Supervision - Monitoring

Lucas Nussbaum
lucas.nussbaum@univ-lorraine.fr

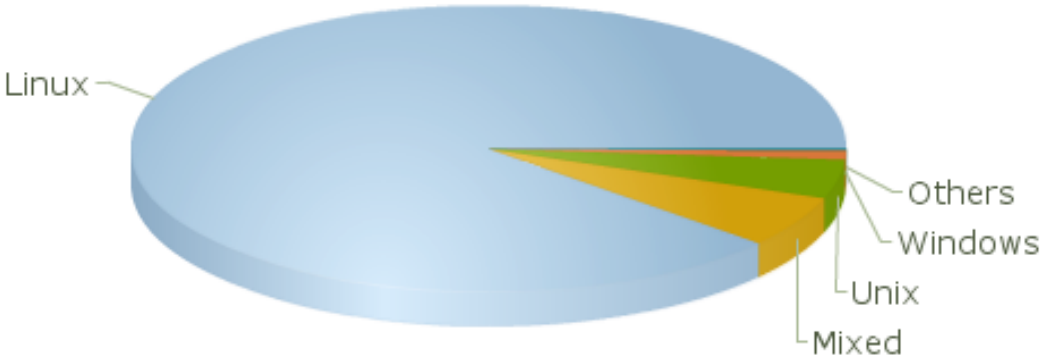
Licence professionnelle ASRALL

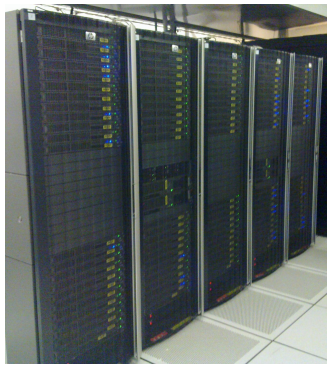
Administration de systèmes, réseaux et applications à base de logiciels libres

https://members.loria.fr/lnussbaum/asrall.html

Lucas Nussbaum

- Enseignant-chercheur (Maître de conférences) à l'univ. de Lorraine
  Actuellement en détachement à l'Inria
- En ASRALL:
  - ~~Administration Système (septembre - novembre)~~
  - Outils libres ; Supervision ~~; Projets tutorés (novembre - mars)~~
  - Virtualisation ; Gestion d'infrastructures (janvier - mars)
  - ~~Responsable des projets tuteurés (janvier - mars)~~
  - ~~Direction des études~~
- Recherche: Systèmes distribués, calcul à haute performance, cloud





- Logiciel libre:
  - Développeur Debian (et Ubuntu, par le passé)
    Debian Project Leader de 2013 à 2015, *Quality Assurance*, Ruby
  - Quelques développements en Ruby
    feed2imap, xmpp4r, tuns, etc.

Administrative stuff

- Yes, this course is in English
  - I will speak in French though
  - Goal: get you used to reading technical documentation in English
- This module:
  - 6 slots of 3 hours
  - Evaluation: practical work (*TPs*) + possibly exam
  - Goals:
    - General knowledge of *infrastructure monitoring*
    - Master standard tools of the field
    - Know about the current trends in this field (e.g. impact of cloud and elasticity)
- The other part of this module (*Supervision - Annuaire*) is totally independent (and with a different tutor: *Fabien Pascale*)
- Warning about plagiarism

Introduction

- Success criteria for sysadmins: infrastructure that *just works*
  - Avoid incidents if possible
  - If not possible, minimize downtime
- How? Well-designed infrastructure
  - Choose reliable technologies and software
  - Add HA (*high-availability*), failover, redundancy, etc.
- Not enough: Murphy's law (*Anything that can go wrong will go wrong*)
- **Monitoring**:
  - Collect information about the state of the infrastructure
  - Detect problems (before users have to report them)
  - Predict problems

  $\rightsquigarrow$ Usual components:
  - Probes to acquire data
  - Database to store all measurements
  - Dashboard to show results
  - Notification system (email, SMS, etc.)

Example: Icinga

242 UP | 2 / 0 / 0 DOWN | 0 / 0 / 0 UNREACHABLE | 0 PENDING | 2 / 244 TOTAL
16934 OK | 4 / 0 / 0 WARNING | 5 / 0 / 93 CRITICAL | 0 / 0 / 0 UNKNOWN | 0 PENDING | 102 / 17036 TOTAL

ICINGA

244 / 0 / 0 — 17036 / 0 / 0
0.02 / 3.24 / 0.156 s — 0.01 / 50.19 / 0.482 s
0.29 / 0.68 / 0.496 s — 0.00 / 0.83 / 0.227 s

General
- Home
- Docs www
- Search:

Status
- Tactical Overview
- Host Detail
- Service Detail
- Hostgroup Overview
- Hostgroup Summary
- Servicegroup Overview
- Servicegroup Summary
- Status Map

Problems
- Service Problems
- Unhandled Services
- Host Problems
- Unhandled Hosts
- All Unhandled Problems
- All Problems
- Network Outages

System
- Comments
- Downtime
- Process Info
- Performance Info
- Scheduling Queue

Reporting
- Trends
- Availability
- Alert Histogram
- Alert History
- Alert Summary
- Notifications
- Event Log

Configuration
- View Config

Tactical Monitoring Overview
Last Updated: Wed Oct 31 11:14:36 UTC 2018 - Update in 8 seconds [pause]
Icinga Classic UI 1.13.4 (Backend 1.13.4) - Logged in as *dsa-guest*

Network Outages
0 Outages

Network Health
Host Health:
Service Health:

Hosts

2 Down	0 Unreachable	242 Up	0 Pending
Unhandled Problems 2 Active			

Services

98 Critical	4 Warning	0 Unknown	16934 Ok	0 Pending
Unhandled Problems 5 Active	Unhandled Problems 4 Active			
on Problem Hosts Unacknowledged 93 Active				

Host Checks

Active		Passive	
Enabled	244 Enabled	Enabled	No Passive Checks

Service Checks

Active		Passive	
Enabled	17036 Enabled	Enabled	No Passive Checks

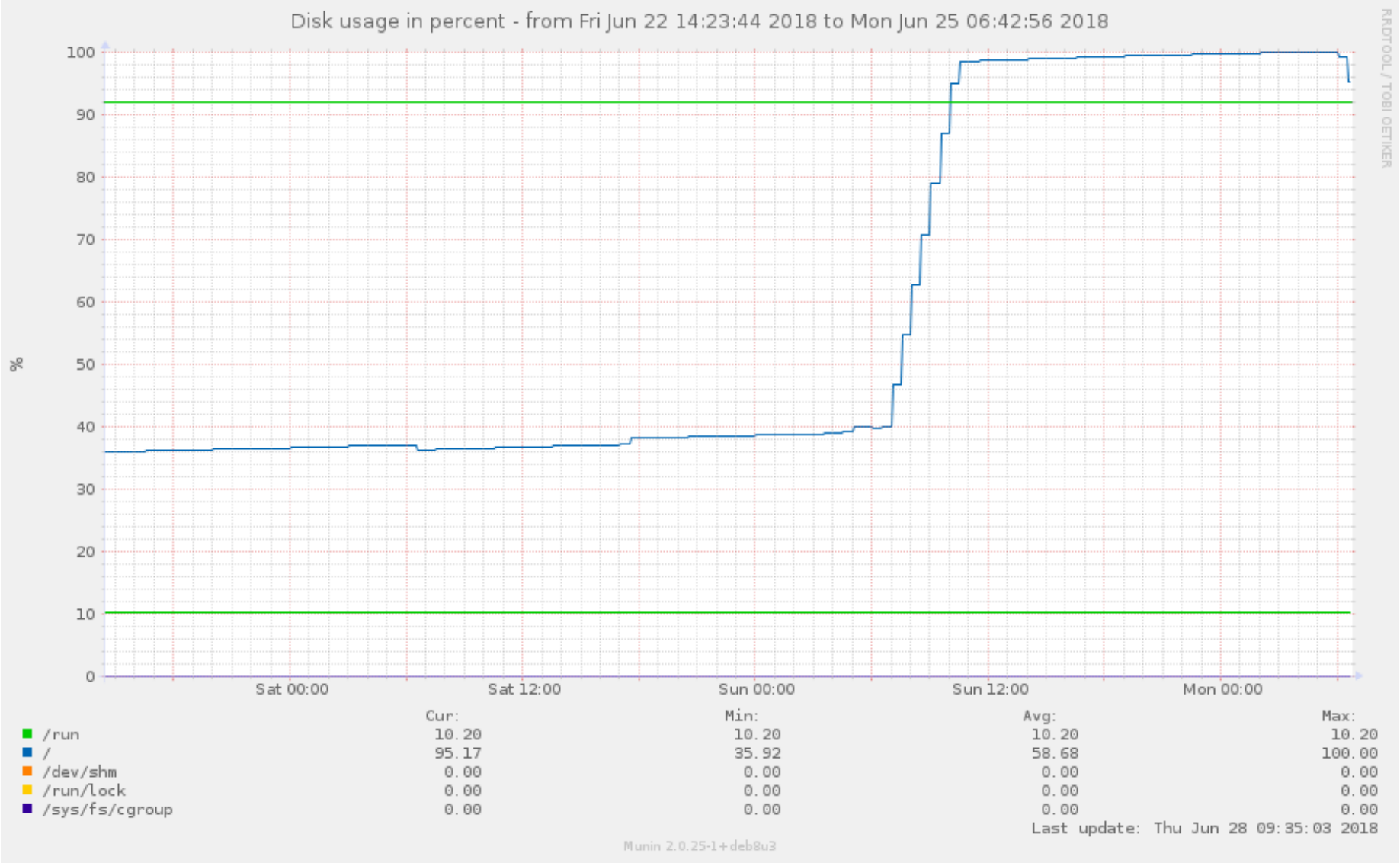
Monitoring Features

Flap Detection		Notifications		Event Handlers	
Enabled	All Services Enabled 4 Services Flapping All Hosts Enabled 4 Hosts Flapping	Enabled	All Services Enabled All Hosts Enabled	Enabled	All Services Enabled All Hosts Enabled

`https://nagios.debian.org/icinga/` – login: dsa-guest / password: dsa-guest

Example: graph from Munin

- Disk usage on a server



Two sides of the same coin: Metrology

Goal: collect lots of metrics about how the system behaves to track performance of the system over time $\rightsquigarrow$ *telemetry*

- Example: collect statistics about network traffic, HTTP req/s, disk I/Os, ...
- Required for performance analysis
  - USE method
    = When there's a performance problem, check for each resource: Utilisation, Saturation, Errors
  - Also related to the RED method
    = When there's a performance problem, check for each service: Rate (number of req/s), Errors (number of failing req/s), Duration
  - Also related to the *four golden signals* (Latency, Traffic, Errors, Saturation)

$\rightsquigarrow$ You need to keep track of all those metrics to be able to answer *is this the expected/usual value?*

Two sides of the same coin: Monitoring

Goal: actively check the system to verify that it works as expected, and to detect problems before users report them

- Example: issue a request to a web server, to check that it answers correctly
- Usually service-oriented, not really resource-oriented
  - Sometimes active checks that a resource' value are within limits

Terminology

- There's some disagreement about the terminology
  - Supervision = Metrology + Monitoring ?
  - Supervision = Monitoring ?
  - Monitoring = Supervision + Metrology ?
- The names are not very important, but it's important to know the difference because software solutions usually focus on one of the two sides
  - Metrology: **Munin**, Cacti, Ganglia, collectd, Zabbix, ...
  - Monitoring: **Nagios**, check_MK, Icinga, Shinken, ...

Supervision is related to other topics

... which are not covered in this course

- Systems inventory / IT assets management
  - Database of all servers, network equipment, etc.
  - Example setup: GLPI (*Gestion Libre de Parc Informatique*), with a FusionInventory agent running on systems (to be seen in *Outils Libres côté serveur*)
  - Another example: LibreNMS (auto-discovery and monitoring of network equipment)
- Configuration Management Database (CMDB), services directory
  - Database about an information system: services, and how they are deployed
  - Typical solutions: Consul, iTop
  - Can be used as a source for the monitoring system (to get the list of services to monitor)
- Tools to manage (configure) the infrastructure: Puppet, Ansible, etc.
  - To be seen in *Gestion d'infrastructures*

Other aspects of monitoring

... also not covered in this course

- Security-oriented monitoring
  - Intrusion Detection Systems (IDS)
    - Examples: Suricata (network), chkrootkit (systems)
- Log files analysis
  - Gather all logs in a central place and analyze them
  - Typical solutions:
    - Syslog (see *Exposé technique*)
    - logcheck
    - Graylog
    - Splunk
    - ELK (Logstash → ElasticSearch → Kibana)

Outline of this course

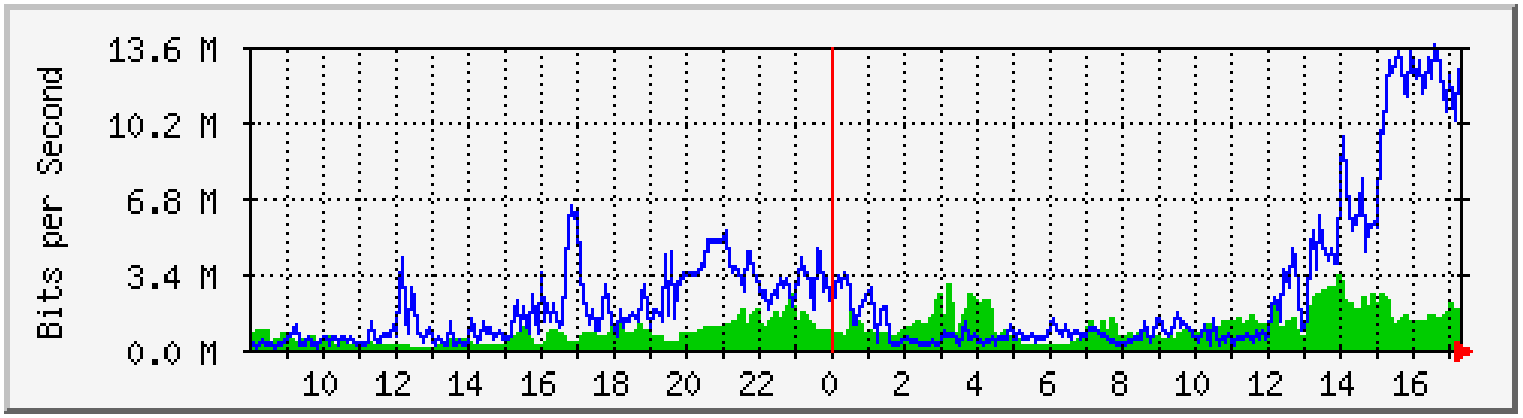
1. Introduction
2. Metrology & Munin
3. Monitoring with Icinga
4. SNMP

Metrology & Munin

- Reminder: see slide 8

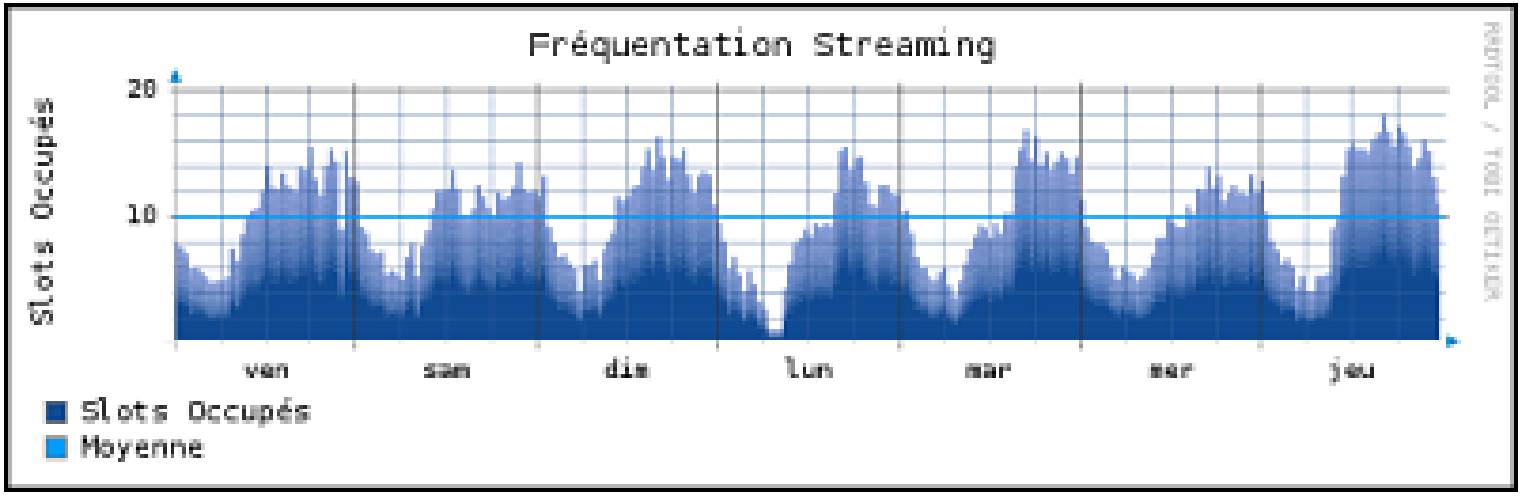
Historical root: MRTG

- https://oss.oetiker.ch/mrtg/
- Multi Router Traffic Grapher
- Developed by Tobias Oetiker, first version in 1995
- Focus on monitoring network devices using SNMP



Common foundation: RRDTOOL

- 1999: Tobias Oetiker started the development of RRDTOOL
- `https://oss.oetiker.ch/rrdtool/`
- Much more flexible than MRTG
- Used for data storage and graphing by many metrology solutions: Cacti, collectd, Munin, Smokeping, ...



Landscape of metrology solutions

- Munin – `http://munin-monitoring.org/`
  - Implemented in Perl
  - Simple architecture, configured through configuration files
  - Easy to extend through plugins
- Cacti – `https://www.cacti.net/`
  - Implemented in PHP
  - More focused on monitoring network traffic
  - Complex application; configuration using web interface; multi-user
- collectd – `https://collectd.org/`
  - Implemented in C $\rightsquigarrow$ fast $\rightsquigarrow$ popular in low-power devices
  - Many plugins
  - Does not include a web frontend, but third-party projects do
- Ganglia – `http://ganglia.info/`
  - Focused on monitoring clusters of servers
    - Aggregate metrics for groups of servers
    - Originally from the High Performance Computing world
- Zabbix – `https://www.zabbix.com/`
  - Integrated, full-featured solution. Includes monitoring features.

Landscape of metrology solutions (2)

- Graphite – `https://graphiteapp.org/`
  - Not based on RRDTOOL
  - Does not handle data collection, only storage and graphing
  - Can be used with collectd (for collection), or many other tools
- Prometheus – `https://prometheus.io/`
  - Modern solution, suited to elastic infrastructures
  - Dynamic aggregation of metrics
  - Alerting based on metrics
  - Videos:
    - ★ Prometheus - A Next Generation Monitoring System (FOSDEM 2016)
    - ★ Alerting with Time Series (FOSDEM 2017)
    - ★ Deploying Prometheus at Wikimedia Foundation (FOSDEM 2017)
    - ★ Evolving Prometheus for the Cloud Native World (FOSDEM 2018)
- Summary of the current status:
  - Prometheus looks like the future (but is still a young project)
  - Traditional solutions are still very much used in production, especially on traditional infrastructures $\rightsquigarrow$ focus on Munin in this course

Munin

- `http://munin-monitoring.org/`
  - Documentation: `http://guide.munin-monitoring.org`
- Implemented in Perl
- Simple and robust
- Configured through configuration files (`/etc/munin/`)
- Easy to extend through plugins (in any language)
- Generates static pages and graphs, no access control
  - Graphs can also be generated using a CGI script
- Basic alerting (warning/critical thresholds)
- Video: Alexandre Simon, Luc Didry. Munin : superviser simplement la machine à café ... enfin une realité ! (JRES 2013)
- Example instances:
  - Debian: `https://munin.debian.org/` (dsa-guest / dsa-guest)
  - Framasoft: munin data exported to Grafana, with many custom plugins (see blog from Luc Didry) : link

Munin web interface

Interface graphique

Page d'accueil

Barre de navigation persistante

Données et graphes

Groupes de nœuds

Les nœuds supervisés

Seuils en alerte

Période d'affichage

MUNIN Overview

Problems
- Critical (0)
- Warning (0)
- Unknown (0)

Groups
- munin-monitoring.org

Categories
- 1sec::network [d w m y]
- 1sec::system [d w m y]
- active_inactive [d w m y]
- apache [d w m y]
- cache [d w m y]
- candidates [d w m y]
- direct_map [d w m y]
- disk [d w m y]
- exim [d w m y]
- free [d w m y]
- gimbal [d w m y]
- http [d w m y]

- munin-monitoring.org :: [day week month year]
  - buildd.munin-monitoring.org [disk gimbal http network processes system]
  - demo.munin-monitoring.org [1sec::network 1sec::system apache disk logins memory munin network network:services network:traffic processes system time]
    - slab [reclaim]
  - www.munin-monitoring.org [apache disk exim network other postgresql processes system time varnish]

Catégories de supervision

http://reseau.ciril.fr

source: talk at JRES 2013 (see before)

Munin web interface (2)

Interface graphique

Détails d'un nœud

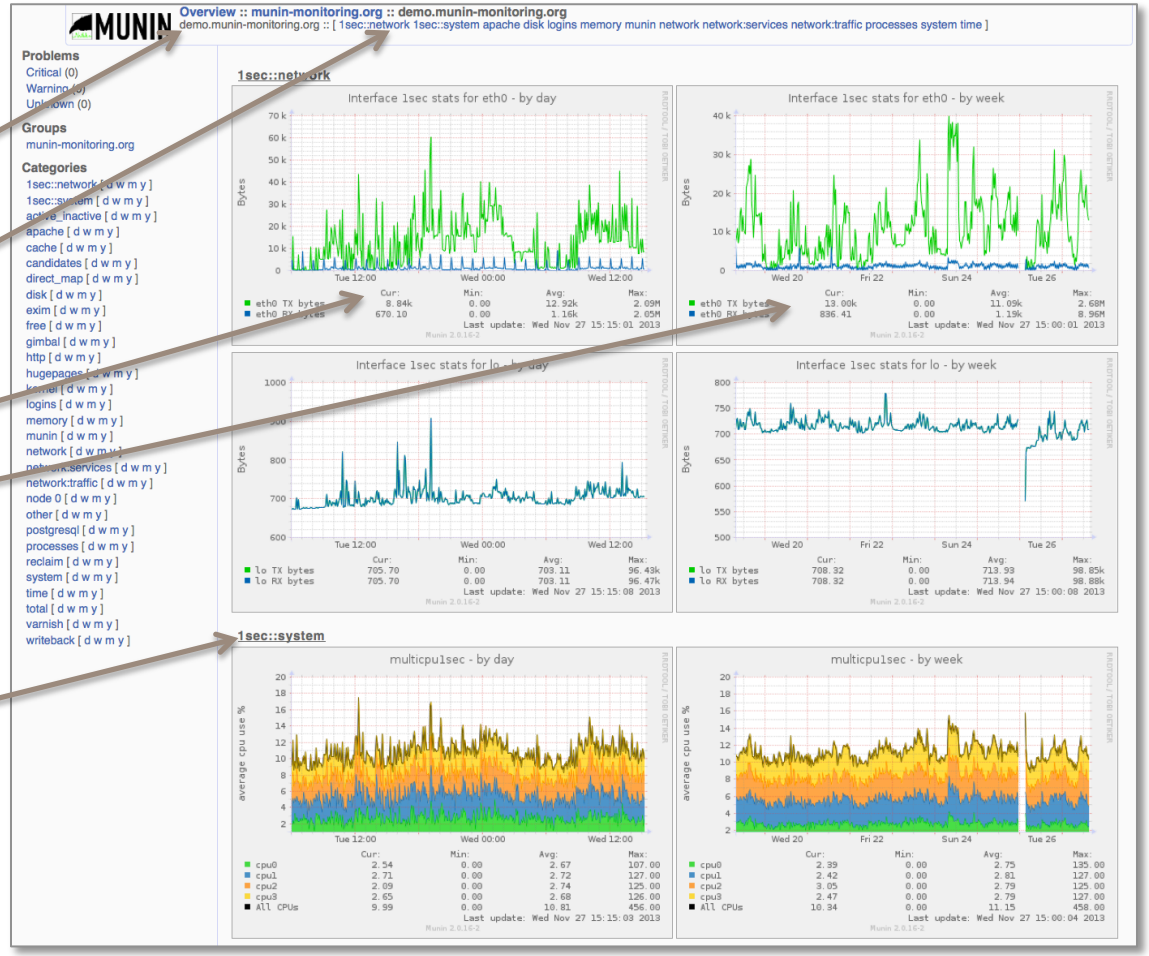
Rappel du nom du nœud

Accès direct aux catégories de ce nœud

Période de 24h

Période de 7j

Catégories des graphes suivants



http://reseau.ciril.fr

source: talk at JRES 2013 (see before)

Munin web interface (3)

Interface graphique

Détails d'un graph
aka*. un *plugin

Rappel du nom du nœud

Rappel du nom du *plugin*

Période de 24h

Période de 7j

Période de 1m

Période de 1y

Métriques supervisées

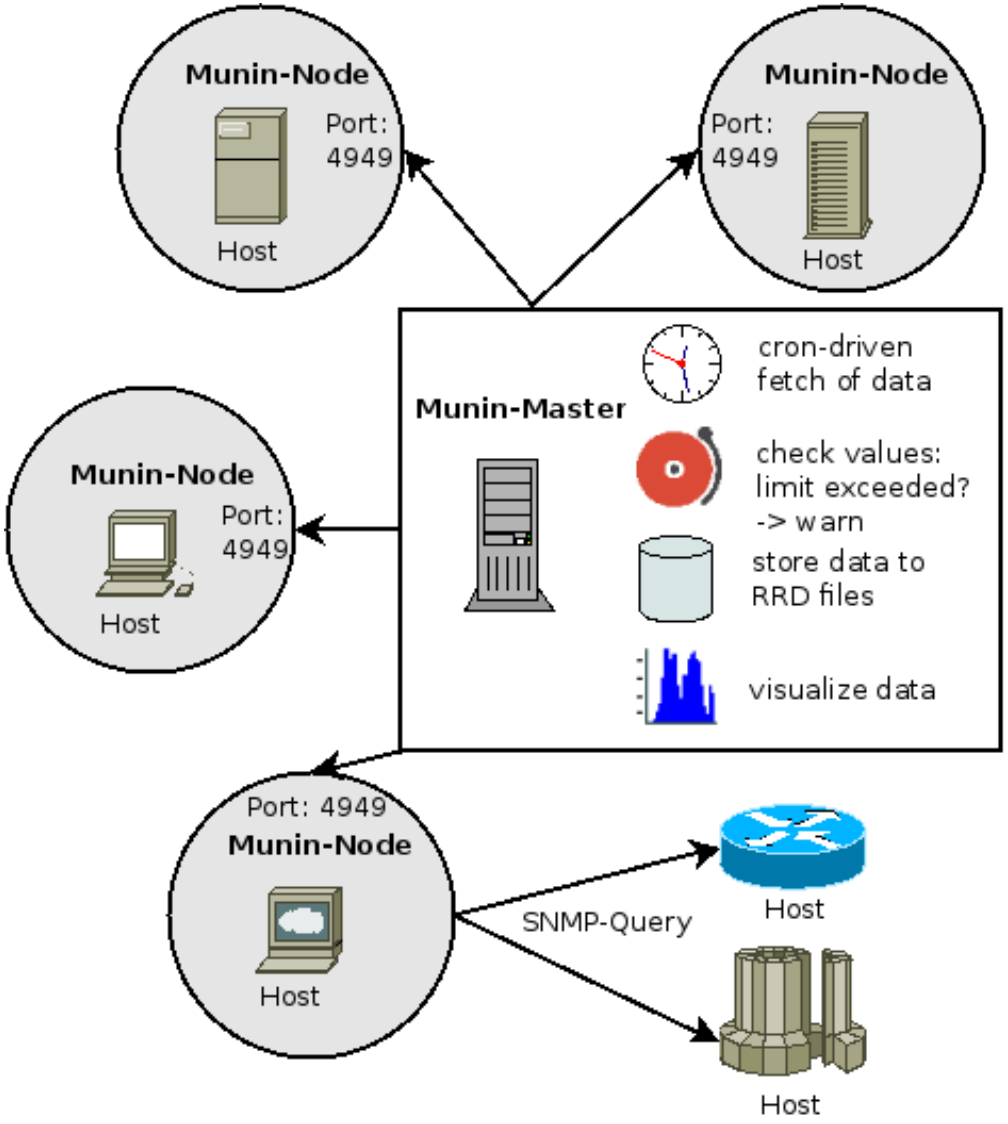
Seuils d'alerte

Démo de l'interface disponible sur :
http://demo.munin-monitoring.org/

http://reseau.ciril.fr

source: talk at JRES 2013 (see before)

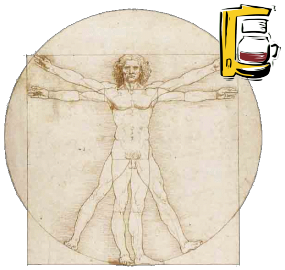
Munin architecture



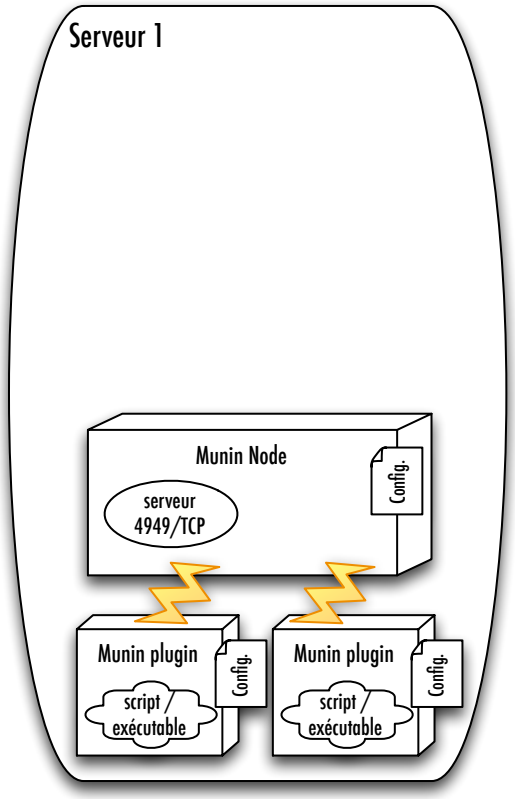
source: `http://guide.munin-monitoring.org/en/master/architecture/index.html`

Munin architecture (2)

Architecture



- Munin *node*
  - sur les machines à superviser
  - serveur écoutant sur 4949/TCP
  - exécute des *plugins* à la demande du maître



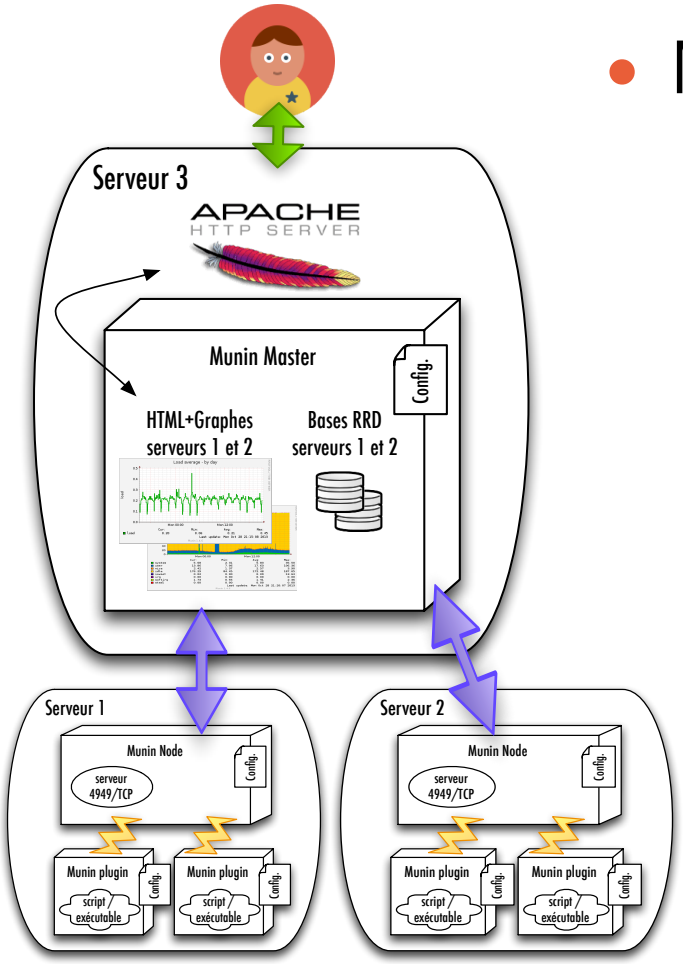
Exécution locale

http://reseau.ciril.fr

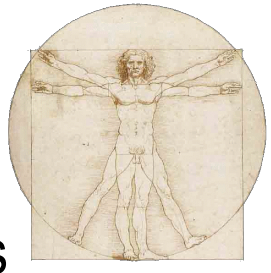
source: talk at JRES 2013 (see before)

Munin architecture (3)

Architecture



- Munin *master*
  - le “superviseur”
  - instance surveillant 1 ou plusieurs nœuds
  - client se connectant sur le 4949/TCP des nœuds, fait exécuter les *plugins*
  - archive la collecte et génère les graphes
  - publie les pages HTML et les graphes



source: talk at JRES 2013 (see before)

Munin installation notes

- On nodes: `apt install munin-node`
  - By default, it only allows the local machine to connect
- On the master: `apt install munin`
  - Munin runs in a cron job, see `/etc/cron.d/munin`
  - Target nodes must be added to `/etc/munin/munin.conf` or `/etc/munin/munin.conf.d/*`
  - Install a web server to expose files generated by Munin. Munin provides an example configuration file for Apache, so:
    - `apt install apache2`
    - `ln -s /etc/munin/apache24.conf /etc/apache2/conf-enabled/`
    - Allow all addresses to connect: in `/etc/munin/apache24.conf`, change `require local` to `require all granted`
    - `service apache2 restart`
- Additional plugins are configured in `/etc/munin/plugins`
  - To get suggestions about which plugins to configure: `munin-node-configure --suggest` (or `--shell`)
  - Restart munin-node when its configuration is changed

Troubleshooting

- Logs are in `/var/log/munin/` (for example: `/var/log/munin/munin-update.log` for the log describing what happens when the munin master tries to contact munin nodes)
- You can run the cron job manually using `sudo -u munin munin-cron`
  - You can force the generation of HTML pages using: `sudo -u munin /usr/share/munin/munin-html --debug`
  - You can force the generation of graphs using: `sudo -u munin /usr/share/munin/munin-graph --debug --nofork --nolazy`
- You can debug munin-node by connecting to it using `telnet` on port 4949. Just send junk to get the list of commands.
  - It is a good way to check whether the munin master can talk to munin nodes.
- The default update rate is 5 minutes. You can reduce it to one minute:
  - Add `update_rate 60` to `/etc/munin/munin.conf`
  - Change `/etc/cron.d/munin` so that it runs every minute
  - Remove all files in `/var/lib/munin/`

Writing Munin plugins

- Simple scripts, to be placed in `/etc/munin/plugins`
- With the `config` parameter, must display the configuration:

```
# /etc/munin/plugins/threads config
graph_title Number of threads
graph_vlabel number of threads
graph_category processes
graph_info This graph shows the number of threads.
threads.label threads
threads.info The current number of threads.
```

- Without parameter, must display the value:

```
# /etc/munin/plugins/threads
threads.value 112
```

- Documentation: `http://guide.munin-monitoring.org/en/latest/develop/plugins/howto-write-plugins.html`
  - Also see the video from JRES 2013

TP 1

Monitoring & Icinga

- Reminder: see slide 9

Side note: Open Core model

- Principle:
  - The central piece of the software is free software
  - Some features are for paying customers only
- Variant:
  - The software is free, but new versions (or even worse: security updates) are available first to paying customers
- Example (outside of monitoring): GitLab
  - Integrated development environment
    - $\approx$ *libre GitHub*: Git + bug tracker + CI + ...
  - *community* version free (as in freedom and as in beer)
  - Other versions available for a fee, with added features

Common ancestor: Nagios

- Initiated in 1999 (under the *NetSaint* name)
- Around 2009: dissatisfaction in the community about the project management $\rightsquigarrow$ forks (Icinga, etc.)
- Now developed by Nagios Enterprises using an Open Core model
  - The core software is now named *Nagios Core*
- Features:
  - Monitoring of hosts
  - Monitoring of hosts resources (but not as detailed as Munin, and without good history by default)
  - Monitoring of services
  - Many plugins available
  - Ability to run remote scripts using NRPE (*Nagios Remote Plugin Executor*)

Landscape of Monitoring solutions

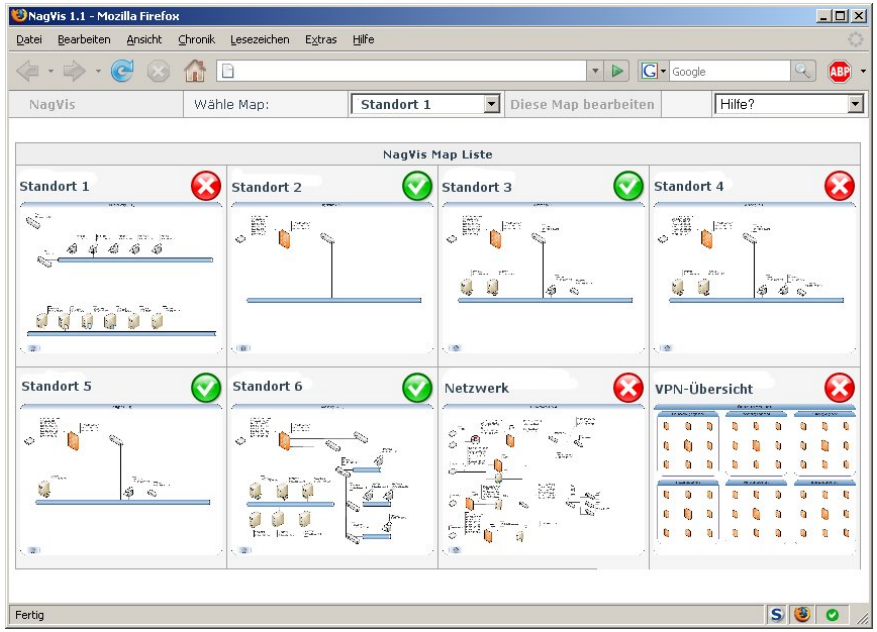
- Icinga – `https://icinga.com/`
  - Forked from Nagios in 2009
  - Version 1 was strongly based on Nagios
    - See those slides from RMLL 2011 about Icinga
  - Version 2 is a rewrite
    - See this video about Icinga 2
- Check_MK – `https://mathias-kettner.com/`
  - Initially started as a Nagios extension to improve scalability
  - Now a standalone monitoring solution
- Shinken – `http://www.shinken-monitoring.org/`
  - Rewrite of Nagios in Python
  - See this talk from RMLL 2010 for an introduction to Shinken
  - Dead project?

Landscape of Monitoring solutions (2)

- Fully Automated Nagios – `http://www.fullyautomatednagios.org/`
  - Appliance – useful in mixed Windows/Linux environments, or when the monitoring system is the only Linux system in the infrastructure
- Centreon – `https://www.centreon.com/`
  - Another appliance solution
  - French company – popular in French infrastructures
  - Open Core

Landscape of Monitoring solutions (3)

- Nagvis – `http://www.nagvis.org/`
  - Nagios addon to provide graphical representations using the hierarchy of hosts and services



Current status of Monitoring

- Many Nagios-like solutions
  - Some of them are forks of Nagios, some of them are rewrites
  - Some of them are Open Core
  - Not clear which one will win, especially since many of them are company-backed
- Focus in this course: Icinga 1.x
  - Nagios-compatible, but still better than Nagios
    - Note: Icinga 2.x is very different from Icinga 1.x
  - In terms of spirit, it is fairly Free software
  - Concepts of all those solutions are similar anyway
- Also, important to keep an eye on modern solutions such as Prometheus

Icinga 1.x

- Most of what you can read about Nagios is true for Icinga 1.x
- Icinga documentation: `https://icinga.com/docs/icinga1/latest/en/`
- Components:
  - *Icinga Core* – the core software
  - Icinga classic Web UI is a CGI-based interface
    - It reads the status of Icinga from `/var/lib/icinga/status.dat`
    - It can submit changes to the core using `/var/lib/icinga/rw/`
  - *IDODB* (Icinga Data Out Database) is a database used to store results. It is required for the new Web UI (not used in this course)
  - Plugins (for checks) are in `/usr/lib/nagios/plugins/` (from the `monitoring-plugins-*` packages)
    - See `/etc/nagios-plugins/config/` for their command definitions
  - NRPE (*Nagios Remote PLugin Execution*) is used to run checks on remote hosts
    - `nagios-nrpe-plugin` must be installed on the *master* node, and `nagios-nrpe-server` must be installed on the *slave* nodes
  - NSCA (*Nagios Service Check Acceptor*) is used to receive data for passive checks

More information about Nagios & Icinga

- MOOC *Supervision de Réseaux et Services*
  - `https://www.fun-mooc.fr/courses/course-v1:lorraine+30008+session02/about`
- Videos available from `https://videos.univ-lorraine.fr/index.php?act=view&id_col=404`
- Specifically *Semaine 2* (total of 45 minutes)

Icinga installation

- Install Icinga: `apt-get install icinga`
  - Enable external commands (used to trigger actions from the web interface)
- The classic Web UI should be available at `http://ops.local/icinga/` in the Vagrant environment
  - By default, it monitors the local machine

Icinga configuration

- In `/etc/icinga/`
- Object-oriented: inheritance, classes (groups) of hosts/services/contacts
- Main objects: hosts, services
- Basic example:

```
define host {
  use        generic-host ; generic-host is a template
  host_name  web.local
  alias      web
}

define service {
  use generic-service ; generic-service is a template
  host_name web.local ; host(s) for this service
  service_description HTTP
  check_command check_http
}
```

Icinga configuration (2)

- Using hostgroups to group similar hosts

```
define hostgroup {
  hostgroup_name http-servers
  alias           HTTP servers
  members localhost,web.local
}
define service {
   hostgroup_name http-servers
   service_description HTTP
   check_command       check_http
   use                 generic-service
}
```

Icinga configuration (3)

- Check commands can use parameters
- For example `check_http2` (with VHost, warning delay, critical delay) (see `https://www.monitoring-plugins.org/doc/man/check_http.html`)
- In `/etc/nagios-plugins/config/http.cfg`:

```
define command {
  command_name check_http2
  command_line /usr/lib/nagios/plugins/check_http -H '
      $ARG1$' -I '$HOSTADDRESS$' -w '$ARG2$' -c '$ARG3$' '
      $ARG4$'
}
```

- Service definition:

```
define service {
   use                  generic-service
   host_name            webserver
   service_description  HTTP
   check_command        check_http2!www.myvhost.com!5!10
}
```

Icinga configuration: other objects

- Host, Hostgroup
- Service, Servicegroup
- Contact, Contactgroup
- Timeperiod
- Command
- Servicedependency
- Serviceescalation
- Hostdependency
- Hostescalation
- Hostextinfo
- Serviceextinfo
- Module

`https://icinga.com/docs/icinga1/latest/en/objectdefinitions.html`

NRPE (*Nagios Remote PLugin Execution*)

- Problem: all checks are run from the Icinga machine
  - How can we test things that are not visible from a remote machine?
- Server running on remote hosts that answers requests from `check_nrpe`
- Installation:
  - On the Icinga machine: `nagios-nrpe-plugin` (provides `check_nrpe`)
  - On the target machine: `nagios-nrpe-server`
    - By default, only 127.0.0.1 is allowed to connect
- Documentation:
  `https://icinga.com/docs/icinga1/latest/en/nrpe.html`

NRPE Configuration

- On the target host, in `/etc/nagios/nrpe*`
  - Command definition example:
    `command[check_users]=/usr/lib/nagios/plugins/check_users -w 5 -c 10`
- On the Icinga machine:

```
define service {
  use generic-service
  host_name web.local
  service_description Current users
  check_command check_nrpe!check_users
}
```

Writing Icinga plugins

- Simple scripts or compiled programs
- Documentation:
  `https://icinga.com/docs/icinga1/latest/en/pluginapi.html`
- Return code: 0 (OK), 1 (WARNING), 2 (CRITICAL), 3 (UNKNOWN)
- Output: see documentation. Example:
  `DISK OK - free space: / 3326 MB (56%)`

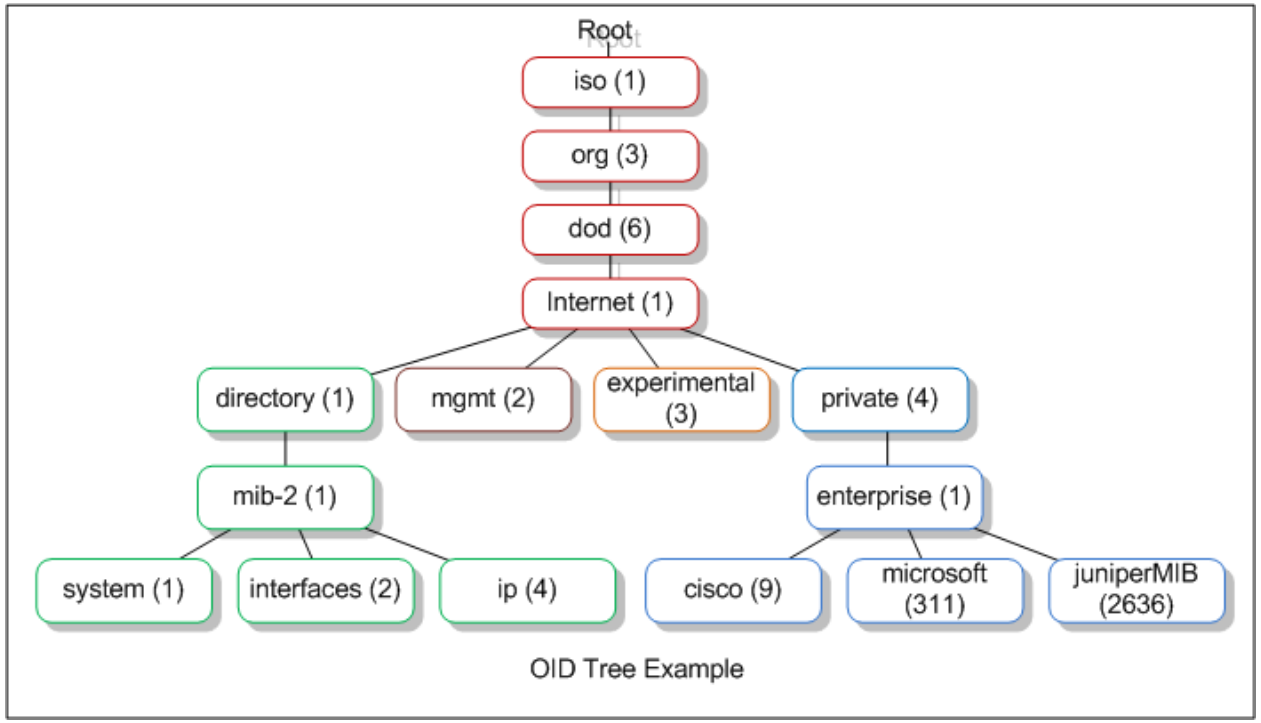
TP 2

SNMP

- Simple **Network Management** Protocol
- Three use cases (the first one is the only used frequently in practice):
  - Get information from a device: the manager sends a request to the agent running on the managed device
  - Configure a device (SNMP is sometimes a good way to automate things in scripts)
  - Receive traps from the agent: typically, get notified when errors happen on the managed device
- Typical compatible devices: routers, switches, printers, servers management cards (BMC, Baseboard Management Controller)
  - SNMP = the standard protocol to retrieve information from those devices
  - Also possible to install an SNMP agent on a server, but rarely done

MIB: Management Information Base

- Hierarchy of Managed Objects (MO)
- Each MO is identified by an Object Identifier (OID)
- Example: .1.3.6.1.2.1.1.3.0 = DISMAN-EVENT-MIB::sysUpTimeInstance
- Some parts are standard, some are vendor-specific
- Sub-trees are defined in additional MIBs



source: Wikipedia

Basic use

- `snmpwalk -v 1 -c public remoteagent [OID]`
  - `-v 1` is the SNMP version to use (usually 1 or 2c)
  - `-c public` is the SNMP *community* to use. This is used similarly to a password
- Other useful options:
  - `-m ALL`: load all MIBs, not just the default set
  - `-O n`: display OIDs numerically
  - `-C c`: disable check for increasing OIDs
- Other useful command: `snmpget` (get a single OID)
- Examples:

```
snmpwalk -v 2c -c public asrall-sw24 IF-MIB::ifMtu
snmpget -v 2c -c public asrall-sw24 IF-MIB::ifMtu.1
```

Installing additional MIBs on Debian

- Make sure that the APT source for `non-free` is enabled
- Install the `snmp-mibs-downloader` package
- Comment the last line of `/etc/snmp/snmp.conf`

TP 3

Conclusions

- Monitoring is a critical element of any infrastructure
  - To detect problems before users
  - To prevent problems before they occur
- Two components:
  - Metrology: collect historical data about the infrastructure
    $\rightsquigarrow$ understand trends
  - Monitoring: check that the infrastructure works as expected
- Useful to have even if the rest of the infrastructure is not automated
  - But even better when you combine it with systems inventories, services directories, configuration management