Martin Pietsch

Ansible++ – Object orientation with Ansible

Dresden, May 18th, 2021

Agenda

Agenda

About me

- Martin Pietsch
- at the TU Dresden since 2005
- IT topics of interest:
  - Programming
  - Automation
  - Open Source
  - ...
- Co-organiser of the “Dresden OpenSource UserGroup” (DDOSUG)
  - Goal: a platform for everyone to get in touch with OpenSource
  - Social Media: Meetup, LinkedIn, Telegram and YouTube

TU Dresden

- founded in 1828
- largest “Technische Universitäten” in Germany
- one of the “Universities of Excellence” (since 2012)
- member of “Dresden Concepts”
- 17 faculties in five schools with 124 disciplines
- 32.000 students and 8.300 employees
- central computer centre is called “ZIH”
  - provides the IT infrastructure and IT services
  - founding member of the Gauss Alliance

Agenda

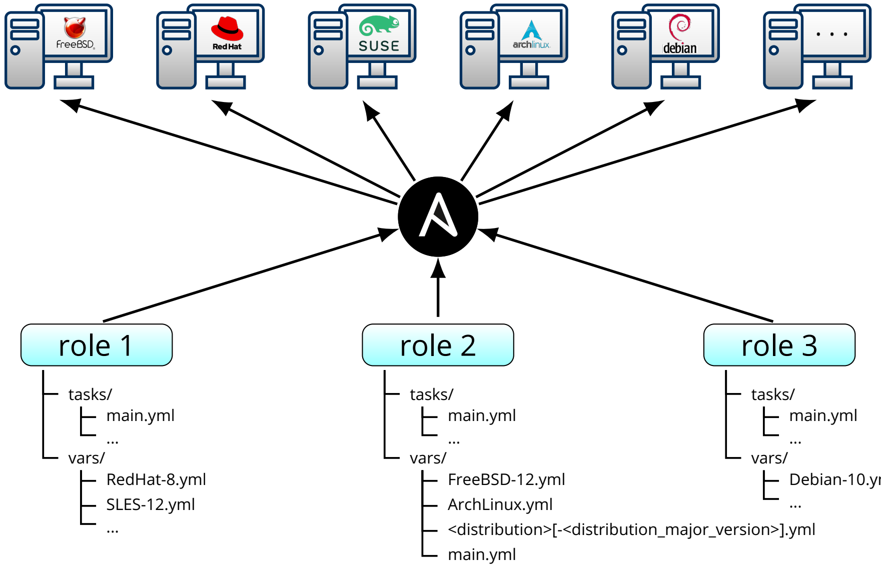
Background

- **S**imple **D**eploy- and **M**anagement (SDM)-Framework
- Start of development in 2016
- Diploma thesis in 2018
- Goals of the framework:
  $\rightarrow$ ease-of-use
  $\rightarrow$ modular and expandable components (roles)
  $\rightarrow$ low number of playbooks
  $\rightarrow$ platform-independent usage
  $\rightarrow$ automation of installation, update and migration processes
- License: BSD-3-Clauses (mostly)
- Website: `https://sdm.mn.tu-dresden.de`

Object orientation?

- Ansible? Object orientation? Programming?
- Ansible is an interpreter like Shell, Basic, etc.
  - has control structures (loop, when, include, etc.)
  - play $\equiv$ main routine
  - task $\equiv$ command
  - included tasks $\equiv$ function / procedure
  - role $\equiv$ class

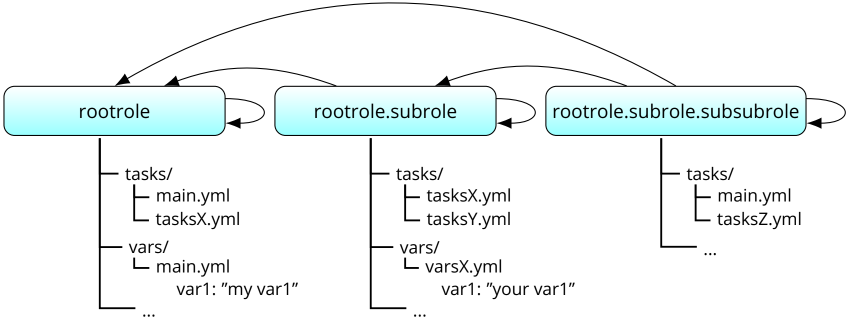
Object orientation – What? and Why?



Object orientation – What? and Why?

```
role/
├─ defaults/ ⇒ default values of properties
│                  ├─ <distribution>[-<distribution_major_version>].yml   ↑ loading
│                  └─ main.yml
├─ files/
├─ handlers/ ⇒ event handlers ⇒ main.yml
├─ meta/
├─ tasks/ ⇒ methods ⇒ filename ≙ name of a method
├─ templates/
├─ tests/
└─ vars/ ⇒ values of properties
              ├─ <distribution>[-<distribution_major_version>].yml   ↑ loading
              └─ main.yml
```

Object orientation – What? and Why?



Object orientation – How?

native action plugins:

- **include_role** ⇒ includes [parent-]roles
- **include_tasks** ⇒ includes [role-]tasks
- **include_vars**, **set_fact** ⇒ loads and set variable values

SDM plugins:

- strategy: **sdmlinear**, **sdmfree** ⇒ loads tasks and adjust variables
- callback: **sdmoor** ⇒ realises object-oriented roles
- action plugins:
  - ▶ **call_role** ⇒ loads a role and executes tasks
  - ▶ **call_tasks** ⇒ executes (inherited) tasks
  - ▶ **load_role_vars**, **set_role_fact** ⇒ loads and set role variables

Agenda

Practical part!

Agenda

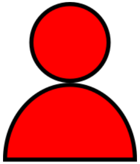
Conclusion

- (currently) no object orientation supported by Ansible
- advantages of this approach
  - clear structure
  - faster development
  - more fexibility
  - better collaboration
- disadvantages of this approach
  - learning curve is steeper
  - additional plugins necessary

Thanks for your interest!

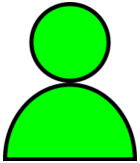
Why? - A simple example

Initial situation



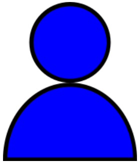
Customer 1

“We need a webserver based on nginx with SSL- and PHP-Support.”



Customer 2

“We need a webserver based on Apache HTTPD without SSL-Support, but with PHP-Support.”

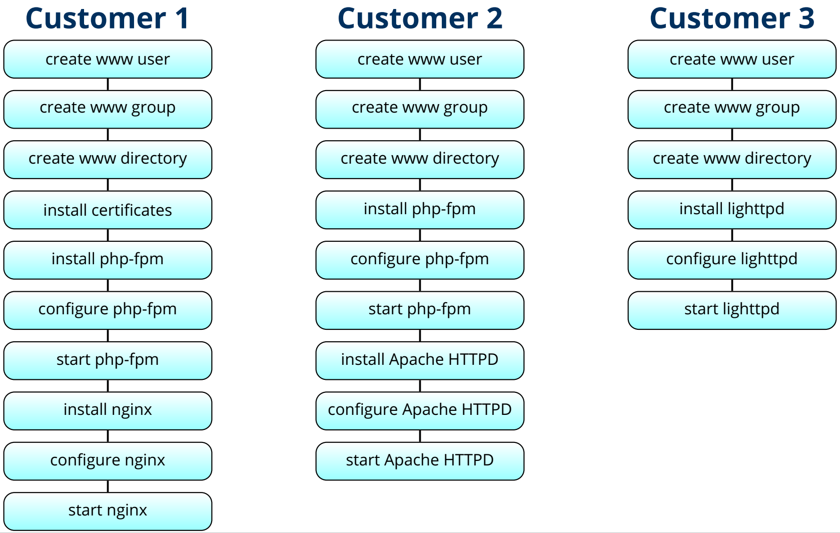


Customer 3

“We need a webserver based on lighttpd without SSL- and PHP-Support.”

Why? - A simple example

Solution 1: single playbook or role



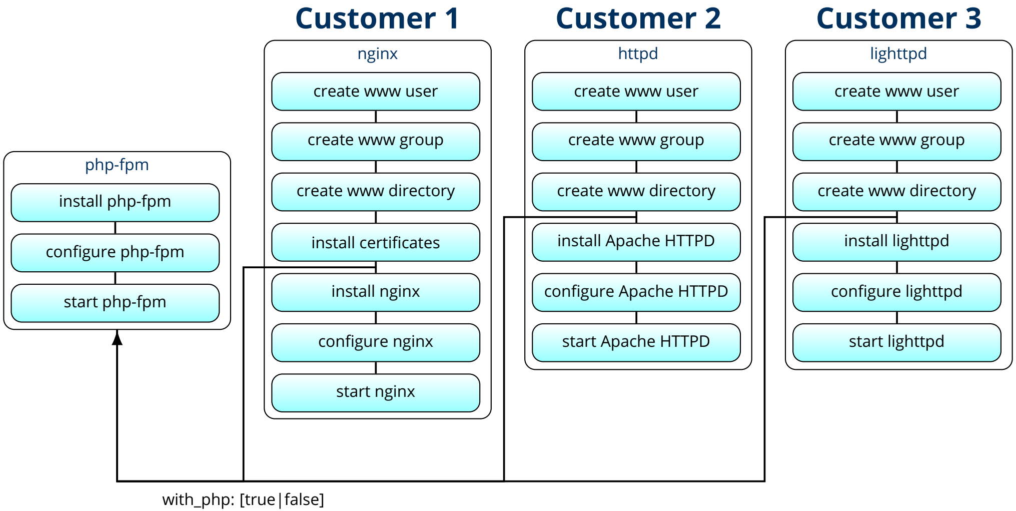
Why? - A simple example

Solution 1: single playbook or role

- Advantages:
  - very individual
- Disadvantages:
  - inflexible
  - high maintenance effort
  - many redundant tasks
- Risks:
  - Content can become confusing.
  - inconsistent variables complicate interchangeability
  - Extensibility can become more difficult.

Why? - A simple example

Solution 2: split software in different roles



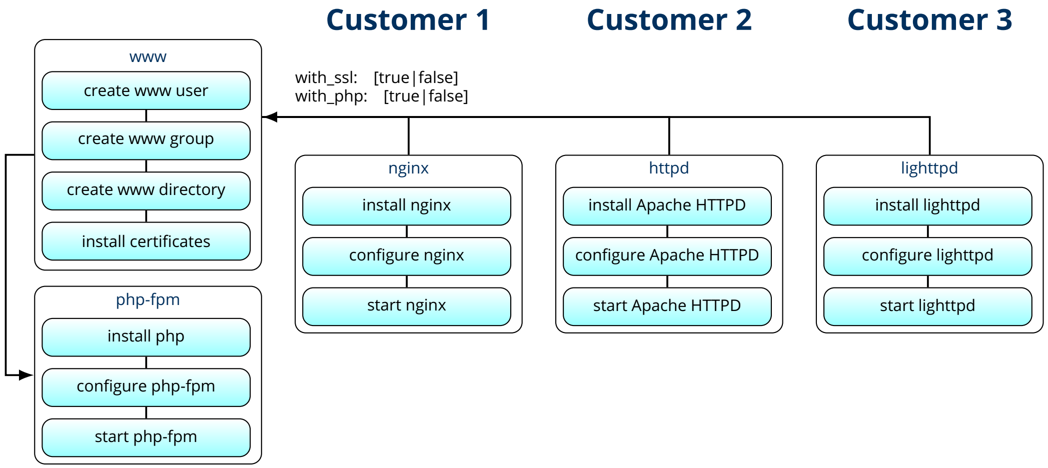
Why? - A simple example

Solution 2: split software in different roles

- Advantages:
  - more modular than solution 1
  - easier to maintain than solution 1
- Disadvantages:
  - high maintenance effort
  - many redundant tasks
- Risks:
  - Content can become confusing.
  - inconsistent variables complicate interchangeability
  - Extensibility can become more difficult.

Why? - A simple example

Solution 3: split software and software-independent tasks in different roles



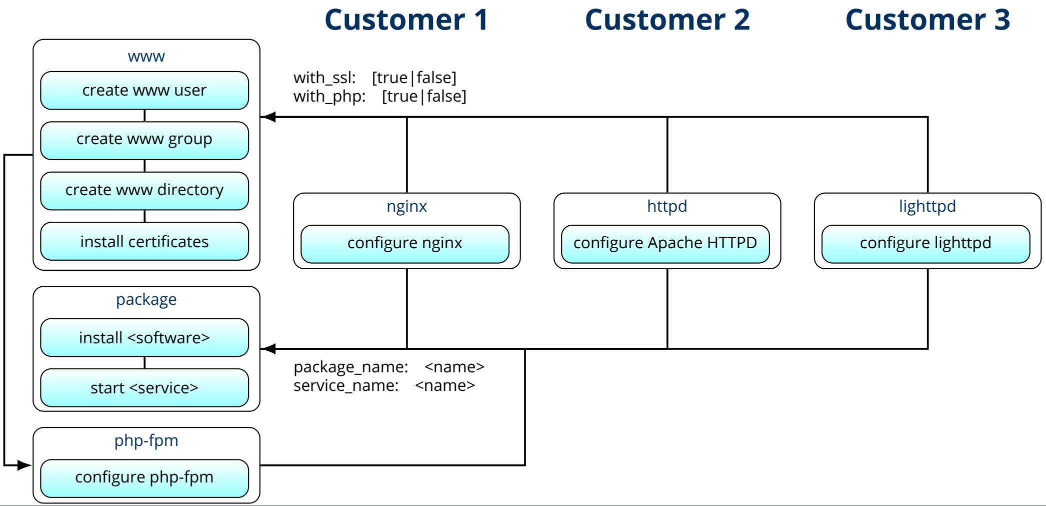
Why? - A simple example

Solution 3: split software and software-independent tasks in different roles

- Advantages:
  - a good modularity
  - easy to maintain
- Disadvantages:
  - few redundant tasks
- Risks:
  - Relations between roles can become confusing.

Why? - A simple example

Solution 4: split software, software-independent and common software tasks in different roles



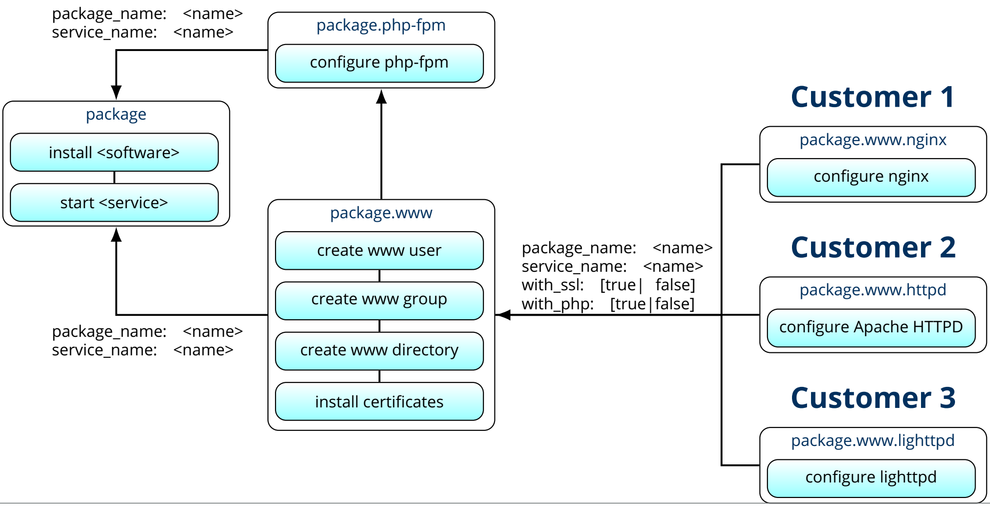
Why? - A simple example

Solution 4: split software, software-independent and common software tasks in different roles

- Advantages:
  - best modularity
  - easy to maintain
- Risks:
  - Relations between roles can become confusing.
  - Roles can overwrite each other’s variables.

Why? - A simple example

Solution 5: Convert solution 4 to the SDM object oriented approach



Why? - A simple example

Solution 5: Convert solution 4 to the SDM object oriented approach

- Advantages:
  - best modularity
  - easy to maintain
  - clear structure
- Disadvantages:
  - learning curve is steeper
  - additional plugins necessary

Why? - A simple example

Code ratio of the solutions*

	Solution 1	Solution 2	Solution 3	Solution 4	Solution 5
Solution 1	100%	80%	$53.\overline{3}\%$	$33.\overline{3}\%$	$33.\overline{3}\%$
Solution 2	125%	100%	$66.\overline{6}\%$	$41.\overline{6}\%$	$41.\overline{6}\%$
Solution 3	187.5%	150%	100%	62.5%	62.5%
Solution 4	300%	240%	160%	100%	100%
Solution 5	300%	240%	160%	100%	100%

* without include-tasks