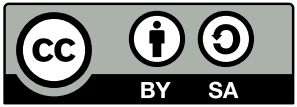
L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien



CM Tools

1 CM Tools
2 Key-value Access
3 CM Design
4 Meeting

Learning Outcomes

Students will be able to

- describe systematic approaches for configuration management and exemplary configuration management tools.

Definition

Configuration Management

system administrator:

- ensures computers are assembled from desired parts (inventory list)
- ensures correct applications are installed
- maintains files/databases
- monitors infrastructure
- ***manipulates configuration settings***

Definition

Configuration Management Tools:

- help system administrators in configuration management
- describe the desired configuration of the whole managed system
- converge the actual configuration to the desired one [1]

Cloning

It all started with:

- clone all files with rdist, NFS, rsync or unison ("golden image")
- then do necessary modifications with scripts or profiles

+ works good for many identical stateless machines
– fails if differences between machines are too big

Profiles

Profiles are groups of configuration settings between which the user can easily switch.

- by hostname, EEPROM, ...
- can be activated via the `profile` plugin:

```
1 [application/profile]
2 type := string
3 opt := p
4 opt/long := profile
5 default := current
```

with a config like:

```
1 application/current/key = "current"
2 application/myprofile/key = "myprofile"
3 application/%/key = "default"
```

Scripts

Next improvement: have a script to create the “golden image”. Possible benefits:

- Documentation
- Customization (using configuration settings)
- **Reproducability**: Reproduce creation using different operating system versions

Possible problems:

- imperative style
- configuration drift

Possible Benefits of CM tools

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [3])
- Error handling
- Reusability
- Abstractions

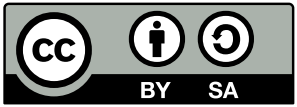
List of CM tools

- CFengine (1993)
- Puppet (2005)
- Chef (2009)
- cdist (2010)
- Salt (2011)
- Ansible (2012)
- Itamae (2014)
- Bolt (2018)
- Transilience (2020, no release yet)

L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien



Key-value Access

1 CM Tools

2 Key-value Access

3 CM Design

4 Meeting

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory
- *Key/value view of configuration settings*

- CM code to instantiate settings in the whole network
- Global optimization: allocation of nodes and decision regarding topology

Precise Editing

- Precise editing is natural for humans.
- Preserves (security-relevant!) defaults.

In CM following methods are used:

- replace full content of configuration files (with templates)
- line based manipulation (e.g., file_line): match line and replace it
- Augeas/XML: match a key with XPath and replace it
- Elektra: key/value access

Key/value access in puppet-libelektra [4]:

```
1 kdbkey {'system:/slapd/threads/listener':
2     ensure => 'present',
3     value => '4'
4 }
```

Key/value access in puppet-libelektra:

```
kdbmount {'system:/sw/samba':
    ensure => 'present',
    file => '/etc/samba/smb.conf',
    plugins => 'ini'
}
kdbkey {'system:/sw/samba/global/workgroup':
    ensure => 'present',
    value => 'MY_WORKGROUP'
}
kdbkey {'system:/sw/samba/global/log level':
    ensure => 'absent'
}
```

Uniqueness of keys is essential. Ideally, applications already mount their configuration at installation.

Key/value specifications in puppet-libelektra:

```
kdbkey {'system:/sw/samba/global/log level':
    ensure => 'present',
    value => 'MY_WORKGROUP', # not an int
    check => {
        'type' => 'short',
        'range' => '0-10',
        'default' => '1',
        'description' => 'Sets the amount of log/
            debug messages that are sent to the
            log file. 0 is none, 3 is consider-
            able.'
}
```

Key/value specifications in puppet-libelektra:

```
1 kdbkey {'spec:/xfce/pointers/Mouse/RightHanded':
2     ensure => 'present',
3     check => {
4         'visibility' => 'important',
5         'default' => 'false',
6         'check/type' => 'boolean'
7 }
```

Ideally, applications already specify their settings.

Key/value access in Chef:

```
1 kdbset 'system:/sw/samba/global/workgroup' do
2     value 'MY_WORKGROUP'
3     action :create
4 end
```

Key/value access in Chef:

```
1 kdbset 'system:/slapd/threads/listener' do
2     value '4'
3     action :create
4 end
```

Finding

We have CM code representing the settings.

Key/value access in Ansible:

```
1 - name: setup LDAP
2   connection: local
3   hosts: localhost
4   tasks:
5   - name: set listening threads
6     elektra:
7       mountpoint: system:/slapd
8       keys:
9         threads:
10          listener: 4
```

Key/Values

Decide about **changeability** per key:

- Who is responsible (end user, packages, system administrator manual or CM)?
- Who can see it (visibility)?
- Who can edit it (system administrator, end user, both)?
- Which configuration values are allowed (validation)?

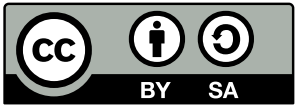
Changeability

Ownership of every key must be very clear and documented.

L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien



CM Design

1 CM Tools

2 Key-value Access

3 CM Design

4 Meeting

Idempotence

idem + potence (same + power)

Yield same result with any number of applications ($n \geq 1$):

$$f(f(x)) = f(x)$$

Siméon and Wadler [5] describe two further properties:

Self-describing means that from the configuration file alone we are able to derive the correct data structure [5].

Round-tripping means that if a data structure is serialized and then parsed again, we end up with an identical data structure [5].

Design Rules [2]

- Maintain clear separation of ownership (for every key).
- Factor processes into containers to avoid overlaps in settings.
- Specify replicated settings in a single source (use links and derivations).
- Document all remaining overlaps (in the specification).
- The manageability of settings is reduced by the number of possible configuration values.

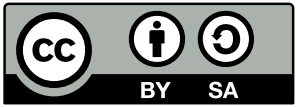
Conclusion

- unique identifiers
  $\rightarrow$ allows to get/set configurations and specifications
- solving CM is solving constraints
  $\rightarrow$ be aware of the specifications
- do not design around tools but design tools around you
- change only settings you need
- use all help you can get: e.g. build tools, preseeding, installer automation, virtualization, package managers, distributions

L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien



Meeting

1 CM Tools

2 Key-value Access

3 CM Design

4 Meeting

[1] Mark Burgess. A site configuration engine. In *USENIX Computing systems*, volume 8, pages 309–337, 1995.

[2] Mark Burgess and Alva L Couch. Modeling next generation configuration management tools. In *LISA*, pages 131–147, 2006.

[3] Waldemar Hummer, Florian Rosenberg, Fábio Oliveira, and Tamar Eilam. Testing idempotence for infrastructure as code. In David Eyers and Karsten Schwan, editors, *Middleware 2013*, pages 368–388, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-45065-5.

[4] Markus Raab, Bernhard Denner, Stefan Hahnenberg, and Jürgen Cito. Unified configuration setting access in configuration management systems. In *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*, pages 331–341. ACM, 2020. doi: 10.1145/3387904.3389257. URL `https://doi.org/10.1145/3387904.3389257`.

[5] Jérôme Siméon and Philip Wadler. The essence of xml. pages 1–13, 2003. doi: 10.1145/604131.604132. URL `http://dx.doi.org/10.1145/604131.604132`.