Comparative Study Between Popular Infrastructure as Code Tools

Joakim Olsson - joakiols@kth.se
Mikael Jafari - mjafari@kth.se

May 9, 2022

Contents

1 Introduction

As the complexity of software has grown exponentially over the years, the importance of automation and the need for efficiency has followed. The increasing complexity of software can make setting it up on servers more difficult and time-consuming. Automating this process can save companies a lot of time and as a result from that, a lot of money. The automation of the process also allows for more rigid checks that the software behaves as it is expected to, leading to increased reliability of services.

DevOps (Software development and IT operations) is a practice that deals with automation and integration in software development. It has become more prevalent over the years as collaboration in teams has become more complex with the goal of a high degree of efficiency and reliability for the software. This makes Infrastructure as Code (IaC) naturally part of DevOps, as a big part of IaC is used for automating configurations.

The need for efficient automation of infrastructure has resulted in a market of tools that provides solutions to this. As with most software, there isn't a perfect solution to every case, and a lot of the time the software has to be adapted and/or modified in some way for a specific use case. The same goes for the automation of infrastructure. Some IaC tools might be a better fit for some infrastructure as they may offer features that will be used, in another case an IaC tool is better fitted for the competence of a team.

There are several IaC tools out there that serve different purposes. In this essay, the focus will be on three different IaC tools: Ansible, Chef (formerly known as Progress Chef), and Salt (also known as SaltStack). These tools are among the 10 most common IaC tools in 2022 [9].

1.1 Problem statement

Knowing what tool to use for what purpose when it comes to infrastructure management and automation is not an easy task. The answer to what tool to use for what purpose can be difficult to answer even for an experienced software developer. This essay aims to answer the question of what the different tools are used for, describing their defining characteristics and comparing the features of each of the tools.

2 Ansible

Ansible is an open-source IT automation tool written in Python and Ruby that is used to automate different processes such as cloud provisioning, configuration management, and deployment to name a few [2]. It is designed for configuring multiple deployments systems and is orchestrated through Ansible Playbooks which are written in YAML. The approach that the creators of Ansible have gone for when it comes to writing configurations is being

as close to the English language as possible and has therefore opted to use YAML.

2.1 Architecture

Ansible consists of several parts working together. These parts have different responsibilities such as connecting, managing, and controlling nodes (machines). In figure 1 we see an example overview of how these parts could work together.

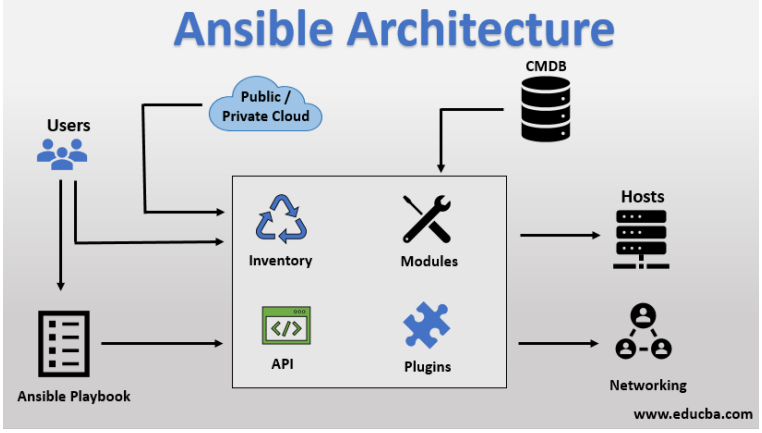

Figure 1: Example architectural overview of Ansible

2.1.1 Modules

Ansible consists of nodes and there are so-called Ansible Scripts which are the modules. The modules describe the desired state of the system and are executed over SSH. Although you can write your own modules, there already exists a collection of available modules that can be used. Any modules can be written in a language that can return JSON files [1].

2.1.2 Plugins

Plugins work as the body of Ansible as it is responsible for transforming data, logging the output and connecting to inventory, etc. As with modules, there already exist useful plugs that can be used and writing your own plugin must be written in Python [1].

2.1.3 Inventory

Ansible consists of multiple nodes and is managed using a list or group of lists known as inventory. These are defined in an INI or YAML format [1].

2.1.4 Playbooks

The Ansible Playbooks serve as a blueprint for automating the task allowing actions to be executed automatically. As modules execute tasks, multiple tasks are combined as a list of tasks to make a playbook and automatically execute against specified hosts [1].

2.2 Configuration example

Figure 2 is an example of a playbook execution consisting of two plays. The playbooks execute the plays in a top to bottom order.

This playbook contains two plays where the first one targets the web servers and the second one on the database server. The inventory, in this case, consists of an Apache web server and a MySQL database. These are all defined in the inventory file of the configuration. The playbook needs to specify the user for each play and because of the SSH execution of plays, this is listed with the *remote_user* field.

The first play that targets the web server verifies that it is running the latest version. For the second play, the configuration verifies that the database is using the latest version and that it is currently up and running.

```
- name: Update web servers
  hosts: webservers
  remote_user: root

    tasks:
    - name: Verify that apache is running the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest

- name: Update database servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Verify that mysql is running the latest version
    ansible.builtin.yum:
      name: mysql
      state: latest
  - name: Verify that mysql is currently running
    ansible.builtin.yum:
      name: mysql
      state: started
```

Figure 2: Example configuration for Ansible

3 Progress Chef

Progress Chef, renamed from Chef, is an open-source configuration management tool written in Ruby and Erlang [3]. The software uses the Ruby language for configuration and uses cooking terms for the different components used in configurations such as *recipes* and *cookbooks*. These components are used to streamline the configuration of company servers. The tool can be integrated with many popular cloud-based such as Amazon EC2, Microsoft Azure, and many others.

3.1 Architecture

User-written *recipes* which describes how server applications and utilities should be managed can be grouped into *cookbooks*. In the *recipes*, users can describe the state of resources. Such as which packages should be installed or which services should be running.

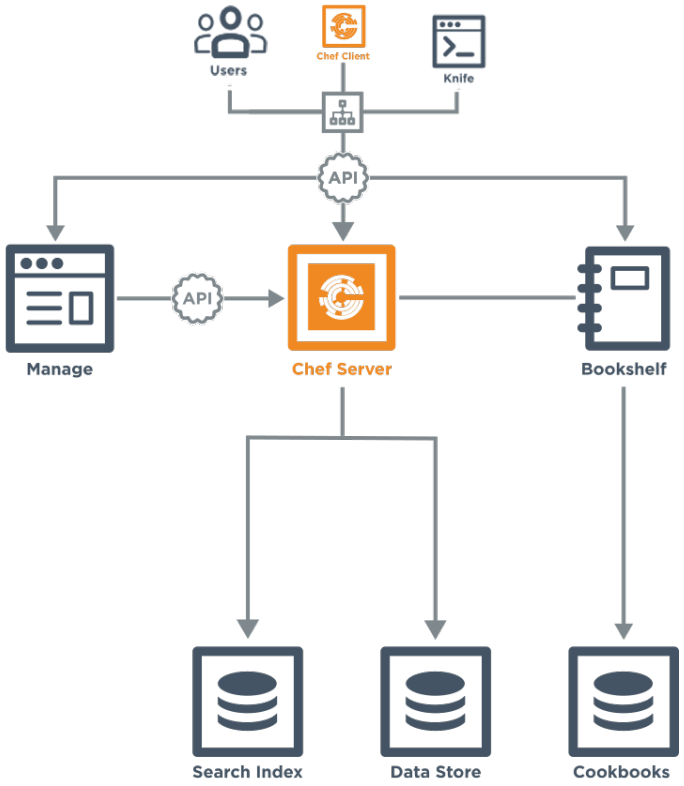

Figure 3: Example architectural overview of Chef in client/server mode

The Chef tool can be run in two different modes. The first mode is a classic client/server mode which an example overview of can be seen in figure 3, while the second one is called

"chef-solo". While running in the client/server mode the node which is running the client sends data to a Chef server running Elasticsearch which indexes the data and provides an API that Chef *recipes* can query for the configuration of the node. The "chef-solo" mode is quite similar to the client/server mode except that there is no server which the node communicates with. Instead, everything is run locally. This does however require that *cookbooks* be available locally on the device.

3.2 Configuration example

Recipe files for Chef are written in Ruby, which has the file extension .rb. Recipe files are the most fundamental configuration elements in the tool [4].

In figure 4, the *ruby_block* block declares a piece of Ruby code that is run during the execution phase of the tool. Anything that can be done with the Ruby language can be done in these recipes. This example Chef recipe will install an Apache web server, with the scripting languages PHP or Perl to be chosen at random.

```
package 'httpd' do
  action :install
end

ruby_block 'randomly_choose_language' do
  block do
    if Random.rand > 0.5
      node.run_state['scripting_language'] = 'php'
    else
      node.run_state['scripting_language'] = 'perl'
    end
  end
end

package 'scripting_language' do
  package_name lazy { node.run_state['scripting_language'] }
  action :install
end
```

Figure 4: Example configuration for Progress Chef

4 Salt

Salt (often referred to as SaltStack) is an open-source software tool written in Python that is used for configuration management, automation, provisioning, and orchestration [6]. It has a modular design with different modules that serve different purposes. The modularity

of Salt and the ability to modify its modules allow Salt to be adapted to fit the user's needs.

4.1 Architecture

A basic setup of Salt consists of a Salt master that manages at least one Salt minion. The master is a server and the minions are clients that are being managed by the tool. In addition to this more elements can be added to transform the setup into a more advanced one. A Salt proxy can be added which allows the execution of commands on devices that are unable to run the Salt minion service. Communication between minions and masters can also be done through SSH as an alternative. In figure 5, an example setup can be seen.

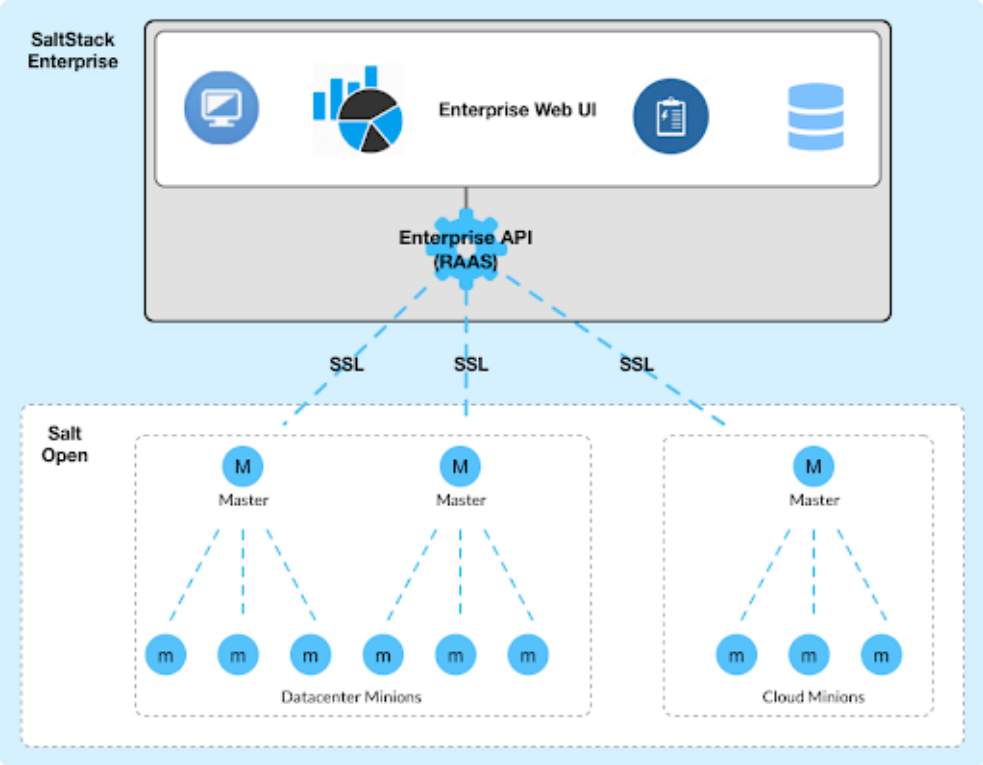

Figure 5: Example architectural overview of Salt

4.1.1 Modular design

The many modules of Salt can be separated into six different groups. *Execution modules* provide the main executive power of Salt, they contain the functions that are available for direct execution. *State modules* make up the back-end of the configuration management

system. They execute the code which interacts with the configuration of a target system. *Grains* is a system of modules that are used for gathering static information about a system. *Renderer modules* are used to display the information which a Salt system receives. *Returners* return generated information from a system to an arbitrary location. This arbitrary location is managed by the *Returners*. *Runners* contain convenience applications for Salt masters.

4.1.2 Open Event System

Salt uses an event for communication between Salt masters and minions where events can be seen, monitored, and evaluated by both masters and minions. Minions can subscribe to events that are published on the event system to be able to get information regarding jobs and results. This allows Salt to achieve a very fast speed of execution [5].

4.2 Configuration example

The configuration files for Salt are written but use the file extensions .sls to signify that it is a configuration file for Salt instead of the usual .yaml file extension that YAML uses.

Configuration files for a Salt setup are generally very long, so for the sake of clarity, an example has not been embedded in this document. Example configuration files can be found on the official website of the Salt project [8].

5 Discussion

While the three tools that have been covered are seen as IaC tools, they have different use cases. An overview of these tools can be seen in table 1. This table contains some high-level information about the tools that can be used to quickly see what sets these tools apart. The three tools that this paper covers are all used for configuration management, for example in servers. Even though the three tools have the same kind of broad use case, there are still differences between them that sets them apart.

When it comes to mutable infrastructure, the tools are all similar to each other. Usually, this constitutes that software updates on existing configurations can happen in place via a reboot for example. Mutable infrastructure gives the freedom of being able to customize servers to better fit the desired requirements. It also allows for easy adjustments to a server in the case of an emergency.

There are two different language paradigms that are used for the different tools and they are procedural and declarative. Both Ansible and Chef are procedural while Salt is declarative. Procedural is code written in steps on how the desired end state is achieved. Declarative on the other hand is where only the desired end state is described. This does have some advantages as it defers the work to the tool itself to figure out how that desired state is

IaC tools			
	Ansible	**Chef**	**Salt**
Created in	Python, Ruby	Ruby, Erlang	Python
Language	YAML	DSL (Ruby)	YAML
Language type	Procedural	Procedural	Declarative
Infrastructure mutability	Mutable	Mutable	Mutable
Usage	Configuration management	Configuration management	Configuration management

Table 1: Comparison table between the different tools

achieved. However, reduces the ability to control in more detail how the tool should achieve the desired state when compared to a tool that uses an imperative language.

Two of the languages, Ansible and Salt, use YAML as the language for their configuration files. Tools sharing the same language have the benefit of saving the user the time of having to learn a new tool-specific syntax when using a new tool making the swap between tools easier for the user. YAML is also used in many other cases for configuration purposes, such as in Kubernetes.

The two different modes that Chef can run makes it a good choice for users who in some case need to use the tool where the client can not communicate with a server. However, the configuration files are written in Ruby, a language that is not as popular as more common languages such as Python or Java [7] which might make finding developers knowledgeable in it more difficult. Ansible has a vast collection of available modules to be used, and if the module does not exist it can be written by the user themselves. This can make setting up the tool quite simple as already existing modules can be used. Since Ansible consists of many different parts that work together it can become confusing and convoluted as the scope of the project grows bigger. Salt's modularity can be seen as both a benefit and a drawback. In the hands of a skilled user, it is able to be lean and effective, but for a newcomer, it might be confusing to choose only the useful modules. This makes the tool have both a high skill floor but also a high skill ceiling.

Although this essay is of comparative nature, it is important to note that the tools are not mutually exclusive from each other. Each tool has its own perks and drawbacks which makes them useful in different scenarios. This makes it unrealistic to declare a definitive best tool for all uses.

6 Conclusion

IaC tools play an important role in software development. Automation of processes can save companies and people both money and time. Automating these processes allows for more complex systems to be built and maintained as the number of systems in a cluster increases.

The three IaC tools covered in this essay are just a few of the many IaC tools that exist today. Although there is no one-size-fits-all IaC tool for every problem, their strengths allow for usage in different scenarios and even to be used in conjunction with each other for optimal solutions. It is important to identify the needs that the tool should fulfill. In scenarios where a server is not always available to serve clients, Chef's solo mode fulfills that need well. If instead minimalism is of the essence, Salt and its modularity make great sense. Finally, if the user does not want to spend much time writing the configuration files, then Ansible might be the choice since it boasts a big module library that users can choose from.

There is no tool that is the best choice in every situation, it is up to the user to determine what needs they need the tool to fulfill and choose accordingly.

References

[1] Red Hat. *Architectural overview*. URL: https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html#id4.

[2] Red Hat. *How Ansible Works*. URL: https://www.ansible.com/overview/how-ansible-works.

[3] ProgressChef. *Platform Overview*. URL: https://docs.chef.io/platform_overview/.

[4] ProgressChef. *Recipes*. URL: https://docs.chef.io/recipes/.

[5] Salt Project. *Salt system architecture*. URL: https://docs.saltproject.io/en/3004/topics/salt_system_architecture.html.

[6] Salt project. *Salt overview*. URL: https://docs.saltproject.io/salt/user-guide/en/latest/topics/overview.html.

[7] PYPL. *PYPL PopularitY of Programming Language*. URL: https://pypl.github.io/PYPL.html.

[8] SaltDocs. *Configuration file examples*. URL: https://docs.saltproject.io/en/latest/ref/configuration/examples.html.

[9] N. Singh Gill. *Infrastructure as Code Tools to Boost Your Productivity in 2022*. URL: https://www.nexastack.com/blog/best-iac-tools.