Configuration Management

Chapter 23 in Unix and Linux System Administration Handbook

Haakon André Reme-Ness

HVL

Haakon.Andre.Reme-Ness@hvl.no

January 30, 2025

Configuration Management (CM)

- The general consensus within system administration is that *changes* should be
  - Structured
  - Automated
  - Applied consistently among machines
- But that's easier said than done:
  - System administrators are usually confronted with a fleet of heterogeneous networks in different state of health
- Configuration management software aims to automate the management of operating systems on a network

CM in a nutshell

- The traditional approach to sysadmin automation is an intricate complex of home-grown shell scripts
  - Supplemented by ad hoc firefighting when a script fails
- This scheme works as "well" as you'd expect...
  - Such systems tend to heavily degrade over time
- Often known as the "snowflake model", because no two systems are ever alike

CM in a nutshell

Better approach: configuration management

- Capture the desired state of a system using code
- Changes and updates can then be tracked over time in a version control system
  - By creating an audit trail and a point of reference
- The code also acts as informal documentation of a network

CM in a nutshell

- When setting up, a CM system effectively acts as both an inventory database and a command-and-control centre for the network
- These systems also provide orchestration features, enabling changes and ad hoc commands to be applied remotely
  - Can be directed towards a group of hosts with a certain pattern in their hostnames
  - Or whose configuration variables match a given set of values
- Managed clients report information about themselves to the central database for analysis and monitoring

CM in a nutshell

- The configuration management code uses a declarative idiom
  - Instead of writing scripts to tell the system what changes to make, sysadmins describe the desired state to achieve
  - The CM system then uses its own logic to adjust the target systems as necessary
- Ultimately, the job of a CM system is to apply a series of configuration specifications (aka operations) on individual machines
- Operations vary in granularity, but typically include items such as: creating user accounts, changing settings and/or installing software packages, etc.

Dangers of configuration management

- CM systems are a major improvement over using ad hoc scripts
- However, sysadims should be aware of some particularly important aspects:
- Firstly, there is a lack of conformity and standardisation among CM systems
  - Knowledge of one system is not necessarily portable to another
  - The same applies for their terminologies, which
    – Differ greatly, despite having similar conceptual models

Dangers of configuration management

- Secondly, as a site grows, the infrastructure needed to support its configuration management system must also grow
  - E.g., a site with a few thousand servers will need a handful of servers dedicated to CM workloads
- This leads to both direct and indirect costs in the form of hardware resources and maintenance

Dangers of configuration management

- Third, once a system is under the control of a CM system, it must not be modified manually again
  - Otherwise, it immediately reverts to a snowflake system
- Some manual updates override the expected state of the system, and prevents further updates by the CM system
  - It also causes confusion, as it is not always trivial to determine the cause when a desired change is not applied

Dangers of configuration management

- Finally, regardless that some CM systems are easier to pick up than others, they all have steep learning curves
- It is advised to practice **before** using CM systems on a production network

Elements of configuration management

- The next slides will review the components and configuration concepts of CM systems
- To avoid using the terminology of one particular CM system, a neutral terminology has been defined as follows:

Our term	Ansible	Salt	Puppet	Chef
operation	task	state	resource	resource
op type	module	function	resource type, provider	provider
op list	tasks	states	class, manifest	recipe
parameter	parameter	parameter	property, attribute	attribute
binding	play(book)	top file	classification, declaration	run list
master host	control	master	master	server
client host	host	minion	agent, node	node
client group	group	nodegroup	node group	role
variable	variable	variable	parameter, variable	attribute
fact	fact	grain	fact	automatic attribute
notification	notification	requisite	notify	notifies
handler	handler	state	subscribe	subscribes
bundle	role	formula	module	cookbook
bundle repo	galaxy	GitHub	forge	supermarket

Figure: Similar terms are named differently depending on the CM system

Elements of configuration management

Operations and parameters

- Operations are the small-scale actions and checks used by a CM system to achieve a particular state
- Operations are scripts, usually written in the same language as the CM system and using the system's standard tools and libraries
- Most operations accept parameters, e.g.,
  - a package management operation would often support parameters that specify package name, version, and if it is to be installed or removed
- Variable values can be used to define parameters

Elements of configuration management

Variables

- Named values that influence how configurations are applied to individual machines
- They are often used to set parameter values, and fill in the blanks in configuration templates
- Typically defined in many different places and contexts within the configuration base
- Each definition has a scope in which it's visible: may encompass a single machine, or a group of machines, or a particular set of operations
- Multiple scopes can be active in any given context, and can also be nested
- Multiple scopes can define values for the same variables
  $\rightarrow$ some form of conflict resolution is necessary

Elements of configuration management

Facts

- CM systems investigate each configuration client to determine descriptive facts, like
  – the IP address of the primary interface, and the OS type
- This information is then accessible from within the configuration base through variable values
- As with any other variable, facts can be used to define parameter values or to expand templates

Elements of configuration management

Change handlers

- Operations that run in response to events or situations, instead of being a part of the configuration
- For instance, restart the web server when a change is added to the configuration file

Elements of configuration management

Bindings

- Associates specific sets of operations to specific hosts or groups of hosts
- These can also be dynamic, defined by a *fact* or *variable*

Elements of configuration management

Bundles and bundle repositories

- A bundle is a collection of operations that perform a specific function, e.g.: installing, configuring and running a web server
- CM systems allow packaging bundles into a format suitable for distribution or reuse
- CM system vendors tend to maintain public repositories, containing both official and user added bundled

Elements of configuration management

Environments

- It is often useful to divide configuration managed clients into multiple "worlds", like
  - the traditional categories for development,
  - test
  - production, etc.
- These different worlds are known as environments, both inside and outside the configuration management context
- The segregation can help to creating finer distinctions to support processes like gradual rollout of new code into production

Elements of configuration management

Client inventory and registration

- Since CM systems define many ways in which how clients can be categorised, the overall universe of machines under configuration management must be well defined.
- The inventory of hosts under management can live in a flat file, or in a database, or may also be dynamic.
- The process of putting a new client under configuration management can be made as simple as installing some client-side software
  - If the environment has been configured for automatic bootstrapping, a new client will then automatically contact the config server, authenticate itself, and initiate the configuration process.

Example: Adding a new client

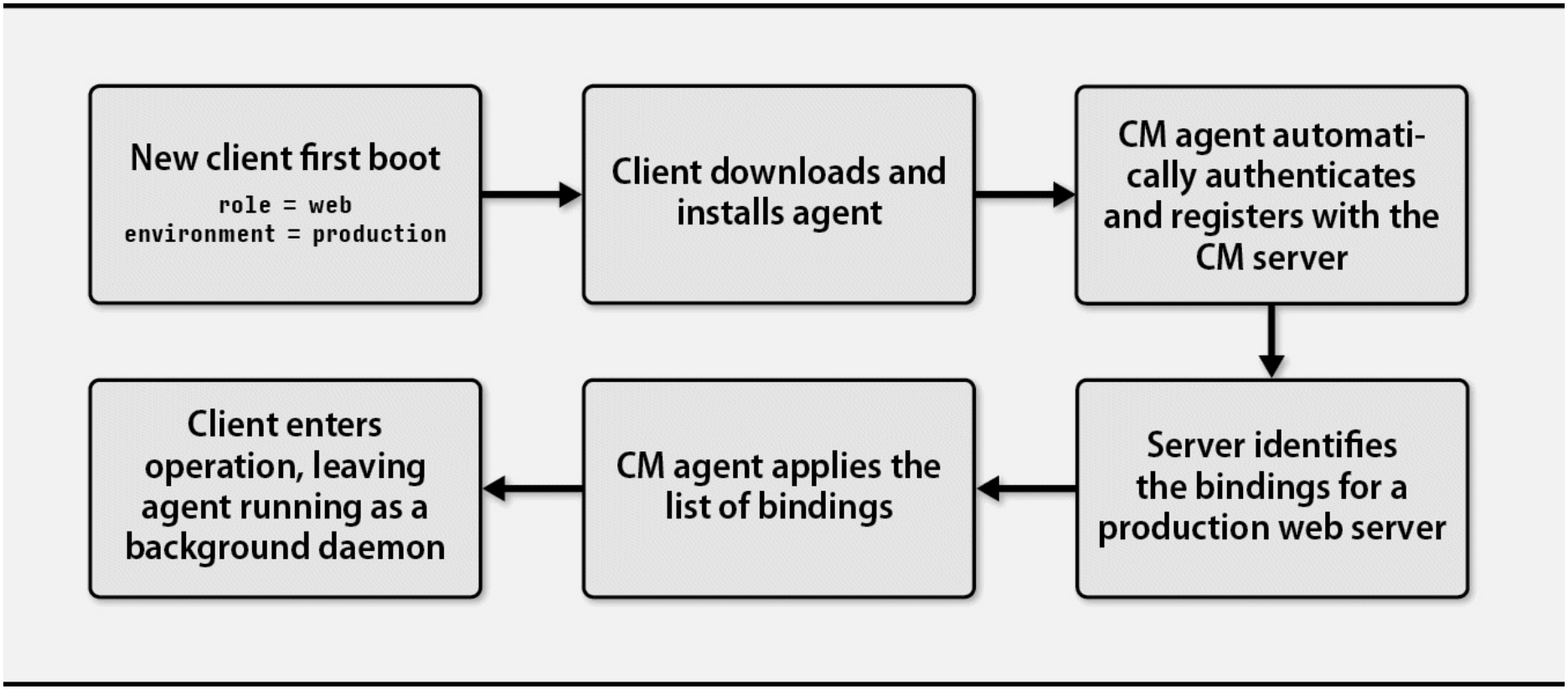

Figure: Adding a new client to a CM system

Distributing configurations

- The exact mechanism through which configuration code is distributed, parsed and executed, varies depending on the CM system
- Some of the more common approaches are:
  - Having a background service running on each client, checking against a CM server
  - Having a central CM server push the configurations data to the client. This can either be done periodically on a schedule, or manually by admins
  - Each managed node runs a client that wakes up periodically, reads the config data from a local clone of the configuration base, and applies it to itself. No central configuration server here.

Comparing CM systems

Currently, four major players completely own the market for configuration management on Linux/Unix systems:

- Ansible
- Salt
- Puppet
- Chef

Out of these, Puppet claims the largest market share thanks to its early head start[1]

[1] Released in 2005, four years before the next major competitor, Chef.

Comparing CM systems

		Languages and formats			Daemons	
System	**Web site**	**Impl**	**Config**	**Template**	**Server**	**Client**
Ansible	ansible.com	Python	YAML	Jinja	No	No
Salt	saltstack.com	Python	YAML	Jinja	Optional	Optional
Puppet	puppet.com	Ruby	custom	ERB[a]	Optional	Optional
Chef	chef.io	Ruby	Ruby	ERB	Optional	Yes

a. ERB (embedded Ruby) is a basic syntax for embedding Ruby code in templates.

Figure: Overview of the four major configuration management systems

Comparing CM systems

Business models

- All the previously mentioned CM systems use a "freemium" model
  - Each basic system is open-source and free, with a corporate backer that sells support, consulting services and add-on packages
- It is common for vendors to withhold functionality from the free, open-source, releases. However, this has not been evident with CM system vendors so far.

Comparing CM systems

Architectural options

- By default, Salt, Puppet and Chef have services intended to run on both the master- and client machines
- Except Ansible, which uses SSH to push configurations to the clients, without the need for a centralised server or client-side software.[2]
- Using SSH with no client-side services is simpler, but having both master- and client-side services has its advantages
  - It's faster
  - Some features can't exist without central coordination
  - The client-side complexity is usually simple anyways
  - Having active agents on both the clients and masters enables more architectural options.

[2] Salt can also run in a similar SSH mode

Comparing CM systems

Architectural options

- In general, Chef is considered to require more investment to maintain and master
- Ansible, Salt and Puppet have roughly the same complexity
- Because of its serverless model, Ansible is often tagged as being an "easy option" for configuration management, while Salt and Puppet should be similarly approachable.

Comparing CM systems

Language options

- Both Ansible and Salt are written in Python, but Python code is rarely seen in their configurations
- These systems use YAML[3] as their primary configuration language.
- As YAML simply structures data, YAML files are usually supported by the Jinja templating system to make them more dynamically expressive

[3] An alternate syntax for expressing JavaScript object notation, aka JSON

Comparing CM systems

Language options

- Puppet and Chef are both written in Ruby, and use Ruby-based domain-specific languages for their configurations
- Chef's approach to Ruby is similar to how it works in Rails (commonly used in web development)
- While Puppet implements it in a unique manner, more similar to the declarative YAML syntax.
  - This makes it hard to use existing Ruby skills!

Dependency management options

- In general, clients in a CM system execute operations
  - But some of these operations may be execution-order dependent
  - For instance, for a web server, we want to add the user “www” to its own dedicated group “www”
  - This will likely fail if the group "www" doesn't already existing
  - Ansible handles this as follows:

```
- name: Ensure that www group exists
  group: name=www state=present

- name: Ensure that www user exists
  user: name=www group=www state=present createhome=no
```

  - Note that Ansible executes operations in the order they are presented. Chef works the same way, too.

Comparing CM systems

Dependency management options

- In contrast, Puppet and Salt allow their dependencies to be explicitly declared.
- It does not matter in what order they are placed in the configuration

```
www-user:
  user.present:
    - name: www
    - gid: www
    - createhome: false
    - require:
      - www-group

www-group:
  group.present:
    - name: www
```

- Compared to the Ansible example, the order is inverted. But for Salt, it doesn't matter (by using `require` declaration)

Final thoughts

- Chef is complete, robust and scalable – more so than the alternatives
  - But comes with a big overhead, so only ideal for huge networks
  - Other CM systems can practically do the same, with less
- Puppet is the oldest CM system mentioned here, and the age is starting to show
  - The "historical baggage" tends to scare away newcomers
  - Known for server-side bottlenecks
  - Its approach to Ruby is less than ideal
- Ansible and Salt are often recommended today
  - Deceptively similar on the surface
  - Ansible is generally slower, but "simpler"
  - Salt is generally faster. Depending on the configuration, can also be "simpler" than Ansible