(s) specified in `/etc/resolv.conf` will be used.

Custom attributes passed as command parameters:

Name	Description
dns_lookup	**Optional.** The hostname or IP to query the DNS for. Defaults to "$host_name$".

Name	Description
dns_server	**Optional.** The DNS server to query. Defaults to the server configured in the OS.
dns_query_type	**Optional.** The DNS record query type where TYPE =(A, AAAA, SRV, TXT, MX, ANY). The default query type is 'A' (IPv4 host entry)
dns_expected_answers	**Optional.** The answer(s) to look for. A hostname must end with a dot. Multiple answers must be defined as array.
dns_authoritative	**Optional.** Expect the server to send an authoritative answer.
dns_accept_cname	**Optional.** Accept cname responses as a valid result to a query.
dns_wtime	**Optional.** Return warning if elapsed time exceeds value.

Name	Description
dns_ctime	**Optional.** Return critical if elapsed time exceeds value.
dns_timeout	**Optional.** Seconds before connection times out. Defaults to 10.

10.3.10 file_age

The check_file_age plugin checks a file's size and modification time to make sure it's not empty and that it's sufficiently recent.

Custom attributes passed as command parameters:

Name	Description
file_age_file	**Required.** File to monitor.
file_age_warning_time	**Optional.** File must be no more than this many seconds old as warning threshold. Defaults to "240s".
file_age_critical_time	**Optional.** File must be no more than this many seconds old as critical threshold. Defaults to "600s".
file_age_warning_size	**Optional.** File must be at least this many bytes long as warning threshold. No default given.
file_age_critical_size	**Optional.** File must be at least this many bytes long as critical threshold. Defaults to "0B".
file_age_ignoremissing	**Optional.** Return OK if the file does not exist. Defaults to false.

10.3.11 flexlm

The check_flexlm plugin checks available flexlm license managers. Requires the `lmstat` command.

Custom attributes passed as command parameters:

Name	Description
flexlm_licensefile	**Required.** Name of license file (usually license.dat).
flexlm_timeout	**Optional.** Plugin time out in seconds. Defaults to 15.

10.3.12 fping4

The check_fping plugin uses the `fping` command to ping the specified host for a fast check. Note that it is necessary to set the `suid` flag on `fping`.

This CheckCommand expects an IPv4 address.

Custom attributes passed as command parameters:

Name	Description
fping_address	**Optional.** The host's IPv4 address. Defaults to "*address*".
fping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 100.
fping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 5.
fping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 200.
fping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 15.
fping_number	**Optional.** The number of packets to send. Defaults to 5.
fping_interval	**Optional.** The interval between packets in milli-seconds. Defaults to 500.

Name	Description
fping_bytes	**Optional.** The size of ICMP packet.
fping_target_timeout	**Optional.** The target timeout in milli-seconds.
fping_source_ip	**Optional.** The name or ip address of the source ip.
fping_source_interface	**Optional.** The source interface name.

10.3.13 fping6

The check_fping plugin will use the `fping` command to ping the specified host for a fast check. Note that it is necessary to set the `suid` flag on `fping`.

This CheckCommand expects an IPv6 address.

Custom attributes passed as command parameters:

Name	Description
fping_address	**Optional.** The host's IPv6 address. Defaults to "*address*6".
fping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 100.
fping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 5.

Name	Description
fping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 200.
fping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 15.
fping_number	**Optional.** The number of packets to send. Defaults to 5.
fping_interval	**Optional.** The interval between packets in milli-seconds. Defaults to 500.
fping_bytes	**Optional.** The size of ICMP packet.
fping_target_timeout	**Optional.** The target timeout in milli-seconds.
fping_source_ip	**Optional.** The name or ip address of the source ip.
fping_source_interface	**Optional.** The source interface name.

10.3.14 ftp

The check_ftp plugin tests FTP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
ftp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ftp_port	**Optional.** The FTP port number.
ftp_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
ftp_all	**Optional.** All expect strings need to occur in server response. Defaults to false.
ftp_escape_send	**Optional.** Enable usage of \n, \r, \t or \\ in send string.
ftp_send	**Optional.** String to send to the server.
ftp_escape_quit	**Optional.** Enable usage of \n, \r, \t or \\ in quit string.
ftp_quit	**Optional.** String to send server to initiate a clean close of the connection.

Name	Description
ftp_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit. Defaults to crit.
ftp_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit. Defaults to warn.
ftp_jail	**Optional.** Hide output from TCP socket.
ftp_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
ftp_delay	**Optional.** Seconds to wait between sending string and polling for response.
ftp_certificate	**Optional.** Minimum number of days a certificate has to be valid. 1st value is number of days for warning, 2nd is critical (if not specified: 0) – separated by comma.

Name	Description
ftp_ssl	**Optional.** Use SSL for the connection. Defaults to false.
ftp_wtime	**Optional.** Response time to result in warning status (seconds).
ftp_ctime	**Optional.** Response time to result in critical status (seconds).
ftp_timeout	**Optional.** Seconds before connection times out. Defaults to 10.
ftp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
ftp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.15 game

The check_game plugin tests game server connections with the specified host. This plugin uses the 'qstat' command, the popular game server status query tool. If you don't have the package installed, you will need to download or install the package `quakestat` before you can use this plugin.

Custom attributes passed as command parameters:

Name	Description
game_game	**Required.** Name of the game.

Name	Description
game_ipaddress	**Required.** Ipaddress of the game server to query.
game_timeout	**Optional.** Seconds before connection times out. Defaults to 10.
game_port	**Optional.** Port to connect to.
game_gamefield	**Optional.** Field number in raw qstat output that contains game name.
game_mapfield	**Optional.** Field number in raw qstat output that contains map name.
game_pingfield	**Optional.** Field number in raw qstat output that contains ping time.
game_gametime	**Optional.** Field number in raw qstat output that contains game time.
game_hostname	**Optional.** Name of the host running the game.

10.3.16 hostalive

Check command object for the check_ping plugin with host check default values. This variant uses the host's `address` attribute if available and falls back to using the `address6` attribute if the `address` attribute is not set.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 3000.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 80.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 5000.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 100.
ping_packets	**Optional.** The number of packets to send. Defaults to 5.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.3.17 hostalive4

Check command object for the check_ping plugin with host check default values. This variant uses the host's `address` attribute.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The host's IPv4 address. Defaults to "*address*".
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 3000.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 80.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 5000.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 100.
ping_packets	**Optional.** The number of packets to send. Defaults to 5.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.3.18 hostalive6

Check command object for the check_ping plugin with host check default values. This variant uses the host's `address6` attribute.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The host's IPv6 address. Defaults to "*address*6".
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 3000.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 80.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 5000.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 100.
ping_packets	**Optional.** The number of packets to send. Defaults to 5.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.3.19 hpjd

The check_hpjd plugin tests the state of an HP printer with a JetDirect card. Net-snmp must be installed on the computer running the plugin.

Custom attributes passed as command parameters:

Name	Description
hpjd_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
hpjd_port	**Optional.** The host's SNMP port. Defaults to 161.
hpjd_community	**Optional.** The SNMP community. Defaults to "public".

10.3.20 http

The check_http plugin tests the HTTP service on the specified host. It can test normal (http) and secure (https) servers, follow redirects, search for strings and regular expressions, check connection times, and report on certificate expiration times.

Custom attributes passed as command parameters:

Name	Description
http_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
http_vhost	**Optional.** The virtual host that should be sent in the "Host" header.
http_uri	**Optional.** The request URI for GET or POST. Defaults to `/`.

Name	Description
http_port	**Optional.** The TCP port. Defaults to 80 when not using SSL, 443 otherwise.
http_ssl	**Optional.** Whether to use SSL. Defaults to false.
http_ssl_force_tlsv1	**Optional.** Whether to force TLSv1.
http_ssl_force_tlsv1_1	**Optional.** Whether to force TLSv1.1.
http_ssl_force_tlsv1_2	**Optional.** Whether to force TLSv1.2.
http_ssl_force_sslv2	**Optional.** Whether to force SSLv2.
http_ssl_force_sslv3	**Optional.** Whether to force SSLv3.
http_ssl_force_tlsv1_or_higher	**Optional.** Whether to force TLSv1 or higher.
http_ssl_force_tlsv1_1_or_higher	**Optional.** Whether to force TLSv1.1 or higher.
http_ssl_force_tlsv1_2_or_higher	**Optional.** Whether to force TLSv1.2 or higher.
http_ssl_force_sslv2_or_higher	**Optional.** Whether to force SSLv2 or higher.
http_ssl_force_sslv3_or_higher	**Optional.** Whether to force SSLv3 or higher.
http_sni	**Optional.** Whether to use SNI. Defaults to false.
http_auth_pair	**Optional.** Add 'username:password' authorization pair.
http_proxy_auth_pair	**Optional.** Add 'username:password' authorization pair for proxy.
http_ignore_body	**Optional.** Don't download the body, just the headers.
http_linespan	**Optional.** Allow regex to span newline.
http_expect_body_regex	**Optional.** A regular expression which the body must match against. Incompatible with http_ignore_body.

Name	Description
http_expect_body_eregi	**Optional.** A case-insensitive expression which the body must match against. Incompatible with http_ignore_body.
http_invertregex	**Optional.** Changes behavior of http_expect_body_regex and http_expect_body_eregi to return CRITICAL if found, OK if not.
http_warn_time	**Optional.** The warning threshold.
http_critical_time	**Optional.** The critical threshold.
http_expect	**Optional.** Comma-delimited list of strings, at least one of them is expected in the first (status) line of the server response. Default: HTTP/1.
http_certificate	**Optional.** Minimum number of days a certificate has to be valid. Port defaults to 443. When this option is used the URL is not checked. The first parameter defines the warning threshold (in days), the second parameter the critical threshold (in days). (Example `http_certificate = "30,20"`).
http_clientcert	**Optional.** Name of file contains the client certificate (PEM format).
http_privatekey	**Optional.** Name of file contains the private key (PEM format).
http_headerstring	**Optional.** String to expect in the response headers.
http_string	**Optional.** String to expect in the content.
http_post	**Optional.** URL encoded http POST data.
http_method	**Optional.** Set http method (for example: HEAD, OPTIONS, TRACE, PUT, DELETE).
http_maxage	**Optional.** Warn if document is more than seconds old.

Name	Description
http_contenttype	**Optional.** Specify Content-Type header when POSTing.
http_useragent	**Optional.** String to be sent in http header as User Agent.
http_header	**Optional.** Any other tags to be sent in http header.
http_extendedperfdata	**Optional.** Print additional perfdata. Defaults to false.
http_onredirect	**Optional.** How to handle redirect pages. Possible values: "ok" (default), "warning", "critical", "follow", "sticky" (like follow but stick to address), "stickyport" (like sticky but also to port)
http_pagesize	**Optional.** Minimum page size required:Maximum page size required.
http_timeout	**Optional.** Seconds before connection times out.
http_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
http_ipv6	**Optional.** Use IPv6 connection. Defaults to false.
http_link	**Optional.** Wrap output in HTML link. Defaults to false.
http_verbose	**Optional.** Show details for command-line debugging. Defaults to false.

10.3.21 icmp

The check_icmp plugin check_icmp allows for checking multiple hosts at once compared to `check_ping`. The main difference is that check_ping executes the system's ping(1) command and parses its output while `check_icmp` talks ICMP itself. `check_icmp` must be installed with `setuid` root.

Custom attributes passed as command parameters:

Name	Description
icmp_address	**Optional.** The host's address. This can either be a single address or an array of addresses. Defaults to "*address*".
icmp_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 100.
icmp_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 5.
icmp_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 200.
icmp_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 15.
icmp_source	**Optional.** The source IP address to send packets from.
icmp_packets	**Optional.** The number of packets to send. Defaults to 5.
icmp_packet_interval	**Optional** The maximum packet interval. Defaults to 80 (milliseconds).

Name	Description
icmp_target_interval	**Optional.** The maximum target interval.
icmp_hosts_alive	**Optional.** The number of hosts which have to be alive for the check to succeed.
icmp_data_bytes	**Optional.** Payload size for each ICMP request. Defaults to 8.
icmp_timeout	**Optional.** The plugin timeout in seconds. Defaults to 10 (seconds).
icmp_ttl	**Optional.** The TTL on outgoing packets.

10.3.22 imap

The check_imap plugin tests IMAP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
imap_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
imap_port	**Optional.** The port that should be checked. Defaults to 143.
imap_escape	**Optional.** Can use \n, \r, \t or \ in send or quit string. Must come before send or quit option. Default: nothing added to send, \r\n added to end of quit.
imap_send	**Optional.** String to send to the server.
imap_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
imap_all	**Optional.** All expect strings need to occur in server response. Default is any.
imap_quit	**Optional.** String to send server to initiate a clean close of the connection.

Name	Description
imap_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit (default: crit).
imap_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit (default: warn).
imap_jail	**Optional.** Hide output from TCP socket.
imap_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
imap_delay	**Optional.** Seconds to wait between sending string and polling for response.
imap_certificate_age	**Optional.** Minimum number of days a certificate has to be valid.
imap_ssl	**Optional.** Use SSL for the connection.
imap_warning	**Optional.** Response time to result in warning status (seconds).

Name	Description
imap_critical	**Optional.** Response time to result in critical status (seconds).
imap_timeout	**Optional.** Seconds before connection times out (default: 10).
imap_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
imap_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.23 ldap

The check_ldap plugin can be used to check LDAP servers.

The plugin can also be used for monitoring ldaps connections instead of the deprecated `check_ldaps`. This can be ensured by enabling `ldap_starttls` or `ldap_ssl`.

Custom attributes passed as command parameters:

Name	Description
ldap_address	**Optional.** Host name, IP Address, or unix socket (must be an absolute path). Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
ldap_port	**Optional.** Port number. Defaults to 389.
ldap_attr	**Optional.** LDAP attribute to search for (default: “(objectclass=*)“)
ldap_base	**Required.** LDAP base (eg. ou=myunit,o=myorg,c=at).
ldap_bind	**Optional.** LDAP bind DN (if required).
ldap_pass	**Optional.** LDAP password (if required).
ldap_starttls	**Optional.** Use STARTSSL mechanism introduced in protocol version 3.
ldap_ssl	**Optional.** Use LDAPS (LDAP v2 SSL method). This also sets the default port to 636.
ldap_v2	**Optional.** Use LDAP protocol version 2 (enabled by default).
ldap_v3	**Optional.** Use LDAP protocol version 3 (disabled by default)

Name	Description
ldap_warning	**Optional.** Response time to result in warning status (seconds).
ldap_critical	**Optional.** Response time to result in critical status (seconds).
ldap_warning_entries	**Optional.** Number of found entries to result in warning status.
ldap_critical_entries	**Optional.** Number of found entries to result in critical status.
ldap_timeout	**Optional.** Seconds before connection times out (default: 10).
ldap_verbose	**Optional.** Show details for command-line debugging (disabled by default)

10.3.24 load

The check_load plugin tests the current system load average.

Custom attributes passed as command parameters:

Name	Description
load_wload1	**Optional.** The 1-minute warning threshold. Defaults to 5.
load_wload5	**Optional.** The 5-minute warning threshold. Defaults to 4.
load_wload15	**Optional.** The 15-minute warning threshold. Defaults to 3.
load_cload1	**Optional.** The 1-minute critical threshold. Defaults to 10.
load_cload5	**Optional.** The 5-minute critical threshold. Defaults to 6.
load_cload15	**Optional.** The 15-minute critical threshold. Defaults to 4.
load_percpu	**Optional.** Divide the load averages by the number of CPUs (when possible). Defaults to false.

10.3.25 mailq

The check_mailq plugin checks the number of messages in the mail queue (supports multiple sendmail queues, qmail).

Custom attributes passed as command parameters:

Name	Description
mailq_warning	**Required.** Min. number of messages in queue to generate warning.
mailq_critical	**Required.** Min. number of messages in queue to generate critical alert (w < c).
mailq_domain_warning	**Optional.** Min. number of messages for same domain in queue to generate warning
mailq_domain_critical	**Optional.** Min. number of messages for same domain in queue to generate critical alert (W < C).
mailq_timeout	**Optional.** Plugin timeout in seconds (default = 15).
mailq_servertype	**Optional.** [sendmail | qmail | postfix | exim | nullmailer] (default = autodetect).

Name	Description
mailq_sudo	**Optional.** Use sudo to execute the mailq command.

10.3.26 mysql

The check_mysql plugin tests connections to a MySQL server.

Custom attributes passed as command parameters:

Name	Description
mysql_hostname	**Optional.** Host name, IP Address, or unix socket (must be an absolute path).
mysql_port	**Optional.** Port number (default: 3306).
mysql_socket	**Optional.** Use the specified socket (has no effect if `mysql_hostname` is used).
mysql_ignore_auth	**Optional.** Ignore authentication failure and check for mysql connectivity only.
mysql_database	**Optional.** Check database with indicated name.
mysql_file	**Optional.** Read from the specified client options file.
mysql_group	**Optional.** Use a client options group.
mysql_username	**Optional.** Connect using the indicated username.
mysql_password	**Optional.** Use the indicated password to authenticate the connection.
mysql_check_slave	**Optional.** Check if the slave thread is running properly.
mysql_warning	**Optional.** Exit with WARNING status if slave server is more than INTEGER seconds behind master.
mysql_critical	**Optional.** Exit with CRITICAL status if slave server is more then INTEGER seconds behind master.
mysql_ssl	**Optional.** Use ssl encryption.
mysql_cacert	**Optional.** Path to CA signing the cert.
mysql_cert	**Optional.** Path to SSL certificate.
mysql_key	**Optional.** Path to private SSL key.
mysql_cadir	**Optional.** Path to CA directory.
mysql_ciphers	**Optional.** List of valid SSL ciphers.

10.3.27 mysql_query

The check_mysql_query plugin checks a query result against threshold levels. The result from the query should be numeric. For extra security, create a user with minimal access.

Note: You must specify `mysql_query_password` with an empty string to force an empty password, overriding any my.cnf settings.

Custom attributes passed as command parameters:

Name	Description
mysql_query_hostname	**Optional.** Host name, IP Address, or unix socket (must be an absolute path).
mysql_query_port	**Optional.** Port number (default: 3306).
mysql_query_database	**Optional.** Check database with indicated name.
mysql_query_file	**Optional.** Read from the specified client options file.
mysql_query_group	**Optional.** Use a client options group.
mysql_query_username	**Optional.** Connect using the indicated username.
mysql_query_password	**Optional.** Use the indicated password to authenticate the connection.
mysql_query_execute	**Required.** SQL Query to run on the MySQL Server.
mysql_query_warning	**Optional.** Exit with WARNING status if query is outside of the range (format: start:end).
mysql_query_critical	**Optional.** Exit with CRITICAL status if query is outside of the range.

10.3.28 negate

The negate plugin negates the status of a plugin (returns OK for CRITICAL and vice-versa). Additional switches can be used to control which state becomes what.

Custom attributes passed as command parameters:

Name	Description
negate_timeout	**Optional.** Seconds before plugin times out (default: 11).
negate_timeout_result	**Optional.** Custom result on Negate timeouts, default to UNKNOWN.
negate_ok	**Optional.** OK, WARNING, CRITICAL or UNKNOWN.
negate_warning	Numeric values are accepted.
negate_critical	If nothing is specified,
negate_unknown	permutes OK and CRITICAL.

Name	Description
negate_substitute	**Optional.** Substitute output text as well. Will only substitute text in CAPITALS.
negate_command	**Required.** Command to be negated.
negate_arguments	**Optional.** Arguments for the negated command.

10.3.29 nrpe

The `check_nrpe` plugin can be used to query an NRPE server or NSClient++. **Note**: This plugin is considered insecure/deprecated.

Custom attributes passed as command parameters:

Name	Description
nrpe_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
nrpe_port	**Optional.** The NRPE port. Defaults to 5666.
nrpe_command	**Optional.** The command that should be executed.
nrpe_no_ssl	**Optional.** Whether to disable SSL or not. Defaults to `false`.
nrpe_timeout_unknown	**Optional.** Whether to set timeouts to unknown instead of critical state. Defaults to `false`.

Name	Description
nrpe_timeout	**Optional.** The timeout in seconds.
nrpe_arguments	**Optional.** Arguments that should be passed to the command. Multiple arguments must be defined as array.
nrpe_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
nrpe_ipv6	**Optional.** Use IPv6 connection. Defaults to false.
nrpe_version_2	**Optional.** Use this if you want to connect using NRPE v2 protocol. Defaults to false.

10.3.30 nscp

The check_nt plugin collects data from the NSClient++ service.

Custom attributes passed as command parameters:

Name	Description
nscp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
nscp_port	**Optional.** The NSClient++ port. Defaults to 12489.
nscp_password	**Optional.** The NSClient++ password.
nscp_variable	**Required.** The variable that should be checked.
nscp_params	**Optional.** Parameters for the query. Multiple parameters must be defined as array.
nscp_warn	**Optional.** The warning threshold.
nscp_crit	**Optional.** The critical threshold.
nscp_timeout	**Optional.** The query timeout in seconds.

Name	Description
nscp_showall	**Optional.** Use with SERVICESTATE to see working services or PROCSTATE for running processes. Defaults to false.

10.3.31 ntp_time

The check_ntp_time plugin checks the clock offset between the local host and a remote NTP server.

Note: If you want to monitor an NTP server, please use `ntp_peer`.

Custom attributes passed as command parameters:

Name	Description
ntp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ntp_port	**Optional.** Port number (default: 123).
ntp_quiet	**Optional.** Returns UNKNOWN instead of CRITICAL if offset cannot be found.
ntp_warning	**Optional.** Offset to result in warning status (seconds).

Name	Description
ntp_critical	**Optional.** Offset to result in critical status (seconds).
ntp_timeoffset	**Optional.** Expected offset of the ntp server relative to local server (seconds).
ntp_timeout	**Optional.** Seconds before connection times out (default: 10).
ntp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
ntp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.32 ntp_peer

The check_ntp_peer plugin checks the health of an NTP server. It supports checking the offset with the sync peer, the jitter and stratum. This plugin will not check the clock offset between the local host and NTP server; please use `ntp_time` for that purpose.

Custom attributes passed as command parameters:

Name	Description
ntp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ntp_port	**Optional.** The port to use. Default to 123.
ntp_quiet	**Optional.** Returns UNKNOWN instead of CRITICAL or WARNING if server isn't synchronized.
ntp_warning	**Optional.** Offset to result in warning status (seconds).
ntp_critical	**Optional.** Offset to result in critical status (seconds).
ntp_wstratum	**Optional.** Warning threshold for stratum of server's synchronization peer.
ntp_cstratum	**Optional.** Critical threshold for stratum of server's synchronization peer.

Name	Description
ntp_wjitter	**Optional.** Warning threshold for jitter.
ntp_cjitter	**Optional.** Critical threshold for jitter.
ntp_wsource	**Optional.** Warning threshold for number of usable time sources.
ntp_csource	**Optional.** Critical threshold for number of usable time sources.
ntp_timeout	**Optional.** Seconds before connection times out (default: 10).
ntp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
ntp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.33 pgsql

The check_pgsql plugin tests a PostgreSQL DBMS to determine whether it is active and accepting queries. If a query is specified using the `pgsql_query` attribute, it will be executed after connecting to the server. The result from the query has to be numeric in order to compare it against the query thresholds if set.

Custom attributes passed as command parameters:

Name	Description
pgsql_hostname	**Optional.** Host name, IP Address, or unix socket (must be an absolute path).
pgsql_port	**Optional.** Port number (default: 5432).
pgsql_database	**Optional.** Database to check (default: template1).
pgsql_username	**Optional.** Login name of user.
pgsql_password	**Optional.** Password (BIG SECURITY ISSUE).
pgsql_options	**Optional.** Connection parameters (keyword = value), see below.
pgsql_warning	**Optional.** Response time to result in warning status (seconds).
pgsql_critical	**Optional.** Response time to result in critical status (seconds).
pgsql_timeout	**Optional.** Seconds before connection times out (default: 10).
pgsql_query	**Optional.** SQL query to run. Only first column in first row will be read.
pgsql_query_warning	**Optional.** SQL query value to result in warning status (double).
pgsql_query_critical	**Optional.** SQL query value to result in critical status (double).

10.3.34 ping

The check_ping plugin uses the ping command to probe the specified host for packet loss (percentage) and round trip average (milliseconds).

This command uses the host's `address` attribute if available and falls back to using the `address6` attribute if the `address` attribute is not set.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 100.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 5.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 200.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 15.
ping_packets	**Optional.** The number of packets to send. Defaults to 5.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.3.35 ping4

The check_ping plugin uses the ping command to probe the specified host for packet loss (percentage) and round trip average (milliseconds).

This command uses the host's `address` attribute if not explicitly specified using the `ping_address` attribute.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The host's IPv4 address. Defaults to "*address*".
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 100.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 5.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 200.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 15.
ping_packets	**Optional.** The number of packets to send. Defaults to 5.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.3.36 ping6

The check_ping plugin uses the ping command to probe the specified host for packet loss (percentage) and round trip average (milliseconds).

This command uses the host's `address6` attribute if not explicitly specified using the `ping_address` attribute.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The host's IPv6 address. Defaults to "*address*6".
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 100.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 5.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 200.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 15.
ping_packets	**Optional.** The number of packets to send. Defaults to 5.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.3.37 pop

The check_pop plugin tests POP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
pop_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
pop_port	**Optional.** The port that should be checked. Defaults to 110.
pop_escape	**Optional.** Can use \n, \r, \t or \ in send or quit string. Must come before send or quit option. Default: nothing added to send, \r\n added to end of quit.
pop_send	**Optional.** String to send to the server.
pop_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
pop_all	**Optional.** All expect strings need to occur in server response. Default is any.

Name	Description
pop_quit	**Optional.** String to send server to initiate a clean close of the connection.
pop_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit (default: crit).
pop_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit (default: warn).
pop_jail	**Optional.** Hide output from TCP socket.
pop_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
pop_delay	**Optional.** Seconds to wait between sending string and polling for response.
pop_certificate_age	**Optional.** Minimum number of days a certificate has to be valid.
pop_ssl	**Optional.** Use SSL for the connection.

Name	Description
pop_warning	**Optional.** Response time to result in warning status (seconds).
pop_critical	**Optional.** Response time to result in critical status (seconds).
pop_timeout	**Optional.** Seconds before connection times out (default: 10).
pop_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
pop_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.38 procs

The check_procs plugin checks all processes and generates WARNING or CRITICAL states if the specified metric is outside the required threshold ranges. The metric defaults to number of processes. Search filters can be applied to limit the processes to check.

Custom attributes passed as command parameters:

Name	Description
procs_warning	**Optional.** The process count warning threshold. Defaults to 250.

Name	Description
procs_critical	**Optional.** The process count critical threshold. Defaults to 400.
procs_metric	**Optional.** Check thresholds against metric.
procs_timeout	**Optional.** Seconds before plugin times out.
procs_traditional	**Optional.** Filter own process the traditional way by PID instead of /proc/pid/exe. Defaults to false.
procs_state	**Optional.** Only scan for processes that have one or more of the status flags you specify.
procs_ppid	**Optional.** Only scan for children of the parent process ID indicated.
procs_vsz	**Optional.** Only scan for processes with VSZ higher than indicated.
procs_rss	**Optional.** Only scan for processes with RSS higher than indicated.

Name	Description
procs_pcpu	**Optional.** Only scan for processes with PCPU higher than indicated.
procs_user	**Optional.** Only scan for processes with user name or ID indicated.
procs_argument	**Optional.** Only scan for processes with args that contain STRING.
procs_argument_regex	**Optional.** Only scan for processes with args that contain the regex STRING.
procs_command	**Optional.** Only scan for exact matches of COMMAND (without path).
procs_nokthreads	**Optional.** Only scan for non kernel threads. Defaults to false.

10.3.39 radius

The check_radius plugin checks a RADIUS server to see if it is accepting connections. The server to test must be specified in the invocation, as well as a user name and password. A configuration file may also be present. The format of the configuration file is described in the radiusclient library sources. The password option presents a substantial security issue because the password can possibly be determined by careful watching of the command line in a process listing. This risk is exacerbated because the plugin will typically be executed

at regular predictable intervals. Please be sure that the password used does not allow access to sensitive system resources.

Custom attributes passed as command parameters:

Name	Description
radius_address	**Optional.** The radius server's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
radius_config_file	**Required.** The radius configuration file.
radius_username	**Required.** The radius username to test.
radius_password	**Required.** The radius password to test.
radius_port	**Optional.** The radius port number (default 1645).
radius_nas_id	**Optional.** The NAS identifier.
radius_nas_address	**Optional.** The NAS IP address.
radius_expect	**Optional.** The response string to expect from the server.
radius_retries	**Optional.** The number of times to retry a failed connection.

Name	Description
radius_timeout	**Optional.** The number of seconds before connection times out (default: 10).

10.3.40 rpc

The check_rpc plugin tests if a service is registered and running using `rpcinfo -H host -C rpc_command`.

Custom attributes passed as command parameters:

Name	Description
rpc_address	**Optional.** The rpc host address. Defaults to "*address*" if the host `address` attribute is set,"*address*6" otherwise.
rpc_command	**Required.** The programm name (or number).

Name	Description
rpc_port	**Optional.** The port that should be checked.
rpc_version	**Optional.** The version you want to check for (one or more).
rpc_udp	**Optional.** Use UDP test. Defaults to false.
rpc_tcp	**Optional.** Use TCP test. Defaults to false.

Name	Description
rpc_verbose	**Optional.** Show verbose output. Defaults to false.

10.3.41 simap

The check_simap plugin tests SIMAP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
simap_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
simap_port	**Optional.** The port that should be checked. Defaults to 993.

Name	Description
simap_escape	**Optional.** Can use \n, \r, \t or \ in send or quit string. Must come before send or quit option. Default: nothing added to send, \r\n added to end of quit.
simap_send	**Optional.** String to send to the server.
simap_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
simap_all	**Optional.** All expect strings need to occur in server response. Default is any.
simap_quit	**Optional.** String to send server to initiate a clean close of the connection.
simap_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit (default: crit).

Name	Description
simap_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit (default: warn).
simap_jail	**Optional.** Hide output from TCP socket.
simap_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
simap_delay	**Optional.** Seconds to wait between sending string and polling for response.
simap_certificate_age	**Optional.** Minimum number of days a certificate has to be valid.
simap_ssl	**Optional.** Use SSL for the connection.
simap_warning	**Optional.** Response time to result in warning status (seconds).
simap_critical	**Optional.** Response time to result in critical status (seconds).

Name	Description
simap_timeout	**Optional.** Seconds before connection times out (default: 10).
simap_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
simap_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.42 smart

The check_ide_smart plugin checks a local hard drive with the (Linux specific) SMART interface. Requires installation of `smartctl`.

Custom attributes passed as command parameters:

Name	Description
smart_device	**Required.** The name of a local hard drive to monitor.

10.3.43 smtp

The check_smtp plugin will attempt to open an SMTP connection with the host.

Custom attributes passed as command parameters:

Name	Description
smtp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
smtp_port	**Optional.** The port that should be checked. Defaults to 25.
smtp_mail_from	**Optional.** Test a MAIL FROM command with the given email address.
smtp_expect	**Optional.** String to expect in first line of server response (default: '220').
smtp_command	**Optional.** SMTP command (may be used repeatedly).
smtp_response	**Optional.** Expected response to command (may be used repeatedly).
smtp_helo_fqdn	**Optional.** FQDN used for HELO
smtp_certificate_age	**Optional.** Minimum number of days a certificate has to be valid.
smtp_starttls	**Optional.** Use STARTTLS for the connection.

Name	Description
smtp_authtype	**Optional.** SMTP AUTH type to check (default none, only LOGIN supported).
smtp_authuser	**Optional.** SMTP AUTH username.
smtp_authpass	**Optional.** SMTP AUTH password.
smtp_ignore_quit	**Optional.** Ignore failure when sending QUIT command to server.
smtp_warning	**Optional.** Response time to result in warning status (seconds).
smtp_critical	**Optional.** Response time to result in critical status (seconds).
smtp_timeout	**Optional.** Seconds before connection times out (default: 10).
smtp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
smtp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.44 snmp

The check_snmp plugin checks the status of remote machines and obtains system information via SNMP.

Note: This plugin uses the `snmpget` command included with the NET-SNMP package.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_oid	**Required.** The SNMP OID.
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port. Defaults to "161".
snmp_retries	**Optional.** Number of retries to be used in the SNMP requests.
snmp_warn	**Optional.** The warning threshold.
snmp_crit	**Optional.** The critical threshold.

Name	Description
snmp_string	**Optional.** Return OK state if the string matches exactly with the output value
snmp_ereg	**Optional.** Return OK state if extended regular expression REGEX matches with the output value
snmp_eregi	**Optional.** Return OK state if case-insensitive extended REGEX matches with the output value
snmp_label	**Optional.** Prefix label for output value
snmp_invert_search	**Optional.** Invert search result and return CRITICAL state if found
snmp_units	**Optional.** Units label(s) for output value (e.g., 'sec.').
snmp_version	**Optional.** Version to use. E.g. 1, 2, 2c or 3.

Name	Description
snmp_miblist	**Optional.** MIB's to use, comma separated. Defaults to "ALL".
snmp_rate_multiplier	**Optional.** Converts rate per second. For example, set to 60 to convert to per minute.
snmp_rate	**Optional.** Boolean. Enable rate calculation.
snmp_getnext	**Optional.** Boolean. Use SNMP GETNEXT. Defaults to false.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 10 seconds.
snmp_offset	**Optional.** Add/subtract the specified OFFSET to numeric sensor data.
snmp_output_delimiter	**Optional.** Separates output on multiple OID requests.

Name	Description
snmp_perf_oids	**Optional.** Label performance data with OIDs instead of –label's.

10.3.45 snmpv3

Check command object for the check_snmp plugin, using SNMPv3 authentication and encryption options.

Custom attributes passed as command parameters:

Name	Description
snmpv3_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmpv3_getnext	**Optional.** Use SNMP GETNEXT instead of SNMP GET.
snmpv3_seclevel	**Optional.** The security level. Defaults to authPriv.
snmpv3_auth_alg	**Optional.** The authentication algorithm. Defaults to SHA.
snmpv3_user	**Required.** The username to log in with.

Name	Description
snmpv3_auth_key	**Required,** The authentication key. Required if `snmpv3_seclevel` is set to `authPriv` otherwise optional.
snmpv3_priv_key	**Required.** The encryption key.
snmpv3_oid	**Required.** The SNMP OID.
snmpv3_priv_alg	**Optional.** The encryption algorithm. Defaults to AES.
snmpv3_warn	**Optional.** The warning threshold.
snmpv3_crit	**Optional.** The critical threshold.
snmpv3_string	**Optional.** Return OK state (for that OID) if STRING is an exact match.
snmpv3_ereg	**Optional.** Return OK state (for that OID) if extended regular expression REGEX matches.

Name	Description
snmpv3_eregi	**Optional.** Return OK state (for that OID) if case-insensitive extended REGEX matches.
snmpv3_invert_search	**Optional.** Invert search result and return CRITICAL if found
snmpv3_label	**Optional.** Prefix label for output value.
snmpv3_units	**Optional.** Units label(s) for output value (e.g., 'sec.').
snmpv3_rate_multiplier	**Optional.** Converts rate per second. For example, set to 60 to convert to per minute.
snmpv3_rate	**Optional.** Boolean. Enable rate calculation.
snmpv3_timeout	**Optional.** The command timeout in seconds. Defaults to 10 seconds.

10.3.46 snmp-uptime

Check command object for the check_snmp plugin, using the uptime OID by default.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_oid	**Optional.** The SNMP OID. Defaults to "1.3.6.1.2.1.1.3.0".
snmp_community	**Optional.** The SNMP community. Defaults to "public".

10.3.47 spop

The check_spop plugin tests SPOP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
spop_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
spop_port	**Optional.** The port that should be checked. Defaults to 995.

Name	Description
spop_escape	**Optional.** Can use \n, \r, \t or \ in send or quit string. Must come before send or quit option. Default: nothing added to send, \r\n added to end of quit.
spop_send	**Optional.** String to send to the server.
spop_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
spop_all	**Optional.** All expect strings need to occur in server response. Default is any.
spop_quit	**Optional.** String to send server to initiate a clean close of the connection.
spop_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit (default: crit).

Name	Description
spop_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit (default: warn).
spop_jail	**Optional.** Hide output from TCP socket.
spop_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
spop_delay	**Optional.** Seconds to wait between sending string and polling for response.
spop_certificate_age	**Optional.** Minimum number of days a certificate has to be valid.
spop_ssl	**Optional.** Use SSL for the connection.
spop_warning	**Optional.** Response time to result in warning status (seconds).
spop_critical	**Optional.** Response time to result in critical status (seconds).

Name	Description
spop_timeout	**Optional.** Seconds before connection times out (default: 10).
spop_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
spop_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.48 ssh

The check_ssh plugin connects to an SSH server at a specified host and port.

Custom attributes passed as command parameters:

Name	Description
ssh_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ssh_port	**Optional.** The port that should be checked. Defaults to 22.
ssh_timeout	**Optional.** Seconds before connection times out. Defaults to 10.

Name	Description
ssh_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
ssh_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.49 ssl

Check command object for the check_tcp plugin, using ssl-related options.

Custom attributes passed as command parameters:

Name	Description
ssl_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ssl_port	**Optional.** The port that should be checked. Defaults to 443.
ssl_timeout	**Optional.** Timeout in seconds for the connect and handshake. The plugin default is 10 seconds.

Name	Description
ssl_cert_valid_days_warn	**Optional.** Warning threshold for days before the certificate will expire. When used, the default for ssl_cert_valid_days_critical is 0.
ssl_cert_valid_days_critical	**Optional.** Critical threshold for days before the certificate will expire. When used, ssl_cert_valid_days_warn must also be set.
ssl_sni	**Optional.** The `server_name` that is send to select the SSL certificate to check. Important if SNI is used.

10.3.50 ssmtp

The check_ssmtp plugin tests SSMTP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
ssmtp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ssmtp_port	**Optional.** The port that should be checked. Defaults to 465.
ssmtp_escape	**Optional.** Can use \n, \r, \t or \ in send or quit string. Must come before send or quit option. Default: nothing added to send, \r\n added to end of quit.
ssmtp_send	**Optional.** String to send to the server.
ssmtp_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
ssmtp_all	**Optional.** All expect strings need to occur in server response. Default is any.

Name	Description
ssmtp_quit	**Optional.** String to send server to initiate a clean close of the connection.
ssmtp_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit (default: crit).
ssmtp_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit (default: warn).
ssmtp_jail	**Optional.** Hide output from TCP socket.
ssmtp_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
ssmtp_delay	**Optional.** Seconds to wait between sending string and polling for response.
ssmtp_certificate_age	**Optional.** Minimum number of days a certificate has to be valid.
ssmtp_ssl	**Optional.** Use SSL for the connection.

Name	Description
ssmtp_warning	**Optional.** Response time to result in warning status (seconds).
ssmtp_critical	**Optional.** Response time to result in critical status (seconds).
ssmtp_timeout	**Optional.** Seconds before connection times out (default: 10).
ssmtp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
ssmtp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.51 swap

The check_swap plugin checks the swap space on a local machine.

Custom attributes passed as command parameters:

Name	Description
swap_wfree	**Optional.** The free swap space warning threshold in % (enable `swap_integer` for number values). Defaults to `50%`.

Name	Description
swap_cfree	**Optional.** The free swap space critical threshold in % (enable `swap_integer` for number values). Defaults to `25%`.
swap_integer	**Optional.** Specifies whether the thresholds are passed as number or percent value. Defaults to false (percent values).
swap_allswaps	**Optional.** Conduct comparisons for all swap partitions, one by one. Defaults to false.
swap_noswap	**Optional.** Resulting state when there is no swap regardless of thresholds. Possible values are “ok”, “warning”, “critical”, “unknown”. Defaults to “critical”.

10.3.52 tcp

The check_tcp plugin tests TCP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
tcp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
tcp_port	**Required.** The port that should be checked.
tcp_expect	**Optional.** String to expect in server response. Multiple strings must be defined as array.
tcp_all	**Optional.** All expect strings need to occur in server response. Defaults to false.
tcp_escape_send	**Optional.** Enable usage of \n, \r, \t or \\ in send string.
tcp_send	**Optional.** String to send to the server.
tcp_escape_quit	**Optional.** Enable usage of \n, \r, \t or \\ in quit string.

Name	Description
tcp_quit	**Optional.** String to send server to initiate a clean close of the connection.
tcp_refuse	**Optional.** Accept TCP refusals with states ok, warn, crit. Defaults to crit.
tcp_mismatch	**Optional.** Accept expected string mismatches with states ok, warn, crit. Defaults to warn.
tcp_jail	**Optional.** Hide output from TCP socket.
tcp_maxbytes	**Optional.** Close connection once more than this number of bytes are received.
tcp_delay	**Optional.** Seconds to wait between sending string and polling for response.

Name	Description
tcp_certificate	**Optional.** Minimum number of days a certificate has to be valid. 1st value is number of days for warning, 2nd is critical (if not specified: 0) – separated by comma.
tcp_ssl	**Optional.** Use SSL for the connection. Defaults to false.
tcp_wtime	**Optional.** Response time to result in warning status (seconds).
tcp_ctime	**Optional.** Response time to result in critical status (seconds).
tcp_timeout	**Optional.** Seconds before connection times out. Defaults to 10.
tcp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
tcp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.53 udp

The check_udp plugin tests UDP connections with the specified host (or unix socket).

Custom attributes passed as command parameters:

Name	Description
udp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
udp_port	**Required.** The port that should be checked.
udp_send	**Required.** The payload to send in the UDP datagram.
udp_expect	**Required.** The payload to expect in the response datagram.
udp_quit	**Optional.** The payload to send to 'close' the session.
udp_ipv4	**Optional.** Use IPv4 connection. Defaults to false.
udp_ipv6	**Optional.** Use IPv6 connection. Defaults to false.

10.3.54 ups

The check_ups plugin tests the UPS service on the specified host. Network UPS Tools must be running for this plugin to work.

Custom attributes passed as command parameters:

Name	Description
ups_address	**Required.** The address of the host running upsd. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ups_name	**Required.** The UPS name. Defaults to `ups`.
ups_port	**Optional.** The port to which to connect. Defaults to 3493.
ups_variable	**Optional.** The variable to monitor. Must be one of LINE, TEMP, BATTPCT or LOADPCT. If this is not set, the check only relies on the value of `ups.status`.
ups_warning	**Optional.** The warning threshold for the selected variable.

Name	Description
ups_critical	**Optional.** The critical threshold for the selected variable.
ups_celsius	**Optional.** Display the temperature in degrees Celsius instead of Fahrenheit. Defaults to `false`.
ups_timeout	**Optional.** The number of seconds before the connection times out. Defaults to 10.

10.3.55 users

The check_users plugin checks the number of users currently logged in on the local system and generates an error if the number exceeds the thresholds specified.

Custom attributes passed as command parameters:

Name	Description
users_wgreater	**Optional.** The user count warning threshold. Defaults to 20.
users_cgreater	**Optional.** The user count critical threshold. Defaults to 50.

10.4 Windows Plugins for Icinga 2

To allow a basic monitoring of Windows clients Icinga 2 comes with a set of Windows only plugins. While trying to mirror the functionalities of their linux cousins from the monitoring-plugins package, the differences between Windows and Linux are too big to be able use the same CheckCommands for both systems.

A check-commands-windows.conf comes with Icinga 2, it assumes that the Windows Plugins are installed in the PluginDir set in your constants.conf. To enable them the following include directive is needed in you icinga2.conf:

```
include <windows-plugins>
```

One of the differences between the Windows plugins and their linux counterparts is that they consistently do not require thresholds to run, functioning like dummies without.

10.4.1 Threshold syntax

So not specified differently the thresholds for the plugins all follow the same pattern

Threshold	Meaning
"29"	The threshold is 29.
"!29"	The threshold is 29, but the negative of the result is returned.
"[10-40]"	The threshold is a range from (including) 10 to 40, a value inside means the threshold has been exceeded.
"[10-40]"	Same as above, but the result is inverted.

10.4.2 disk-windows

Check command object for the `check_disk.exe` plugin. Aggregates the disk space of all volumes and mount points it can find, or the ones defined in `disk_win_path`. Ignores removable storage like flash drives and discs (CD, DVD etc.). The data collection is instant and free disk space (default, see `disk_win_show_used`) is used for threshold computation.

> **Note**
>
> Percentage based thresholds can be used by adding a '%' to the threshold value.

Custom attributes:

Name	Description
disk_win_warn	**Optional**. The warning threshold.
disk_win_crit	**Optional**. The critical threshold.
disk_win_path	**Optional**. Check only these paths, default checks all.
disk_win_unit	**Optional**. Use this unit to display disk space, thresholds are interpreted in this unit. Defaults to "mb", possible values are: b, kb, mb, gb and tb.
disk_win_exclude	**Optional**. Exclude these drives from check.

Name	Description
disk_win_show_used	**Optional**. Use used instead of free space.

10.4.3 load-windows

Check command object for the `check_load.exe` plugin. This plugin collects the inverse of the performance counter `\Processor(_Total)\% Idle Time` two times, with a wait time of one second between the collection. To change this wait time use `perfmon-windows`.

Custom attributes:

Name	Description
load_win_warn	**Optional**. The warning threshold.
load_win_crit	**Optional**. The critical threshold.

10.4.4 memory-windows

Check command object for the `check_memory.exe` plugin. The memory collection is instant and free memory is used for threshold computation.

> **Note**
>
> Percentage based thresholds can be used by adding a '%' to the threshold value. Keep in mind that memory_win_unit is applied before the value is calculated.

Custom attributes:

Name	Description
memory_win_warn	**Optional**. The warning threshold.
memory_win_crit	**Optional**. The critical threshold.

Name	Description
memory_win_unit	**Optional**. The unit to display the received value in, thresholds are interpreted in this unit. Defaults to "mb" (megabyte), possible values are: b, kb, mb, gb and tb.

10.4.5 network-windows

Check command object for the `check_network.exe` plugin. Collects the total Bytes inbound and outbound for all interfaces in one second, to itemise interfaces or use a different collection interval use `perfmon-windows`.

Custom attributes:

Name	Description
network_win_warn	**Optional**. The warning threshold.
network_win_crit	**Optional**. The critical threshold.
network_no_isatap	**Optional**. Do not print ISATAP interfaces.

10.4.6 perfmon-windows

Check command object for the `check_perfmon.exe` plugin. This plugins allows to collect data from a Performance Counter. After the first data collection a second one is done after `perfmon_win_wait` milliseconds. When you know `perfmon_win_counter` only requires one set of data to provide valid data you can set `perfmon_win_wait` to `0`.

To receive a list of possible Performance Counter Objects run `check_perfmon.exe --print-objects` and to view an objects instances and counters run `check_perfmon.exe --print-object-info -P "name of object"`

Custom attributes:

Name	Description
perfmon_win_warn	**Optional**. The warning threshold.
perfmon_win_crit	**Optional**. The critical threshold.
perfmon_win_counter	**Required**. The Performance Counter to use. Ex. `\Processor(_Total)\% Idle Time`.
perfmon_win_wait	**Optional**. Time in milliseconds to wait between data collection (default: 1000).
perfmon_win_type	**Optional**. Format in which to expect performance values. Possible are: long, int64 and double (default).
perfmon_win_syntax	**Optional**. Use this in the performance output instead of `perfmon\_win\_counter`. Exists for graphics compatibility reasons.

10.4.7 ping-windows

Check command object for the `check_ping.exe` plugin. ping-windows should automatically detect whether `ping_win_address` is an IPv4 or IPv6 address. If not, use ping4-windows and ping6-windows. Also note that check_ping.exe waits at least `ping_win_timeout` milliseconds between the pings.

Custom attributes:

Name	Description
ping_win_warn	**Optional**. The warning threshold. RTA and package loss separated by comma.
ping_win_crit	**Optional**. The critical threshold. RTA and package loss separated by comma.
ping_win_address	**Required**. An IPv4 or IPv6 address.
ping_win_packets	**Optional**. Number of packages to send. Default: 5.
ping_win_timeout	**Optional**. The timeout in milliseconds. Default: 1000

10.4.8 procs-windows

Check command object for `check_procs.exe` plugin. When using `procs_win_user` this plugins needs administrative privileges to access the processes of other users, to just enumerate them no additional privileges are required.

Custom attributes:

Name	Description
procs_win_warn	**Optional**. The warning threshold.
procs_win_crit	**Optional**. The critical threshold.
procs_win_user	**Optional**. Count this users processes.

10.4.9 service-windows

Check command object for `check_service.exe` plugin. This checks thresholds work different since the binary decision whether a service is running or not does not allow for three states. As a default `check_service.exe` will return CRITICAL when `service_win_service` is not running, the `service_win_warn` flag changes this to WARNING.

Custom attributes:

Name	Description
service_win_warn	**Optional**. Warn when service is not running.
service_win_description	**Optional**. If this is set, `service\_win\_service` looks at the service description.
service_win_service	**Required**. Name of the service to check.

10.4.10 swap-windows

Check command object for `check_swap.exe` plugin. The data collection is instant.

Custom attributes:

Name	Description
swap_win_warn	**Optional**. The warning threshold.
swap_win_crit	**Optional**. The critical threshold.
swap_win_unit	**Optional**. The unit to display the received value in, thresholds are interpreted in this unit. Defaults to "mb" (megabyte).

10.4.11 update-windows

Check command object for `check_update.exe` plugin. Querying Microsoft for Windows updates can take multiple seconds to minutes. An update is treated as important when it has the WSUS flag for SecurityUpdates or CriticalUpdates.

> **Note**
>
> The Network Services Account which runs Icinga 2 by default does not have the required permissions to run this check.

Custom attributes:

Name	Description
update_win_warn	**Optional**. If set, returns warning when important updates are available.

Name	Description
update_win_crit	**Optional**. If set, return critical when important updates that require a reboot are available.
update_win_reboot	**Optional**. Set to treat 'may need update' as 'definitely needs update'. Please Note that this is true for almost every update and is therefore not recommended.

In contrast to most other plugins, the values of check_update's custom attributes do not set thresholds, but just enable/disable the behavior described in the table above.
It can be enabled/disabled for example by setting them to "true" or "false", "1" or "0" would also work.
Thresholds will always be "1".

> **Note**
>
> If they are enabled, performance data will be shown in the web interface.
> If run without the optional parameters, the plugin will output critical if any important updates are available.

10.4.12 uptime-windows

Check command object for `check_uptime.exe` plugin. Uses GetTickCount64 to get the uptime, so boot time is not included.

Custom attributes:

Name	Description
uptime_win_warn	**Optional**. The warning threshold.
uptime_win_crit	**Optional**. The critical threshold.
uptime_win_unit	**Optional**. The unit to display the received value in, thresholds are interpreted in this unit. Defaults to "s"(seconds), possible values are ms (milliseconds), s, m (minutes), h (hours).

10.4.13 users-windows

Check command object for `check_users.exe` plugin.

Custom attributes:

Name	Description
users_win_warn	**Optional**. The warning threshold.
users_win_crit	**Optional**. The critical threshold.

10.5 Plugin Check Commands for NSClient++

There are two methods available for querying NSClient++:

- Query the HTTP API locally or remotely (requires a running NSClient++ service)
- Run a local CLI check (does not require NSClient++ as a service)

Both methods have their advantages and disadvantages. One thing to note:

If you rely on performance counter delta calculations such as CPU utilization, please use the HTTP API instead of the CLI sample call.

10.5.1 nscp_api

`check_nscp_api` is part of the Icinga 2 plugins. This plugin is available for both, Windows and Linux/Unix.

Verify that the ITL CheckCommand is included in the icinga2.conf configuration file:

```
vim /etc/icinga2/icinga2.conf
```

```
include <plugins>
```

`check_nscp_api` runs queries against the NSClient++ API. Therefore NSClient++ needs to have the `webserver` module enabled, configured and loaded.

You can install the webserver using the following CLI commands:

```
./nscp.exe web install
./nscp.exe web password - -set icinga
```

Now you can define specific queries and integrate them into Icinga 2.

The check plugin `check_nscp_api` can be integrated with the `nscp_api` CheckCommand object:

Custom attributes:

Name	Description
nscp_api_host	**Required**. NSCP API host address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
nscp_api_port	**Optional**. NSCP API port. Defaults to `8443`.
nscp_api_password	**Required**. NSCP API password. Please check the NSCP documentation for setup details.

Name	Description
nscp_api_query	**Required**. NSCP API query endpoint. Refer to the NSCP documentation for possible values.
nscp_api_arguments	**Optional**. NSCP API arguments dictionary either as single strings or key-value pairs using `=`. Refer to the NSCP documentation.

`nscp_api_arguments` can be used to pass required thresholds to the executed check. The example below checks the CPU utilization and specifies warning and critical thresholds.

```
check_nscp_api --host 10.0.10.148 --password icinga --query check_cpu --arguments show-all warn
check_cpu CRITICAL: critical(5m: 48%, 1m: 36%), 5s: 0% | 'total 5m'=48%;40;30 'total 1m'=36%;40;
```

10.5.2 nscp-local

Icinga 2 can use the `nscp client` command to run arbitrary NSClient++ checks locally on the client.

You can enable these check commands by adding the following the include directive in your icinga2.conf configuration file:

```
include <nscp>
```

You can also optionally specify an alternative installation directory for NSClient++ by adding the NscpPath constant in your constants.conf configuration file:

```
const NscpPath = "C:\\Program Files (x86)\\NSClient++"
```

By default Icinga 2 uses the Microsoft Installer API to determine where NSClient++ is installed. It should not be necessary to manually set this constant.

Note that it is not necessary to run NSClient++ as a Windows service for these commands to work.

The check command object for NSClient++ is available as `nscp-local`.

Custom attributes passed as command parameters:

Name	Description
nscp_log_level	**Optional.** The log level. Defaults to "critical".
nscp_load_all	**Optional.** Whether to load all modules. Defaults to false.
nscp_modules	**Optional.** An array of NSClient++ modules to load. Defaults to `[ "CheckSystem" ]`.
nscp_boot	**Optional.** Whether to use the –boot option. Defaults to true.
nscp_query	**Required.** The NSClient++ query. Try `nscp client -q x` for a list.
nscp_arguments	**Optional.** An array of query arguments.
nscp_showall	**Optional.** Shows more details in plugin output, default to false.

10.5.3 nscp-local-cpu

Check command object for the `check_cpu` NSClient++ plugin.

Name	Description
nscp_cpu_time	**Optional.** Calculate average usage for the given time intervals. Value has to be an array, default to [“1m”, “5m”, “15m”].
nscp_cpu_warning	**Optional.** Threshold for WARNING state in percent, default to 80.
nscp_cpu_critical	**Optional.** Threshold for CRITICAL state in percent, default to 90.
nscp_cpu_arguments	**Optional.** Additional arguments.
nscp_cpu_showall	**Optional.** Shows more details in plugin output, default to false.

10.5.4 nscp-local-memory

Check command object for the `check_memory` NSClient++ plugin.

Name	Description
nscp_memory_committed	**Optional.** Check for committed memory, default to false.
nscp_memory_physical	**Optional.** Check for physical memory, default to true.

Name	Description
nscp_memory_free	**Optional.** Switch between checking free (true) or used memory (false), default to false.
nscp_memory_warning	**Optional.** Threshold for WARNING state in percent or absolute (use MB, GB, …), default to 80 (free=false) or 20 (free=true).
nscp_memory_critical	**Optional.** Threshold for CRITICAL state in percent or absolute (use MB, GB, …), default to 90 (free=false) or 10 (free=true).
nscp_memory_arguments	**Optional.** Additional arguments.
nscp_memory_showall	**Optional.** Shows more details in plugin output, default to false.

10.5.5 nscp-local-os-version

Check command object for the `check_os_version` NSClient++ plugin.

This command has the same custom attributes like the `nscp-local` check command.

10.5.6 nscp-local-pagefile

Check command object for the `check_pagefile` NSClient++ plugin.

This command has the same custom attributes like the `nscp-local` check command.

10.5.7 nscp-local-process

Check command object for the `check_process` NSClient++ plugin.

This command has the same custom attributes like the `nscp-local` check command.

10.5.8 nscp-local-service

Check command object for the `check_service` NSClient++ plugin.

Name	Description
nscp_service_name	**Required.** Name of service to check.
nscp_service_type	**Optional.** Type to check, default to state.
nscp_service_ok	**Optional.** State for return an OK, i.e. for type=state running, stopped, …
nscp_service_otype	**Optional.** Dedicate type for nscp_service_ok, default to nscp_service_state.
nscp_service_warning	**Optional.** State for return an WARNING.
nscp_service_wtype	**Optional.** Dedicate type for nscp_service_warning, default to nscp_service_state.
nscp_service_critical	**Optional.** State for return an CRITICAL.
nscp_service_ctype	**Optional.** Dedicate type for nscp_service_critical, default to nscp_service_state.

Name	Description
nscp_service_arguments	**Optional.** Additional arguments.
nscp_service_showall	**Optional.** Shows more details in plugin output, default to true.

10.5.9 nscp-local-uptime

Check command object for the `check_uptime` NSClient++ plugin.

This command has the same custom attributes like the `nscp-local` check command.

10.5.10 nscp-local-version

Check command object for the `check_version` NSClient++ plugin.

This command has the same custom attributes like the `nscp-local` check command. In addition to that the default value for `nscp_modules` is set to `[ "CheckHelpers" ]`.

10.5.11 nscp-local-disk

Check command object for the `check_drivesize` NSClient++ plugin.

Name	Description
nscp_disk_drive	**Optional.** Drive character, default to all drives.
nscp_disk_free	**Optional.** Switch between checking free space (free=true) or used space (free=false), default to false.

Name	Description
nscp_disk_warning	**Optional.** Threshold for WARNING in percent or absolute (use MB, GB, …), default to 80 (used) or 20 percent (free).
nscp_disk_critical	**Optional.** Threshold for CRITICAL in percent or absolute (use MB, GB, …), default to 90 (used) or 10 percent (free).
nscp_disk_arguments	**Optional.** Additional arguments.
nscp_disk_showall	**Optional.** Shows more details in plugin output, default to true.
nscp_modules	**Optional.** An array of NSClient++ modules to load. Defaults to `[ "CheckDisk" ]`.

10.5.12 nscp-local-counter

Check command object for the `check_pdh` NSClient++ plugin.

Name	Description
nscp_counter_name	**Required.** Performance counter name.
nscp_counter_warning	**Optional.** WARNING Threshold.

Name	Description
nscp_counter_critical	**Optional.** CRITICAL Threshold.
nscp_counter_arguments	**Optional.** Additional arguments.
nscp_counter_showall	**Optional.** Shows more details in plugin output, default to false.
nscp_counter_perfsyntax	**Optional.** Apply performance data label, e.g. `Total Processor Time` to avoid special character problems. Defaults to `nscp_counter_name`.

10.6 Plugin Check Commands for Manubulon SNMP

The `SNMP Manubulon Plugin Check Commands` provide configuration for plugin check commands provided by the SNMP Manubulon project.

Note: Some plugin parameters are only available in Debian packages or in a forked repository with patches applied.

The SNMP manubulon plugin check commands assume that the global constant named `ManubulonPluginDir` is set to the path where the Manubublon SNMP plugins are installed.

You can enable these plugin check commands by adding the following the include directive in your icinga2.conf configuration file:

```
include <manubulon>
```

10.6.1 Checks by Host Type

N/A : Not available for this type.

SNMP : Available for simple SNMP query.

?? : Untested.

Specific : Script name for platform specific checks.

Host type	Interface	storage	load/cpu	mem	process	env	specific
Linux	Yes	Yes	Yes	Yes	Yes	No	
Windows	Yes	Yes	Yes	Yes	Yes	No	check_snmp_win.p
Cisco router/switch	Yes	N/A	Yes	Yes	N/A	Yes	
HP router/switch	Yes	N/A	Yes	Yes	N/A	No	
Bluecoat proxy	Yes	SNMP	Yes	SNMP	No	Yes	
CheckPoint on SPLAT	Yes	Yes	Yes	Yes	Yes	No	check_snmp_cpfw.
CheckPoint on Nokia IP	Yes	Yes	Yes	No	??	No	check_snmp_vrrp.p
Boostedge	Yes	Yes	Yes	Yes	??	No	check_snmp_boost
AS400	Yes	Yes	Yes	Yes	No	No	
NetsecureOne Netbox	Yes	Yes	Yes	??	Yes	No	
Radware Linkproof	Yes	N/A	SNMP	SNMP	No	No	check_snmp_linkpr check_snmp_vrrp.p
IronPort	Yes	SNMP	SNMP	SNMP	No	Yes	
Cisco CSS	Yes	??	Yes	Yes	No	??	check_snmp_css.pl

10.6.2 snmp-env

Check command object for the check_snmp_env.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to "snmpuser".
snmp_password	**Required.** SNMP version 3 password. No value defined as default.
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.

Name	Description
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to “md5,des”.
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default.
snmp_env_type	**Optional.** Environment Type [cisco
snmp_env_fan	**Optional.** Minimum fan rpm value (only needed for ‘iron’ & ‘linux’)
snmp_env_celsius	**Optional.** Maximum temp in degrees celsius (only needed for ‘iron’ & ‘linux’)
snmp_perf	**Optional.** Enable perfdata values. Defaults to true.

Name	Description
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.

10.6.3 snmp-load

Check command object for the check_snmp_load.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.

Name	Description
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to “snmpuser”.
snmp_password	**Required.** SNMP version 3 password. No value defined as default.
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to “md5,des”.
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default.

Name	Description
snmp_warn	**Optional.** The warning threshold. Change the `snmp_load_type` var to "netsl" for using 3 values.
snmp_crit	**Optional.** The critical threshold. Change the `snmp_load_type` var to "netsl" for using 3 values.
snmp_load_type	**Optional.** Load type. Defaults to "stand". Check all available types in the snmp load documentation.
snmp_perf	**Optional.** Enable perfdata values. Defaults to true.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.

10.6.4 snmp-memory

Check command object for the check_snmp_mem.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to "snmpuser".
snmp_password	**Required.** SNMP version 3 password. No value defined as default.

Name	Description
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to "md5,des".
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default.
snmp_warn	**Optional.** The warning threshold.
snmp_crit	**Optional.** The critical threshold.
snmp_is_cisco	**Optional.** Change OIDs for Cisco switches. Defaults to false.

Name	Description
snmp_is_hp	**Optional.** Change OIDs for HP/Procurve switches. Defaults to false.
snmp_perf	**Optional.** Enable perfdata values. Defaults to true.
snmp_memcached	**Optional.** Include cached memory in used memory, Defaults to false.
snmp_membuffer	**Optional.** Exclude buffered memory in used memory, Defaults to false.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.

10.6.5 snmp-storage

Check command object for the check_snmp_storage.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to "snmpuser".
snmp_password	**Required.** SNMP version 3 password. No value defined as default.

Name	Description
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to "md5,des".
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default.
snmp_warn	**Optional.** The warning threshold.
snmp_crit	**Optional.** The critical threshold.
snmp_storage_name	**Optional.** Storage name. Default to regex "^/$$". More options available in the snmp storage documentation.

Name	Description
snmp_perf	**Optional.** Enable perfdata values. Defaults to true.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.
snmp_storage_olength	**Optional.** Max-size of the SNMP message, usefull in case of Too Long responses.

10.6.6 snmp-interface

Check command object for the check_snmp_int.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).

Name	Description
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to "snmpuser".
snmp_password	**Required.** SNMP version 3 password. No value defined as default.
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.

Name	Description
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to “md5,des”.
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default.
snmp_warn	**Optional.** The warning threshold.
snmp_crit	**Optional.** The critical threshold.
snmp_interface	**Optional.** Network interface name. Default to regex “eth0”.
snmp_interface_inverse	**Optional.** Inverse Interface check, down is ok. Defaults to false as it is missing.
snmp_interface_perf	**Optional.** Check the input/output bandwidth of the interface. Defaults to true.

Name	Description
snmp_interface_label	**Optional.** Add label before speed in output: in=, out=, errors-out=, etc.
snmp_interface_bits_bytes	**Optional.** Output performance data in bits/s or Bytes/s. **Depends** on snmp_interface_kbits set to true. Defaults to true.
snmp_interface_percent	**Optional.** Output performance data in % of max speed. Defaults to false.
snmp_interface_kbits	**Optional.** Make the warning and critical levels in KBits/s. Defaults to true.
snmp_interface_megabytes	**Optional.** Make the warning and critical levels in Mbps or MBps. **Depends** on snmp_interface_kbits set to true. Defaults to true.

Name	Description
snmp_interface_64bit	**Optional.** Use 64 bits counters instead of the standard counters when checking bandwidth & performance data for interface >= 1Gbps. Defaults to false.
snmp_interface_errors	**Optional.** Add error & discard to Perfparse output. Defaults to true.
snmp_interface_noregexp	**Optional.** Do not use regexp to match interface name in description OID. Defaults to false.
snmp_interface_delta	**Optional.** Delta time of perfcheck. Defaults to "300" (5 min).
snmp_interface_warncrit_percent	**Optional.** Make the warning and critical levels in % of reported interface speed. If set, **snmp_interface_megabytes** needs to be set to false. Defaults to false.

Name	Description
snmp_interface_ifname	**Optional.** Switch from IF-MIB::ifDescr to IF-MIB::ifName when looking up the interface's name.
snmp_interface_ifalias	**Optional.** Switch from IF-MIB::ifDescr to IF-MIB::ifAlias when looking up the interface's name.
snmp_interface_weathermap	**Optional.** Output data for "weathermap" lines in NagVis. **Depends** on `snmp_interface_perf` set to true. Defaults to `false`. **Note**: Available in `check_snmp_int.pl v2.1.0`.
snmp_perf	**Optional.** Enable perfdata values. Defaults to true.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.

10.6.7 snmp-process

Check command object for the check_snmp_process.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to "snmpuser".

Name	Description
snmp_password	**Required.** SNMP version 3 password. No value defined as default.
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to “md5,des”.
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default..
snmp_warn	**Optional.** The warning threshold.
snmp_crit	**Optional.** The critical threshold.
snmp_process_name	**Optional.** Name of the process (regexp). No trailing slash!. Defaults to “.*“.

Name	Description
snmp_perf	**Optional.** Enable perfdata values. Defaults to true.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.
snmp_process_use_params	**Optional.** Add process parameters to process name for regexp matching. Example: "named.*-t /var/named/chroot" will only select named process with this parameter. Defaults to false.
snmp_process_mem_usage	**Optional.** Define to check memory usage for the process. Defaults to false.
snmp_process_mem_threshold	**Optional.** Defines the warning and critical thresholds in Mb when snmp_process_mem_usage set to true. Example "512,1024". Defaults to "0,0".

Name	Description
snmp_process_cpu_usage	**Optional.** Define to check CPU usage for the process. Defaults to false.
snmp_process_cpu_threshold	**Optional.** Defines the warning and critical thresholds in % when snmp_process_cpu_usage set to true. If more than one CPU, value can be > 100% : 100%=1 CPU. Example "15,50". Defaults to "0,0".

10.6.8 snmp-service

Check command object for the check_snmp_win.pl plugin.

Custom attributes passed as command parameters:

Name	Description
snmp_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
snmp_nocrypt	**Optional.** Define SNMP encryption. If set to `false`, `snmp_v3` needs to be enabled. Defaults to `true` (no encryption).
snmp_community	**Optional.** The SNMP community. Defaults to "public".
snmp_port	**Optional.** The SNMP port connection.
snmp_v2	**Optional.** SNMP version to 2c. Defaults to false.
snmp_v3	**Optional.** SNMP version to 3. Defaults to false.
snmp_login	**Optional.** SNMP version 3 username. Defaults to "snmpuser".
snmp_password	**Required.** SNMP version 3 password. No value defined as default.
snmp_v3_use_privpass	**Optional.** Define to use SNMP version 3 priv password. Defaults to false.

Name	Description
snmp_v3_use_authprotocol	**Optional.** Define to use SNMP version 3 authentication protocol. Defaults to false.
snmp_authprotocol	**Optional.** SNMP version 3 authentication protocol. Defaults to “md5,des”.
snmp_privpass	**Required.** SNMP version 3 priv password. No value defined as default.
snmp_timeout	**Optional.** The command timeout in seconds. Defaults to 5 seconds.
snmp_service_name	**Optional.** Comma separated names of services (perl regular expressions can be used for every one). By default, it is not case sensitive. eg. ^dns$. Defaults to “.*“.

Name	Description
snmp_service_count	**Optional.** Compare matching services with a specified number instead of the number of names provided.
snmp_service_showall	**Optional.** Show all services in the output, instead of only the non-active ones. Defaults to false.
snmp_service_noregexp	**Optional.** Do not use regexp to match NAME in service description. Defaults to false.

10.7 Contributed Plugin Check Commands

The contributed Plugin Check Commands provides various additional command definitions contributed by community members.

These check commands assume that the global constant named `PluginContribDir` is set to the path where the user installs custom plugins and can be enabled by uncommenting the corresponding line in icinga2.conf:

```
vim /etc/icinga2/icinga2.conf
```

```
include <plugin-contrib>
```

This is enabled by default since Icinga 2 2.5.0.

10.7.1 Databases

This category contains plugins for various database servers.

10.7.1.1 db2_health

The check_db2_health plugin uses the `DBD::DB2` Perl library to monitor a DB2 database.

The Git repository is located on GitHub.

Custom attributes passed as command parameters:

Name	Description
db2_health_database	**Required.** The name of the database. (If it was catalogued locally, this parameter is the only you need. Otherwise you must specify database, hostname and port)
db2_health_username	**Optional.** The username for the database connection.
db2_health_password	**Optional.** The password for the database connection.
db2_health_port	**Optional.** The port where DB2 is listening.
db2_health_warning	**Optional.** The warning threshold depending on the mode.
db2_health_critical	**Optional.** The critical threshold depending on the mode.
db2_health_mode	**Required.** The mode uses predefined keywords for the different checks. For example “connection-time”, “database-usage” or “sql”.
db2_health_method	**Optional.** This tells the plugin how to connect to the database. The only method implemented yet is “dbi” which is the default. (It means, the plugin uses the perl module DBD::DB2).
db2_health_name	**Optional.** The tablespace, datafile, wait event, latch, enqueue depending on the mode or SQL statement to be executed with “db2_health_mode” sql.
db2_health_name2	**Optional.** If “db2_health_name” is a sql statement, “db2_health_name2” can be used to appear in the output and the performance data.
db2_health_regexp	**Optional.** If set to true, “db2_health_name” will be interpreted as a regular expression. Defaults to false.
db2_health_units	**Optional.** This is used for a better output of mode=sql and for specifying thresholds for mode=tablespace-free. Possible values are “%”, “KB”, “MB” and “GB”.
db2_health_maximum_time	**Optional.** Used for the maximum amount of time a certain event has not happened.
db2_health_mitigation	**Optional.** Classifies the severity of an offline tablespace.
db2_health_lookback	**Optional.** How many days in the past db2_health check should look back to calculate exitcode.

Name	Description
db2_health_report	**Optional.** Report can be used to output only the bad news. Possible values are “short”, “long”, “html”. Defaults to `short`.
db2_health_env_db2_home	**Required.** Specifies the location of the db2 client libraries as environment variable `DB2_HOME`. Defaults to “/opt/ibm/db2/V10.5”.
db2_health_env_db2_version	**Optional.** Specifies the DB2 version as environment variable `DB2_VERSION`.

10.7.1.2 mssql_health

The check_mssql_health plugin uses the `DBD::Sybase` Perl library based on FreeTDS to monitor a MS SQL server.

The Git repository is located on GitHub.

Custom attributes passed as command parameters:

Name	Description
mssql_health_hostname	**Optional.** Specifies the database hostname or address. No default because you typically use “mssql_health_server”.
mssql_health_username	**Optional.** The username for the database connection.
mssql_health_password	**Optional.** The password for the database connection.
mssql_health_port	**Optional.** Specifies the database port. No default because you typically use “mssql_health_server”.
mssql_health_server	**Optional.** The name of a predefined connection (in freetds.conf).
mssql_health_currentdb	**Optional.** The name of a database which is used as the current database for the connection.
mssql_health_offlineok	**Optional.** Set this to true if offline databases are perfectly ok for you. Defaults to false.
mssql_health_noofflines	**Optional.** Set this to true to ignore offline databases. Defaults to false.
mssql_health_dbthresholds	**Optional.** With this parameter thresholds are read from the database table check_mssql_health_thresholds.
mssql_health_notemptydbs	**Optional.** Set this to true to ignore temporary databases/tablespaces. Defaults to false.
mssql_health_commit	**Optional.** Set this to true to turn on autocommit for the dbd::sybase module. Defaults to false.
mssql_health_method	**Optional.** How the plugin should connect to the database (dbi for the perl module `DBD::Sybase` (default) and `sqlrelay` for the SQLRelay proxy).

Name	Description
mssql_health_mode	**Required.** The mode uses predefined keywords for the different checks. For example “connection-time”, “database-free” or “sql”.
mssql_health_regexp	**Optional.** If set to true, “mssql_health_name” will be interpreted as a regular expression. Defaults to false.
mssql_health_warning	**Optional.** The warning threshold depending on the mode.
mssql_health_critical	**Optional.** The critical threshold depending on the mode.
mssql_health_warningx	**Optional.** A possible override for the warning threshold.
mssql_health_criticalx	**Optional.** A possible override for the critical threshold.
mssql_health_units	**Optional.** This is used for a better output of mode=sql and for specifying thresholds for mode=tablespace-free. Possible values are “%”, “KB”, “MB” and “GB”.
mssql_health_name	**Optional.** Depending on the mode this could be the database name or a SQL statement.
mssql_health_name2	**Optional.** If “mssql_health_name” is a sql statement, “mssql_health_name2” can be used to appear in the output and the performance data.
mssql_health_name3	**Optional.** Additional argument used for ‘database-file-free’ mode for example.
mssql_health_extraopts	**Optional.** Read command line arguments from an external file.
mssql_health_blacklist	**Optional.** Blacklist some (missing/failed) components
mssql_health_mitigation	**Optional.** The parameter allows you to change a critical error to a warning.
mssql_health_lookback	**Optional.** The amount of time you want to look back when calculating average rates.
mssql_health_environment	**Optional.** Add a variable to the plugin’s environment.
mssql_health_negate	**Optional.** Emulate the negate plugin. –negate warning=critical –negate unknown=critical.
mssql_health_morphmessage	**Optional.** Modify the final output message.
mssql_health_morphperfdata	**Optional.** The parameter allows you to change performance data labels.
mssql_health_selectedperfdata	**Optional.** The parameter allows you to limit the list of performance data.
mssql_health_report	**Optional.** Report can be used to output only the bad news. Possible values are “short”, “long”, “html”. Defaults to `short`.
mssql_health_multiline	**Optional.** Multiline output.
mssql_health_withmymodules_dyn_dir	**Optional.** Additional directory where additional my-modules for the my-modes will be searched in this directory.
mssql_health_statefilesdir	**Optional.** An alternate directory where the plugin can save files.

Name	Description
mssql_health_isvalidtime	**Optional.** Signals the plugin to return OK if now is not a valid check time.
mssql_health_timeout	**Optional.** Plugin timeout. Defaults to 15s.

10.7.1.3 mysql_health

The check_mysql_health plugin uses the `DBD::MySQL` Perl library to monitor a MySQL or MariaDB database.

The Git repository is located on GitHub.

Custom attributes passed as command parameters:

Name	Description
mysql_health_hostname	**Required.** Specifies the database hostname or address. Defaults to "*address*" or "*address*6" if the `address` attribute is not set.
mysql_health_port	**Optional.** Specifies the database port. Defaults to 3306 (or 1186 for "mysql_health_mode" cluster).
mysql_health_socket	**Optional.** Specifies the database unix socket. No default.
mysql_health_username	**Optional.** The username for the database connection.
mysql_health_password	**Optional.** The password for the database connection.
mysql_health_database	**Optional.** The database to connect to. Defaults to information_schema.
mysql_health_warning	**Optional.** The warning threshold depending on the mode.
mysql_health_critical	**Optional.** The critical threshold depending on the mode.
mysql_health_warningx	**Optional.** The extended warning thresholds depending on the mode.
mysql_health_criticalx	**Optional.** The extended critical thresholds depending on the mode.
mysql_health_mode	**Required.** The mode uses predefined keywords for the different checks. For example "connection-time", "slave-lag" or "sql".
mysql_health_method	**Optional.** How the plugin should connect to the database (`dbi` for using DBD::Mysql (default), `mysql` for using the mysql-Tool).
mysql_health_commit	**Optional.** Turns on autocommit for the dbd::* module.
mysql_health_notemp	**Optional.** Ignore temporary databases/tablespaces.
mysql_health_nooffline	**Optional.** Skip the offline databases.
mysql_health_regexp	**Optional.** Parameter name/name2/name3 will be interpreted as (perl) regular expression.
mysql_health_name	**Optional.** The name of a specific component to check.
mysql_health_name2	**Optional.** The secondary name of a component.

Name	Description
mysql_health_name3	**Optional.** The tertiary name of a component.
mysql_health_units	**Optional.** This is used for a better output of mode=sql and for specifying thresholds for mode=tablespace-free. Possible values are “%”, “KB”, “MB” and “GB”.
mysql_health_labelformat	**Optional.** One of those formats pnp4nagios or groundwork. Defaults to pnp4nagios.
mysql_health_extra-opts	**Optional.** Read command line arguments from an external file.
mysql_health_blacklist	**Optional.** Blacklist some (missing/failed) components
mysql_health_mitigation	**Optional.** The parameter allows you to change a critical error to a warning.
mysql_health_lookback	**Optional.** The amount of time you want to look back when calculating average rates.
mysql_health_environment	**Optional.** Add a variable to the plugin’s environment.
mysql_health_morphmessage	**Optional.** Modify the final output message.
mysql_health_morphperfdata	**Optional.** The parameter allows you to change performance data labels.
mysql_health_selectedperfdata	**Optional.** The parameter allows you to limit the list of performance data.
mysql_health_report	**Optional.** Can be used to shorten the output.
mysql_health_multiline	**Optional.** Multiline output.
mysql_health_negate	**Optional.** Emulate the negate plugin. –negate warning=critical –negate unknown=critical.
mysql_health_with-mymodules-dyn-dir	**Optional.** Also in modules for the my-modes will be searched in this directory.
mysql_health_statefilesdir	**Optional.** An alternate directory where the plugin can save files.
mysql_health_isvalidtime	**Optional.** Signals the plugin to return OK if now is not a valid check time.
mysql_health_timeout	**Optional.** Plugin timeout. Defaults to 60s.

10.7.1.4 oracle_health

The check_oracle_health plugin uses the `DBD::Oracle` Perl library to monitor an Oracle database.

The Git repository is located on GitHub.

Custom attributes passed as command parameters:

Name	Description
oracle_health_connect	**Required.** Specifies the database connection string (from tnsnames.ora).
oracle_health_username	**Optional.** The username for the database connection.

Name	Description
oracle_health_password	**Optional.** The password for the database connection.
oracle_health_warning	**Optional.** The warning threshold depending on the mode.
oracle_health_critical	**Optional.** The critical threshold depending on the mode.
oracle_health_mode	**Required.** The mode uses predefined keywords for the different checks. For example “connection-time”, “flash-recovery-area-usage” or “sql”.
oracle_health_method	**Optional.** How the plugin should connect to the database (`dbi` for using DBD::Oracle (default), `sqlplus` for using the sqlplus-Tool).
oracle_health_name	**Optional.** The tablespace, datafile, wait event, latch, enqueue depending on the mode or SQL statement to be executed with “oracle_health_mode” sql.
oracle_health_name2	**Optional.** If “oracle_health_name” is a sql statement, “oracle_health_name2” can be used to appear in the output and the performance data.
oracle_health_regexp	**Optional.** If set to true, “oracle_health_name” will be interpreted as a regular expression. Defaults to false.
oracle_health_units	**Optional.** This is used for a better output of mode=sql and for specifying thresholds for mode=tablespace-free. Possible values are “%”, “KB”, “MB” and “GB”.
oracle_health_ident	**Optional.** If set to true, outputs instance and database names. Defaults to false.
oracle_health_commit	**Optional.** Set this to true to turn on autocommit for the dbd::oracle module. Defaults to false.
oracle_health_noperfdata	**Optional.** Set this to true if you want to disable perfdata. Defaults to false.
oracle_health_timeout	**Optional.** Plugin timeout. Defaults to 60s.
oracle_health_report	**Optional.** Select the plugin output format. Can be short or long. Default to long.

Environment Macros:

Name	Description
ORACLE_HOME	**Required.** Specifies the location of the oracle instant client libraries. Defaults to “/usr/lib/oracle/11.2/client64/lib”. Can be overridden by setting the custom attribute `oracle_home`.
LD_LIBRARY_PATH	**Required.** Specifies the location of the oracle instant client libraries for the run-time shared library loader. Defaults to “/usr/lib/oracle/11.2/client64/lib”. Can be overridden by setting the custom attribute `oracle_ld_library_path`.

Name	Description
TNS_ADMIN	**Required.** Specifies the location of the tnsnames.ora including the database connection strings. Defaults to "/etc/icinga2/plugin-configs". Can be overridden by setting the custom attribute `oracle_tns_admin`.

10.7.1.5 postgres

The check_postgres plugin uses the `psql` binary to monitor a PostgreSQL database.

The Git repository is located on GitHub.

Custom attributes passed as command parameters:

Name	Description
postgres_host	**Optional.** Specifies the database hostname or address. Defaults to "*address*" or "*address*6" if the `address` attribute is not set. If "postgres_unixsocket" is set to true, falls back to unix socket.
postgres_port	**Optional.** Specifies the database port. Defaults to 5432.
postgres_dbname	**Optional.** Specifies the database name to connect to. Defaults to "postgres" or "template1".
postgres_dbuser	**Optional.** The username for the database connection. Defaults to "postgres".
postgres_dbpass	**Optional.** The password for the database connection. You can use a .pgpass file instead.
postgres_dbservice	**Optional.** Specifies the service name to use inside of pg_service.conf.
postgres_warning	**Optional.** Specifies the warning threshold, range depends on the action.
postgres_critical	**Optional.** Specifies the critical threshold, range depends on the action.
postgres_include	**Optional.** Specifies name(s) items to specifically include (e.g. tables), depends on the action.
postgres_exclude	**Optional.** Specifies name(s) items to specifically exclude (e.g. tables), depends on the action.
postgres_includeuser	**Optional.** Include objects owned by certain users.
postgres_excludeuser	**Optional.** Exclude objects owned by certain users.
postgres_standby	**Optional.** Assume that the server is in continuous WAL recovery mode if set to true. Defaults to false.
postgres_production	**Optional.** Assume that the server is in production mode if set to true. Defaults to false.
postgres_action	**Required.** Determines the test executed.

Name	Description
postgres_unixsocket	**Optional.** If "postgres_unixsocket" is set to true, the unix socket is used instead of an address. Defaults to false.
postgres_query	**Optional.** Query for "custom_query" action.
postgres_valtype	**Optional.** Value type of query result for "custom_query".
postgres_reverse	**Optional.** If "postgres_reverse" is set, warning and critical values are reversed for "custom_query" action.
postgres_tempdir	**Optional.** Specify directory for temporary files. The default directory is dependent on the OS. More details here.

10.7.1.6 mongodb

The check_mongodb.py plugin uses the `pymongo` Python library to monitor a MongoDB instance.

Custom attributes passed as command parameters:

Name	Description
mongodb_host	**Required.** Specifies the hostname or address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
mongodb_port	**Required.** The port mongodb is running on.
mongodb_user	**Optional.** The username you want to login as.
mongodb_passwd	**Optional.** The password you want to use for that user.
mongodb_authdb	**Optional.** The database you want to authenticate against.
mongodb_warning	**Optional.** The warning threshold we want to set.
mongodb_critical	**Optional.** The critical threshold we want to set.
mongodb_action	**Required.** The action you want to take.
mongodb_maxlag	**Optional.** Get max replication lag (for replication_lag action only).
mongodb_mapped_memory	**Optional.** Get mapped memory instead of resident (if resident memory can not be read).
mongodb_perfdata	**Optional.** Enable output of Nagios performance data.
mongodb_database	**Optional.** Specify the database to check.
mongodb_alldatabases	**Optional.** Check all databases (action database_size).
mongodb_ssl	**Optional.** Connect using SSL.
mongodb_replicaset	**Optional.** Connect to replicaset.
mongodb_replcheck	**Optional.** If set to true, will enable the mongodb_replicaset value needed for "replica_primary" check.

Name	Description
mongodb_querytype	**Optional.** The query type to check [query|insert|update|delete|getmore|command] from queries_per_second.
mongodb_collection	**Optional.** Specify the collection to check.
mongodb_sampletime	**Optional.** Time used to sample number of pages faults.

10.7.1.7 elasticsearch

The check_elasticsearch plugin uses the HTTP API to monitor an Elasticsearch node.

Custom attributes passed as command parameters:

Name	Description
elasticsearch_host	**Optional.** Hostname or network address to probe. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
elasticsearch_failure_options	**Optional.** A comma-separated list of ElasticSearch attributes that make up your cluster's failure domain.
elasticsearch_master_options	**Optional.** Issue a warning if the number of master-eligible nodes in the cluster drops below this number. By default, do not monitor the number of nodes in the cluster.
elasticsearch_port	**Optional.** TCP port to probe. The ElasticSearch API should be listening here. Defaults to 9200.
elasticsearch_prefix	**Optional.** Optional prefix (e.g. 'es') for the ElasticSearch API. Defaults to ''.
elasticsearch_yellow_ok	**Optional.** Instead of issuing a 'warning' for a yellow cluster state, issue a 'critical' alert. Defaults to false.

10.7.1.8 redis

The check_redis.pl plugin uses the `Redis` Perl library to monitor a Redis instance. The plugin can measure response time, hitrate, memory utilization, check replication synchronization, etc. It is also possible to test data in a specified key and calculate averages or summaries on ranges.

Custom attributes passed as command parameters:

Name	Description
redis_hostname	**Required.** Hostname or IP Address to check. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.

Name	Description
redis_port	**Optional.** Port number to query. Default to "6379".
redis_database	**Optional.** Database name (usually a number) to query, needed for **redis_query**.
redis_password	**Optional.** Password for Redis authentication. Safer alternative is to put them in a file and use **redis_credentials**.
redis_credentials	**Optional.** Credentials file to read for Redis authentication.
redis_timeout	**Optional.** Allows to set timeout for execution of this plugin.
redis_variables	**Optional.** List of variables from info data to do threshold checks on.
redis_warn	**Optional.** This option can only be used if **redis_variables** is used and the number of values listed here must exactly match number of variables specified.
redis_crit	**Optional.** This option can only be used if **redis_variables** is used and the number of values listed here must exactly match number of variables specified.
redis_perfparse	**Optional.** This should only be used with variables and causes variable data not only to be printed as part of main status line but also as perfparse compatible output. Defaults to false.
redis_perfvars	**Optional.** This allows to list variables which values will go only into perfparse output (and not for threshold checking).
redis_prev_perfdata	**Optional.** If set to true, previous performance data are used to calculate rate of change for counter statistics variables and for proper calculation of hitrate. Defaults to false.
redis_rate_label	**Optional.** Prefix or Suffix label used to create a new variable which has rate of change of another base variable. You can specify PREFIX or SUFFIX or both as one string separated by ",". Default if not specified is suffix "_rate".
redis_query	**Optional.** Option specifies key to query and optional variable name to assign the results to after.
redis_option	**Optional.** Specifiers are separated by "," and must include NAME or PATTERN.
redis_response_time	**Optional.** If this is used, plugin will measure and output connection response time in seconds. With **redis_perfparse** this would also be provided on perf variables.
redis_hitrate	**Optional.** Calculates Hitrate and specify values are interpreted as WARNING and CRITICAL thresholds.

Name	Description
redis_memory_utilization	**Optional.** This calculates percent of total memory on system used by redis. Total_memory on server must be specified with **redis_total_memory**. If you specify by itself, the plugin will just output this info. Parameter values are interpreted as WARNING and CRITICAL thresholds.
redis_total_memory	**Optional.** Amount of memory on a system for memory utilization calculation. Use system memory or max_memory setting of redis.
redis_replication_delay	**Optional.** Allows to set threshold on replication delay info.

10.7.2 Hardware

This category includes all plugin check commands for various hardware checks.

10.7.2.1 hpasm

The check_hpasm plugin monitors the hardware health of HP Proliant Servers, provided that the `hpasm` (HP Advanced Server Management) software is installed. It is also able to monitor the system health of HP Bladesystems and storage systems.

The plugin can run in two different ways:

1. Local execution using the `hpasmcli` command line tool.
2. Remote SNMP query which invokes the HP Insight Tools on the remote node.

You can either set or omit `hpasm_hostname` custom attribute and select the corresponding node.

The `hpasm_remote` attribute enables the plugin to execute remote SNMP queries if set to `true`. For compatibility reasons this attribute uses `true` as default value, and ensures that specifying the `hpasm_hostname` always enables remote checks.

Custom attributes passed as command parameters:

Name	Description
hpasm_hostname	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
hpasm_community	**Optional.** SNMP community of the server (SNMP v1/2 only).
hpasm_protocol	**Optional.** The SNMP protocol to use (default: 2c, other possibilities: 1,3).
hpasm_port	**Optional.** The SNMP port to use (default: 161).

Name	Description
hpasm_blacklist	**Optional.** Blacklist some (missing/failed) components.
hpasm_ignore-dimms	**Optional.** Ignore "N/A"-DIMM status on misc. servers (e.g. older DL320).
hpasm_ignore-fan-redundancy	**Optional.** Ignore missing redundancy partners.
hpasm_customthresholds	**Optional.** Use custom thresholds for certain temperatures.
hpasm_eventrange	**Optional.** Period of time before critical IML events respectively become warnings or vanish. A range is described as a number and a unit (s, m, h, d), e.g. –eventrange 1h/20m.
hpasm_perfdata	**Optional.** Output performance data. If your performance data string becomes too long and is truncated by Nagios, then you can use –perfdata=short instead. This will output temperature tags without location information.
hpasm_username	**Optional.** The securityName for the USM security model (SNMPv3 only).
hpasm_authpassword	**Optional.** The authentication password for SNMPv3.
hpasm_authprotocol	**Optional.** The authentication protocol for SNMPv3 (md5|sha).
hpasm_privpassword	**Optional.** The password for authPriv security level.
hpasm_privprotocol	**Optional.** The private protocol for SNMPv3 (des|aes|aes128|3des|3desde).
hpasm_servertype	**Optional.** The type of the server: proliant (default) or bladesystem.
hpasm_eval-nics	**Optional.** Check network interfaces (and groups). Try it and report me whyt you think about it. I need to build up some know how on this subject. If you get an error and think, it is not justified for your configuration, please tell me about it. (always send the output of "snmpwalk -On …. 1.3.6.1.4.1.232" and a description how you setup your nics and why it is correct opposed to the plugins error message.
hpasm_remote	**Optional.** Run remote SNMP checks if enabled. Otherwise checks are executed locally using the `hpasmcli` binary. Defaults to `true`.

10.7.2.2 openmanage

The check_openmanage plugin checks the hardware health of Dell PowerEdge (and some PowerVault) servers. It uses the Dell OpenManage Server Administrator (OMSA) software, which must be running on the monitored system. check_openmanage can be used remotely with SNMP or locally with icinga2 agent, check_by_ssh or similar, whichever suits your needs and particular taste.

The plugin checks the health of the storage subsystem, power supplies, memory modules, temperature probes etc., and gives an alert if any of the components are faulty or operate outside normal parameters.

Custom attributes passed as command parameters:

Name	Description
openmanage_all	**Optional.** Check everything, even log content
openmanage_blacklist	**Optional.** Blacklist missing and/or failed components
openmanage_check	**Optional.** Fine-tune which components are checked
openmanage_community	**Optional.** SNMP community string [default=public]
openmanage_config	**Optional.** Specify configuration file
openmanage_critical	**Optional.** Custom temperature critical limits
openmanage_extinfo	**Optional.** Append system info to alerts
openmanage_fahrenheit	**Optional.** Use Fahrenheit as temperature unit
openmanage_hostname	**Optional.** Hostname or IP (required for SNMP)
openmanage_htmlinfo	**Optional.** HTML output with clickable links
openmanage_info	**Optional.** Prefix any alerts with the service tag
openmanage_ipv6	**Optional.** Use IPv6 instead of IPv4 [default=no]
openmanage_legacy_perfdata	**Optional.** Legacy performance data output
openmanage_no_storage	**Optional.** Don't check storage
openmanage_only	**Optional.** Only check a certain component or alert type
openmanage_perfdata	**Optional.** Output performance data [default=no]
openmanage_port	**Optional.** SNMP port number [default=161]
openmanage_protocol	**Optional.** SNMP protocol version [default=2c]
openmanage_short_state	**Optional.** Prefix alerts with alert state abbreviated
openmanage_show_blacklist	**Optional.** Show blacklistings in OK output
openmanage_state	**Optional.** Prefix alerts with alert state
openmanage_tcp	**Optional.** Use TCP instead of UDP [default=no]
openmanage_timeout	**Optional.** Plugin timeout in seconds [default=30]
openmanage_vdisk_critical	**Optional.** Make any alerts on virtual disks critical
openmanage_warning	**Optional.** Custom temperature warning limits

10.7.2.3 adaptec-raid

The check_adaptec_raid plugin uses the `arcconf` binary to monitor Adaptec RAID controllers.

Custom attributes passed as command parameters:

Name	Description
adaptec_controller_number	**Required.** Controller number to monitor.
arcconf_path	**Required.** Path to the `arcconf` binary, e.g. "/sbin/arcconf".

10.7.2.4 lsi-raid

The check_lsi_raid plugin uses the `storcli` binary to monitor MegaRAID RAID controllers.

Custom attributes passed as command parameters:

Name	Description
lsi_controller_number	**Required.** Controller number to monitor.
storcli_path	**Required.** Path to the `storcli` binary, e.g. "/usr/sbin/storcli".

10.7.2.5 smart-attributes

The check_smart_attributes plugin uses the `smartctl` binary to monitor SMART values of SSDs and HDDs.

Custom attributes passed as command parameters:

Name	Description
smart_attributes_config_path	**Required.** Path to the smart attributes config file (e.g. check_smartdb.json).
smart_attributes_device	**Required.** Device name (e.g. /dev/sda) to monitor.

10.7.3 IcingaCLI

This category includes all plugins using the icingacli provided by Icinga Web 2.

The user running Icinga 2 needs sufficient permissions to read the Icinga Web 2 configuration directory. e.g. `usermod -a -G icingaweb2 icinga`. You need to restart, not reload Icinga 2 for the new group membership to work.

10.7.3.1 Business Process

This subcommand is provided by the business process module and executed as `icingacli businessprocess` CLI command.

Custom attributes passed as command parameters:

Name	Description
icingacli_businessprocess_process	**Required.** Business process to monitor.
icingacli_businessprocess_config	**Optional.** Configuration file containing your business process without file extension.
icingacli_businessprocess_details	**Optional.** Get details for root cause analysis. Defaults to false.
icingacli_businessprocess_statetype	**Optional.** Define which state type to look at, `soft` or `hard`. Overrides the default value inside the businessprocess module, if configured.

10.7.3.2 Director

This subcommand is provided by the director module > 1.4.2 and executed as `icingacli director health check`. Please refer to the documentation for all available sub-checks.

Custom attributes passed as command parameters:

Name	Description
icingacli_director_check	**Optional.** Run only a specific test suite.
icingacli_director_db	**Optional.** Use a specific Icinga Web DB resource.

10.7.4 IPMI Devices

This category includes all plugins for IPMI devices.

10.7.4.1 ipmi-sensor

The check_ipmi_sensor plugin uses the `ipmimonitoring` binary to monitor sensor data for IPMI devices. Please read the documentation for installation and configuration details.

Custom attributes passed as command parameters:

Name	Description
ipmi_address	**Required.** Specifies the remote host (IPMI device) to check. Defaults to "*address*".

Name	Description
ipmi_config_file	**Optional.** Path to the FreeIPMI configuration file. It should contain IPMI username, IPMI password, and IPMI privilege-level.
ipmi_username	**Optional.** The IPMI username.
ipmi_password	**Optional.** The IPMI password.
ipmi_privilege_level	**Optional.** The IPMI privilege level of the IPMI user.
ipmi_backward_compatibility	**Optional.** Enable backward compatibility mode, useful for FreeIPMI 0.5.* (this omits FreeIPMI options "–quiet-cache" and "–sdr-cache-recreate").
ipmi_sensor_type	**Optional.** Limit sensors to query based on IPMI sensor type. Examples for IPMI sensor types are 'Fan', 'Temperature' and 'Voltage'.
ipmi_sel_type	**Optional.** Limit SEL entries to specific types, run 'ipmi-sel -L' for a list of types. All sensors are populated to the SEL and per default all sensor types are monitored.
ipmi_exclude_sensor_id	**Optional.** Exclude sensor matching ipmi_sensor_id.
ipmi_exclude_sensor_type	**Optional.** Exclude sensor based on IPMI sensor type. (Comma-separated)
ipmi_exclude_sel	**Optional.** Exclude SEL entries of specific sensor types. (comma-separated list).
ipmi_sensor_id	**Optional.** Include sensor matching ipmi_sensor_id.
ipmi_protocol_lan_version	**Optional.** Change the protocol LAN version. Defaults to "LAN_2_0".
ipmi_number_of_active_fans	**Optional.** Number of fans that should be active. Otherwise a WARNING state is returned.
ipmi_show_fru	**Optional.** Print the product serial number if it is available in the IPMI FRU data.
ipmi_no_sel_checking	**Optional.** Turn off system event log checking via ipmi-sel.
ipmi_no_thresholds	**Optional.** Turn off performance data thresholds from output-sensor-thresholds.
ipmi_verbose	**Optional.** Be Verbose multi line output, also with additional details for warnings.
ipmi_debug	**Optional.** Be Verbose debugging output, followed by normal multi line output.

10.7.4.2 ipmi-alive

The `ipmi-alive` check commands allows you to create a ping check for the IPMI Interface.

Custom attributes passed as command parameters:

Name	Description
ping_address	**Optional.** The address of the IPMI interface. Defaults to "*address*" if the IPMI interface's `address` attribute is set, "*address*6" otherwise.
ping_wrta	**Optional.** The RTA warning threshold in milliseconds. Defaults to 5000.
ping_wpl	**Optional.** The packet loss warning threshold in %. Defaults to 100.
ping_crta	**Optional.** The RTA critical threshold in milliseconds. Defaults to 5000.
ping_cpl	**Optional.** The packet loss critical threshold in %. Defaults to 100.
ping_packets	**Optional.** The number of packets to send. Defaults to 1.
ping_timeout	**Optional.** The plugin timeout in seconds. Defaults to 0 (no timeout).

10.7.5 Log Management

This category includes all plugins for log management, for example Logstash.

10.7.5.1 logstash

The logstash plugin connects to the Node API of Logstash. This plugin requires at least Logstash version 5.0.x.

The Node API is not activated by default. You have to configure your Logstash installation in order to allow plugin connections.

Name	Description
logstash_hostname	**Optional.** Hostname where Logstash is running. Defaults to `check_address`
logstash_port	**Optional.** Port where Logstash is listening for API requests. Defaults to 9600
logstash_file_descr_warn	**Optional.** Warning threshold of file descriptor usage in percent. Defaults to 85 (percent).
logstash_file_descr_crit	**Optional.** Critical threshold of file descriptor usage in percent. Defaults to 95 (percent).
logstash_heap_warn	**Optional.** Warning threshold of heap usage in percent. Defaults to 70 (percent).
logstash_heap_crit	**Optional.** Critical threshold of heap usage in percent Defaults to 80 (percent).
logstash_inflight_warn	**Optional.** Warning threshold of inflight events.
logstash_inflight_crit	**Optional.** Critical threshold of inflight events.

Name	Description
logstash_cpu_warn	**Optional.** Warning threshold for cpu usage in percent.
logstash_cpu_crit	**Optional.** Critical threshold for cpu usage in percent.

10.7.6 Metrics

This category includes all plugins for metric-based checks.

10.7.6.1 graphite

The check_graphite plugin uses the `rest-client` Ruby library to monitor a Graphite instance.

Custom attributes passed as command parameters:

Name	Description
graphite_url	**Required.** Target url.
graphite_metric	**Required.** Metric path string.
graphite_shortname	**Optional.** Metric short name (used for performance data).
graphite_duration	**Optional.** Length, in minute of data to parse (default: 5).
graphite_function	**Optional.** Function applied to metrics for thresholds (default: average).
graphite_warning	**Required.** Warning threshold.
graphite_critical	**Required.** Critical threshold.
graphite_units	**Optional.** Adds a text tag to the metric count in the plugin output. Useful to identify the metric units. Doesn't affect data queries.
graphite_message	**Optional.** Text message to output (default: "metric count:").
graphite_zero_on_error	**Optional.** Return 0 on a graphite 500 error.
graphite_link_graph	**Optional.** Add a link in the plugin output, showing a 24h graph for this metric in graphite.

10.7.7 Network Components

This category includes all plugins for various network components like routers, switches and firewalls.

10.7.7.1 interfacetable

The check_interfacetable_v3t plugin generates a html page containing information about the monitored node and all of its interfaces.

The Git repository is located on GitHub.

Custom attributes passed as command parameters:

Name	Description
interfacetable_hostquery	**Required.** Specifies the remote host to poll. Defaults to "*address*".
interfacetable_hostdisplay	**Optional.** Specifies the hostname to display in the HTML link. Defaults to "$host.display_name$".
interfacetable_regex	**Optional.** Interface names and property names for some other options will be interpreted as regular expressions. Defaults to false.
interfacetable_outputsimple	**Optional.** Reduce the verbosity of the plugin output. Defaults to false.
interfacetable_exclude	**Optional.** Comma separated list of interfaces globally excluded from the monitoring.
interfacetable_include	**Optional.** Comma separated list of interfaces globally included in the monitoring.
interfacetable_aliasmatching	**Optional.** Allow you to specify alias in addition to interface names. Defaults to false.
interfacetable_excludetraffic	**Optional.** Comma separated list of interfaces excluded from traffic checks.
interfacetable_includetraffic	**Optional.** Comma separated list of interfaces included for traffic checks.
interfacetable_warningtraffic	**Optional.** Interface traffic load percentage leading to a warning alert.
interfacetable_criticaltraffic	**Optional.** Interface traffic load percentage leading to a critical alert.
interfacetable_pkt	**Optional.** Add unicast/non-unicast pkt stats for each interface.
interfacetable_trafficwithpkt	**Optional.** Enable traffic calculation using pkt counters instead of octet counters. Useful when using 32-bit counters to track the load on > 1GbE interfaces. Defaults to false.
interfacetable_trackproperty	**Optional.** List of tracked properties.
interfacetable_excludeproperty	**Optional.** Comma separated list of interfaces excluded from the property tracking.
interfacetable_includeproperty	**Optional.** Comma separated list of interfaces included in the property tracking.
interfacetable_community	**Optional.** Specifies the snmp v1/v2c community string. Defaults to "public" if using snmp v1/v2c, ignored using v3.
interfacetable_snmpv2	**Optional.** Use snmp v2c. Defaults to false.

Name	Description
interfacetable_login	**Optional.** Login for snmpv3 authentication.
interfacetable_passwd	**Optional.** Auth password for snmpv3 authentication.
interfacetable_privpass	**Optional.** Priv password for snmpv3 authentication.
interfacetable_protocol	**Optional.** Authentication protocol,Priv protocol for snmpv3 authentication.
interfacetable_domain	**Optional.** SNMP transport domain.
interfacetable_context	**Optional.** Context name for the snmp requests.
interfacetable_port	**Optional.** SNMP port. Defaults to standard port.
interfacetable_64bits	**Optional.** Use SNMP 64-bits counters. Defaults to false.
interfacetable_maxrepetitions	**Optional.** Increasing this value may enhance snmp query performances by gathering more results at one time.
interfacetable_snmptimeout	**Optional.** Define the Transport Layer timeout for the snmp queries.
interfacetable_snmpretries	**Optional.** Define the number of times to retry sending a SNMP message.
interfacetable_snmpmsgsize	**Optional.** Size of the SNMP message in octets, useful in case of too long responses. Be careful with network filters. Range 484 - 65535. Apply only to netsnmp perl bindings. The default is 1472 octets for UDP/IPv4, 1452 octets for UDP/IPv6, 1460 octets for TCP/IPv4, and 1440 octets for TCP/IPv6.
interfacetable_unixsnmp	**Optional.** Use unix snmp utilities for snmp requests. Defaults to false, which means use the perl bindings.
interfacetable_enableperfdata	**Optional.** Enable port performance data. Defaults to false.
interfacetable_perfdatamode	**Optional.** Define which performance data will be generated. Possible values are “full” (default), “loadonly”, “globalonly”.
interfacetable_perfdatathresholds	**Optional.** Define which thresholds are printed in the generated performance data. Possible values are “full” (default), “loadonly”, “globalonly”.
interfacetable_perfdatadir	**Optional.** When specified, the performance data are also written directly to a file, in the specified location.
interfacetable_perfdatatemplate	**Optional.** Specify additional parameters for output performance data to PNP. Defaults to “*service.name*”, only affects **interfacetable_perfdatadir**.
interfacetable_grapher	**Optional.** Specify the used graphing solution. Possible values are “pnp4nagios” (default), “nagiosgrapher”, “netwaysgrapherv2” and “ingraph”.
interfacetable_grapherurl	**Optional.** Graphing system url. Default depends on **interfacetable_grapher**.

Name	Description
interfacetable_portpercent	**Optional.** Traffic could be reported in bits (counters) or in bps (calculated value).
interfacetable_nodetype	**Optional.** Specify the node type, for specific information to be printed / specific oids to be used. Possible values: “standard” (default), “cisco”, “hp”, “netscreen”, “netapp”, “bigip”, “bluecoat”, “brocade”, “brocade-nos”, “nortel”, “hpux”.
interfacetable_duplex	**Optional.** Add the duplex mode property for each interface in the interface table. Defaults to false.
interfacetable_stp	**Optional.** Add the stp state property for each interface in the interface table. Defaults to false.
interfacetable_vlan	**Optional.** Add the vlan attribution property for each interface in the interface table. Defaults to false. This option is available only for the following nodetypes: “cisco”, “hp”, “nortel”
interfacetable_noipinfo	**Optional.** Remove the ip information for each interface from the interface table. Defaults to false.
interfacetable_alias	**Optional.** Add the alias information for each interface in the interface table. Defaults to false.
interfacetable_accessmethod	**Optional.** Access method for a shortcut to the host in the HTML page. Format is : [:] Where method can be: ssh, telnet, http or https.
interfacetable_htmltabletarget	**Optional.** Specifies the windows or the frame where the [details] link will load the generated html page. Possible values are: “_blank“,”_self” (default), “_parent“,”_top“, or a frame name.
interfacetable_delta	**Optional.** Set the delta used for interface throughput calculation in seconds.
interfacetable_ifs	**Optional.** Input field separator. Defaults to “,”.
interfacetable_cache	**Optional.** Define the retention time of the cached data in seconds.
interfacetable_noifloadgradient	**Optional.** Disable color gradient from green over yellow to red for the load percentage. Defaults to false.
interfacetable_nohuman	**Optional.** Do not translate bandwidth usage in human readable format. Defaults to false.
interfacetable_snapshot	**Optional.** Force the plugin to run like if it was the first launch. Defaults to false.
interfacetable_timeout	**Optional.** Define the global timeout limit of the plugin in seconds. Defaults to “15s”.
interfacetable_css	**Optional.** Define the css stylesheet used by the generated html files. Possible values are “classic”, “icinga” or “icinga-alternate1”.
interfacetable_config	**Optional.** Specify a config file to load.

Name	Description
interfacetable_noconfigtable	**Optional.** Disable configuration table on the generated HTML page. Defaults to false.
interfacetable_notips	**Optional.** Disable the tips in the generated html tables. Defaults to false.
interfacetable_defaultsorting	**Optional.** Default table sorting can be "index" (default) or "name".
interfacetable_tablesperif	**Optional.** Generate multiple interface tables, one per interface type. Defaults to false.
interfacetable_notype	**Optional.** Remove the interface type for each interface. Defaults to false.

10.7.7.2 iftraffic

The check_iftraffic plugin checks the utilization of a given interface name using the SNMP protocol.

Custom attributes passed as command parameters:

Name	Description
iftraffic_address	**Required.** Specifies the remote host. Defaults to "*address*".
iftraffic_community	**Optional.** SNMP community. Defaults to "public"' if omitted.
iftraffic_interface	**Required.** Queried interface name.
iftraffic_bandwidth	**Required.** Interface maximum speed in kilo/mega/giga/bits per second.
iftraffic_units	**Optional.** Interface units can be one of these values: `g` (gigabits/s),`m` (megabits/s), `k` (kilobits/s),`b` (bits/s)
iftraffic_warn	**Optional.** Percent of bandwidth usage necessary to result in warning status (defaults to `85`).
iftraffic_crit	**Optional.** Percent of bandwidth usage necessary to result in critical status (defaults to `98`).
iftraffic_max_counter	**Optional.** Maximum counter value of net devices in kilo/mega/giga/bytes.

10.7.7.3 iftraffic64

The check_iftraffic64 plugin checks the utilization of a given interface name using the SNMP protocol.

Custom attributes passed as command parameters:

Name	Description
iftraffic64_address	**Required.** Specifies the remote host. Defaults to "*address*".
iftraffic64_community	**Optional.** SNMP community. Defaults to "public"' if omitted.
iftraffic64_interface	**Required.** Queried interface name.
iftraffic64_bandwidth	**Required.** Interface maximum speed in kilo/mega/giga/bits per second.
iftraffic64_units	**Optional.** Interface units can be one of these values: `g` (gigabits/s),`m` (megabits/s), `k` (kilobits/s),`b` (bits/s)
iftraffic64_warn	**Optional.** Percent of bandwidth usage necessary to result in warning status (defaults to `85`).
iftraffic64_crit	**Optional.** Percent of bandwidth usage necessary to result in critical status (defaults to `98`).
iftraffic64_max_counter	**Optional.** Maximum counter value of net devices in kilo/mega/giga/bytes.

10.7.7.4 interfaces

The check_interfaces plugin uses SNMP to monitor network interfaces and their utilization.

Custom attributes passed as command parameters:

Name	Description
interfaces_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
interfaces_regex	**Optional.** Interface list regexp.
interfaces_exclude_regex	**Optional.** Interface list negative regexp.
interfaces_errors	**Optional.** Number of in errors (CRC errors for cisco) to consider a warning (default 50).
interface_out_errors	**Optional.** Number of out errors (collisions for cisco) to consider a warning (default same as in errors).
interfaces_perfdata	**Optional.** perfdata from last check result.
interfaces_prefix	**Optional.** Prefix interface names with this label.
interfaces_lastcheck	**Optional.** Last checktime (unixtime).
interfaces_bandwidth	**Optional.** Bandwidth warn level in percent.
interfaces_speed	**Optional.** Override speed detection with this value (bits per sec).
interfaces_trim	**Optional.** Cut this number of characters from the start of interface descriptions.

Name	Description
interfaces_mode	**Optional.** Special operating mode (default,cisco,nonbulk,bintec).
interfaces_auth_proto	**Optional.** SNMPv3 Auth Protocol (SHA|MD5)
interfaces_auth_phrase	**Optional.** SNMPv3 Auth Phrase
interfaces_priv_proto	**Optional.** SNMPv3 Privacy Protocol (AES|DES)
interfaces_priv_phrase	**Optional.** SNMPv3 Privacy Phrase
interfaces_user	**Optional.** SNMPv3 User
interfaces_down_is_ok	**Optional.** Disables critical alerts for down interfaces.
interfaces_aliases	**Optional.** Retrieves the interface description.
interfaces_match_aliases	**Optional.** Also match against aliases (Option –aliases automatically enabled).
interfaces_timeout	**Optional.** Sets the SNMP timeout (in ms).
interfaces_sleep	**Optional.** Sleep between every SNMP query (in ms).
interfaces_names	**Optional.** If set to true, use ifName instead of ifDescr.

10.7.7.5 nwc_health

The check_nwc_health plugin uses SNMP to monitor network components. The plugin is able to generate interface statistics, check hardware (CPU, memory, fan, power, etc.), monitor firewall policies, HRSP, load-balancer pools, processor and memory usage.

Currently the following network components are supported: Cisco IOS, Cisco Nexus, Cisco ASA, Cisco PIX, F5 BIG-IP, CheckPoint Firewall1, Juniper NetScreen, HP Procurve, Nortel, Brocade 4100/4900, EMC DS 4700, EMC DS 24, Allied Telesyn. Blue Coat SG600, Cisco Wireless Lan Controller 5500, Brocade ICX6610-24-HPOE, Cisco UC Telefonzeugs, FOUNDRY-SN-AGENT-MIB, FRITZ!BOX 7390, FRITZ!DECT 200, Juniper IVE, Pulse-Gateway MAG4610, Cisco IronPort AsyncOS, Foundry, etc. A complete list can be found in the plugin documentation.

Custom attributes passed as command parameters:

Name	Description
nwc_health_hostname	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
nwc_health_mode	**Optional.** The plugin mode. A list of all available modes can be found in the plugin documentation.

Name	Description
nwc_health_timeout	**Optional.** Seconds before plugin times out (default: 15)
nwc_health_blacklist	**Optional.** Blacklist some (missing/failed) components.
nwc_health_port	**Optional.** The SNMP port to use (default: 161).
nwc_health_domain	**Optional.** The transport domain to use (default: udp/ipv4, other possible values: udp6, udp/ipv6, tcp, tcp4, tcp/ipv4, tcp6, tcp/ipv6).
nwc_health_protocol	**Optional.** The SNMP protocol to use (default: 2c, other possibilities: 1,3).
nwc_health_community	**Optional.** SNMP community of the server (SNMP v1/2 only).
nwc_health_username	**Optional.** The securityName for the USM security model (SNMPv3 only).
nwc_health_authpassword	**Optional.** The authentication password for SNMPv3.
nwc_health_authprotocol	**Optional.** The authentication protocol for SNMPv3 (md5|sha).
nwc_health_privpassword	**Optional.** The password for authPriv security level.
nwc_health_privprotocol	**Optional.** The private protocol for SNMPv3 (des|aes|aes128|3des|3desde).
nwc_health_contextengineid	**Optional.** The context engine id for SNMPv3 (10 to 64 hex characters).
nwc_health_contextname	**Optional.** The context name for SNMPv3 (empty represents the default context).
nwc_health_name	**Optional.** The name of an interface (ifDescr).
nwc_health_dreckspktdb	**Optional.** This parameter must be used instead of –name, because Devel::ptkdb is stealing the latter from the command line.
nwc_health_alias	**Optional.** The alias name of a 64bit-interface (ifAlias)
nwc_health_regexp	**Optional.** A flag indicating that –name is a regular expression
nwc_health_ifspeedin	**Optional.** Override the ifspeed oid of an interface (only inbound)
nwc_health_ifspeedout	**Optional.** Override the ifspeed oid of an interface (only outbound)
nwc_health_ifspeed	**Optional.** Override the ifspeed oid of an interface
nwc_health_units	**Optional.** One of %, B, KB, MB, GB, Bit, KBi, MBi, GBi. (used for e.g. mode interface-usage)

Name	Description
nwc_health_name2	**Optional.** The secondary name of a component.
nwc_health_role	**Optional.** The role of this device in a hsrp group (active/standby/listen).
nwc_health_report	**Optional.** Can be used to shorten the output. Possible values are: 'long' (default), 'short' (to shorten if available), or 'html' (to produce some html outputs if available)
nwc_health_lookback	**Optional.** The amount of time you want to look back when calculating average rates. Use it for mode interface-errors or interface-usage. Without –lookback the time between two runs of check_nwc_health is the base for calculations. If you want your checkresult to be based for example on the past hour, use –lookback 3600.
nwc_health_warning	**Optional.** The warning threshold
nwc_health_critical	**Optional.** The critical threshold
nwc_health_warningx	**Optional.** The extended warning thresholds
nwc_health_criticalx	**Optional.** The extended critical thresholds
nwc_health_mitigation	**Optional.** The parameter allows you to change a critical error to a warning (1) or ok (0).
nwc_health_selectedperfdata	**Optional.** The parameter allows you to limit the list of performance data. It's a perl regexp. Only matching perfdata show up in the output.
nwc_health_morphperfdata	**Optional.** The parameter allows you to change performance data labels. It's a perl regexp and a substitution. –morphperfdata '*(.)ISATAP(.)*'='$1patasi$2'
nwc_health_negate	**Optional.** The parameter allows you to map exit levels, such as warning=critical.
nwc_health_mymodules-dyn-dir	**Optional.** A directory where own extensions can be found.
nwc_health_servertype	**Optional.** The type of the network device: cisco (default). Use it if auto-detection is not possible.
nwc_health_statefilesdir	**Optional.** An alternate directory where the plugin can save files.
nwc_health_oids	**Optional.** A list of oids which are downloaded and written to a cache file. Use it together with –mode oidcache.

Name	Description
nwc_health_offline	**Optional.** The maximum number of seconds since the last update of cache file before it is considered too old.
nwc_health_multiline	**Optional.** Multiline output

10.7.8 Operating System

This category contains plugins which receive details about your operating system or the guest system.

10.7.8.1 mem

The check_mem.pl plugin checks the memory usage on linux and unix hosts. It is able to count cache memory as free when compared to thresholds. More details can be found on this blog entry.

Custom attributes passed as command parameters:

Name	Description
mem_used	**Optional.** Tell the plugin to check for used memory in opposite of **mem_free**. Must specify one of these as true.
mem_free	**Optional.** Tell the plugin to check for free memory in opposite of **mem_used**. Must specify one of these as true.
mem_cache	**Optional.** If set to true, plugin will count cache as free memory. Defaults to false.
mem_warning	**Required.** Specify the warning threshold as number interpreted as percent.
mem_critical	**Required.** Specify the critical threshold as number interpreted as percent.

10.7.8.2 running_kernel

The check_running_kernel plugin is provided by the `nagios-plugin-contrib` package on Debian/Ubuntu.

Custom attributes:

Name	Description
running_kernel_use_sudo	Whether to run the plugin with `sudo`. Defaults to false except on Ubuntu where it defaults to true.

10.7.8.3 iostats

The check_iostats plugin uses the `iostat` binary to monitor I/O on a Linux host. The default thresholds are rather high so you can use a grapher for baselining before setting your own.

Custom attributes passed as command parameters:

Name	Description
iostats_disk	**Required.** The device to monitor without path. e.g. sda or vda. (default: sda).
iostats_warning_tps	**Required.** Warning threshold for tps (default: 3000).
iostats_warning_read	**Required.** Warning threshold for KB/s reads (default: 50000).
iostats_warning_write	**Required.** Warning threshold for KB/s writes (default: 10000).
iostats_warning_iowait	**Required.** Warning threshold for % iowait (default: 50).
iostats_critical_tps	**Required.** Critical threshold for tps (default: 5000).
iostats_critical_read	**Required.** Critical threshold for KB/s reads (default: 80000).
iostats_critical_write	**Required.** Critical threshold for KB/s writes (default: 25000).
iostats_critical_iowait	**Required.** Critical threshold for % iowait (default: 80).

10.7.8.4 iostat

The check_iostat plugin uses the `iostat` binary to monitor disk I/O on a Linux host. The default thresholds are rather high so you can use a grapher for baselining before setting your own.

Custom attributes passed as command parameters:

Name	Description
iostat_disk	**Required.** The device to monitor without path. e.g. sda or vda. (default: sda).
iostat_wtps	**Required.** Warning threshold for tps (default: 100).
iostat_wread	**Required.** Warning threshold for KB/s reads (default: 100).

Name	Description
iostat_wwrite	**Required.** Warning threshold for KB/s writes (default: 100).
iostat_ctps	**Required.** Critical threshold for tps (default: 200).
iostat_cread	**Required.** Critical threshold for KB/s reads (default: 200).
iostat_cwrite	**Required.** Critical threshold for KB/s writes (default: 200).

10.7.8.5 yum

The check_yum plugin checks the YUM package management system for package updates. The plugin requires the `yum-plugin-security` package to differentiate between security and normal updates.

Custom attributes passed as command parameters:

Name	Description
yum_all_updates	**Optional.** Set to true to not distinguish between security and non-security updates, but returns critical for any available update. This may be used if the YUM security plugin is absent or you want to maintain every single package at the latest version. You may want to use **yum_warn_on_any_update** instead of this option. Defaults to false.
yum_warn_on_any_update	**Optional.** Set to true to warn if there are any (non-security) package updates available. Defaults to false.
yum_cache_only	**Optional.** If set to true, plugin runs entirely from cache and does not update the cache when running YUM. Useful if you have `yum makecache` cronned. Defaults to false.
yum_no_warn_on_lock	**Optional.** If set to true, returns OK instead of WARNING when YUM is locked and fails to check for updates due to another instance running. Defaults to false.
yum_no_warn_on_updates	**Optional.** If set to true, returns OK instead of WARNING even when updates are available. The plugin output still shows the number of available updates. Defaults to false.
yum_enablerepo	**Optional.** Explicitly enables a repository when calling YUM. Can take a comma separated list of repositories. Note that enabling repositories can lead to unexpected results, for example when protected repositories are enabled.
yum_disablerepo	**Optional.** Explicitly disables a repository when calling YUM. Can take a comma separated list of repositories. Note that enabling repositories can lead to unexpected results, for example when protected repositories are enabled.
yum_installroot	**Optional.** Specifies another installation root directory (for example a chroot).
yum_timeout	**Optional.** Set a timeout in seconds after which the plugin will exit (defaults to 55 seconds).

10.7.9 Storage

This category includes all plugins for various storage and object storage technologies.

10.7.9.1 glusterfs

The glusterfs plugin is used to check the GlusterFS storage health on the server. The plugin requires `sudo` permissions.

Custom attributes passed as command parameters:

Name	Description
glusterfs_perfdata	**Optional.** Print perfdata of all or the specified volume.
glusterfs_warn_on_failed_heal	**Optional.** Warn if the *heal-failed* log contains entries. The log can be cleared by restarting glusterd.
glusterfs_volume	**Optional.** Only check the specified *VOLUME*. If –volume is not set, all volumes are checked.
glusterfs_disk_warning	**Optional.** Warn if disk usage is above *DISKWARN*. Defaults to 90 (percent).
glusterfs_disk_critical	**Optional.** Return a critical error if disk usage is above *DISKCRIT*. Defaults to 95 (percent).
glusterfs_inode_warning	**Optional.** Warn if inode usage is above *DISKWARN*. Defaults to 90 (percent).
glusterfs_inode_critical	**Optional.** Return a critical error if inode usage is above *DISKCRIT*. Defaults to 95 (percent).

10.7.10 Virtualization

This category includes all plugins for various virtualization technologies.

10.7.10.1 esxi_hardware

The check_esxi_hardware.py plugin uses the pywbem Python library to monitor the hardware of ESXi servers through the VMWare API and CIM service.

Custom attributes passed as command parameters:

Name	Description
esxi_hardware_host	**Required.** Specifies the host to monitor. Defaults to "*address*".
esxi_hardware_user	**Required.** Specifies the user for polling. Must be a local user of the root group on the system. Can also be provided as a file path file:/path/to/.passwdfile, then first string of file is used.

Name	Description
esxi_hardware_password	**Required.** Password of the user. Can also be provided as a file path file:/path/to/.passwdfile, then second string of file is used.
esxi_hardware_port	**Optional.** Specifies the CIM port to connect to. Defaults to 5989.
esxi_hardware_vendor	**Optional.** Defines the vendor of the server: “auto”, “dell”, “hp”, “ibm”, “intel”, “unknown” (default).
esxi_hardware_weblink	**Optional.** Add web-links to hardware manuals for Dell servers (use your country extension). Only useful with **esxi_hardware_vendor** = dell.
esxi_hardware_ignore	**Optional.** Comma separated list of elements to ignore.
esxi_hardware_perfdata	**Optional.** Add performcedata for graphers like PNP4Nagios to the output. Defaults to false.
esxi_hardware_nopower	**Optional.** Do not collect power performance data, when **esxi_hardware_perfdata** is set to true. Defaults to false.
esxi_hardware_novolts	**Optional.** Do not collect voltage performance data, when **esxi_hardware_perfdata** is set to true. Defaults to false.
esxi_hardware_nocurrent	**Optional.** Do not collect current performance data, when **esxi_hardware_perfdata** is set to true. Defaults to false.
esxi_hardware_notemp	**Optional.** Do not collect temperature performance data, when **esxi_hardware_perfdata** is set to true. Defaults to false.
esxi_hardware_nofan	**Optional.** Do not collect fan performance data, when **esxi_hardware_perfdata** is set to true. Defaults to false.
esxi_hardware_nolcd	**Optional.** Do not collect lcd/display status data. Defaults to false.

10.7.10.2 VMware

Check commands for the check_vmware_esx plugin.

vmware-esx-dc-volumes

Check command object for the `check_vmware_esx` plugin. Shows all datastore volumes info.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_subselect	**Optional.** Volume name to be checked the free space.
vmware_gigabyte	**Optional.** Output in GB instead of MB.
vmware_usedspace	**Optional.** Output used space instead of free. Defaults to “false”.
vmware_alertonly	**Optional.** List only alerting volumes. Defaults to “false”.
vmware_exclude	**Optional.** Blacklist volumes name. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist volumes name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_dc_volume_used	**Optional.** Output used space instead of free. Defaults to "true".
vmware_warn	**Optional.** The warning threshold for volumes. Defaults to "80%".
vmware_crit	**Optional.** The critical threshold for volumes. Defaults to "90%".

vmware-esx-dc-runtime-info

Check command object for the `check_vmware_esx` plugin. Shows all runtime info for the datacenter/Vcenter.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-dc-runtime-listvms

Check command object for the `check_vmware_esx` plugin. List of vmware machines and their power state. BEWARE!! In larger environments systems can cause trouble displaying the informations needed due to the mass of data. Use **vmware_alertonly** to avoid this.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_alertonly	**Optional.** List only alerting VMs. Important here to avoid masses of data.
vmware_exclude	**Optional.** Blacklist VMs name. No value defined as default.
vmware_include	**Optional.** Whitelist VMs name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

Name	Description
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-dc-runtime-listhost

Check command object for the `check_vmware_esx` plugin. List of VMware ESX hosts and their power state.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_alertonly	**Optional.** List only alerting hosts. Important here to avoid masses of data.
vmware_exclude	**Optional.** Blacklist VMware ESX hosts. No value defined as default.
vmware_include	**Optional.** Whitelist VMware ESX hosts. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

Name	Description
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-dc-runtime-listcluster

Check command object for the `check_vmware_esx` plugin. List of VMware clusters and their states.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_alertonly	**Optional.** List only alerting hosts. Important here to avoid masses of data.
vmware_exclude	**Optional.** Blacklist VMware cluster. No value defined as default.
vmware_include	**Optional.** Whitelist VMware cluster. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

Name	Description
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-dc-runtime-issues

Check command object for the `check_vmware_esx` plugin. All issues for the host.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist issues. No value defined as default.
vmware_include	**Optional.** Whitelist issues. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-dc-runtime-status

Check command object for the `check_vmware_esx` plugin. Overall object status (gray/green/red/yellow).

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-dc-runtime-tools

Check command object for the `check_vmware_esx` plugin. Vmware Tools status.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Required.** Datacenter/vCenter hostname.
vmware_cluster	**Optional.** ESX or ESXi clustername.
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.

Name	Description
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to “/var/spool/icinga2/tmp”.
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to “false”.
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username’s password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_poweredonly	**Optional.** List only VMs which are powered on. No value defined as default.
vmware_alertonly	**Optional.** List only alerting VMs. Important here to avoid masses of data.
vmware_exclude	**Optional.** Blacklist VMs. No value defined as default.
vmware_include	**Optional.** Whitelist VMs. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

Name	Description
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.
vmware_openvmtools	**Optional** Prevent CRITICAL state for installed and running Open VM Tools.

vmware-esx-soap-host-check

Check command object for the `check_vmware_esx` plugin. Simple check to verify a successful connection to VMware SOAP API.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-uptime

Check command object for the `check_vmware_esx` plugin. Displays uptime of the VMware host.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-cpu

Check command object for the `check_vmware_esx` plugin. CPU usage in percentage.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. Defaults to "80%".
vmware_crit	**Optional.** The critical threshold in percent. Defaults to "90%".

vmware-esx-soap-host-cpu-ready

Check command object for the `check_vmware_esx` plugin. Percentage of time that the virtual machine was ready, but could not get scheduled to run on the physical CPU. CPU ready time is dependent on the number of virtual machines on the host and their CPU loads. High or growing ready time can be a hint CPU bottlenecks.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-cpu-wait

Check command object for the `check_vmware_esx` plugin. CPU time spent in wait state. The wait total includes time spent the CPU idle, CPU swap wait, and CPU I/O wait states. High or growing wait time can be a hint I/O bottlenecks.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-cpu-usage

Check command object for the `check_vmware_esx` plugin. Actively used CPU of the host, as a percentage of the total available CPU. Active CPU is approximately equal to the ratio of the used CPU to the available CPU.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. Defaults to "80%".

Name	Description
vmware_crit	**Optional.** The critical threshold in percent. Defaults to "90%".

vmware-esx-soap-host-mem

Check command object for the `check_vmware_esx` plugin. All mem info(except overall and no thresholds).

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-mem-usage

Check command object for the `check_vmware_esx` plugin. Average mem usage in percentage.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. Defaults to "80%".
vmware_crit	**Optional.** The critical threshold in percent. Defaults to "90%".

vmware-esx-soap-host-mem-consumed

Check command object for the `check_vmware_esx` plugin. Amount of machine memory used on the host. Consumed memory includes Includes memory used by the Service Console, the VMkernel vSphere services, plus the total consumed metrics for all running virtual machines in MB.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold in percent. No value defined as default.

vmware-esx-soap-host-mem-swapused

Check command object for the `check_vmware_esx` plugin. Amount of memory that is used by swap. Sum of memory swapped of all powered on VMs and vSphere services on the host in MB. In case of an error all VMs with their swap used will be displayed.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. No value defined as default.
vmware_crit	**Optional.** The critical threshold in percent. No value defined as default.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-mem-overhead

Check command object for the `check_vmware_esx` plugin. Additional mem

used by VM Server in MB.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Auhentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold in percent. No value defined as default.

vmware-esx-soap-host-mem-memctl

Check command object for the `check_vmware_esx` plugin. The sum of all vmmemctl values in MB for all powered-on virtual machines, plus vSphere services on the host. If the balloon target value is greater than the balloon value, the VMkernel inflates the balloon, causing more virtual machine memory to be reclaimed. If the balloon target value is less than the balloon value, the VMkernel deflates the balloon, which allows the virtual machine to consume additional memory if needed (used by VM memory control driver). In case of an error all VMs with their vmmemctl values will be displayed.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in percent. No value defined as default.
vmware_crit	**Optional.** The critical threshold in percent. No value defined as default.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-net

Check command object for the `check_vmware_esx` plugin. Shows net info.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist NICs. No value defined as default.

Name	Description
vmware_isregexp	**Optional.** Treat blacklist expression as regexp.

vmware-esx-soap-host-net-usage

Check command object for the `check_vmware_esx` plugin. Overall network usage in KBps(Kilobytes per Second).

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in KBps(Kilobytes per Second). No value defined as default.
vmware_crit	**Optional.** The critical threshold in KBps(Kilobytes per Second). No value defined as default.

vmware-esx-soap-host-net-receive

Check command object for the `check_vmware_esx` plugin. Data receive in KBps(Kilobytes per Second).

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

Name	Description
vmware_warn	**Optional.** The warning threshold in KBps(Kilobytes per Second). No value defined as default.
vmware_crit	**Optional.** The critical threshold in KBps(Kilobytes per Second). No value defined as default.

vmware-esx-soap-host-net-send

Check command object for the `check_vmware_esx` plugin. Data send in KBps(Kilobytes per Second).

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold in KBps(Kilobytes per Second). No value defined as default.
vmware_crit	**Optional.** The critical threshold in KBps(Kilobytes per Second). No value defined as default.

vmware-esx-soap-host-net-nic

Check command object for the `check_vmware_esx` plugin. Check all active NICs.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist NICs. No value defined as default.

Name	Description
vmware_isregexp	**Optional.** Treat blacklist expression as regexp.

vmware-esx-soap-host-volumes

Check command object for the `check_vmware_esx` plugin. Shows all datastore volumes info.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_subselect	**Optional.** Volume name to be checked the free space.
vmware_gigabyte	**Optional.** Output in GB instead of MB.
vmware_usedspace	**Optional.** Output used space instead of free. Defaults to “false”.
vmware_alertonly	**Optional.** List only alerting volumes. Defaults to “false”.
vmware_exclude	**Optional.** Blacklist volumes name. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist volumes name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_warn	**Optional.** The warning threshold for volumes. Defaults to "80%".
vmware_crit	**Optional.** The critical threshold for volumes. Defaults to "90%".
vmware_spaceleft	**Optional.** This has to be used in conjunction with thresholds as mentioned above.

vmware-esx-soap-host-io

Check command object for the `check_vmware_esx` plugin. Shows all disk io info. Without subselect no thresholds can be given. All I/O values are aggregated from historical intervals over the past 24 hours with a 5 minute sample rate.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-io-aborted

Check command object for the `check_vmware_esx` plugin. Number of aborted SCSI commands.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-resets

Check command object for the `check_vmware_esx` plugin. Number of SCSI bus resets.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-read

Check command object for the `check_vmware_esx` plugin. Average number of kilobytes read from the disk each second.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-read-latency

Check command object for the `check_vmware_esx` plugin. Average amount of time (ms) to process a SCSI read command issued from the Guest OS to the virtual machine.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-write

Check command object for the `check_vmware_esx` plugin. Average number of kilobytes written to disk each second.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-write-latency

Check command object for the `check_vmware_esx` plugin. Average amount of time (ms) taken to process a SCSI write command issued by the Guest OS to the virtual machine.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-usage

Check command object for the `check_vmware_esx` plugin. Aggregated disk I/O rate. For hosts, this metric includes the rates for all virtual machines running on the host.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-kernel-latency

Check command object for the `check_vmware_esx` plugin. Average amount of time (ms) spent by VMkernel processing each SCSI command.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-device-latency

Check command object for the `check_vmware_esx` plugin. Average amount of time (ms) to complete a SCSI command from the physical device.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-queue-latency

Check command object for the `check_vmware_esx` plugin. Average amount of time (ms) spent in the VMkernel queue.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-io-total-latency

Check command object for the `check_vmware_esx` plugin. Average amount of time (ms) taken during the collection interval to process a SCSI command issued by the guest OS to the virtual machine. The sum of kernelWriteLatency and deviceWriteLatency.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-host-media

Check command object for the `check_vmware_esx` plugin. List vm's with attached host mounted media like cd,dvd or floppy drives. This is important for monitoring because a virtual machine with a mount cd or dvd drive can not be moved to another host.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist VMs name. No value defined as default.
vmware_include	**Optional.** Whitelist VMs name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-service

Check command object for the `check_vmware_esx` plugin. Shows host service info.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist services name. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist services name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-runtime

Check command object for the `check_vmware_esx` plugin. Shows runtime info: VMs, overall status, connection state, health, storagehealth, temperature and sensor are represented as one value and without thresholds.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-runtime-con

Check command object for the `check_vmware_esx` plugin. Shows connection state.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-runtime-listvms

Check command object for the `check_vmware_esx` plugin. List of VMware machines and their status.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist VMs name. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist VMs name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-runtime-status

Check command object for the `check_vmware_esx` plugin. Overall object status (gray/green/red/yellow).

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-host-runtime-health

Check command object for the `check_vmware_esx` plugin. Checks cpu/storage/memory/sensor status.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist status name. No value defined as default.
vmware_include	**Optional.** Whitelist status name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

vmware-esx-soap-host-runtime-health-listsensors

Check command object for the `check_vmware_esx` plugin. List all available sensors(use for listing purpose only).

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist status name. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist status name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

vmware-esx-soap-host-runtime-health-nostoragestatus

Check command object for the `check_vmware_esx` plugin. This is to avoid a double alarm if you use **vmware-esx-soap-host-runtime-health** and **vmware-esx-soap-host-runtime-storagehealth**.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist status name. No value defined as default.
vmware_include	**Optional.** Whitelist status name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

vmware-esx-soap-host-runtime-storagehealth

Check command object for the `check_vmware_esx` plugin. Local storage status check.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist storage name. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist storage name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-runtime-temp

Check command object for the `check_vmware_esx` plugin. Lists all temperature sensors.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist sensor name. No value defined as default.
vmware_include	**Optional.** Whitelist sensor name. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

Name	Description
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-runtime-issues

Check command object for the `check_vmware_esx` plugin. Lists all configuration issues for the host.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist configuration issues. No value defined as default.
vmware_include	**Optional.** Whitelist configuration issues. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

Name	Description
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-storage

Check command object for the `check_vmware_esx` plugin. Shows Host storage info.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist adapters, luns and paths. No value defined as default.
vmware_include	**Optional.** Whitelist adapters, luns and paths. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.

vmware-esx-soap-host-storage-adapter

Check command object for the `check_vmware_esx` plugin. List host bus adapters.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist adapters. No value defined as default.

Name	Description
vmware_include	**Optional.** Whitelist adapters. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-storage-lun

Check command object for the `check_vmware_esx` plugin. List SCSI logical units. The listing will include: LUN, canonical name of the disc, all of displayed name which is not part of the canonical name and status.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_exclude	**Optional.** Blacklist luns. No value defined as default.
vmware_include	**Optional.** Whitelist luns. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

vmware-esx-soap-host-storage-path

Check command object for the `check_vmware_esx` plugin. List multipaths and the associated paths.

Custom attributes passed as command parameters:

Name	Description
vmware_host	**Required.** ESX or ESXi hostname.
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. In case the check is done through a Datacenter/vCenter host.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

Name	Description
vmware_alertonly	**Optional.** List only alerting units. Important here to avoid masses of data. Defaults to “false”.
vmware_exclude	**Optional.** Blacklist paths. No value defined as default.
vmware_include	**Optional.** Whitelist paths. No value defined as default.
vmware_isregexp	**Optional.** Treat blacklist and whitelist expressions as regexp.
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

Name	Description
vmware_standbyok	**Optional.** For storage systems where a standby multipath is ok and not a warning. Defaults to false.

vmware-esx-soap-vm-cpu

Check command object for the `check_vmware_esx` plugin. Shows all CPU usage info.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-cpu-ready

Check command object for the `check_vmware_esx` plugin. Percentage of time that the virtual machine was ready, but could not get scheduled to run on the physical CPU.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.

Name	Description
vmware_sslport	**Optional.** SSL port connection. Defaults to “443”.
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.

Name	Description
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-cpu-wait

Check command object for the `check_vmware_esx` plugin. CPU time spent in wait state. The wait total includes time spent the CPU idle, CPU swap wait, and CPU I/O wait states. High or growing wait time can be a hint I/O bottlenecks.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-cpu-usage

Check command object for the `check_vmware_esx` plugin. Amount of actively used virtual CPU, as a percentage of total available CPU. This is the host's view of the CPU usage, not the guest operating system view. It is the average CPU utilization over all available virtual CPUs in the virtual machine.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** Warning threshold in percent. Defaults to "80%".
vmware_crit	**Optional.** Critical threshold in percent. Defaults to "90%".

vmware-esx-soap-vm-mem

Check command object for the `check_vmware_esx` plugin. Shows all memory info, except overall.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-mem-usage

Check command object for the `check_vmware_esx` plugin. Average mem usage in percentage of configured virtual machine "physical" memory.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** Warning threshold in percent. Defaults to "80%".

Name	Description
vmware_crit	**Optional.** Critical threshold in percent. Defaults to "90%".

vmware-esx-soap-vm-mem-consumed

Check command object for the `check_vmware_esx` plugin. Amount of guest physical memory in MB consumed by the virtual machine for guest memory. Consumed memory does not include overhead memory. It includes shared memory and memory that might be reserved, but not actually used. Use this metric for charge-back purposes. **vm consumed memory = memory granted – memory saved**

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-mem-memctl

Check command object for the `check_vmware_esx` plugin. Amount of guest physical memory that is currently reclaimed from the virtual machine through ballooning. This is the amount of guest physical memory that has been allocated and pinned by the balloon driver.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-net

Check command object for the `check_vmware_esx` plugin. Shows net info.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-net-usage

Check command object for the `check_vmware_esx` plugin. Overall network usage in KBps(Kilobytes per Second).

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to “false”.
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to “false”.
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to “90”.
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-net-receive

Check command object for the `check_vmware_esx` plugin. Receive in KBps(Kilobytes per Second).

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-net-send

Check command object for the `check_vmware_esx` plugin. Send in KBps(Kilobytes per Second).

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-io

Check command object for the `check_vmware_esx` plugin. Shows all disk io info. Without subselect no thresholds can be given. All I/O values are aggregated from historical intervals over the past 24 hours with a 5 minute sample rate.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-io-read

Check command object for the `check_vmware_esx` plugin. Average number of kilobytes read from the disk each second.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session - IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-io-write

Check command object for the `check_vmware_esx` plugin. Average number of kilobytes written to disk each second.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-io-usage

Check command object for the `check_vmware_esx` plugin. Aggregated disk I/O rate.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.

Name	Description
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-runtime

Check command object for the `check_vmware_esx` plugin. Shows virtual machine runtime info.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-runtime-con

Check command object for the `check_vmware_esx` plugin. Shows the connection state.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-runtime-powerstate

Check command object for the `check_vmware_esx` plugin. Shows virtual machine power state: poweredOn, poweredOff or suspended.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-runtime-status

Check command object for the `check_vmware_esx` plugin. Overall object status (gray/green/red/yellow).

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-runtime-consoleconnections

Check command object for the `check_vmware_esx` plugin. Console connections to virtual machine.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_warn	**Optional.** The warning threshold. No value defined as default.
vmware_crit	**Optional.** The critical threshold. No value defined as default.

vmware-esx-soap-vm-runtime-gueststate

Check command object for the `check_vmware_esx` plugin. Guest OS status. Needs VMware Tools installed and running.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.

Name	Description
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

vmware-esx-soap-vm-runtime-tools

Check command object for the `check_vmware_esx` plugin. Guest OS status. VMware tools status.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".

Name	Description
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".

Name	Description
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd

Name	Description
vmware_openvmtools	**Optional** Prevent CRITICAL state for installed and running Open VM Tools.

vmware-esx-soap-vm-runtime-issues

Check command object for the `check_vmware_esx` plugin. All issues for the virtual machine.

Custom attributes passed as command parameters:

Name	Description
vmware_datacenter	**Optional.** Datacenter/vCenter hostname. Conflicts with **vmware_host**.
vmware_host	**Optional.** ESX or ESXi hostname. Conflicts with **vmware_datacenter**.
vmware_vmname	**Required.** Virtual machine name.
vmware_sslport	**Optional.** SSL port connection. Defaults to "443".

Name	Description
vmware_ignoreunknown	**Optional.** Sometimes 3 (unknown) is returned from a component. But the check itself is ok. With this option the plugin will return OK (0) instead of UNKNOWN (3). Defaults to "false".
vmware_ignorewarning	**Optional.** Sometimes 2 (warning) is returned from a component. But the check itself is ok (from an operator view). With this option the plugin will return OK (0) instead of WARNING (1). Defaults to "false".
vmware_timeout	**Optional.** Seconds before plugin times out. Defaults to "90".
vmware_trace	**Optional.** Set verbosity level of vSphere API request/respond trace.

Name	Description
vmware_sessionfile	**Optional.** Session file name enhancement.
vmware_sessionfiledir	**Optional.** Path to store the **vmware_sessionfile** file. Defaults to "/var/spool/icinga2/tmp".
vmware_nosession	**Optional.** No auth session – IT SHOULD BE USED FOR TESTING PURPOSES ONLY!. Defaults to "false".
vmware_username	**Optional.** The username to connect to Host or vCenter server. No value defined as default.
vmware_password	**Optional.** The username's password. No value defined as default.

Name	Description
vmware_authfile	**Optional.** Use auth file instead username/password to session connect. No effect if **vmware_username** and **vmware_password** are defined **Authentication file content:** username=vmuser password=p@ssw0rd
vmware_multiline	**Optional.** Multiline output in overview. This mean technically that a multiline output uses a HTML ** ** for the GUI. No value defined as default.

10.7.11 Web

This category includes all plugins for web-based checks.

10.7.11.1 apache_status

The check_apache_status.pl plugin uses the /server-status HTTP endpoint to monitor status metrics for the Apache webserver.

Custom attributes passed as command parameters:

Name	Description
apache_status_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, `address6` otherwise.

Name	Description
apache_status_port	**Optional.** the http port.
apache_status_url	**Optional.** URL to use, instead of the default (http://`apache_status_address`/server-status).
apache_status_ssl	**Optional.** set to use ssl connection
apache_status_timeout	**Optional.** timeout in seconds
apache_status_warning	**Optional.** Warning threshold (number of open slots, busy workers and idle workers that will cause a WARNING) like ':20,50,:50'.
apache_status_critical	**Optional.** Critical threshold (number of open slots, busy workers and idle workers that will cause a CRITICAL) like ':10,25,:20'.

10.7.12 cert

The check_ssl_cert plugin uses the openssl binary (and optional curl) to check a X.509 certificate.

Custom attributes passed as command parameters:

Name	Description
ssl_cert_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
ssl_cert_port	**Optional.** TCP port number (default: 443).
ssl_cert_file	**Optional.** Local file path. Works only if `ssl_cert_address` is set to "localhost".
ssl_cert_warn	**Optional.** Minimum number of days a certificate has to be valid.

Name	Description
ssl_cert_critical	**Optional.** Minimum number of days a certificate has to be valid to issue a critical status.
ssl_cert_cn	**Optional.** Pattern to match the CN of the certificate.
ssl_cert_altnames	**Optional.** Matches the pattern specified in -n with alternate
ssl_cert_issuer	**Optional.** Pattern to match the issuer of the certificate.
ssl_cert_org	**Optional.** Pattern to match the organization of the certificate.
ssl_cert_email	**Optional.** Pattern to match the email address contained in the certificate.
ssl_cert_serial	**Optional.** Pattern to match the serial number.
ssl_cert_noauth	**Optional.** Ignore authority warnings (expiration only)

Name	Description
ssl_cert_match_host	**Optional.** Match CN with the host name.
ssl_cert_selfsigned	**Optional.** Allow self-signed certificate.
ssl_cert_sni	**Optional.** Sets the TLS SNI (Server Name Indication) extension.
ssl_cert_timeout	**Optional.** Seconds before connection times out (default: 15)
ssl_cert_protocol	**Optional.** Use the specific protocol {http,smtp,pop3,imap,ftp,xmpp,irc,ldap} (default: http).
ssl_cert_clientcert	**Optional.** Use client certificate to authenticate.
ssl_cert_clientpass	**Optional.** Set passphrase for client certificate.
ssl_cert_ssllabs	**Optional.** SSL Labs assessment
ssl_cert_ssllabs_nocache	**Optional.** Forces a new check by SSL Labs
ssl_cert_rootcert	**Optional.** Root certificate or directory to be used for certificate validation.

Name	Description
ssl_cert_ignore_signature	**Optional.** Do not check if the certificate was signed with SHA1 od MD5.
ssl_cert_ssl_version	**Optional.** Force specific SSL version out of {ssl2,ssl3,tls1,tls1_1,tls1_2}.
ssl_cert_disable_ssl_versions	**Optional.** Disable specific SSL versions out of {ssl2,ssl3,tls1,tls1_1,tls1_2}. Multiple versions can be given as array.
ssl_cert_cipher	**Optional.** Cipher selection: force {ecdsa,rsa} authentication.
ssl_cert_ignore_expiration	**Optional.** Ignore expiration date.
ssl_cert_ignore_ocsp	**Optional.** Do not check revocation with OCSP.

10.7.12.1 jmx4perl

The check_jmx4perl plugin uses the HTTP API exposed by the Jolokia web application and queries Java message beans on an application server. It is part of the `JMX::Jmx4Perl` Perl module which includes detailed documentation.

Custom attributes passed as command parameters:

Name	Description
jmx4perl_url	**Required.** URL to agent web application. Defaults to "http://*address*:8080/jolokia".
jmx4perl_product	**Optional.** Name of app server product (e.g. jboss), by default is uses an auto detection facility.

Name	Description
jmx4perl_alias	**Optional.** Alias name for attribute (e.g. MEMORY_HEAP_USED). All available aliases can be viewed by executing `jmx4perl aliases` on the command line.
jmx4perl_mbean	**Optional.** MBean name (e.g. java.lang:type=Memory).
jmx4perl_attribute	**Optional.** Attribute name (e.g. HeapMemoryUsage).
jmx4perl_operation	**Optional.** Operation to execute.
jmx4perl_value	**Optional.** Shortcut for specifying mbean/attribute/path. Slashes within names must be escaped with backslash.
jmx4perl_delta	**Optional.** Switches on incremental mode. Optional argument are seconds used for normalizing.
jmx4perl_path	**Optional.** Inner path for extracting a single value from a complex attribute or return value (e.g. used).
jmx4perl_target	**Optional.** JSR-160 Service URL specifing the target server.
jmx4perl_target_user	**Optional.** Username to use for JSR-160 connection.
jmx4perl_target_password	**Optional.** Password to use for JSR-160 connection.
jmx4perl_proxy	**Optional.** Proxy to use.
jmx4perl_user	**Optional.** User for HTTP authentication.
jmx4perl_password	**Optional.** Password for HTTP authentication.
jmx4perl_name	**Optional.** Name to use for output, by default a standard value based on the MBean and attribute will be used.
jmx4perl_method	**Optional.** HTTP method to use, either get or post. By default a method is determined automatically based on the request type.
jmx4perl_base	**Optional.** Base name, which when given, interprets critical and warning values as relative in the range 0 .. 100%. Must be given in the form mbean/attribute/path.
jmx4perl_base_mbean	**Optional.** Base MBean name, interprets critical and warning values as relative in the range 0 .. 100%. Requires "jmx4perl_base_attribute".
jmx4perl_base_attribute	**Optional.** Base attribute for a relative check. Requires "jmx4perl_base_mbean".
jmx4perl_base_path	**Optional.** Base path for relative checks, where this path is used on the base attribute's value.
jmx4perl_unit	**Optional.** Unit of measurement of the data retrieved. Recognized values are [B|KB|MN|GB|TB] for memory values and [us|ms|s|m|h|d] for time values.
jmx4perl_null	**Optional.** Value which should be used in case of a null return value of an operation or attribute. Defaults to null.
jmx4perl_string	**Optional.** Force string comparison for critical and warning checks. Defaults to false.
jmx4perl_numeric	**Optional.** Force numeric comparison for critical and warning checks. Defaults to false.
jmx4perl_critical	**Optional.** Critical threshold for value.
jmx4perl_warning	**Optional.** Warning threshold for value.

Name	Description
jmx4perl_label	**Optional.** Label to be used for printing out the result of the check. For placeholders which can be used see the documentation.
jmx4perl_perfdata	**Optional.** Whether performance data should be omitted, which are included by default. Defaults to "on" for numeric values, to "off" for strings.
jmx4perl_unknown_is_critical	**Optional.** Map UNKNOWN errors to errors with a CRITICAL status. Defaults to false.
jmx4perl_timeout	**Optional.** Seconds before plugin times out. Defaults to "15".
jmx4perl_config	**Optional.** Path to configuration file.
jmx4perl_server	**Optional.** Symbolic name of server url to use, which needs to be configured in the configuration file.
jmx4perl_check	**Optional.** Name of a check configuration as defined in the configuration file, use array if you need arguments.

10.7.12.2 kdc

The check_kdc plugin uses the Kerberos `kinit` binary to monitor Kerberos 5 KDC by acquiring a ticket.

Custom attributes passed as command parameters:

Name	Description
kdc_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, `address6` otherwise.
kdc_port	**Optional** Port on which KDC runs (default 88).
kdc_principal	**Required** Principal name to authenticate as (including realm).
kdc_keytab	**Required** Keytab file containing principal's key.

10.7.12.3 nginx_status

The check_nginx_status.pl plugin uses the /nginx_status HTTP endpoint which provides metrics for monitoring Nginx.

Custom attributes passed as command parameters:

Name	Description
nginx_status_host_address	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, `address6` otherwise.
nginx_status_port	**Optional.** the http port.

Name	Description
nginx_status_url	**Optional.** URL to use, instead of the default (http://`nginx_status_hostname`/nginx_status).
nginx_status_servername	**Optional.** ServerName to use if you specified an IP to match the good Virtualhost in your target.
nginx_status_ssl	**Optional.** set to use ssl connection.
nginx_status_disable_sslverify	**Optional.** set to disable SSL hostname verification.
nginx_status_user	**Optional.** Username for basic auth.
nginx_status_pass	**Optional.** Password for basic auth.
nginx_status_realm	**Optional.** Realm for basic auth.
nginx_status_maxreach	**Optional.** Number of max processes reached (since last check) that should trigger an alert.
nginx_status_timeout	**Optional.** timeout in seconds.
nginx_status_warn	**Optional.** Warning threshold (number of active connections, ReqPerSec or ConnPerSec that will cause a WARNING) like '10000,100,200'.
nginx_status_critical	**Optional.** Critical threshold (number of active connections, ReqPerSec or ConnPerSec that will cause a CRITICAL) like '20000,200,300'.

10.7.12.4 rbl

The check_rbl plugin uses the `Net::DNS` Perl library to check whether your SMTP server is blacklisted.

Custom attributes passed as command parameters:

Name	Description
rbl_hostname	**Optional.** The address or name of the SMTP server to check. Defaults to "*address*" if the host's `address` attribute is set, `address6` otherwise.
rbl_server	**Required** List of RBL servers as an array.
rbl_warning	**Optional** Number of blacklisting servers for a warning.
rbl_critical	**Optional** Number of blacklisting servers for a critical.
tbl_timeout	**Optional** Seconds before plugin times out (default: 15).

10.7.12.5 squid

The check_squid plugin uses the `squidclient` binary to monitor a Squid proxy.

Custom attributes passed as command parameters:

Name	Description
squid_hostname	**Optional.** The host's address. Defaults to "*address*" if the host's `address` attribute is set, "*address*6" otherwise.
squid_data	**Optional.** Data to fetch (default: Connections) available data: Connections Cache Resources Memory FileDescriptors.
squid_port	**Optional.** Port number (default: 3128).
squid_user	**Optional.** WWW user.
squid_password	**Optional.** WWW password.
squid_warning	**Optional.** Warning threshold. See http://nagiosplug.sourceforge.net/developer-guidelines.html#THRESHOLDFORMAT for the threshold format.
squid_critical	**Optional.** Critical threshold. See http://nagiosplug.sourceforge.net/developer-guidelines.html#THRESHOLDFORMAT for the threshold format.
squid_client	**Optional.** Path of squidclient (default: /usr/bin/squidclient).
squid_timeout	**Optional.** Seconds before plugin times out (default: 15).

10.7.12.6 webinject

The check_webinject plugin uses WebInject to test web applications and web services in an automated fashion. It can be used to test individual system components that have HTTP interfaces (JSP, ASP, CGI, PHP, AJAX, Servlets, HTML Forms, XML/SOAP Web Services, REST, etc), and can be used as a test harness to create a suite of HTTP level automated functional, acceptance, and regression tests. A test harness allows you to run many test cases and collect/report your results. WebInject offers real-time results display and may also be used for monitoring system response times.

Custom attributes passed as command parameters:

Name	Description
webinject_config_file	**Optional.** There is a configuration file named 'config.xml' that is used to store configuration settings for your project. You can use this to specify which test case files to run and to set some constants and settings to be used by WebInject.

Name	Description
webinject_output	**Optional.** This option is followed by a directory name or a prefix to prepended to the output files. This is used to specify the location for writing output files (http.log, results.html, and results.xml). If a directory name is supplied (use either an absolute or relative path and make sure to add the trailing slash), all output files are written to this directory. If the trailing slash is omitted, it is assumed to a prefix and this will be prepended to the output files. You may also use a combination of a directory and prefix.

Name	Description
webinject_no_output	**Optional.** Suppresses all output to STDOUT except the results summary.
webinject_timeout	**Optional.** The value [given in seconds] will be compared to the global time elapsed to run all the tests. If the tests have all been successful, but have taken more time than the 'globaltimeout' value, a warning message is sent back to Icinga.
webinject_report_type	**Optional.** This setting is used to enable output formatting that is compatible for use with specific external programs. The available values you can set this to are: nagios, mrtg, external and standard.

Name	Description
webinject_testcase_file	**Optional.** When you launch WebInject in console mode, you can optionally supply an argument for a testcase file to run. It will look for this file in the directory that webinject.pl resides in. If no filename is passed from the command line, it will look in config.xml for testcasefile declarations. If no files are specified, it will look for a default file named 'testcases.xml' in the current [webinject] directory. If none of these are found, the engine will stop and give you an error.

11 Icinga 2 CLI Commands

Icinga 2 comes with a number of CLI commands which support bash autocompletion.

These CLI commands will allow you to use certain functionality provided by and around Icinga 2.

Each CLI command provides its own help and usage information, so please make sure to always run them with the `--help` parameter.

Run `icinga2` without any arguments to get a list of all available global options.

```
# icinga2
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * api setup (setup for API)
  * api user (API user creation helper)
  * ca list (lists all certificate signing requests)
  * ca sign (signs an outstanding certificate request)
  * console (Icinga console)
  * daemon (starts Icinga 2)
  * feature disable (disables specified feature)
  * feature enable (enables specified feature)
  * feature list (lists all available features)
  * node setup (set up node)
  * node wizard (wizard for node setup)
  * object list (lists all objects)
  * pki new-ca (sets up a new CA)
  * pki new-cert (creates a new CSR)
  * pki request (requests a certificate)
  * pki save-cert (saves another Icinga 2 instance's certificate)
  * pki sign-csr (signs a CSR)
  * pki ticket (generates a ticket)
  * troubleshoot (collect information for troubleshooting)
  * variable get (gets a variable)
  * variable list (lists all variables)

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
 --color              use VT100 color codes even when stdout is not a
                           terminal
  -D [ --define ] arg       define a constant
 -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
 -x [ --log-level ] arg   specify the log level for the console log.
```

```
                          The valid value is either debug, notice,
                        information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.1 Icinga 2 CLI Bash Autocompletion

Bash Auto-Completion (pressing `<TAB>`) is provided only for the corresponding context.

While `--config` suggests and auto-completes files and directories on disk, `feature enable` only suggests disabled features.

RPM and Debian packages install the bash completion files into `/etc/bash_completion.d/icinga2`.

You need to install the `bash-completion` package if not already installed.

RHEL/CentOS/Fedora:

```
# yum install bash-completion
```

SUSE:

```
# zypper install bash-completion
```

Debian/Ubuntu:

```
# apt-get install bash-completion
```

Ensure that the `bash-completion.d` directory is added to your shell environment. You can manually source the icinga2 bash-completion file into your current session and test it:

```
# source /etc/bash-completion.d/icinga2
```

11.2 Icinga 2 CLI Global Options

11.2.1 Application Type

By default the `icinga2` binary loads the `icinga` library. A different application type can be specified with the `--app` command-line option. Note: This is not needed by the average Icinga user, only developers.

11.2.2 Libraries

Instead of loading libraries using the `library` config directive you can also use the `--library` command-line option. Note: This is not needed by the average Icinga user, only developers.

11.2.3 Constants

Global constants can be set using the `--define` command-line option.

11.2.4 Config Include Path

When including files you can specify that the include search path should be checked. You can do this by putting your configuration file name in angle brackets like this:

```
include <test.conf>
```

This causes Icinga 2 to search its include path for the configuration file `test.conf`. By default the installation path for the Icinga Template Library is the only search directory.

Using the `--include` command-line option additional search directories can be added.

11.3 CLI command: Api

Provides the helper functions `api setup` and `api user`. The first to enable the REST API, the second to create ApiUser objects with hashed password strings. More details in the Icinga 2 API chapter.

```
# icinga2 api --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * api setup (setup for API)
  * api user (API user creation helper)

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
 --color             use VT100 color codes even when stdout is not a
```

```
                          terminal
  -D [ --define ] arg       define a constant
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                           The valid value is either debug, notice,
                         information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.4 CLI command: Ca

List and manage incoming certificate signing requests. More details can be found in the signing methods chapter. This CLI command is available since v2.8.

```
# icinga2 ca --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * ca list (lists all certificate signing requests)
  * ca sign (signs an outstanding certificate request)

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
  --color             use VT100 color codes even when stdout is not a
                          terminal
  -D [ --define ] arg       define a constant
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                           The valid value is either debug, notice,
                         information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.5 CLI command: Console

The CLI command `console` can be used to debug and evaluate Icinga 2 config expressions, e.g. to test functions in your local sandbox.

```
$ icinga2 console
Icinga 2 (version: v2.8.0)
<1> => function test(name) {
<1> ..   log("Hello " + name)
<1> .. }
null
<2> => test("World")
information/config: Hello World
null
<3> =>
```

Further usage examples can be found in the library reference chapter.

```
# icinga2 console --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 console [<arguments>]

Interprets Icinga script expressions.

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
 --color             use VT100 color codes even when stdout is not a
                           terminal
  -D [ --define ] arg       define a constant
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                            The valid value is either debug, notice,
                         information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Command options:
  -c [ --connect ] arg      connect to an Icinga 2 instance
  -e [ --eval ] arg         evaluate expression and terminate
  -r [ --file ] arg         evaluate a file and terminate
 --syntax-only         only validate syntax (requires --eval or --file)
  --sandbox                 enable sandbox mode
```

```
Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

On operating systems without the `libedit` library installed there is no support for line-editing or a command history. However you can use the `rlwrap` program if you require those features:

```
$ rlwrap icinga2 console
```

The debug console can be used to connect to a running Icinga 2 instance using the REST API. API permissions are required for executing config expressions and auto-completion.

Note

The debug console does not currently support SSL certificate verification.

Runtime modifications are not validated and might cause the Icinga 2 daemon to crash or behave in an unexpected way. Use these runtime changes at your own risk and rather *inspect and debug objects read-only*.

You can specify the API URL using the `--connect` parameter.

Although the password can be specified there process arguments on UNIX platforms are usually visible to other users (e.g. through `ps`). In order to securely specify the user credentials the debug console supports two environment variables:

Environment variable	Description
ICINGA2_API_USERNAME	The API username.
ICINGA2_API_PASSWORD	The API password.

Here's an example:

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/'
Icinga 2 (version: v2.8.0)
<1> =>
```

Once connected you can inspect variables and execute other expressions by entering them at the prompt:

```
<1> => var h = get_host("icinga2-client1.localdomain")
null
<2> => h.last_check_result
{
        active = true
        check_source = "icinga2-client1.localdomain"
     command = [ "/usr/local/sbin/check_ping", "-H", "127.0.0.1", "-c", "5000,100%", "-w", "30
```

```
        execution_end = 1446653527.174983
        execution_start = 1446653523.152673
        exit_status = 0.000000
        output = "PING OK - Packet loss = 0%, RTA = 0.11 ms"
      performance_data = [ "rta=0.114000ms;3000.000000;5000.000000;0.000000", "pl=0%;80;100;0'
        schedule_end = 1446653527.175133
        schedule_start = 1446653583.150000
        state = 0.000000
        type = "CheckResult"
        vars_after = {
                attempt = 1.000000
                reachable = true
                state = 0.000000
                state_type = 1.000000
        }
        vars_before = {
                attempt = 1.000000
                reachable = true
                state = 0.000000
                state_type = 1.000000
        }
}
<3> =>
```

You can use the `--eval` parameter to evaluate a single expression in batch mode. Using the `--file` option you can specify a file which should be evaluated. The output format for batch mode is JSON.

The `--syntax-only` option can be used in combination with `--eval` or `--file` to check a script for syntax errors. In this mode the script is parsed to identify syntax errors but not evaluated.

Here's an example that retrieves the command that was used by Icinga to check the `icinga2-client1.localdomain` host:

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/' --eval
[
    "/usr/local/sbin/check_ping",
    "-H",
    "127.0.0.1",
    "-c",
    "5000,100%",
    "-w",
    "3000,80%"
]
```

11.6 CLI command: Daemon

The CLI command `daemon` provides the functionality to start/stop Icinga 2. Furthermore it allows to run the configuration validation.

```
# icinga2 daemon --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 daemon [<arguments>]

Starts Icinga 2.

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
 --color             use VT100 color codes even when stdout is not a
                           terminal
  -D [ --define ] arg       define a constant
 -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
 -x [ --log-level ] arg   specify the log level for the console log.
                           The valid value is either debug, notice,
                        information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Command options:
  -c [ --config ] arg       parse a configuration file
  -z [ --no-config ]        start without a configuration file
  -C [ --validate ]        exit after validating the configuration
 -e [ --errorlog ] arg    log fatal errors to the specified log file (only
                           works in combination with --daemonize)
  -d [ --daemonize ]        detach from the controlling terminal

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.6.1 Config Files

You can specify one or more configuration files with the `--config` option. Configuration files are processed in the order they're specified on the command-line.

When no configuration file is specified and the `--no-config` is not used Icinga 2 automatically falls back to using the configuration file `SysconfDir + "/icinga2/icinga2.conf"` (where SysconfDir is usually `/etc`).

11.6.2 Config Validation

The `--validate` option can be used to check if configuration files contain errors. If any errors are found, the exit status is 1, otherwise 0 is returned. More details in the configuration validation chapter.

11.7 CLI command: Feature

The `feature enable` and `feature disable` commands can be used to enable and disable features:

```
# icinga2 feature disable <tab>
--app          --define       --include      --log-level     --version       checker
--color        --help         --library      --script-debugger  api          command

# icinga2 feature enable <tab>
--app          --define       --include      --log-level     --version       debuglog
--color        --help         --library      --script-debugger  compatlog      gelf
```

The `feature list` command shows which features are currently enabled:

```
# icinga2 feature list
Disabled features: compatlog debuglog gelf ido-pgsql influxdb livestatus opentsdb perfdata stat
Enabled features: api checker command graphite ido-mysql mainlog notification
```

11.8 CLI command: Node

Provides the functionality to setup master and client nodes in a distributed monitoring scenario.

```
# icinga2 node --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * node setup (set up node)
  * node wizard (wizard for node setup)

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
  --color             use VT100 color codes even when stdout is not a
                            terminal
  -D [ --define ] arg       define a constant
```

```
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                            The valid value is either debug, notice,
                           information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.9 CLI command: Object

The `object` CLI command can be used to list all configuration objects and their attributes. The command also shows where each of the attributes was modified and as such provides debug information for further configuration problem analysis. That way you can also identify which objects have been created from your apply rules.

Runtime modifications via the REST API are not immediately updated. Furthermore there is a known issue with group assign expressions which are not reflected in the host object output. You need to restart Icinga 2 in order to update the `icinga2.debug` cache file.

More information can be found in the troubleshooting section.

```
# icinga2 object --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.7.1-196-g23e8a6253; debug)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * object list (lists all objects)

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
  --color              use VT100 color codes even when stdout is not a
                            terminal
  -D [ --define ] arg       define a constant
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                            The valid value is either debug, notice,
```

```
                        information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.10 CLI command: Pki

Provides the CLI commands to

- generate a new certificate authority (CA)
- generate a new CSR or self-signed certificate
- sign a CSR and return a certificate
- save a master certificate manually
- request a signed certificate from the master
- generate a new ticket for the client setup

This functionality is used by the node setup/wizard CLI commands. You will need them in the distributed monitoring chapter.

```
# icinga2 pki --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * pki new-ca (sets up a new CA)
  * pki new-cert (creates a new CSR)
  * pki request (requests a certificate)
  * pki save-cert (saves another Icinga 2 instance's certificate)
  * pki sign-csr (signs a CSR)
  * pki ticket (generates a ticket)

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
  --color              use VT100 color codes even when stdout is not a
                            terminal
  -D [ --define ] arg       define a constant
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                            The valid value is either debug, notice,
                        information (default), warning, or critical
```

```
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.11 CLI command: Troubleshoot

Collects basic information like version, paths, log files and crash reports for troubleshooting purposes and prints them to a file or the console. See troubleshooting.

Its output defaults to a file named `troubleshooting-[TIMESTAMP].log` so it won't overwrite older troubleshooting files.

Keep in mind that this tool can not collect information from other icinga2 nodes, you will have to run it on each of one of you instances. This is only a tool to collect information to help others help you, it will not attempt to fix anything.

```
# icinga2 troubleshoot --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0)

Usage:
  icinga2 troubleshoot [<arguments>]

Collect logs and other relevant information for troubleshooting purposes.

Global options:
  -h [ --help ]             show this help message
  -V [ --version ]          show version information
  --color             use VT100 color codes even when stdout is not a
                            terminal
  -D [ --define ] arg       define a constant
  -a [ --app ] arg        application library name (default: icinga)
  -l [ --library ] arg      load a library
  -I [ --include ] arg      add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                           The valid value is either debug, notice,
                         information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Command options:
  -c [ --console ]          print to console instead of file
  -o [ --output ] arg       path to output file
  --include-objects      Print the whole objectfile (like `object list`)
  --include-vars         Print all Variables (like `variable list`)
```

```
Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.12 CLI command: Variable

Lists all configured variables (constants) in a similar fashion like object list.

```
# icinga2 variable --help
icinga2 - The Icinga 2 network monitoring daemon (version: v2.8.0; debug)

Usage:
  icinga2 <command> [<arguments>]

Supported commands:
  * variable get (gets a variable)
  * variable list (lists all variables)

Global options:
  -h [ --help ]              show this help message
  -V [ --version ]           show version information
  --color              use VT100 color codes even when stdout is not a
                             terminal
  -D [ --define ] arg        define a constant
  -a [ --app ] arg         application library name (default: icinga)
  -l [ --library ] arg       load a library
  -I [ --include ] arg       add include search directory
  -x [ --log-level ] arg   specify the log level for the console log.
                            The valid value is either debug, notice,
                          information (default), warning, or critical
  -X [ --script-debugger ]  whether to enable the script debugger

Report bugs at <https://github.com/Icinga/icinga2>
Icinga home page: <https://www.icinga.com/>
```

11.13 Enabling/Disabling Features

Icinga 2 provides configuration files for some commonly used features. These are installed in the `/etc/icinga2/features-available` directory and can be enabled and disabled using the `icinga2 feature enable` and `icinga2 feature disable` CLI commands, respectively.

The `icinga2 feature enable` CLI command creates symlinks in the `/etc/icinga2/features-enabled` directory which is included by default in the example configuration file.

You can view a list of enabled and disabled features:

```
# icinga2 feature list
Disabled features: api command compatlog debuglog graphite icingastatus ido-mysql ido-pgsql liv
Enabled features: checker mainlog notification
```

Using the `icinga2 feature enable` command you can enable features:

```
# icinga2 feature enable graphite
Enabling feature graphite. Make sure to restart Icinga 2 for these changes to take effect.
```

You can disable features using the `icinga2 feature disable` command:

```
# icinga2 feature disable ido-mysql livestatus
Disabling feature ido-mysql. Make sure to restart Icinga 2 for these changes to take effect.
Disabling feature livestatus. Make sure to restart Icinga 2 for these changes to take effect.
```

The `icinga2 feature enable` and `icinga2 feature disable` commands do not restart Icinga 2. You will need to restart Icinga 2 using the init script after enabling or disabling features.

11.14 Configuration Validation

Once you've edited the configuration files make sure to tell Icinga 2 to validate the configuration changes. Icinga 2 will log any configuration error including a hint on the file, the line number and the affected configuration line itself.

The following example creates an apply rule without any `assign` condition.

```
apply Service "my-ping4" {
  import "generic-service"
  check_command = "ping4"
  //assign where host.address
}
```

Validate the configuration:

```
# icinga2 daemon -C

[2014-05-22 17:07:25 +0200] critical/ConfigItem: Location:
/etc/icinga2/conf.d/tests/my.conf(5): }
/etc/icinga2/conf.d/tests/my.conf(6):
/etc/icinga2/conf.d/tests/my.conf(7): apply Service "my-ping4" {
                                        ^^^^^^^^^^^^^^
/etc/icinga2/conf.d/tests/my.conf(8):  import "test-generic-service"
/etc/icinga2/conf.d/tests/my.conf(9):   check_command = "ping4"

Config error: 'apply' is missing 'assign'
[2014-05-22 17:07:25 +0200] critical/ConfigItem: 1 errors, 0 warnings.
Icinga 2 detected configuration errors.
```

If you encounter errors during configuration validation, please make sure to read the troubleshooting chapter.

You can also use the CLI command `icinga2 object list` after validation passes to analyze object attributes, inheritance or created objects by apply rules. Find more on troubleshooting with `object list` in this chapter.

11.15 Reload on Configuration Changes

Every time you have changed your configuration you should first tell Icinga 2 to validate. If there are no validation errors, you can safely reload the Icinga 2 daemon.

```
# systemctl reload icinga2
```

The `reload` action will send the `SIGHUP` signal to the Icinga 2 daemon which will validate the configuration in a separate process and not stop the other events like check execution, notifications, etc.

12 Icinga 2 API

12.1 Setting up the API

You can run the CLI command `icinga2 api setup` to enable the `api` feature and set up certificates as well as a new API user `root` with an auto-generated password in the `/etc/icinga2/conf.d/api-users.conf` configuration file:

```
# icinga2 api setup
```

Make sure to restart Icinga 2 to enable the changes you just made:

```
# service icinga2 restart
```

If you prefer to set up the API manually, you will have to perform the following steps:

- Set up X.509 certificates for Icinga 2
- Enable the `api` feature (`icinga2 feature enable api`)
- Create an `ApiUser` object for authentication

The next chapter provides a quick overview of how you can use the API.

12.1.1 Creating ApiUsers

The CLI command `icinga2 api user` allows you to create an ApiUser object with a hashed password string, ready to be added to your configuration. Example:

```
$ icinga2 api user --user icingaweb2 --passwd icinga
object ApiUser "icingaweb2" {
 password_hash ="$5$d5f1a17ea308acb6$9e9fd5d24a9373a16e8811765cc5a5939687faf9ef8ed496db6e7f1
  // client_cn = ""

  permissions = [ "*" ]
}
```

Optionally a salt can be provided with `--salt`, otherwise a random value will be used. When ApiUsers are stored this way, even somebody able to read the configuration files won't be able to authenticate using this information. There is no way to recover your password should you forget it, you'd need to create it anew.

12.2 Introduction

The Icinga 2 API allows you to manage configuration objects and resources in a simple, programmatic way using HTTP requests.

The URL endpoints are logically separated allowing you to easily make calls to

- query, create, modify and delete config objects
- perform actions (reschedule checks, etc.)
- subscribe to event streams
- manage configuration packages
- evaluate script expressions

12.2.1 Requests

Any tool capable of making HTTP requests can communicate with the API, for example curl.

Requests are only allowed to use the HTTPS protocol so that traffic remains encrypted.

By default the Icinga 2 API listens on port `5665` which is shared with the cluster stack. The port can be changed by setting the `bind_port` attribute for the ApiListener object in the `/etc/icinga2/features-available/api.conf` configuration file.

Supported request methods:

Method	Usage
GET	Retrieve information about configuration objects. Any request using the GET method is read-only and does not affect any objects.
POST	Update attributes of a specified configuration object.
PUT	Create a new object. The PUT request must include all attributes required to create a new object.

Method	Usage
DELETE	Remove an object created by the API. The DELETE method is idempotent and does not require any check if the object actually exists.

All requests apart from `GET` require that the following `Accept` header is set:

```
Accept: application/json
```

Each URL is prefixed with the API version (currently "/v1").

12.2.2 Responses

Successful requests will send back a response body containing a `results` list. Depending on the number of affected objects in your request, the `results` list may contain more than one entry.

The output will be sent back as a JSON object:

```
{
    "results": [
        {
            "code": 200.0,
            "status": "Object was created."
        }
    ]
}
```

Tip

You can use the `pretty` parameter to beautify the JSON response with Icinga v2.9+.

You can also use jq or `python -m json.tool` in combination with curl on the CLI.

```
curl ... | python -m json.tool
```

Note

Future versions of Icinga 2 might set additional fields. Your application should gracefully handle fields it is not familiar with, for example by ignoring them.

12.2.3 HTTP Statuses

The API will return standard HTTP statuses including error codes.

When an error occurs, the response body will contain additional information about the problem and its source.

A status code between 200 and 299 generally means that the request was successful.

Return codes within the 400 range indicate that there was a problem with the request. Either you did not authenticate correctly, you are missing the authorization for your requested action, the requested object does not exist or the request was malformed.

A status in the range of 500 generally means that there was a server-side problem and Icinga 2 is unable to process your request.

12.2.4 Authentication

There are two different ways for authenticating against the Icinga 2 API:

- username and password using HTTP basic auth
- X.509 certificate

In order to configure a new API user you'll need to add a new ApiUser configuration object. In this example `root` will be the basic auth username and the `password` attribute contains the basic auth password.

```
# vim /etc/icinga2/conf.d/api-users.conf

object ApiUser "root" {
  password = "icinga"
}
```

Alternatively you can use X.509 client certificates by specifying the `client_cn` the API should trust. The X.509 certificate has to be signed by the CA certificate that is configured in the ApiListener object.

```
# vim /etc/icinga2/conf.d/api-users.conf

object ApiUser "root" {
  client_cn = "CertificateCommonName"
}
```

An `ApiUser` object can have both authentication methods configured.

You can test authentication by sending a GET request to the API:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1'
```

In case you get an error message make sure to check the API user credentials.

When using client certificates for authentication you'll need to pass your client certificate and private key to the curl call:

```
$ curl -k --cert example.localdomain.crt --key example.localdomain.key 'https://example.locald
```

In case of an error make sure to verify the client certificate and CA.

The curl parameter `-k` disables certificate verification and should therefore only be used for testing. In order to securely check each connection you'll need to specify the trusted CA certificate using the curl parameter`--cacert`:

```
$ curl -u root:icinga --cacert ca.crt 'icinga2.node1.localdomain:5665/v1'
```

Read the next chapter on API permissions in order to configure authorization settings for your newly created API user.

12.2.5 Permissions

By default an API user does not have any permissions to perform actions on the URL endpoints.

Permissions for API users must be specified in the `permissions` attribute as array. The array items can be a list of permission strings with wildcard matches.

Example for an API user with all permissions:

```
permissions = [ "*" ]
```

Note that you can use wildcards. Here's another example that only allows the user to perform read-only object queries for hosts and services:

```
permissions = [ "objects/query/Host", "objects/query/Service" ]
```

You can also further restrict permissions by specifying a filter expression. The filter expression has to be a lambda function which must return a boolean value.

The following example allows the API user to query all hosts and services which have a custom attribute `os` that matches the regular expression `^Linux`. The regex function is available as global function.

```
permissions = [
  {
    permission = "objects/query/Host"
    filter = {{ regex("^Linux", host.vars.os) }}
  },
  {
    permission = "objects/query/Service"
    filter = {{ regex("^Linux", service.vars.os) }}
  }
]
```

More information about filters can be found in the filters chapter.

Permissions are tied to a maximum HTTP request size to prevent abuse, responses sent by Icinga are not limited. An API user with all permissions ("*") may send up to 512 MB regardless of the endpoint.

Available permissions for specific URL endpoints:

Permissions	URL Endpoint	Supports filters	Max body size in MB
actions/<action>	/v1/actions	Yes	1
config/query	/v1/config	No	1
config/modify	/v1/config	No	512
console	/v1/console	No	1
events/<type>	/v1/events	No	1
objects/query/<type>	/v1/objects	Yes	1
objects/create/<type>	/v1/objects	No	1
objects/modify/<type>	/v1/objects	Yes	1
objects/delete/<type>	/v1/objects	Yes	1
status/query	/v1/status	Yes	1
templates/<type>	/v1/templates	Yes	1
types	/v1/types	Yes	1
variables	/v1/variables	Yes	1

The required actions or types can be replaced by using a wildcard match ("*").

12.2.6 Parameters

Depending on the request method there are two ways of passing parameters to the request:

- JSON object as request body (all request methods other than `GET`)
- Query string as URL parameter (all request methods)

Reserved characters by the HTTP protocol must be URL-encoded as query string, e.g. a space character becomes `%20`.

Example for a URL-encoded query string:

```
/v1/objects/hosts?filter=match(%22example.localdomain*%22,host.name)&attrs=name&attrs=state
```

Here are the exact same query parameters as a JSON object:

```
{ "filter": "match(\"example.localdomain*\",host.name)", "attrs": [ "host.name", "host.state"
```

The match function is available as global function in Icinga 2.

12.2.7 Request Method Override

`GET` requests do not allow you to send a request body. In case you cannot pass everything as URL parameters (e.g. complex filters or JSON-encoded dictionaries) you can use the `X-HTTP-Method-Override` header. This comes in handy when you are using HTTP proxies disallowing `PUT` or `DELETE` requests too.

Query an existing object by sending a `POST` request with `X-HTTP-Method-Override: GET` as request header:

```
$ curl -k -s -u 'root:icinga' -H 'Accept: application/json' -X POST -H 'X-HTTP-Method-Override:
```

Delete an existing object by sending a `POST` request with `X-HTTP-Method-Override: DELETE` as request header:

```
$ curl -k -s -u 'root:icinga' -H 'Accept: application/json' -X POST -H 'X-HTTP-Method-Override:
```

12.2.8 Filters

12.2.8.1 Simple Filters

By default actions and queries operate on all objects unless further restricted by the user. For example, the following query returns all `Host` objects:

```
https://localhost:5665/v1/objects/hosts
```

If you're only interested in a single object, you can limit the output to that object by specifying its name:

```
https://localhost:5665/v1/objects/hosts?host=localhost
```

The name of the URL parameter is the lower-case version of the type the query applies to. For example, for `Host` objects the URL parameter therefore is `host`, for `Service` objects it is `service` and so on.

You can also specify multiple objects:

```
https://localhost:5665/v1/objects/hosts?hosts=first-host&hosts=second-host
```

Again – like in the previous example – the name of the URL parameter is the lower-case version of the type. However, because we're specifying multiple objects here the **plural form** of the type is used.

When specifying names for objects which have composite names like for example services the full name has to be used:

```
https://localhost:5665/v1/objects/services?service=localhost!ping6
```

The full name of an object can be obtained by looking at the `__name` attribute.

12.2.8.2 Advanced Filters

Most of the information provided in this chapter applies to both permission filters (as used when configuring `ApiUser` objects) and filters specified in queries.

Advanced filters allow users to filter objects using lambda expressions. The syntax for these filters is the same like for apply rule expressions.

> **Note**
>
> Filters used as URL parameter must be URL-encoded. The following examples are **not URL-encoded** for better readability.

Example matching all services in NOT-OK state:

```
https://localhost:5665/v1/objects/services?filter=service.state!=ServiceOK
```

Example matching all hosts by a name string pattern:

```
https://localhost:5665/v1/objects/hosts?filter=match("example.localdomain*",host.name)
```

Example for all hosts which are in the host group `linux-servers`:

```
https://localhost:5665/v1/objects/hosts?filter="linux-servers" in host.groups
```

User-specified filters are run in a sandbox environment which ensures that filters cannot modify Icinga's state, for example object attributes or global variables.

When querying objects of a specific type the filter expression is evaluated for each object of that type. The object is made available to the filter expression as a variable whose name is the lower-case version of the object's type name.

For example when querying objects of type `Host` the variable in the filter expression is named `host`. Additionally related objects such as the host's check command are also made available (e.g., via the `check_command` variable). The variable names are the exact same as for the `joins` query parameter; see object query joins for details.

The object is also made available via the `obj` variable. This makes it easier to build filters which can be used for more than one object type (e.g., for permissions).

Some queries can be performed for more than just one object type. One example is the 'reschedule-check' action which can be used for both hosts and services. When using advanced filters you will also have to specify the type using the `type` parameter:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "type": "Service", "filter": "service.name==\"ping6\"", "pretty": true }'
```

When building filters you have to ensure that values such as `"linux-servers"` are escaped properly according to the rules of the Icinga 2 configuration language.

To make using the API in scripts easier you can use the `filter_vars` attribute to specify variables which should be made available to your filter expression. This way you don't have to worry about escaping values:

```
$ curl -k -s -u 'root:icinga' -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X
-d '{ "filter": "host.vars.os == os", "filter_vars": { "os": "Linux" }, "pretty": true }'
```

We're using X-HTTP-Method-Override here because the HTTP specification does not allow message bodies for GET requests.

The `filters_vars` attribute can only be used inside the request body, but not as a URL parameter because there is no way to specify a dictionary in a URL.

12.3 Config Objects

Provides methods to manage configuration objects:

- creating objects
- querying objects
- modifying objects
- deleting objects

12.3.1 API Objects and Cluster Config Sync

Newly created or updated objects can be synced throughout your Icinga 2 cluster. Set the `zone` attribute to the zone this object belongs to and let the API and cluster handle the rest.

Objects without a zone attribute are only synced in the same zone the Icinga instance belongs to.

> **Note**
>
> Cluster nodes must accept configuration for creating, modifying and deleting objects. Ensure that `accept_config` is set to `true` in the ApiListener object on each node.

If you add a new cluster instance, or reconnect an instance which has been offline for a while, Icinga 2 takes care of the initial object sync for all objects created by the API.

12.3.2 Querying Objects

You can request information about configuration objects by sending a `GET` query to the `/v1/objects/<type>` URL endpoint. `<type` has to be replaced with the plural name of the object type you are interested in:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/objects/hosts'
```

A list of all available configuration types is available in the object types chapter.

The following URL parameters are available:

Parameters	Type	Description
attrs	Array	**Optional.** Limited attribute list in the output.
joins	Array	**Optional.** Join related object types and their attributes specified as list (`?joins=host` for the entire set, or selectively by `?joins=host.name`).
meta	Array	**Optional.** Enable meta information using `?meta=used_by` (references from other objects) and/or `?meta=location` (location information) specified as list. Defaults to disabled.

In addition to these parameters a filter may be provided.

Instead of using a filter you can optionally specify the object name in the URL path when querying a single object. For objects with composite names (e.g. services) the full name (e.g. `example.localdomain!http`) must be specified:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/objects/services/example.localdomain!ht
```

You can limit the output to specific attributes using the `attrs` URL parameter:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/objects/hosts/example.localdomain?attrs
{
    "results": [
        {
            "attrs": {
                "name": "example.localdomain"
                "address": "192.168.1.1"
            },
            "joins": {},
```

```
            "meta": {},
            "name": "example.localdomain",
            "type": "Host"
        }
    ]
}
```

12.3.2.1 Object Queries Result

Each response entry in the results array contains the following attributes:

Attribute	Type	Description
name	String	Full object name.
type	String	Object type.
attrs	Dictionary	Object attributes (can be filtered using the URL parameter `attrs`).
joins	Dictionary	Joined object types as key, attributes as nested dictionary. Disabled by default.
meta	Dictionary	Contains `used_by` object references. Disabled by default, enable it using `?meta=used_by` as URL parameter.

12.3.2.2 Object Query Joins

Icinga 2 knows about object relations. For example it can optionally return information about the host when querying service objects.

The following query retrieves all host attributes:

`https://localhost:5665/v1/objects/services?joins=host`

Instead of requesting all host attributes you can also limit the output to specific attributes:

`https://localhost:5665/v1/objects/services?joins=host.name&joins=host.address`

You can request that all available joins are returned in the result set by using the `all_joins` query parameter.

`https://localhost:5665/v1/objects/services?all_joins=1`

Note

For performance reasons you should only request attributes which your application requires.

The following joins are available:

Object Type	Object Relations (`joins` prefix name)
Service	host, check_command, check_period, event_command, command_endpoint
Host	check_command, check_period, event_command, command_endpoint
Notification	host, service, command, period
Dependency	child_host, child_service, parent_host, parent_service, period
User	period
Zones	parent

Here's an example that retrieves all service objects for hosts which have had their `os` custom attribute set to `Linux`. The result set contains the `display_name` and `check_command` attributes for the service. The query also returns the host's `name` and `address` attribute via a join:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/objects/services?attrs=display_name&at
```

```
{
    "results": [
        {
            "attrs": {
                "check_command": "ping4",
                "display_name": "ping4"
            },
            "joins": {
                "host": {
                    "address": "192.168.1.1",
                    "name": "example.localdomain"
```

```
                }
            },
            "meta": {},
            "name": "example.localdomain!ping4",
            "type": "Service"
        },
        {
            "attrs": {
                "check_command": "ssh",
                "display_name": "ssh"
            },
            "joins": {
                "host": {
                    "address": "192.168.1.1",
                    "name": "example.localdomain"
                }
            },
            "meta": {},
            "name": "example.localdomain!ssh",
            "type": "Service"
        }
    ]
}
```


In case you want to fetch all comments for hosts and services, you can use the following query URL (similar example for downtimes):

```
https://localhost:5665/v1/objects/comments?joins=host&joins=service
```

This is another example for listing all service objects which are unhandled problems (state is not OK and no downtime or acknowledgement set). We're using X-HTTP-Method-Override here because we want to pass all query attributes in the request body.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X PO
-d '{ "joins": [ "host.name", "host.address" ], "attrs": [ "name", "state", "downtime_depth", "a
```


```
{
    "results": [
        {
            "attrs": {
                "acknowledgement": 0.0,
                "downtime_depth": 0.0,
                "name": "10807-service",
                "state": 3.0
            },
            "joins": {
                "host": {
```

```
                    "address": "",
                    "name": "10807-host"
                }
            },
            "meta": {},
            "name": "10807-host!10807-service",
            "type": "Service"
        }
    ]
}
```


In order to list all acknowledgements without expire time, you query the `/v1/objects/comments` URL endpoint with `joins` and `filter` request parameters using the X-HTTP-Method-Override method:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X PO
-d '{ "joins": [ "service.name", "service.acknowledgement", "service.acknowledgement_expiry" ]
```


```
{
    "results": [
        {
            "attrs": {
                "author": "icingaadmin",
                "text": "maintenance work"
            },
            "joins": {
                "service": {
                    "__name": "example.localdomain!disk /",
                    "acknowledgement": 1.0,
                    "acknowledgement_expiry": 0.0
                }
            },
            "meta": {},
        "name": "example.localdomain!disk /!example.localdomain-1495457222-0",
            "type": "Comment"
        }
    ]
}
```


12.3.3 Creating Config Objects

New objects must be created by sending a PUT request. The following parameters need to be passed inside the JSON body:

Parameters	Type	Description
templates	Array	**Optional.** Import existing configuration templates for this object type. Note: These templates must either be statically configured or provided in config packages-
attrs	Dictionary	**Required.** Set specific object attributes for this object type.
ignore_on_error	Boolean	**Optional.** Ignore object creation errors and return an HTTP 200 status instead.

The object name must be specified as part of the URL path. For objects with composite names (e.g. services) the full name (e.g. `example.localdomain!http`) must be specified.

If attributes are of the Dictionary type, you can also use the indexer format. This might be necessary to only override specific custom variables and keep all other existing custom variables (e.g. from templates):

```
"attrs": { "vars.os": "Linux" }
```

Example for creating the new host object `example.localdomain`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X PUT 'https://localhost:5665/v1/obj
-d '{ "templates": [ "generic-host" ], "attrs": { "address": "192.168.1.1", "check_command": "h
{
    "results": [
        {
            "code": 200.0,
            "status": "Object was created."
        }
    ]
}
```

If the configuration validation fails, the new object will not be created and the response body contains a detailed error message. The following example is missing the `check_command` attribute which is required for host objects:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X PUT 'https://localhost:5665/v1/obj
-d '{ "attrs": { "address": "192.168.1.1", "vars.os" : "Linux" }, "pretty": true }'
{
```

```
    "results": [
        {
            "code": 500.0,
            "errors": [
           "Error: Validation failed for object 'example.localdomain' of type 'Host'; Attribute
            ],
            "status": "Object could not be created."
        }
    ]
}
```

Service objects must be created using their full name ("hostname!servicename") referencing an existing host object:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X PUT 'https://localhost:5665/v1/obj
-d '{ "templates": [ "generic-service" ], "attrs": { "check_command": "load", "check_interval":
```

Example for a new CheckCommand object:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X PUT 'https://localhost:5665/v1/obj
-d '{ "templates": [ "plugin-check-command" ], "attrs": { "command": [ "/usr/local/sbin/check_h
```

12.3.4 Modifying Objects

Existing objects must be modified by sending a `POST` request. The following parameters need to be passed inside the JSON body:

Parameters	Type	Description
attrs	Dictionary	**Required.** Set specific object attributes for this object type.

In addition to these parameters a filter should be provided.

> **Note**:
>
> Modified attributes do not trigger a re-evaluation of existing static apply rules and group assignments. Delete and re-create the objects if you require such changes.
>
> Furthermore you cannot modify templates which have already been resolved during object creation. There are attributes which can only be set for PUT requests such as `groups` or `zone`. A complete list of `no_user_modify` attributes can be fetched from the types URL endpoint.

If attributes are of the Dictionary type, you can also use the indexer format:

```
"attrs": { "vars.os": "Linux" }
```

The following example updates the `address` attribute and the custom attribute `os` for the `example.localdomain` host:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ob
-d '{ "attrs": { "address": "192.168.1.2", "vars.os" : "Windows" }, "pretty": true }'
{
    "results": [
        {
            "code": 200.0,
            "name": "example.localdomain",
            "status": "Attributes updated.",
            "type": "Host"
        }
    ]
}
```

12.3.5 Deleting Objects

You can delete objects created using the API by sending a `DELETE` request.

Parameters	Type	Description
cascade	Boolean	**Optional.** Delete objects depending on the deleted objects (e.g. services on a host).

In addition to these parameters a filter should be provided.

Example for deleting the host object `example.localdomain`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X DELETE 'https://localhost:5665/v1/
{
    "results": [
        {
            "code": 200.0,
            "name": "example.localdomain",
            "status": "Object was deleted.",
            "type": "Host"
        }
    ]
}
```

12.4 Config Templates

Provides methods to manage configuration templates:

- querying templates

Creation, modification and deletion of templates at runtime is not supported.

12.4.1 Querying Templates

You can request information about configuration templates by sending a `GET` query to the `/v1/templates/<type>` URL endpoint. `<type` has to be replaced with the plural name of the object type you are interested in:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/templates/hosts'
```

A list of all available configuration types is available in the object types chapter.

A filter may be provided for this query type. The template object can be accessed in the filter using the `tmpl` variable. In this example the match function is used to check a wildcard string pattern against `tmpl.name`. The `filter` attribute is passed inside the request body thus requiring to use X-HTTP-Method-Override here.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X PO
-d '{ "filter": "match(\"g*\", tmpl.name)" }'
```

Instead of using a filter you can optionally specify the template name in the URL path when querying a single object:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/templates/hosts/generic-host'
```

The result set contains the type, name as well as the location of the template.

12.5 Variables

Provides methods to manage global variables:

- querying variables

12.5.1 Querying Variables

You can request information about global variables by sending a `GET` query to the `/v1/variables/` URL endpoint:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/variables'
```

A filter may be provided for this query type. The variable information object can be accessed in the filter using the `variable` variable. The `filter` attribute is passed inside the request body thus requiring to use X-HTTP-Method-Override here.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X PO
-d '{ "filter": "variable.type in [ \"String\", \"Number\" ]" }'
```

Instead of using a filter you can optionally specify the variable name in the URL path when querying a single variable:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/variables/PrefixDir'
```

The result set contains the type, name and value of the global variable.

12.6 Actions

There are several actions available for Icinga 2 provided by the `/v1/actions` URL endpoint. You can run actions by sending a `POST` request.

In case you have been using the external commands in the past, the API actions provide a similar interface with filter capabilities for some of the more common targets which do not directly change the configuration.

All actions return a 200 `OK` or an appropriate error code for each action performed on each object matching the supplied filter.

Actions which affect the Icinga Application itself such as disabling notification on a program-wide basis must be applied by updating the IcingaApplication object called `app`.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ob
```

12.6.1 process-check-result

Process a check result for a host or a service.

Send a `POST` request to the URL endpoint `/v1/actions/process-check-result`.

Parameter	Type	Description

Parameter	Type	Description
exit_status	Number	**Required.** For services: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN, for hosts: 0=OK, 1=CRITICAL.
plugin_output	String	**Required.** One or more lines of the plugin main output. Does **not** contain the performance data.
performance_data	Array String	
check_command	Array	**Optional.** The first entry should be the check commands path, then one entry for each command line option followed by an entry for each of its argument.
check_source	String	**Optional.** Usually the name of the `command_endpoint`

Parameter	Type	Description
execution_start	Timestamp	**Optional.** The timestamp where a script/process started its execution.
execution_end	Timestamp	**Optional.** The timestamp where a script/process ended its execution. This timestamp is used in features to determine e.g. the metric timestamp.
ttl	Number	**Optional.** Time-to-live duration in seconds for this check result. The next expected check result is `now + ttl` where freshness checks are executed.

In addition to these parameters a filter must be provided. The valid types for this action are `Host` and `Service`.

Example for the service `passive-ping6`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "exit_status": 2, "plugin_output": "PING CRITICAL - Packet loss = 100%", "performance_data

{
```

```
    "results": [
        {
            "code": 200.0,
        "status": "Successfully processed check result for object 'localdomain!passive-ping6'.
        }
    ]
}
```

Example for using the `Host` type and filter by the host name:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "filter": "host.name==\"example.localdomain\"", "type": "Host", "exit_status": 1, "plugi
```

You can avoid URL encoding of white spaces in object names by using the `filter` attribute in the request body.

Note

Multi-line plugin output requires the following format: The first line is treated as `short` plugin output corresponding to the first line of the plugin output. Subsequent lines are treated as `long` plugin output. Please note that the performance data is separated from the plugin output and has to be passed as `performance_data` attribute.

12.6.2 reschedule-check

Reschedule a check for hosts and services. The check can be forced if required.

Send a `POST` request to the URL endpoint `/v1/actions/reschedule-check`.

Parameter	Type	Description
next_check	Timestamp	**Optional.** The next check will be run at this time. If omitted, the current time is used.

Parameter	Type	Description
force	Boolean	**Optional.** Defaults to `false`. If enabled, the checks are executed regardless of time period restrictions and checks being disabled per object or on a global basis.

In addition to these parameters a filter must be provided. The valid types for this action are `Host` and `Service`.

The example reschedules all services with the name "ping6" to immediately perform a check (`next_check` default), ignoring any time periods or whether active checks are allowed for the service (`force=true`).

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "type": "Service", "filter": "service.name==\"ping6\"", "force": true, "pretty": true }'

{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully rescheduled check for object 'example.localdomain!ping6'."
        }
    ]
}
```

12.6.3 send-custom-notification

Send a custom notification for hosts and services. This notification type can be forced being sent to all users.

Send a `POST` request to the URL endpoint `/v1/actions/send-custom-notification`.

Parameter	Type	Description
author	String	**Required.** Name of the author, may be empty.
comment	String	**Required.** Comment text, may be empty.
force	Boolean	**Optional.** Default: false. If true, the notification is sent regardless of downtimes or whether notifications are enabled or not.

In addition to these parameters a filter must be provided. The valid types for this action are `Host` and `Service`.

Example for a custom host notification announcing a global maintenance to host owners:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "type": "Host", "author": "icingaadmin", "comment": "System is going down for maintenance'

{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully sent custom notification for object 'host0'."
        },
        {
            "code": 200.0,
        "status": "Successfully sent custom notification for object 'host1'."
        }
}
```

12.6.4 delay-notification

Delay notifications for a host or a service. Note that this will only have an effect if the service stays in the same problem state that it is currently in. If

the service changes to another state, a new notification may go out before the time you specify in the `timestamp` argument.

Send a `POST` request to the URL endpoint `/v1/actions/delay-notification`.

Parameter	Type	Description
timestamp	Timestamp	**Required.** Delay notifications until this timestamp.

In addition to these parameters a filter must be provided. The valid types for this action are `Host` and `Service`.

Example:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "type": "Service", "timestamp": 1446389894, "pretty": true }'

{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully delayed notifications for object 'host0!service0'."
        },
        {
            "code": 200.0,
        "status": "Successfully delayed notifications for object 'host1!service1'."
        }
}
```

12.6.5 acknowledge-problem

Allows you to acknowledge the current problem for hosts or services. By acknowledging the current problem, future notifications (for the same state if `sticky` is set to `false`) are disabled.

Send a `POST` request to the URL endpoint `/v1/actions/acknowledge-problem`.

Parameter	Type	Description
author	String	**Required.** Name of the author, may be empty.
comment	String	**Required.** Comment text, may be empty.

Parameter	Type	Description
expiry	Timestamp	**Optional.** Whether the acknowledgement will be removed at the timestamp.
sticky	Boolean	**Optional.** Whether the acknowledgement will be set until the service or host fully recovers. Defaults to `false`.
notify	Boolean	**Optional.** Whether a notification of the `Acknowledgement` type will be sent. Defaults to `false`.
persistent	Boolean	**Optional.** When the comment is of type `Acknowledgement` and this is set to `true`, the comment will remain after the acknowledgement recovers or expires. Defaults to `false`.

In addition to these parameters a filter must be provided. The valid types for this action are `Host` and `Service`.

The following example acknowledges all services which are in a hard critical state and sends out a notification for them:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "author": "icingaadmin", "comment": "Global outage. Working on it.", "notify": true, "pret

{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully acknowledged problem for object 'example2.localdomain!ping4'."
        },
        {
            "code": 200.0,
        "status": "Successfully acknowledged problem for object 'example.localdomain!ping4'."
        }
}
```

12.6.6 remove-acknowledgement

Removes the acknowledgements for services or hosts. Once the acknowledgement has been removed notifications will be sent out again.

Send a `POST` request to the URL endpoint `/v1/actions/remove-acknowledgement`.

A filter must be provided. The valid types for this action are `Host` and `Service`.

The example removes all service acknowledgements:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac

{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully removed acknowledgement for object 'host0!service0'."
        },
        {
            "code": 200.0,
        "status": "Successfully removed acknowledgement for object 'example2.localdomain!aws-k
        }
}
```

12.6.7 add-comment

Adds a `comment` from an `author` to services or hosts.

Send a `POST` request to the URL endpoint `/v1/actions/add-comment`.

Parameter	Type	Description
author	string	**Required.** Name of the author, may be empty.
comment	string	**Required.** Comment text, may be empty.

In addition to these parameters a filter must be provided. The valid types for this action are `Host` and `Service`.

The following example adds a comment for all `ping4` services:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
{
    "results": [
        {
            "code": 200.0,
            "legacy_id": 26.0,
        "name": "example.localdomain!ping4!example.localdomain-1446824161-0",
        "status": "Successfully added comment 'example.localdomain!ping4!example.localdomain-
        },
        {
            "code": 200.0,
            "legacy_id": 27.0,
        "name": "example2.localdomain!ping4!example.localdomain-1446824161-1",
        "status": "Successfully added comment 'example2.localdomain!ping4!example.localdomain
        }
    ]
}
```

12.6.8 remove-comment

Remove the comment using its `name` attribute , returns `OK` if the comment did not exist. **Note**: This is **not** the legacy ID but the comment name returned by Icinga 2 when adding a comment.

Send a `POST` request to the URL endpoint `/v1/actions/remove-comment`.

A filter must be provided. The valid types for this action are `Host`, `Service` and `Comment`.

Example for a simple filter using the `comment` URL parameter:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully removed comment 'example2.localdomain!ping4!mbmif.local-14469
```

```
        }
    ]
}
```

Example for removing all service comments using a service name filter for `ping4`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully removed all comments for object 'example2.localdomain!ping4'."
        },
        {
            "code": 200.0,
        "status": "Successfully removed all comments for object 'example.localdomain!ping4'."
        }
    ]
}
```

12.6.9 schedule-downtime

Schedule a downtime for hosts and services.

Send a `POST` request to the URL endpoint `/v1/actions/schedule-downtime`.

Parameter	Type	Description
author	String	**Required.** Name of the author.
comment	String	**Required.** Comment text.
start_time	Timestamp	**Required.** Timestamp marking the beginning of the downtime.
end_time	Timestamp	**Required.** Timestamp marking the end of the downtime.

Parameter	Type	Description
fixed	Boolean	**Optional.** Defaults to `true`. If true, the downtime is `fixed` otherwise `flexible`. See downtimes for more information.
duration	Number	**Required for flexible downtimes.** Duration of the downtime in seconds if `fixed` is set to false.
trigger_name	String	**Optional.** Sets the trigger for a triggered downtime. See downtimes for more information on triggered downtimes.
child_options	Number	**Optional.** Schedule child downtimes. `0` does not do anything, `1` schedules child downtimes triggered by this downtime, `2` schedules non-triggered downtimes. Defaults to `0`.

In addition to these parameters a filter must be provided. The valid types for

this action are `Host` and `Service`.

Example:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
{
    "results": [
        {
            "code": 200.0,
            "legacy_id": 2.0,
        "name": "example2.localdomain!ping4!example.localdomain-1446822004-0",
        "status": "Successfully scheduled downtime 'example2.localdomain!ping4!example.locald
        },
        {
            "code": 200.0,
            "legacy_id": 3.0,
        "name": "example.localdomain!ping4!example.localdomain-1446822004-1",
        "status": "Successfully scheduled downtime 'example.localdomain!ping4!example.localdo
        }
    ]
}
```

12.6.10 remove-downtime

Remove the downtime using its `name` attribute , returns `OK` if the downtime did not exist. **Note**: This is **not** the legacy ID but the downtime name returned by Icinga 2 when scheduling a downtime.

Send a `POST` request to the URL endpoint `/v1/actions/remove-downtime`.

A filter must be provided. The valid types for this action are `Host`, `Service` and `Downtime`.

Example for a simple filter using the `downtime` URL parameter:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully removed downtime 'example.localdomain!ping4!mbmif.local-14469
        }
    ]
}
```

Example for removing all host downtimes using a host name filter for `example.localdomain`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
```

```
{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully removed all downtimes for object 'example.localdomain'."
        }
    ]
}
```

Example for removing a downtime from a host but not the services filtered by the author name. This example uses filter variables explained in the advanced filters chapter.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
        -d $'{
  "type": "Downtime",
 "filter": "host.name == filterHost && !service && downtime.author == filterAuthor",
  "filter_vars": {
    "filterHost": "example.localdomain",
    "filterAuthor": "icingaadmin"
  },
  "pretty": true
}'

{
    "results": [
        {
            "code": 200.0,
        "status": "Successfully removed downtime 'example.localdomain!mbmif.local-1463043129-
        }
    ]
}
```

12.6.11 shutdown-process

Shuts down Icinga2. May or may not return.

Send a `POST` request to the URL endpoint `/v1/actions/shutdown-process`.

This action does not support a target type or filter.

Example:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac

{
    "results": [
        {
```

```
            "code": 200.0,
            "status": "Shutting down Icinga 2."
        }
    ]
}
```

12.6.12 restart-process

Restarts Icinga2. May or may not return.

Send a `POST` request to the URL endpoint `/v1/actions/restart-process`.

This action does not support a target type or filter.

Example:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
```

```
{
    "results": [
        {
            "code": 200.0,
            "status": "Restarting Icinga 2."
        }
    ]
}
```

12.6.13 generate-ticket

Generates a PKI ticket for CSR auto-signing. This can be used in combination with satellite/client setups requesting this ticket number.

Send a `POST` request to the URL endpoint `/v1/actions/generate-ticket`.

Parameter	Type	Description
cn	String	**Required.** The host's common name for which the ticket should be geenerated.

Example:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ac
-d '{ "cn": "icinga2-client1.localdomain", "pretty": true }'
```

```
{
    "results": [
        {
            "code": 200.0,
        "status": "Generated PKI ticket '4f75d2ecd253575fe9180938ebff7cbca262f96e' for common
            "ticket": "4f75d2ecd253575fe9180938ebff7cbca262f96e"
        }
    ]
}
```

12.7 Event Streams

You can subscribe to event streams by sending a `POST` request to the URL endpoint `/v1/events`. The following parameters need to be specified (either as URL parameters or in a JSON-encoded message body):

Parameter	Type	Description
types	Array	**Required.** Event type(s). Multiple types as URL parameters are supported.
queue	String	**Required.** Unique queue name. Multiple HTTP clients can use the same queue as long as they use the same event types and filter.
filter	String	**Optional.** Filter for specific event attributes using filter expressions.

12.7.1 Event Stream Types

The following event stream types are available:

Type	Description
CheckResult	Check results for hosts and services.
StateChange	Host/service state changes.
Notification	Notification events including notified users for hosts and services.
AcknowledgementSet	Acknowledgement set on hosts and services.
AcknowledgementCleared	Acknowledgement cleared on hosts and services.
CommentAdded	Comment added for hosts and services.
CommentRemoved	Comment removed for hosts and services.
DowntimeAdded	Downtime added for hosts and services.
DowntimeRemoved	Downtime removed for hosts and services.
DowntimeStarted	Downtime started for hosts and services.
DowntimeTriggered	Downtime triggered for hosts and services.

Note: Each type requires API permissions being set.

Example for all downtime events:

```
&types=DowntimeAdded&types=DowntimeRemoved&types=DowntimeTriggered
```

12.7.1.1 Event Stream Type: CheckResult

Name	Type	Description
type	String	Event type `CheckResult`.
timestamp	Timestamp	Unix timestamp when the event happened.
host	String	Host name.
service	String	Service name. Optional if this is a host check result.
check_result	CheckResult	Serialized CheckResult value type.

12.7.1.2 Event Stream Type: StateChange

Name	Type	Description
type	String	Event type `StateChange`.
timestamp	Timestamp	Unix timestamp when the event happened.
host	String	Host name.
service	String	Service name. Optional if this is a host state change.
state	Number	Host or service state.
state_type	Number	Host or service state type.
check_result	CheckResult	Serialized CheckResult value type.

12.7.1.3 Event Stream Type: Notification

Name	Type	Description
type	String	Event type `Notification`.
timestamp	Timestamp	Unix timestamp when the event happened.
host	String	Host name.
service	String	Service name. Optional if this is a host notification.
users	Array	List of notified user names.
notification_type	String	*notification.type* runtime macro value.
author	String	*notification.author* runtime macro value.
text	String	*notification.comment* runtime macro value.
check_result	CheckResult	Serialized CheckResult value type.

12.7.1.4 Event Stream Type: Flapping

Name	Type	Description
type	String	Event type `Flapping`.
timestamp	Timestamp	Unix timestamp when the event happened.
host	String	Host name.
service	String	Service name. Optional if this is a host flapping event.
state	Number	Host or service state.
state_type	Number	Host or service state type.
is_flapping	Boolean	Whether this object is flapping.
current_flapping	Number	Current flapping value in percent (added in 2.8).
threshold_low	Number	Low threshold in percent (added in 2.8).
threshold_high	Number	High threshold in percent (added in 2.8).

12.7.1.5 Event Stream Type: AcknowledgementSet

Name	Type	Description
type	String	Event type `AcknowledgementSet`.
timestamp	Timestamp	Unix timestamp when the event happened.
host	String	Host name.
service	String	Service name. Optional if this is a host acknowledgement.
state	Number	Host or service state.
state_type	Number	Host or service state type.
author	String	Acknowledgement author set via acknowledge-problem action.
comment	String	Acknowledgement comment set via acknowledge-problem action.
acknowledgement_type	Number	0 = None, 1 = Normal, 2 = Sticky. `sticky` can be set via acknowledge-problem action.
notify	Boolean	Notifications were enabled via acknowledge-problem action.

Name	Type	Description
expiry	Timestamp	Acknowledgement expire time set via acknowledge-problem action.

12.7.1.6 Event Stream Type: AcknowledgementCleared

Name	Type	Description
type	String	Event type `AcknowledgementCleared`.
timestamp	Timestamp	Unix timestamp when the event happened.
host	String	Host name.
service	String	Service name. Optional if this is a host acknowledgement.
state	Number	Host or service state.
state_type	Number	Host or service state type.

12.7.1.7 Event Stream Type: CommentAdded

Name	Type	Description
type	String	Event type `CommentAdded`.
timestamp	Timestamp	Unix timestamp when the event happened.
comment	Dictionary	Serialized Comment object.

12.7.1.8 Event Stream Type: CommentRemoved

Name	Type	Description
type	String	Event type `CommentRemoved`.
timestamp	Timestamp	Unix timestamp when the event happened.
comment	Dictionary	Serialized Comment object.

12.7.1.9 Event Stream Type: DowntimeAdded

Name	Type	Description
type	String	Event type `DowntimeAdded`.
timestamp	Timestamp	Unix timestamp when the event happened.
downtime	Dictionary	Serialized Comment object.

12.7.1.10 Event Stream Type: DowntimeRemoved

Name	Type	Description
type	String	Event type `DowntimeRemoved`.
timestamp	Timestamp	Unix timestamp when the event happened.
downtime	Dictionary	Serialized Comment object.

12.7.1.11 Event Stream Type: DowntimeStarted

Name	Type	Description
type	String	Event type `DowntimeStarted`.
timestamp	Timestamp	Unix timestamp when the event happened.
downtime	Dictionary	Serialized Comment object.

12.7.1.12 Event Stream Type: DowntimeTriggered

Name	Type	Description
type	String	Event type `DowntimeTriggered`.
timestamp	Timestamp	Unix timestamp when the event happened.
downtime	Dictionary	Serialized Comment object.

12.7.2 Event Stream Filter

Event streams can be filtered by attributes using the prefix `event.`.

Example for the `CheckResult` type with the `exit_code` set to 2:

```
&types=CheckResult&filter=event.check_result.exit_status==2
```

Example for the `CheckResult` type with the service matching the string pattern "random*":

```
&types=CheckResult&filter=match%28%22random*%22,event.service%29
```

12.7.3 Event Stream Response

The event stream response is separated with new lines. The HTTP client must support long-polling and HTTP/1.1. HTTP/1.0 is not supported.

Example:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ev
```

```
{"check_result":{ ... },"host":"example.localdomain","service":"ping4","timestamp":144542131
{"check_result":{ ... },"host":"example.localdomain","service":"ping4","timestamp":144542132
{"check_result":{ ... },"host":"example.localdomain","service":"ping4","timestamp":144542132
```


12.8 Status and Statistics

Send a `GET` request to the URL endpoint `/v1/status` to retrieve status information and statistics for Icinga 2.

Example:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/status?pretty=1'
{
    "results": [
        {
            "name": "ApiListener",
            "perfdata": [ ... ],
            "status": [ ... ]
        },
        ...
        {
            "name": "IcingaAplication",
            "perfdata": [ ... ],
            "status": [ ... ]
        },
        ...
    ]
}
```

You can limit the output by specifying a status type in the URL, e.g. `IcingaApplication`:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/status/IcingaApplication?pretty=1'
{
    "results": [
        {
            "perfdata": [],
            "status": {
                "icingaapplication": {
                    "app": {
                        "enable_event_handlers": true,
                        "enable_flapping": true,
                        "enable_host_checks": true,
                        "enable_notifications": true,
                        "enable_perfdata": true,
                        "enable_service_checks": true,
                        "node_name": "example.localdomain",
```

```
                        "pid": 59819.0,
                        "program_start": 1443019345.093372,
                        "version": "v2.3.0-573-g380a131"
                    }
                }
            }
        }
    ]
}
```


12.9 Configuration Management

The main idea behind configuration management is to allow external applications creating configuration packages and stages based on configuration files and directory trees. This replaces any additional SSH connection and whatnot to dump configuration files to Icinga 2 directly. In case you are pushing a new configuration stage to a package, Icinga 2 will validate the configuration asynchronously and populate a status log which can be fetched in a separated request.

12.9.1 Creating a Config Package

Send a `POST` request to a new config package called `example-cmdb` in this example. This will create a new empty configuration package.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST \
'https://localhost:5665/v1/config/packages/example-cmdb?pretty=1'
{
    "results": [
        {
            "code": 200.0,
            "package": "example-cmdb",
            "status": "Created package."
        }
    ]
}
```

Package names starting with an underscore are reserved for internal packages and must not be used.

12.9.2 Uploading configuration for a Config Package

Configuration files in packages are managed in stages. Stages provide a way to maintain multiple configuration versions for a package.

Send a `POST` request to the URL endpoint `/v1/config/stages` and add the name of an existing configuration package to the URL path (e.g. `example-cmdb`). The request body must contain the `files` attribute with the value being a dictionary of file targets and their content. You can also specify an optional `reload` attribute that will tell icinga2 to reload after stage config validation. By default this is set to `true`.

The file path requires one of these two directories inside its path:

Directory	Description
conf.d	Local configuration directory.
zones.d	Configuration directory for cluster zones, each zone must be put into its own zone directory underneath. Supports the cluster config sync.

Example for a local configuration in the `conf.d` directory:

```
"files": { "conf.d/host1.conf": "object Host \"local-host\" { address = \"127.0.0.1\", check_co
```

Example for a host configuration inside the `satellite` zone in the `zones.d` directory:

```
"files": { "zones.d/satellite/host2.conf": "object Host \"satellite-host\" { address = \"192.16
```

The example below will create a new file called `test.conf` in the `conf.d` directory. Note: This example contains an error (`chec_command`). This is intentional.

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST \
-d '{ "files": { "conf.d/test.conf": "object Host \"cmdb-host\" { chec_command = \"dummy\" }" },
'https://localhost:5665/v1/config/stages/example-cmdb'
{
    "results": [
        {
            "code": 200.0,
            "package": "example-cmdb",
            "stage": "example.localdomain-1441625839-0",
            "status": "Created stage. Icinga2 will reload."
        }
    ]
}
```

The Icinga 2 API returns the `package` name this stage was created for, and also generates a unique name for the `stage` attribute you'll need for later requests.

Icinga 2 automatically restarts the daemon in order to activate the new config stage. This can be disabled by setting `reload` to `false` in the request. If the

validation for the new config stage failed, the old stage and its configuration objects will remain active.

Note

Old stages are not purged automatically. You can remove stages that are no longer in use.

Icinga 2 will create the following files in the configuration package stage after configuration validation:

File	Description
status	Contains the configuration validation exit code (everything else than 0 indicates an error).
startup.log	Contains the configuration validation output.

You can fetch these files in order to verify that the new configuration was deployed successfully.

12.9.3 List Configuration Packages and their Stages

A list of packages and their stages can be retrieved by sending a `GET` request to the URL endpoint `/v1/config/packages`.

The following example contains one configuration package `example-cmdb`. The package does not currently have an active stage.

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/config/packages?pretty=1'
{
    "results": [
        {
            "active-stage": "",
            "name": "example-cmdb",
            "stages": [
                "example.localdomain-1441625839-0"
            ]
        }
```

```
    ]
}
```

12.9.4 List Configuration Packages and their Stages

In order to retrieve a list of files for a stage you can send a `GET` request to the URL endpoint `/v1/config/stages`. You need to include the package name (`example-cmdb`) and stage name (`example.localdomain-1441625839-0`) in the URL:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/config/stages/example-cmdb/example.loca
{
    "results": [
...
        {
            "name": "startup.log",
            "type": "file"
        },
        {
            "name": "status",
            "type": "file"
        },
        {
            "name": "conf.d",
            "type": "directory"
        },
        {
            "name": "zones.d",
            "type": "directory"
        },
        {
            "name": "conf.d/test.conf",
            "type": "file"
        }
    ]
}
```

12.9.5 Fetch Configuration Package Stage Files

Send a `GET` request to the URL endpoint `/v1/config/files` and add the package name, the stage name and the relative path to the file to the URL path.

Note

The returned files are plain-text instead of JSON-encoded.

The following example fetches the configuration file `conf.d/test.conf`:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/config/files/example-cmdb/example.local

object Host "cmdb-host" { chec_command = "dummy" }
```

You can fetch a list of existing files in a configuration stage and then specifically request their content.

12.9.6 Configuration Package Stage Errors

Now that we don't have an active stage for `example-cmdb` yet seen here, there must have been an error.

In order to check for validation errors you can fetch the `startup.log` file by sending a `GET` request to the URL endpoint `/v1/config/files`. You must include the package name, stage name and the `startup.log` in the URL path.

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/config/files/example-cmdb/example.local
...

critical/config: Error: Attribute 'chec_command' does not exist.
Location:
/var/lib/icinga2/api/packages/example-cmdb/example.localdomain-1441133065-1/conf.d/test.conf
                                                                              ^^^^^^^^^^^^^^^^^^^^^^^


critical/config: 1 error
```

The output is similar to the manual configuration validation.

> **Note**
>
> The returned output is plain-text instead of JSON-encoded.

12.9.7 Deleting Configuration Package Stage

You can send a `DELETE` request to the URL endpoint `/v1/config/stages` in order to purge a configuration stage. You must include the package and stage name inside the URL path.

The following example removes the failed configuration stage `example.localdomain-1441133065-1` in the `example-cmdb` configuration package:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X DELETE \
'https://localhost:5665/v1/config/stages/example-cmdb/example.localdomain-1441133065-1?pretty
{
    "results": [
        {
            "code": 200.0,
```

```
            "status": "Stage deleted."
        }
    ]
}
```

12.9.8 Deleting Configuration Package

In order to completely purge a configuration package and its stages you can send a `DELETE` request to the URL endpoint `/v1/config/packages` with the package name in the URL path.

This example entirely deletes the configuration package `example-cmdb`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X DELETE \
'https://localhost:5665/v1/config/packages/example-cmdb?pretty=1'
{
    "results": [
        {
            "code": 200.0,
            "package": "example-cmdb",
            "status": "Deleted package."
        }
    ]
}
```

12.10 Types

You can retrieve the configuration object types by sending a `GET` request to URL endpoint `/v1/types`.

Each response entry in the results array contains the following attributes:

Attribute	Type	Description
name	String	The type name.
plural_name	String	The plural type name.
fields	Dictionary	Available fields including details on e.g. the type and attribute accessibility.
abstract	Boolean	Whether objects can be instantiated for this type.

Attribute	Type	Description
base	Boolean	The base type (e.g. `Service` inherits fields and prototype methods from `Checkable`).
prototype_keys	Array	Available prototype methods.

In order to view a specific configuration object type specify its name inside the URL path:

```
$ curl -k -s -u root:icinga 'https://localhost:5665/v1/types/Object?pretty=1'
{
    "results": [
        {
            "abstract": false,
            "fields": {
                "type": {
                    "array_rank": 0.0,
                    "attributes": {
                        "config": false,
                        "navigation": false,
                        "no_user_modify": false,
                        "no_user_view": false,
                        "required": false,
                        "state": false
                    },
                    "id": 0.0,
                    "type": "String"
                }
            },
            "name": "Object",
            "plural_name": "Objects",
            "prototype_keys": [
                "clone",
                "notify_attribute",
                "to_string"
            ]
        }
    ]
}
```

12.11 Console

You can inspect variables and execute other expressions by sending a `POST` request to the URL endpoint `/v1/console/execute-script`. In order to receive auto-completion suggestions, send a `POST` request to the URL endpoint `/v1/console/auto-complete-script`.

The following parameters need to be specified (either as URL parameters or in a JSON-encoded message body):

Parameter	Type	Description
session	String	**Optional.** The session ID. Ideally this should be a GUID or some other unique identifier.
command	String	**Required.** Command expression for execution or auto-completion.
sandboxed	Number	**Optional.** Whether runtime changes are allowed or forbidden. Defaults to disabled.

The API permission `console` is required for executing expressions.

> **Note**
>
> Runtime modifications via `execute-script` calls are not validated and might cause the Icinga 2 daemon to crash or behave in an unexpected way. Use these runtime changes at your own risk.

If you specify a session identifier, the same script context can be reused for multiple requests. This allows you to, for example, set a local variable in a request and use that local variable in another request. Sessions automatically expire after a set period of inactivity (currently 30 minutes).

Example for fetching the command line from the local host's last check result:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/co
{
    "results": [
        {
            "code": 200.0,
            "result": [
                "/usr/local/sbin/check_ping",
                "-H",
                "127.0.0.1",
                "-c",
                "5000,100%",
                "-w",
                "3000,80%"
            ],
            "status": "Executed successfully."
        }
    ]
}
```

Example for fetching auto-completion suggestions for the `Host.` type. This works in a similar fashion when pressing TAB inside the console CLI command:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/co
{
    "results": [
        {
            "code": 200.0,
            "status": "Auto-completed successfully.",
            "suggestions": [
                "Host.type",
                "Host.name",
                "Host.prototype",
                "Host.base",
                "Host.register_attribute_handler",
                "Host.clone",
                "Host.notify_attribute",
                "Host.to_string"
            ]
        }
    ]
}
```

12.12 API Clients

There are a couple of existing clients which can be used with the Icinga 2 API:

- curl or any other HTTP client really
- Icinga 2 console (CLI command)
- Icinga Web 2 Director

Demo cases:

- Dashing
- API examples

Additional programmatic examples will help you getting started using the Icinga 2 API in your environment.

12.12.1 Icinga 2 Console

By default the console CLI command evaluates expressions in a local interpreter, i.e. independently from your Icinga 2 daemon. Add the `--connect` parameter to debug and evaluate expressions via the API.

12.12.2 API Clients Programmatic Examples

The programmatic examples use HTTP basic authentication and SSL certificate verification. The CA file is expected in `pki/icinga2-ca.crt` but you may adjust the examples for your likings.

The request method is `POST` using X-HTTP-Method-Override: GET which allows you to send a JSON request body. The examples request specific service attributes joined with host attributes. `attrs` and `joins` are therefore specified as array. The `filter` attribute matches on all services with `ping` in their name.

12.12.2.1 Example API Client in Python

The following example uses **Python** and the `requests` and `json` module:

```
# pip install requests
# pip install json

$ vim icinga2-api-example.py

#!/usr/bin/env python

import requests, json

# Replace 'localhost' with your FQDN and certificate CN
```

```
# for SSL verification
request_url = "https://localhost:5665/v1/objects/services"
headers = {
        'Accept': 'application/json',
        'X-HTTP-Method-Override': 'GET'
        }
data = {
        "attrs": [ "name", "state", "last_check_result" ],
     "joins": [ "host.name", "host.state", "host.last_check_result" ],
        "filter": "match(\"ping*\", service.name)",
}

r = requests.post(request_url,
        headers=headers,
        auth=('root', 'icinga'),
        data=json.dumps(data),
        verify="pki/icinga2-ca.crt")

print "Request URL: " + str(r.url)
print "Status code: " + str(r.status_code)

if (r.status_code == 200):
        print "Result: " + json.dumps(r.json())
else:
        print r.text
        r.raise_for_status()

$ python icinga2-api-example.py
```

12.12.2.2 Example API Client in Ruby

The following example uses **Ruby** and the `rest_client` gem:

```
# gem install rest_client

$ vim icinga2-api-example.rb

#!/usr/bin/ruby

require 'rest_client'

# Replace 'localhost' with your FQDN and certificate CN
# for SSL verification
request_url = "https://localhost:5665/v1/objects/services"
headers = {
        "Accept" => "application/json",
```

```
        "X-HTTP-Method-Override" => "GET"
}
data = {
        "attrs" => [ "name", "state", "last_check_result" ],
     "joins" => [ "host.name", "host.state", "host.last_check_result" ],
        "filter" => "match(\"ping*\", service.name)",
}

r = RestClient::Resource.new(
        URI.encode(request_url),
        :headers => headers,
        :user => "root",
        :password => "icinga",
        :ssl_ca_file => "pki/icinga2-ca.crt")

begin
        response = r.post(data.to_json)
rescue => e
        response = e.response
end

puts "Status: " + response.code.to_s
if response.code == 200
     puts "Result: " + (JSON.pretty_generate JSON.parse(response.body))
else
        puts "Error: " + response
end
```

```
$ ruby icinga2-api-example.rb
```

A more detailed example can be found in the Dashing demo.

12.12.2.3 Example API Client in PHP

The following example uses **PHP** and its `curl` library:

```
$ vim icinga2-api-example.php
```

```
#!/usr/bin/env php
<?php
# Replace 'localhost' with your FQDN and certificate CN
# for SSL verification
$request_url = "https://localhost:5665/v1/objects/services";
$username = "root";
$password = "icinga";
$headers = array(
```

```
        'Accept: application/json',
        'X-HTTP-Method-Override: GET'
);
$data = array(
        attrs => array('name', 'state', 'last_check_result'),
      joins => array('host.name', 'host.state', 'host.last_check_result'),
        filter => 'match("ping*", service.name)',
);

$ch = curl_init();
curl_setopt_array($ch, array(
        CURLOPT_URL => $request_url,
        CURLOPT_HTTPHEADER => $headers,
        CURLOPT_USERPWD => $username . ":" . $password,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_CAINFO => "pki/icinga2-ca.crt",
        CURLOPT_POST => count($data),
        CURLOPT_POSTFIELDS => json_encode($data)
));

$response = curl_exec($ch);
if ($response === false) {
       print "Error: " . curl_error($ch) . "(" . $response . ")\n";
}

$code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
curl_close($ch);
print "Status: " . $code . "\n";

if ($code == 200) {
        $response = json_decode($response, true);
        print_r($response);
}
?>

$ php icinga2-api-example.php
```

12.12.2.4 Example API Client in Perl

The following example uses **Perl** and the `Rest::Client` module:

```
# perl -MCPAN -e 'install REST::Client'
# perl -MCPAN -e 'install JSON'
# perl -MCPAN -e 'install MIME::Base64'
# perl -MCPAN -e 'install Data::Dumper'
```

```
$ vim icinga2-api-example.pl

#!/usr/bin/env perl

use strict;
use warnings;
use REST::Client;
use MIME::Base64;
use JSON;
use Data::Dumper;

# Replace 'localhost' with your FQDN and certificate CN
# for SSL verification
my $request_host = "https://localhost:5665";
my $userpass = "root:icinga";

my $client = REST::Client->new();
$client->setHost($request_host);
$client->setCa("pki/icinga2-ca.crt");
$client->addHeader("Accept", "application/json");
$client->addHeader("X-HTTP-Method-Override", "GET");
$client->addHeader("Authorization", "Basic " . encode_base64($userpass));
my %json_data = (
        attrs => ['name', 'state', 'last_check_result'],
     joins => ['host.name', 'host.state', 'host.last_check_result'],
        filter => 'match("ping*", service.name)',
);
my $data = encode_json(\%json_data);
$client->POST("/v1/objects/services", $data);

my $status = $client->responseCode();
print "Status: " . $status . "\n";
my $response = $client->responseContent();
if ($status == 200) {
        print "Result: " . Dumper(decode_json($response)) . "\n";
} else {
        print "Error: " . $response . "\n";
}

$ perl icinga2-api-example.pl
```

13 Icinga 2 Addons

13.1 Graphing

13.1.1 PNP

PNP is a graphing addon.

PNP is an addon which adds a graphical representation of the performance data collected by the monitoring plugins. The data is stored as rrd (round robin database) files.

Use your distribution's package manager to install the `pnp4nagios` package.

If you're planning to use it, configure it to use the bulk mode with npcd and npcdmod in combination with Icinga 2's PerfdataWriter. NPCD collects the performance data files which Icinga 2 generates.

Enable performance data writer in icinga 2

```
# icinga2 feature enable perfdata
```

Configure npcd to use the performance data created by Icinga 2:

```
vim /etc/pnp4nagios/npcd.cfg
```

Set `perfdata_spool_dir = /var/spool/icinga2/perfdata` and restart the `npcd` daemon.

There's also an Icinga Web 2 module for direct PNP graph integration available at Icinga Exchange.

More information on action_url as attribute and graph template names.

13.1.2 Graphite

Graphite is a time-series database storing collected metrics and making them available through restful apis and web interfaces.

Graphite consists of 3 software components:

- carbon – a Twisted daemon that listens for time-series data
- whisper – a simple database library for storing time-series data (similar in design to RRD)
- graphite webapp – a Django webapp that renders graphs on-demand using Cairo

Use the GraphiteWriter feature for sending real-time metrics from Icinga 2 to Graphite.

```
# icinga2 feature enable graphite
```

There are Graphite addons available for collecting the performance data files too (e.g. `Graphios`).

A popular alternative frontend for Graphite is for example Grafana.

13.1.3 InfluxDB

InfluxDB is a time series, metrics, and analytics database. It's written in Go and has no external dependencies.

Use the InfluxdbWriter feature for sending real-time metrics from Icinga 2 to InfluxDB.

```
# icinga2 feature enable influxdb
```

A popular frontend for InfluxDB is for example Grafana.

13.2 Visualization

13.2.1 Icinga Reporting

By enabling the DB IDO feature you can use the Icinga Reporting package.

13.2.2 NagVis

By using either Livestatus or DB IDO as a backend you can create your own network maps based on your monitoring configuration and status data using NagVis.

The configuration in nagvis.ini.php should look like this for Livestatus for example:

```
[backend_live_1]
backendtype="mklivestatus"
socket="unix:/var/run/icinga2/cmd/livestatus"
```

If you are planning an integration into Icinga Web 2, look at this module.

13.2.3 Thruk

Thruk is an alternative web interface which can be used with Icinga 2 and the Livestatus feature.

13.3 Log Monitoring

Using Logstash or Graylog in your infrastructure and correlate events with your monitoring is even simpler these days.

- Use the `GelfWriter` feature to write Icinga 2's check and notification events to Graylog or Logstash.
- Configure the logstash `nagios` output to send passive traps to Icinga 2 using the external command pipe.
- Execute a plugin to check Graylog alert streams.

More details can be found in this blog post.

13.4 Notification Scripts and Interfaces

There's a variety of resources available, for example different notification scripts such as:

- E-Mail (examples provided)
- SMS
- Pager (XMPP, etc.)
- Twitter
- IRC
- Ticket systems
- etc.

Additionally external services can be integrated with Icinga 2:

- Pagerduty
- VictorOps
- StackStorm

More information can be found on the Icinga Website.

13.5 Configuration Management Tools

If you require your favourite configuration tool to export the Icinga 2 configuration, please get in touch with their developers. The Icinga project does not provide a configuration web interface yet. Follow the Icinga Blog for updates on this topic.

If you're looking for puppet manifests, chef cookbooks, ansible recipes, etc. – we're happy to integrate them upstream, so please get in touch with the Icinga team.

These tools are currently in development and require feedback and tests:

- Ansible Roles

- Puppet Module
- Chef Cookbook

13.6 More Addon Integration Hints

13.6.1 PNP Action Url

They work in a similar fashion for Icinga 2 and are used for 1.x web interfaces (Icinga Web 2 doesn't require the action url attribute in its own module).

```
template Host "pnp-hst" {
  action_url = "/pnp4nagios/graph?host=$HOSTNAME$"
}

template Service "pnp-svc" {
 action_url = "/pnp4nagios/graph?host=$HOSTNAME$&srv=$SERVICEDESC$"
}
```

13.6.2 PNP Custom Templates with Icinga 2

PNP automatically determines the graph template from the check command name (or the argument's name). This behavior changed in Icinga 2 compared to Icinga 1.x. Though there are certain possibilities to fix this:

- Create a symlink for example from the `templates.dist/check_ping.php` template to the actual check name in Icinga 2 (`templates/ping4.php`)
- Pass the check command name inside the format template configuration

The latter becomes difficult with agent based checks like NRPE or SSH where the first command argument acts as graph template identifier. There is the possibility to define the pnp template name as custom attribute and use that inside the formatting templates as `SERVICECHECKCOMMAND` for instance.

Example for services:

```
# vim /etc/icinga2/features-enabled/perfdata.conf

service_format_template = "DATATYPE::SERVICEPERFDATA\tTIMET::$icinga.timet$\tHOSTNAME::$host

# vim /etc/icinga2/conf.d/services.conf

template Service "pnp-svc" {
 action_url = "/pnp4nagios/graph?host=$HOSTNAME$&srv=$SERVICEDESC$"
  vars.pnp_check_arg1 = ""
}
```

```
apply Service "nrpe-check" {
  import "pnp-svc"
  check_command = nrpe
  vars.nrpe_command = "check_disk"

  vars.pnp_check_arg1 = "!$nrpe_command$"
}
```

If there are warnings about unresolved macros, make sure to specify a default value for `vars.pnp_check_arg1` inside the

In PNP, the custom template for nrpe is then defined in `/etc/pnp4nagios/custom/nrpe.cfg` and the additional command arg string will be seen in the xml too for other templates.

14 Icinga 2 Features

14.1 Logging

Icinga 2 supports three different types of logging:

- File logging
- Syslog (on Linux/UNIX)
- Console logging (`STDOUT` on tty)

You can enable additional loggers using the `icinga2 feature enable` and `icinga2 feature disable` commands to configure loggers:

Feature	Description
debuglog	Debug log (path: `/var/log/icinga2/debug.log`, severity: `debug` or higher)
mainlog	Main log (path: `/var/log/icinga2/icinga2.log`, severity: `information` or higher)
syslog	Syslog (severity: `warning` or higher)

By default file the `mainlog` feature is enabled. When running Icinga 2 on a terminal log messages with severity `information` or higher are written to the console.

Packages will install a configuration file for logrotate on supported platforms. This configuration ensures that the `icinga2.log`, `error.log` and `debug.log` files are rotated on a daily basis.

14.2 DB IDO

The IDO (Icinga Data Output) feature for Icinga 2 takes care of exporting all configuration and status information into a database. The IDO database is used by Icinga Web 2 as data backend.

Details on the installation can be found in the Configuring DB IDO chapter. Details on the configuration can be found in the IdoMysqlConnection and IdoPgsqlConnection object configuration documentation. The DB IDO feature supports High Availability in the Icinga 2 cluster.

14.2.1 DB IDO Health

If the monitoring health indicator is critical in Icinga Web 2, you can use the following queries to manually check whether Icinga 2 is actually updating the IDO database.

Icinga 2 writes its current status to the `icinga_programstatus` table every 10 seconds. The query below checks 60 seconds into the past which is a reasonable amount of time – adjust it for your requirements. If the condition is not met, the query returns an empty result.

Tip

Use check plugins to monitor the backend.

Replace the `default` string with your instance name if different.

Example for MySQL:

```
# mysql -u root -p icinga -e "SELECT status_update_time FROM icinga_programstatus ps
  JOIN icinga_instances i ON ps.instance_id=i.instance_id
 WHERE (UNIX_TIMESTAMP(ps.status_update_time) > UNIX_TIMESTAMP(NOW())-60)
  AND i.instance_name='default';"

+---------------------+
| status_update_time  |
+---------------------+
| 2014-05-29 14:29:56 |
+---------------------+
```

Example for PostgreSQL:

```
# export PGPASSWORD=icinga; psql -U icinga -d icinga -c "SELECT ps.status_update_time FROM icing
  JOIN icinga_instances AS i ON ps.instance_id=i.instance_id
 WHERE ((SELECT extract(epoch from status_update_time) FROM icinga_programstatus) > (SELECT ex
  AND i.instance_name='default'";

status_update_time
------------------------
 2014-05-29 15:11:38+02
(1 Zeile)
```

A detailed list on the available table attributes can be found in the DB IDO Schema documentation.

14.2.2 DB IDO Tuning

As with any application database, there are ways to optimize and tune the database performance.

General tips for performance tuning:

- MariaDB KB
- PostgreSQL Wiki

Re-creation of indexes, changed column values, etc. will increase the database size. Ensure to add health checks for this, and monitor the trend in your Grafana dashboards.

In order to optimize the tables, there are different approaches. Always keep in mind to have a current backup and schedule maintenance downtime for these kind of tasks!

MySQL:

```
mariadb> OPTIMIZE TABLE icinga_statehistory;
```

> **Important**
>
> Tables might not support optimization at runtime. This can take a **long** time.
>
> `Table does not support optimize, doing recreate + analyze instead`.

If you want to optimize all tables in a specified database, there is a script called `mysqlcheck`. This also allows to repair broken tables in the case of emergency.

```
mysqlcheck --optimize icinga
```

PostgreSQL:

```
icinga=# vacuum;
VACUUM
```

Note

Don't use `VACUUM FULL` as this has a severe impact on performance.

14.3 External Commands

Note

Please use the REST API as modern and secure alternative for external actions.

Icinga 2 provides an external command pipe for processing commands triggering specific actions (for example rescheduling a service check through the web interface).

In order to enable the `ExternalCommandListener` configuration use the following command and restart Icinga 2 afterwards:

```
# icinga2 feature enable command
```

Icinga 2 creates the command pipe file as `/var/run/icinga2/cmd/icinga2.cmd` using the default configuration.

Web interfaces and other Icinga addons are able to send commands to Icinga 2 through the external command pipe, for example for rescheduling a forced service check:

```
# /bin/echo "[`date +%s`] SCHEDULE_FORCED_SVC_CHECK;localhost;ping4;`date +%s`" >> /var/run/ic

# tail -f /var/log/messages

Oct 17 15:01:25 icinga-server icinga2: Executing external command: [1382014885] SCHEDULE_FORCED
Oct 17 15:01:25 icinga-server icinga2: Rescheduling next check for service 'ping4'
```

A list of currently supported external commands can be found here.

Detailed information on the commands and their required parameters can be found on the Icinga 1.x documentation.

14.4 Performance Data

When a host or service check is executed plugins should provide so-called `performance data`. Next to that additional check performance data can be fetched using Icinga 2 runtime macros such as the check latency or the current service state (or additional custom attributes).

The performance data can be passed to external applications which aggregate and store them in their backends. These tools usually generate graphs for historical reporting and trending.

Well-known addons processing Icinga performance data are PNP4Nagios, Graphite or OpenTSDB.

14.4.1 Writing Performance Data Files

PNP4Nagios and Graphios use performance data collector daemons to fetch the current performance files for their backend updates.

Therefore the Icinga 2 PerfdataWriter feature allows you to define the output template format for host and services helped with Icinga 2 runtime vars.

```
host_format_template = "DATATYPE::HOSTPERFDATA\tTIMET::$icinga.timet$\tHOSTNAME::$host.name$
service_format_template = "DATATYPE::SERVICEPERFDATA\tTIMET::$icinga.timet$\tHOSTNAME::$host
```

The default templates are already provided with the Icinga 2 feature configuration which can be enabled using

```
# icinga2 feature enable perfdata
```

By default all performance data files are rotated in a 15 seconds interval into the `/var/spool/icinga2/perfdata/` directory as `host-perfdata.<timestamp>` and `service-perfdata.<timestamp>`. External collectors need to parse the rotated performance data files and then remove the processed files.

14.4.2 Graphite Carbon Cache Writer

While there are some Graphite collector scripts and daemons like Graphios available for Icinga 1.x it's more reasonable to directly process the check and plugin performance in memory in Icinga 2. Once there are new metrics available, Icinga 2 will directly write them to the defined Graphite Carbon daemon tcp socket.

You can enable the feature using

```
# icinga2 feature enable graphite
```

By default the GraphiteWriter feature expects the Graphite Carbon Cache to listen at `127.0.0.1` on TCP port `2003`.

14.4.2.1 Current Graphite Schema

The current naming schema is defined as follows. The Icinga Web 2 Graphite module depends on this schema.

The default prefix for hosts and services is configured using runtime macroslike this:

```
icinga2.$host.name$.host.$host.check_command$
icinga2.$host.name$.services.$service.name$.$service.check_command$
```

You can customize the prefix name by using the `host_name_template` and `service_name_template` configuration attributes.

The additional levels will allow fine granular filters and also template capabilities, e.g. by using the check command `disk` for specific graph templates in web applications rendering the Graphite data.

The following characters are escaped in prefix labels:

Character	Escaped character
whitespace	_
.	_
	_
/	_

Metric values are stored like this:

```
<prefix>.perfdata.<perfdata-label>.value
```

The following characters are escaped in perfdata labels:

Character	Escaped character
whitespace	_
	_
/	_
::	.

Note that perfdata labels may contain dots (`.`) allowing to add more subsequent levels inside the Graphite tree. `::` adds support for multi performance labels and is therefore replaced by `..`

By enabling `enable_send_thresholds` Icinga 2 automatically adds the following threshold metrics:

```
<prefix>.perfdata.<perfdata-label>.min
<prefix>.perfdata.<perfdata-label>.max
<prefix>.perfdata.<perfdata-label>.warn
<prefix>.perfdata.<perfdata-label>.crit
```

By enabling `enable_send_metadata` Icinga 2 automatically adds the following metadata metrics:

```
<prefix>.metadata.current_attempt
<prefix>.metadata.downtime_depth
<prefix>.metadata.acknowledgement
<prefix>.metadata.execution_time
<prefix>.metadata.latency
<prefix>.metadata.max_check_attempts
<prefix>.metadata.reachable
<prefix>.metadata.state
<prefix>.metadata.state_type
```

Metadata metric overview:

metric	description
current_attempt	current check attempt
max_check_attempts	maximum check attempts until the hard state is reached
reachable	checked object is reachable
downtime_depth	number of downtimes this object is in
acknowledgement	whether the object is acknowledged or not
execution_time	check execution time
latency	check latency
state	current state of the checked object
state_type	0=SOFT, 1=HARD state

The following example illustrates how to configure the storage schemas for Graphite Carbon Cache.

```
[icinga2_default]
# intervals like PNP4Nagios uses them per default
pattern = ^icinga2\.
retentions = 1m:2d,5m:10d,30m:90d,360m:4y
```

14.4.3 InfluxDB Writer

Once there are new metrics available, Icinga 2 will directly write them to the defined InfluxDB HTTP API.

You can enable the feature using

```
# icinga2 feature enable influxdb
```

By default the InfluxdbWriter feature expects the InfluxDB daemon to listen at `127.0.0.1` on port `8086`.

Measurement names and tags are fully configurable by the end user. The InfluxdbWriter object will automatically add a `metric` tag to each data point. This correlates to the perfdata label. Fields (value, warn, crit, min, max, unit) are created from data if available and the configuration allows it. If a value

associated with a tag is not able to be resolved, it will be dropped and not sent to the target host.

Backslashes are allowed in tag keys, tag values and field keys, however they are also escape characters when followed by a space or comma, but cannot be escaped themselves. As a result all trailling slashes in these fields are replaced with an underscore. This predominantly affects Windows paths e.g. `C:\` becomes `C:_`.

The database is assumed to exist so this object will make no attempt to create it currently.

If SELinux is enabled, it will not allow access for Icinga 2 to InfluxDB until the boolean `icinga2_can_connect_all` is set to true as InfluxDB is not providing its own policy.

More configuration details can be found here.

14.4.3.1 Instance Tagging

Consider the following service check:

```
apply Service "disk" for (disk => attributes in host.vars.disks) {
  import "generic-service"
  check_command = "disk"
  display_name = "Disk " + disk
  vars.disk_partitions = disk
  assign where host.vars.disks
}
```

This is a typical pattern for checking individual disks, NICs, SSL certificates etc associated with a host. What would be useful is to have the data points tagged with the specific instance for that check. This would allow you to query time series data for a check on a host and for a specific instance e.g. /dev/sda. To do this quite simply add the instance to the service variables:

```
apply Service "disk" for (disk => attributes in host.vars.disks) {
  ...
  vars.instance = disk
  ...
}
```

Then modify your writer configuration to add this tag to your data points if the instance variable is associated with the service:

```
object InfluxdbWriter "influxdb" {
  ...
  service_template = {
    measurement = "$service.check_command$"
    tags = {
```

```
      hostname = "$host.name$"
      service = "$service.name$"
      instance = "$service.vars.instance$"
    }
  }
  ...
}
```

14.4.4 Elastic Stack Integration

Icingabeat is an Elastic Beat that fetches data from the Icinga 2 API and sends it either directly to Elasticsearch or Logstash.

More integrations:

- Logstash output for the Icinga 2 API.
- Logstash Grok Pattern for Icinga 2 logs.

14.4.4.1 Elasticsearch Writer

This feature forwards check results, state changes and notification events to an Elasticsearch installation over its HTTP API.

The check results include parsed performance data metrics if enabled.

> **Note**
>
> Elasticsearch 5.x is required. This feature has been successfully tested with Elasticsearch 5.6.4.

Enable the feature and restart Icinga 2.

```
# icinga2 feature enable elasticsearch
```

The default configuration expects an Elasticsearch instance running on `localhost` on port `9200` and writes to an index called`icinga2`.

More configuration details can be found here.

14.4.4.2 Current Elasticsearch Schema

The following event types are written to Elasticsearch:

- icinga2.event.checkresult
- icinga2.event.statechange
- icinga2.event.notification

Performance data metrics must be explicitly enabled with the `enable_send_perfdata` attribute.

Metric values are stored like this:

`check_result.perfdata.<perfdata-label>.value`

The following characters are escaped in perfdata labels:

Character	Escaped character
whitespace	_
	_
/	_
::	.

Note that perfdata labels may contain dots (`.`) allowing to add more subsequent levels inside the tree. `::` adds support for multi performance labels and is therefore replaced by `..`

Icinga 2 automatically adds the following threshold metrics if existing:

```
check_result.perfdata.<perfdata-label>.min
check_result.perfdata.<perfdata-label>.max
check_result.perfdata.<perfdata-label>.warn
check_result.perfdata.<perfdata-label>.crit
```

14.4.5 Graylog Integration

14.4.5.1 GELF Writer

The `Graylog Extended Log Format` (short: GELF) can be used to send application logs directly to a TCP socket.

While it has been specified by the Graylog project as their input resource standard, other tools such as Logstash also support `GELF` as input type.

You can enable the feature using

```
# icinga2 feature enable gelf
```

By default the `GelfWriter` object expects the GELF receiver to listen at `127.0.0.1` on TCP port `12201`. The default `source` attribute is set to `icinga2`. You can customize that for your needs if required.

Currently these events are processed: * Check results * State changes * Notifications

14.4.6 OpenTSDB Writer

While there are some OpenTSDB collector scripts and daemons like tcollector available for Icinga 1.x it's more reasonable to directly process the check and

plugin performance in memory in Icinga 2. Once there are new metrics available, Icinga 2 will directly write them to the defined TSDB TCP socket.

You can enable the feature using

```
# icinga2 feature enable opentsdb
```

By default the `OpenTsdbWriter` object expects the TSD to listen at `127.0.0.1` on port `4242`.

The current naming schema is

```
icinga.host.<metricname>
icinga.service.<servicename>.<metricname>
```

for host and service checks. The tag host is always applied.

To make sure Icinga 2 writes a valid metric into OpenTSDB some characters are replaced with `_` in the target name:

```
\  (and space)
```

The resulting name in OpenTSDB might look like:

```
www-01 / http-cert / response time
icinga.http_cert.response_time
```

In addition to the performance data retrieved from the check plugin, Icinga 2 sends internal check statistic data to OpenTSDB:

metric	description
current_attempt	current check attempt
max_check_attempts	maximum check attempts until the hard state is reached
reachable	checked object is reachable
downtime_depth	number of downtimes this object is in
acknowledgement	whether the object is acknowledged or not
execution_time	check execution time
latency	check latency
state	current state of the checked object
state_type	0=SOFT, 1=HARD state

While reachable, state and state_type are metrics for the host or service the other metrics follow the current naming schema

```
icinga.check.<metricname>
```

with the following tags

tag	description
type	the check type, one of [host, service]

tag	description
host	hostname, the check ran on
service	the service name (if type=service)

Note

You might want to set the tsd.core.auto_create_metrics setting to `true` in your opentsdb.conf configuration file.

14.5 Livestatus

The MK Livestatus project implements a query protocol that lets users query their Icinga instance for status information. It can also be used to send commands.

The Livestatus component that is distributed as part of Icinga 2 is a re-implementation of the Livestatus protocol which is compatible with MK Livestatus.

Tip

Only install the Livestatus feature if your web interface or addon requires you to do so. Icinga Web 2 does not need Livestatus.

Details on the available tables and attributes with Icinga 2 can be found in the Livestatus Schema section.

You can enable Livestatus using icinga2 feature enable:

```
# icinga2 feature enable livestatus
```

After that you will have to restart Icinga 2:

```
# systemctl restart icinga2
```

By default the Livestatus socket is available in `/var/run/icinga2/cmd/livestatus`.

In order for queries and commands to work you will need to add your query user (e.g. your web server) to the `icingacmd` group:

```
# usermod -a -G icingacmd www-data
```

The Debian packages use `nagios` as the user and group name. Make sure to change `icingacmd` to `nagios` if you're using Debian.

Change `www-data` to the user you're using to run queries.

In order to use the historical tables provided by the livestatus feature (for example, the `log` table) you need to have the `CompatLogger` feature enabled. By default these logs are expected to be in `/var/log/icinga2/compat`. A different path can be set using the `compat_log_path` configuration attribute.

```
# icinga2 feature enable compatlog
```

14.5.1 Livestatus Sockets

Other to the Icinga 1.x Addon, Icinga 2 supports two socket types

- Unix socket (default)
- TCP socket

Details on the configuration can be found in the LivestatusListener object configuration.

14.5.2 Livestatus GET Queries

> **Note**
>
> All Livestatus queries require an additional empty line as query end identifier. The `nc` tool (`netcat`) provides the `-U` parameter to communicate using a unix socket.

There also is a Perl module available in CPAN for accessing the Livestatus socket programmatically: Monitoring::Livestatus

Example using the unix socket:

```
# echo -e "GET services\n" | /usr/bin/nc -U /var/run/icinga2/cmd/livestatus
```

Example using the tcp socket listening on port `6558`:

```
# echo -e 'GET services\n' | netcat 127.0.0.1 6558

# cat servicegroups <<EOF
GET servicegroups

EOF

(cat servicegroups; sleep 1) | netcat 127.0.0.1 6558
```

14.5.3 Livestatus COMMAND Queries

A list of available external commands and their parameters can be found here

```
$ echo -e 'COMMAND <externalcommandstring>' | netcat 127.0.0.1 6558
```

14.5.4 Livestatus Filters

and, or, negate

Operator	Negate
=	!=
~	!~
=~	!=~
~~	!~~
<	
>	
<=	
>=	

14.5.5 Livestatus Stats

Schema: "Stats: aggregatefunction aggregateattribute"

Aggregate Function	Description
sum	
min	
max	
avg	sum / count
std	standard deviation
suminv	sum (1 / value)
avginv	suminv / count
count	ordinary default for any stats query if not aggregate function defined

Example:

```
GET hosts
Filter: has_been_checked = 1
Filter: check_type = 0
Stats: sum execution_time
Stats: sum latency
Stats: sum percent_state_change
Stats: min execution_time
Stats: min latency
Stats: min percent_state_change
Stats: max execution_time
Stats: max latency
```

```
Stats: max percent_state_change
OutputFormat: json
ResponseHeader: fixed16
```

14.5.6 Livestatus Output

- CSV

CSV output uses two levels of array separators: The members array separator is a comma (1st level) while extra info and host|service relation separator is a pipe (2nd level).

Separators can be set using ASCII codes like:

```
Separators: 10 59 44 124
```

- JSON

Default separators.

14.5.7 Livestatus Error Codes

Code	Description
200	OK
404	Table does not exist
452	Exception on query

14.5.8 Livestatus Tables

Table	Join	Description
hosts		host config and status attributes, services counter
hostgroups		hostgroup config, status attributes and host/service counters
services	hosts	service config and status attributes
servicegroups		servicegroup config, status attributes and service counters
contacts		contact config and status attributes
contactgroups		contact config, members
commands		command name and line

Table	Join	Description
status		programstatus, config and stats
comments	services	status attributes
downtimes	services	status attributes
timeperiods		name and is inside flag
endpoints		config and status attributes
log	services, hosts, contacts, commands	parses compatlog and shows log attributes
statehist	hosts, services	parses compatlog and aggregates state change attributes
hostsbygroup	hostgroups	host attributes grouped by hostgroup and its attributes
servicesbygroup	servicegroups	service attributes grouped by servicegroup and its attributes
servicesbyhostgroup	hostgroups	service attributes grouped by hostgroup and its attributes

The `commands` table is populated with `CheckCommand`, `EventCommand` and `NotificationCommand` objects.

A detailed list on the available table attributes can be found in the Livestatus Schema documentation.

14.6 Status Data Files

Icinga 1.x writes object configuration data and status data in a cyclic interval to its `objects.cache` and `status.dat` files. Icinga 2 provides the `StatusDataWriter` object which dumps all configuration objects and status updates in a regular interval.

```
# icinga2 feature enable statusdata
```

If you are not using any web interface or addon which uses these files, you can safely disable this feature.

14.7 Compat Log Files

The Icinga 1.x log format is considered being the `Compat Log` in Icinga 2 provided with the `CompatLogger` object.

These logs are used for informational representation in external web interfaces parsing the logs, but also to generate SLA reports and trends. The Livestatus feature uses these logs for answering queries to historical tables.

The `CompatLogger` object can be enabled with

```
# icinga2 feature enable compatlog
```

By default, the Icinga 1.x log file called `icinga.log` is located in `/var/log/icinga2/compat`. Rotated log files are moved into `var/log/icinga2/compat/archives`.

14.8 Check Result Files

> **Note** This feature is deprecated and will be removed with Icinga 2.10.0

Icinga 1.x writes its check result files to a temporary spool directory where they are processed in a regular interval. While this is extremely inefficient in performance regards it has been rendered useful for passing passive check results directly into Icinga 1.x skipping the external command pipe.

Several clustered/distributed environments and check-aggregation addons use that method. In order to support step-by-step migration of these environments, Icinga 2 supports the `CheckResultReader` object.

There is no feature configuration available, but it must be defined on-demand in your Icinga 2 objects configuration.

```
object CheckResultReader "reader" {
  spool_dir = "/data/check-results"
}
```

15 Icinga 2 Troubleshooting

15.1 Required Information

Please ensure to provide any detail which may help reproduce and understand your issue. Whether you ask on the community channels or you create an issue at GitHub, make sure that others can follow your explanations. If necessary, draw a picture and attach it for better illustration. This is especially helpful if you are troubleshooting a distributed setup.

We've come around many community questions and compiled this list. Add your own findings and details please.

- Describe the expected behavior in your own words.
- Describe the actual behavior in one or two sentences.

- Ensure to provide general information such as:
  - How was Icinga 2 installed (and which repository in case) and which distribution are you using
  - `icinga2 --version`
  - `icinga2 feature list`
  - `icinga2 daemon -C`
  - Icinga Web 2 version (screenshot from System - About)
  - Icinga Web 2 modules e.g. the Icinga Director (optional)
- Configuration insights:
  - Provide complete configuration snippets explaining your problem in detail
  - Your icinga2.conf file
  - If you run multiple Icinga 2 instances, the zones.conf file (or `icinga2 object list --type Endpoint` and `icinga2 object list --type Zone`) from all affected nodes.
- Logs
  - Relevant output from your main and debug log in `/var/log/icinga2`. Please add step-by-step explanations with timestamps if required.
  - The newest Icinga 2 crash log if relevant, located in `/var/log/icinga2/crash`
- Additional details
  - If the check command failed, what's the output of your manual plugin tests?
  - In case of debugging Icinga 2, the full back traces and outputs

15.2 Analyze your Environment

There are many components involved on a server running Icinga 2. When you analyze a problem, keep in mind that basic system administration knowledge is also key to identify bottlenecks and issues.

Tip

Monitor Icinga 2 and use the hints for further analysis.

- Analyze the system's performance and dentify bottlenecks and issues.
- Collect details about all applications (e.g. Icinga 2, MySQL, Apache, Graphite, Elastic, etc.).
- If data is exchanged via network (e.g. central MySQL cluster) ensure to monitor the bandwidth capabilities too.
- Add graphs and screenshots to your issue description

Install tools which help you to do so. Opinions differ, let us know if you have any additions here!

15.2.1 Analyse your Linux/Unix Environment

htop is a better replacement for `top` and helps to analyze processes interactively.

```
yum install htop
apt-get install htop
```

If you are for example experiencing performance issues, open `htop` and take a screenshot. Add it to your question and/or bug report.

Analyse disk I/O performance in Grafana, take a screenshot and obfuscate any sensitive details. Attach it when posting a question to the community channels.

The sysstat package provides a number of tools to analyze the performance on Linux. On FreeBSD you could use `systat` for example.

```
yum install sysstat
apt-get install sysstat
```

Example for `vmstat` (summary of memory, processes, etc.):

```
// summary
vmstat -s
// print timestamps, format in MB, stats every 1 second, 5 times
vmstat -t -S M 1 5
```

Example for `iostat`:

```
watch -n 1 iostat
```

Example for `sar`:

```
sar //cpu
sar -r //ram
sar -q //load avg
sar -b //I/O
```

`sysstat` also provides the `iostat` binary. On FreeBSD you could use `systat` for example.

If you are missing checks and metrics found in your analysis, add them to your monitoring!

15.2.2 Analyze your Windows Environment

A good tip for Windows are the tools found inside the Sysinternals Suite.

You can also start `perfmon` and analyze specific performance counters. Keep notes which could be important for your monitoring, and add service checks later on.

15.3 Enable Debug Output

15.3.1 Enable Debug Output on Linux/Unix

Enable the `debuglog` feature:

```
# icinga2 feature enable debuglog
# service icinga2 restart
```

The debug log file can be found in `/var/log/icinga2/debug.log`.

Alternatively you may run Icinga 2 in the foreground with debugging enabled. Specify the console log severity as an additional parameter argument to `-x`.

```
# /usr/sbin/icinga2 daemon -x notice
```

The log severity can be one of `critical`, `warning`, `information`, `notice` and `debug`.

15.3.2 Enable Debug Output on Windows

Open a command prompt with administrative privileges and enable the debug log feature.

```
C:> icinga2.exe feature enable debuglog
```

Ensure that the Icinga 2 service already writes the main log into `C:\ProgramData\icinga2\var\log\icinga2`. Restart the Icinga 2 service and open the newly created `debug.log` file.

```
C:> net stop icinga2
C:> net start icinga2
```

15.4 Configuration Troubleshooting

15.4.1 List Configuration Objects

The `icinga2 object list` CLI command can be used to list all configuration objects and their attributes. The tool also shows where each of the attributes was modified.

Tip

Use the Icinga 2 API to access config objects at runtime directly.

That way you can also identify which objects have been created from your apply rules.

```
# icinga2 object list

Object 'localhost!ssh' of type 'Service':
```

```
  * __name = 'localhost!ssh'
  * check_command = 'ssh'
   % = modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 5:3-5:23
  * check_interval = 60
   % = modified in '/etc/icinga2/conf.d/templates.conf', lines 24:3-24:21
  * host_name = 'localhost'
   % = modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 4:3-4:25
  * max_check_attempts = 3
   % = modified in '/etc/icinga2/conf.d/templates.conf', lines 23:3-23:24
  * name = 'ssh'
  * retry_interval = 30
   % = modified in '/etc/icinga2/conf.d/templates.conf', lines 25:3-25:22
  * templates = [ 'ssh', 'generic-service' ]
   % += modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 1:0-7:1
   % += modified in '/etc/icinga2/conf.d/templates.conf', lines 22:1-26:1
  * type = 'Service'
  * vars
   % += modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 6:3-6:19
    * sla = '24x7'
    % = modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 6:3-6:19

[...]
```

You can also filter by name and type:

```
# icinga2 object list --name *ssh* --type Service
Object 'localhost!ssh' of type 'Service':
  * __name = 'localhost!ssh'
  * check_command = 'ssh'
   % = modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 5:3-5:23
  * check_interval = 60
   % = modified in '/etc/icinga2/conf.d/templates.conf', lines 24:3-24:21
  * host_name = 'localhost'
   % = modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 4:3-4:25
  * max_check_attempts = 3
   % = modified in '/etc/icinga2/conf.d/templates.conf', lines 23:3-23:24
  * name = 'ssh'
  * retry_interval = 30
   % = modified in '/etc/icinga2/conf.d/templates.conf', lines 25:3-25:22
  * templates = [ 'ssh', 'generic-service' ]
   % += modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 1:0-7:1
   % += modified in '/etc/icinga2/conf.d/templates.conf', lines 22:1-26:1
  * type = 'Service'
  * vars
   % += modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 6:3-6:19
    * sla = '24x7'
    % = modified in '/etc/icinga2/conf.d/hosts/localhost/ssh.conf', lines 6:3-6:19
```

```
Found 1 Service objects.

[2014-10-15 14:27:19 +0200] information/cli: Parsed 175 objects.
```

Runtime modifications via the REST API are not immediately updated. Furthermore there is a known issue with group assign expressions which are not reflected in the host object output. You need to restart Icinga 2 in order to update the `icinga2.debug` cache file.

15.4.2 Apply rules do not match

You can analyze apply rules and matching objects by using the script debugger.

15.4.3 Where are the check command definitions?

Icinga 2 features a number of built-in check command definitions which are included with

```
include <itl>
include <plugins>
```

in the icinga2.conf configuration file. These files are not considered configuration files and will be overridden on upgrade, so please send modifications as proposed patches upstream. The default include path is set to `LocalStateDir + "/share/icinga2/includes"`.

You should add your own command definitions to a new file in `conf.d/` called `commands.conf` or similar.

15.4.4 Configuration is ignored

- Make sure that the line(s) are not commented out (starting with `//` or `#`, or encapsulated by `/* ... */`).
- Is the configuration file included in icinga2.conf?

Run the configuration validation and add `notice` as log severity. Search for the file which should be included i.e. using the `grep` CLI command.

```
# icinga2 daemon -C -x notice | grep command
```

15.4.5 Configuration attributes are inherited from

Icinga 2 allows you to import templates using the import keyword. If these templates contain additional attributes, your objects will automatically inherit them. You can override or modify these attributes in the current object.

The object list CLI command allows you to verify the attribute origin.

15.4.6 Configuration Value with Single Dollar Sign

In case your configuration validation fails with a missing closing dollar sign error message, you did not properly escape the single dollar sign preventing its usage as runtime macro.

```
critical/config: Error: Validation failed for Object 'ping4' (Type: 'Service') at /etc/icinga2/
```

Correct the custom attribute value to

```
"top-syntax=$${list}"
```

15.5 Checks Troubleshooting

15.5.1 Executed Command for Checks

- Use the Icinga 2 API to query host/service objects for their check result containing the executed shell command.
- Use the Icinga 2 console cli command to fetch the checkable object, its check result and the executed shell command.
- Alternatively enable the debug log and look for the executed command.

Example for a service object query using a regex match on the name:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X PO
-d '{ "filter": "regex(pattern, service.name)", "filter_vars": { "pattern": "^http" }, "attrs":
{
    "results": [
        {
            "attrs": {
                "__name": "example.localdomain!http",
                "last_check_result": {
                    "active": true,
                    "check_source": "example.localdomain",
                    "command": [
                        "/usr/local/sbin/check_http",
                        "-I",
                        "127.0.0.1",
                        "-u",
                        "/"
                    ],

  ...
```

```
                }
            },
            "joins": {},
            "meta": {},
            "name": "example.localdomain!http",
            "type": "Service"
        }
    ]
}
```

Example for using the `icinga2 console` CLI command evaluation functionality:

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/' \
--eval 'get_service("example.localdomain", "http").last_check_result.command' | python -m jso
[
    "/usr/local/sbin/check_http",
    "-I",
    "127.0.0.1",
    "-u",
    "/"
]
```

Example for searching the debug log:

```
# icinga2 feature enable debuglog
# systemctl restart icinga2
# tail -f /var/log/icinga2/debug.log | grep "notice/Process"
```

15.5.2 Checks are not executed

- Check the debug log to see if the check command gets executed.
- Verify that failed depedencies do not prevent command execution.
- Make sure that the plugin is executable by the Icinga 2 user (run a manual test).
- Make sure the checker feature is enabled.
- Use the Icinga 2 API event streams to receive live check result streams.

Examples:

```
# sudo -u icinga /usr/lib/nagios/plugins/check_ping -4 -H 127.0.0.1 -c 5000,100% -w 3000,80%

# icinga2 feature enable checker
The feature 'checker' is already enabled.
```

Fetch all check result events matching the `event.service` name `random`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ev
```

15.5.3 Analyze Check Source

Sometimes checks are not executed on the remote host, but on the master and so on. This could lead into unwanted results or NOT-OK states.

The `check_source` attribute is the best indication where a check command was actually executed. This could be a satellite with synced configuration or a client as remote command bridge – both will return the check source as where the plugin is called.

Example for retrieving the check source from all `disk` services using a regex match on the name:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -H 'X-HTTP-Method-Override: GET' -X PO
-d '{ "filter": "regex(pattern, service.name)", "filter_vars": { "pattern": "^disk" }, "attrs":
{
    "results": [
        {
            "attrs": {
                "__name": "icinga2-client1.localdomain!disk",
                "last_check_result": {
                    "active": true,
                    "check_source": "icinga2-client1.localdomain",

  ...

                }
            },
            "joins": {},
            "meta": {},
            "name": "icinga2-client1.localdomain!disk",
            "type": "Service"
        }
    ]
}
```

Example for using the `icinga2 console` CLI command evaluation functionality:

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/' \
--eval 'get_service("icinga2-client1.localdomain", "disk").last_check_result.check_source' |

"icinga2-client1.localdomain"
```

15.5.4 NSClient++ Check Errors with nscp-local

The nscp-local CheckCommand object definitions call the local `nscp.exe` command. If a Windows client service check fails to find the `nscp.exe` command,

the log output would look like this:

```
Command ".\nscp.exe" "client" "-a" "drive=d" "-a" "show-all" "-b" "-q" "check_drivesize" failed
```

or

```
Command ".
scp.exe" "client" "-a" "drive=d" "-a" "show-all" "-b" "-q" "check_drivesize" failed to execute:
```

The above actually prints `.\\nscp.exe` where the escaped `\n` character gets interpreted as new line.

Both errors lead to the assumption that the `NscpPath` constant is empty or set to a `.` character. This could mean the following:

- The command is **not executed on the Windows client**. Check the check_source attribute from the check result.
- You are using an outdated NSClient++ version (0.3.x or 0.4.x) which is not compatible with Icinga 2.
- You are using a custom NSClient++ installer which does not register the correct GUID for NSClient++

More troubleshooting:

Retrieve the `NscpPath` constant on your Windows client:

```
C:\Program Files\ICINGA2\sbin\icinga2.exe variable get NscpPath
```

If the variable is returned empty, manually test how Icinga 2 would resolve its path (this can be found inside the ITL):

```
C:\Program Files\ICINGA2\sbin\icinga2.exe console --eval "dirname(msi_get_component_path(\"{5
```

If this command does not return anything, NSClient++ is not properly installed. Verify that inside the `Programs and Features` (`appwiz.cpl`) control panel.

You can run the bundled NSClient++ installer from the Icinga 2 Windows package. The msi package is located in `C:\Program Files\ICINGA2\sbin`.

The bundled NSClient++ version has properly been tested with Icinga 2. Keep that in mind when using a different package.

15.5.5 Check Thresholds Not Applied

This could happen with clients as command endpoint execution.

If you have for example a client host `icinga2-client1.localdomain` and a service `disk` check defined on the master, the warning and critical thresholds are sometimes to applied and unwanted notification alerts are raised.

This happens because the client itself includes a host object with its `NodeName` and a basic set of checks in the conf.d directory, i.e. `disk` with the default thresholds.

Clients which have the `checker` feature enabled will attempt to execute checks for local services and send their results back to the master.

If you now have the same host and service objects on the master you will receive wrong check results from the client.

Solution:

- Disable the `checker` feature on clients: `icinga2 feature disable checker`.
- Remove the inclusion of conf.d as suggested in the client setup docs.

15.5.6 Check Fork Errors

Newer versions of Systemd on Linux limit spawned processes for services.

- v227 introduces the `TasksMax` setting to units which allows to specify the spawned process limit.
- v228 adds `DefaultTasksMax` in the global `systemd-system.conf` with a default setting of 512 processes.
- v231 changes the default value to 15%

This can cause problems with Icinga 2 in large environments with many commands executed in parallel starting with Systemd v228. Some distributions also may have changed the defaults.

The error message could look like this:

```
2017-01-12T11:55:40.742685+01:00 icinga2-master1 kernel: [65567.582895] cgroup: fork rejected
```

In order to solve the problem, increase the value for `DefaultTasksMax` or set it to `infinity`.

```
mkdir /etc/systemd/system/icinga2.service.d
cat >/etc/systemd/system/icinga2.service.d/limits.conf <<EOF
[Service]
DefaultTasksMax=infinity
EOF

systemctl daemon-reload
systemctl restart icinga2
```

An example is available inside the GitHub repository in etc/initsystem.

External Resources:

- Fork limit for cgroups
- Systemd changelog
- Icinga 2 upstream issue
- Systemd upstream discussion

15.5.7 Late Check Results

Icinga Web 2 provides a dashboard overview for `overdue checks`.

The REST API provides the status URL endpoint with some generic metrics on Icinga and its features.

```
# curl -k -s -u root:icinga 'https://localhost:5665/v1/status?pretty=1' | less
```

You can also calculate late check results via the REST API:

- Fetch the `last_check` timestamp from each object
- Compare the timestamp with the current time and add `check_interval` multiple times (change it to see which results are really late, like five times check_interval)

You can use the icinga2 console to connect to the instance, fetch all data and calculate the differences. More infos can be found in this blogpost.

```
# ICINGA2_API_USERNAME=root ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://loc

<1> => var res = []; for (s in get_objects(Service).filter(s => s.last_check < get_time() - 2 * s

[ [ "10807-host!10807-service", "2016-06-10 15:54:55 +0200" ], [ "mbmif.int.netways.de!disk /",
```

Or if you are just interested in numbers, call len on the result array `res`:

```
<2> => var res = []; for (s in get_objects(Service).filter(s => s.last_check < get_time() - 2 * s

2.000000
```

If you need to analyze that problem multiple times, just add the current formatted timestamp and repeat the commands.

```
<23> => DateTime(get_time()).to_string()

"2017-04-04 16:09:39 +0200"

<24> => var res = []; for (s in get_objects(Service).filter(s => s.last_check < get_time() - 2 *

8287.000000
```

More details about the Icinga 2 DSL and its possibilities can be found in the language and library reference chapters.

15.5.8 Late Check Results in Distributed Environments

When it comes to a distributed HA setup, each node is responsible for a load-balanced amount of checks. Host and Service objects provide the attribute

`paused`. If this is set to `false`, the current node actively attempts to schedule and execute checks. Otherwise the node does not feel responsible.

```
<3> => var res = {}; for (s in get_objects(Service).filter(s => s.last_check < get_time() - 2 * s
{
  @false = 2.000000
  @true = 1.000000
}
```

You may ask why this analysis is important? Fair enough - if the numbers are not inverted in a HA zone with two members, this may give a hint that the cluster nodes are in a split-brain scenario, or you've found a bug in the cluster.

If you are running a cluster setup where the master/satellite executes checks on the client via top down command endpoint mode, you might want to know which zones are affected.

This analysis assumes that clients which are not connected, have the string `connected` in their service check result output and their state is `UNKNOWN`.

```
<4> => var res = {}; for (s in get_objects(Service)) { if (s.state==3) { if (match("*connected*",

{
  Asia = 31.000000
  Europe = 214.000000
  USA = 207.000000
}
```

The result set shows the configured zones and their affected hosts in a unique list. The output also just prints the numbers but you can adjust this by omitting the `len()` call inside the for loop.

15.6 Notifications Troubleshooting

15.6.1 Notifications are not sent

- Check the debug log to see if a notification is triggered.
- If yes, verify that all conditions are satisfied.
- Are any errors on the notification command execution logged?

Please ensure to add these details with your own description to any question or issue posted to the community channels.

Verify the following configuration:

- Is the host/service `enable_notifications` attribute set, and if so, to which value?
- Do the notification attributes `states`, `types`, `period` match the notification conditions?

- Do the user attributes `states`, `types`, `period` match the notification conditions?
- Are there any notification `begin` and `end` times configured?
- Make sure the notification feature is enabled.
- Does the referenced NotificationCommand work when executed as Icinga user on the shell?

If notifications are to be sent via mail, make sure that the mail program specified inside the NotificationCommand object exists. The name and location depends on the distribution so the preconfigured setting might have to be changed on your system.

Examples:

```
# icinga2 feature enable notification
The feature 'notification' is already enabled.
```

```
# icinga2 feature enable debuglog
# systemctl restart icinga2
```

```
# grep Notification /var/log/icinga2/debug.log > /root/analyze_notification_problem.log
```

You can use the Icinga 2 API event streams to receive live notification streams:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ev
```

15.7 Feature Troubleshooting

15.7.1 Feature is not working

- Make sure that the feature configuration is enabled by symlinking from `features-available/` to `features-enabled` and that the latter is included in icinga2.conf.
- Are the feature attributes set correctly according to the documentation?
- Any errors on the logs?

Look up the object type for the required feature and verify it is enabled:

```
# icinga2 object list --type <feature object type>
```

Example for the `graphite` feature:

```
# icinga2 object list --type GraphiteWriter
```

Look into the log and check whether the feature logs anything specific for this matter.

```
grep GraphiteWriter /var/log/icinga2/icinga2.log
```

15.8 Certificate Troubleshooting

15.8.1 Certificate Verification

If the TLS handshake fails when a client connects to the cluster or the REST API, ensure to verify the used certificates.

Print the CA and client certificate and ensure that the following attributes are set:

- Version must be 3.
- Serial number is a hex-encoded string.
- Issuer should be your certificate authority (defaults to `Icinga CA` for all CLI commands).
- Validity, meaning to say the certificate is not expired.
- Subject with the common name (CN) matches the client endpoint name and its FQDN.
- v3 extensions must set the basic constraint for `CA:TRUE` (ca.crt) or `CA:FALSE` (client certificate).
- Subject Alternative Name is set to a proper DNS name (required for REST API and browsers).

```
# cd /var/lib/icinga2/certs/
```

CA certificate:

```
# openssl x509 -in ca.crt -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=Icinga CA
        Validity
            Not Before: Feb 23 14:45:32 2016 GMT
            Not After : Feb 19 14:45:32 2031 GMT
        Subject: CN=Icinga CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
```

```
...
```

Client public certificate:

```
# openssl x509 -in icinga2-client1.localdomain.crt -text

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
        86:47:44:65:49:c6:65:6b:5e:6d:4f:a5:fe:6c:76:05:0b:1a:cf:34
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=Icinga CA
        Validity
            Not Before: Aug 20 16:20:05 2016 GMT
            Not After : Aug 17 16:20:05 2031 GMT
        Subject: CN=icinga2-client1.localdomain
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Alternative Name:
                DNS:icinga2-client1.localdomain
    Signature Algorithm: sha256WithRSAEncryption
...
```

Make sure to verify the client's certificate and its received `ca.crt` in `/var/lib/icinga2/certs` and ensure that both instances are signed by the **same CA**.

```
# openssl verify -verbose -CAfile /var/lib/icinga2/certs/ca.crt /var/lib/icinga2/certs/icinga2
icinga2-master1.localdomain.crt: OK

# openssl verify -verbose -CAfile /var/lib/icinga2/certs/ca.crt /var/lib/icinga2/certs/icinga2
icinga2-client1.localdomain.crt: OK
```

Fetch the `ca.crt` file from the client node and compare it to your master's `ca.crt` file:

```
# scp icinga2-client1:/var/lib/icinga2/certs/ca.crt test-client-ca.crt
# diff -ur /var/lib/icinga2/certs/ca.crt test-client-ca.crt
```

On SLES11 you'll need to use the `openssl1` command instead of `openssl`.

15.8.2 Certificate Problems with OpenSSL 1.1.0

Users have reported problems with SSL certificates inside a distributed monitoring setup when they

- updated their Icinga 2 package to 2.7.0 on Windows or
- upgraded their distribution which included an update to OpenSSL 1.1.0.

Example during startup on a Windows client:

```
critical/SSL: Error loading and verifying locations in ca key file 'C:\ProgramData\icinga2\etc/
critical/config: Error: Cannot make SSL context for cert path: 'C:\ProgramData\icinga2\etc/icin
```

A technical analysis and solution for re-creating the public CA certificate is available in this advisory.

15.9 Cluster and Clients Troubleshooting

This applies to any Icinga 2 node in a distributed monitoring setup.

You should configure the cluster health checks if you haven't done so already.

> **Note**
>
> Some problems just exist due to wrong file permissions or applied packet filters. Make sure to check these in the first place.

15.9.1 Cluster Troubleshooting Connection Errors

General connection errors could be one of the following problems:

- Incorrect network configuration
- Packet loss
- Firewall rules preventing traffic

Use tools like `netstat`, `tcpdump`, `nmap`, etc. to make sure that the cluster communication works (default port is `5665`).

```
# tcpdump -n port 5665 -i any

# netstat -tulpen | grep icinga

# nmap icinga2-client1.localdomain
```

15.9.2 Cluster Troubleshooting SSL Errors

If the cluster communication fails with SSL error messages, make sure to check the following

- File permissions on the SSL certificate files
- Does the used CA match for all cluster endpoints?
- Verify the `Issuer` being your trusted CA
- Verify the `Subject` containing your endpoint's common name (CN)
- Check the validity of the certificate itself

Try to manually connect from `icinga2-client1.localdomain` to the master node `icinga2-master1.localdomain`:

```
# openssl s_client -CAfile /var/lib/icinga2/certs/ca.crt -cert /var/lib/icinga2/certs/icinga2-

CONNECTED(00000003)
---
...
```

If the connection attempt fails or your CA does not match, verify the certificates.

15.9.2.1 Cluster Troubleshooting Unauthenticated Clients

Unauthenticated nodes are able to connect. This is required for client setups.

Master:

```
[2015-07-13 18:29:25 +0200] information/ApiListener: New client connection for identity 'icinga
```

Client as command execution bridge:

```
[2015-07-13 18:29:26 +1000] notice/ClusterEvents: Discarding 'execute command' message from 'ic
```

If these messages do not go away, make sure to verify the master and client certificates.

15.9.3 Cluster Troubleshooting Message Errors

When the network connection is broken or gone, the Icinga 2 instances will be disconnected. If the connection can't be re-established between endpoints in the same HA zone, they remain in a Split-Brain-mode and history may differ.

Although the Icinga 2 cluster protocol stores historical events in a replay log for later synchronisation, you should make sure to check why the network connection failed.

Ensure to setup cluster health checks to monitor all endpoints and zones connectivity.

15.9.4 Cluster Troubleshooting Command Endpoint Errors

Command endpoints can be used for clients as well as inside an High-Availability cluster.

There is no cli command for manually executing the check, but you can verify the following (e.g. by invoking a forced check from the web interface):

- `/var/log/icinga2/icinga2.log` contains connection and execution errors.
- The ApiListener is not enabled to accept commands.
- `CheckCommand` definition not found on the remote client.
- Referenced check plugin not found on the remote client.
- Runtime warnings and errors, e.g. unresolved runtime macros or configuration problems.
- Specific error messages are also populated into `UNKNOWN` check results including a detailed error message in their output.
- Verify the `check_source` object attribute. This is populated by the node executing the check.
- More verbose logs are found inside the debug log.
- Use the Icinga 2 API event streams to receive live check result streams.

Fetch all check result events matching the `event.service` name `remote-client`:

```
$ curl -k -s -u root:icinga -H 'Accept: application/json' -X POST 'https://localhost:5665/v1/ev
```

15.9.5 Cluster Troubleshooting Config Sync

If the cluster zones do not sync their configuration, make sure to check the following:

- Within a config master zone, only one configuration master is allowed to have its config in `/etc/icinga2/zones.d`. ** The master syncs the configuration to `/var/lib/icinga2/api/zones/` during startup and only syncs valid configuration to the other nodes. ** The other nodes receive the configuration into `/var/lib/icinga2/api/zones/`.
- The `icinga2.log` log file in `/var/log/icinga2` will indicate whether this ApiListener accepts config, or not.

Verify the object's version attribute on all nodes to check whether the config update and reload was successful or not.

15.9.6 Cluster Troubleshooting Overdue Check Results

If your master does not receive check results (or any other events) from the child zones (satellite, clients, etc.), make sure to check whether the client sending in events is allowed to do so.

Tip

General troubleshooting hints on late check results are documented here.

The distributed monitoring conventions apply. So, if there's a mismatch between your client node's endpoint name and its provided certificate's CN, the master will deny all events.

Tip

Icinga Web 2 provides a dashboard view for overdue check results.

Enable the debug log on the master for more verbose insights.

If the client cannot authenticate, it's a more general problem.

The client's endpoint is not configured on nor trusted by the master node:

```
Discarding 'check result' message from 'icinga2-client1.localdomain': Invalid endpoint origin (
```

The check result message sent by the client does not belong to the zone the checkable object is in on the master:

```
Discarding 'check result' message from 'icinga2-client1.localdomain': Unauthorized access.
```

15.9.7 Cluster Troubleshooting Replay Log

If your `/var/lib/icinga2/api/log` directory grows, it generally means that your cluster cannot replay the log on connection loss and re-establishment. A master node for example will store all events for not connected endpoints in the same and child zones.

Check the following:

- All clients are connected? (e.g. cluster health check).
- Check your connection in general.
- Does the log replay work, e.g. are all events processed and the directory gets cleared up over time?
- Decrease the `log_duration` attribute value for that specific endpoint.

16 Upgrading Icinga 2

Upgrading Icinga 2 is usually quite straightforward. Ordinarily the only manual steps involved are scheme updates for the IDO database.

Specific version upgrades are described below. Please note that version updates are incremental. An upgrade from v2.6 to v2.8 requires to follow the instructions for v2.7 too.

16.1 Upgrading to v2.8.2

With version 2.8.2 the location of settings formerly found in `/etc/icinga2/init.conf` has changed. They are now located in the sysconfig, `/etc/sysconfig/icinga2` (RPM) or `/etc/default/icinga2` (DPKG) on most systems. The `init.conf` file has been removed and its settings will be ignored. These changes are only relevant if you edited the `init.conf`. Below is a table displaying the new names for the affected settings.

Old `init.conf`	New `sysconfig/icinga2`
RunAsUser	ICINGA2_USER
RunAsGroup	ICINGA2_GROUP
RLimitFiles	ICINGA2_RLIMIT_FILES
RLimitProcesses	ICINGA2_RLIMIT_PROCESSES
RLimitStack	ICINGA2_RLIMIT_STACK

16.2 Upgrading to v2.8

16.2.1 DB IDO Schema Update to 2.8.0

There are additional indexes and schema fixes which require an update.

Please proceed here for MySQL or PostgreSQL.

Note

`2.8.1.sql` fixes a unique constraint problem with fresh 2.8.0 installations. You don't need this update if you are upgrading from an older version.

16.2.2 Changed Certificate Paths

The default certificate path was changed from `/etc/icinga2/pki` to `/var/lib/icinga2/certs`.

Old Path	New Path
`/etc/icinga2/pki/icinga2-client1.localdomain.crt`	`/var/lib/icinga2/certs/icinga2-client1.localdomain.crt`
`/etc/icinga2/pki/icinga2-client1.localdomain.key`	`/var/lib/icinga2/certs/icinga2-client1.localdomain.key`
`/etc/icinga2/pki/ca.crt`	`/var/lib/icinga2/certs/ca.crt`

This applies to Windows clients in the same way: `%ProgramData%\etc\icinga2\pki` was moved to `%ProgramData%\var\lib\icinga2\certs`.

Old Path	New Path
%ProgramData%\etc\icinga2\pki\icinga2-client1.localdomain.crt	%ProgramData%\var\lib\icinga2\certs\icinga2-client1.localdomain.crt
%ProgramData%\etc\icinga2\pki\icinga2-client1.localdomain.key	%ProgramData%\var\lib\icinga2\certs\icinga2-client1.localdomain.key
%ProgramData%\etc\icinga2\pki\ca.crt	%ProgramData%\var\lib\icinga2\certs\ca.crt

Note

The default expected path for client certificates is `/var/lib/icinga2/certs/ + NodeName + {.crt,.key}`. The `NodeName` constant is usually the FQDN and certificate common name (CN). Check the conventions section inside the Distributed Monitoring chapter.

The setup CLI commands and the default ApiListener configuration have been adjusted to these paths too.

The ApiListener object attributes `cert_path`, `key_path` and `ca_path` have been deprecated and removed from the example configuration.

16.2.2.1 Migration Path

Note

Icinga 2 automatically migrates the certificates to the new default location if they are configured and detected in `/etc/icinga2/pki`.

During startup, the migration kicks in and ensures to copy the certificates to the new location. This will also happen if someone updates the certificate files in `/etc/icinga2/pki` to ensure that the new certificate location always has the latest files.

This has been implemented in the Icinga 2 binary to ensure it works on both Linux/Unix and the Windows platform.

If you are not using the built-in CLI commands and setup wizards to deploy the client certificates, please ensure to update your deployment tools/scripts. This mainly affects

- Puppet modules
- Ansible playbooks
- Chef cookbooks
- Salt recipes
- Custom scripts, e.g. Windows Powershell or self-made implementations

In order to support a smooth migration between versions older than 2.8 and future releases, the built-in certificate migration path is planned to exist as long as the deprecated `ApiListener` object attributes exist.

You are safe to use the existing configuration paths inside the `api` feature.

Example

Look at the following example taken from the Director Linux deployment script for clients.

- Ensure that the default certificate path is changed from `/etc/icinga2/pki` to `/var/lib/icinga2/certs`.

```
-ICINGA2_SSL_DIR="${ICINGA2_CONF_DIR}/pki"
+ICINGA2_SSL_DIR="${ICINGA2_STATE_DIR}/lib/icinga2/certs"
```

- Remove the ApiListener configuration attributes.

```
object ApiListener "api" {
-  cert_path = SysconfDir + "/icinga2/pki/${ICINGA2_NODENAME}.crt"
-  key_path = SysconfDir + "/icinga2/pki/${ICINGA2_NODENAME}.key"
-  ca_path = SysconfDir + "/icinga2/pki/ca.crt"
  accept_commands = true
  accept_config = true
}
```

Test the script with a fresh client installation before putting it into production.

Tip

Please support module and script developers in their migration. If you find any project which would require these changes, create an issue or a patchset in a PR and help them out. Thanks in advance!

16.2.3 On-Demand Signing and CA Proxy

Icinga 2 v2.8 supports the following features inside the cluster:

- Forward signing requests from clients through a satellite instance to a signing master ("CA Proxy").
- Signing requests without a ticket. The master instance allows to list and sign CSRs ("On-Demand Signing").

In order to use these features, **all instances must be upgraded to v2.8**.

More details in this chapter.

16.2.4 Windows Client

Windows versions older than Windows 10/Server 2016 require the Universal C Runtime for Windows.

16.2.5 Removed Bottom Up Client Mode

This client mode was deprecated in 2.6 and was removed in 2.8.

The node CLI command does not provide `list` or `update-config` anymore.

Note

The old migration guide can be found on GitHub.

The clients don't need to have a local `conf.d` directory included.

Icinga 2 continues to run with the generated and imported configuration. You are advised to migrate any existing configuration to the "top down" mode with the help of the Icinga Director or config management tools such as Puppet, Ansible, etc.

16.2.6 Removed Classic UI Config Package

The config meta package `classicui-config` and the configuration files have been removed. You need to manually configure this legacy interface. Create a backup of the configuration before upgrading and re-configure it afterwards.

16.2.7 Flapping Configuration

Icinga 2 v2.8 implements a new flapping detection algorithm which splits the threshold configuration into low and high settings.

`flapping_threshold` is deprecated and does not have any effect when flapping is enabled. Please remove `flapping_threshold` from your configuration. This attribute will be removed in v2.9.

Instead you need to use the `flapping_threshold_low` and `flapping_threshold_high` attributes. More details can be found here.

16.2.8 Deprecated Configuration Attributes

Object	Attribute
ApiListener	cert_path (migration happens)
ApiListener	key_path (migration happens)
ApiListener	ca_path (migration happens)
Host, Service	flapping_threshold (has no effect)

16.3 Upgrading to v2.7

v2.7.0 provided new notification scripts and commands. Please ensure to update your configuration accordingly. An advisory has been published here.

In case are having troubles with OpenSSL 1.1.0 and the public CA certificates,

please read this advisory and check the troubleshooting chapter.

If Icinga 2 fails to start with an empty reference to `$ICINGA2_CACHE_DIR` ensure to set it inside `/etc/sysconfig/icinga2` (RHEL) or `/etc/default/icinga2` (Debian).

RPM packages will put a file called `/etc/sysconfig/icinga2.rpmnew` if you have modified the original file.

Example on CentOS 7:

```
vim /etc/sysconfig/icinga2

ICINGA2_CACHE_DIR=/var/cache/icinga2

systemctl restart icinga2
```

16.4 Upgrading the MySQL database

If you want to upgrade an existing Icinga 2 instance, check the `/usr/share/icinga2-ido-mysql/schema/upgra` directory for incremental schema upgrade file(s).

> **Note**
>
> If there isn't an upgrade file for your current version available, there's nothing to do.

Apply all database schema upgrade files incrementally.

```
# mysql -u root -p icinga < /usr/share/icinga2-ido-mysql/schema/upgrade/<version>.sql
```

The Icinga 2 DB IDO feature checks the required database schema version on startup and generates an log message if not satisfied.

Example: You are upgrading Icinga 2 from version `2.4.0` to `2.8.0`. Look into the `upgrade` directory:

```
$ ls /usr/share/icinga2-ido-mysql/schema/upgrade/
2.0.2.sql 2.1.0.sql 2.2.0.sql 2.3.0.sql 2.4.0.sql 2.5.0.sql 2.6.0.sql 2.8.0.sql
```

There are two new upgrade files called `2.5.0.sql`, `2.6.0.sql` and `2.8.0.sql` which must be applied incrementally to your IDO database.

```
# mysql -u root -p icinga < /usr/share/icinga2-ido-mysql/schema/upgrade/2.5.0.sql
# mysql -u root -p icinga < /usr/share/icinga2-ido-mysql/schema/upgrade/2.6.0.sql
# mysql -u root -p icinga < /usr/share/icinga2-ido-mysql/schema/upgrade/2.8.0.sql
```

16.5 Upgrading the PostgreSQL database

If you want to upgrade an existing Icinga 2 instance, check the `/usr/share/icinga2-ido-pgsql/schema/upgra` directory for incremental schema upgrade file(s).

> **Note**
>
> If there isn't an upgrade file for your current version available, there's nothing to do.

Apply all database schema upgrade files incrementally.

```
# export PGPASSWORD=icinga
# psql -U icinga -d icinga < /usr/share/icinga2-ido-pgsql/schema/upgrade/<version>.sql
```

The Icinga 2 DB IDO feature checks the required database schema version on startup and generates an log message if not satisfied.

Example: You are upgrading Icinga 2 from version `2.4.0` to `2.8.0`. Look into the `upgrade` directory:

```
$ ls /usr/share/icinga2-ido-pgsql/schema/upgrade/
2.0.2.sql 2.1.0.sql 2.2.0.sql 2.3.0.sql 2.4.0.sql 2.5.0.sql 2.6.0.sql 2.8.0.sql
```

There are two new upgrade files called `2.5.0.sql`, `2.6.0.sql` and `2.8.0.sql` which must be applied incrementally to your IDO database.

```
# export PGPASSWORD=icinga
# psql -U icinga -d icinga < /usr/share/icinga2-ido-pgsql/schema/upgrade/2.5.0.sql
# psql -U icinga -d icinga < /usr/share/icinga2-ido-pgsql/schema/upgrade/2.6.0.sql
# psql -U icinga -d icinga < /usr/share/icinga2-ido-pgsql/schema/upgrade/2.8.0.sql
```

17 Language Reference

17.1 Object Definition

Icinga 2 features an object-based configuration format. You can define new objects using the `object` keyword:

```
object Host "host1.example.org" {
  display_name = "host1"

  address = "192.168.0.1"
  address6 = "::1"
}
```

In general you need to write each statement on a new line. Expressions started with `{`, `(` and `[` extend until the matching closing character and can be broken up into multiple lines.

Alternatively you can write multiple statements on a single line by separating them with a semicolon:

```
object Host "host1.example.org" {
  display_name = "host1"

  address = "192.168.0.1"; address6 = "::1"
}
```

Each object is uniquely identified by its type (`Host`) and name (`host1.example.org`). Some types have composite names, e.g. the `Service` type which uses the `host_name` attribute and the name you specified to generate its object name.

Exclamation marks (!) are not permitted in object names.

Objects can contain a comma-separated list of property declarations. Instead of commas semicolons may also be used. The following data types are available for property values:

All objects have at least the following attributes:

Attribute	Description
name	The name of the object. This attribute can be modified in the object definition to override the name specified with the `object` directive.
type	The type of the object.

17.2 Expressions

The following expressions can be used on the right-hand side of assignments.

17.2.1 Numeric Literals

A floating-point number.

Example:

```
27.3
```

17.2.2 Duration Literals

Similar to floating-point numbers except for the fact that they support suffixes to help with specifying time durations.

Example:

```
2.5m
```

Supported suffixes include ms (milliseconds), s (seconds), m (minutes), h (hours) and d (days).

Duration literals are converted to seconds by the config parser and are treated like numeric literals.

17.2.3 String Literals

A string.

Example:

```
"Hello World!"
```

17.2.3.1 String Literals Escape Sequences

Certain characters need to be escaped. The following escape sequences are supported:

Character	Escape sequence
“	\“
\	\\
<TAB>	\t
<CARRIAGE-RETURN>	\r
<LINE-FEED>	\n
<BEL>	\b
<FORM-FEED>	\f

In addition to these pre-defined escape sequences you can specify arbitrary ASCII characters using the backslash character (\) followed by an ASCII character in octal encoding.

17.2.4 Multi-line String Literals

Strings spanning multiple lines can be specified by enclosing them in {{{ and }}}.

Example:

```
{{{This
is
a multi-line
```

```
string.}}}
```

Unlike in ordinary strings special characters do not have to be escaped in multi-line string literals.

17.2.5 Boolean Literals

The keywords `true` and `false` are used to denote truth values.

17.2.6 Null Value

The `null` keyword can be used to specify an empty value.

17.2.7 Dictionary

An unordered list of key-value pairs. Keys must be unique and are compared in a case-sensitive manner.

Individual key-value pairs must either be comma-separated or on separate lines. The comma after the last key-value pair is optional.

Example:

```
{
  address = "192.168.0.1"
  port = 443
}
```

Identifiers may not contain certain characters (e.g. space) or start with certain characters (e.g. digits). If you want to use a dictionary key that is not a valid identifier, you can enclose the key in double quotes.

17.2.8 Array

An ordered list of values.

Individual array elements must be comma-separated. The comma after the last element is optional.

Example:

```
[ "hello", 42 ]
```

An array may simultaneously contain values of different types, such as strings and numbers.

17.2.9 Operators

The following operators are supported in expressions. The operators are sorted by descending precedence.

Operator	Precedence	Examples (Result)	Description
()	1	(3 + 3) * 5	Groups sub-expressions
()	1	Math.random()	Calls a function
[]	1	a[3]	Array subscript
.	1	a.b	Element access
!	2	¡‘Hello” (false), !false (true)	Logical negation of the operand
~	2	~true (false)	Bitwise negation of the operand
+	2	+3	Unary plus
-	2	-3	Unary minus
*	3	5m * 10 (3000)	Multiplies two numbers
/	3	5m / 5 (60)	Divides two numbers
%	3	17 % 12 (5)	Remainder after division
+	4	1 + 3 (4), “hello” + “world” (“hello world”)	Adds two numbers; concatenates strings
-	4	3 - 1 (2)	Subtracts two numbers
<<	5	4 << 8 (1024)	Left shift
>>	5	1024 >> 4 (64)	Right shift
<	6	3 < 5 (true)	Less than
>	6	3 > 5 (false)	Greater than
<=	6	3 <= 3 (true)	Less than or equal
>=	6	3 >= 3 (true)	Greater than or equal
in	7	“foo” in [“foo”, “bar”] (true)	Element contained in array
!in	7	“foo” !in [“bar”, “baz”] (true)	Element not contained in array
==	8	“hello” == “hello” (true), 3 == 5 (false)	Equal to
!=	8	“hello” != “world” (true), 3 != 3 (false)	Not equal to
&	9	7 & 3 (3)	Binary AND
^	10	17 ^ 12 (29)	Bitwise XOR
|	11	2 | 3 (3)	Binary OR
&&	13	true && false (false), 3 && 7 (7), 0 && 7 (0)	Logical AND

Operator	Precedence	Examples (Result)	Description
||	14	true || false (true), 0 || 7 (7)	Logical OR
=	12	a = 3	Assignment
=>	15	x => x * x (function with arg x)	Lambda, for loop

17.2.10 Function Calls

Functions can be called using the `()` operator:

```
const MyGroups = [ "test1", "test" ]

{
  check_interval = len(MyGroups) * 1m
}
```

A list of available functions is available in the Library Reference chapter.

17.3 Assignments

In addition to the `=` operator shown above a number of other operators to manipulate attributes are supported. Here's a list of all available operators:

17.3.1 Operator =

Sets an attribute to the specified value.

Example:

```
{
  a = 5
  a = 7
}
```

In this example `a` has the value `7` after both instructions are executed.

17.3.2 Operator +=

The += operator is a shortcut. The following expression:

```
{
  a = [ "hello" ]
  a += [ "world" ]
}
```

is equivalent to:

```
{
  a = [ "hello" ]
  a = a + [ "world" ]
}
```

17.3.3 Operator -=

The -= operator is a shortcut. The following expression:

```
{
  a = 10
  a -= 5
}
```

is equivalent to:

```
{
  a = 10
  a = a - 5
}
```

17.3.4 Operator *=

The *= operator is a shortcut. The following expression:

```
{
  a = 60
  a *= 5
}
```

is equivalent to:

```
{
  a = 60
  a = a * 5
}
```

17.3.5 Operator /=

The /= operator is a shortcut. The following expression:

```
{
  a = 300
  a /= 5
}
```

is equivalent to:

```
{
  a = 300
  a = a / 5
}
```

17.4 Indexer

The indexer syntax provides a convenient way to set dictionary elements.

Example:

```
{
  hello.key = "world"
}
```

Example (alternative syntax):

```
{
  hello["key"] = "world"
}
```

This is equivalent to writing:

```
{
  hello += {
    key = "world"
  }
}
```

If the `hello` attribute does not already have a value, it is automatically initialized to an empty dictionary.

17.5 Template Imports

Objects can import attributes from other objects.

Example:

```
template Host "default-host" {
  vars.colour = "red"
}

template Host "test-host" {
  import "default-host"

  vars.colour = "blue"
}
```

```
object Host "localhost" {
  import "test-host"

  address = "127.0.0.1"
  address6 = "::1"
}
```

The `default-host` and `test-host` objects are marked as templates using the `template` keyword. Unlike ordinary objects templates are not instantiated at run-time. Parent objects do not necessarily have to be templates, however in general they are.

The `vars` dictionary for the `localhost` object contains all three custom attributes and the custom attribute `colour` has the value `"blue"`.

Parent objects are resolved in the order they're specified using the `import` keyword.

Default templates which are automatically imported into all object definitions can be specified using the `default` keyword:

```
template CheckCommand "plugin-check-command" default {
  // ...
}
```

Default templates are imported before any other user-specified statement in an object definition is evaluated.

If there are multiple default templates the order in which they are imported is unspecified.

17.6 Constants

Global constants can be set using the `const` keyword:

```
const VarName = "some value"
```

Once defined a constant can be accessed from any file. Constants cannot be changed once they are set.

Tip

Best practice is to manage constants in the constants.conf file.

17.6.1 Icinga 2 Specific Constants

Icinga 2 provides a number of special global constants. Some of them can be overridden using the `--define` command line parameter:

Variable	Description
PrefixDir	**Read-only.** Contains the installation prefix that was specified with cmake -DCMAKE_INSTALL_PREFIX. Defaults to "/usr/local".
SysconfDir	**Read-only.** Contains the path of the sysconf directory. Defaults to PrefixDir + "/etc".
ZonesDir	**Read-only.** Contains the path of the zones.d directory. Defaults to SysconfDir + "/zones.d".
LocalStateDir	**Read-only.** Contains the path of the local state directory. Defaults to PrefixDir + "/var".
RunDir	**Read-only.** Contains the path of the run directory. Defaults to LocalStateDir + "/run".
PkgDataDir	**Read-only.** Contains the path of the package data directory. Defaults to PrefixDir + "/share/icinga2".
StatePath	**Read-write.** Contains the path of the Icinga 2 state file. Defaults to LocalStateDir + "/lib/icinga2/icinga2.state".

Variable	Description
ObjectsPath	**Read-write.** Contains the path of the Icinga 2 objects file. Defaults to LocalStateDir + "/cache/icinga2/icinga2.debug".
PidPath	**Read-write.** Contains the path of the Icinga 2 PID file. Defaults to RunDir + "/icinga2/icinga2.pid".
Vars	**Read-write.** Contains a dictionary with global custom attributes. Not set by default.
NodeName	**Read-write.** Contains the cluster node name. Set to the local hostname by default.
RunAsUser	**Read-write.** Defines the user the Icinga 2 daemon is running as. Set in the Icinga 2 sysconfig.
RunAsGroup	**Read-write.** Defines the group the Icinga 2 daemon is running as. Set in the Icinga 2 sysconfig.
PlatformName	**Read-only.** The name of the operating system, e.g. "Ubuntu".
PlatformVersion	**Read-only.** The version of the operating system, e.g. "14.04.3 LTS".

Variable	Description
PlatformKernel	**Read-only.** The name of the operating system kernel, e.g. “Linux”.
PlatformKernelVersion	**Read-only.** The version of the operating system kernel, e.g. “3.13.0-63-generic”.
BuildCompilerName	**Read-only.** The name of the compiler Icinga was built with, e.g. “Clang”.
BuildCompilerVersion	**Read-only.** The version of the compiler Icinga was built with, e.g. “7.3.0.7030031”.
BuildHostName	**Read-only.** The name of the host Icinga was built on, e.g. “acheron”.
MaxConcurrentChecks	**Read-write**. The number of max checks run simultaneously. Defaults to 512.

Advanced runtime constants. Please only use them if advised by support or developers.

Variable	Description
EventEngine	**Read-write.** The name of the socket event engine, can be `poll` or `epoll`. The epoll interface is only supported on Linux.

Variable	Description
AttachDebugger	**Read-write.** Whether to attach a debugger when Icinga 2 crashes. Defaults to `false`.
ICINGA2_RLIMIT_FILES	**Read-write.** Defines the resource limit for RLIMIT_NOFILE that should be set at start-up. Value cannot be set lower than the default `16 * 1024`. 0 disables the setting. Set in Icinga 2 sysconfig.
ICINGA2_RLIMIT_PROCESSES	**Read-write.** Defines the resource limit for RLIMIT_NPROC that should be set at start-up. Value cannot be set lower than the default `16 * 1024`. 0 disables the setting. Set in Icinga 2 sysconfig.
ICINGA2_RLIMIT_STACK	**Read-write.** Defines the resource limit for RLIMIT_STACK that should be set at start-up. Value cannot be set lower than the default `256 * 1024`. 0 disables the setting. Set in Icinga 2 sysconfig.

17.7 Apply

The `apply` keyword can be used to create new objects which are associated with another group of objects.

```
apply Service "ping" to Host {
  import "generic-service"

  check_command = "ping4"

  assign where host.name == "localhost"
}
```

In this example the `assign where` condition is a boolean expression which is evaluated for all objects of type `Host` and a new service with name "ping" is created for each matching host. Expression operators may be used in `assign where` conditions.

The `to` keyword and the target type may be omitted if there is only one target type, e.g. for the `Service` type.

Depending on the object type used in the `apply` expression additional local variables may be available for use in the `where` condition:

Source Type	Target Type	Variables
Service	Host	host
Dependency	Host	host
Dependency	Service	host, service
Notification	Host	host
Notification	Service	host, service
ScheduledDowntime	Host	host
ScheduledDowntime	Service	host, service

Any valid config attribute can be accessed using the `host` and `service` variables. For example, `host.address` would return the value of the host's "address" attribute – or null if that attribute isn't set.

More usage examples are documented in the monitoring basics chapter.

17.8 Apply For

Apply rules can be extended with the for loop keyword.

```
apply Service "prefix-" for (key => value in host.vars.dictionary) to Host {
  import "generic-service"

  check_command = "ping4"
  vars.host_value = value
}
```

Any valid config attribute can be accessed using the `host` and `service` variables. The attribute must be of the Array or Dictionary type. In this example `host.vars.dictionary` is of the Dictionary type which needs a key-value-pair as iterator.

In this example all generated service object names consist of `prefix-` and the value of the `key` iterator. The prefix string can be omitted if not required.

The `key` and `value` variables can be used for object attribute assignment, e.g. for setting the `check_command` attribute or custom attributes as command parameters.

`apply for` rules are first evaluated against all objects matching the `for loop` list and afterwards the `assign where` and `ignore where` conditions are evaluated.

It is not necessary to check attributes referenced in the `for loop` expression for their existance using an additional `assign where` condition.

More usage examples are documented in the monitoring basics chapter.

17.9 Group Assign

Group objects can be assigned to specific member objects using the `assign where` and `ignore where` conditions.

```
object HostGroup "linux-servers" {
  display_name = "Linux Servers"

  assign where host.vars.os == "Linux"
}
```

In this example the `assign where` condition is a boolean expression which is evaluated for all objects of the type `Host`. Each matching host is added as member to the host group with the name "linux-servers". Membership exclusion can be controlled using the `ignore where` condition. Expression operators may be used in `assign where` and `ignore where` conditions.

Source Type	Variables
HostGroup	host
ServiceGroup	host, service
UserGroup	user

17.10 Boolean Values

The `assign where`, `ignore where`, `if` and `while` statements, the `!` operator as well as the `bool()` function convert their arguments to a boolean value based

on the following rules:

Description	Example Value	Boolean Value
Empty value	null	false
Zero	0	false
Non-zero integer	-23945	true
Empty string	“”	false
Non-empty string	“Hello”	true
Empty array	[]	false
Non-empty array	[“Hello”]	true
Empty dictionary	{}	false
Non-empty dictionary	{ key = “value” }	true

For a list of supported expression operators for `assign where` and `ignore where` statements, see expression operators.

17.11 Comments

The Icinga 2 configuration format supports C/C++-style and shell-style comments.

Example:

```
/*
 This is a comment.
 */
object Host "localhost" {
  check_interval = 30 // this is also a comment.
  retry_interval = 15 # yet another comment
}
```

17.12 Includes

Other configuration files can be included using the `include` directive. Paths must be relative to the configuration file that contains the `include` directive.

Example:

```
include "some/other/file.conf"
include "conf.d/*.conf"
```

Wildcard includes are not recursive.

Icinga also supports include search paths similar to how they work in a C/C++ compiler:

```
include <itl>
```

Note the use of angle brackets instead of double quotes. This causes the config compiler to search the include search paths for the specified file. By default $PREFIX/share/icinga2/include is included in the list of search paths. Additional include search paths can be added using command-line options.

Wildcards are not permitted when using angle brackets.

17.13 Recursive Includes

The `include_recursive` directive can be used to recursively include all files in a directory which match a certain pattern.

Example:

```
include_recursive "conf.d", "*.conf"
include_recursive "templates"
```

The first parameter specifies the directory from which files should be recursively included.

The file names need to match the pattern given in the second parameter. When no pattern is specified the default pattern "*.conf" is used.

17.14 Zone Includes

The `include_zones` recursively includes all subdirectories for the given path.

In addition to that it sets the `zone` attribute for all objects created in these subdirectories to the name of the subdirectory.

Example:

```
include_zones "etc", "zones.d", "*.conf"
include_zones "puppet", "puppet-zones"
```

The first parameter specifies a tag name for this directive. Each `include_zones` invocation should use a unique tag name. When copying the zones' configuration files Icinga uses the tag name as the name for the destination directory in `/var/lib/icinga2/api/config`.

The second parameter specifies the directory which contains the subdirectories.

The file names need to match the pattern given in the third parameter. When no pattern is specified the default pattern "*.conf" is used.

17.15 Library directive

The `library` directive was used to manually load additional libraries. Starting with version 2.9 it is no longer necessary to explicitly load libraries and this directive has no effect.

17.16 Functions

Functions can be defined using the `function` keyword.

Example:

```
function multiply(a, b) {
  return a * b
}
```

When encountering the `return` keyword further execution of the function is terminated and the specified value is supplied to the caller of the function:

```
log(multiply(3, 5))
```

In this example the `multiply` function we declared earlier is invoked with two arguments (3 and 5). The function computes the product of those arguments and makes the result available to the function's caller.

When no value is supplied for the `return` statement the function returns `null`.

Functions which do not have a `return` statement have their return value set to the value of the last expression which was performed by the function. For example, we could have also written our `multiply` function like this:

```
function multiply(a, b) {
  a * b
}
```

Anonymous functions can be created by omitting the name in the function definition. The resulting function object can be used like any other value:

```
var fn = function() { 3 }

fn() /* Returns 3 */
```

17.17 Lambda Expressions

Functions can also be declared using the alternative lambda syntax.

Example:

```
f = (x) => x * x
```

Multiple statements can be used by putting the function body into braces:

```
f = (x) => {
  log("Lambda called")
  x * x
}
```

Just like with ordinary functions the return value is the value of the last statement.

For lambdas which take exactly one argument the braces around the arguments can be omitted:

```
f = x => x * x
```

17.18 Abbreviated Lambda Syntax

Lambdas which take no arguments can also be written using the abbreviated lambda syntax.

Example:

```
f = {{ 3 }}
```

This creates a new function which returns the value 3.

17.19 Variable Scopes

When setting a variable Icinga checks the following scopes in this order whether the variable already exists there:

- Local Scope
- `this` Scope
- Global Scope

The local scope contains variables which only exist during the invocation of the current function, object or apply statement. Local variables can be declared using the `var` keyword:

```
function multiply(a, b) {
  var temp = a * b
  return temp
}
```

Each time the `multiply` function is invoked a new `temp` variable is used which is in no way related to previous invocations of the function.

When setting a variable which has not previously been declared as local using the `var` keyword the `this` scope is used.

The `this` scope refers to the current object which the function or object/apply statement operates on.

```
object Host "localhost" {
  check_interval = 5m
}
```

In this example the `this` scope refers to the "localhost" object. The `check_interval` attribute is set for this particular host.

You can explicitly access the `this` scope using the `this` keyword:

```
object Host "localhost" {
  var check_interval = 5m

  /* This explicitly specifies that the attribute should be set
   * for the host, if we had omitted `this.` the (poorly named)
  * local variable `check_interval` would have been modified instead.
   */
  this.check_interval = 1m

}
```

Similarly the keywords `locals` and `globals` are available to access the local and global scope.

Functions also have a `this` scope. However unlike for object/apply statements the `this` scope for a function is set to whichever object was used to invoke the function. Here's an example:

```
hm = {
  h_word = null

  function init(word) {
    h_word = word
  }
}

/* Let's invoke the init() function */
hm.init("hello")
```

We're using `hm.init` to invoke the function which causes the value of `hm` to become the `this` scope for this function call.

17.20 Closures

By default `functions`, `objects` and `apply` rules do not have access to variables declared outside of their scope (except for global variables).

In order to access variables which are defined in the outer scope the `use` keyword can be used:

```
function MakeHelloFunction(name) {
  return function() use(name) {
    log("Hello, " + name)
  }
}
```

In this case a new variable `name` is created inside the inner function's scope which has the value of the `name` function argument.

Alternatively a different value for the inner variable can be specified:

```
function MakeHelloFunction(name) {
  return function() use (greeting = "Hello, " + name) {
    log(greeting)
  }
}
```

17.21 Conditional Statements

Sometimes it can be desirable to only evaluate statements when certain conditions are met. The if/else construct can be used to accomplish this.

Example:

```
a = 3

if (a < 5) {
  a *= 7
} else if (a > 10) {
  a *= 5
} else {
  a *= 2
}
```

An if/else construct can also be used in place of any other value. The value of an if/else statement is the value of the last statement which was evaluated for the branch which was taken:

```
a = if (true) {
  log("Taking the 'true' branch")
  7 * 3
} else {
  log("Taking the 'false' branch")
  9
}
```

This example prints the log message "Taking the 'true' branch" and the `a` variable is set to 21 (7 * 3).

The value of an if/else construct is null if the condition evaluates to false and no else branch is given.

17.22 While Loops

The `while` statement checks a condition and executes the loop body when the condition evaluates to `true`. This is repeated until the condition is no longer true.

Example:

```
var num = 5

while (num > 5) {
    log("Test")
    num -= 1
}
```

The `continue` and `break` keywords can be used to control how the loop is executed: The `continue` keyword skips over the remaining expressions for the loop body and begins the next loop evaluation. The `break` keyword breaks out of the loop.

17.23 For Loops

The `for` statement can be used to iterate over arrays and dictionaries.

Example:

```
var list = [ "a", "b", "c" ]

for (item in list) {
  log("Item: " + item)
}
```

The loop body is evaluated once for each item in the array. The variable `item` is declared as a local variable just as if the `var` keyword had been used.

Iterating over dictionaries can be accomplished in a similar manner:

```
var dict = { a = 3, b = 7 }

for (key => value in dict) {
  log("Key: " + key + ", Value: " + value)
}
```

The `continue` and `break` keywords can be used to control how the loop is executed: The `continue` keyword skips over the remaining expressions for the loop body and begins the next loop evaluation. The `break` keyword breaks out of the loop.

17.24 Constructors

In order to create a new value of a specific type constructor calls may be used.

Example:

```
var pd = PerfdataValue()
pd.label = "test"
pd.value = 10
```

You can also try to convert an existing value to another type by specifying it as an argument for the constructor call.

Example:

```
var s = String(3) /* Sets s to "3". */
```

17.25 Throwing Exceptions

Built-in commands may throw exceptions to signal errors such as invalid arguments. User scripts can throw exceptions using the `throw` keyword.

Example:

```
throw "An error occurred."
```

17.26 Handling Exceptions

Exceptions can be handled using the `try` and `except` keywords. When an exception occurs while executing code in the `try` clause no further statements in the `try` clause are evaluated and the `except` clause is executed instead.

Example:

```
try {
    throw "Test"

    log("This statement won't get executed.")
} except {
    log("An error occurred in the try clause.")
}
```

17.27 Breakpoints

The `debugger` keyword can be used to insert a breakpoint. It may be used at any place where an assignment would also be a valid expression.

By default breakpoints have no effect unless Icinga is started with the `--script-debugger` command-line option. When the script debugger is enabled Icinga stops execution of the script when it encounters a breakpoint and spawns a console which lets the user inspect the current state of the execution environment.

17.28 Types

All values have a static type. The `typeof` function can be used to determine the type of a value:

```
typeof(3) /* Returns an object which represents the type for numbers */
```

The following built-in types are available:

Type	Examples	Description
Number	3.7	A numerical value.
Boolean	true, false	A boolean value.
String	“hello”	A string.
Array	[“a”, “b”]	An array.
Dictionary	{ a = 3 }	A dictionary.

Depending on which libraries are loaded additional types may become available. The `icinga` library implements a whole bunch of other object types, e.g. Host, Service, CheckCommand, etc.

Each type has an associated type object which describes the type's semantics. These type objects are made available using global variables which match the type's name:

```
/* This logs 'true' */
log(typeof(3) == Number)
```

The type object's `prototype` property can be used to find out which methods a certain type supports:

```
/* This returns: ["contains","find","len","lower","replace","reverse","split","substr","to_st
keys(String.prototype)
```

Additional documentation on type methods is available in the library reference.

17.29 Location Information

The location of the currently executing script can be obtained using the `current_filename` and `current_line` keywords.

Example:

```
log("Hello from '" + current_filename + "' in line " + current_line)
```

17.30 Reserved Keywords

These keywords are reserved and must not be used as constants or custom attributes.

```
object
template
include
include_recursive
include_zones
library
null
true
false
const
var
this
globals
locals
use
default
ignore_on_error
current_filename
current_line
apply
to
where
import
assign
ignore
function
return
break
continue
for
if
else
```

```
while
throw
try
except
in
```

You can escape reserved keywords using the `@` character. The following example tries to set `vars.include` which references a reserved keyword and generates an error:

```
[2014-09-15 17:24:00 +0200] critical/config: Location:
/etc/icinga2/conf.d/hosts/localhost.conf(13):   vars.sla = "24x7"
/etc/icinga2/conf.d/hosts/localhost.conf(14):
/etc/icinga2/conf.d/hosts/localhost.conf(15):  vars.include = "some cmdb export field"
                                                      ^^^^^^^
/etc/icinga2/conf.d/hosts/localhost.conf(16): }
/etc/icinga2/conf.d/hosts/localhost.conf(17):

Config error: in /etc/icinga2/conf.d/hosts/localhost.conf: 15:8-15:14: syntax error, unexpecte
[2014-09-15 17:24:00 +0200] critical/config: 1 errors, 0 warnings.
```

You can escape the `include` keyword by prefixing it with an additional `@` character:

```
object Host "localhost" {
  import "generic-host"

  address = "127.0.0.1"
  address6 = "::1"

  vars.os = "Linux"
  vars.sla = "24x7"

  vars.@include = "some cmdb export field"
}
```

18 Library Reference

18.1 Global functions

These functions are globally available in assign/ignore where expressions, functions, API filters and the Icinga 2 debug console.

You can use the Icinga 2 debug console as a sandbox to test these functions before implementing them in your scenarios.

18.1.1 regex

Signature:

```
function regex(pattern, value, mode)
```

Returns true if the regular expression `pattern` matches the `value`, false otherwise. The `value` can be of the type String or Array (which contains string elements).

The `mode` argument is optional and can be either `MatchAll` (in which case all elements for an array have to match) or `MatchAny` (in which case at least one element has to match). The default mode is `MatchAll`.

Tip: In case you are looking for regular expression tests try regex101.

Example for string values:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => host.vars.os_type = "Linux/Unix"
null
<2> => regex("^Linux", host.vars.os_type)
true
<3> => regex("^Linux$", host.vars.os_type)
false
```

Example for an array of string values:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => host.vars.databases = [ "db-prod1", "db-prod2", "db-dev" ]
null
<2> => regex("^db-prod\\d+", host.vars.databases, MatchAny)
true
<3> => regex("^db-prod\\d+", host.vars.databases, MatchAll)
false
```

18.1.2 match

Signature:

```
function match(pattern, text, mode)
```

Returns true if the wildcard (`?*`) `pattern` matches the `value`, false otherwise. The `value` can be of the type String or Array (which contains string elements).

The `mode` argument is optional and can be either `MatchAll` (in which case all elements for an array have to match) or `MatchAny` (in which case at least one element has to match). The default mode is `MatchAll`.

Example for string values:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var name = "db-prod-sfo-657"
null
<2> => match("*prod-sfo*", name)
true
<3> => match("*-dev-*", name)
false
```

Example for an array of string values:

```
$ icinga2 console
Icinga 2 (version: v2.7.0-28)
<1> => host.vars.application_types = [ "web-wp", "web-rt", "db-local" ]
null
<2> => match("web-*", host.vars.application_types, MatchAll)
false
<3> => match("web-*", host.vars.application_types, MatchAny)
true
```

18.1.3 cidr_match

Signature:

```
function cidr_match(pattern, ip, mode)
```

Returns true if the CIDR pattern matches the IP address, false otherwise.

IPv4 addresses are converted to IPv4-mapped IPv6 addresses before being matched against the pattern. The `mode` argument is optional and can be either `MatchAll` (in which case all elements for an array have to match) or `MatchAny` (in which case at least one element has to match). The default mode is `MatchAll`.

Example for a single IP address:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => host.address = "192.168.56.101"
null
<2> => cidr_match("192.168.56.0/24", host.address)
true
<3> => cidr_match("192.168.56.0/26", host.address)
false
```

Example for an array of IP addresses:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
```

```
<1> => host.vars.vhost_ips = [ "192.168.56.101", "192.168.56.102", "10.0.10.99" ]
null
<2> => cidr_match("192.168.56.0/24", host.vars.vhost_ips, MatchAll)
false
<3> => cidr_match("192.168.56.0/24", host.vars.vhost_ips, MatchAny)
true
```

18.1.4 range

Signature:

```
function range(end)
function range(start, end)
function range(start, end, increment)
```

Returns an array of numbers in the specified range. If you specify one parameter, the first element starts at `0`. The following array numbers are incremented by `1` and stop before the specified end. If you specify the start and end numbers, the returned array number are incremented by `1`. They start at the specified start number and stop before the end number. Optionally you can specify the incremented step between numbers as third parameter.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => range(5)
[ 0.000000, 1.000000, 2.000000, 3.000000, 4.000000 ]
<2> => range(2,4)
[ 2.000000, 3.000000 ]
<3> => range(2,10,2)
[ 2.000000, 4.000000, 6.000000, 8.000000 ]
```

18.1.5 len

Signature:

```
function len(value)
```

Returns the length of the value, i.e. the number of elements for an array or dictionary, or the length of the string in bytes.

Note: Instead of using this global function you are advised to use the type's prototype method: Array#len, Dictionary#len and String#len.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
```

```
<1> => host.groups = [ "linux-servers", "db-servers" ]
null
<2> => host.groups.len()
2.000000
<3> => host.vars.disks["/"] = {}
null
<4> => host.vars.disks["/var"] = {}
null
<5> => host.vars.disks.len()
2.000000
<6> => host.vars.os_type = "Linux/Unix"
null
<7> => host.vars.os_type.len()
10.000000
```

18.1.6 union

Signature:

```
function union(array, array, ...)
```

Returns an array containing all unique elements from the specified arrays.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var dev_notification_groups = [ "devs", "slack" ]
null
<2> => var host_notification_groups = [ "slack", "noc" ]
null
<3> => union(dev_notification_groups, host_notification_groups)
[ "devs", "noc", "slack" ]
```

18.1.7 intersection

Signature:

```
function intersection(array, array, ...)
```

Returns an array containing all unique elements which are common to all specified arrays.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var dev_notification_groups = [ "devs", "slack" ]
```

```
null
<2> => var host_notification_groups = [ "slack", "noc" ]
null
<3> => intersection(dev_notification_groups, host_notification_groups)
[ "slack" ]
```

18.1.8 keys

Signature:

```
function keys(dict)
```

Returns an array containing the dictionary's keys.

Note: Instead of using this global function you are advised to use the type's prototype method: Dictionary#keys.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => host.vars.disks["/"] = {}
null
<2> => host.vars.disks["/var"] = {}
null
<3> => host.vars.disks.keys()
[ "/", "/var" ]
```

18.1.9 string

Signature:

```
function string(value)
```

Converts the value to a string.

Note: Instead of using this global function you are advised to use the type's prototype method:

- Number#to_string
- Boolean#to_string
- String#to_string
- Object#to_string for Array and Dictionary types
- DateTime#to_string

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => 5.to_string()
```

```
"5"
<2> => false.to_string()
"false"
<3> => "abc".to_string()
"abc"
<4> => [ "dev", "slack" ].to_string()
"[ \"dev\", \"slack\" ]"
<5> => { "/" = {}, "/var" = {} }.to_string()
"{\n\t\"/\" = {\n\t}\n\t\"/var\" = {\n\t}\n}"
<6> => DateTime(2016, 11, 25).to_string()
"2016-11-25 00:00:00 +0100"
```

18.1.10 number

Signature:

```
function number(value)
```

Converts the value to a number.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => number(false)
0.000000
<2> => number("78")
78.000000
```

18.1.11 bool

Signature:

```
function bool(value)
```

Converts the value to a bool.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => bool(1)
true
<2> => bool(0)
false
```

18.1.12 random

Signature:

```
function random()
```

Returns a random value between 0 and RAND_MAX (as defined in stdlib.h).

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => random()
1263171996.000000
<2> => random()
108402530.000000
```

18.1.13 log

Signature:

```
function log(value)
```

Writes a message to the log. Non-string values are converted to a JSON string.

Signature:

```
function log(severity, facility, value)
```

Writes a message to the log. `severity` can be one of `LogDebug`, `LogNotice`, `LogInformation`, `LogWarning`, and `LogCritical`.

Non-string values are converted to a JSON string.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => log(LogCritical, "Console", "First line")
critical/Console: First line
null
<2> => var groups = [ "devs", "slack" ]
null
<3> => log(LogCritical, "Console", groups)
critical/Console: ["devs","slack"]
null
```

18.1.14 typeof

Signature:

```
function typeof(value)
```

Returns the Type object for a value.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => typeof(3) == Number
true
<2> => typeof("str") == String
true
<3> => typeof(true) == Boolean
true
<4> => typeof([ 1, 2, 3]) == Array
true
<5> => typeof({ a = 2, b = 3 }) == Dictionary
true
```

18.1.15 get_time

Signature:

```
function get_time()
```

Returns the current UNIX timestamp as floating point number.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => get_time()
1480072135.633008
<2> => get_time()
1480072140.401207
```

18.1.16 parse_performance_data

Signature:

```
function parse_performance_data(pd)
```

Parses a performance data string and returns an array describing the values.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var pd = "'time'=1480074205.197363;;;"
null
<2> => parse_performance_data(pd)
```

```
{
    counter = false
    crit = null
    label = "time"
    max = null
    min = null
    type = "PerfdataValue"
    unit = ""
    value = 1480074205.197363
    warn = null
}
```

18.1.17 dirname

Signature:

```
function dirname(path)
```

Returns the directory portion of the specified path.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var path = "/etc/icinga2/scripts/xmpp-notification.pl"
null
<2> => dirname(path)
"/etc/icinga2/scripts"
```

18.1.18 basename

Signature:

```
function basename(path)
```

Returns the filename portion of the specified path.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var path = "/etc/icinga2/scripts/xmpp-notification.pl"
null
<2> => basename(path)
"xmpp-notification.pl"
```

18.1.19 path_exists

Signature:

```
function path_exists(path)
```

Returns true if the specified path exists, false otherwise.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var path = "/etc/icinga2/scripts/xmpp-notification.pl"
null
<2> => path_exists(path)
true
```

18.1.20 glob

Signature:

```
function glob(pathSpec, type)
```

Returns an array containing all paths which match the `pathSpec` argument.

The `type` argument is optional and specifies which types of paths are matched. This can be a combination of the `GlobFile` and `GlobDirectory` constants. The default value is `GlobFile | GlobDirectory`.

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var pathSpec = "/etc/icinga2/conf.d/*.conf"
null
<2> => glob(pathSpec)
[ "/etc/icinga2/conf.d/app.conf", "/etc/icinga2/conf.d/commands.conf", ... ]
```

18.1.21 glob_recursive

Signature:

```
function glob_recursive(path, pattern, type)
```

Recursively descends into the specified directory and returns an array containing all paths which match the `pattern` argument.

The `type` argument is optional and specifies which types of paths are matched. This can be a combination of the `GlobFile` and `GlobDirectory` constants. The default value is `GlobFile | GlobDirectory`.

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => var path = "/etc/icinga2/zones.d/"
null
<2> => var pattern = "*.conf"
null
<3> => glob_recursive(path, pattern)
[ "/etc/icinga2/zones.d/global-templates/templates.conf", "/etc/icinga2/zones.d/master/hosts
```

18.1.22 escape_shell_arg

Signature:

```
function escape_shell_arg(text)
```

Escapes a string for use as a single shell argument.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => escape_shell_arg("'$host.name$' '$service.name$'")
"''\\''$host.name$'\\'' '\\''$service.name$'\\'''"
```

18.1.23 escape_shell_cmd

Signature:

```
function escape_shell_cmd(text)
```

Escapes shell meta characters in a string.

Example:

```
$ icinga2 console
Icinga 2 (version: v2.7.0)
<1> => escape_shell_cmd("/bin/echo 'shell test' $ENV")
"/bin/echo 'shell test' \\$ENV"
```

18.1.24 escape_create_process_arg

Signature:

```
function escape_create_process_arg(text)
```

Escapes a string for use as an argument for CreateProcess(). Windows only.

18.1.25 sleep

Signature:

`function sleep(interval)`

Sleeps for the specified amount of time (in seconds).

18.2 Scoped Functions

This chapter describes functions which are only available in a specific scope.

18.2.1 macro

Signature:

`function macro("$macro_name$")`

The `macro` function can be used to resolve runtime macro strings into their values. The returned value depends on the attribute value which is resolved from the specified runtime macro.

This function is only available in runtime evaluated functions, e.g. for custom attributes which use the abbreviated lambda syntax.

This example sets the `snmp_address` custom attribute based on `$address$` and `$address6`.

```
vars.snmp_address = {{
  var addr_v4 = macro("$address$")
  var addr_v6 = macro("$address6$")

  if (addr_v4) {
      return addr_v4
  } else {
      return "udp6:[" + addr_v6 + "]"
  }
}}
```

More reference examples are available inside the Icinga Template Library and the object accessors chapter.

18.3 Object Accessor Functions

These functions can be used to retrieve a reference to another object by name.

18.3.1 get_check_command

Signature:

```
function get_check_command(name);
```

Returns the CheckCommand object with the specified name, or `null` if no such CheckCommand object exists.

18.3.2 get_event_command

Signature:

```
function get_event_command(name);
```

Returns the EventCommand object with the specified name, or `null` if no such EventCommand object exists.

18.3.3 get_notification_command

Signature:

```
function get_notification_command(name);
```

Returns the NotificationCommand object with the specified name, or `null` if no such NotificationCommand object exists.

18.3.4 get_host

Signature:

```
function get_host(host_name);
```

Returns the Host object with the specified name, or `null` if no such Host object exists.

18.3.5 get_service

Signature:

```
function get_service(host_name, service_name);
function get_service(host, service_name);
```

Returns the Service object with the specified host name or object and service name pair, or `null` if no such Service object exists.

Example in the debug console which fetches the `disk` service object from the current Icinga 2 node:

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/'
Icinga 2 (version: v2.7.0)

<1> => get_service(NodeName, "disk")
<2> => get_service(NodeName, "disk").__name
"icinga2-master1.localdomain!disk"

<3> => get_service(get_host(NodeName), "disk").__name
"icinga2-master1.localdomain!disk"
```

18.3.6 get_services

Signature:

```
function get_services(host_name);
function get_services(host);
```

Returns an array of service objects for the specified host name or object, or `null` if no such host object exists.

Example in the debug console which fetches all service objects from the current Icinga 2 node:

```
$ ICINGA2_API_PASSWORD=icinga icinga2 console --connect 'https://root@localhost:5665/'
Icinga 2 (version: v2.7.0)

<1> => get_services(NodeName).map(s => s.name)
[ "disk", "disk /", "http", "icinga", "load", "ping4", "ping6", "procs", "ssh", "users" ]
```

Note: map takes a lambda function as argument. In this example we only want to collect and print the `name` attribute with `s => s.name`.

This works in a similar fashion for a host object where you can extract all service states in using the map functionality:

```
<2> => get_services(get_host(NodeName)).map(s => s.state)
[ 2.000000, 2.000000, 2.000000, 0.000000, 0.000000, 0.000000, 2.000000, 0.000000, 0.000000, 1.0
```

18.3.7 get_user

Signature:

```
function get_user(name);
```

Returns the User object with the specified name, or `null` if no such User object exists.

18.3.8 get_host_group

Signature:

```
function get_host_group(name);
```

Returns the HostGroup object with the specified name, or `null` if no such HostGroup object exists.

18.3.9 get_service_group

Signature:

```
function get_service_group(name);
```

Returns the ServiceGroup object with the specified name, or `null` if no such ServiceGroup object exists.

18.3.10 get_user_group

Signature:

```
function get_user_group(name);
```

Returns the UserGroup object with the specified name, or `null` if no such UserGroup object exists.

18.3.11 get_time_period

Signature:

```
function get_time_period(name);
```

Returns the TimePeriod object with the specified name, or `null` if no such TimePeriod object exists.

18.3.12 get_object

Signature:

```
function get_object(type, name);
```

Returns the object with the specified type and name, or `null` if no such object exists. `type` must refer to a type object.

18.3.13 get_objects

Signature:

```
function get_objects(type);
```

Returns an array of objects whose type matches the specified type. `type` must refer to a type object.

18.4 Math object

The global `Math` object can be used to access a number of mathematical constants and functions.

18.4.1 Math.E

Euler's constant.

18.4.2 Math.LN2

Natural logarithm of 2.

18.4.3 Math.LN10

Natural logarithm of 10.

18.4.4 Math.LOG2E

Base 2 logarithm of E.

18.4.5 Math.PI

The mathematical constant Pi.

18.4.6 Math.SQRT1_2

Square root of 1/2.

18.4.7 Math.SQRT2

Square root of 2.

18.4.8 Math.abs

Signature:

```
function abs(x);
```

Returns the absolute value of `x`.

18.4.9 Math.acos

Signature:

```
function acos(x);
```

Returns the arccosine of `x`.

18.4.10 Math.asin

Signature:

```
function asin(x);
```

Returns the arcsine of `x`.

18.4.11 Math.atan

Signature:

```
function atan(x);
```

Returns the arctangent of `x`.

18.4.12 Math.atan2

Signature:

```
function atan2(y, x);
```

Returns the arctangent of the quotient of `y` and `x`.

18.4.13 Math.ceil

Signature:

```
function ceil(x);
```

Returns the smallest integer value not less than `x`.

18.4.14 Math.cos

Signature:

```
function cos(x);
```

Returns the cosine of `x`.

18.4.15 Math.exp

Signature:

```
function exp(x);
```

Returns E raised to the `x`th power.

18.4.16 Math.floor

Signature:

```
function floor(x);
```

Returns the largest integer value not greater than `x`.

18.4.17 Math.isinf

Signature:

```
function isinf(x);
```

Returns whether `x` is infinite.

18.4.18 Math.isnan

Signature:

```
function isnan(x);
```

Returns whether `x` is NaN (not-a-number).

18.4.19 Math.log

Signature:

```
function log(x);
```

Returns the natural logarithm of `x`.

18.4.20 Math.max

Signature:

```
function max(...);
```

Returns the largest argument. A variable number of arguments can be specified. If no arguments are given, -Infinity is returned.

18.4.21 Math.min

Signature:

```
function min(...);
```

Returns the smallest argument. A variable number of arguments can be specified. If no arguments are given, +Infinity is returned.

18.4.22 Math.pow

Signature:

```
function pow(x, y);
```

Returns `x` raised to the `y`th power.

18.4.23 Math.random

Signature:

```
function random();
```

Returns a pseudo-random number between 0 and 1.

18.4.24 Math.round

Signature:

```
function round(x);
```

Returns `x` rounded to the nearest integer value.

18.4.25 Math.sign

Signature:

```
function sign(x);
```

Returns -1 if `x` is negative, 1 if `x` is positive and 0 if `x` is 0.

18.4.26 Math.sin

Signature:

```
function sin(x);
```

Returns the sine of `x`.

18.4.27 Math.sqrt

Signature:

```
function sqrt(x);
```

Returns the square root of `x`.

18.4.28 Math.tan

Signature:

```
function tan(x);
```

Returns the tangent of `x`.

18.5 Json object

The global `Json` object can be used to encode and decode JSON.

18.5.1 Json.encode

Signature:

```
function encode(x);
```

Encodes an arbitrary value into JSON.

18.5.2 Json.decode

Signature:

```
function decode(x);
```

Decodes a JSON string.

18.6 Number type

18.6.1 Number#to_string

Signature:

```
function to_string();
```

The `to_string` method returns a string representation of the number.

Example:

```
var example = 7
example.to_string() /* Returns "7" */
```

18.7 Boolean type

18.7.1 Boolean#to_string

Signature:

```
function to_string();
```

The `to_string` method returns a string representation of the boolean value.

Example:

```
var example = true
example.to_string() /* Returns "true" */
```

18.8 String type

18.8.1 String#find

Signature:

```
function find(str, start);
```

Returns the zero-based index at which the string `str` was found in the string. If the string was not found, -1 is returned. `start` specifies the zero-based index at which `find` should start looking for the string (defaults to 0 when not specified).

Example:

```
"Hello World".find("World") /* Returns 6 */
```

18.8.2 String#contains

Signature:

```
function contains(str);
```

Returns `true` if the string `str` was found in the string. If the string was not found, `false` is returned. Use find for getting the index instead.

Example:

```
"Hello World".contains("World") /* Returns true */
```

18.8.3 String#len

Signature

```
function len();
```

Returns the length of the string in bytes. Note that depending on the encoding type of the string this is not necessarily the number of characters.

Example:

```
"Hello World".len() /* Returns 11 */
```

18.8.4 String#lower

Signature:

```
function lower();
```

Returns a copy of the string with all of its characters converted to lower-case.

Example:

```
"Hello World".lower() /* Returns "hello world" */
```

18.8.5 String#upper

Signature:

```
function upper();
```

Returns a copy of the string with all of its characters converted to upper-case.

Example:

```
"Hello World".upper() /* Returns "HELLO WORLD" */
```

18.8.6 String#replace

Signature:

```
function replace(search, replacement);
```

Returns a copy of the string with all occurences of the string specified in `search` replaced with the string specified in `replacement`.

18.8.7 String#split

Signature:

```
function split(delimiters);
```

Splits a string into individual parts and returns them as an array. The `delimiters` argument specifies the characters which should be used as delimiters between parts.

Example:

```
"x-7,y".split("-,") /* Returns [ "x", "7", "y" ] */
```

18.8.8 String#substr

Signature:

```
function substr(start, len);
```

Returns a part of a string. The `start` argument specifies the zero-based index at which the part begins. The optional `len` argument specifies the length of the part ("until the end of the string" if omitted).

Example:

```
"Hello World".substr(6) /* Returns "World" */
```

18.8.9 String#to_string

Signature:

```
function to_string();
```

Returns a copy of the string.

18.8.10 String#reverse

Signature:

```
function reverse();
```

Returns a copy of the string in reverse order.

18.8.11 String#trim

Signature:

```
function trim();
```

Removes trailing whitespaces and returns the string.

18.9 Object type

This is the base type for all types in the Icinga application.

18.9.1 Object#clone

Signature:

```
function clone();
```

Returns a copy of the object. Note that for object elements which are reference values (e.g. objects such as arrays or dictionaries) the entire object is recursively copied.

18.9.2 Object#to_string

Signature:

```
function to_string();
```

Returns a string representation for the object. Unless overridden this returns a string of the format "Object of type ‘’ " where is the name of the object's type.

Example:

```
[ 3, true ].to_string() /* Returns "[ 3.000000, true ]" */
```

18.9.3 Object#type

Signature:

```
String type;
```

Returns the object's type name. This attribute is read-only.

Example:

```
get_host("localhost").type /* Returns "Host" */
```

18.10 Type type

Inherits methods from the Object type.

The `Type` type provides information about the underlying type of an object or scalar value.

All types are registered as global variables. For example, in order to obtain a reference to the `String` type the global variable `String` can be used.

18.10.1 Type#base

Signature:

```
Type base;
```

Returns a reference to the type's base type. This attribute is read-only.

Example:

```
Dictionary.base == Object /* Returns true, because the Dictionary type inherits directly from th
```

18.10.2 Type#name

Signature:

```
String name;
```

Returns the name of the type.

18.10.3 Type#prototype

Signature:

```
Object prototype;
```

Returns the prototype object for the type. When an attribute is accessed on an object that doesn't exist the prototype object is checked to see if an attribute with the requested name exists. If it does, the attribute's value is returned.

The prototype functionality is used to implement methods.

Example:

```
3.to_string() /* Even though '3' does not have a to_string property the Number type's prototype c
```

18.11 Array type

Inherits methods from the Object type.

18.11.1 Array#add

Signature:

```
function add(value);
```

Adds a new value after the last element in the array.

18.11.2 Array#clear

Signature:

```
function clear();
```

Removes all elements from the array.

18.11.3 Array#shallow_clone

```
function shallow_clone();
```

Returns a copy of the array. Note that for elements which are reference values (e.g. objects such as arrays and dictionaries) only the references are copied.

18.11.4 Array#contains

Signature:

```
function contains(value);
```

Returns true if the array contains the specified value, false otherwise.

18.11.5 Array#freeze

Signature:

```
function freeze()
```

Disallows further modifications to this array. Trying to modify the array will result in an exception.

18.11.6 Array#len

Signature:

```
function len();
```

Returns the number of elements contained in the array.

18.11.7 Array#remove

Signature:

```
function remove(index);
```

Removes the element at the specified zero-based index.

18.11.8 Array#set

Signature:

```
function set(index, value);
```

Sets the element at the zero-based index to the specified value. The `index` must refer to an element which already exists in the array.

18.11.9 Array#get

Signature:

```
function get(index);
```

Retrieves the element at the specified zero-based index.

18.11.10 Array#sort

Signature:

```
function sort(less_cmp);
```

Returns a copy of the array where all items are sorted. The items are compared using the `<` (less-than) operator. A custom comparator function can be specified with the `less_cmp` argument.

18.11.11 Array#join

Signature:

```
function join(separator);
```

Joins all elements of the array using the specified separator.

18.11.12 Array#reverse

Signature:

```
function reverse();
```

Returns a new array with all elements of the current array in reverse order.

18.11.13 Array#map

Signature:

```
function map(func);
```

Calls `func(element)` for each of the elements in the array and returns a new array containing the return values of these function calls.

18.11.14 Array#reduce

Signature:

```
function reduce(func);
```

Reduces the elements of the array into a single value by calling the provided function `func` as `func(a, b)` repeatedly where `a` and `b` are elements of the array or results from previous function calls.

18.11.15 Array#filter

Signature:

```
function filter(func);
```

Returns a copy of the array containing only the elements for which `func(element)` is true.

18.11.16 Array#any

Signature:

```
function any(func);
```

Returns true if the array contains at least one element for which `func(element)` is true, false otherwise.

18.11.17 Array#all

Signature:

```
function all(func);
```

Returns true if the array contains only elements for which `func(element)` is true, false otherwise.

18.11.18 Array#unique

Signature:

```
function unique();
```

Returns a copy of the array with all duplicate elements removed. The original order of the array is not preserved.

18.12 Dictionary type

Inherits methods from the Object type.

18.12.1 Dictionary#shallow_clone

Signature:

```
function shallow_clone();
```

Returns a copy of the dictionary. Note that for elements which are reference values (e.g. objects such as arrays and dictionaries) only the references are copied.

18.12.2 Dictionary#contains

Signature:

```
function contains(key);
```

Returns true if a dictionary item with the specified `key` exists, false otherwise.

18.12.3 Dictionary#freeze

Signature:

```
function freeze()
```

Disallows further modifications to this dictionary. Trying to modify the dictionary will result in an exception.

18.12.4 Dictionary#len

Signature:

```
function len();
```

Returns the number of items contained in the dictionary.

18.12.5 Dictionary#remove

Signature:

```
function remove(key);
```

Removes the item with the specified `key`. Trying to remove an item which does not exist is a no-op.

18.12.6 Dictionary#set

Signature:

```
function set(key, value);
```

Creates or updates an item with the specified `key` and `value`.

18.12.7 Dictionary#get

Signature:

```
function get(key);
```

Retrieves the value for the specified `key`. Returns `null` if they `key` does not exist in the dictionary.

18.12.8 Dictionary#keys

Signature:

```
function keys();
```

Returns a list of keys for all items that are currently in the dictionary.

18.12.9 Dictionary#values

Signature:

```
function values();
```

Returns a list of values for all items that are currently in the dictionary.

18.13 Function type

Inherits methods from the Object type.

18.13.1 Function#call

Signature:

```
function call(thisArg, ...);
```

Invokes the function using an alternative `this` scope. The `thisArg` argument specifies the `this` scope for the function. All other arguments are passed directly to the function.

Example:

```
function set_x(val) {
  this.x = val
}

dict = {}

set_x.call(dict, 7) /* Invokes set_x using `dict` as `this` */
```

18.13.2 Function#callv

Signature:

```
function callv(thisArg, args);
```

Invokes the function using an alternative `this` scope. The `thisArg` argument specifies the `this` scope for the function. The items in the `args` array are passed to the function as individual arguments.

Example:

```
function set_x(val) {
  this.x = val
}

var dict = {}

var args = [ 7 ]

set_x.callv(dict, args) /* Invokes set_x using `dict` as `this` */
```

18.14 DateTime type

Inherits methods from the Object type.

18.14.1 DateTime constructor

Signature:

```
function DateTime()
function DateTime(unixTimestamp)
function DateTime(year, month, day)
function DateTime(year, month, day, hours, minutes, seconds)
```

Constructs a new DateTime object. When no arguments are specified for the constructor a new DateTime object representing the current time is created.

Example:

```
var d1 = DateTime() /* current time */
var d2 = DateTime(2016, 5, 21) /* midnight April 21st, 2016 (local time) */
```

18.14.2 DateTime arithmetic

Subtracting two DateTime objects yields the interval between them, in seconds.

Example:

```
var delta = DateTime() - DateTime(2016, 5, 21) /* seconds since midnight April 21st, 2016 */
```

Subtracting a number from a DateTime object yields a new DateTime object that is further in the past:

Example:

```
var dt = DateTime() - 2 * 60 * 60 /* Current time minus 2 hours */
```

Adding a number to a DateTime object yields a new DateTime object that is in the future:

Example:

```
var dt = DateTime() + 24 * 60 60 /* Current time plus 24 hours */
```

18.14.3 DateTime#format

Signature:

```
function format(fmt)
```

Returns a string representation for the DateTime object using the specified format string. The format string may contain format conversion placeholders as specified in strftime(3).

Example:

```
var s = DateTime(2016, 4, 21).format("%A") /* Sets s to "Thursday". */
```

18.14.4 DateTime#to_string

Signature:

```
function to_string()
```

Returns a string representation for the DateTime object. Uses a suitable default format.

Example:

```
var s = DateTime(2016, 4, 21).to_string() /* Sets s to "2016-04-21 00:00:00 +0200". */
```

19 Technical Concepts

This chapter provides insights into specific Icinga 2 components, libraries, features and any other technical concept and design.

19.1 Features

Features are implemented in specific libraries and can be enabled using CLI commands.

Features either write specific data or receive data.

Examples for writing data: DB IDO, Graphite, InfluxDB. GELF, etc. Examples for receiving data: REST API, etc.

The implementation of features makes use of existing libraries and functionality. This makes the code more abstract, but shorter and easier to read.

Features register callback functions on specific events they want to handle. For example the `GraphiteWriter` feature subscribes to new CheckResult events.

Each time Icinga 2 receives and processes a new check result, this event is triggered and forwarded to all subscribers.

The GraphiteWriter feature calls the registered function and processes the received data. Features which connect Icinga 2 to external interfaces normally parse and reformat the received data into an applicable format.

The GraphiteWriter uses a TCP socket to communicate with the carbon cache daemon of Graphite. The InfluxDBWriter is instead writing bulk metric messages to InfluxDB's HTTP API.

19.2 Cluster

19.2.1 Communication

Icinga 2 uses its own certificate authority (CA) by default. The public and private CA keys can be generated on the signing master.

Each node certificate must be signed by the private CA key.

Note: The following description uses `parent node` and `child node`. This also applies to nodes in the same cluster zone.

During the connection attempt, an SSL handshake is performed. If the public certificate of a child node is not signed by the same CA, the child node is not trusted and the connection will be closed.

If the SSL handshake succeeds, the parent node reads the certificate's common name (CN) of the child node and looks for a local Endpoint object name configuration.

If there is no Endpoint object found, further communication (runtime and config sync, etc.) is terminated.

The child node also checks the CN from the parent node's public certificate. If the child node does not find any local Endpoint object name configuration, it will not trust the parent node.

Both checks prevent accepting cluster messages from an untrusted source endpoint.

If an Endpoint match was found, there is one additional security mechanism in place: Endpoints belong to a Zone hierarchy.

Several cluster messages can only be sent "top down", others like check results are allowed being sent from the child to the parent node.

Once this check succeeds the cluster messages are exchanged and processed.

19.2.2 CSR Signing

In order to make things easier, Icinga 2 provides built-in methods to allow child nodes to request a signed certificate from the signing master.

Icinga 2 v2.8 introduces the possibility to request certificates from indirectly connected nodes. This is required for multi level cluster environments with masters, satellites and clients.

CSR Signing in general starts with the master setup. This step ensures that the master is in a working CSR signing state with:

- public and private CA key in `/var/lib/icinga2/ca`
- private `TicketSalt` constant defined inside the `api` feature
- Cluster communication is ready and Icinga 2 listens on port 5665

The child node setup which is run with CLI commands will now attempt to connect to the parent node. This is not necessarily the signing master instance, but could also be a parent satellite node.

During this process the child node asks the user to verify the parent node's public certificate to prevent MITM attacks.

There are two methods to request signed certificates:

- Add the ticket into the request. This ticket was generated on the master beforehand and contains hashed details for which client it has been created. The signing master uses this information to automatically sign the certificate request.
- Do not add a ticket into the request. It will be sent to the signing master which stores the pending request. Manual user interaction with CLI commands is necessary to sign the request.

The certificate request is sent as `pki::RequestCertificate` cluster message to the parent node.

If the parent node is not the signing master, it stores the request in `/var/lib/icinga2/certificate-requests` and forwards the cluster message to its parent node.

Once the message arrives on the signing master, it first verifies that the sent certificate request is valid. This is to prevent unwanted errors or modified

requests from the "proxy" node.

After verification, the signing master checks if the request contains a valid signing ticket. It hashes the certificate's common name and compares the value to the received ticket number.

If the ticket is valid, the certificate request is immediately signed with CA key. The request is sent back to the client inside a `pki::UpdateCertificate` cluster message.

If the child node was not the certificate request origin, it only updates the cached request for the child node and send another cluster message down to its child node (e.g. from a satellite to a client).

If no ticket was specified, the signing master waits until the `ca sign` CLI command manually signed the certificate.

> **Note**
>
> Push notifications for manual request signing is not yet implemented (TODO).

Once the child node reconnects it synchronizes all signed certificate requests. This takes some minutes and requires all nodes to reconnect to each other.

19.2.2.1 CSR Signing: Clients without parent connection

There is an additional scenario: The setup on a child node does not necessarily need a connection to the parent node.

This mode leaves the node in a semi-configured state. You need to manually copy the master's public CA key into `/var/lib/icinga2/certs/ca.crt` on the client before starting Icinga 2.

The parent node needs to actively connect to the child node. Once this connections succeeds, the child node will actively request a signed certificate.

The update procedure works the same way as above.

19.2.3 High Availability

High availability is automatically enabled between two nodes in the same cluster zone.

This requires the same configuration and enabled features on both nodes.

HA zone members trust each other and share event updates as cluster messages. This includes for example check results, next check timestamp updates, acknowledgements or notifications.

This ensures that both nodes are synchronized. If one node goes away, the remaining node takes over and continues as normal.

Cluster nodes automatically determine the authority for configuration objects. This results in activated but paused objects. You can verify that by querying the `paused` attribute for all objects via REST API or debug console.

Nodes inside a HA zone calculate the object authority independent from each other.

The number of endpoints in a zone is defined through the configuration. This number is used inside a local modulo calculation to determine whether the node feels responsible for this object or not.

This object authority is important for selected features explained below.

Since features are configuration objects too, you must ensure that all nodes inside the HA zone share the same enabled features. If configured otherwise, one might have a checker feature on the left node, nothing on the right node. This leads to late check results because one half is not executed by the right node which holds half of the object authorities.

19.2.4 High Availability: Checker

The `checker` feature only executes checks for `Checkable` objects (Host, Service) where it is authoritative.

That way each node only executes checks for a segment of the overall configuration objects.

The cluster message routing ensures that all check results are synchronized to nodes which are not authoritative for this configuration object.

19.2.5 High Availability: Notifications

The `notification` feature only sends notifications for `Notification` objects where it is authoritative.

That way each node only executes notifications for a segment of all notification objects.

Notified users and other event details are synchronized throughout the cluster. This is required if for example the DB IDO feature is active on the other node.

19.2.6 High Availability: DB IDO

If you don't have HA enabled for the IDO feature, both nodes will write their status and historical data to their own separate database backends.

In order to avoid data separation and a split view (each node would require its own Icinga Web 2 installation on top), the high availability option was added to the DB IDO feature. This is enabled by default with the `enable_ha` setting.

This requires a central database backend. Best practice is to use a MySQL cluster with a virtual IP.

Both Icinga 2 nodes require the connection and credential details configured in their DB IDO feature.

During startup Icinga 2 calculates whether the feature configuration object is authoritative on this node or not. The order is an alpha-numeric comparison, e.g. if you have `master1` and `master2`, Icinga 2 will enable the DB IDO feature on `master2` by default.

If the connection between endpoints drops, the object authority is re-calculated.

In order to prevent data duplication in a split-brain scenario where both nodes would write into the same database, there is another safety mechanism in place.

The split-brain decision which node will write to the database is calculated from a quorum inside the `programstatus` table. Each node verifies whether the `endpoint_name` column is not itself on database connect. In addition to that the DB IDO feature compares the `last_update_time` column against the current timestamp plus the configured `failover_timeout` offset.

That way only one active DB IDO feature writes to the database, even if they are not currently connected in a cluster zone. This prevents data duplication in historical tables.

19.2.7 Health Checks

19.2.7.1 cluster-zone

This built-in check provides the possibility to check for connectivity between zones.

If you for example need to know whether the `master` zone is connected and processing messages with the child zone called `satellite` in this example, you can configure the cluster-zone check as new service on all `master` zone hosts.

```
vim /etc/zones.d/master/host1.conf

object Service "cluster-zone-satellite" {
  check_command = "cluster-zone"
  host_name = "host1"

  vars.cluster_zone = "satellite"
}
```

The check itself changes to NOT-OK if one or more child endpoints in the child zone are not connected to parent zone endpoints.

In addition to the overall connectivity check, the log lag is calculated based on the to-be-sent replay log. Each instance stores that for its configured endpoint objects.

This health check iterates over the target zone (`cluster_zone`) and their endpoints.

The log lag is greater than zero if

- the replay log synchronization is in progress and not yet finished or
- the endpoint is not connected, and no replay log sync happened (obviously).

The final log lag value is the worst value detected. If satellite1 has a log lag of `1.5` and satellite2 only has `0.5`, the computed value will be `1.5`..

You can control the check state by using optional warning and critical thresholds for the log lag value.

If this service exists multiple times, e.g. for each master host object, the log lag may differ based on the execution time. This happens for example on restart of an instance when the log replay is in progress and a health check is executed at different times. If the endpoint is not connected, both master instances may have saved a different log replay position from the last synchronisation.

The lag value is returned as performance metric key `slave_lag`.

Icinga 2 v2.9+ adds more performance metrics for these values:

- `last_messages_sent` and `last_messages_received` as UNIX timestamp
- `sum_messages_sent_per_second` and `sum_messages_received_per_second`
- `sum_bytes_sent_per_second` and `sum_bytes_received_per_second`

20 Script Debugger

You can run the Icinga 2 daemon with the `-X` (`--script-debugger`) parameter to enable the script debugger:

```
# icinga2 daemon -X
```

When an exception occurs or the debugger keyword is encountered in a user script, Icinga 2 launches a console that allows the user to debug the script.

You can also attach the script debugger to the configuration validation:

```
# icinga2 daemon -C -X
```

Here is a list of common errors which can be diagnosed with the script debugger:

- Configuration errors e.g. apply rules
- Errors in user-defined functions

20.1 Debugging Configuration Errors

The following example illustrates the problem of a service apply rule which expects a dictionary value for `config`, but the host custom attribute only provides a string value:

```
object Host "script-debugger-host" {
  check_command = "icinga"

 vars.http_vhosts["example.org"] = "192.168.1.100" // a string value
}

apply Service for (http_vhost => config in host.vars.http_vhosts) {
  import "generic-service"

  vars += config // expects a dictionary

  check_command = "http"
}
```

The error message on config validation will warn about the wrong value type, but does not provide any context which objects are affected.

Enable the script debugger and run the config validation:

```
# icinga2 daemon -C -X

Breakpoint encountered in /etc/icinga2/conf.d/services.conf: 59:67-65:1
Exception: Error: Error while evaluating expression: Cannot convert value of type 'String' to an
Location:
/etc/icinga2/conf.d/services.conf(62):   check_command = "http"
/etc/icinga2/conf.d/services.conf(63):
/etc/icinga2/conf.d/services.conf(64):   vars += config
                                         ^^^^^^^^^^^^^^

/etc/icinga2/conf.d/services.conf(65): }
/etc/icinga2/conf.d/services.conf(66):
You can inspect expressions (such as variables) by entering them at the prompt.
To leave the debugger and continue the program use "$continue".
<1> =>
```

You can print the variables `vars` and `config` to get an idea about their values:

```
<1> => vars
null
<2> => config
```

```
"192.168.1.100"
<3> =>
```

The `vars` attribute has to be a dictionary. Trying to set this attribute to a string caused the error in our configuration example.

In order to determine the name of the host where the value of the `config` variable came from you can inspect attributes of the service object:

```
<3> => host_name
"script-debugger-host-01"
<4> => name
"http"
```

Additionally you can view the service object attributes by printing the value of `this`.

20.2 Using Breakpoints

In order to halt execution in a script you can use the `debugger` keyword:

```
object Host "script-debugger-host-02" {
  check_command = "dummy"
  check_interval = 5s

  vars.dummy_text = {{
    var text = "Hello from " + macro("$name$")
    debugger
    return text
  }}
}
```

Icinga 2 will spawn a debugger console every time the function is executed:

```
# icinga2 daemon -X
...
Breakpoint encountered in /etc/icinga2/tests/script-debugger.conf: 7:5-7:12
You can inspect expressions (such as variables) by entering them at the prompt.
To leave the debugger and continue the program use "$continue".
<1> => text
"Hello from script-debugger-host-02"
<2> => $continue
```

21 Develop Icinga 2

This chapter provides hints on Icinga 2 development especially for debugging purposes.

Note

If you are planning to build your own development environment, please consult the `INSTALL.md` file from the source tree.

21.1 Debug Requirements

Make sure that the debug symbols are available for Icinga 2. The Icinga 2 packages provide a debug package which must be installed separately for all involved binaries, like `icinga2-bin` or `icinga2-ido-mysql`.

Debian/Ubuntu:

```
# apt-get install icinga2-dbg
```

RHEL/CentOS:

```
# yum install icinga2-debuginfo
```

SLES/openSUSE:

```
# zypper install icinga2-bin-debuginfo icinga2-ido-mysql-debuginfo
```

Furthermore, you may also have to install debug symbols for Boost and your C library.

If you're building your own binaries, you should use the `-DCMAKE_BUILD_TYPE=Debug` cmake build flag for debug builds.

21.2 GDB

Install gdb:

Debian/Ubuntu:

```
# apt-get install gdb
```

RHEL/CentOS/Fedora:

```
# yum install gdb
```

SLES/openSUSE:

```
# zypper install gdb
```

Install the `boost`, `python` and `icinga2` pretty printers. Absolute paths are required, so please make sure to update the installation paths accordingly (`pwd`).

```
$ mkdir -p ~/.gdb_printers && cd ~/.gdb_printers
```

Boost Pretty Printers compatible with Python 3:

```
$ git clone https://github.com/mateidavid/Boost-Pretty-Printer.git && cd Boost-Pretty-Printer
$ git checkout python-3
$ pwd
/home/michi/.gdb_printers/Boost-Pretty-Printer
```

Python Pretty Printers:

```
$ cd ~/.gdb_printers
$ svn co svn://gcc.gnu.org/svn/gcc/trunk/libstdc++-v3/python
```

Icinga 2 Pretty Printers:

```
$ mkdir -p ~/.gdb_printers/icinga2 && cd ~/.gdb_printers/icinga2
$ wget https://raw.githubusercontent.com/Icinga/icinga2/master/tools/debug/gdb/icingadbg.py
```

Now you'll need to modify/setup your `~/.gdbinit` configuration file. You can download the one from Icinga 2 and modify all paths.

Example on Fedora 22:

```
$ wget https://raw.githubusercontent.com/Icinga/icinga2/master/tools/debug/gdb/gdbinit -O ~/.
$ vim ~/.gdbinit

set print pretty on

python
import sys
sys.path.insert(0, '/home/michi/.gdb_printers/icinga2')
from icingadbg import register_icinga_printers
register_icinga_printers()
end

python
import sys
sys.path.insert(0, '/home/michi/.gdb_printers/python')
from libstdcxx.v6.printers import register_libstdcxx_printers
try:
    register_libstdcxx_printers(None)
except:
    pass
end

python
import sys
sys.path.insert(0, '/home/michi/.gdb_printers/Boost-Pretty-Printer')
import boost_print
boost_print.register_printers()
end
```

If you are getting the following error when running gdb, the `libstdcxx` printers

are already preloaded in your environment and you can remove the duplicate import in your `~/.gdbinit` file.

```
RuntimeError: pretty-printer already registered: libstdc++-v6
```

21.2.1 GDB Run

Call GDB with the binary (`/usr/sbin/icinga2` is a wrapper script calling `/usr/lib64/icinga2/sbin/icinga2` since 2.4) and all arguments and run it in foreground.

```
# gdb --args /usr/lib64/icinga2/sbin/icinga2 daemon -x debug --no-stack-rlimit
```

The exact path to the Icinga 2 binary differs on each distribution. On Ubuntu it is installed into `/usr/lib/x86_64-linux-gnu/icinga2/sbin/icinga2` on 64-bit systems for example.

Note

If gdb tells you it's missing debug symbols, quit gdb and install them:
`Missing separate debuginfos, use: debuginfo-install ...`

Run the application.

```
(gdb) r
```

Kill the running application.

```
(gdb) k
```

Continue after breakpoint.

```
(gdb) c
```

21.2.2 GDB Core Dump

Either attach to the running process using `gdb -p PID` or start a new gdb run.

```
(gdb) r
(gdb) generate-core-file
```

21.2.3 GDB Backtrace

If Icinga 2 aborted its operation abnormally, generate a backtrace.

```
(gdb) bt
(gdb) thread apply all bt full
```

If gdb stops at a SIGPIPE signal please disable the signal before running Icinga 2.

```
(gdb) handle SIGPIPE nostop noprint pass
(gdb) r
```

If you create a bug report, make sure to attach as much detail as possible.

21.2.4 GDB Backtrace from Running Process

If Icinga 2 is still running, generate a full backtrace from the running process and store it into a new file (e.g. for debugging dead locks):

```
# gdb -p $(pidof icinga2) -batch -ex "thread apply all bt full" -ex "detach" -ex "q" > gdb_bt.log
```

21.2.5 GDB Backtrace Stepping

Identifying the problem may require stepping into the backtrace, analysing the current scope, attributes, and possible unmet requirements. `p` prints the value of the selected variable or function call result.

```
(gdb) up
(gdb) down
(gdb) p checkable
(gdb) p checkable.px->m_Name
```

21.2.6 GDB Breakpoints

To set a breakpoint to a specific function call, or file specific line.

```
(gdb) b checkable.cpp:125
(gdb) b icinga::Checkable::SetEnablePerfdata
```

GDB will ask about loading the required symbols later, select `yes` instead of `no`.

Then run Icinga 2 until it reaches the first breakpoint. Continue with `c` afterwards.

```
(gdb) run
(gdb) c
```

If you want to delete all breakpoints, use `d` and select `yes`.

```
(gdb) d
```

Tip

When debugging exceptions, set your breakpoint like this: `b __cxa_throw`.

Breakpoint Example:

```
(gdb) b __cxa_throw
(gdb) r
(gdb) up
....
(gdb) up
#11 0x00007ffff7cbf9ff in icinga::Utility::GlobRecursive(icinga::String const&, icinga::Strin
    at /home/michi/coding/icinga/icinga2/lib/base/utility.cpp:609
609         callback(cpath);
(gdb) l
604
605 #endif /* _WIN32 */
606
607     std::sort(files.begin(), files.end());
608     BOOST_FOREACH(const String& cpath, files) {
609         callback(cpath);
610     }
611
612     std::sort(dirs.begin(), dirs.end());
613     BOOST_FOREACH(const String& cpath, dirs) {
(gdb) p files
$3 = std::vector of length 11, capacity 16 = {{static NPos = 18446744073709551615, m_Data = "/etc
   m_Data = "/etc/icinga2/conf.d/commands.conf"}, {static NPos = 18446744073709551615, m_Data =
   m_Data = "/etc/icinga2/conf.d/groups.conf"}, {static NPos = 18446744073709551615, m_Data = "
   m_Data = "/etc/icinga2/conf.d/satellite.conf"}, {static NPos = 18446744073709551615, m_Data
   m_Data = "/etc/icinga2/conf.d/templates.conf"}, {static NPos = 18446744073709551615, m_Data
   m_Data = "/etc/icinga2/conf.d/timeperiods.conf"}, {static NPos = 18446744073709551615, m_Da
```

21.3 Core Dump

When the Icinga 2 daemon crashes with a `SIGSEGV` signal a core dump file should be written. This will help developers to analyze and fix the problem.

21.3.1 Core Dump File Size Limit

This requires setting the core dump file size to `unlimited`.

Example for Systemd:

```
vim /usr/lib/systemd/system/icinga2.service

[Service]
...
LimitCORE=infinity

systemctl daemon-reload
```

```
systemctl restart icinga2
```

Example for init script:

```
vim /etc/init.d/icinga2
...
ulimit -c unlimited

service icinga2 restart
```

Verify that the Icinga 2 process core file size limit is set to `unlimited`.

```
cat /proc/`pidof icinga2`/limits
...
Max core file size      unlimited          unlimited          bytes
```

21.3.2 Core Dump Kernel Format

The Icinga 2 daemon runs with the SUID bit set. Therefore you need to explicitly enable core dumps for SUID on Linux.

```
sysctl -w fs.suid_dumpable=1
```

Adjust the coredump kernel format and file location on Linux:

```
sysctl -w kernel.core_pattern=/var/lib/cores/core.%e.%p

install -m 1777 -d /var/lib/cores
```

MacOS:

```
sysctl -w kern.corefile=/cores/core.%P

chmod 777 /cores
```

21.3.3 Core Dump Analysis

Once Icinga 2 crashes again a new coredump file will be written. Please attach this file to your bug report in addition to the general details.

Simple test case for a `SIGSEGV` simulation with `sleep`:

```
ulimit -c unlimited
sleep 1800&
[1] <PID>
kill -SEGV <PID>
gdb `which sleep` /var/lib/cores/core.sleep.<PID>
(gdb) bt
rm /var/lib/cores/core.sleep.*
```

Analyzing Icinga 2:

```
gdb /usr/lib64/icinga2/sbin/icinga2 core.icinga2.<PID>
(gdb) bt
```

22 SELinux

22.1 Introduction

SELinux is a mandatory access control (MAC) system on Linux which adds a fine-grained permission system for access to all system resources such as files, devices, networks and inter-process communication.

The most important questions are answered briefly in the FAQ of the SELinux Project. For more details on SELinux and how to actually use and administrate it on your system have a look at Red Hat Enterprise Linux 7 - SELinux User's and Administrator's Guide. For a simplified (and funny) introduction download the SELinux Coloring Book.

This documentation will use a format similar to the SELinux User's and Administrator's Guide.

22.1.1 Policy

Icinga 2 provides its own SELinux policy. Development target is a policy package for Red Hat Enterprise Linux 7 and derivatives running the targeted policy which confines Icinga 2 with all features and all checks executed. All other distributions will require some tweaks.

22.1.2 Installation

There are two ways of installing the SELinux Policy for Icinga 2 on Enterprise Linux 7. The preferred way is to install the package. The other option involves installing the SELinux policy manually which might be necessary if you need some fixes which haven't made their way into a release yet.

If the system runs in enforcing mode and you encounter problems you can set Icinga 2's domain to permissive mode.

```
# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
```

```
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

You can change the configured mode by editing `/etc/selinux/config` and the current mode by executing `setenforce 0`.

22.1.2.1 Package installation

Simply add the `icinga2-selinux` package to your installation.

```
# yum install icinga2-selinux
```

Ensure that the `icinga2` process is running in its own `icinga2_t` domain after installing the policy package:

```
# systemctl restart icinga2.service
# ps -eZ | grep icinga2
system_u:system_r:icinga2_t:s0   2825 ?        00:00:00 icinga2
```

22.1.2.2 Manual installation

This section describes the installation to support development and testing. It assumes that Icinga 2 is already installed from packages and running on the system.

As a prerequisite install the `git`, `selinux-policy-devel` and `audit` packages. Enable and start the audit daemon afterwards:

```
# yum install git selinux-policy-devel audit
# systemctl enable auditd.service
# systemctl start auditd.service
```

After that clone the icinga2 git repository:

```
# git clone https://github.com/icinga/icinga2
```

To create and install the policy package run the installation script which also labels the resources. (The script assumes Icinga 2 was started once after system startup, the labeling of the port will only happen once and fail later on.)

```
# cd tools/selinux/
# ./icinga.sh
```

After that restart Icinga 2 and verify it running in its own domain `icinga2_t`.

```
# systemctl restart icinga2.service
# ps -eZ | grep icinga2
system_u:system_r:icinga2_t:s0   2825 ?        00:00:00 icinga2
```

22.1.3 General

When the SELinux policy package for Icinga 2 is installed, the Icinga 2 daemon (icinga2) runs in its own domain `icinga2_t` and is separated from other confined services.

Files have to be labeled correctly in order for Icinga 2 to be able to access them. For example the Icinga 2 log files have to have the `icinga2_log_t` label. Also the API port is labeled with `icinga_port_t`. Furthermore Icinga 2 can open high ports and UNIX sockets to connect to databases and features like Graphite. It executes the Nagios plugins and transitions to their context if those are labeled for example `nagios_services_plugin_exec_t` or `nagios_system_plugin_exec_t`.

Additionally the Apache web server is allowed to connect to Icinga 2's command pipe in order to allow web interfaces to send commands to icinga2. This will perhaps change later on while investigating Icinga Web 2 for SELinux!

22.1.4 Types

The command pipe is labeled `icinga2_command_t` and other services can request access to it by using the interface `icinga2_send_commands`.

The nagios plugins use their own contexts and icinga2 will transition to it. This means plugins have to be labeled correctly for their required permissions. The plugins installed from package should have set their permissions by the corresponding policy module and you can restore them using `restorecon -R -v /usr/lib64/nagios/plugins/`. To label your own plugins use `chcon -t type /path/to/plugin`, for the type have a look at table below.

Type	Domain	Use case	Provided by policy package
nagios_admin_plugin_exec_t	nagios_admin_plugin_t	Plugins which require require read access on all file attributes	nagios
nagios_checkdisk_plugin_exec_t	nagios_checkdisk_plugin_t	Plugins which require read access to all filesystem attributes	nagios
nagios_mail_plugin_exec_t	nagios_mail_plugin_t	Plugins which access the local mail service	nagios
nagios_services_plugin_exec_t	nagios_services_plugin_t	Plugins monitoring network services	nagios
nagios_system_plugin_exec_t	nagios_system_plugin_t	Plugins checking local system state	nagios

Type	Domain	Use case	Provided by policy package
nagios_unconfined_plugin_t	nagios_unconfined_plugin_domain	Plugins running without confinement	nagios
nagios_eventhandler_plugin_t	nagios_eventhandler_plugin_domain	Eventhandler (actually running unconfined)	nagios
nagios_openshift_plugin_t	nagios_openshift_plugin_domain	Plugins monitoring openshift	nagios
nagios_notification_plugin_t	nagios_notification_plugin_domain	Notification commands	icinga (will be moved later)

If one of those plugin domains causes problems you can set it to permissive by executing `semanage permissive -a domain`.

The policy provides a role `icinga2adm_r` for confining an user which enables an administrative user managing only Icinga 2 on the system. This user will also execute the plugins in their domain instead of the users one, so you can verify their execution with the same restrictions like they have when executed by icinga2.

22.1.5 Booleans

SELinux is based on the least level of access required for a service to run. Using booleans you can grant more access in a defined way. The Icinga 2 policy package provides the following booleans.

icinga2_can_connect_all

Having this boolean enabled allows icinga2 to connect to all ports. This can be necessary if you use features which connect to unconfined services, for example the influxdb writer.

httpd_can_write_icinga2_command

To allow httpd to write to the command pipe of icinga2 this boolean has to be enabled. This is enabled by default, if not needed you can disable it for more security.

httpd_can_connect_icinga2_api

Enabling this boolean allows httpd to connect to the API of icinga2 (Ports labeled `icinga2_port_t`). This is enabled by default, if not needed you can disable it for more security.

22.1.6 Configuration Examples

22.1.6.1 Run the icinga2 service permissive

If problems occur while running the system in enforcing mode and those problems are only caused by the policy of the icinga2 domain, you can set this domain to permissive instead of the complete system. This can be done by executing `semanage permissive -a icinga2_t`.

Make sure to report the bugs in the policy afterwards.

22.1.6.2 Confining a plugin

Download and install a plugin, for example check_mysql_health.

```
# wget https://labs.consol.de/download/shinken-nagios-plugins/check_mysql_health-2.1.9.2.tar
# tar xvzf check_mysql_health-2.1.9.2.tar.gz
# cd check_mysql_health-2.1.9.2/
# ./configure --libexecdir /usr/lib64/nagios/plugins
# make
# make install
```

It is labeled `nagios_unconfined_plugins_exec_t` by default, so it runs without restrictions.

```
# ls -lZ /usr/lib64/nagios/plugins/check_mysql_health
-rwxr-xr-x. root root system_u:object_r:nagios_unconfined_plugin_exec_t:s0 /usr/lib64/nagios/
```

In this case the plugin is monitoring a service, so it should be labeled `nagios_services_plugin_exec_t` to restrict its permissions.

```
# chcon -t nagios_services_plugin_exec_t /usr/lib64/nagios/plugins/check_mysql_health
# ls -lZ /usr/lib64/nagios/plugins/check_mysql_health
-rwxr-xr-x. root root system_u:object_r:nagios_services_plugin_exec_t:s0 /usr/lib64/nagios/pl
```

The plugin still runs fine but if someone changes the script to do weird stuff it will fail to do so.

22.1.6.3 Allow icinga to connect to all ports.

You are running graphite on a different port than `2003` and want `icinga2` to connect to it.

Change the port value for the graphite feature according to your graphite installation before enabling it.

```
# cat /etc/icinga2/features-enabled/graphite.conf
/**
 * The GraphiteWriter type writes check result metrics and
 * performance data to a graphite tcp socket.
```

```
 */

library "perfdata"

object GraphiteWriter "graphite" {
  //host = "127.0.0.1"
  //port = 2003
  port = 2004
}
# icinga2 feature enable graphite
```

Before you restart the icinga2 service allow it to connect to all ports by enabling the boolean ´icinga2_can_connect_all‘ (now and permanent).

```
# setsebool icinga2_can_connect_all true
# setsebool -P icinga2_can_connect_all true
```

If you restart the daemon now it will successfully connect to graphite.

22.1.6.4 Confining a user

If you want to have an administrative account capable of only managing icinga2 and not the complete system, you can restrict the privileges by confining this user. This is completly optional!

Start by adding the Icinga 2 administrator role `icinga2adm_r` to the administrative SELinux user `staff_u`.

```
# semanage user -m -R "staff_r sysadm_r system_r unconfined_r icinga2adm_r" staff_u
```

Confine your user login and create a sudo rule.

```
# semanage login -a dirk -s staff_u
# echo "dirk ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/dirk
```

Login to the system using ssh and verify your id.

```
$ id -Z
staff_u:staff_r:staff_t:s0-s0:c0.c1023
```

Try to execute some commands as root using sudo.

```
$ sudo id -Z
staff_u:staff_r:staff_t:s0-s0:c0.c1023
$ sudo vi /etc/icinga2/icinga2.conf
"/etc/icinga2/icinga2.conf" [Permission Denied]
$ sudo cat /var/log/icinga2/icinga2.log
cat: /var/log/icinga2/icinga2.log: Keine Berechtigung
$ sudo systemctl reload icinga2.service
Failed to get D-Bus connection: No connection to service manager.
```

Those commands fail because you only switch to root but do not change your SELinux role. Try again but tell sudo also to switch the SELinux role and type.

```
$ sudo -r icinga2adm_r -t icinga2adm_t id -Z
staff_u:icinga2adm_r:icinga2adm_t:s0-s0:c0.c1023
$ sudo -r icinga2adm_r -t icinga2adm_t vi /etc/icinga2/icinga2.conf
"/etc/icinga2/icinga2.conf"
$ sudo -r icinga2adm_r -t icinga2adm_t cat /var/log/icinga2/icinga2.log
[2015-03-26 20:48:14 +0000] information/DynamicObject: Dumping program state to file '/var/lib/
$ sudo -r icinga2adm_r -t icinga2adm_t systemctl reload icinga2.service
```

Now the commands will work, but you have always to remember to add the arguments, so change the sudo rule to set it by default.

```
# echo "dirk ALL=(ALL) ROLE=icinga2adm_r TYPE=icinga2adm_t NOPASSWD: ALL" > /etc/sudoers.d/dirk
```

Now try the commands again without providing the role and type and they will work, but if you try to read apache logs or restart apache for example it will still fail.

```
$ sudo cat /var/log/httpd/error_log
/bin/cat: /var/log/httpd/error_log: Keine Berechtigung
$ sudo systemctl reload httpd.service
Failed to issue method call: Access denied
```

22.2 Bugreports

If you experience any problems while running in enforcing mode try to reproduce it in permissive mode. If the problem persists it is not related to SELinux because in permissive mode SELinux will not deny anything.

After some feedback Icinga 2 is now running in a enforced domain, but still adds also some rules for other necessary services so no problems should occure at all. But you can help to enhance the policy by testing Icinga 2 running confined by SELinux.

Please add the following information to bug reports:

- Versions, configuration snippets, etc.
- Output of `semodule -l | grep -e icinga2 -e nagios -e apache`
- Output of `ps -eZ | grep icinga2`
- Output of `semanage port -l | grep icinga2`
- Output of `audit2allow -li /var/log/audit/audit.log`

If access to a file is blocked and you can tell which one please provided the output of `ls -lZ /path/to/file` (and perhaps the directory above).

If asked for full audit.log add `-w /etc/shadow -p w` to `/etc/audit/rules.d/audit.rules`, restart the audit daemon, reproduce the problem and add `/var/log/audit/audit.log`

to the bug report. With the added audit rule it will include the path of files access was denied to.

If asked to provide full audit log with dontaudit rules disabled executed `semodule -DB` before reproducing the problem. After that enable the rules again to prevent auditd spamming your logfile by executing `semodule -B`.

23 Migration from Icinga 1.x

23.1 Configuration Migration

The Icinga 2 configuration format introduces plenty of behavioural changes. In order to ease migration from Icinga 1.x, this section provides hints and tips on your migration requirements.

23.1.1 Manual Config Migration

For a long-term migration of your configuration you should consider re-creating your configuration based on the proposed Icinga 2 configuration paradigm.

Please read the next chapter to find out more about the differences between 1.x and 2.

23.1.2 Manual Config Migration Hints

These hints should provide you with enough details for manually migrating your configuration, or to adapt your configuration export tool to dump Icinga 2 configuration instead of Icinga 1.x configuration.

The examples are taken from Icinga 1.x test and production environments and converted straight into a possible Icinga 2 format. If you found a different strategy, please let us know!

If you require in-depth explanations, please check the next chapter.

23.1.2.1 Manual Config Migration Hints for Intervals

By default all intervals without any duration literal are interpreted as seconds. Therefore all existing Icinga 1.x `*_interval` attributes require an additional `m` duration literal.

Icinga 1.x:

```
define service {
  service_description             service1
  host_name                       localhost1
  check_command                   test_customvar
  use                             generic-service
  check_interval                  5
  retry_interval                  1
}
```

Icinga 2:

```
object Service "service1" {
  import "generic-service"
  host_name = "localhost1"
  check_command = "test_customvar"
  check_interval = 5m
  retry_interval = 1m
}
```

23.1.2.2 Manual Config Migration Hints for Services

If you have used the `host_name` attribute in Icinga 1.x with one or more host names this service belongs to, you can migrate this to the apply rules syntax.

Icinga 1.x:

```
define service {
  service_description             service1
  host_name                       localhost1,localhost2
  check_command                   test_check
  use                             generic-service
}
```

Icinga 2:

```
apply Service "service1" {
  import "generic-service"
  check_command = "test_check"

  assign where host.name in [ "localhost1", "localhost2" ]
}
```

In Icinga 1.x you would have organized your services with hostgroups using the `hostgroup_name` attribute like the following example:

```
define service {
  service_description             servicewithhostgroups
  hostgroup_name                  hostgroup1,hostgroup3
  check_command                   test_check
```

```
  use                             generic-service
}
```

Using Icinga 2 you can migrate this to the apply rules syntax:

```
apply Service "servicewithhostgroups" {
  import "generic-service"
  check_command = "test_check"

  assign where "hostgroup1" in host.groups
  assign where "hostgroup3" in host.groups
}
```

23.1.2.3 Manual Config Migration Hints for Group Members

The Icinga 1.x hostgroup `hg1` has two members `host1` and `host2`. The hostgroup `hg2` has `host3` as a member and includes all members of the `hg1` hostgroup.

```
define hostgroup {
  hostgroup_name                  hg1
  members                         host1,host2
}

define hostgroup {
  hostgroup_name                  hg2
  members                         host3
  hostgroup_members               hg1
}
```

This can be migrated to Icinga 2 and using group assign. The additional nested hostgroup `hg1` is included into `hg2` with the `groups` attribute.

```
object HostGroup "hg1" {
  assign where host.name in [ "host1", "host2" ]
}

object HostGroup "hg2" {
  groups = [ "hg1" ]
  assign where host.name == "host3"
}
```

These assign rules can be applied for all groups: `HostGroup`, `ServiceGroup` and `UserGroup` (requires renaming from `contactgroup`).

Tip

Define custom attributes and assign/ignore members based on these attribute pattern matches.

23.1.2.4 Manual Config Migration Hints for Check Command Arguments

Host and service check command arguments are separated by a `!` in Icinga 1.x. Their order is important and they are referenced as `$ARGn$` where `n` is the argument counter.

```
define command {
  command_name                    my-ping
  command_line                $USER1$/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5
}

define service {
  use                             generic-service
  host_name                       my-server
  service_description             my-ping
  check_command                 my-ping-check!100.0,20%!500.0,60%
}
```

While you could manually migrate this like (please note the new generic command arguments and default argument values!):

```
object CheckCommand "my-ping-check" {
  command = [
    PluginDir + "/check_ping", "-4"
  ]

  arguments = {
    "-H" = "$ping_address$"
    "-w" = "$ping_wrta$,$ping_wpl$%"
    "-c" = "$ping_crta$,$ping_cpl$%"
    "-p" = "$ping_packets$"
    "-t" = "$ping_timeout$"
  }

  vars.ping_address = "$address$"
  vars.ping_wrta = 100
  vars.ping_wpl = 5
  vars.ping_crta = 200
  vars.ping_cpl = 15
}

object Service "my-ping" {
  import "generic-service"
  host_name = "my-server"
  check_command = "my-ping-check"

  vars.ping_wrta = 100
```

```
  vars.ping_wpl = 20
  vars.ping_crta = 500
  vars.ping_cpl = 60
}
```

23.1.2.5 Manual Config Migration Hints for Runtime Macros

Runtime macros have been renamed. A detailed comparison table can be found here.

For example, accessing the service check output looks like the following in Icinga 1.x:

```
$SERVICEOUTPUT$
```

In Icinga 2 you will need to write:

```
$service.output$
```

Another example referencing the host's address attribute in Icinga 1.x:

```
$HOSTADDRESS$
```

In Icinga 2 you'd just use the following macro to access all `address` attributes (even overridden from the service objects):

```
$address$
```

23.1.2.6 Manual Config Migration Hints for Runtime Custom Attributes

Custom variables from Icinga 1.x are available as Icinga 2 custom attributes.

```
define command {
  command_name                    test_customvar
 command_line              echo "Host CV: $_HOSTCVTEST$ Service CV: $_SERVICECVTEST$\n"
}

define host {
  host_name                       localhost1
  check_command                   test_customvar
  use                             generic-host
  _CVTEST                         host cv value
}

define service {
  service_description             service1
  host_name                       localhost1
  check_command                   test_customvar
  use                             generic-service
```

```
  _CVTEST                         service cv value
}
```

Can be written as the following in Icinga 2:

```
object CheckCommand "test_customvar" {
 command = "echo "Host CV: $host.vars.CVTEST$ Service CV: $service.vars.CVTEST$\n""
}

object Host "localhost1" {
  import "generic-host"
  check_command = "test_customvar"
  vars.CVTEST = "host cv value"
}

object Service "service1" {
  host_name = "localhost1"
  check_command = "test_customvar"
  vars.CVTEST = "service cv value"
}
```

If you are just defining `$CVTEST$` in your command definition, its value depends on the execution scope – the host check command will fetch the host attribute value of `vars.CVTEST` while the service check command resolves its value to the service attribute attribute `vars.CVTEST`.

Note

Custom attributes in Icinga 2 are case-sensitive. `vars.CVTEST` is not the same as `vars.CvTest`.

23.1.2.7 Manual Config Migration Hints for Contacts (Users)

Contacts in Icinga 1.x act as users in Icinga 2, but do not have any notification commands specified. This migration part is explained in the next chapter.

```
define contact{
  contact_name                    testconfig-user
  use                             generic-user
  alias                           Icinga Test User
  service_notification_options    c,f,s,u
  email                           icinga@localhost
}
```

The `service_notification_options` can be mapped into generic `state` and `type` filters, if additional notification filtering is required. `alias` gets renamed to `display_name`.

```
object User "testconfig-user" {
```

```
  import "generic-user"
  display_name = "Icinga Test User"
  email = "icinga@localhost"
}
```

This user can be put into usergroups (former contactgroups) or referenced in newly migration notification objects.

23.1.2.8 Manual Config Migration Hints for Notifications

If you are migrating a host or service notification, you'll need to extract the following information from your existing Icinga 1.x configuration objects

- host/service attribute `contacts` and `contact_groups`
- host/service attribute `notification_options`
- host/service attribute `notification_period`
- host/service attribute `notification_interval`

The clean approach is to refactor your current contacts and their notification command methods into a generic strategy

- host or service has a notification type (for example mail)
- which contacts (users) are notified by mail?
- do the notification filters, periods, intervals still apply for them? (do a cleanup during migration)
- assign users and groups to these notifications
- Redesign the notifications into generic apply rules

The ugly workaround solution could look like this:

Extract all contacts from the remaining groups, and create a unique list. This is required for determining the host and service notification commands involved.

- contact attributes `host_notification_commands` and `service_notification_commands` (can be a comma separated list)
- get the command line for each notification command and store them for later
- create a new notification name and command name

Generate a new notification object based on these values. Import the generic template based on the type (`host` or `service`). Assign it to the host or service and set the newly generated notification command name as `command` attribute.

```
object Notification "<notificationname>" {
  import "mail-host-notification"
  host_name = "<thishostname>"
  command = "<notificationcommandname>"
```

Convert the `notification_options` attribute from Icinga 1.x to Icinga 2 `states` and `types`. Details here. Add the notification period.

```
  states = [ OK, Warning, Critical ]
  types = [ Recovery, Problem, Custom ]
  period = "24x7"
```

The current contact acts as `users` attribute.

```
  users = [ "<contactwithnotificationcommand>" ]
}
```

Do this in a loop for all notification commands (depending if host or service contact). Once done, dump the collected notification commands.

The result of this migration are lots of unnecessary notification objects and commands but it will unroll the Icinga 1.x logic into the revamped Icinga 2 notification object schema. If you are looking for code examples, try LConf.

23.1.2.9 Manual Config Migration Hints for Notification Filters

Icinga 1.x defines all notification filters in an attribute called `notification_options`. Using Icinga 2 you will have to split these values into the `states` and `types` attributes.

Note

`Recovery` type requires the `Ok` state. `Custom` and `Problem` should always be set as `type` filter.

Icinga 1.x option	Icinga 2 state	Icinga 2 type
o	OK (Up for hosts)	
w	Warning	Problem
c	Critical	Problem
u	Unknown	Problem
d	Down	Problem
s	.	DowntimeStart / DowntimeEnd / DowntimeRemoved
r	Ok	Recovery
f	.	FlappingStart / FlappingEnd
n	0 (none)	0 (none)
.	.	Custom

23.1.2.10 Manual Config Migration Hints for Escalations

Escalations in Icinga 1.x are a bit tricky. By default service escalations can be applied to hosts and hostgroups and require a defined service object.

The following example applies a service escalation to the service `dep_svc01` and all hosts in the `hg_svcdep2` hostgroup. The default `notification_interval` is set to `10` minutes notifying the `cg_admin` contact. After 20 minutes (`10*2`, notification_interval * first_notification) the notification is escalated to the

cg_ops contactgroup until 60 minutes (10*6) have passed.

```
define service {
  service_description             dep_svc01
  host_name                       dep_hostsvc01,dep_hostsvc03
  check_command                   test2
  use                             generic-service
  notification_interval           10
  contact_groups                  cg_admin
}

define hostgroup {
  hostgroup_name                  hg_svcdep2
  members                         dep_hostsvc03
}

# with hostgroup_name and service_description
define serviceescalation {
  hostgroup_name                  hg_svcdep2
  service_description             dep_svc01
  first_notification              2
  last_notification               6
  contact_groups                  cg_ops
}
```

In Icinga 2 the service and hostgroup definition will look quite the same. Save the notification_interval and contact_groups attribute for an additional notification.

```
apply Service "dep_svc01" {
  import "generic-service"

  check_command = "test2"

  assign where host.name == "dep_hostsvc01"
  assign where host.name == "dep_hostsvc03"
}

object HostGroup "hg_svcdep2" {
  assign where host.name == "dep_hostsvc03"
}

apply Notification "email" to Service {
  import "service-mail-notification"

  interval = 10m
  user_groups = [ "cg_admin" ]
```

```
  assign where service.name == "dep_svc01" && (host.name == "dep_hostsvc01" || host.name == "dep
}
```

Calculate the begin and end time for the newly created escalation notification:

- begin = first_notification * notification_interval = 2 * 10m = 20m
- end = last_notification * notification_interval = 6 * 10m = 60m = 1h

Assign the notification escalation to the service `dep_svc01` on all hosts in the hostgroup `hg_svcdep2`.

```
apply Notification "email-escalation" to Service {
  import "service-mail-notification"

  interval = 10m
  user_groups = [ "cg_ops" ]

  times = {
    begin = 20m
    end = 1h
  }

  assign where service.name == "dep_svc01" && "hg_svcdep2" in host.groups
}
```

The assign rule could be made more generic and the notification be applied to more than just this service belonging to hosts in the matched hostgroup.

Note

When the notification is escalated, Icinga 1.x suppresses notifications to the default contacts. In Icinga 2 an escalation is an additional notification with a defined begin and end time. The `email` notification will continue as normal.

23.1.2.11 Manual Config Migration Hints for Dependencies

There are some dependency examples already in the basics chapter. Dependencies in Icinga 1.x can be confusing in terms of which host/service is the parent and which host/service acts as the child.

While Icinga 1.x defines `notification_failure_criteria` and `execution_failure_criteria` as dependency filters, this behaviour has changed in Icinga 2. There is no 1:1 migration but generally speaking the state filter defined in the `execution_failure_criteria` defines the Icinga 2 `state` attribute. If the state filter matches, you can define whether to disable checks and notifications or not.

The following example describes service dependencies. If you migrate from Icinga 1.x, you will only want to use the classic `Host-to-Host` and `Service-to-Service` dependency relationships.

```
define service {
  service_description             dep_svc01
  hostgroup_name                  hg_svcdep1
  check_command                   test2
  use                             generic-service
}

define service {
  service_description             dep_svc02
  hostgroup_name                  hg_svcdep2
  check_command                   test2
  use                             generic-service
}

define hostgroup {
  hostgroup_name                  hg_svcdep2
  members                         host2
}

define host{
  use                             linux-server-template
  host_name                       host1
  address                         192.168.1.10
}

# with hostgroup_name and service_description
define servicedependency {
  host_name                       host1
  dependent_hostgroup_name        hg_svcdep2
  service_description             dep_svc01
  dependent_service_description   *
  execution_failure_criteria      u,c
  notification_failure_criteria   w,u,c
  inherits_parent                 1
}
```

Map the dependency attributes accordingly.

Icinga 1.x	Icinga 2
host_name	parent_host_name
dependent_host_name	child_host_name (used in assign/ignore)
dependent_hostgroup_name	all child hosts in group (used in assign/ignore)

Icinga 1.x	Icinga 2
service_description	parent_service_name
dependent_service_description	child_service_name (used in assign/ignore)

And migrate the host and services.

```
object Host "host1" {
  import "linux-server-template"
  address = "192.168.1.10"
}

object HostGroup "hg_svcdep2" {
  assign where host.name == "host2"
}

apply Service "dep_svc01" {
  import "generic-service"
  check_command = "test2"

  assign where "hp_svcdep1" in host.groups
}

apply Service "dep_svc02" {
  import "generic-service"
  check_command = "test2"

  assign where "hp_svcdep2" in host.groups
}
```

When it comes to the `execution_failure_criteria` and `notification_failure_criteria` attribute migration, you will need to map the most common values, in this example `u,c` (`Unknown` and `Critical` will cause the dependency to fail). Therefore the `Dependency` should be ok on Ok and Warning. `inherits_parents` is always enabled.

```
apply Dependency "all-svc-for-hg-hg_svcdep2-on-host1-dep_svc01" to Service {
  parent_host_name = "host1"
  parent_service_name = "dep_svc01"

  states = [ Ok, Warning ]
  disable_checks = true
  disable_notifications = true

  assign where "hg_svcdep2" in host.groups
}
```

Host dependencies are explained in the next chapter.

23.1.2.12 Manual Config Migration Hints for Host Parents

Host parents from Icinga 1.x are migrated into `Host-to-Host` dependencies in Icinga 2.

The following example defines the `vmware-master` host as parent host for the guest virtual machines `vmware-vm1` and `vmware-vm2`.

By default all hosts in the hostgroup `vmware` should get the parent assigned. This isn't really solvable with Icinga 1.x parents, but only with host dependencies.

```
define host{
  use                             linux-server-template
  host_name                       vmware-master
  hostgroups                      vmware
  address                         192.168.1.10
}

define host{
  use                             linux-server-template
  host_name                       vmware-vm1
  hostgroups                      vmware
  address                         192.168.27.1
  parents                         vmware-master
}

define host{
  use                             linux-server-template
  host_name                       vmware-vm2
  hostgroups                      vmware
  address                         192.168.28.1
  parents                         vmware-master
}
```

By default all hosts in the hostgroup `vmware` should get the parent assigned (but not the `vmware-master` host itself). This isn't really solvable with Icinga 1.x parents, but only with host dependencies as shown below:

```
define hostdependency {
  dependent_hostgroup_name        vmware
  dependent_host_name             !vmware-master
  host_name                       vmware-master
  inherits_parent                 1
  notification_failure_criteria   d,u
  execution_failure_criteria      d,u
```

```
  dependency_period               testconfig-24x7
}
```

When migrating to Icinga 2, the parents must be changed to a newly created host dependency.

Map the following attributes

Icinga 1.x	Icinga 2
host_name	parent_host_name
dependent_host_name	child_host_name (used in assign/ignore)
dependent_hostgroup_name	all child hosts in group (used in assign/ignore)

The Icinga 2 configuration looks like this:

```
object Host "vmware-master" {
  import "linux-server-template"
  groups += [ "vmware" ]
  address = "192.168.1.10"
  vars.is_vmware_master = true
}

object Host "vmware-vm1" {
  import "linux-server-template"
  groups += [ "vmware" ]
  address = "192.168.27.1"
}

object Host "vmware-vm2" {
  import "linux-server-template"
  groups += [ "vmware" ]
  address = "192.168.28.1"
}

apply Dependency "vmware-master" to Host {
  parent_host_name = "vmware-master"

  assign where "vmware" in host.groups
  ignore where host.vars.is_vmware_master
  ignore where host.name == "vmware-master"
}
```

For easier identification you could add the `vars.is_vmware_master` attribute to the `vmware-master` host and let the dependency ignore that instead of the hardcoded host name. That's different to the Icinga 1.x example and a best practice hint only.

Another way to express the same configuration would be something like:

```
object Host "vmware-master" {
  import "linux-server-template"
  groups += [ "vmware" ]
  address = "192.168.1.10"
}

object Host "vmware-vm1" {
  import "linux-server-template"
  groups += [ "vmware" ]
  address = "192.168.27.1"
  vars.parents = [ "vmware-master" ]
}

object Host "vmware-vm2" {
  import "linux-server-template"
  groups += [ "vmware" ]
  address = "192.168.28.1"
  vars.parents = [ "vmware-master" ]
}

apply Dependency "host-to-parent-" for (parent in host.vars.parents) to Host {
  parent_host_name = parent
}
```

This example allows finer grained host-to-host dependency, as well as multiple dependency support.

23.1.2.13 Manual Config Migration Hints for Distributed Setups

- Icinga 2 does not use active/passive instances calling OSCP commands and requiring the NSCA daemon for passing check results between instances.
- Icinga 2 does not support any 1.x NEB addons for check load distribution
- If your current setup consists of instances distributing the check load, you should consider building a load distribution setup with Icinga 2.
- If your current setup includes active/passive clustering with external tools like Pacemaker/DRBD, consider the High Availability setup.
- If you have build your own custom configuration deployment and check result collecting mechanism, you should re-design your setup and re-evaluate your requirements, and how they may be fulfilled using the Icinga 2 cluster capabilities.

23.2 Differences between Icinga 1.x and 2

23.2.1 Configuration Format

Icinga 1.x supports two configuration formats: key-value-based settings in the `icinga.cfg` configuration file and object-based in included files (`cfg_dir`, `cfg_file`). The path to the `icinga.cfg` configuration file must be passed to the Icinga daemon at startup.

icinga.cfg:

```
enable_notifications=1
```

objects.cfg:

```
define service {
   notifications_enabled    0
}
```

Icinga 2 supports objects and (global) variables, but does not make a difference between the main configuration file or any other included file.

icinga2.conf:

```
const EnableNotifications = true

object Service "test" {
    enable_notifications = false
}
```

23.2.1.1 Sample Configuration and ITL

While Icinga 1.x ships sample configuration and templates spread in various object files, Icinga 2 moves all templates into the Icinga Template Library (ITL) and includes them in the sample configuration.

Additional plugin check commands are shipped with Icinga 2 as well.

The ITL will be updated on every release and must not be edited by the user.

There are still generic templates available for your convenience which may or may not be re-used in your configuration. For instance, `generic-service` includes all required attributes except `check_command` for a service.

Sample configuration files are located in the `conf.d/` directory which is included in `icinga2.conf` by default.

> **Note**
>
> Add your own custom templates in the `conf.d/` directory as well, e.g. inside the templates.conf file.

23.2.2 Main Config File

In Icinga 1.x there are many global configuration settings available in `icinga.cfg`. Icinga 2 only uses a small set of global constants allowing you to specify certain different setting such as the `NodeName` in a cluster scenario.

Aside from that, the icinga2.conf should take care of including global constants, enabled features and the object configuration.

23.2.3 Include Files and Directories

In Icinga 1.x the `icinga.cfg` file contains `cfg_file` and `cfg_dir` directives. The `cfg_dir` directive recursively includes all files with a `.cfg` suffix in the given directory. Only absolute paths may be used. The `cfg_file` and `cfg_dir` directives can include the same file twice which leads to configuration errors in Icinga 1.x.

```
cfg_file=/etc/icinga/objects/commands.cfg
cfg_dir=/etc/icinga/objects
```

Icinga 2 supports wildcard includes and relative paths, e.g. for including `conf.d/*.conf` in the same directory.

```
include "conf.d/*.conf"
```

If you want to include files and directories recursively, you need to define a separate option and add the directory and an optional pattern.

```
include_recursive "conf.d"
```

A global search path for includes is available for advanced features like the Icinga Template Library (ITL) or additional monitoring plugins check command configuration.

```
include <itl>
include <plugins>
```

By convention the `.conf` suffix is used for Icinga 2 configuration files.

23.2.4 Resource File and Global Macros

Global macros such as for the plugin directory, usernames and passwords can be set in the `resource.cfg` configuration file in Icinga 1.x. By convention the `USER1` macro is used to define the directory for the plugins.

Icinga 2 uses global constants instead. In the default config these are set in the `constants.conf` configuration file:

```
/**
 * This file defines global constants which can be used in
 * the other configuration files. At a minimum the
 * PluginDir constant should be defined.
 */

const PluginDir = "/usr/lib/nagios/plugins"
```

Global macros can only be defined once. Trying to modify a global constant will result in an error.

23.2.5 Configuration Comments

In Icinga 1.x comments are made using a leading hash (`#`) or a semi-colon (`;`) for inline comments.

In Icinga 2 comments can either be encapsulated by `/*` and `*/` (allowing for multi-line comments) or starting with two slashes (`//`). A leading hash (`#`) could also be used.

23.2.6 Object Names

Object names must not contain an exclamation mark (`!`). Use the `display_name` attribute to specify user-friendly names which should be shown in UIs (supported by Icinga Web 2 for example).

Object names are not specified using attributes (e.g. `service_description` for services) like in Icinga 1.x but directly after their type definition.

```
define service {
    host_name  localhost
    service_description  ping4
}

object Service "ping4" {
  host_name = "localhost"
}
```

23.2.7 Templates

In Icinga 1.x templates are identified using the `register 0` setting. Icinga 2 uses the `template` identifier:

```
template Service "ping4-template" { }
```

Icinga 1.x objects inherit from templates using the `use` attribute. Icinga 2 uses the keyword `import` with template names in double quotes.

```
define service {
    service_description testservice
    use                 tmpl1,tmpl2,tmpl3
}

object Service "testservice" {
  import "tmpl1"
  import "tmpl2"
  import "tmpl3"
}
```

The last template overrides previously set values.

23.2.8 Object attributes

Icinga 1.x separates attribute and value pairs with whitespaces/tabs. Icinga 2 requires an equal sign (=) between them.

```
define service {
    check_interval  5
}

object Service "test" {
    check_interval = 5m
}
```

Please note that the default time value is seconds if no duration literal is given. `check_interval = 5` behaves the same as `check_interval = 5s`.

All strings require double quotes in Icinga 2. Therefore a double quote must be escaped by a backslash (e.g. in command line). If an attribute identifier starts with a number, it must be enclosed in double quotes as well.

23.2.8.1 Alias vs. Display Name

In Icinga 1.x a host can have an `alias` and a `display_name` attribute used for a more descriptive name. A service only can have a `display_name` attribute. The `alias` is used for group, timeperiod, etc. objects too. Icinga 2 only supports the `display_name` attribute which is also taken into account by Icinga web interfaces.

23.2.9 Custom Attributes

Icinga 2 allows you to define custom attributes in the `vars` dictionary. The `notes`, `notes_url`, `action_url`, `icon_image`, `icon_image_alt` attributes for host and service objects are still available in Icinga 2.

`2d_coords` and `statusmap_image` are not supported in Icinga 2.

23.2.9.1 Custom Variables

Icinga 1.x custom variable attributes must be prefixed using an underscore (_). In Icinga 2 these attributes must be added to the `vars` dictionary as custom attributes.

```
vars.dn = "cn=icinga2-dev-host,ou=icinga,ou=main,ou=IcingaConfig,ou=LConf,dc=icinga,dc=org"
vars.cv = "my custom cmdb description"
```

These custom attributes are also used as command parameters.

While Icinga 1.x only supports numbers and strings as custom attribute values, Icinga 2 extends that to arrays and (nested) dictionaries. For more details look here.

23.2.10 Host Service Relation

In Icinga 1.x a service object is associated with a host by defining the `host_name` attribute in the service definition. Alternate methods refer to `hostgroup_name` or behaviour changing regular expression.

The preferred way of associating hosts with services in Icinga 2 is by using the apply keyword.

Direct object relations between a service and a host still allow you to use the `host_name` Service object attribute.

23.2.11 Users

Contacts have been renamed to users (same for groups). A contact does not only provide (custom) attributes and notification commands used for notifications, but is also used for authorization checks in Icinga 1.x.

Icinga 2 changes that behavior and makes the user an attribute provider only. These attributes can be accessed using runtime macros inside notification command definitions.

In Icinga 2 notification commands are not directly associated with users. Instead the notification command is specified inside `Notification` objects next to user and user group relations.

The `StatusDataWriter`, `IdoMySqlConnection` and `LivestatusListener` types will provide the contact and contactgroups attributes for services for compatibility reasons. These values are calculated from all services, their notifications, and their users.

23.2.12 Macros

Various object attributes and runtime variables can be accessed as macros in commands in Icinga 1.x – Icinga 2 supports all required custom attributes.

23.2.12.1 Command Arguments

If you have previously used Icinga 1.x, you may already be familiar with user and argument definitions (e.g., `USER1` or `ARG1`). Unlike in Icinga 1.x the Icinga 2 custom attributes may have arbitrary names and arguments are no longer specified in the `check_command` setting.

In Icinga 1.x arguments are specified in the `check_command` attribute and are separated from the command name using an exclamation mark (`!`).

Please check the migration hints for a detailed migration example.

> **Note**
>
> The Icinga 1.x feature named `Command Expander` does not work with Icinga 2.

23.2.12.2 Environment Macros

The global configuration setting `enable_environment_macros` does not exist in Icinga 2.

Macros exported into the environment can be set using the `env` attribute in command objects.

23.2.12.3 Runtime Macros

Icinga 2 requires an object specific namespace when accessing configuration and stateful runtime macros. Custom attributes can be accessed directly.

If a runtime macro from Icinga 1.x is not listed here, it is not supported by Icinga 2.

Changes to user (contact) runtime macros

Icinga 1.x	Icinga 2
CONTACTNAME	user.name
CONTACTALIAS	user.display_name
CONTACTEMAIL	user.email
CONTACTPAGER	user.pager

`CONTACTADDRESS*` is not supported but can be accessed as `$user.vars.address1$`

if set.

Changes to service runtime macros

Icinga 1.x	Icinga 2
SERVICEDESC	service.name
SERVICEDISPLAYNAME	service.display_name
SERVICECHECKCOMMAND	service.check_command
SERVICESTATE	service.state
SERVICESTATEID	service.state_id
SERVICESTATETYPE	service.state_type
SERVICEATTEMPT	service.check_attempt
MAXSERVICEATTEMPT	service.max_check_attempts
LASTSERVICESTATE	service.last_state
LASTSERVICESTATEID	service.last_state_id
LASTSERVICESTATETYPE	service.last_state_type
LASTSERVICESTATECHANGE	service.last_state_change
SERVICEDOWNTIME	service.downtime_depth
SERVICEDURATIONSEC	service.duration_sec
SERVICELATENCY	service.latency
SERVICEEXECUTIONTIME	service.execution_time
SERVICEOUTPUT	service.output
SERVICEPERFDATA	service.perfdata
LASTSERVICECHECK	service.last_check
SERVICENOTES	service.notes
SERVICENOTESURL	service.notes_url
SERVICEACTIONURL	service.action_url

Changes to host runtime macros

Icinga 1.x	Icinga 2
HOSTNAME	host.name
HOSTADDRESS	host.address
HOSTADDRESS6	host.address6
HOSTDISPLAYNAME	host.display_name
HOSTALIAS	(use `host.display_name` instead)
HOSTCHECKCOMMAND	host.check_command
HOSTSTATE	host.state
HOSTSTATEID	host.state_id
HOSTSTATETYPE	host.state_type
HOSTATTEMPT	host.check_attempt
MAXHOSTATTEMPT	host.max_check_attempts
LASTHOSTSTATE	host.last_state
LASTHOSTSTATEID	host.last_state_id

Icinga 1.x	Icinga 2
LASTHOSTSTATETYPE	host.last_state_type
LASTHOSTSTATECHANGE	host.last_state_change
HOSTDOWNTIME	host.downtime_depth
HOSTDURATIONSEC	host.duration_sec
HOSTLATENCY	host.latency
HOSTEXECUTIONTIME	host.execution_time
HOSTOUTPUT	host.output
HOSTPERFDATA	host.perfdata
LASTHOSTCHECK	host.last_check
HOSTNOTES	host.notes
HOSTNOTESURL	host.notes_url
HOSTACTIONURL	host.action_url
TOTALSERVICES	host.num_services
TOTALSERVICESOK	host.num_services_ok
TOTALSERVICESWARNING	host.num_services_warning
TOTALSERVICESUNKNOWN	host.num_services_unknown
TOTALSERVICESCRITICAL	host.num_services_critical

Changes to command runtime macros

Icinga 1.x	Icinga 2
COMMANDNAME	command.name

Changes to notification runtime macros

Icinga 1.x	Icinga 2
NOTIFICATIONTYPE	notification.type
NOTIFICATIONAUTHOR	notification.author
NOTIFICATIONCOMMENT	notification.comment
NOTIFICATIONAUTHORNAME	(use `notification.author`)
NOTIFICATIONAUTHORALIAS	(use `notification.author`)

Changes to global runtime macros:

Icinga 1.x	Icinga 2
TIMET	icinga.timet
LONGDATETIME	icinga.long_date_time
SHORTDATETIME	icinga.short_date_time
DATE	icinga.date
TIME	icinga.time

Icinga 1.x	Icinga 2
PROCESSSTARTTIME	icinga.uptime

Changes to global statistic macros:

Icinga 1.x	Icinga 2
TOTALHOSTSUP	icinga.num_hosts_up
TOTALHOSTSDOWN	icinga.num_hosts_down
TOTALHOSTSUNREACHABLE	icinga.num_hosts_unreachable
TOTALHOSTSDOWNUNHANDLED	–
TOTALHOSTSUNREACHABLEUNHANDLED	
TOTALHOSTPROBLEMS	down
TOTALHOSTPROBLEMSUNHANDLED	down-(downtime+acknowledged)
TOTALSERVICESOK	icinga.num_services_ok
TOTALSERVICESWARNING	icinga.num_services_warning
TOTALSERVICESCRITICAL	icinga.num_services_critical
TOTALSERVICESUNKNOWN	icinga.num_services_unknown
TOTALSERVICESWARNINGUNHANDLED	–
TOTALSERVICESCRITICALUNHANDLED	–
TOTALSERVICESUNKNOWNUNHANDLED	
TOTALSERVICEPROBLEMS	ok+warning+critical+unknown
TOTALSERVICEPROBLEMSUNHANDLED	warning+critical+unknown-(downtime+acknowledged)

23.2.13 External Commands

`CHANGE_CUSTOM_CONTACT_VAR` was renamed to `CHANGE_CUSTOM_USER_VAR`.

The following external commands are not supported:

```
CHANGE_*MODATTR
CHANGE_CONTACT_HOST_NOTIFICATION_TIMEPERIOD
CHANGE_HOST_NOTIFICATION_TIMEPERIOD
CHANGE_SVC_NOTIFICATION_TIMEPERIOD
DEL_DOWNTIME_BY_HOSTGROUP_NAME
DEL_DOWNTIME_BY_START_TIME_COMMENT
DISABLE_ALL_NOTIFICATIONS_BEYOND_HOST
DISABLE_CONTACT_HOST_NOTIFICATIONS
DISABLE_CONTACT_SVC_NOTIFICATIONS
DISABLE_CONTACTGROUP_HOST_NOTIFICATIONS
DISABLE_CONTACTGROUP_SVC_NOTIFICATIONS
DISABLE_FAILURE_PREDICTION
```

```
DISABLE_HOST_AND_CHILD_NOTIFICATIONS
DISABLE_HOST_FRESHNESS_CHECKS
DISABLE_NOTIFICATIONS_EXPIRE_TIME
DISABLE_SERVICE_FRESHNESS_CHECKS
ENABLE_ALL_NOTIFICATIONS_BEYOND_HOST
ENABLE_CONTACT_HOST_NOTIFICATIONS
ENABLE_CONTACT_SVC_NOTIFICATIONS
ENABLE_CONTACTGROUP_HOST_NOTIFICATIONS
ENABLE_CONTACTGROUP_SVC_NOTIFICATIONS
ENABLE_FAILURE_PREDICTION
ENABLE_HOST_AND_CHILD_NOTIFICATIONS
ENABLE_HOST_FRESHNESS_CHECKS
ENABLE_SERVICE_FRESHNESS_CHECKS
READ_STATE_INFORMATION
SAVE_STATE_INFORMATION
SET_HOST_NOTIFICATION_NUMBER
SET_SVC_NOTIFICATION_NUMBER
START_ACCEPTING_PASSIVE_HOST_CHECKS
START_ACCEPTING_PASSIVE_SVC_CHECKS
START_OBSESSING_OVER_HOST
START_OBSESSING_OVER_HOST_CHECKS
START_OBSESSING_OVER_SVC
START_OBSESSING_OVER_SVC_CHECKS
STOP_ACCEPTING_PASSIVE_HOST_CHECKS
STOP_ACCEPTING_PASSIVE_SVC_CHECKS
STOP_OBSESSING_OVER_HOST
STOP_OBSESSING_OVER_HOST_CHECKS
STOP_OBSESSING_OVER_SVC
STOP_OBSESSING_OVER_SVC_CHECKS
```

23.2.14 Asynchronous Event Execution

Unlike Icinga 1.x, Icinga 2 does not block when it's waiting for a command being executed – whether if it's a check, a notification, an event handler, a performance data writing update, etc. That way you'll recognize low to zero (check) latencies with Icinga 2.

23.2.15 Checks

23.2.15.1 Check Output

Icinga 2 does not make a difference between `output` (first line) and `long_output` (remaining lines) like in Icinga 1.x. Performance Data is provided separately.

There is no output length restriction as known from Icinga 1.x using an 8KB static buffer.

The `StatusDataWriter`, `IdoMysqlConnection` and `LivestatusListener` types split the raw output into `output` (first line) and `long_output` (remaining lines) for compatibility reasons.

23.2.15.2 Initial State

Icinga 1.x uses the `max_service_check_spread` setting to specify a timerange where the initial state checks must have happened. Icinga 2 will use the `retry_interval` setting instead and `check_interval` divided by 5 if `retry_interval` is not defined.

23.2.16 Comments

Icinga 2 doesn't support non-persistent comments.

23.2.17 Commands

Unlike in Icinga 1.x there are three different command types in Icinga 2: `CheckCommand`, `NotificationCommand`, and `EventCommand`.

For example in Icinga 1.x it is possible to accidentally use a notification command as an event handler which might cause problems depending on which runtime macros are used in the notification command.

In Icinga 2 these command types are separated and will generate an error on configuration validation if used in the wrong context.

While Icinga 2 still supports the complete command line in command objects, it's recommended to use command arguments with optional and conditional command line parameters instead.

It's also possible to define default argument values for the command itself which can be overridden by the host or service then.

23.2.17.1 Command Timeouts

In Icinga 1.x there were two global options defining a host and service check timeout. This was essentially bad when there only was a couple of check plugins requiring some command timeouts to be extended.

Icinga 2 allows you to specify the command timeout directly on the command. So, if your VMVware check plugin takes 15 minutes, increase the timeout accordingly.

23.2.18 Groups

In Icinga 2 hosts, services, and users are added to groups using the `groups` attribute in the object. The old way of listing all group members in the group's `members` attribute is available through `assign where` and `ignore where` expressions by using group assign.

```
object Host "web-dev" {
  import "generic-host"
}

object HostGroup "dev-hosts" {
  display_name = "Dev Hosts"
  assign where match("*-dev", host.name)
}
```

23.2.18.1 Add Service to Hostgroup where Host is Member

In order to associate a service with all hosts in a host group the apply keyword can be used:

```
apply Service "ping4" {
  import "generic-service"

  check_command = "ping4"

  assign where "dev-hosts" in host.groups
}
```

23.2.19 Notifications

Notifications are a new object type in Icinga 2. Imagine the following notification configuration problem in Icinga 1.x:

- Service A should notify contact X via SMS
- Service B should notify contact X via Mail
- Service C should notify contact Y via Mail and SMS
- Contact X and Y should also be used for authorization

The only way achieving a semi-clean solution is to

- Create contact X-sms, set service_notification_command for sms, assign contact to service A
- Create contact X-mail, set service_notification_command for mail, assign contact to service B
- Create contact Y, set service_notification_command for sms and mail, assign contact to service C

- Create contact X without notification commands, assign to service A and B

Basically you are required to create duplicated contacts for either each notification method or used for authorization only.

Icinga 2 attempts to solve that problem in this way

- Create user X, set SMS and Mail attributes, used for authorization
- Create user Y, set SMS and Mail attributes, used for authorization
- Create notification A-SMS, set command for sms, add user X, assign notification A-SMS to service A
- Create notification B-Mail, set command for mail, add user X, assign notification Mail to service B
- Create notification C-SMS, set command for sms, add user Y, assign notification C-SMS to service C
- Create notification C-Mail, set command for mail, add user Y, assign notification C-Mail to service C

Previously in Icinga 1.x it looked like this:

```
service -> (contact, contactgroup) -> notification command
```

In Icinga 2 it will look like this:

```
Service -> Notification -> NotificationCommand
                        -> User, UserGroup
```

23.2.19.1 Escalations

Escalations in Icinga 1.x require a separated object matching on existing objects. Escalations happen between a defined start and end time which is calculated from the notification_interval:

```
start = notification start + (notification_interval * first_notification)
end = notification start + (notification_interval * last_notification)
```

In theory first_notification and last_notification can be set to readable numbers. In practice users are manipulating those attributes in combination with notification_interval in order to get a start and end time.

In Icinga 2 the notification object can be used as notification escalation if the start and end times are defined within the 'times' attribute using duration literals (e.g. 30m).

The Icinga 2 escalation does not replace the current running notification. In Icinga 1.x it's required to copy the contacts from the service notification to the escalation to guarantee the normal notifications once an escalation happens. That's not necessary with Icinga 2 only requiring an additional notification object for the escalation itself.

23.2.19.2 Notification Options

Unlike Icinga 1.x with the 'notification_options' attribute with comma-separated state and type filters, Icinga 2 uses two configuration attributes for that. All state and type filter use long names OR'd with a pipe together

```
notification_options w,u,c,r,f,s
```

```
states = [ Warning, Unknown, Critical ]
types = [ Problem, Recovery, FlappingStart, FlappingEnd, DowntimeStart, DowntimeEnd, DowntimeRe
```

Icinga 2 adds more fine-grained type filters for acknowledgements, downtime, and flapping type (start, end, ...).

23.2.20 Dependencies and Parents

In Icinga 1.x it's possible to define host parents to determine network reachability and keep a host's state unreachable rather than down. Furthermore there are host and service dependencies preventing unnecessary checks and notifications. A host must not depend on a service, and vice versa. All dependencies are configured as separate objects and cannot be set directly on the host or service object.

A service can now depend on a host, and vice versa. A service has an implicit dependency (parent) to its host. A host to host dependency acts implicitly as host parent relation.

The former `host_name` and `dependent_host_name` have been renamed to `parent_host_name` and `child_host_name` (same for the service attribute). When using apply rules the child attributes may be omitted.

For detailed examples on how to use the dependencies please check the dependencies chapter.

Dependencies can be applied to hosts or services using the apply rules.

The `StatusDataWriter`, `IdoMysqlConnection` and `LivestatusListener` types support the Icinga 1.x schema with dependencies and parent attributes for compatibility reasons.

23.2.21 Flapping

The Icinga 1.x flapping detection uses the last 21 states of a service. This value is hardcoded and cannot be changed. The algorithm on determining a flapping state is as follows:

```
flapping value = (number of actual state changes / number of possible state changes)
```

The flapping value is then compared to the low and high flapping thresholds.

The algorithm used in Icinga 2 does not store the past states but calculates the flapping threshold from a single value based on counters and half-life values. Icinga 2 compares the value with a single flapping threshold configuration attribute.

23.2.22 Check Result Freshness

Freshness of check results must be enabled explicitly in Icinga 1.x. The attribute `freshness_threshold` defines the threshold in seconds. Once the threshold is triggered, an active freshness check is executed defined by the `check_command` attribute. Both check methods (active and passive) use the same freshness check method.

In Icinga 2 active check freshness is determined by the `check_interval` attribute and no incoming check results in that period of time (last check + check interval). Passive check freshness is calculated from the `check_interval` attribute if set. There is no extra `freshness_threshold` attribute in Icinga 2. If the freshness checks are invalid, a new service check is forced.

23.2.23 Real Reload

In Nagios / Icinga 1.x a daemon reload does the following:

- receive reload signal SIGHUP
- stop all events (checks, notifications, etc.)
- read the configuration from disk and validate all config objects in a single threaded fashion
- validation NOT ok: stop the daemon (cannot restore old config state)
- validation ok: start with new objects, dump status.dat / ido

Unlike Icinga 1.x the Icinga 2 daemon reload does not block any event execution during config validation:

- receive reload signal SIGHUP
- fork a child process, start configuration validation in parallel work queues
- parent process continues with old configuration objects and the event scheduling (doing checks, replicating cluster events, triggering alert notifications, etc.)
- validation NOT ok: child process terminates, parent process continues with old configuration state (this is **essential** for the cluster config synchronisation)
- validation ok: child process signals parent process to terminate and save its current state (all events until now) into the icinga2 state file
- parent process shuts down writing icinga2.state file

- child process waits for parent process gone, reads the icinga2 state file and synchronizes all historical and status data
- child becomes the new session leader

The DB IDO configuration dump and status/historical event updates use a queue not blocking event execution. Same goes for any other enabled feature. The configuration validation itself runs in parallel allowing fast verification checks.

That way your monitoring does not stop during a configuration reload.

23.2.24 State Retention

Icinga 1.x uses the `retention.dat` file to save its state in order to be able to reload it after a restart. In Icinga 2 this file is called `icinga2.state`.

The format is **not** compatible with Icinga 1.x.

23.2.25 Logging

Icinga 1.x supports syslog facilities and writes its own `icinga.log` log file and archives. These logs are used in Icinga 1.x to generate historical reports.

Icinga 2 compat library provides the CompatLogger object which writes the icinga.log and archive in Icinga 1.x format in order to stay compatible with addons.

The native Icinga 2 logging facilities are split into three configuration objects: SyslogLogger, FileLogger, StreamLogger. Each of them has their own severity and target configuration.

The Icinga 2 daemon log does not log any alerts but is considered an application log only.

23.2.26 Broker Modules and Features

Icinga 1.x broker modules are incompatible with Icinga 2.

In order to provide compatibility with Icinga 1.x the functionality of several popular broker modules was implemented for Icinga 2:

- IDOUtils
- Livestatus
- Cluster (allows for high availability and load balancing)

23.2.27 Distributed Monitoring

Icinga 1.x uses the native "obsess over host/service" method which requires the NSCA addon passing the slave's check results passively onto the master's external command pipe. While this method may be used for check load distribution, it does not provide any configuration distribution out-of-the-box. Furthermore comments, downtimes, and other stateful runtime data is not synced between the master and slave nodes. There are addons available solving the check and configuration distribution problems Icinga 1.x distributed monitoring currently suffers from.

Icinga 2 implements a new built-in distributed monitoring architecture, including config and check distribution, IPv4/IPv6 support, SSL certificates and zone support for DMZ. High Availability and load balancing are also part of the Icinga 2 Cluster feature, next to local replay logs on connection loss ensuring that the event history is kept in sync.

24 Appendix

24.1 External Commands List

Additional details can be found in the Icinga 1.x Documentation

Command name	Parameters	Description
PROCESS_HOST_CHECK_RESULT	;<host_name>;<status_code>;<plugin_output> (3)	
PROCESS_SERVICE_CHECK_RESULT	;<host_name>;<service_name>;<return_code>;<plugin_output> (4)	
SCHEDULE_HOST_CHECK	;<host_name>;<check_time> (2)	
SCHEDULE_FORCED_HOST_CHECK	;<host_name>;<check_time> (2)	
SCHEDULE_SVC_CHECK	;<host_name>;<service_name>;<check_time> (3)	
SCHEDULE_FORCED_SVC_CHECK	;<host_name>;<service_name>;<check_time> (3)	
ENABLE_HOST_CHECK	;<host_name> (1)	-
DISABLE_HOST_CHECK	;<host_name> (1)	-
ENABLE_SVC_CHECK	;<host_name>;<service_name> (2)	
DISABLE_SVC_CHECK	;<host_name>;<service_name> (2)	
SHUTDOWN_PROCESS	-	-
RESTART_PROCESS	-	-

Command name	Parameters	Description
SCHEDULE_FORCED_HOST_SVC_CHECKS	<host_name>;<service_name>;<check_time> (2)	-
SCHEDULE_HOST_SVC_CHECKS	<host_name>;<check_time> (2)	-
ENABLE_HOST_SVC_CHECKS	<host_name> (1)	-
DISABLE_HOST_SVC_CHECKS	<host_name> (1)	-
ACKNOWLEDGE_SVC_PROBLEM	<host_name>;<service_name>;<sticky>;<notify>;<persistent>;<author> (7)	Note: Icinga 2 treats all comments as persistent.
ACKNOWLEDGE_SVC_PROBLEM_EXPIRE	<host_name>;<service_name>;<sticky>;<notify>;<persistent>;<timesta (8)	Note: Icinga 2 treats all comments as persistent.
REMOVE_SVC_ACKNOWLEDGEMENT	<host_name>;<service_name> (2)	-
ACKNOWLEDGE_HOST_PROBLEM	<host_name>;<sticky>;<notify>;<persistent>;<author>;<comment> (6)	Note: Icinga 2 treats all comments as persistent.
ACKNOWLEDGE_HOST_PROBLEM_EXPIRE	<host_name>;<sticky>;<notify>;<persistent>;<timestamp>;<author>;< (7)	Note: Icinga 2 treats all comments as persistent.
REMOVE_HOST_ACKNOWLEDGEMENT	<host_name> (1)	-
DISABLE_HOST_FLAP_DETECTION	<host_name> (1)	-
ENABLE_HOST_FLAP_DETECTION	<host_name> (1)	-
DISABLE_SVC_FLAP_DETECTION	<host_name>;<service_name> (2)	-
ENABLE_SVC_FLAP_DETECTION	<host_name>;<service_name> (2)	-
ENABLE_HOSTGROUP_SVC_CHECKS	<hostgroup_name> (1)	-
DISABLE_HOSTGROUP_SVC_CHECKS	<hostgroup_name> (1)	-
ENABLE_SERVICEGROUP_SVC_CHECKS	<servicegroup_name> (1)	-
DISABLE_SERVICEGROUP_SVC_CHECKS	<servicegroup_name> (1)	-
ENABLE_PASSIVE_HOST_CHECKS	<host_name> (1)	-
DISABLE_PASSIVE_HOST_CHECKS	<host_name> (1)	-
ENABLE_PASSIVE_SVC_CHECKS	<host_name>;<service_name> (2)	-

Command name	Parameters	Description
DISABLE_PASSIVE_SVC_CHECKS	;<host_name>;<service_name> (2)	-
ENABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS	;<servicegroup_name> (1)	-
DISABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS	;<servicegroup_name> (1)	-
ENABLE_HOSTGROUP_PASSIVE_SVC_CHECKS	;<hostgroup_name> (1)	-
DISABLE_HOSTGROUP_PASSIVE_SVC_CHECKS	;<hostgroup_name> (1)	-
PROCESS_FILE	;<file_name>;<delete> (2)	-
SCHEDULE_SVC_DOWNTIME	;<host_name>;<service_name>;<start_time>;<end_time>;<fixed>;<trig (9)	-
DEL_SVC_DOWNTIME	;<downtime_id> (1)	-
SCHEDULE_AND_PROPAGATE_HOST_DOWNTIME	;<host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duratio (8)	-
SCHEDULE_AND_PROPAGATE_TRIGGERED_HOST_DOWNTIME	;<host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duratio (8)	-
SCHEDULE_HOST_DOWNTIME	;<host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duratio (8)	-
DEL_HOST_DOWNTIME	;<downtime_id> (1)	-
DEL_DOWNTIME_BY_HOST_NAME	;<host_name>[;<service_name>;[;<start_time;>[;<comment_text;>]]] (1)	-
SCHEDULE_HOST_SVC_DOWNTIME	;<host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duratio (8)	-
SCHEDULE_HOSTGROUP_HOST_DOWNTIME	;<hostgroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<d (8)	-
SCHEDULE_HOSTGROUP_SVC_DOWNTIME	;<hostgroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<d (8)	-
SCHEDULE_SERVICEGROUP_HOST_DOWNTIME	;<servicegroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>; (8)	-
SCHEDULE_SERVICEGROUP_SVC_DOWNTIME	;<servicegroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>; (8)	-
ADD_HOST_COMMENT	;<host_name>;<persistent>;<author>;<comment> (4)	Note: Icinga 2 treats all comments as persistent.
DEL_HOST_COMMENT	;<comment_id> (1)	-
ADD_SVC_COMMENT	;<host_name>;<service_name>;<persistent>;<author>;<comment> (5)	Note: Icinga 2 treats all comments as persistent.
DEL_SVC_COMMENT	;<comment_id> (1)	-

Command name	Parameters	Description
DEL_ALL_HOST_COMMENTS	<host_name> (1)	-
DEL_ALL_SVC_COMMENTS	<host_name>;<service_name> (2)	-
SEND_CUSTOM_HOST_NOTIFICATION	<host_name>;<options>;<author>;<comment> (4)	-
SEND_CUSTOM_SVC_NOTIFICATION	<host_name>;<service_name>;<options>;<author>;<comment> (5)	-
DELAY_HOST_NOTIFICATION	<host_name>;<notification_time> (2)	-
DELAY_SVC_NOTIFICATION	<host_name>;<service_name>;<notification_time> (3)	-
ENABLE_HOST_NOTIFICATIONS	<host_name> (1)	-
DISABLE_HOST_NOTIFICATIONS	<host_name> (1)	-
ENABLE_SVC_NOTIFICATIONS	<host_name>;<service_name> (2)	-
DISABLE_SVC_NOTIFICATIONS	<host_name>;<service_name> (2)	-
ENABLE_HOST_SVC_NOTIFICATIONS	<host_name> (1)	-
DISABLE_HOST_SVC_NOTIFICATIONS	<host_name> (1)	-
DISABLE_HOSTGROUP_HOST_CHECKS	<hostgroup_name> (1)	-
DISABLE_HOSTGROUP_PASSIVE_HOST_CHECKS	<hostgroup_name> (1)	-
DISABLE_SERVICEGROUP_HOST_CHECKS	<servicegroup_name> (1)	-
DISABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS	<servicegroup_name> (1)	-
ENABLE_HOSTGROUP_HOST_CHECKS	<hostgroup_name> (1)	-
ENABLE_HOSTGROUP_PASSIVE_HOST_CHECKS	<hostgroup_name> (1)	-
ENABLE_SERVICEGROUP_HOST_CHECKS	<servicegroup_name> (1)	-
ENABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS	<servicegroup_name> (1)	-
ENABLE_NOTIFICATIONS	-	-
DISABLE_NOTIFICATIONS	-	-
ENABLE_FLAP_DETECTION	-	-
DISABLE_FLAP_DETECTION	-	-
ENABLE_EVENT_HANDLERS	-	-
DISABLE_EVENT_HANDLERS	-	-
ENABLE_PERFORMANCE_DATA	-	-
DISABLE_PERFORMANCE_DATA	-	-
START_EXECUTING_HOST_CHECKS	-	-

Command name	Parameters	Description
STOP_EXECUTING_HOST_CHECKS		-
START_EXECUTING_SVC_CHECKS		-
STOP_EXECUTING_SVC_CHECKS		-
CHANGE_NORMAL_SVC_CHECK_INTERVAL (3)	<host_name>;<service_name>;<check_interval>	
CHANGE_NORMAL_HOST_CHECK_INTERVAL (2)	<host_name>;<check_interval>	
CHANGE_RETRY_SVC_CHECK_INTERVAL (3)	<host_name>;<service_name>;<check_interval>	
CHANGE_RETRY_HOST_CHECK_INTERVAL (2)	<host_name>;<check_interval>	
ENABLE_HOST_EVENT_HANDLER (1)	<host_name>	-
DISABLE_HOST_EVENT_HANDLER (1)	<host_name>	-
ENABLE_SVC_EVENT_HANDLER (2)	<host_name>;<service_name>	
DISABLE_SVC_EVENT_HANDLER (2)	<host_name>;<service_name>	
CHANGE_HOST_EVENT_HANDLER (2)	<host_name>;<event_command_name>	
CHANGE_SVC_EVENT_HANDLER (3)	<host_name>;<service_name>;<event_command_name>	
CHANGE_HOST_CHECK_COMMAND (2)	<host_name>;<check_command_name>	
CHANGE_SVC_CHECK_COMMAND (3)	<host_name>;<service_name>;<check_command_name>	
CHANGE_MAX_HOST_CHECK_ATTEMPTS (2)	<host_name>;<check_attempts>	
CHANGE_MAX_SVC_CHECK_ATTEMPTS (3)	<host_name>;<service_name>;<check_attempts>	
CHANGE_HOST_CHECK_TIMEPERIOD (2)	<host_name>;<timeperiod_name>	
CHANGE_SVC_CHECK_TIMEPERIOD	<host_name>;<service_name>;<timeperiod_name>	
CHANGE_CUSTOM_HOST_VAR (3)	<host_name>;<var_name>;<var_value>	
CHANGE_CUSTOM_SVC_VAR (4)	<host_name>;<service_name>;<var_name>;<var_value>	
CHANGE_CUSTOM_USER_VAR (3)	<user_name>;<var_name>;<var_value>	
CHANGE_CUSTOM_CHECKCOMMAND_VAR (3)	<checkcommand_name>;<var_name>;<var_value>	
CHANGE_CUSTOM_EVENTCOMMAND_VAR (3)	<eventcommand_name>;<var_name>;<var_value>	
CHANGE_CUSTOM_NOTIFICATIONCOMMAND_VAR (3)	<notificationcommand_name>;<var_name>;<var_value>	

Command name	Parameters	Description
ENABLE_HOSTGROUP_HOST_NOTIFICATIONS	<hostgroup_name> (1)	-
ENABLE_HOSTGROUP_SVC_NOTIFICATIONS	<hostgroup_name> (1)	-
DISABLE_HOSTGROUP_HOST_NOTIFICATIONS	<hostgroup_name> (1)	-
DISABLE_HOSTGROUP_SVC_NOTIFICATIONS	<hostgroup_name> (1)	-
ENABLE_SERVICEGROUP_HOST_NOTIFICATIONS	<servicegroup_name> (1)	-
DISABLE_SERVICEGROUP_HOST_NOTIFICATIONS	<servicegroup_name> (1)	-
ENABLE_SERVICEGROUP_SVC_NOTIFICATIONS	<servicegroup_name> (1)	-
DISABLE_SERVICEGROUP_SVC_NOTIFICATIONS	<servicegroup_name> (1)	-

24.2 Schemas

By convention `CheckCommand`, `EventCommand`, and `NotificationCommand` objects are exported using a prefix. This is mandatory for unique objects in the command tables.

Object	Prefix
CheckCommand	check_
EventCommand	event_
NotificationCommand	notification_

24.2.1 DB IDO Schema

There is a detailed documentation for the Icinga IDOUtils 1.x database schema available on [https://docs.icinga.com/latest/en/db_model.html]

24.2.1.1 DB IDO Schema Extensions

Icinga 2 specific extensions are shown below:

New table: `endpointstatus`

Table	Column	Type	Default	Description

Table	Column	Type	Default	Description
endpoints	endpoint_object_id	bigint	NULL	FK: objects table
endpoints	identity	TEXT	NULL	endpoint name
endpoints	node	TEXT	NULL	local node name
endpoints	zone_object_id	bigint	NULL	zone object where this endpoint is a member of

New table: `endpointstatus`

Table	Column	Type	Default	Description
endpointstatus	endpoint_object_id	bigint	NULL	FK: objects table
endpointstatus	identity	TEXT	NULL	endpoint name
endpointstatus	node	TEXT	NULL	local node name
endpointstatus	is_connected	smallint	0	update on endpoint connect/disconnect
endpointstatus	zone_object_id	bigint	NULL	zone object where this endpoint is a member of

New tables: `zones` and `zonestatus`:

Table	Column	Type	Default	Description
zones	zone_object_id	bigint	NULL	FK: objects table
zones	parent_zone_object_id	bigint	NULL	FK: zones table
zones	is_global	smallint	0	zone is global

New columns:

Table	Column	Type	Default	Description
all status/history	endpoint_object_id	bigint	NULL	FK: objects table
servicestatus	check_source	TEXT	NULL	node name where check was executed
hoststatus	check_source	TEXT	NULL	node name where check was executed
statehistory	check_source	TEXT	NULL	node name where check was executed
servicestatus	is_reachable	integer	NULL	object reachability
hoststatus	is_reachable	integer	NULL	object reachability
logentries	object_id	bigint	NULL	FK: objects table (service associated with column)
{host,service}group	notes	TEXT	NULL	-
{host,service}group	notes_url	TEXT	NULL	-
{host,service}group	action_url	TEXT	NULL	-
customvariable*	is_json	integer	0	Defines whether `varvalue` is a json encoded string from custom attributes, or not
servicestatus	original_attributes	TEXT	NULL	JSON encoded dictionary of original attributes if modified at runtime.

Table	Column	Type	Default	Description
hoststatus	original_attributes	TEXT	NULL	JSON encoded dictionary of original attributes if modified at runtime.

Additional command custom variables populated from 'vars' dictionary. Additional global custom variables populated from 'Vars' constant (object_id is NULL).

24.2.2 Livestatus Schema

24.2.2.1 Livestatus Schema Extensions

Icinga 2 specific extensions are shown below:

New table: `endpoints`:

Table	Column
endpoints	name
endpoints	identity
endpoints	node
endpoints	is_connected
endpoints	zone

New table: `zones`:

Table	Column
zone	name
zone	endpoints
zone	parent
zone	global

New columns:

Table	Column
hosts	is_reachable
services	is_reachable

Table	Column
hosts	cv_is_json
services	cv_is_json
contacts	cv_is_json
hosts	check_source
services	check_source
downtimes	triggers
downtimes	trigger_time
commands	custom_variable_names
commands	custom_variable_values
commands	custom_variables
commands	modified_attributes
commands	modified_attributes_list
status	custom_variable_names
status	custom_variable_values
status	custom_variables
hosts	original_attributes
services	original_attributes

Command custom variables reflect the local 'vars' dictionary. Status custom variables reflect the global 'Vars' constant.

24.2.2.2 Livestatus Hosts Table Attributes

Key	Type	Note
name	string	.
display_name	string	.
alias	string	same as display_name.
address	string	.
address6	string	NEW in Icinga.
check_command	string	.
check_command_expanded	string	.
event_handler	string	.
notification_period	string	host with notifications: period.
check_period	string	.
notes	string	.
notes_expanded	string	.
notes_url	string	.
notes_url_expanded	string	.
action_url	string	.
action_url_expanded	string	.
plugin_output	string	.

Key	Type	Note
perf_data	string	.
icon_image	string	.
icon_image_expanded	string	.
icon_image_alt	stirng	.
statusmap_image	string	.
long_plugin_output	string	.
max_check_attempts	int	.
flap_detection_enabled	int	.
check_freshness	int	.
process_performance_data	int	.
accept_passive_checks	int	.
event_handler_enabled	int	.
acknowledgement_type	int	Only 0 or 1.
check_type	int	.
last_state	int	.
last_hard_state	int	.
current_attempt	int	.
last_notification	int	host with notifications: last notification.
next_notification	int	host with notifications: next notification.
next_check	int	.
last_hard_state_change	int	.
has_been_checked	int	.
current_notification_number	int	host with notifications: number.
total_services	int	.
checks_enabled	int	.
notifications_enabled	int	.
acknowledged	int	.
state	int	.
state_type	int	.
no_more_notifications	int	notification_interval == 0 && volatile == false.
last_check	int	.
last_state_change	int	.
last_time_up	int	.
last_time_down	int	.
last_time_unreachable	int	.
is_flapping	int	.
scheduled_downtime_depth	int	.
active_checks_enabled	int	.
modified_attributes	array	.
modified_attributes_list	array	.

Key	Type	Note
check_interval	double	.
retry_interval	double	.
notification_interval	double	host with notifications: smallest interval.
low_flap_threshold	double	flapping_threshold
high_flap_threshold	double	flapping_threshold
latency	double	.
execution_time	double	.
percent_state_change	double	flapping.
in_notification_period	int	host with notifications: matching period.
in_check_period	int	.
contacts	array	host with notifications, users and user groups.
downtimes	array	id.
downtimes_with_info	array	id+author+comment.
comments	array	id.
comments_with_info	array	id+author+comment.
comments_with_extra_info	array	id+author+comment+entry_type+entry_time.
custom_variable_names	array	.
custom_variable_values	array	.
custom_variables	array	Array of custom variable array pair.
parents	array	Direct host parents.
childs	array	Direct host children (Note: `childs` is inherited from the origin MK_Livestatus protocol).
num_services	int	.
worst_service_state	int	All services and their worst state.
num_services_ok	int	All services with Ok state.
num_services_warn	int	All services with Warning state.
num_services_crit	int	All services with Critical state.
num_services_unknown	int	All services with Unknown state.
worst_service_hard_state	int	All services and their worst hard state.
num_services_hard_ok	int	All services in a hard state with Ok state.

Key	Type	Note
num_services_hard_warn	int	All services in a hard state with Warning state.
num_services_hard_crit	int	All services in a hard state with Critical state.
num_services_hard_unknown	int	All services in a hard state with Unknown state.
hard_state	int	Returns OK if state is OK. Returns current state if now a hard state type. Returns last hard state otherwise.
staleness	int	Indicates time since last check normalized onto the check_interval.
groups	array	All hostgroups this host is a member of.
contact_groups	array	All usergroups associated with this host through notifications.
services	array	All services associated with this host.
services_with_state	array	All services associated with this host with state and hasbeenchecked.
services_with_info	array	All services associated with this host with state, hasbeenchecked and output.

Not supported: `initial_state`, `pending_flex_downtime`, `check_flapping_recovery_notification`, `is_executing`, `check_options`, `obsess_over_host`, `first_notification_delay`, `x_3d`, `y_3d`, `z_3d`, `x_2d`, `y_2d`, `filename`, `pnpgraph_present`.

24.2.2.3 Livestatus Hostgroups Table Attributes

Key	Type	Note
name	string	.
alias	string	`display_name` attribute.
notes	string	.
notes_url	string	.
action_url	string	.

Key	Type	Note
members	array	.
members_with_state	array	Host name and state.
worst_host_state	int	Of all group members.
num_hosts	int	In this group.
num_hosts_pending	int	.
num_hosts_up	int	.
num_hosts_down	int	.
num_hosts_unreach	int	.
num_services	int	Number of services associated with hosts in this hostgroup.
worst_services_state	int	.
num_services_pending	int	.
num_services_ok	int	.
num_services_warn	int	.
num_services_crit	int	.
num_services_unknown	int	.
worst_service_hard_state	int	.
num_services_hard_ok	int	.
num_services_hard_warn	int	.
num_services_hard_crit	int	.
num_services_hard_unknown	int	.

24.2.2.4 Livestatus Services Table Attributes

Key	Type	Note
description	string	.
display_name	string	.
alias	string	same as display_name.
check_command	string	.
check_command_expanded	string	.
event_handler	string	.
notification_period	string	host with notifications: period.
check_period	string	.
notes	string	.
notes_expanded	string	.
notes_url	string	.
notes_url_expanded	string	.
action_url	string	.
action_url_expanded	string	.
plugin_output	string	.
perf_data	string	.

Key	Type	Note
icon_image	string	.
icon_image_expanded	string	.
icon_image_alt	stirng	.
statusmap_image	string	.
long_plugin_output	string	.
max_check_attempts	int	.
flap_detection_enabled	int	.
check_freshness	int	.
process_performance_data	int	.
accept_passive_checks	int	.
event_handler_enabled	int	.
acknowledgement_type	int	Only 0 or 1.
check_type	int	.
last_state	int	.
last_hard_state	int	.
current_attempt	int	.
last_notification	int	service with notifications: last notification.
next_notification	int	service with notifications: next notification.
next_check	int	.
last_hard_state_change	int	.
has_been_checked	int	.
current_notification_number	int	service with notifications: number.
checks_enabled	int	.
notifications_enabled	int	.
acknowledged	int	.
state	int	.
state_type	int	.
no_more_notifications	int	notification_interval == 0 && volatile == false.
last_check	int	.
last_state_change	int	.
last_time_ok	int	.
last_time_warning	int	.
last_time_critical	int	.
last_time_unknown	int	.
is_flapping	int	.
scheduled_downtime_depth	int	.
active_checks_enabled	int	.
modified_attributes	array	.
modified_attributes_list	array	.
check_interval	double	.

Key	Type	Note
retry_interval	double	.
notification_interval	double	service with notifications: smallest interval.
low_flap_threshold	double	flapping_threshold
high_flap_threshold	double	flapping_threshold
latency	double	.
execution_time	double	.
percent_state_change	double	flapping.
in_notification_period	int	service with notifications: matching period.
in_check_period	int	.
contacts	array	service with notifications, users and user groups.
downtimes	array	id.
downtimes_with_info	array	id+author+comment.
comments	array	id.
comments_with_info	array	id+author+comment.
comments_with_extra_info	array	id+author+comment+entry_type+entry_time.
custom_variable_names	array	.
custom_variable_values	array	.
custom_variables	array	Array of custom variable array pair.
hard_state	int	Returns OK if state is OK. Returns current state if now a hard state type. Returns last hard state otherwise.
staleness	int	Indicates time since last check normalized onto the check_interval.
groups	array	All hostgroups this host is a member of.
contact_groups	array	All usergroups associated with this host through notifications.
host_	join	Prefix for attributes from implicit join with hosts table.

Not supported: `initial_state`, `is_executing`, `check_options`, `obsess_over_service`, `first_notification_delay`, `pnpgraph_present`.

24.2.2.5 Livestatus Servicegroups Table Attributes

Key	Type	Note
name	string	.
alias	string	`display_name` attribute.
notes	string	.
notes_url	string	.
action_url	string	.
members	array	CSV format uses `host|service` syntax.
members_with_state	array	Host, service, hoststate, servicestate.
worst_service_state	int	.
num_services	int	.
num_services_pending	int	.
num_services_ok	int	.
num_services_warn	int	.
num_services_crit	int	.
num_services_unknown	int	.
num_services_hard_ok	int	.
num_services_hard_warn	int	.
num_services_hard_crit	int	.
num_services_hard_unknown	int	.

24.2.2.6 Livestatus Contacts Table Attributes

Key	Type	Note
name	string	.
alias	string	`display_name` attribute.
email	string	.
pager	string	.
host_notification_period	string	.
service_notification_period	string	.
host_notifications_enabled	int	.
service_notifications_enabled	int	.
in_host_notification_period	int	.
in_service_notification_period	int	.
custom_variable_names	array	.
custom_variable_values	array	.
custom_variables	array	Array of customvariable array pairs.
modified_attributes	array	.
modified_attributes_list	array	.

Not supported: `can_submit_commands`.

24.2.2.7 Livestatus Contactgroups Table Attributes

Key	Type	Note
name	string	.
alias	string	`display_name` attribute.
members	array	.

24.2.2.8 Livestatus Commands Table Attributes

Key	Type	Note
name	string	3 types of commands in Icinga 2.
line	string	.

24.2.2.9 Livestatus Status Table Attributes

Key	Type	Note
connections	int	Since application start.
connections_rate	double	.
service_checks	int	Since application start.
service_checks_rate	double	.
host_checks	int	Since application start.
host_checks_rate	double	.
external_commands	int	Since application start.
external_commands_rate	double	.
nagios_pid	string	Application PID.
enable_notifications	int	.
execute_service_checks	int	.
accept_passive_service_checks	int	.
execute_host_checks	int	.
accept_passive_host_checks	int	.
enable_event_handlers	int	.
check_service_freshness	int	.
check_host_freshness	int	.
enable_flap_detection	int	.
process_performance_data	int	.
check_external_commands	int	Always enabled.
program_start	int	In seconds.
last_command_check	int	Always.
interval_length	int	Compatibility mode: 60.
num_hosts	int	.
num_services	int	.
program_version	string	2.0.
livestatus_active_connections	string	.

Not supported: `neb_callbacks`, `neb_callbacks_rate`, `requests`, `requests_rate`, `forks`, `forks_rate`, `log_messages`, `log_messages_rate`, `livechecks`, `livechecks_rate`, `livecheck_overflows`, `livecheck_overflows_rate`, `obsess_over_services`, `obsess_over_hosts`, `last_log_rotation`, `external_command_buffer_slots`, `external_command_buffer_usage`, `external_command_buffer_max`, `cached_log_messages`, `livestatus_queued_connections`, `livestatus_threads`.

24.2.2.10 Livestatus Comments Table Attributes

Key	Type	Note
author	string	.
comment	string	.
id	int	legacy_id.
entry_time	string	Seconds.
type	int	1=host, 2=service.
is_service	int	.
persistent	int	Always.
source	string	Always external (1).
entry_type	int	.
expires	int	.
expire_time	string	Seconds.
service_	join	Prefix for attributes from implicit join with services table.
host_	join	Prefix for attributes from implicit join with hosts table.

24.2.2.11 Livestatus Downtimes Table Attributes

Key	Type	Note
author	string	.
comment	string	.
id	int	legacy_id.
entry_time	string	Seconds.
type	int	1=active, 0=pending.
is_service	int	.
start_time	string	Seconds.
end_time	string	Seconds.
fixed	int	0=flexible, 1=fixed.
duration	int	.
triggered_by	int	legacy_id.
triggers	int	NEW in Icinga 2.

Key	Type	Note
trigger_time	string	NEW in Icinga 2.
service_	join	Prefix for attributes from implicit join with services table.
host_	join	Prefix for attributes from implicit join with hosts table.

24.2.2.12 Livestatus Timeperiod Table Attributes

Key	Type	Note
name	string	.
alias	string	`display_name` attribute.
in	int	Current time is in timeperiod or not.

24.2.2.13 Livestatus Log Table Attributes

Key	Type	Note
time	int	Time of log event (unix timestamp).
lineno	int	Line number in `CompatLogger` log file.
class	int	Log message class: 0=info, 1=state, 2=program, 3=notification, 4=passive, 5=command.
message	string	Complete message line.
type	string	Text before the colon `:`.
options	string	Text after the colon `:`.
comment	string	Comment if available.
plugin_output	string	Check output if available.
state	int	Host or service state.
state_type	int	State type if available.
attempt	int	Current check attempt.
service_description	string	.
host_name	string	.
contact_name	string	.
command_name	string	.

Key	Type	Note
current_service_	join	Prefix for attributes from implicit join with services table.
current_host_	join	Prefix for attributes from implicit join with hosts table.
current_contact_	join	Prefix for attributes from implicit join with contacts table.
current_command_	join	Prefix for attributes from implicit join with commands table.

24.2.2.14 Livestatus Statehist Table Attributes

Key	Type	Note
time	int	Time of log event (unix timestamp).
lineno	int	Line number in `CompatLogger` log file.
from	int	Start timestamp (unix timestamp).
until	int	End timestamp (unix timestamp).
duration	int	until-from.
duration_part	double	duration / query_part.
state	int	State: 0=ok, 1=warn, 2=crit, 3=unknown, -1=notmonitored.
host_down	int	Host associated with the service is down or not.
in_downtime	int	Host/service is in downtime.
in_host_downtime	int	Host associated with the service is in a downtime or not.
is_flapping	int	Host/service is flapping.
in_notification_period	int	Host/service notification periods match or not.
notification_period	string	Host/service notification period.
host_name	string	.
service_description	string	.

Key	Type	Note
log_output	string	Log file output for this state.
duration_ok	int	until-from for OK state.
duration_part_ok	double	.
duration_warning	int	until-from for Warning state.
duration_part_warning	double	.
duration_critical	int	until-from for Critical state.
duration_part_critical	double	.
duration_unknown	int	until-from for Unknown state.
duration_part_unknown	double	.
duration_unmonitored	int	until-from for Not-Monitored state.
duration_part_unmonitored	double	.
current_service_	join	Prefix for attributes from implicit join with services table.
current_host_	join	Prefix for attributes from implicit join with hosts table.

Not supported: `debug_info`.

24.2.2.15 Livestatus Hostsbygroup Table Attributes

All hosts table attributes grouped with the hostgroups table prefixed with `hostgroup_`.

24.2.2.16 Livestatus Servicesbygroup Table Attributes

All services table attributes grouped with the servicegroups table prefixed with `servicegroup_`.

24.2.2.17 Livestatus Servicesbyhostgroup Table Attributes

All services table attributes grouped with the hostgroups table prefixed with `hostgroup_`.