L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

21.04.2021

History of CM

1. History of CM
2. CM Languages
3. CM Tools
4. Meeting
   - Recapitulation
   - Assignments
   - Preview

Learning Outcomes

Students will be able to

- remember differences between CM languages and historical approaches.
- write simple configuration management scripts.

Definition

Configuration Management:

- is a discipline in which configuration (in the broader sense) is administered.
- makes sure computers are assembled from desired parts and the correct applications are installed.
- ensures that the execution environment of installed applications is as required.

Definition

Configuration Management Tools:

Definition

Configuration Management Tools:

- help people involved in configuration management.
- have means to describe the desired configuration of the whole managed system.
- try to converge the actual configuration to the desired one [4].

Challenging tasks in configuration management:

Challenging tasks in configuration management:

- inventory list
- installing packages
- monitoring
- add/replace machines
- maintaining files/databases
- ***configuration file manipulation***

Cloning

It all started with:

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

Cloning

It all started with:

- clone all files with dd, rdist, 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

Scripts

First improvement: have a script to create the “golden image”.

Possible benefits:

- Documentation

Scripts

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

- Documentation
- Customization (using configuration settings)

Scripts

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

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

Profiles

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

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

```
[application/profile]
  type:=string
  opt:=p
  opt/long:=profile
  default:=current
```

with a config like:

```
application/current/key = "current"
application/myprofile/key = "myprofile"
application/%/key = "default"
```

First four configuration management tools

Cloning, and then NIS/NFS, was state of the art for a long time, until in 1994 when *"the community nearly exploded with four new configuration systems"* [6]:

lcfg	from Anderson [2]. The development of lcfg started first in 1991 [1, 2]. Nevertheless, its development still continues [3, 9].
GeNUAdmin	from Harlander [7].
omniconf	from Hideyo [8].
config	from Rouillard and Martin [12].

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])
- Less configuration drift

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])
- Less configuration drift
- Error handling

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])
- Less configuration drift
- Error handling
- Pull/Push

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])
- Less configuration drift
- Error handling
- Pull/Push
- Reusability

Possible Benefits

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])
- Less configuration drift
- Error handling
- Pull/Push
- Reusability
- (Resource) Abstractions

L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

21.04.2021

CM Languages

1. History of CM
2. CM Languages
3. CM Tools
4. Meeting
   - Recapitulation
   - Assignments
   - Preview

See Reading Text

See accompanied reading text for this section.

L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

21.04.2021

CM Tools

1. History of CM
2. CM Languages
3. CM Tools
4. Meeting
   - Recapitulation
   - Assignments
   - Preview

List of CM tools

- CFengine (1993)

List of CM tools

- CFengine (1993)
- LCFG (1994)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)
- Chef (2009)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)
- Chef (2009)
- Salt (2011)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)
- Chef (2009)
- Salt (2011)
- Ansible (2012)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)
- Chef (2009)
- Salt (2011)
- Ansible (2012)
- Itamae (2014)

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)
- Chef (2009)
- Salt (2011)
- Ansible (2012)
- Itamae (2014)
- Puppet

List of CM tools

- CFengine (1993)
- LCFG (1994)
- Quattor (2005)
- Puppet (2005), Bolt (2018)
- Chef (2009)
- Salt (2011)
- Ansible (2012)
- Itamae (2014)
- Puppet
- OpsMops (2019)

Key/value access in puppet-libelektra [11]:

```
kdbkey {’/slapd/threads/listener’:
    ensure => ’present’,
    value => ’4’
    check => {
        ’type’ => ’short’,
        ’range’ => ’1,2,4,8,16’,
        ’default’ => ’1’
    }
}
```

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',
    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:

```
kdbkey {'spec:/xfce/pointers/Mouse/RightHanded':
    ensure => 'present',
    check => {
        'namespaces/#0' => 'user',
        'namespaces/#1' => 'system',
        'visibility' => 'important',
        'default' => 'false',
        'check/type' => 'boolean'
}
```

Ideally, applications already specify their settings.

Key/value access in Chef:

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

Key/value access in Chef:

```
kdbset ’/slapd/threads/listener’ do
    value ’4’
    action :create
end
```

Key/value access in Chef:

```
kdbset '/slapd/threads/listener' do
    value '4'
    action :create
end
```

Finding

We have CM code representing the settings.

Key/value access in Ansible:

```
- name: setup LDAP
  connection: local
  hosts: localhost
  tasks:
  - name: set listening threads
    elektra:
      key: ’/slapd/threads/listener’
      value: ’4’
```

Key/value access in Ansible:

```
- name: setup samba
  connection: local
  hosts: localhost
  tasks:
  - name: set workgroup
    elektra:
      mountpoint: system:/sw/samba
      file: /etc/samba/smb.conf
      plugins: ini
    elektra:
      key: 'system:/sw/samba/global/workgroup'
      value: 'MY_WORKGROUP'
```

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).
- In which namespaces apps search the key (cascading lookup).

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).
- In which namespaces apps search the key (cascading lookup).
- Who can see it (visibility).

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).
- In which namespaces apps search the key (cascading lookup).
- Who can see it (visibility).
- Who can edit it (admin, end user, both).

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).
- In which namespaces apps search the key (cascading lookup).
- Who can see it (visibility).
- Who can edit it (admin, end user, both).
- Which configuration values are allowed (validation).

Key/Values Revisited

Decide about **changeability** per key:

- Who is responsible (end user, packages, admin manual or CM).
- In which namespaces apps search the key (cascading lookup).
- Who can see it (visibility).
- Who can edit it (admin, end user, both).
- Which configuration values are allowed (validation).

Changeability

Ownership of every key must be very clear and documented.

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

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

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory
- Key/value view of configuration settings
- Goals/specifications of settings per node and instantiations of modules

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory
- Key/value view of configuration settings
- Goals/specifications of settings per node and instantiations of modules

- CM code to instantiate settings in the whole network

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory
- Key/value view of configuration settings
- Goals/specifications of settings per node and instantiations of modules

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

Layers of Abstractions

Recursively define useful abstractions (meta-levels):

- Bits in (configuration) files and memory
- Key/value view of configuration settings
- Goals/specifications of settings per node and instantiations of modules

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

Design Rules [5]

- Factor processes into containers to avoid overlaps in settings.

Design Rules [5]

- Factor processes into containers to avoid overlaps in settings.
- Maintain clear separation of ownership (for every key).

Design Rules [5]

- Factor processes into containers to avoid overlaps in settings.
- Maintain clear separation of ownership (for every key).
- Specify replicated settings in a single source (use links and derivations).

Design Rules [5]

- Factor processes into containers to avoid overlaps in settings.
- Maintain clear separation of ownership (for every key).
- Specify replicated settings in a single source (use links and derivations).
- Document all remaining overlaps (in the specification).

Design Rules [5]

- Factor processes into containers to avoid overlaps in settings.
- Maintain clear separation of ownership (for every key).
- 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.

Open Topics

- global optimizations/self-healing

Open Topics

- global optimizations/self-healing
- configuration integration

Open Topics

- global optimizations/self-healing
- configuration integration
- safe migrations of settings and data

Open Topics

- global optimizations/self-healing
- configuration integration
- safe migrations of settings and data
- collaboration

Open Topics

- global optimizations/self-healing
- configuration integration
- safe migrations of settings and data
- collaboration
- management (including knowledge)

Open Topics

- global optimizations/self-healing
- configuration integration
- safe migrations of settings and data
- collaboration
- management (including knowledge)
- centralized vs. distributed

Conclusion

- have unique identifier for your configurations settings
  → allows to get/set configurations and specifications

Conclusion

- have unique identifier for your configurations settings
  → allows to get/set configurations and specifications
- solving CM is solving constraints
  → be aware of the specifications

Conclusion

- have unique identifier for your configurations settings
  → allows to get/set configurations and specifications
- solving CM is solving constraints
  → be aware of the specifications
- do not design around tools but design tools around you

Conclusion

- have unique identifier for your configurations settings
  → allows to get/set configurations and specifications
- solving CM is solving constraints
  → be aware of the specifications
- do not design around tools but design tools around you
- be brave and remove all configuration settings you can

Conclusion

- have unique identifier for your configurations settings
  → allows to get/set configurations and specifications
- solving CM is solving constraints
  → be aware of the specifications
- do not design around tools but design tools around you
- be brave and remove all configuration settings you can
- use all help you can get: e.g. build tools, preseeding, installer automation, virtualization, package managers, distributions

Conclusion

- have unique identifier for your configurations settings
  → allows to get/set configurations and specifications
- solving CM is solving constraints
  → be aware of the specifications
- do not design around tools but design tools around you
- be brave and remove all configuration settings you can
- use all help you can get: e.g. build tools, preseeding, installer automation, virtualization, package managers, distributions
- complexity in CM vs. complexity in applications' specification

L05 Configuration Management

Markus Raab

Institute of Information Systems Engineering, TU Wien

21.04.2021

Meeting

1 History of CM

2 CM Languages

3 CM Tools

4 Meeting
- Recapitulation
- Assignments
- Preview

Question

How can we reuse the same configuration setting for different applications?

Question

How can we reuse the same configuration setting for different applications?

Answer

- Use CM code to copy the settings to all places as needed.
- Implement support directly in application to fetch setting from central location.
- Override/fallback links in specification.
- Calculate/transform values in specification.

Definition

Configuration Management:

Definition

Configuration Management:

- is a discipline in which configuration (in the broader sense) is administered.
- makes sure computers are assembled from desired parts and the correct applications are installed.
- ensures that the execution environment of installed applications is as required.

Definition

Configuration Management Tools:

Definition

Configuration Management Tools:

- help people involved in configuration management.
- have means to describe the desired configuration of the whole managed system.
- try to converge the actual configuration to the desired one [4].

Task

Break.

Possible Benefits of CM

Possible Benefits of CM

- All advantages scripts have:
  Documentation, Customization, Reproducability
- Declarative description of the system
  (Infrastructure as Code [10])
- Less configuration drift
- Error handling
- Pull/Push
- Reusability
- (Resource) Abstractions

Design Rules of CM [5]

Design Rules of CM [5]

- Factor processes into containers to avoid overlaps in settings.
- Maintain clear separation of ownership (for every key).
- 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.

CM Languages

- What is the relationship to software configuration management (Proteus/PCL)?

CM Languages

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.

CM Languages

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.
- How is it possible to provide referential transparency both for the configuration specification language and for the system itself (NIX, GNU Guix)?

CM Languages

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.

- How is it possible to provide referential transparency both for the configuration specification language and for the system itself (NIX, GNU Guix)?

  By functional languages and file system (layouts).

CM Languages

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.
- How is it possible to provide referential transparency both for the configuration specification language and for the system itself (NIX, GNU Guix)?

  By functional languages and file system (layouts).
- Which notations for CM exist?

CM Languages

- What is the relationship to software configuration management (Proteus/PCL)?

  Build systems may provide configuration management features.
- How is it possible to provide referential transparency both for the configuration specification language and for the system itself (NIX, GNU Guix)?

  By functional languages and file system (layouts).
- Which notations for CM exist?

  Text, Graphical (UML), Semi-structured, Key-value, Structured

Task

Break.

Today: T1/H1 corrections

Assignments

Please add slides for talk in private git repo at least one week in advance.

Task for H3?

Which CM language do you want to use for T3?

Outlook

Will be online soon:

- validation techniques

Outlook

Will be online soon:

- validation techniques
- writing plugins

[1] Paul Anderson. Local system configuration for syssies. Technical report, CS-TN-38, Department of Computer Science, University of Edinburgh, Edinburgh, 1991.

[2] Paul Anderson. Towards a high-level machine configuration system. In *LISA*, volume 94, pages 19–26, 1994.

[3] Paul Anderson, Alastair Scobie, et al. Lcfg: The next generation. In *UKUUG Winter conference*, pages 4–7, 2002.

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

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

[6] Rémy Evard. An analysis of UNIX system configuration. In *LISA*, volume 97, pages 179–194, 1997.

[7] Magnus Harlander. Central system administration in a heterogeneous Unix environment: GeNUAdmin. In *LISA VIII Proceedings*, 1994.

[8] Imazu Hideyo. OMNICONF–making os upgrades and disk crash recovery easier. In *LISA VIII Proceedings*, 1994.

[9] Johannes Hintsch, Carsten Görling, and Klaus Turowski. A review of the literature on configuration management tools. 2016.

[10] 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.

[11] 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`.

[12] John P. Rouillard and Richard B. Martin. Config: A mechanism for installing and tracking system configurations. In *LISA*, 1994.