3. Creating an inventory

3. Creating an Inventory

In chapter 2, you installed Ansible on your nodes. In this chapter, we'll look at what an inventory file is, and how to leverage the inventory file when you have a complex inventory of machines you need to interact with.

3.1 What is an Inventory?

In the configuration management, the tool you use needs to know which machine it should run on. This list of machines is called the inventory. Without an inventory, you have a set of playbooks that define the desires of the system. Once the inventory is defined, you can use patterns to select the hosts or groups you want to run Ansible against.

The default location of the inventory file is */etc/ansible/hosts.* Using this file is not recommended. You should maintain a different inventory file for each project you make. You can specify a different inventory file with the command: *ansible all –i <path>.*
The inventory file can be an INI file, a JSON file or a YAML file. The JSON file is only used when the inventory is dynamically created.

Depending on the plugins you have, you can create the inventory file in various ways or formats. The most common formats are INI and YAML. They can be as simple as a list of hostnames to run against. A basic INI file might look like this:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

The heading in brackets are the group names, these are used to define the hosts and deciding what hosts you are controlling and for what purpose.
Here is that same basic inventory file in YAML format:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

There are two default groups: *all* and *ungrouped*. The *all* group contains every host.
The *ungrouped* group contains any hosts that doesn't belong to another group besides from *all*. Every host belongs at least to two groups, namely to *all* and *ungrouped* or *all* and another group. The groups *all* and *ungrouped* are always present.

Each host can be put in more than one group. If there are a lot of hosts with a similar pattern, you can add the hosts as a range, as shown here:

```
host[1:3].example.com
```

This is equivalent to:

```
host1.example.com
host2.example.com
host3.example.com
```

These ranges supports leading zeros ([01:03]) and alphabetic ranges ([a:z]). Only the range [a:z] is supported, you can't specify a range [aa:zz]. If you need to define a range like this you need to use two ranges [a:z][a:z].
There can also be a third parameter specified, this parameter is optional and is called *step*:

```
host[min:max:step].example.com
```

This parameter allows the user to specify the increment between the hosts. It would look like this:

```
host[1:6:2].example.com
```

This is equivalent to:

```
host1.example.com
host3.example.com
host5.example.com
```

3.2 Adding variables to the inventory

You can store variables that are related to hosts in your inventory. These variables can be added directly to the hosts or groups in your inventory file. A variable can be easily assigned to a single host, then use it in your Playbook later. In INI it would look like this:

```
[Delft]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

In YAML it would look like this:

```
Delft:
   host1:
       http_port: 80
       maxRequestsPerChild: 808
   host2:
       http_port: 303
       maxRequestsPerChild: 909
```

if you run SSH on a non-standard port, you can specify that certain port in your inventory file. To do this you can add a colon followed with the port number at the end of your hostname:

```
badwolf.example.com:5309
```

> **Note**
>
> If you use non-standard SSH ports in your SSH configuration file, the openssh connection will find and use them, but the paramiko connection will not.

In your inventory you can also define aliases. In INI:

```
jumper ansible_port=5555 ansible_host=192.0.2.50
```

In YAML:

```
...
  hosts:
    jumper:
      ansible_port: 5555
      ansible_host: 192.0.2.50
```

In the example above, Ansible will be running against the host alias “jumper”. This will connect to 192.0.2.50 on port 5555. This only works when the host has a static IP or when it is connected through tunnels.

> **Note**
>
> Values in the INI format using the key=value syntax are interpreted differently depending on where they are declared:
>
> - Declared in line with the host. The INI values are interpreted as Python literal structures.
> - Declared in the :vars section, the INI values are interpreted as strings.
> - If the value is set in the INI inventory, it must be a certain type (for example, a string or a Boolean).
>
> Try to use YAML format for your inventory to avoid confusion. A YAML inventory will processes variables consistently and correctly.

The variables can also be shared within a group, you can apply that variable to an entire group.
In INI:

```
[Delft]
host1
host2

[Delft:vars]
ntp_server=ntp.Delft.example.com
proxy=proxy.Delft.example.com
```

In YAML:

```
Delft:
  hosts:
    host1:
    host2:
  vars:
    ntp_server: ntp.Delft.example.com
    proxy: proxy.Delft.example.com
```

The group variables are an easy way to apply the variables to multiple hosts. But if a host is a member of different groups and you assign different variables to the same host in different groups, Ansible will choose which variable they use based on the internal rules.

It is possible to make a group out of another group, this can be done using *:children* suffix for INI or *children:* entry for YAML. The variables can be added with *:vars* or *vars:.* The child groups have a couple of properties to note:

- If a host is member of a child group it is automatically member of the parent group.
- The variable of a child group will have a higher precedence than the parent group variables.
- A group can have multiple parents and children, but there can't be circular dependencies.
- A host can be member of multiple groups, but there is only one instance of the host.

3.3 Usage

3.3.1 Organizing host and group variables

You can store every variable in the main inventory file, but sorting them may help to organize your variable values. The host and group files must use one specific syntax, YAML.
Ansible gets the host and group variable files by searching a path that is relative to the inventory file. If the inventory file is located at */etc/ansible/hosts* and contains a host named "soccer" and this host belongs to two groups, "Delft" and "webservers". The host will use the variables in the YAML files at the locations:

/etc/ansible/group_vars/Delft
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/soccer

For example, if you group hosts in your inventory by datacentre, and each datacentre use its own NTP and database. You can create a file named */etc/ansible/group_vars/Delft* to store the variables for the Delft group.
It is also possible to create directories named after your groups or hosts. Ansible will read these directories in an alphabetic order. All hosts of the groups have variables defined in different files. This can be helpful to keep your variables organized when a single file gets too big. You can also add these directories to your playbook. But if you load inventory files from your playbook directory and inventory directory, the variables in the playbook directory will override the variables in the inventory directory.

3.3.2 Merging variables

By default, the variables are merged to the specific host before the play is run. This will keep Ansible focused on the hosts and tasks. The order of merging variables is (from lowest to highest):

- The *all* group
- Parent group
- Child group
- Host

Ansible merges groups by default alphabetically, and the last group overrides the previous groups. For example, if you have an *a_group* that will merge with a *b_group*. The *b_group vars* that matches will override the ones in the *a_group.*
This behaviour setting can be changed by changing the group variable *ansible_group_priority*. The larger the number you set, the later it will be merged, so you give it a higher priority. This is default 1. The *ansible_group_priority* can only be set in the inventory source.

3.3.3 Multiple inventories

If you want to target multiple inventories at the same time, you can give multiple inventory parameters from the command line or you can configure *ANSIBLE_INVENTORY*. Target two sources from the command line can be done like this:

```
ansible-playbook get_logs.yml -i staging -i production
```

If there are variables that conflict in the inventories, it will be resolved according to the rules described in *3.4 merging variables*.
You can also pass a directory to Ansible. Ansible will read every file in that directory as an inventory and merge them together. The directory would look like this:

```
inventory
|openstack.yml                          # configure inventory plugin
|dynamic-inventory.py                   # add additional hosts with dynamic
                                             inventory script
|static-inventory                       # add static hosts and groups
|group_vars
|  |all.yml                                # assign variables to all hosts
```

You can target this directory with the following command:

```
ansible-playbook  example.yml -i inventory
```

the inventories are merged based on alphabetic order according to the filenames. This can be controlled by adding prefixes to the files:

```
inventory
|01-openstack.yml                          # configure inventory plugin
|02-dynamic-inventory.py                   # add additional hosts with dynamic
                                             inventory script
|03-static-inventory                       # add static hosts and groups
|group_vars
|  |all.yml                                # assign variables to all hosts
```

This means that the playbook will be run with number 3. This is because *01-openstack.yml* defines *myvar = 1* for the group all, *02-dynamic-inventory.py* defines *myvar = 2*, and *03-static-inventory* defines *myvar = 3*.