Comparison Study Between Popular Infrastructure as Code Tools

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

April 21, 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 to automate configurations.

There are several IaC tools out there that serves different purposes. In this essay, the focus will be on four different IaC tools: Ansible, Terraform, Chef (formerly known as Progress Chef), and Salt (also known as SaltStack). These tools are among the 10 most common tools in 2022[7].

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, deployment etc[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).

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].

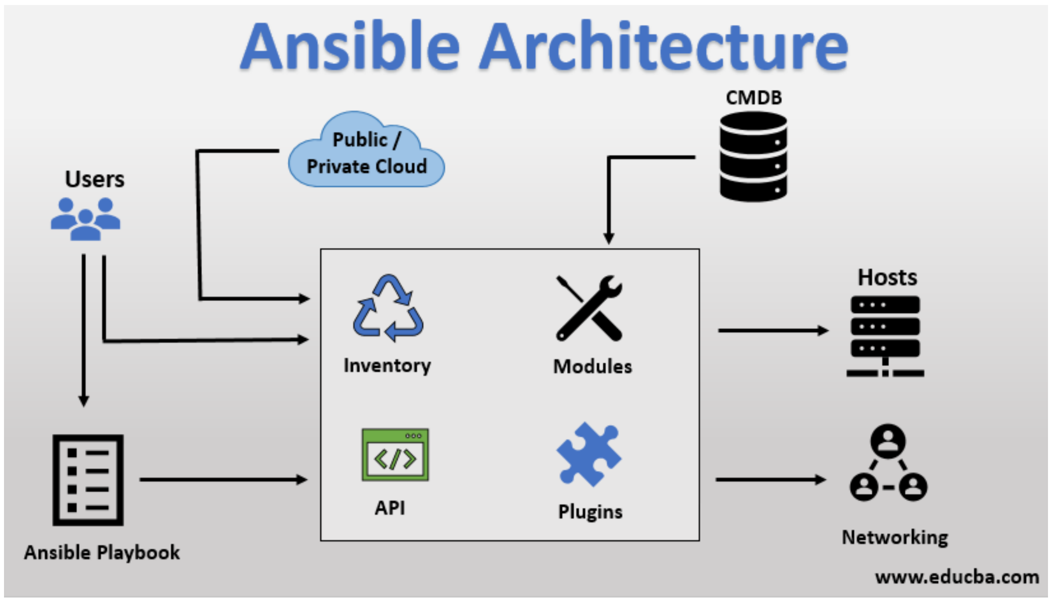

Figure 1: Architectural overview of Ansible

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

Below 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 Terraform

Terraform is an open-source software tool written in Go. It is created by HashiCorp and uses Hashicorps own configuration language called *HCL*, or can alternatively be configured using JSON[10]. Terraform primarily serves as an orchestration tool rather than configuration management, which is the process of creating the infrastructure rather than managing existing ones. More on the difference between these terms is found in the discussion section.

3.1 Architecture

Terraform consists of two main components called *Terraform Core* and *Terraform Plugins* which serves separate responsibilities. These two components depend on each other and therefore communicate with each other using *Remote Procedure Calls* (RPC).

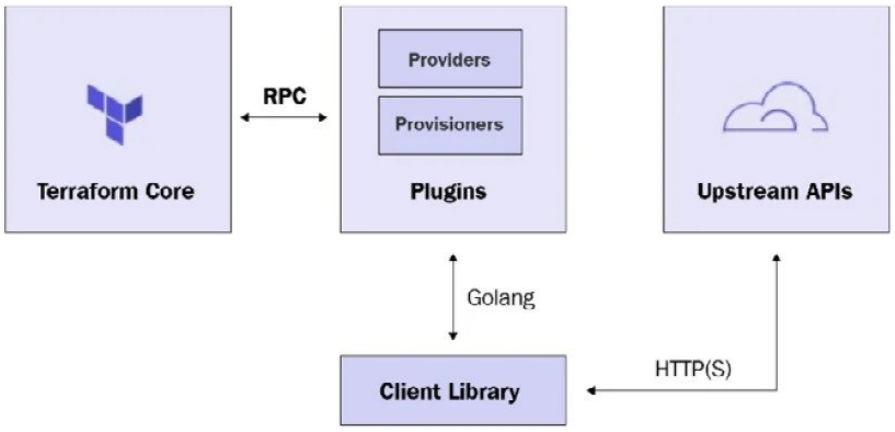

Figure 3: Architectural overview of Terraform

3.1.1 Terraform Core

Terraform Core is a compiled binary written in Go and is used as a *Command Line Interface (CLI)*. This CLI is responsible for executing the configuration, resource management, and communication with plugins. Therefore as with most CLIs, the Terraform core works as an entry point and is used when setting up Terraform for example[10].

3.1.2 Terraform plugins

Terraform plugins are written in Go and are executable plugins that are invoked by Terraform Core through RPC. The plugins specify an implementation from a cloud service like AWS, which are called providers[8]. The providers are responsible for domain specifics such as initialization of necessary libraries and authentication for an infrastructure provider like AWS etc. This example uses a Virtual Private Cloud (VPC) network from the Google Cloud Platform (GCP)[10].

3.2 Configuration example

All configuration files are ends in .tf or .tf.json, depending on the syntax choice by the user. The example below is written in HCL and is therefore ending with .tf.

The first block is a so-called terraform block and contains settings, which include the providers that will be used for the infrastructure. In this example, the provider used is GCP. This is followed by the authentication and specifications for the project such as the specific project id and location. The last block is a resource that defines the components of the infrastructure. This can be anything from virtual networks to compute instances such as VMs. There is room to use additional parameters for the resource such as specifying a Maximum Transmission Unit (MTU) for the network.

This configuration assumes that the user has set up a project on Google Cloud.

4 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.

4.1 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.

The Chef tool can be run in two different modes. The first mode is a classic client/server mode 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.

```
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "latest"
    }
  }
}

provider "google" {
  credentials: = file("<config_file >.json")
  project = "<project id>"
  region = "eu-west2"
  zone = "eu-central1"
}

resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}
```

Figure 4: Example configuration for Terraform

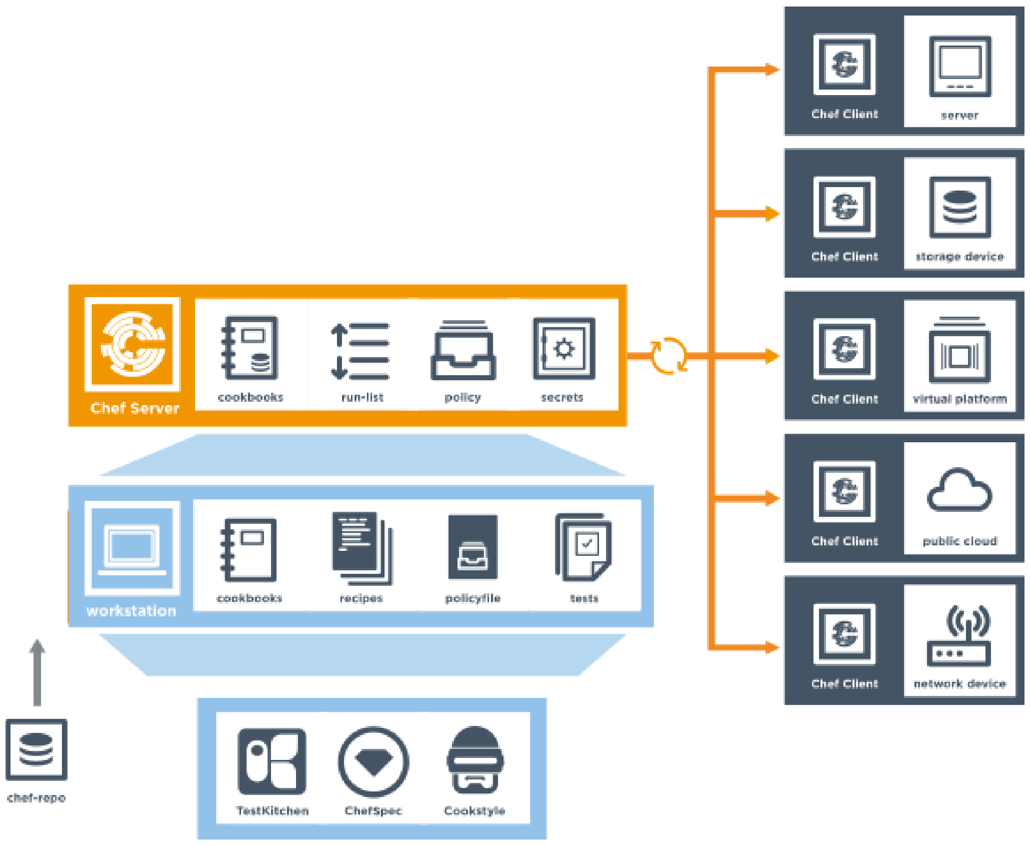

Figure 5: Architectural overview of Chef in client/server mode

4.2 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].

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 6: Example configuration for Progress Chef

5 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[5]. 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.

5.1 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.

5.1.1 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 backend 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* is 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* contains convenience applications for Salt masters.

5.1.2 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. Since the component for remote execution is the component that all other components are built upon it allows Salt to spread the load across resources[9].

5.2 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 [6].

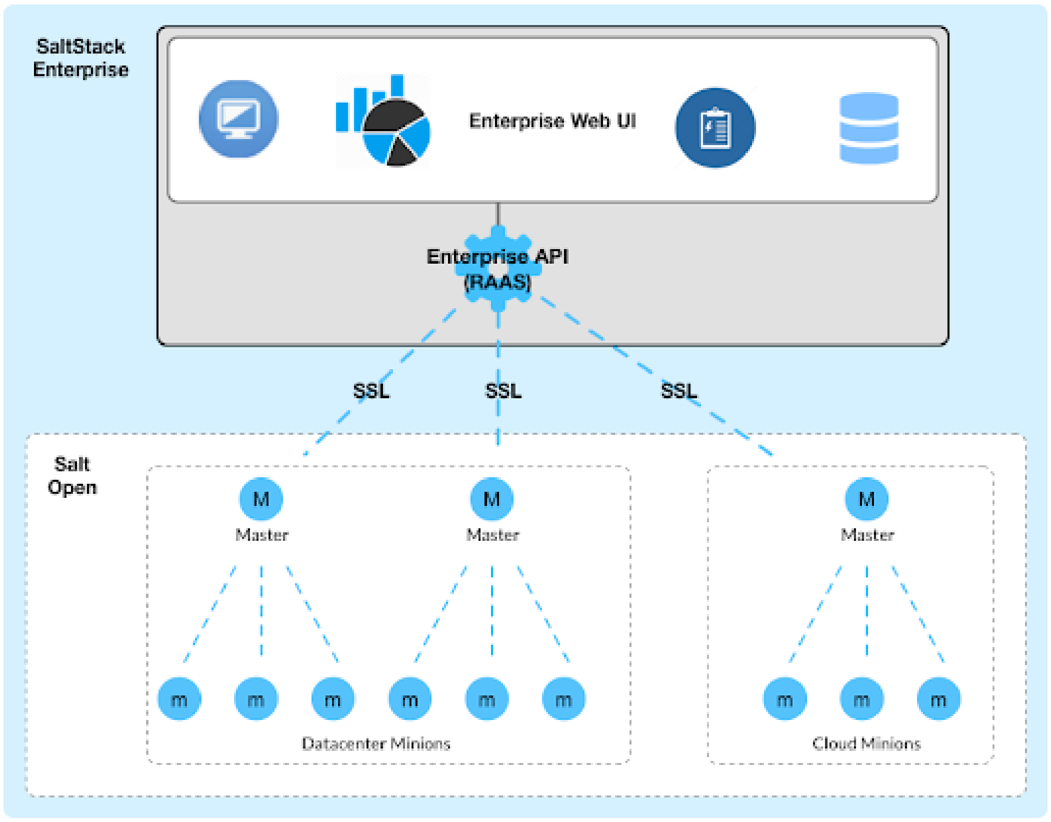

Figure 7: Architectural overview of Salt

IaC tools				
	Ansible	**Terraform**	**Chef**	**Salt**
Created in	Python, Ruby	Go	Ruby	Python
Language	YAML	HCL(or JSON)	DSL (Ruby)	YAML
Language type	Procedural	Declarative	Procedural	Declarative
Infrastructure mutability	Mutable	Immutable	Mutable	Mutable
Usage	Configuration management	Orchestration	Configuration management	Configuration management

Table 1: Comparison table between the different tools

6 Comparison table

7 Discussion

While the four tools that have been covered are seen as IaC tools, they have different use cases. Unlike the three other tools, Terraform is used for orchestration purposes. This is to coordinate multiple servers or systems to run the same tasks at the same time. The three other tools are used for configuration management. They are used to configure a server. Even though the three tools have the same kind of use case, there are still differences between them. Ansible, Salt, and Chef produce mutable configurations while Terraform produces immutable ones. Immutable configurations require the setup to be removed and then replaced by a new instance, hence it will never the modified after deployment.

Immutable configurations have the benefit of not being affected by the problems that can occur when modifying configurations over time, such as configuration drift. Although this makes deployments of configurations more predictable in their behavior, it is not suitable for configurations that are frequently reconfigured such as a firewall or other similar services.

When it comes to mutable infrastructure, Ansible and Chef 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 types that are used for the four different tools and they are procedural and declarative. Both Ansible and Chef are procedural while Terraform and Salt are declarative. Imperative is code written in steps on how the desired end state is achieved. Procedural 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 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. Depending on how skilled the user of the tool is, different language types might be preferred.

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.

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

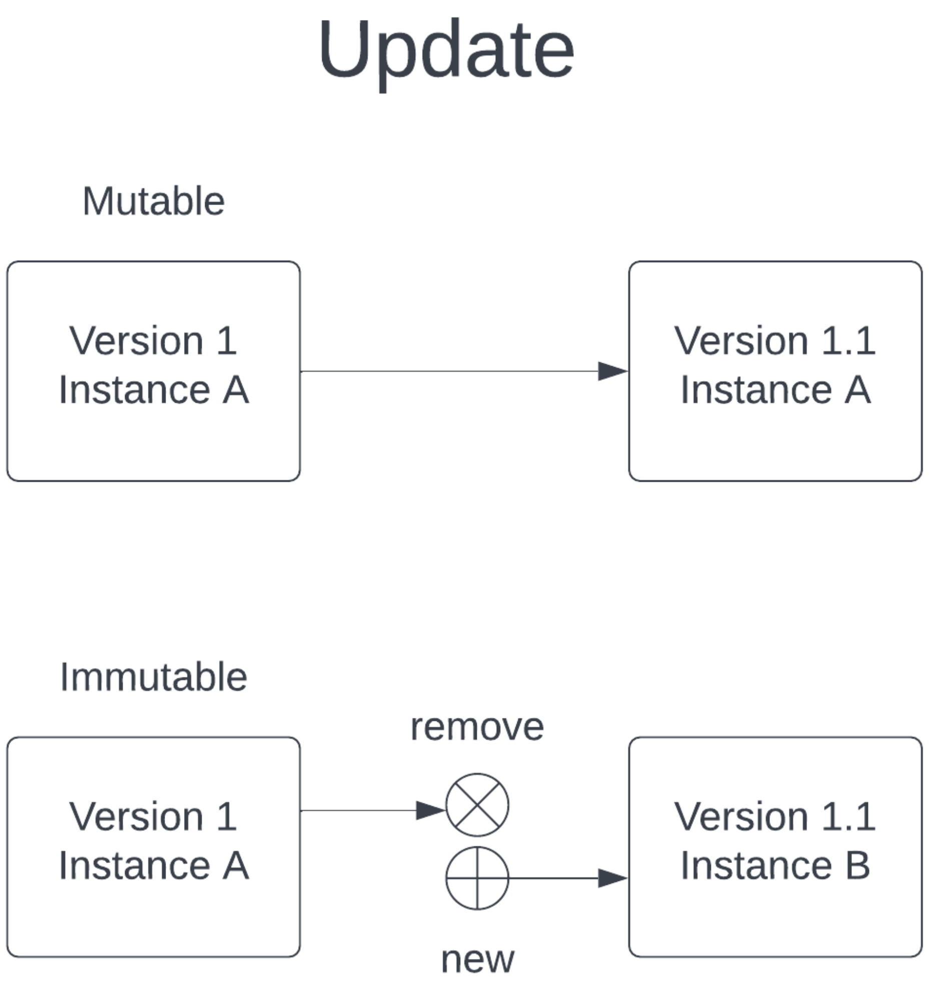

Figure 8: Immutable vs mutable infrastructure update

makes them useful in different scenarios. This makes it basically impossible to declare a definitive best tool for all uses.

8 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 four 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.

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 overview.* URL: https://docs.saltproject.io/salt/user-guide/en/latest/topics/overview.html.

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

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

[8] Terraform. *Plugin Development.* URL: https://www.terraform.io/plugin.

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

[10] Terraform. *What is Terraform.* URL: https://www.terraform.io/intro.