Infrastructure as Code with Ansible and Terraform

Kristian Alvarez Jörgensen
krijor@kth.se

Anders Sjöbom
asjobom@kth.se

April 2019

1 Introduction

> *"Infrastructure as code allowed us to perform 10x more builds without adding a single person to our team."*
>
> (Michael Knight, Build Engineer at Atlassian [1])

One of the key parts of DevOps is to enable a better and more smooth cooperation between development and operations. One practice that can be used to assist in this is Infrastructure as Code (IaC). It is is the process of managing and provisioning computer data centers through machine-readable definition files[2]. This means that the configuration of servers, virtual machines, containers etc. should be able to be performed automatically by using the machine-readable definition files that have been created.

By using IaC developers become more involved in setting up and defining the operations configuration and the operation team also get involved earlier in the development process.

Normally the IaC definiton is stored at the same place as the code. This means that it will be stored in a version control system like git and that all the changes to the infrastructure definition will be documented with version history and can be easily found in one single place.

A great benefit that comes from defining all of your infrastructure in some sort of recipe like the machine-readable definition files is that you can bounce back much faster from a crash or an unplanned restart of a service. This is possible because the infrastructure definition files are machine-readable, which means

they can be used to automatically set up new resources with the correct configuration. It allows for better scaling compared to having machines configured manually.

If you configure and maintain your servers manually you may often run into the problem of snow flakes [3]. It is when configuration changes have been applied to one server that is now in a unique state in comparison to other servers that should have the same configuration. Since the changes were manual it is hard to reproduce the specific setup of this so called snowflake server. With IaC and orchestration tools it is common to work with an immutable infrastructure that doesn't allow for snow flakes to exist since every change has to be done through the definition files and will be applied to the entire infrastructure in a deterministic way.

In this essay, we will be taking a deeper look at Terraform and Ansible, two widely used IaC platforms, and how they might fit the needs of various organisations.

2 Terraform

Terraform is an infrastructure as code tool developed and maintained by HashiCorp. It is was first released in 2014 as open-source with a Mozilla Public License [4]. Terraform is built with Go.

2.1 The basics

Terraform is used to define and provision datacenter infrastructure by creating machine-readable definition files. These files should be written with HashiCorps own language HCL (HashiCorp Configuration Language) [5] or optionally JSON. The HCL language has many similarities with JSON and YAML and is a declarative language. The files are used to describe the infrastructure you want. In listing 1 an HCL file can be seen. It would deploy an EC2 instance (virtual machine) with the specified image and resources in Amazon's US East datacenter.

```
provider "aws" {
  region = "us-east-1"
}
resource "aws_instance" "example" {
  ami = "ami-2d39803a"
  instance_type = "t2.micro"
}
```

Listing 1: Terraform Infrastructure HCL file

To actually perform a deployment Terraform uses a command-line interface tool [6] that checks your definition files and decides what steps that need to be taken to achieve the desired state that the user wants for the infrastructure. If a user runs the `terraform apply` command the definition will be deployed if all necessary authentication files for the cloud provider to be used exists. The command `terraform plan` can be used to perform a dry run to check what would happen if the apply command would be executed.

2.2 State

Each time a Terraform definition is applied a Terraform state file is created. This file is used to keep track of what resources have been created by Terraform. When a definition file is changed Terraform can check the state of the already created resources and decide what action to take to apply the changes. If there are major changes a restart might be needed for example. In listing 2 an example of a Terraform state file can be seen. It records all important things about the resources that have been created by Terraform, like instance id, ip address and such.

```
"aws_instance.example": {
  "type": "aws_instance",
  "primary": {
    "id": "i-66ba8957",
    "attributes": {
      "ami": "ami-2d39803a",
      "availability_zone": "us-east-1d",
      "id": "i-66ba8957",
      "instance_state": "running",
      "instance_type": "t2.micro",
      "network_interface_id": "eni-7c4fcf6e",
      "private_dns": "ip-172-31-53-99.ec2.internal",
      "private_ip": "172.31.53.99",
      "public_dns": "ec2-54-159-88-79.compute-1.amazonaws.com",
      "public_ip": "54.159.88.79",
      "subnet_id": "subnet-3b29db10"
    }
  }
}
```

Listing 2: Terraform State File

A problem that arises with state files is how to use them distributed with a team of developers. You are going to need a shared storage where you can lock the files so that there are no conflicts when multiple developers work on the same definition files.

The state files should not be stored in your version control system if there are secrets in them. This is an issue that yet remains unsolved by the Terraform developers [7]. What seems to be a good solution is to use a remote store such as Amazon S3 where you can manage access and encrypt your files. HashiCorp also has a paid service called Terraform Enterprise [8] where they provide a remote store and also allows for locking of the state files. However, if you don't want use Hashicorp's paid services you could for example use the open-source tool Terragrunt [9] that enables locking of state files on many different remote state stores.

2.3 Multiple environments

In many projects it is a great idea to use separate environments to separate development from production. You may even want more environments like a staging environment to be used before deployment where QA testers can go wild in a production-like environment. Without a codified production environment it might be hard to reproduce a large and complex production environment for the staging purpose.

However, when all of the setup needed for the production environment are saved in definition files with Terraform or some other IaC product it is very simple to create copies of a production environment for purposes like staging.

3 Ansible

Ansible is an open source agentless configuration management tool maintained by Red Hat that also supports orchestration. It runs on Python and configuration is written in the commonly used YAML-format.

3.1 The basics

Ansible configuration is organised by having **playbooks** that list a set of task to be preformed. Each task is specified by invoking a specific module, modules that are provided by Red Hat, the community or are custom-written in python. Ansible invokes the steps specified in the configuration on the **managed node** over the network using SSH (or similar), and can be deployed from any machine (known as a **control node**) with python installed. Sensitive data (passwords etc) can be managed using **Ansible Vault**, in order to avoid having secrets in the version control. [10]

Hosts are specified in the **Ansible inventory**, which is just a configuration file that contains all the targets and nodes reachable by ansible [11]. It's also

possible to use a dynamic inventory script if the number of nodes are not static (e.g. a cloud service like AWS) [12].

3.2 Modules

Modules make up the core of Ansible, and map to a specific task to be performed on the managed node. Modules should be idempotent - this means that they should avoid to make changes if the machine is already on the desired state. Once a task is performed, it will reply with a status on whether it was successful and if any changes where made to the system. A failure will for instance stop the execution of a playbook.

Modules are typically written in python, although for windows hosts they are written in powershell [13]. They are invoked in playbooks or roles using Yaml and a common syntax, so one does not need to know the internals of a module in order to use them.

3.3 Playbooks

Playbooks are the main way of defining configurations in Ansible. Each playbook has one or more plays targeting one or more hosts (or group of hosts) which contains a list of tasks (or a **role**). Finally, handlers can be triggered by tasks and are preformed after the play is done. Common sequences of tasks can be organised together in roles, which can then be reused across multiple different playbooks.

Listing 3 shows how a simple playbook could look like when setting up a web server. It targets the host group webservers (specified in an inventory file) and defines a few variables. The tasks are performed in sequence, and ensure that the latest version of Apache is installed, transfers over a configuration file and make sure that Apache is running. The second task has a handler that restarts Apache - this will only be invoked in the case that the config file changes.

3.4 Ansible in a DevOps world

As mentioned above with Terraform, Ansible and other IaC tools are ideal in order to set up identical yet isolated development and production environments. This means that together with common CD/CI tools, Ansible can be used to provision, configure, deploy and manage almost any pipeline. It could be for instance be run on containers, making the configuration portable across various different vendors (Docker, Rocket, LXC etc) and also configure the environments on which these containers are run.

```
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service:
      name: httpd
      state: started
  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```

Listing 3: Simple playbook example

Finally, the agentless architecture brings automation into areas where we typically do not find it, such as for provisioning, configuring and managing routers and other network devices. [14]

4 Orchestration vs Configuration Management

Terraform is an orchestration tool while Ansible is a configuration management tool that also has some orchestration features. [15]. In Terraform you describe the state of how you want your infrastructure to look like in a declarative way [16]. Terraform will then work to keep the environment in that order. With Ansible you rather procedurally describe the steps you want to be taken. So if you want to scale up you would have to specify that 5 servers should be created with Ansible, while you would just increase a number in Terraform.

Another large difference between Terraform and Ansible is that the latter being a configuration management tool it is built to work well with a mutable infrastructure paradigm [15]. This means that it runs well on existing servers

and rather updates or changes settings. Terraform is built to work with the immutable infrastructure paradigm where every change most often is the deployment of a new server. This can be compared to functional programming where every "change" to a variable actually returns a new variable.

5 Terraform and Ansible together?

Given that the tools are primarily designed for different tasks (Orchestration vs Configuration Management), the two tools can be used in tandem to compliment each others strengths and weaknesses. Ansible has a Terraform module for instance, which allows one to use Terraform from within a Playbook. We can also do the opposite and use Ansible as a configuration management tool for our servers from within Terraform (achievable by setting localhost as a target in our inventory and have each server call ansible on itself). [17]

The agentless architecture of Ansible also means that it can be used on almost any type of hardware without prior ansible-specific configuration, making ansible useful for much more than only configuring servers. It has for instance been used to manage over 15 000 network devices [14], which would be very hard to do with Terraform. Conversely, managing a large number of variable virtual hosts is a task that might be better suited for Terraform. An organisation with both needs might therefore see value in using both.

Although we have dived a bit into how these tools could be used together in conjunction, there are pitfalls with these approaches. Firstly, there is a question of complexity, since using two tools will undeniably require more knowledge from the developers that setup and maintain such a work flow. For smaller operations, going with the tool that best satisfies the biggest need would be a more sensible option.

6 Conclusions

Ansible and Terraform are powerful IaC tools with different architectures and philosophies, and therefore different strengths and weaknesses. The functional approach from Terraform is ideal for orchestrating a large number of services, while the procedural approach in Ansible is better suited for configuration. Ansible supports orchestration, but if these needs are complex then they are better handled by Terraform.

For large organizations with strong orchestration and configuration needs, the best solution might be to have these two tools working in tandem. This results in an additional overhead in terms of complexity, and so for smaller organisations or for organisations with unbalanced needs this overhead is likely not worth it.

Siding with one of the tools that best fits the organisations most important needs might therefore be preferable.

References

[1] (). DevOps: Breaking the Development-Operations barrier, Atlassian, [Online]. Available: https://www.atlassian.com/devops (visited on 03/25/2019).

[2] A. Wittig and M. Wittig, *Amazon web services in action*. Manning Publications Co., 2015.

[3] *Snowflakeserver*, https://martinfowler.com/bliki/SnowflakeServer.html, (Accessed on 04/28/2019),

[4] (). Terraform license, HashiCorp, [Online]. Available: https://github.com/hashicorp/terraform/blob/master/LICENSE (visited on 04/28/2019).

[5] *Hashicorp/hcl: Hcl is the hashicorp configuration language.* https://github.com/hashicorp/hcl, (Accessed on 04/28/2019),

[6] *An introduction to terraform – gruntwork*, https://blog.gruntwork.io/an-introduction-to-terraform-f17df9c6d180, (Accessed on 04/28/2019),

[7] *Storing sensitive values in state files · issue #516 · hashicorp/terraform*, https://github.com/hashicorp/terraform/issues/516, (Accessed on 04/28/2019),

[8] *Hashicorp terraform: Enterprise pricing, packages & features*, https://www.hashicorp.com/products/terraform/enterprise, (Accessed on 04/28/2019),

[9] *Gruntwork-io/terragrunt: Terragrunt is a thin wrapper for terraform that provides extra tools for working with multiple terraform modules.* https://github.com/gruntwork-io/terragrunt#work-with-multiple-aws-accounts, (Accessed on 04/28/2019),

[10] (2019). What is Ansible? Red Hat, [Online]. Available: https://www.ansible.com/resources/videos/quick-start-video (visited on 04/30/2019).

[11] (2019). Working with Inventory, Red Hat, [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html (visited on 04/30/2019).

[12] (2019). Working With Dynamic Inventory, Red Hat, [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html (visited on 04/30/2019).

[13] (2019). Windows module development walkthrough, Red Hat, [Online]. Available: https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_general_windows.html (visited on 04/30/2019).

[14] (Apr. 2018). Managing 15,000 network devices with Ansible, Red Hat Summit, [Online]. Available: https://www.youtube.com/watch?v=HtMeDbGEylU (visited on 04/29/2019).

[15] *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 04/28/2019),

[16] *Ansible vs. terraform: Fight! - linux academy blog*, https://linuxacademy.com/blog/devops/ansible-vs-terraform-fight/, (Accessed on 04/28/2019),

[17] (2019). Ansible and HashiCorp: Better Together, Hashicorp, [Online]. Available: https://www.youtube.com/watch?v=-gKTeT3BgHE (visited on 03/25/2019).