Why Ansible?

```
 ____________
< Greetings Reader >
 ------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

This small chapter will be covering some basic stuff about Ansible and a fast recipe to kickstart your ansible journey. As I used to say in most of my workshops and demos, Ansible was conceived as a simple and modular pseudo-language for infrastructure automation in the year 2012 by Ansible INC, and later acquired by Red Hat. It's open source and has a great and welcoming community.

First some etymology about Ansible:

The Ansible term was created by Ursula Le Guin more than 60 years ago and was massively adopted later by "Ender's Game" futuristic series by Orson Scott as a device for instant "light speed communications" across very long distances in near time or real time. So Ansible is a synonym of speed. Ansible is intended to be a fun project, easy to understand, easy to develop, and easy to adopt by any Sysadmin. Written in Python, the coolest language for non-dev specialists.

As the Red Hat defines, it has 3 major benefits:

Simple: All playbooks run in linear execution, any loops or any other recursive and complex syntaxes are supported but is encouraged to newcomers to avoid those. Your firsts playbooks (and more) will be simple and straightforward. Variables, loops, handlers, and a lot of stuff are supported as any language, but first playbooks should avoid any of those, and will not be covered in this chapter.

Agentless: One of the most demanding tasks and sources of attack surface are the annoying agents, like for example backup agents, which usually get outdated rather quickly. No one wants to add more processing or sleeping agents on their servers, so Ansible only uses SSH on Unix like environments, WINRM on Microsoft Servers, Netconf is used for networking boxes and last but not least APIs.

Powerful: As being developed by modules (and now collections), every brand or technology is developed separately from the main components of Ansible, giving great flexibility to test and create every module by themselves. Sorta like breaking the monolith into microservices, we break the automation tool in several (3387 at the time this was written) modules. Now Ansible is being delivered by collections, a new way of wrapping roles, modules and playbooks.

From my point of view, the main advantages of ansible are:

Pseudolanguage: It does not have a complex or different syntax. It uses YAML and JSON to interact with the human operator. You talk to ansible with YAML code, and ansible will answer with JSON, making your life easy. There is no need to parse the replies from ansible, killing the need for awk or grep. If you are not familiar with either JSON or YAML we will discuss those later.

FUN: Writing Ansible playbooks instead of scripts or another nanming is chosen to take out any technicism, to avoid overcome fear from non-dev IT specialists. Most IT Specialist usually automates with BASH, but its hard to parse commands, so when Ansible reply in JSON format, we will be able to get everything encoded as variables. Also, in some versions, playbooks will reply with *cowsay* . For any cow hater, you could always disable those in Ansible.cfg with ANSIBLE_NOCOWS=1

DUMB: As many people avoid IA or IA assisted technologies, Ansible is more like hiring extra muscle for your IT department, not replace your crew. Ansible does not make any decisions for you. Ansible does not get written by itself, or comes with pre written playbooks to automate your environment. The Good news is, there are plenty public repos with loads of Ansible playbooks, a great well mantained documentation, and there is a full galaxy of roles and collections (pun intended!).

Infra testing ready: A lot of IT specialists do not test their jobs, they usually make changes directly on production servers, making your infra prone to human error. Most of the reasons are time constraints or complexity. Automation and Ansible provides a "write once, run anywhere" solution for Sysadmin, or from now on SRE/DevOps Specialists working in Infra as Code automation.

Self documented: Ansible does not require external documentation if it is properly written, because is required for every playbook, that every task to be explained. This is very important both for the understanding of the Playbook, and later for further analysis and debugging, because each task name will be printed on the output when we run a Playbook. You might view some cows while running your playbook.

Now The Fun Begins!

> Disclaimer, all commands are tested on Fedora 34, also all commands are intented to run as user with sudo permissions.

Installation steps: First, you need to install Ansible in your Control Node or server. This is the host who will command all the automated hosts, or remote nodes, but for this example, we will run with localhost. Also let's install ansile-lint from Python Installer. More about lint and testing in the example below.

```
sudo dnf install ansible -y
```

```
pip3 install ansible-lint
```

Then we will be installing the collection for containers management (*containers.podman*) with the

ansible-galaxy command, this will gather for us a public collection.

```
ansible-galaxy collection install containers.podman
```

Writting Our first Ansible Playbook

Ansible could run commands from the terminal, but the most powerful way to interact with Ansible is using Playbooks. Playbooks are the scripts for Ansible, Written in YAML (YAML Ain't Markup Language). Also could be understood as Grocery store lists. First you have the directions and information for the whole market, or header, and then you have the tasks or the sections for every corner of the market, or plays and tasks.

> A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple 'plays' can orchestrate multi-machine deployments, running one play on your webservers, then another play on your database servers, then a third play on your network infrastructure, and so on.
>
> — How a Playbook works from Ansible Documentation

In other words, our playbook will be running in sequence. Our playbook will be running the following steps:

- Install podman on localhost
- Start podman service
- Pull a container from a registry, in this case, Docker Hub and Start the container on this host

Syntax: With your favorite IDE, create a file, in this case, we will name it test.yml. Every Playbook should start with three dash (---) and end with three dots (...) to identify the YML file as an Ansible playbook. Then we should start a header with the name, privilege escalation, and variables required for the whole playbook. In our case we will be working with localhost, and we will be granting privilege escalation with *become* .

```
---
- host: localhost
  become: yes
  tasks:
...
```

Then we need to start adding tasks for our playbook, remember, avoid tabs, always use two spaces to keep the indentation. Our first task will call the built in collection, and the module *ansible.builtin.yum* for package install. If you are on debian or another Linux Family you could use *ansible.builtin.package* or *ansible.builtin.apt* modules. This first task, has only two arguments, as you might find those on the ansible module documentation, the first one, is which package we will manage, and the second is which is the state desired. In our case, we need to install or check if podman is present or installed. Ansible idempotency will validate the desired state. Remember not

all modules are idempotent, like the *command* module. Keep it in mind for testing purposes.

```
    - name: Install podman in rpm distros
      ansible.builtin.yum:
        name: podman
        state: present
```

Now our second task will start podman service, not the fat and ugly daemon like cousin Docker.

```
    - name: start podman service
      service:
        name: podman
        state: started
```

Now our host is ready to spin up some new pods or containers. For this example, we will be pulling a grafana container image from Dockerhub, and bind the container port to the host port 3000.

```
    - name: Start grafana server as a container from dockerhub
      containers.podman.podman_container:
        name: container
        image: docker.io/grafana/grafana
        ports: 3000:3000
        state: started
```

Putting altogheter will look like this:

```
---
- hosts: localhost
  become: yes
  tasks:
    - name: Install podman in rpm distros
      ansible.builtin.yum:
        name: podman
        state: present
    - name: start podman service
      service:
        name: podman
        state: started
    - name: Run forest run
      containers.podman.podman_container:
        name: container
        image: docker.io/grafana/grafana
        ports: 3000:3000
        state: started
...
```

Now we have a proper playbook, as we talked about before, testing is important in every SRE/DevOps workflow.

Testing our Playbook Our first test is the static code analysis, we will use *ansible-lint*, this tool will check for linting errors, duplicated spaces, and basic styling errors. Another great usage for *ansible-lint* is to check with the latest version of your old playbooks run well with new Ansible versions. If we have luck *ansible-lint* will give us an empty output if everything is good to go.

```
ansible-lint test.yml
```

Dry Run Last but not least, we will run the playbook as a dry run, this will test connectivity and states for every task. Also, We will be seeing if every task will are able to run and reach our servers. The *ansible-playbook* command, comes bundled with the base Ansible package. We will be using two arguments. -K will allow Ansible to ask for a password to get escalation and -check for the dry run mode.

```
ansible-playbook test.yml -check -K
```

So if everything runs well, now is time for production deployment.

```
ansible-playbook test.yml -K
```

Now we might have our grafana server running on our localhost. We will be using curl to start a connection to the webservice.

```
[gabriel@fedora ~]$ curl localhost:3000
<a href="/login">Found</a>.
```

If we get the HTML reply, we have our podman service, and our pod running grafana on our localhost ready to be used with our web browser.

Sources: Wikipedia was used for names, dates and details about authors. Ansible-doc and Ansible documentation for technical details, playbook examples, and naming conventions. Red Hat's "Ansible For everyone" presentation used as a reference for vocabulary and tech jargon.

The first rough version was written locally on my pc in vscode with asciidoc and Ansible extension, and then uploaded to Github for version control. asciidoctor and asciidoctor-pdf were used to print to pdf. Grammarly web extension was lightly used to check for spelling or grammar errors.