An introduction to Configuration Management Tools Salt and Ansible

Automated Software Testing and DevOps Course

Emelie Tham
KTH Royal Institute of Technology
Stockholm, Sweden
etham@kth.se

Introduction

The increasing shift within organizations towards automation has led to the creation of several configuration management tools. Previously, the configuration management market was dominated by *Puppet* and *Chef*. However, with technology rapidly evolving and applications being released at faster rates, new configuration management tools have been created to cover the weaknesses of the older configuration management tools. Both *Ansible* and *Salt* are the latest additions to the configuration management tool family and have shown to be powerful when utilized to their strengths. As such, it becomes important for an organization to be able to distinguish how these two tools differ from each other and what benefits they provide. The first section introduces the concept of configuration management and how it relates to DevOps. The second and third section describes the respective key components of Salt and Ansible. Lastly, a comparison is made between the two tools.

What is Configuration Management?

In software development, configuration management (CM) refers to the management of components - i.e. tools, servers, source codes e.t.c. - that are essential in facilitating the success of a project [23, 28]. As such, the definition of CM might differ from organization to organization, i.e. depending on the organization's need CM might only encapsulate source code management. The term *configuration management* is not exclusively used by the software industry and can be found in other industries as well - each with their definition of the term. For example, in the automotive industry, CM could refer to the configuration of the assembly line of important manufacturing machines. The process of CM follows three important activities [23, 28]:

- **Configuration identification.** The process of identifying the needed configurations for the system
- **Configuration control.** It is the process of controlling the changes in the configuration needs.
- **Configuration audit.** The process of reviewing whether the configurations of existing systems comply with regulations.

Configuration Management in DevOps

Traditionally, the process of managing the system's configuration and infrastructure was done manually and assigned to operator roles (i.e. system administrators); resulting in time-consuming and tedious tasks as well as difficulty in maintaining a consistent state of systems and software [23, 28]. Recent years have shown a cultural shift within organizations towards automation and embracing of DevOps. With CM tools organizations can practice DevOps more intuitively.

CM delivers the benefits of *infrastructure as code* (IaC) [28]. In simple terms, IaC is the process in which traditional manual methods for setting up the software infrastructure are automated in the form of code. This approach of treating infrastructure as code, if the CM is well-implemented, allows developers to deploy applications at a faster rate. One of the key benefits [21] of well-implemented CM is that it reduces the impact of system disasters; with the configuration automated and documented, service can be recovered more quickly. Another benefit is the minimization of bad deployments. Test servers can more easily replicate the production servers' environment with CM. As such, it is possible to avoid bad deployments caused by differences in the servers. Lastly, CM offers growing enterprises the ability to easily scale running applications, i.e. provisioning servers, by simply running a script or through the click of a button.

Three primary components are necessary for CM in DevOps, namely [23, 28]: (1) source code repository; (2) artifact repository; and (3) configuration management database. The source code repository can be thought of as a container for the development source code and is mostly used during the development phase. Generally, it contains the components that can be read and used by humans such as working code, test scripts, build scripts, deployment scripts and configuration files. On the other hand, the artifact repository can be seen as a container that stores machine files (i.e. binaries) and is used during the development phase as well as operational phases. Lastly, the configuration management database is a relational database that stores information about the system's resources - i.e. hardware and software assets - and their relationship with each other. Subsequently, this allows organizations to more easily monitor and get an overview of the state and relationship of their assets. As such, the configuration management database is used in both the development phase and operations phases.

Difference between Configuration Management and Orchestration Tools

To avoid confusion between the concept of *orchestration* and *configuration management* it is important to distinguish and describe the relationship between them - even if briefly. Much too often CM tools are confused with orchestration tools and are misused as such - leading to increased code complexity and subsequently harder to maintain and scale [22, 24]. The difference between CM and orchestration tools lies in its design and purpose [26]. CM tools such as *Ansible*, *Puppet*, *Chef*, *Salt* e.t.c. are used for installing and managing software on existing servers. In contrast, orchestration tools such as *Terraform*, e.t.c. aims to simply provision servers with the expectation that the required software is already installed and configured. It should be noted that the two categories are not mutually exclusive and most of the tools use both orchestration and CM to a certain degree.

Salt

SaltStack (a.k.a. Salt) is a python based CM tool for managing large and complex infrastructures through commands executed remotely. Salt was created by Thomas S Hatch in 2011 who felt that the open-source CM tools at that time could not deliver high-speed data connectivity [1]. As such, Salt was created as an open-source remote execution engine to facilitate the management of systems at scale. Eventually, Salt progressed towards delivering configuration management on top of the remote execution engine - thus evolving into a configuration management tool.

Salt Architecture

Salt follows the *server-agent* architecture, where a *Salt Master* (the server) controls one or more *Salt Minions* (agents) [16]. The role of the Salt Master is to orchestrate and manage the remote systems by communicating with the Salt minions (Figure 1). This communication channel between the master and minions is bi-directional and secured through encryption. More specifically, any minion that wishes to connect to the master needs to authenticate itself with its public key. Once a connection has been established the minion can receive commands from the master, execute them and return the results. Using *returners* it is possible to re-direct results from minions to another system instead [14].

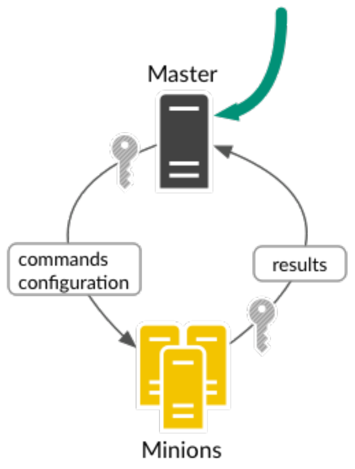

Figure 1. Salt master and minion communication model [16].

Configuration Management and Remote Execution in Salt

As mentioned previously, Salt's delivery of configuration management is built upon its remote execution engine. This engine is utilized through *execution modules* - a set of functions that can be performed on minions [7]. These functions would allow minions to perform various tasks such as restarting a service, running a remote command, installing packages e.t.c. Salt has several built-in modules [20]. However, if the built-in modules are not sufficient, it is also possible to write custom execution modules expressed in Python.

States are the core of Salt's configuration management functionality [17]. They are similar to execution modules with a slight but important difference; states take into consideration the state of the system and can check whether the system is in the correct state. Once confirmed, the state then uses remotes execution modules to perform tasks. States are stored in SLS files and are described using YAML (short for "YAML Ain't Markup Language).

Both states and execution modules are run on the targeted minions, however, to execute commands on the Salt master requires the use of *runners* [15]. These are modules that can be executed on the Salt master with the command line interface `salt-run`. For example, Salt runners can be used to monitor job status, connection status e.t.c. The more advanced task of orchestration is handled by the *orchestrate runner*. This runner can set up the configuration for several minions at once.

Salt Data Interfaces

There are two primary interfaces for extracting data from the system: *grains* [8] and *pillar* [13]. Data containing information about the systems managed by minions are acquired through grains. They contain information about the system properties such as OS type, RAM, CPU type, IP addresses, interfaces, e.t.c. Grains are automatically generated at the start of a minion or when requested through remote execution commands. On the other hand, the pillar data is generated by the Salt master and stores information about the properties of the whole infrastructure, which can be delivered to targeted minions. The pillar data is encrypted and therefore often used for storing sensitive information such as API tokens or passwords.

Event-driven Infrastructure

Another useful functionality offered by Salt is its event-driven infrastructure. This event-driven infrastructure is built around the *event* system and *reactor* system [5, 6]. Most actions from minions will generate events sent to the master's event bus. For example, whether a minion's key is accepted or rejected will return an authentication event to the master. In addition to the event system, Salt offers the capability of the Salt master reacting to these events through a reactor. Once a certain type of event has been sent the reactor is notified and will respond to it - i.e. start a state or runner.

Ansible

Similarly to Salt, Ansible is an open-source python-based CM tool and provides automation for application deployment, IT orchestration as well as software provisioning [9]. Ansible was created by Michael Dehaan in 2012 to solve the weaknesses present in pull-based CM tools. More specifically, at the time, leading CM tools like Chef and Puppet laid the responsibility of initiating communication on the agent instead of the main server. This resulted in less control over the system as a whole and also made development less straight-forward.

Ansible Architecture

Ansible follows an agent-less and push-based architecture where SSH is used for communication between the connected nodes (see Figure 2) - namely, a *control node* and it's *managed nodes* [4]. A control node is a machine with Ansible installed and can be used to issue commands as well as *playbooks* [11] (more on this later) to managed nodes (machines and/or servers that do not have Ansible installed). The agent-less architecture allows for an easier set up since agents or plugins are not required on the remote systems and managed nodes only require only SSH and Python.

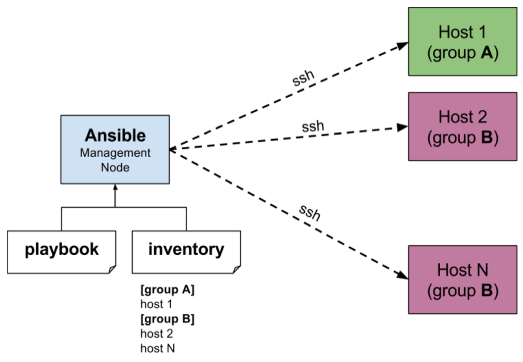

Figure 2. Ansible agentless architecture. [25]

Modules and Plugins

Modules [18] and *plugins* [19] are core to the functionality of Ansible. Modules are standalone python scripts that are used by Ansible API to perform specific work on managed nodes. They can be re-used inside an Ansible playbook (i.e. calling the command `ansible-playbook`) or invoked through the command line (i.e. through `ansible` command). Once a module has finished execution, results are sent back in the form of a JSON string and the module is removed. Ansible provides several pre-defined modules that can be found in [2]. Since modules are scripts it is also possible for the user to write their custom modules and can be written in any programming language if it is for local use. As opposed to modules, plugins are executed on the control node instead.

Playbooks

Playbooks can be seen as the instructions manuals that tells Ansible how to set up the remote system's configuration policies or how to perform defined tasks [11]. This is the core of Ansible's powerful push-based architecture, in which the user can easily write playbooks and "push" these instructions to the managed nodes simultaneously. Playbooks are stored in a files described with YAML format.

Inventory

Inventories contain information (i.e. IP address, databases, servers) on the nodes used with Ansible and can be described in either YAML or INI (stands for "initialization") format [10]. More specifically, inventories can be used to group managed nodes and thus allow the user to specify a group in the inventory as target for a playbook or command.

Comparison between Salt and Ansible

Salt and Ansible are both CM tools, each with their respective benefits and drawbacks in terms of *architecture*, *learning curve*, *syntax*, *customization* and *security*.

Architecture. The respective architecture of Salt and Ansible have their benefits and drawbacks. Ansible with its agentless architecture does not require additional software to be installed on the nodes and relies on SSH for communication. Thus, making Ansible easier to set up. Salt also allows agent-less architecture but encourages the user to use their master-agent architecture [12], which has the advantage of being fast and scalable (i.e. possible to have multiple master servers) [27]. Ansible does not provide any features for an event-driven architecture like Salt (i.e. reactor and beacons). Thus, eliminating the possibility of implementing long-running tasks.

Learning Curve. As previously stated, Ansible is easy to set up and start running. This is one of Ansible's strong selling points as a CM tool. On the other hand, Salt is CM tool rich with features, which could introduce a steeper learning curve than Ansible.

Security. In terms of security, Salt provides their own AES keys-pairs for their master-agent communication [3] whilst Ansible uses SSH to communicate between the connected nodes [9]. Salt also provides features that would allow for a more secure approach in securing data - i.e. being able to store data in an secure external data store through grains and pillars. On the other hand, Ansible does not provide this type of feature, this could lead to some users storing their private keys on an admin's laptop instead.

Syntax. Both CM tools uses YAML to describe their configuration files (i.e. playbooks for Ansible and States for Salt). This makes it relatively easy to learn how to set up the configuration files - with YAML being human-readable.

Customization. Both Salt and Ansible are based on python and can - with python - allow the user to implement customized modules.

Conclusion

Configuration management is one of the crucial cogwheels in the DevOps wheel, and with years several configuration management tools, like Salt and Ansible, have been spawned. The concept of configuration management was born from the practice of managing components necessary for a project within software development. Furthermore, adopting practices of DevOps within organizations meant that manually managing the configuration systems was no longer an accepted solution and should be automated instead. The benefits of adopting configuration management tools include scaling, deploying and recovering running applications with ease.

Salt and Ansible are competitors in the same market that tackle different issues of configuration management. Ansible was created due to the lack of push-based CM tools. Thus, it follows the idea of an agentless architecture with the intent of being user-friendly and quick to set up. On the other hand, Salt was born because CM tools could not deliver data connectivity fast enough. As such, Salt was implemented with a master-agent architecture that was feature-rich and provided high scalability as well as fast data delivery. However, this also makes the infrastructure more complex, which in turn makes it harder to pick up and learn. All in all, both Ansible and Salt are powerful tools that, if utilized to their strengths, can greatly enhance an organization's configuration management process.

References

[1] 2011. FLOSS Weekly 191 Salt. https://twit.tv/shows/floss-weekly/episodes/191 Accessed on 2020-04-05.

[2] 2020. All modules — Ansible Documentation. https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html Accessed on 2020-04-04.

[3] 2020. All modules — Ansible Documentation. https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html Accessed on 2020-04-04.

[4] 2020. Ansible architecture — Ansible Documentation. https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html Accessed on 2020-04-04.

[5] 2020. Event-Driven Infrastructure. https://docs.saltstack.com/en/getstarted/event/ Accessed on 2020-04-04.

[6] 2020. Event System. https://docs.saltstack.com/en/master/topics/event/events.html Accessed on 2020-04-04.

[7] 2020. execution modules. https://docs.saltstack.com/en/latest/ref/modules/all/index.html Accessed on 2020-04-04.

[8] 2020. Grains. https://docs.saltstack.com/en/latest/topics/grains/ Accessed on 2020-04-04.

[9] 2020. How Ansible Works | Ansible.com. https://www.ansible.com/overview/how-ansible-works Accessed on 2020-04-04.

[10] 2020. How to build your inventory — Ansible Documentation. https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#intro-inventory Accessed on 2020-04-04.

[11] 2020. Intro to Playbooks — Ansible Documentation. https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#about-playbooks Accessed on 2020-04-04.

[12] 2020. Introduction. https://docs.saltstack.com/en/getstarted/ssh/ Accessed on 2020-04-04.

[13] 2020. Pillar Walkthrough. https://docs.saltstack.com/en/latest/topics/tutorials/pillar.html Accessed on 2020-04-04.

[14] 2020. Returners. https://docs.saltstack.com/en/latest/ref/returners/ Accessed on 2020-04-04.

[15] 2020. Runners. https://docs.saltstack.com/en/getstarted/system/runners.html Accessed on 2020-04-04.

[16] 2020. SaltStack Components. https://docs.saltstack.com/en/getstarted/overview.html Accessed on 2020-04-05.

[17] 2020. States. https://docs.saltstack.com/en/getstarted/system/states.html Accessed on 2020-04-04.

[18] 2020. Working With Modules — Ansible Documentation. https://docs.ansible.com/ansible/latest/user_guide/modules.html Accessed on 2020-04-04.

[19] 2020. Working With Plugins — Ansible Documentation. https://docs.ansible.com/ansible/latest/plugins/plugins.html Accessed on

2020-04-04.
[20] 2020. Writing Execution Modules. https://docs.saltstack.com/en/latest/ref/modules/index.html Accessed on 2020-04-04.
[21] Lou Bichard. 2019. Configuration Management: What Is It and Why Is It Important? - Plutora. https://www.plutora.com/blog/configuration-management Accessed on 2020-04-04.
[22] Yevgeniy Brikman. 2016. Why we use Terraform and not Chef, Puppet, Ansible, SaltStack, or CloudFormation. https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c Accessed on 2020-04-04.
[23] Abhinav Kaiser. 2019. Role of Code Configuration Management in DevOps | Pluralsight | Pluralsight. https://www.pluralsight.com/guides/role-of-configuration-management-in-devops Accessed on 2020-04-04.
[24] Carlos Nuñez. 2017. Why Configuration Management and Provisioning are Different | ThoughtWorks. https://www.thoughtworks.com/insights/blog/why-configuration-management-and-provisioning-are-different Accessed on 2020-04-04.
[25] Jana Srikanth. [n. d.]. How To Setup Ansible Master-Slave and Install Apache Web Server -. https://blogs.tensult.com/2019/12/19/how-to-setup-ansible-master-slave-install-apache-web-server/ Accessed on 2020-04-21.
[26] Steve Strutt. 2018. Infrastructure as Code: Chef, Ansible, Puppet, or Terraform? | IBM. https://www.ibm.com/cloud/blog/chef-ansible-puppet-terraform Accessed on 2020-04-04.
[27] Paul Venezia. 2015. Puppet vs. Chef vs. Ansible vs. Salt | Network World. https://www.networkworld.com/article/2172097/puppet-vs--chef-vs--ansible-vs--salt.html Accessed on 2020-04-21.
[28] Stephen Watts. 2019. Configuration Management in DevOps – BMC Blogs. https://www.bmc.com/blogs/devops-configuration-management/ Accessed on 2020-04-04.