

An Introduction to Monitoring with Nagios

Laurent Andrey Rémi Badonnel

LORIA - INRIA Grand Est

ISSNSM'2008, Zurich

Introduction

Nagios at a glance

- Key concepts
- Functional architecture
- Services and service states

Nagios configuration

- Object definitions
- Other elements
- Example scenario

Checks and their execution

- Local checks
- Remote checks

Advanced configurations

Conclusions

References

- ▶ www.nagios.org: official distribution (core, plugins and documentation)
- ▶ www.nagiosexchange.org: lots of complementary plugins



W. Barth.

Nagios, System and Network Monitoring.

Open Source Press GmbH, first edition, 2006.

ISBN: 1-59327-070-4.

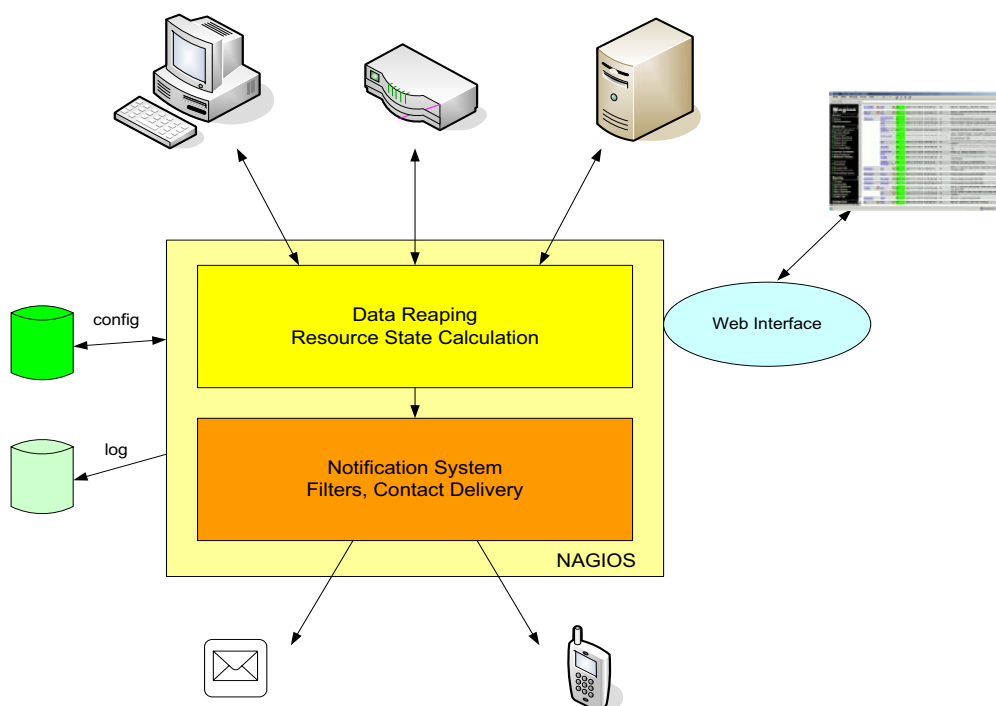
What is Nagios? What is Nagios useful for?

- ▶ A widely-used monitoring tool for trouble-shooting
 - ▶ Simple and open source
 - ▶ Network, system, service levels
- ▶ With a sophisticated (?) notification system to inform administrators when something goes wrong
- ▶ Nagios provides support to administrator(s) for detecting problems **before** users (including the boss!)
 - ▶ Mail server failure
 - ▶ Hard drive overload
 - ▶ Network outage

Key concepts

- ▶ Colored area concept
 - ▶ Green/Yellow/Red (Ok/Warning/Critical)
- ▶ No performance analysis or display (*a priori*)
- ▶ Checks using **external** commands (plugins)
- ▶ Various possibilities for **remote** checks
- ▶ Possibility for **passive checks** (from managed resources)
- ▶ Web interface + notifications

Functional architecture



Architecture at run-time

- ▶ Data reaping + notification system = *nagios processes*
 - ▶ Can be run as a service (rcX.d, soft runlevel)
- ▶ Web interface
 - ▶ External web server (Apache)
 - ▶ Bunch of cgi scripts (part of Nagios)
- ▶ Configuration
 - ▶ Simple text files
 - ▶ Or a postgres database
- ▶ Logs (local files)
- ▶ Named pipe (unix domain socket) to enable nagios to **receive** commands (from cgi, passive asynchronous events)

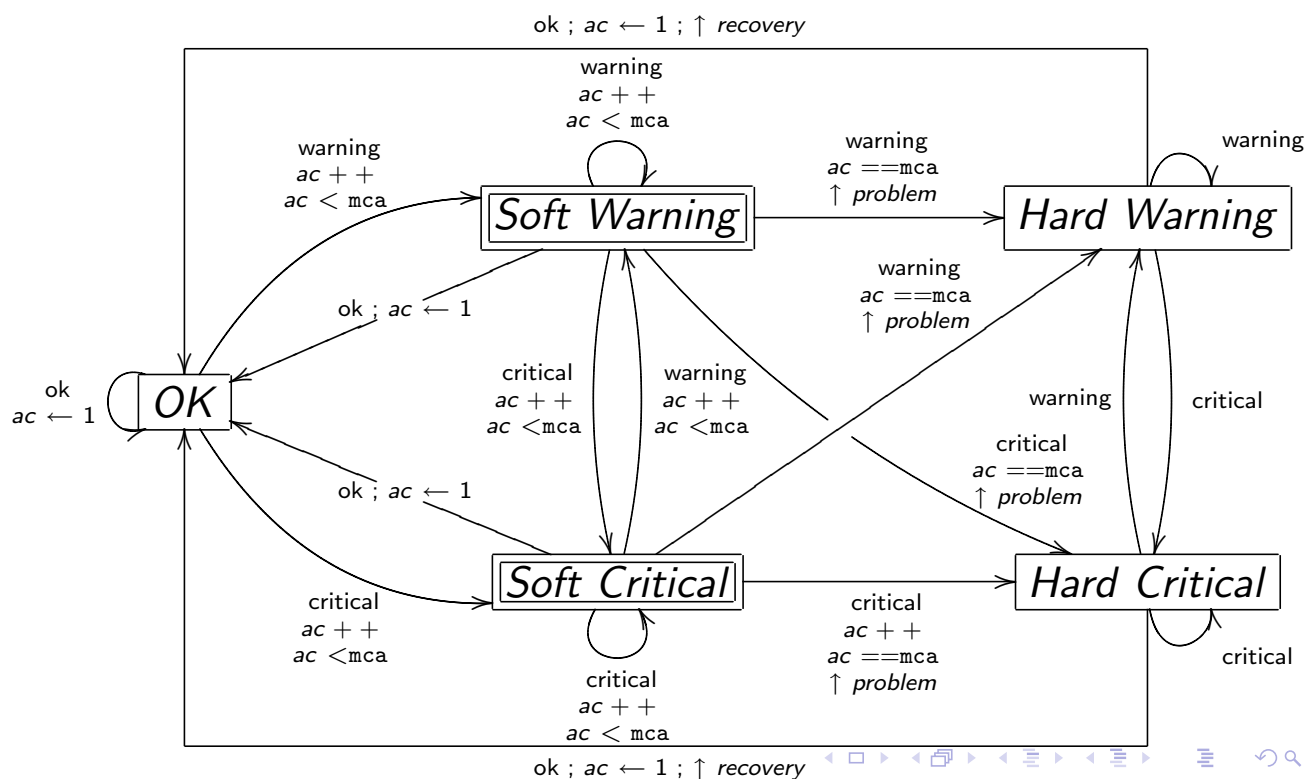
Service, service check

- ▶ Service
 - ▶ Service delivered by a software
 - ▶ Percentage of free space on a partition
 - ▶ Bandwidth usage on a network interface ...
- ▶ Service check
 - ▶ Provides state information on a service
 - ▶ Returns a value: OK, WARNING, CRITICAL (exit status 0, 1, 2), UNKNOWN (exit status 3, due to time out or plugin runtime trouble) to reflect the Nagios view about this service
 - ▶ Can be local (OS calls) or remote (ICMP, NRPE, SNMP ...)
 - ▶ Is implemented by a plugin (external command/script)

Service states

- ▶ Service states are the mirror of what nagios observes
- ▶ States: OK, WARNING, CRITICAL, UNKNOWN
- ▶ Transitions from one state to another one based on results provided by checks
- ▶ Critical and warning states are shadowed by related *soft* states
 - ▶ A service goes first to a *soft* state
 - ▶ Attempt count mechanism to reach a definitive *hard* state
- ▶ User notification can only occur when *hard* states are reached

Service state diagram



Service state diagram legend

- ▶ \boxed{HS} hard state, using **normal_check_interval** between 2 checks
- ▶ \boxed{S} soft state, using **retry_check_interval** between 2 checks
- ▶ \xrightarrow{RS} transition triggered by a check with a return status of $RS \in \{ \text{ok, warning, critical} \}$
- ▶ $\xrightarrow{ac++}$ *Attempt Count* is incremented when transition is triggered

Service state diagram legend (ctd)

- ▶ $\xrightarrow{ac < mca}$ transition is triggered if *Attempt Count* is smaller than the service configuration attribute: **max_check_attempts**
- ▶ $\xrightarrow{ac == mca}$ transition is triggered if *Attempt Count* is equal to the service configuration attribute: **max_check_attempts**
- ▶ $\xrightarrow{ac \leftarrow 1}$ *Attempt Count* is set to 1 then the transition is triggered
- ▶ $\xrightarrow{\uparrow notification}$ a user notification $\in \{ \text{problem, recovery} \}$ is generated when this transition is triggered

Nagios configuration

- ▶ Object-oriented representation
 - ▶ A nagios object describes a specific unit: a service, an host, a contact, a contactgroup ... with attributes and values
 - ▶ kind of inheritance mechanisms, dependencies amongst objets
- ▶ Set of configuration files
 - ▶ Main file: nagios.cfg (ref. to other cfg. files)
 - ▶ 1 file per object type: services.cfg, hosts.cfg, contacts.cfg, checkcommands.cfg, misccommands.cfg, timeperiods.cfg ...
- ▶ Requires an *a priori* knowledge
- ▶ Configuration can also stand into a database

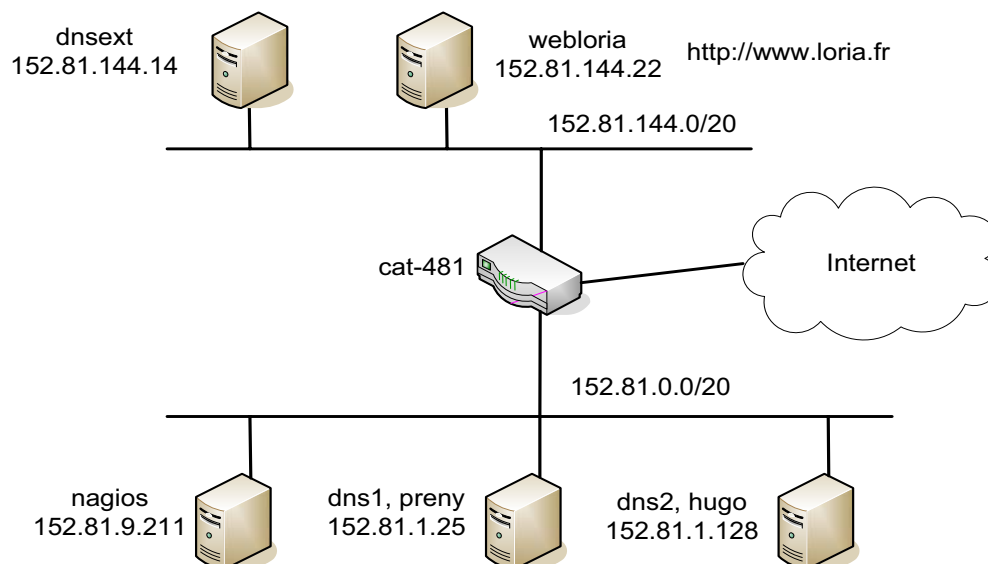
Host

- ▶ A service **have** to be linked to an *host*
- ▶ Only *UP* and *DOWN* states
- ▶ User notification (problem, recovery)
- ▶ Same external checks than services
 - ▶ UP = (WARNING or OK), DOWN = CRITICAL
 - ▶ Typically: ICMP-based checks
- ▶ No active checks if related services are **OK**
- ▶ Host group (cosmetic ...)

Other Nagios elements

- ▶ **contact, contactgroup:** to specify who to notify, how to notify and when to notify
 - ▶ Used in host and service definitions
- ▶ **command:** to execute check plugins or to send user notifications
 - ▶ Links Nagios attributes found in definitions (ex: host name) to plugin parameters
 - ▶ Already provided for most common plugins (see `checkcommands.cfg` file)
 - ▶ Basic wrapping to send emails (`misccommands.cfg` file)

Example scenario



Host definition (example)

hosts.cfg

```
define host {
    host_name          webloria
    alias              webloria linux machine
    address            152.81.144.22
    check_command      check-host-alive
    max_check_attempts 3
    check_period       24x7
    notification_interval 180 # 3 hours
    notification_period 24x7
    notification_options d,r,f,u
    # down, recovery, flapping, unreachable
    contact_groups     administrators
}
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ◀ ≡ ◀

Service definition (example 1)

services.cfg

```
define service{
    host_name          webloria
    service_description http service
    check_command      check_http
    max_check_attempts 3
    normal_check_interval 5
    retry_check_interval 1
    check_period       24x7
    notification_interval 180
    notification_period 24x7
    notification_options w,c,r,f,u
    # warning, critical, recovery, flapping, unreachable
    contact_groups     administrators
}
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ◀ ≡ ◀

Command definitions (example 1)

checkcommands.cfg

```
define command{
  command_name    check-host-alive
  command_line    $USER1$/check_icmp -H $HOSTADDRESS$
}

define command{
  command_name    check_http
  command_line    $USER1$/check_http -H $HOSTADDRESS$
}
```



Service definition (example 2)

services.cfg

```
define service{
  host_name        dnsexst
  service_description    dns service
  check_command    check_name_for_given_dns
                  !www.loria.fr!152.81.144.22

  max_check_attempts    3
  normal_check_interval    5
  retry_check_interval    1
  check_period            24x7
  notification_interval    180
  notification_period    24x7
  notification_options    w,c,r,f,u
  contact_groups          administrators
}
```



Command definition (example 2)

checkcommands.cfg

```
define command{
  command_name check_name_for_given_dns
  command_line $USER1$/check_dns -H $ARG1$ -a $ARG2$
              -s $HOSTADDRESS$
              # $ARG1$: fully qualified name
              # $ARG2$: IP address
}
```

Contact definition (example)

contacts.cfg

```
define contact{
  contact_name andrey
  alias laurent andrey
  service_notification_period 24x7
  host_notification_period 24x7
  service_notification_options w,u,c,r
  host_notification_options d,r
  service_notification_commands notify-by-email
  host_notification_commands host-notify-by-email
  email andrey@loria.fr
}
```

Contactgroup definition (example)

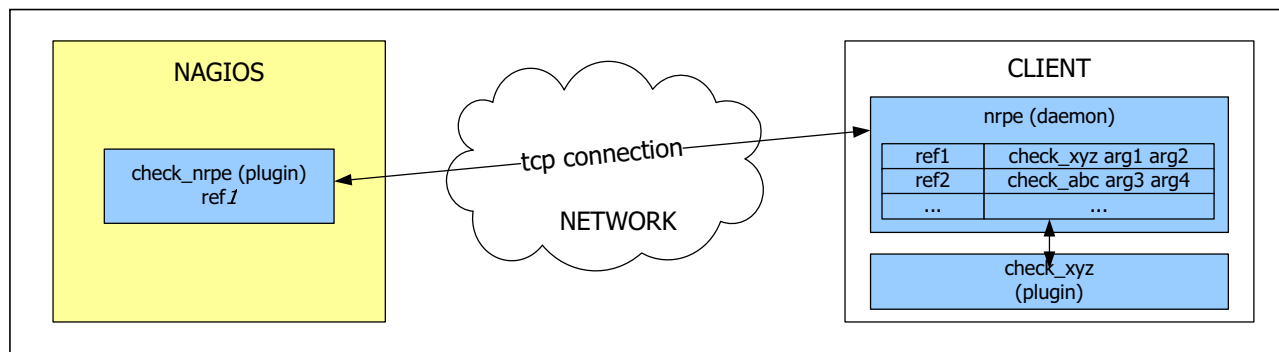
contacts.cfg

```
define contactgroup{  
    contactgroup_name    administrators  
    alias                 group of administrators  
    members              andrey  
}
```

Local checks

- ▶ Getting information about your **local** system
- ▶ Plugins based on system commands (such as ps, df, uptime)
- ▶ `check_disk -w 30% -c 15% -p /var`
- ▶ `check_load -w 2.0,1.0,0.5 -c 4.0,2.0,1.0`
- ▶ `check_procs -w 150 -c 250 --metric=PROCS`

Checks using NRPE (Nagios Remote Plugin Executor)



- ▶ Nagios executes the `check_nrpe` plugin
- ▶ To interact with a dedicated daemon called `nrpe`
- ▶ Based on pre-configured plugin invocations

Other remote checks

- ▶ Checks using SNMP
 - ▶ Collecting management information from SNMP agents
 - ▶ `check_snmp` plugin \Leftrightarrow `net-snmp snmpget`
 - ▶ SNMP reply + warning and critical limits \Rightarrow service state
- ▶ Checks using NSCA (Nagios Service Check Acceptor)
 - ▶ Passive method where checks are initiated by the resources themselves (close to SNMP traps)
 - ▶ A NSCA daemon waits for incoming check results (on the Nagios machine), while the `send_nsca` program (on the remote machines) sends messages containing check results

Making configuration more simple

- ▶ Monitoring the same service on several hosts
 - ▶ setting the **host_name** attribute of the service as a comma separated list of host names
 - ▶ or setting an **hostgroup_name** attribute for the service
- ▶ Defining template-based objects
 - ▶ Notion of inheritance
 - ▶ Factorizing many low-interest attributes
 - ▶ **register** attribute to define a template
 - ▶ **use** attribute to inherit from a template

Service and hostgroup (example)

```
define hostgroup{
    hostgroup_name dns_hosts
    alias          hosts supporting DNS
    members       dns1 , dns2 , dnsex
}

define service{
    hostgroup_name      dns_hosts
    service_description dns service
    check_command       check_name_for_given_dns
                      !www.loria.fr!152.81.144.22
    max_check_attempts 3
    normal_check_interval 5
    retry_check_interval 1
    check_period        24x7
    notification_interval 180
    notification_period 24x7
    notification_options w,c,r,f,u
    contact_groups      administrators
}
```

Specifying dependencies

- ▶ Remainder: service critical state \Rightarrow host check
- ▶ How does Nagios make a difference between:
 - ▶ case 1: **webloria** is really *down*
 - ▶ case 2: **webloria** is *unreachable* due to the network?
- ▶ Solution: defining dependency relationships amongst objects (**parents** attribute)
 - ▶ If an host is detected as **down**, the parent is checked
 - ▶ If the parent is **OK**, the initial host is really declared *down*
 - ▶ If not, the host is declared *unreachable*
- ▶ Ex: **cat-481** router defined as a parent of **webloria**

Conclusions

- ▶ Open source monitoring solution
- ▶ Based on simple concepts: checks, states, notifications
- ▶ Easily extensible and integrable
- ▶ No discovery mechanism
- ▶ Experience it during lab exercises!
 - ▶ Monitoring hosts and their services
 - ▶ Developing and testing our own plugin
 - ▶ Experimenting state transitions and notifications