# IF BSI 01: Monitoring and Configuration Management in Data Centers

**Patricia Jung**
**Informatica Feminale 2013**

# Reoccuring tasks in data centers

- Set up new hosts (physical or virtual) on demand
- Make sure hosts are always (not only after initial setup) configured according to specification
- Make sure they provide expected services reliably
- Make sure problems become known ideally before users start noticing

# How to fullfill these tasks fast and reliably

**Task**

- Set up new hosts (physical or virtual) on demand

**Example solutions**

- Templates
- FAI

# How to fullfill these tasks fast and reliably

**Tasks**

- Set up new hosts (physical or virtual) on demand
- Make sure hosts are always (not only after initial setup) configured according to specification
- Make sure they provide expected services reliably

**Solution:** Configuration management, e.g. using Open Source software like

- Cfengine
- Chef
- Puppet

# How to fullfill these tasks fast and reliably

**Tasks**

- Make sure they provide expected services reliably
- Make sure problems become known ideally before users start noticing

**Solution:** Monitoring

- State monitoring
- Long-term monitoring

https://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems

# Open Source monitoring solutions

- State Monitoring
  - Nagios
  - Icinga
  - Shinken
  - Zabbix
  - ...
- Long-term Monitoring
  - Munin
  - Cacti
  - InGraph
  - Graphite
  - ...

# Configuration Management with Puppet

http://www.puppetlabs.com/

- Dual license: Open Source and extended enterprise product
- Open Source edition (Apache 2.0 license, GPLed until v2.7) ships with most Linux distros, FreeBSD, Solaris from 11.2, Amazon EC2 (Linux AMI), ...
- Available for Windows, AIX, HP-UX, Solaris, ...
- Supports Nagios

# Monitoring with Nagios/Icinga

http://nagios.org/ http://icinga.org/

- Open Source (GPL v2)
- Extensible in all directions
- State monitoring, can be extended to long-term monitoring
- Nagios: de-facto industry standard, part of many commercial solutions
- Icinga: Nagios-compatible, many fixes, Web 2.0 GUI, REST API, made in Germany, ...

# User interaction

- Both, Nagios/Icinga, and Puppet use plain text configuration languages which can easily be stored in version control systems
- User interaction via command line (CLI)
- Browser user interfaces (BUIs):
  - Puppet Enterprise version
  - Nagios/Icinga web interface provides status information, simple administrativetasks, read- only configuration
  - Commercial and Open Source Nagios configuration add-ons available

# Documentation for current versions

http://nagios.sourceforge.net/docs/3_0/toc.html

http://docs.icinga.org/latest/en/

http://docs.puppetlabs.com/puppet/3/reference/

# Test environment

```
$ ssh -i bremen2013.pem ubuntu@host[1-5].if2013.trish.de
$ sudo -s # to become root
```

# Puppet architecture

- **Master:**
  - takes care of set-up descriptions
  - runs as daemon, usually on port 8140
- **Agents:**
  - usually contact master to receive set-up instructions
  - ... and obey them locally
- **Communication:** always SSL encrypted
- **Resource Abstraction Layer (RAL):** provides independence from OS/distribution specifics

# Puppet (basic) commands

```
# puppet <subcommand> <options and args>
```

```
# [FACTERLIB=<path>] facter [-p] [<other options>][<fact>]
```

# Puppet command examples

```
# puppet help

# puppet help resource

# puppet resource user

$ facter -h

$ facter

$ facter puppetversion

# puppet master --verbose --no-daemonize --logdest /var/log/puppet/master.log

# puppet agent --server $(facter fqdn) --waitforcert 60 --onetime --logdest /var/log/puppet/
agent.log

# puppet cert sign host.example.com

# puppet master --genconfig | grep ssl

# puppet node clean host.example.com

# puppet module search icinga
```

# The puppet configuration description language

```
$ puppet resource user root
user { 'root':
  ensure  => 'present',
  comment => 'root',
  gid     => '0',
  home    => '/root',
  shell   => '/bin/bash',
  uid     => '0',
}
```

Caution: puppet resource translates existing set-up into description language

# Puppet manifests

```
/etc/puppet
|-- files
|-- manifests
|   |-- ...
|   `-- site.pp
|-- modules
|   |-- ...
...
|-- puppet.conf
...
`-- templates
```

# Puppet modules

```
|-- modules
|   |-- <module_name>
|   |   |-- files
|   |   |   `-- ...
|   |   |-- lib
|   |   |   |-- facter
|   |   |   |   `-- <fact>.rb
|   |   |   ...
|   |   |-- manifests
|   |   |   |-- <class>.pp
|   |   |   |-- <class>
|   |   |   |   |-- <subclass>.pp
|   |   |   ... `-- <subclass>.pp
|   |   `-- templates
|   |       |-- <filename>.erb
... ...     ...
```

# A simple configuration

/etc/puppet/manifests/site.pp:

```
node 'host.example.com' {
    # Icinga
    package { [ "apache2", "icinga" ] :
        ensure => installed,
    }
}
```

```
# puppet master --verbose --no-daemonize --logdest /var/log/puppet/master.log

# puppet agent --server $(facter fqdn) --waitforcert 60 --onetime --logdest /var/log/puppet/
agent.log

# puppet cert sign host.example.com # not necessary when agent and master on same host
```

# Parametrized classes I

/etc/puppet/manifests/site.pp:

```
node 'host.example.com' {
    package { [ "apache2", "icinga" ] :
        ensure => installed,
    }
    class {'icinga::web':
        install => 'present',
    }
}
```

# Parametrized classes II

/etc/puppet/modules/icinga/manifests/web.pp:

```
class icinga::web($install='absent'){
    package { "icinga-web" :
        ensure => $install,
    }
}
```

# Include definitions

`/etc/puppet/manifests/site.pp`:

```
node 'host.example.com' {
    include 'icinga::base'

    class {'icinga::web':
        install => 'present',
    }
}
```

# Facts and conditionals

`/etc/puppet/modules/icinga/manifests/base.pp`:

```
class icinga::base { package { [ "apache2", "icinga" ] : ensure => installed,}
    case $operatingsystem {
        debian: { service { "icinga":
                require => Package["icinga"],
                provider => debian,
                path => "/etc/init.d/",
                start => "service icinga start",
                stop => "service icinga stop",
                status => "service icinga status",
                ensure  => running,}}
        default: { service { "icinga":
                require => Package["icinga"],
                ensure  => running,}}}}
```

# Task

Make sure (using puppet) the package `nagios-plugins` is installed!

# Recycle the brainwork of others

https://forge.puppetlabs.com/

Install modules via

```
# puppet module install <author>-<name>
```

# Nagios architecture

- Nagios core:
  - provides monitoring logic and infrastructure
  - triggers (actively) checks on hosts to be monitored
  - may accept check results passively
  - takes care of check results (store, feed longterm monitoring tools, ...)
  - notifies according to configuration
- Web interface:
  - presents check results in traffic light manner
  - allows for configuration of scheduled downtimes
  - allows for manipulation of check and notification execution as well as of the nagios process (stop, restart)
  - provides read-only overview over check configuration
  - provides simple history, statistics, uptime reports (advanced SLA reports via add-ons, e.g. Jasper)
- Monitored hosts:
  - may host and run checks themselves (e.g. via ssh, Nagios Remote Plugin Executor (NRPE))

# Nagios checks

- Host checks:
  - Is the monitored host reachable (available on the net)?
  - states: OK, DOWN, UNREACHABLE
  - individually defineable, often a ping check
- Service checks:
  - Are run when host is reachable
  - Provide status information for whatever metric they monitor
  - states: OK, WARNING, CRITICAL, UNKNOWN

Checks are defined as command lines, usually executing dedicated programs, the plugins.

# Nagios plugins

- Can be written in any programming language
- Signal OK (0), WARNING (1), CRITICAL (2), UNKNOWN(3) with their return(exit) values (in parentheses).
- Return a free form single text line on STDOUT describing the state.
- Numerical data may be added as fixed form performance data following a pipe (|) sign.
- May return more verbose information in additional free form text lines on STDOUT.
- Usually offer CLI options -w/-c (thresholds), -v (verbose), -h (help), for remote checks -H (target host).

```
$ ./check_disk -w 60% -c 80% -p /
DISK CRITICAL - free space: / 33375 MB (15% inode=98%);| /=182848MB;91117;45558;0;227795
```

```
$ echo $?
2
```

# Recycle the brainwork of others

http://nagiosplugins.org/ (official plugins)

http://exchange.nagios.org/directory/Plugins

https://www.monitoringexchange.org/inventory/Check-Plugins

# Icinga/Nagios configuration

Main configuration file icinga.cfg/nagios.cfg (e.g. in /etc/icinga/, /etc/nagios/) defines where to look for check configurations :

```
# You can specify individual object config
# files as shown below:
#cfg_file=/usr/local/nagios/etc/objects/commands.cfg

# You can also tell Nagios to process all
# config files (with a .cfg
# extension) in a particular directory by
# using the cfg_dir directive as shown below:

cfg_dir=/usr/local/nagios/etc/global
cfg_dir=/usr/local/nagios/etc/sites
```

# Nagios objects (the most important ones)

- `host` : where to check
- `service` : what to check
- `command` : how to check (plugin execution call)
- `contact` : whom to notify (contact information, when, about which conditions)
- `contactgroup` : contact collections
- `servicedependency` : avoid notification on dependent services
- `serviceescalation , hostescalation` : extraordinary and/or special notifications (e.g. ticket system)
- `timeperiod` : when to check or notify

# command objects

`/usr/local/icinga/global/commands/check_disk.cfg`:

```
define command{
  command_name check_disk
  command_line $USER1$/check_disk -w $ARG1$ -c $ARG2$ -p $ARG3$
}
```

$USER1$ macro is defined in the file given as resource_file in main config file:

```
$ grep resource.cfg /etc/icinga/icinga.cfg
resource_file=/usr/local/icinga/etc/resource.cfg

$ cat /usr/local/icinga/etc/resource.cfg
...
# Nagios supports up to 32 $USERx$ macros ($USER1$ through $USER32$)
# Sets $USER1$ to be the path to the plugins
$USER1$=/usr/local/icinga/libexec
...
```

# host templates

Use templates to avoid duplicate definitions for similar purposes (prerequisite for puppet )

Minimal example (no notifications ):

`/usr/local/icinga/global/templates/host_generic-host_t.cfg`:

```
define host{
  name                    generic-host
  check_command           check-host-alive ; object name
  max_check_attempts          3
  check_period            24x7  ; object name
  notification_interval   30    ; minutes
  notification_period     24x7  ; object name
  register                0 ; This is a template
}
```

http://docs.icinga.org/latest/en/objectdefinitions.html#host

# service templates

Minimal example (no notifications ):

`/usr/local/icinga/global/templates/service_disk-root_t.cfg`:

```
define service{
  name                    disk-root
  service_description     Disk space usage on /
  check_command           check_disk!60%!80%!/
  max_check_attempts      3
  check_interval          5  ; minutes
  retry_interval          2  ; minutes
  check_period            24x7 ; object name
  notification_interval   30    ; minutes
  notification_period     24x7  ; object name
  register                0 ; This is a template
}
```

http://docs.icinga.org/latest/en/objectdefinitions.html#service

# Generating Nagios host definitions with Puppet

`/etc/puppet/modules/icinga/manifests/host.pp`:

```
class icinga::host {
  nagios_host { "$hostname":
     ensure =>  present,
     use =>     "generic-host",
     alias =>   $fqdn,
     address => $ipaddress,
     target => "/usr/local/icinga/sites/hosts/${hostname}.cfg"
  }
}
```

http://docs.puppetlabs.com/references/3.2.latest/type.html#nagioshost

# Generating Nagios service definitions with Puppet

`/etc/puppet/modules/icinga/manifests/disk_root.pp`:

```
class icinga::disk_root {
  nagios_service { "disk-root-on-$hostname":
     ensure =>  present,
     use =>     "disk-root",
     host_name => $hostname,
     target => "/usr/local/icinga/sites/hosts/${hostname}.cfg"
  }
}
```

Caution : resource titles must be unique

http ://docs.puppetlabs .com/references/3.2.latest/type .html#nagiosservice

# Task

Make sure puppet generates the Nagios host and disk check objects for your training host.

Reload your icinga service:

```
# service icinga reload
```

... and make sure your check is being run (http://localhost/icinga)