Ansible Workshop - Exercises

Basics

Get to know Ansible and learn to write your first Ansible Playbooks.

2 - The Ansible Basics

Objective

In this exercise, we are going to explore the Ansible command line utility `ansible-inventory` to learn how to work with inventory files, using the utility `ansible` to run commands on hosts in the inventory file and using the `ansible-inventory` utility.
The goal is to familiarize yourself with some of the different cli tools Ansible provides and how it can be used to enrich your Ansible experience.

This exercise will cover

- Working with inventory files
- Locating and understanding an `ini` formatted inventory file
- Running commands on inventory groups with Ansible Ad-Hoc commands
- Listing modules and getting help when trying to use them

Guide

Step 1 - Check and create Ansible configuration

Ansible can be configured in multiple ways, changes can be made and used in a configuration file which will be searched for in the following order:

1. `ANSIBLE_CONFIG` (environment variable if set)
2. `ansible.cfg` (in the current directory)
3. `~/.ansible.cfg` (in the home directory)
4. `/etc/ansible/ansible.cfg`

Ansible will process the above list and use the first file found, all others are ignored. For additional information (and a possibility to create a *sample* file with all available options and explanation for every parameter), take a look at the documentation.

You will create all Ansible *content* in a separate folder (this makes it easy to *version-control* it if necessary).
On your control host **ansible-1**, create a directory called `ansible-files` in your home directory and change directories into it:

```
[student@ansible-1 ~]$ mkdir ansible-files
[student@ansible-1 ~]$ cd ansible-files/
```

In this folder, create the file `ansible.cfg`:

```
touch ansible.cfg
```

Add the following content to the file:

```
[defaults]
inventory = ~/lab_inventory/hosts
```

Warning

If you don't use a configuration file which specifies the inventory file, you will always have to provide the path to it, when running Ansible!

With the small configuration above, you will be able to do all exercises without having to provide the path to the inventory file. If you don't create the file with the `inventory` parameter, you have to use `-i ~/lab_inventory/hosts` with every playbook or ad-hoc run.

Run the following command to take a look at all parameters with the current value (default values are highlighted in *green*, changed values are *yellow*)

```
ansible-config dump
```

Use `:q` to exit.

Success

Using a small configuration file in every single Ansible project is highly recommended!

Step 2 - Check the managed nodes

The Ansible *master nodes* by default communicates via SSH with all *managed hosts*. As we are automating Linux hosts, this is fine and we need to make sure that we can reach every node with SSH.
If you intend to automate hosts that can't be reached with the default method, e.g. Windows hosts, network infrastructure nodes, firewall hosts and so on, you need to instruct Ansible to use another communication method. In most cases, this is very easy and only requires setting a certain variable. But, let's focus on automating Linux nodes first.

You can reach all your *managed nodes* (the hosts that you want to automate) with *password-less SSH*, you won't need to enter a password (or a user) when connecting to the nodes.

Try it out, SSH to `node1`:

```
[student1@ansible-1 ~]$ ssh node1
[ec2-user@node1 ~]$
```

As you can see, you are now the user `ec2-user` on `node1`. Leave `node1` again:

```
[ec-user@node1 ~]$ exit
[student1@ansible-1 ~]$
```

You can also connect to `node2` and `node3` with the same method. When you are finished, make sure you are back on your Ansible Control node (`ansible-1`), only here you can execute Ansible commands (as the Ansible binary is

only installed on the Controller, Ansible works *agent-less*).

Step 3 - Work with your Inventory

An inventory file is a text file that specifies the nodes that will be managed by the control machine. The nodes to be managed may include a list of hostnames and/or IP addresses of those nodes. The inventory file allows for nodes to be organized into groups by declaring a host group name within square brackets ([ ]).

To use the `ansible-inventory` command for host management, you need to provide an inventory file which defines a list of hosts to be managed from the control node.
In this lab, the inventory is provided by your instructor. The inventory file is an `ini` formatted file listing your hosts, sorted in groups, additionally providing some variables. It looks like:

```
[web]
node1 ansible_host=<X.X.X.X>
node2 ansible_host=<Y.Y.Y.Y>
node3 ansible_host=<Z.Z.Z.Z>

[control]
ansible-1 ansible_host=44.55.66.77
```

Ansible is already configured to use the inventory specific to your environment. We will show you in the next step how that is done. For now, we will execute some simple commands to work with the inventory.

To reference all the inventory hosts, you supply a pattern to the `ansible-inventory` command. The `--list` option can be useful for displaying all the hosts that are part of an inventory file including what groups they are associated with.

Ansible

If `--list` is too verbose, the option of `--graph` can be used to provide a more condensed version of `--list`.

Ansible

```
[student1@ansible-1 ~]$ ansible-inventory --graph
@all:
|--@control:
|  |--ansible-1
|--@ungrouped:
|--@web:
|  |--node1
|  |--node2
|  |--node3
```

Navigator

```
[student1@ansible-1 ~]$ ansible-navigator inventory --graph -m stdout
@all:
|--@control:
|  |--ansible-1
|--@ungrouped:
|--@web:
|  |--node1
|  |--node2
|  |--node3
```

We can clearly see that nodes: `node1`, `node2`, `node3` are part of the `web` group, while `ansible-1` is part of the `control` group.

An inventory file can contain a lot more information, it can organize your hosts in groups or define variables. In our example, the current inventory has the groups `web` and `control`.

Using the `ansible-inventory` command, we can also run commands that provide information only for one host or group. For example, give the following commands a try to see their output.

Ansible

```
[student@ansible-1 ~]$ ansible-inventory --graph web
[student@ansible-1 ~]$ ansible-inventory --graph control
[student@ansible-1 ~]$ ansible-inventory --host node1
```

Navigator

```
[student@ansible-1 ~]$ ansible-navigator inventory --graph web -m stdout
[student@ansible-1 ~]$ ansible-navigator inventory --graph control -m stdout
[student@ansible-1 ~]$ ansible-navigator inventory --host node1 -m stdout
```

Tip

The inventory can contain more data.

Step 4 - Use the inventory with ad-hoc commands

An Ansible ad hoc command uses the `ansible` command-line tool to automate a single task on one or more managed nodes. Ad hoc commands are quick and easy, but they are not reusable. So why learn about ad hoc commands first? Ad hoc commands demonstrate the simplicity and power of Ansible. The concepts you learn here will port over directly to the playbook language.

Ad hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad hoc command looks like this:

```
ansible [pattern] -m [module] -a "[module options]"
```

Ad hoc commands can be used perfectly to check if all hosts in your inventory are reachable. Ansible offers the *ping* module for that (this is not a real ICMP ping, though). Let's try to reach all hosts of the *web* group:

```
[student@ansible-1 ~]$ ansible web -m ping
node2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
node3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
node1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Success! All three nodes are reachable, we get a *pong* back, we proved that we can establish a SSH connection and that the node(s) have a usable Python interpreter.

Try to run the same ad hoc command against the *control* group.

```
[student@ansible-1 ~]$ ansible control -m ping
ansible-1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Let's play around with ad hoc commands a bit more. You can use every *module* that Ansible provides with ad hoc commands, we will learn more about *modules* later today. By default, Ansible will use the *command* module, you can send every linux command you want to all managed nodes, the arguments are provided with the `-a` parameter:

```
[student@ansible-1 ~]$ ansible web -m command -a "cat /etc/os-release"
node2 | CHANGED | rc=0 >>
NAME="Red Hat Enterprise Linux"
VERSION="8.5 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.5"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.5 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8::baseos"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterprise_linux/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.5
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.5"
node3 | CHANGED | rc=0 >>
NAME="Red Hat Enterprise Linux"
VERSION="8.5 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.5"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.5 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8::baseos"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterprise_linux/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.5
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.5"
node1 | CHANGED | rc=0 >>
NAME="Red Hat Enterprise Linux"
VERSION="8.5 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.5"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.5 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8::baseos"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterprise_linux/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.5
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.5"
```

You can shorten the command and leave out `-m command` as this module is used by default:

```
[student@ansible-1 ~]$ ansible control -a "uname -a"
ansible-1 | CHANGED | rc=0 >>
Linux ansible-1.example.com 4.18.0-348.12.2.el8_5.x86_64 #1 SMP Mon Jan 17 07:06:06 EST 2022
x86_64 x86_64 x86_64 GNU/Linux
```

Ad hoc command are very useful to gather information about your managed nodes, the *setup* module is used. Try that against one host alone (so you won't get overwhelmed with output):

```
[student@ansible-1 ~]$ ansible node1 -m setup
node1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.16.9.82"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a4:bff:fea5:6d70"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "10/16/2017",
        "ansible_bios_vendor": "Amazon EC2",
        "ansible_bios_version": "1.0",
...
```

You will get loads of useful information and you can use every bit as variables in your playbooks later on! We will use facts in a later exercise again.