Introduction to NCM
Configuration components

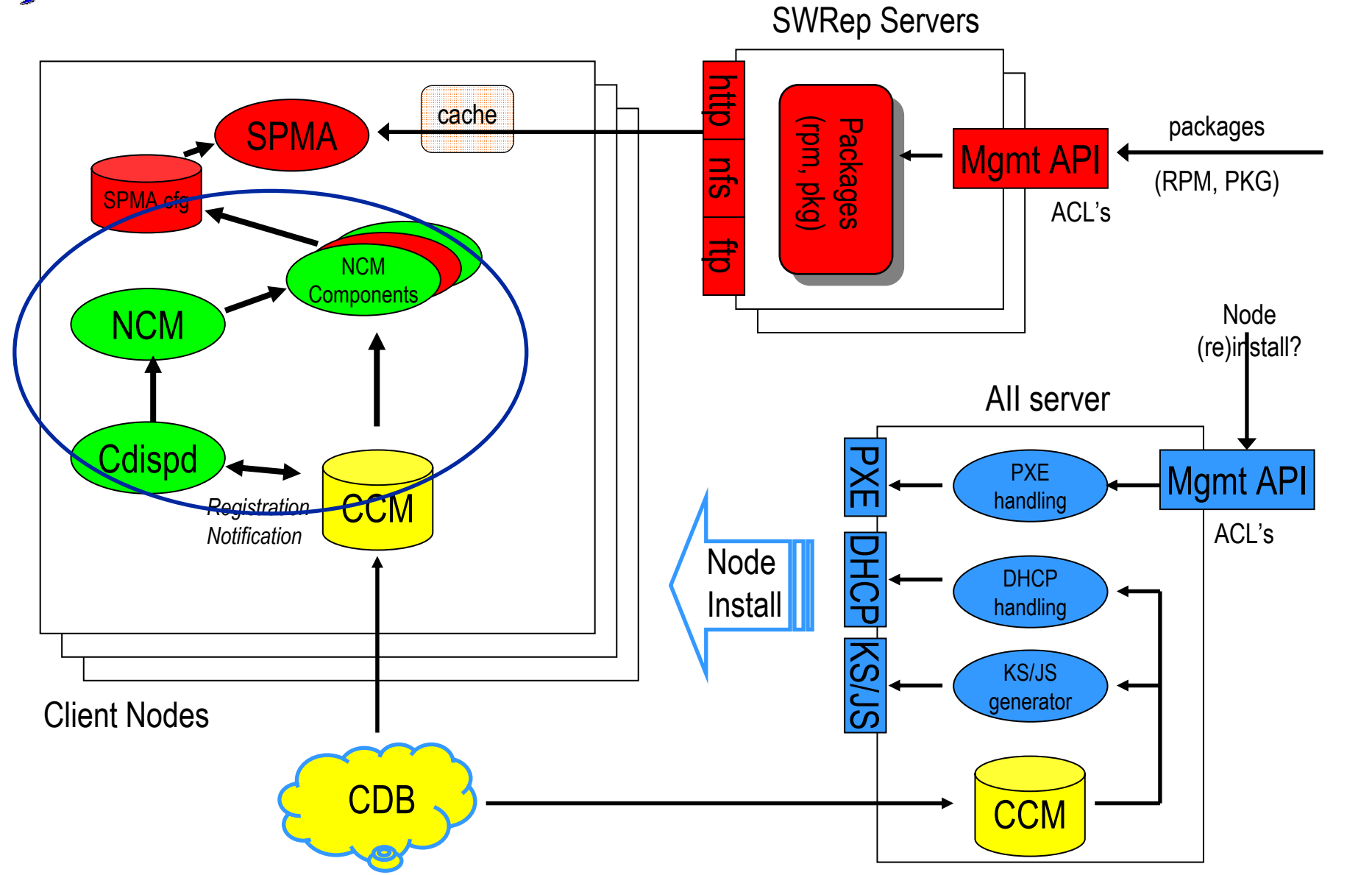
German Cancio

CERN/IT

Contents

- NCM ‘theory’
  - Components – what are they, how to run them
- Some example components
- Exercise: configure components
- How to write components
- Exercise: modify an existing component

NCM environment



What are components? (1/2)

- "Components" (like SUE "features" or LCFG 'objects') are responsible for updating local config files, and notifying services if needed
- Components do only *configure* the system (unlike LCFG!)
  - Usually, this implies regenerating and/or updating local config files (eg. `/etc/sshd_config`)
- Use standard system facilities (SysV scripts) for *managing* services
  - Components can notify services using SysV scripts when their configuration changes.
- Components can be run
  - Manually (via `ncm-ncd`)
  - via hooks (cron, boot time, etc)
  - automatically*:* register their interest in configuration entries or subtrees, and get invoked in case of changes (via `ncm-cdispd`)
- Possible to define configuration dependencies between components
  - Eg. configure *SPMA* before *GRUB*
  - Components won't run if a pre-dependency is *unsatisfied* (eg. failing prerequisite component)

What are components? (2/2)

- Components are written as Perl OO class instances
  - But don't worry, no OO knowledge needed for writing them, just some Perl.
- Each component is packaged as an individual RPM.
- Each component can provide two methods:
- **`Configure():`**
  - invoked when there was a CDB configuration change or on startup
  - *Mandatory method*
- **`Unconfigure():`**
  - invoked when a component is to be removed
  - *Optional method* – most of the components won't need to implement it.

Component (simplified) example


```
sub Configure {
  my ($self,$config) = @_;
  # 1. access configuration information
  my $arch=$config->getValue('/system/architecture'); # NVA API
  $self->Fail ("not supported") unless ($arch eq 'i386');
  # 2. (re)generate and/or update local config file(s)
  open (myconfig,'/etc/myconfig'); …
  # 3. notify affected (SysV) services if required
  if ($changed) {
    system('/sbin/service myservice reload'); …
  }
}
```

Existing components

Over 90 NCM configuration components are currently available:

- Configure basic Quattor and core system services
  - Quattor services: `ccm, spma, cdp`
  - System services: `accesscontrol, accounts, autofs, cron, filecopy, grub, iptables, ldconf, lmsensors, logrotate, mailaliases, netdriver, nfs, ntpd, portmap, profile, serialclient, smartd, ssh, sysctl`
- Configure advanced system services
  - Including `castor, chkconfig, fiberchannel, fmonagent, gdmconf, ipmi, lsfclient, named, quota, screensaver, sysacct`
  - These would need more testing outside CERN
- Configure Grid services *(Cal's talk)*
  - `bdiicfg, ceinfo, cliconfig, cmnconfig, condorconfig, edglcg, gip, globuscfg, gridmapdir, guiconfig, infoproviders, lbconfig, lcas, lcgbdii, lcginfo, lcmaps, mkgridmap, myproxy, pbsclient, rgmaproducer, rm, uicmnconfig, wlconfig, yaim`

Existing components

Over 90 NCM configuration components are currently available:

- Configure basic Quattor and core system services
  - Quattor services: `ccm, spma, cdp`
  - System services: `accesscontrol, accounts, autofs, cron, filecopy, grub, iptables, ldconf, lmsensors, logrotate, mailaliases,`
- C
- C
  - `bdiicfg, ceinfo, cliconfig, cmnconfig, condorconfig, edglcg, gip, globuscfg, gridmapdir, guiconfig, infoproviders, lbconfig, lcas, lcgbdii, lcginfo, lcmaps, mkgridmap, myproxy, pbsclient, rgmaproducer, rm, uicmnconfig, wlconfig, yaim`

- Please check with us before starting a new component!
- Enhancements to existing components are welcome as well as new components for new functionality
- Your contribution will be much appreciated ☺

Components and CDB configuration

- Components can have "private" configuration entries, including:

```
/software/components/<name>/active (bool)              <- component active?
                           dispatch (bool)             <- run automatically via cdispd?
                           dependencies/pre (string[]) <- run components before
                           dependencies/post (string[]) <- run components after
                           foo/...                        (component specific)
                           bar/...                        (component specific)
```

- Components can access configuration information *anywhere* in the node profile (`/system/..`, `/software/..`, `/hardware/..`)
  - Useful to share common configuration entries between components
  - Eg. `/system/kernel/version`

- All components need to declare their "private" config data types, and can define default values

```
pro_declaration_component_<component>.tpl   <- structure
pro_software_component_<component>.tpl      <- default values
```

Example components (I)

ncm-grub:

- Functionality
  - configures the GRUB boot loader.
  - Uses the 'grubby' command line tool.
  - Won't change grub config if inconsistencies found.
- Most important config parameters:

`/system/kernel/version` *(string): kernel version to be used.*

- More info:
  - `man ncm-grub`

Example components (II)

ncm-cron:

- Functionality
  - Adds/removes cron entries.
  - Places them under /etc/cron.d with a log file in /var/log.
  - Respects existing cron.d entries.
- Most important config parameters:

```
/software/components/cron/entries/list/name  (string) cron entry name (eg.
                                                                "example")
                                        user  (string) user (eg. "root")
                                   frequency  (string) eg. "* 1 * * *"
                                     command  (string) "/bin/myexec"
```

- More info:
  - `man ncm-cron`

Example components (III)

ncm-accounts:

- Functionality
  - Controls the /etc/passwd, /etc/group, (/etc/shadow) files.
  - Places them under /etc/cron.d with a log file in /var/log.
  - Respects existing cron.d entries.
- Most important config parameters:

```
/software/components/accounts/rootpwd    (string) crypted root password.
                                  shadowpwd  (boolean) use /etc/shadow.
For every user:
/software/components/accounts/users/<user>/comment  (string) comment field
                                          <user>/uid  (string) groups it belongs to
                                          <user>/passsword  (str) crypted password
                                          <user>/createHome  (bool) make homedir?
```

- More info:
  - `man ncm-accounts`

How to run components? (I)

Manually:

- `ncm-ncd` (Node Configuration Deployer):
  - framework and front-end for executing components (via cron, cdispd, or manually)
  - dependency ordering of components
  - Invoke it with:

```
# ncm-ncd --configure                  runs configure on all active components
# ncm-ncd --configure [<component>]    runs configure method on <component>
                                       and dependent components
# ncm-ncd --unconfigure <component>    runs unconfigure method
# ncm-ncd --list                       gives information about all installed components, and their
                                       dependencies
```

  - You should run it manually eg. for debugging purposes
  - A logfile directory (with all component logs) is found under

```
# /var/log/ncm/ncm-ncd.log              <- general framework log
# /var/log/ncm/component-<component>.log  <- log of every component
```

How to run components? (II)

Automatically:

- `ncm-cdispd` (Configuration Dispatch Daemon)
  - Monitors the config profile, and invokes registered components via `ncm-ncd` if there were changes
  - Looks up for changes for every component in the following entry:

```
/software/components/<component>/...
```

  - Additional entries to watch can be configured (eg. /system/kernel/version for the grub component)
  - Deactivated on your nodes, but you will enable it *later*! You can enable it by running

```
# /sbin/service ncm-cdispd start
```

  - Also, monitor its progress by running in a separate window

```
# tail -f /var/log/ncm-cdispd.log
```

Exercises: configure and run components

Exercise 1

run NCM by hand with default profile

- *Don't* start ncm-cdispd for this exercise.
- Use ncm-query for visualizing the currently configured component list, and to visualize all configuration information:

```
# ncm-query --components
```

- You can also check

```
# ncm-ncd --list
```

- Run now by hand all active components:

```
# ncm-ncd --configure
```

- Run now a specific component:

```
# ncm-ncd --configure grub
```

What other component is run, and why?

- Modify the default profile to deactivate component 'spma'

You can do this by adding to `profile_lxb<xxx>.tpl`:

```
"/software/components/spma/active"=false;
```

What happens when you run now ncm-ncd, and why?

Exercise 2

Modify a component's configuration

- Start up ncm-cdispd to avoid running ncm-ncd by hand. Check the logfile /var/log/ncm-cdispd.log.
- Read the component's man page for 'grub'.
- Examine which kernels are installed on your node, and modify the kernel version to be used on the next reboot
  - Modify "/system/kernel/version" inside the profile_lxb<xxxx>.tpl template.
  - What kernel is currently configured? (using `uname -a` and `ncm-query --dump`)
  - Available kernels: `cat /etc/grub.conf` or `ls /boot/vmlinuz*`
  - **Don't** reboot the node! :-)
- Try out what happens if you configure a non-existing kernel.
- What happens if you run `ncm-ncd --unconfigure grub` and why?
  - Note: The component code is found under

    ```
    # less /usr/lib/perl/NCM/Component/grub.pm
    ```
- Restore the previously configured kernel.

Exercise 3

Modify a component's configuration – advanced..

Cron:

- Configure "cron" component to install a cron job to run `'ls /proc'` every minute, as user 'root'.
- Check the resulting file in `/etc/cron.d`.

Accounts:

- Add a new user:
  - Login id: 'napoleon'
  - Comment: '2Legs Better'
  - uid: 8888
  - Create the home directory.
- Verify the creation of the corresponding `/etc/passwd` and home directory entries.

How to write/modify components

Components and CDB profile access

NVA API: configuration access library

- This library allows to access the node profile's configuration (hierarchical structure)
- Most popular methods:
  - `$value=$config->getValue('/system/kernel/version');`
  - `if ($config->elementExists($path)) {…} else {…}`
  - ```
    $element=$config->getElement($path);
    while ($element->hasNextElement()) {
        my $newel=$element->getNextElement();
        ...
    }
    ```
    ```

Component support libs

- Reporting functions:
  - `$self->log(@array)`: *write @array to component's log file*
  - `$self->report(@array)`: *write @array to log and stdout.*
  - `$self->verbose(@array)`: *verbose output*
  - `$self->debug(level,@array)`: *debug output at debug level 1..5*
  - `$self->warn(@array)`: *writes a `[WARN]` message, increases # of warnings*
  - `$self->error(@array)`: *writes an `[ERROR]` message, increases # of errors*
- Failures of reconfigurations are done using '`error(..)`'.
- Components depending on a failed component are **not** executed.
- Advanced support libraries available (revamped from CERN's SUE):
  - Configuration file manipulation
  - Advanced file operations
  - Process management
  - Exception management libraries

  See `/usr/lib/perl5/site_perl/LC/*.pm` for details

Real (simple) component walktrough

- Please unwind the following tar file:

```
$ cd ~
$ tar xvfz /afs/cern.ch/user/g/gcancio/public/ncm-tutorial.tgz
```

- ncm-state (state.pm)
  - Updates a local configuration file (`/etc/state.desired`) with the node's production state ("production", "standby", etc.)

Packaging components: files (1/3)

- Each component is packaged independently and kept in a separate CVS area.
  - Let's check our example: please `cd` to

    `~/component-cvs/ncm-state`

- Files to be present in a component CVS module:
  - **README** – small intro
  - **ChangeLog** – automatically maintained ChangeLog file.
  - **LICENSE** – contains license or pointer to it
  - **MAINTAINER** – one liner with email
  - **Makefile** – copied from quattor-build-tools, essentially an 'include' of the quattor-build-tools

Packaging components: files (2/3)

- **config.mk** – obligatory and optional definitions
  - COMP – component name
  - DESCR – one liner with component description
  - VERSION – in the format <majorversion>.<minorversion>.<release>
  - RELEASE – RPM release number (always 1 for time being ☹)
  - AUTHOR – author's email
  - DATE – dd/mm/yy hh:mm
  - Any other optional definition(s)
- ***comp*.pm.cin** –component *source* file (a la 'autotools')
  - Source format! @TAGS@ get expanded into configuration variable values
  - Configuration variables set in config.mk or predefined inside **quattor-build-tools** (quattor-Linux.mk, quattor-SunOS.mk)
- ***comp*.pod.cin** – POD doc file. Please follow conventions:
  - NAME
  - SYNOPSIS: Configure() and Unconfigure() documentation
  - RESOURCES: describe all used resources
  - DEPENDENCIES: 'pre' and 'post' dependencies
  - BUGS
  - AUTHOR
  - SEE ALSO
- **pro_declaration_component_*comp*.tpl.cin** – PAN component template

Packaging components: files (3/3)

- **Extra data files** (template config files, etc): Can be stored in a subdirectory called 'DATA'
  - Example: see ncm-state/DATA
  - The file is copied into the directory @NCM_DATA_COMP@ (typically, /usr/lib/ncm/config/*name*/)
- **Extra documentation files**: Can be stored in a subdirectory called 'DOC'
  - The file is then dropped into the standard RPM package documentation directory
- Extra template files: Can be stored in a subdirectory called 'TPL'
  - All files are then dropped into the standard RPM package doc directory
  - The pro_declaration_component… template can be stored either in the top level directory or in the 'TPL' directory
  - All templates are copied to a common directory as well (QTTR_DOCDIR/pan-templates, typically `/usr/share/doc/pan-templates`)

Packaging components: commands

- Generate sources out of `'.cin'` files
  - **`make`**
- Generating packages out of checked out sources
  - **`make rpm`** – *generates RPM (on RHLinux)*
  - **`make pkg`** – *generates PKG (on Solaris)*
- generate new version
  - **`make (release|minorversion|majorversion)`**
  - checks in modified files to CVS
  - Prompts for ChangeLog entry (one line)
  - Increases release / minorversion / majorversion in config.mk
  - Generates a new CVS tag for the component
  - Note: ensure your files are declared in CVS (ie. 'cvs add')
- **Cleaning up**
  - **`make clean`**
  - Removes temp files generated during package build process

Exercise: modify an existing component

Exercise 4

Modify the ncm-state component to add a 'reason' field to /etc/state.desired

1. Add a 'reason' field to the state NCM declaration template
2. Add the wanted value to the 'data' template
3. Add the NCM declaration template to CDB (after running 'make')
4. Regenerate the profile and verify with ncm-query that the new field is there
5. Add handling code to the component source to read out the new field, and to update /etc/state.desired
6. Generate the ncm-state rpm after increasing the 'minorversion' (*not manually*)
7. install the rpm (with rpm –Uvh <filename> - in the 'real' world this would be done via SPMA)
8. Run the component, and verify that /etc/state.desired contains the new data field
9. Did you forget to 'activate' the component?

quattor

http://quattor.org