Imperative Approach and Declarative Approach in Automation

In this article we will have a quick glance at automation tools to compare two fundamental approaches of how to design automation processes. When it comes to choosing automation technologies, we have number of choices however we have to see what categories there are to understand the structure better and to work out which one(s) would serve our purpose best in an informed manner. Two main approaches to infrastructure as code are:

1. Imperative; also, often known as workflow-based or procedural-based approach. Imperative model focuses on how you do something.
2. Declarative; also, often known as model-based approach. Declarative model focuses on what you want, but not how to make something to happen.

We cannot say either one better or worse, hopefully at the end of this article we will be able to shortlist or pick which one is better for our current requirements.

**Imperative Approach:**

Imperative approach is closer to how computers run things. Mentality is reflection of programmer’s position that when writing a code programmer thinks that they need to tell the machine how it needs to do something. In one sense Imperative approach serves the machine.

**Declarative Approach:**

Declarative approach is closer to what the user wants from computers. Mentality is reflection of customer’s position that when receiving a result, the user thinks what the machine should give them something. In one sense Imperative approach serves the user.

There are many infrastructure automations tools available which are using infrastructure as code however we have shortlisted following continuous configuration automation tools for this article. We will not mention Amazon’s CloudFormation as it is not open source.

**Puppet:**

Puppet released in 2005 by Puppet and use both declarative and imperative approach. Written in C++ & Clojure since 4.0, Ruby. Puppet is open source, mutable infrastructure and declarative language.

**Chef:**

Chef released in 2009 by Chef and use both declarative and imperative approaches. Written in Ruby. Chef is open source, mutable infrastructure and procedural language.

**SaltStack:**

SaltStack released in 2011 by SaltStack and use both declarative and imperative approaches. Written in Python. SaltStack is open source, mutable infrastructure and declarative language.

**Ansible and Terraform:**

Let’s have a closer look at the latest released ones to compare them from configuration management, orchestration and DevOps points of view. Both Ansible and Terraform communities are growing fast.

**Ansible:**

Ansible is an open-source IT automation tool by RedHat. Written in Python, Ansible released in 2012. In short Ansible is open-source, agentless, mutable infrastructure and procedural language.

Ansible can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployment. Ansible can set up to manage over 100 remote machines in a single system. There are two types of machine in Ansible, Controlling machine and nodes. Ansible uses SSH protocol to deploy required software to nodes. Ansible uses playbooks for configuration management and multi-machine deployment system.

Playbooks can

1. Declare configuration.
2. Orchestrate steps of any manual ordered process, on number of machines, in a defined order.
3. Launch tasks synchronously or [asynchronously](https://docs.ansible.com/ansible/latest/user_guide/playbooks_async.html#playbooks-async).

Playbooks use YAML syntax. Please refer to <https://yaml.org/> if you are not familiar with YAML.

A playbook is composed of one or more ‘plays’ in an ordered list. The terms ‘playbook’ and ‘play’ are sports analogies. Each play executes one or more tasks. Each task calls an Ansible module. A playbook runs from top to bottom and executes tasks one by one in from top to bottom order. Each play defines two things: The managed nodes to target, using a pattern and at least one task to execute.

Please note following example playbook text. In this example, first play targets the web servers and the second play targets the database servers:

+++q+++

**---**

**-** name**:** update web servers

hosts**:** webservers

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

**-** name**:** update db servers

hosts**:** databases

remote\_user**:** root

tasks**:**

**-** name**:** ensure postgresql is at the latest version

yum**:**

name**:** postgresql

state**:** latest

**-** name**:** ensure that postgresql is started

service**:**

name**:** postgresql

state**:** started

+++uq+++

Advantages of using ansible are simple playbook syntax (in YAML) and it can be quickly installed and deployed due to agentless architecture.

Please refer to <https://docs.ansible.com/core.html> for full information.

**Terraform:**

Terraform is an open-source infrastructure as code software tool by HashiCorp. Written in Go, Terraform released in 2014. In short Terraform is open source, agentless, has immutable infrastructure and declarative language. Terraform cloud has free and paid versions.

Terraform is a tool to build, change and manage infrastructure. Terraform codifies cloud APIs into declarative configuration files. **Terraform** use a configuration language called the **HashiCorp Configuration Language (HCL)** which is human-readable. Terraform widely used to manage environments and automate deployments. Terraform is used to manage and inform infrastructure resources such as physical machines, virtual machines, and containers.

Please refer to <https://www.terraform.io/docs/index.html> for full information.

Infrastructure as code means the process of managing infrastructure in files without needing to manually make changes in command line interface. The files contain information for desired state of resources and Terraform automates creation and maintenance of the same. Once you go into your project folder and initialized Terraform ($ terraform init) you will be able to plan and apply resources configured in the files.

Terraform is platform agnostic. You can manage heterogenous environment.

Terraform has state management. Terraform creates desired resources as well as state files. When / if you make any changes in configuration files Terraform compares them with what actually exist at the time.

Terraform has a 'planning' step. Terraform generates an execution plan. When user uses terraform apply command, user will be prompted to review the changes proposed by the latest apply command and the user must confirm the changes otherwise Terraform will not apply the said changes. This will avoid any surprises when Terraform manipulates infrastructure. And the same is a very important tool to manage user confidence. With this function, disruptions to the environment by actions taken by Terraform can be avoided.

Terraform used in multi-tier applications, software demos, disposable environments, resource scheduler and multi cloud deployments. Please note we can manage Kubernetes resources with Terraform. In order to save some space here, we will skip full detailed command line scenario as the Terraform documentation page has comprehensive information. Please refer to <https://registry.terraform.io/providers/hashicorp/kubernetes/latest/docs/guides/getting-started> for full hands-on guidance.

**Can Ansible and terraform be used in same project?**

One of the important differences between imperative approach and declarative approach from a DevOps Engineer’s container orchestration point of view is that the imperative approach is useful when you want to set up the environment very quickly. If you want to generate YAML files from an environment already running you can do so using --dry-run command. Then you can use the YAML files to update the content and change arrangements of the cluster such as number of replica sets.

With declarative approach you have the infrastructure written in Infrastructure As A Code format. One of the important advantages of declarative approach is it simplifies configuration. Infrastructure files also can easily be modified to use for processes such as version control.

The short answer to the above question is yes. For instance, you can deploy entire infrastructure using Terraform and then you can use Ansible to deploy your applications on top of the servers you have just deployed with Terraform.

I believe this article is short enough so that it can fit in with readers’ busy schedules as a quick read. And at the same time, I hope the same is sufficiently informative that will be able to assist an informed decision-making process.

Best regards

Author:

M. Altun

03Mar2021, London

DevOps Engineer @ Finspire Technology