Ansible

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Ansible’s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans–even those not familiar with the program.

We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.

Ansible manages machines in an agent-less manner. There is never a question of how to upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled. Because OpenSSH is one of the most peer-reviewed open source components, security exposure is greatly reduced. Ansible is decentralized–it relies on your existing OS credentials to control access to remote machines. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

#================================================================================

Ansible concepts

These concepts are common to all uses of Ansible. You need to understand them to use Ansible for any kind of automation. This basic introduction provides the background you need to follow the rest of the User Guide.

Control node

Managed nodes

Inventory

Modules

Tasks

Playbooks

#===============================================================================

Control node

Any machine with Ansible installed. You can run commands and playbooks, invoking /usr/bin/ansible or /usr/bin/ansible-playbook, from any control node. You can use any computer that has Python installed on it as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node. You can have multiple control nodes.

Managed nodes

The network devices (and/or servers) you manage with Ansible. Managed nodes are also sometimes called “hosts”. Ansible is not installed on managed nodes.

Inventory

A list of managed nodes. An inventory file is also sometimes called a “hostfile”. Your inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling. To learn more about inventory, see the Working with Inventory section.

Modules

The units of code Ansible executes. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. You can invoke a single module with a task, or invoke several different modules in a playbook. For an idea of how many modules Ansible includes, take a look at the list of all modules.

Ex. command, scripts, yum, services etc.

Tasks

The units of action in Ansible. You can execute a single task once with an ad-hoc command.

Playbooks

Ordered lists/set of tasks, saved so you can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in YAML and are easy to read, write, share and understand. To learn more about playbooks, see About Playbooks.

#================================================================================

Ansible Setup

Basics / What Will Be Installed

Ansible by default manages machines over the SSH protocol.

Once Ansible is installed, it will not add a database, and there will be no daemons to start or keep running. You only need to install it on one machine (which could easily be a laptop) and it can manage an entire fleet of remote machines from that central point. When Ansible manages remote machines, it does not leave software installed or running on them, so there’s no real question about how to upgrade Ansible when moving to a new version.

Control Node Requirements

Currently Ansible can be run from any machine with Python 2 (version 2.7) or Python 3 (versions 3.5 and higher) installed. Windows is not supported for the control node.

This includes Red Hat, Debian, CentOS, macOS, any of the BSDs, and so on.

When choosing a control node, bear in mind that any management system benefits from being run near the machines being managed. If you are running Ansible in a cloud, consider running it from a machine inside that cloud. In most cases this will work better than on the open Internet.

Managed Node Requirements

On the managed nodes, you need a way to communicate, which is normally ssh. By default this uses sftp. If that’s not available, you can switch to scp in ansible.cfg. You also need Python 2 (version 2.6 or later) or Python 3 (version 3.5 or later).

On RHEL and CentOS:

$ sudo yum install ansible

$ ansible –version

**Getting the configuration**

If installing Ansible from a package manager, the latest ansible.cfg file should be present in /etc/ansible, possibly as a .rpmnew file (or other) as appropriate in the case of updates.

Ansible Inventory

Create a inventory file

$vim my\_invent.txt

Populate below data in it .

child1 ansible\_host=192.168.31.169 ansible\_ssh\_pass=redhat

save and quit the file and run the below command.

$ansible child1 -m ping -i test-child.yml

Output:

child1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

Now again entry the below line ,

Child2 ansible\_host=192.168.31.168 ansible\_ssh\_pass=redhat

save and quit the file and run the below command.

$ansible child1,child2 -m ping -i test-child.yml

Output:

child1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

child2 | SUCCESS => {

"changed": false,

"ping": "pong"

}

Note:

ansible\_host > remote host ip or name

ansible\_ssh\_pass > password for your remote host if it’s linux

ansible\_connection > ssh(linux) , winrm(windows)

ansible\_user > User name

ansible\_password > password for your remote host if it’s windows

YAML (YET ANOTHER MARKUP LANGUAGE)

Ansible playbook is written in YAML.

Yaml is used to represent the data, in below case it’s server data.

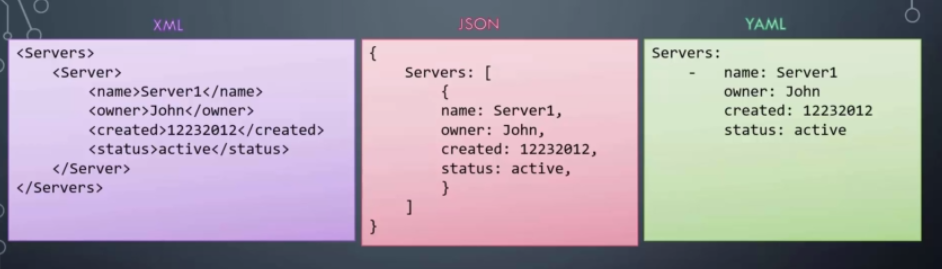
Data-Structure:

Dictionary (Unordered collection)

Dictionary of dictionary

List (Ordered Collection)

List of dictionaries



If you take the data in its simplest form such as Key Value pair.

This is how you will define it in YAML, key and value separated by a colon(:).

fruit: apple

vegetable: carrot

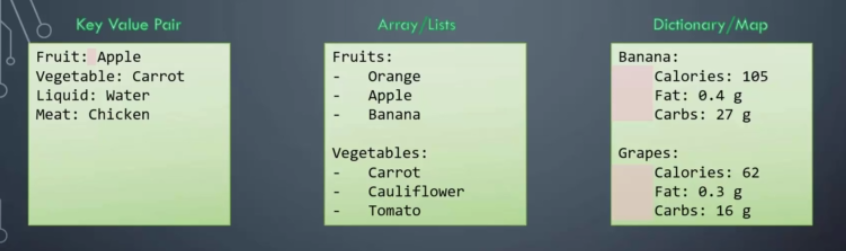
liquid: water

meat: chicken

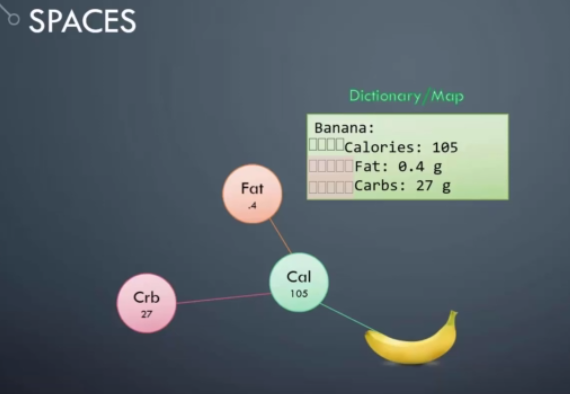
IT: ansible

Remember you must have a space followed by a colon,Differentiating the key and the value.

Let's take a look at how an array,Dictionary is represented.



Here we have a dictionary representing the nutrition information of banana.



The total amount of calories, fat and carbs are shown.

Notice the number of spaces before each property that indicates these Key Value pairs fall within banana.

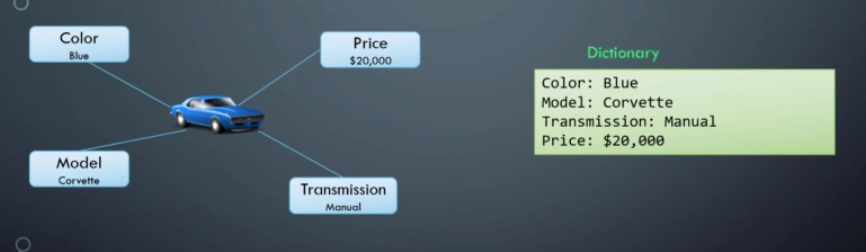
But what if we had extra spaces for fat and carbs then they will fall under calories and thus become properties of calories.Which doesn't make any sense.

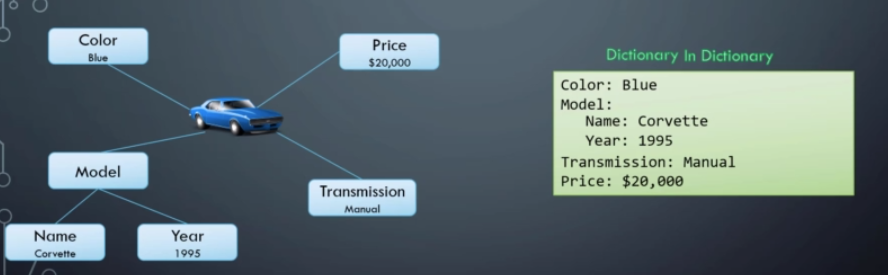
This will result in a syntax error which will tell you that mapping values are not allowed here, because calories already have a value set which is 105.

You can either set a direct value or a hash map.

You cannot have both.So the number of spaces before each property is key in YAML.

You must ensure they are in the right form to represent your data correctly.





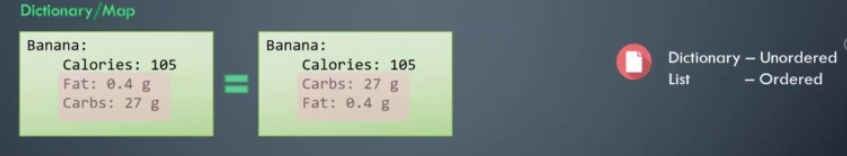




Dictionary is an unordered collection whereas lists are ordered collection.

So what does that mean.

The two dictionaries that you see here have the same properties for banana.



However, you can see that the order of properties fat and carbs do not match. In the first dictionary,fat is defined before carbs and in the second dictionary carbs comes first followed by fat.

But that doesn't really matter.

The properties can be defined in any order. But the two dictionaries will still be the same as long as the values of each property match. This is not the same for a list or arrays. Arrays are ordered collection.

So the order of items matter. The two-list shown are not the same because apple and banana are at different positions.

This is something to keep in mind while working with data structures.

Also remember any line beginning with a hash is automatically ignored and considered as a comment.

Ansible-playbook

[redhat@master1 ansible]$ ansible all -m ping -i test-child.yml

child1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

child2 | SUCCESS => {

"changed": false,

"ping": "pong"

}