Ansible

Ansible is a very powerful open source automation tool. It comes in the category of Configuration Management tools like Puppet, Chef, Saltstack etc. Its simplest among all the configuration management tool in terms of its easy to read & write approach and Masterless/serverless architecture.

Being simple in use it can handle most complex of the task when it comes to Orchestration, where you run automation tasks in chain and order on several different servers and devices.

1. Configuration Management :

In simple terms, tools that manages configuration of IT infrastructures like OS, network devices, Application, softwares and Cloud computing services are called as Configuration Management tools.

Configuration of IT Infrastructure like Softwares & OS changes happens every now and then. We add, remove and update them for good. Doing such changes to hundreds of servers and devices is very time consuming and error prone task if you do it manually.

These tools help you manage and automate all that changes with ease and from a centralised place.

Some features :

Centralised configuration :

Configuration of various servers and devices are managed from a central server. Like you would be managing webservers, dbservers or switches all of their configurations like softwares, patches, config files etc can be setup in the central server. When the node wants to update the latest changes, it can fetch from the server or pushed to the node from server.

Enforcement :

Configuration enforcement may be the single most important feature of a configuration management tool. By running regularly and ensuring the machine is configured to the desired state, configuration management tools prevent configuration to go out of sync. Configuration drift can happen in a variety of ways: Package updates, live debugging, "helpful" co-workers, etc. Whatever the cause, being able to say with confidence, "This is how this machine is configured," is a great way to shorten incident resolution time and reduce surprises.

Abstraction:

Few sysadmins maintain completely homogeneous environments. Even if you're an all-Linux shop, you probably have multiple distros that you support, or at least multiple versions of a distro. With configuration management tools, many of the operating-system-specific implementations of a configuration are abstracted away for you. The same configuration file can be used to manage, for example, the installation of Apache HTTPD on both Red Hat and Ubuntu systems.

Version control friendly:

Of course, the best way to enable cooperation is to have everything in a version control system. All of the tools listed below use some form of text for configuration. This means you can take advantage of the benefits of your favourite version control system.

Replication:

Configuration management makes it easy to replicate environments with the exact same software and configurations. This enables you to effectively build a multistage ecosystem, with production, development, and testing servers.

Some Configuration Management Tools

- Puppet
- Chef
- Ansible
- Saltstack
- CFEngine

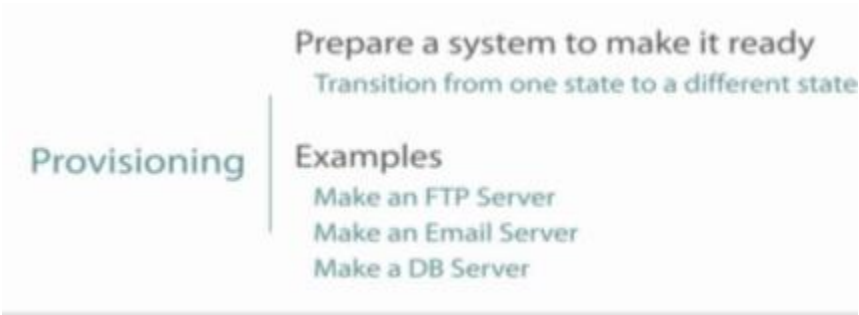
Some Terminologies

Change Management :

It is process when any specific configuration of the machine or software is changed. A typical CM process in IT industry goes through a approval process from customer or higher management, of course we are talking about production systems. A single change on 1, 10 or 100's of servers has to be done very precisely and effectively. For example, upgrading a software package which has a bug on hundreds of servers or restarting a service like webservice to take effect of any new changes.

Provisioning :

In general, provisioning means "providing" or making something available in the IT infrastructure. For example, provisioning a web server means installing and setting up web service softwares and its configuration on a OS. Provisioning a cloud instance means creating a virtual machine on the cloud.



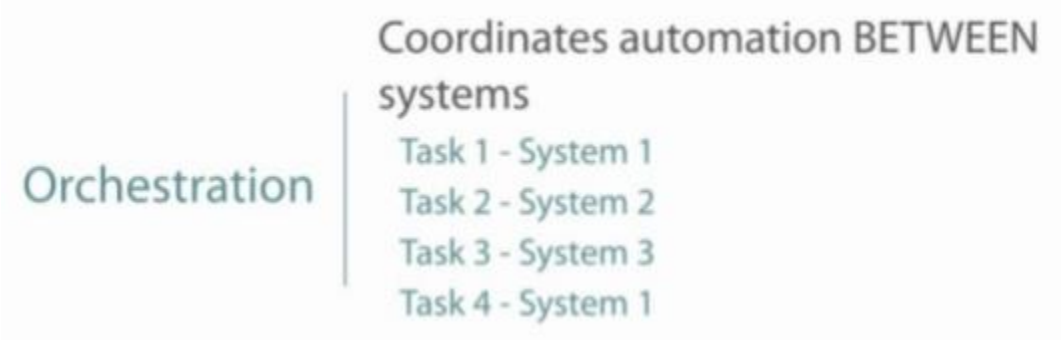
Orchestration :

Automation is when we group list of tasks and execute them in top to down order through a script generally like installing webserver through a script. Whereas Orchestration is the process where we group multiple automated tasks and execute them into an order.

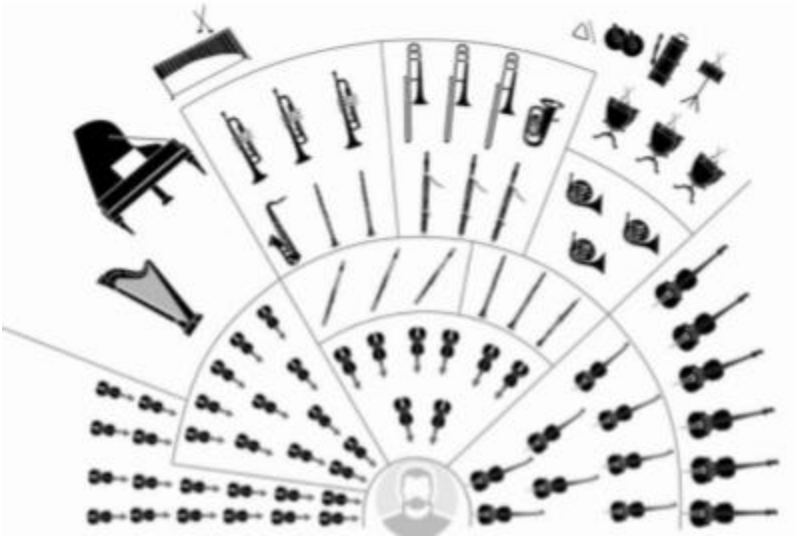
For example, if we are orchestrating setup of a multi-tier web application, we have to setup all the services like Databases, Webservices, Loadbalancer, monitoring in an order so it gets validate when we have entire setup running.

First, we will setup Database service so when webservice is setup it gets connected to database and gets validated. Next, we may setup Loadbalancer and add webservices under it. Monitoring would be setup at the very end.

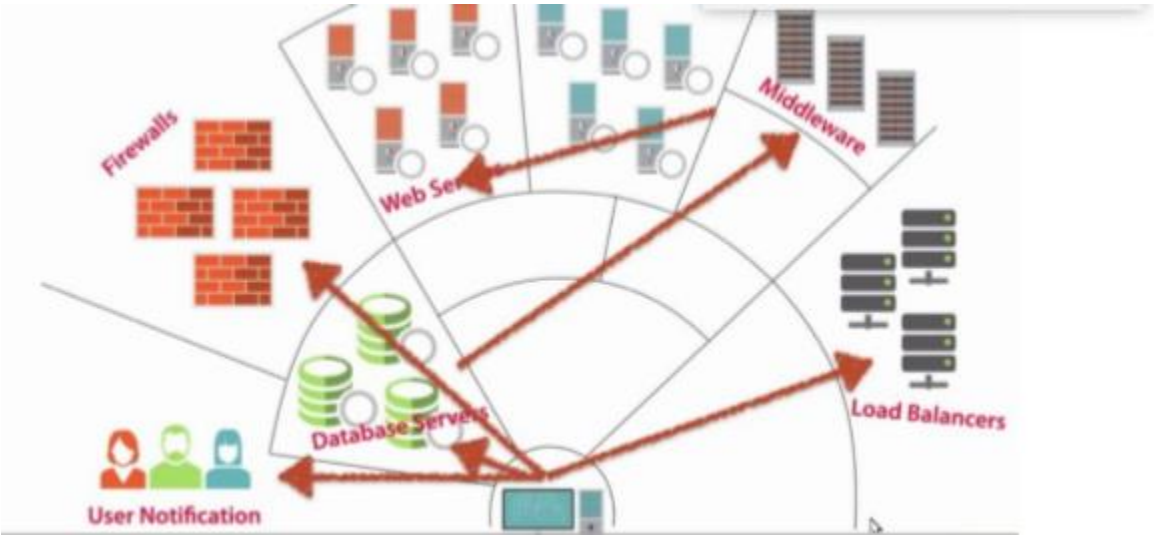
Now monitoring services will return right results from loadbalancer, web service and database service. This process is called as Orchestration but not limited to just this and is done for variety of other IT infrastructure process like cloud computing, network setup etc.



Orchestration word has come from the word Orchestra where different instruments are played in a proper order to generate the beautiful Melody. It's all about order otherwise you don't get music all you get is sound and probably not so good.



Similarly, IT orchestration is all about automating tasks in a proper order



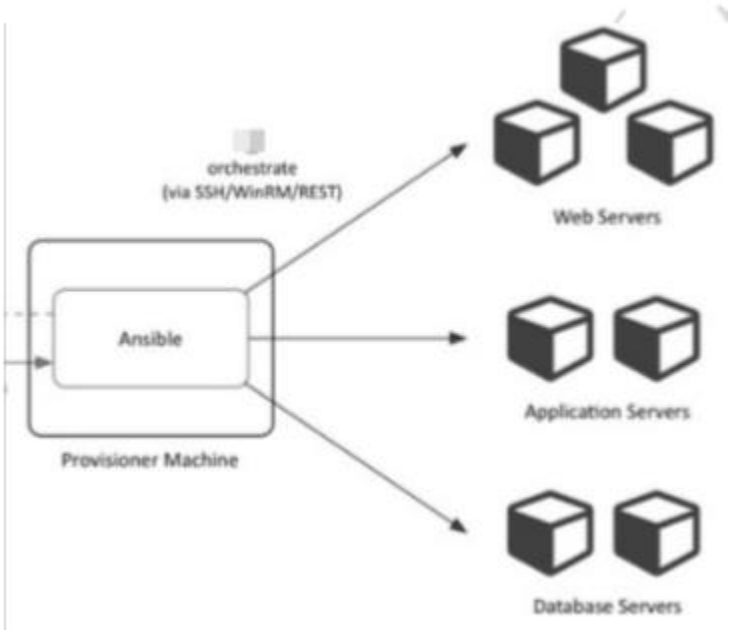
Idempotent Behaviour :

Configuration management tools keep track of the state of resources in order to avoid repeating tasks that were executed before. If a package was already installed, the tool won't try to install it again. The objective is that after each provisioning run the system reaches (or keeps) the desired state, even if you run it multiple times. This is what characterizes these tools as having an idempotent behaviour. One more example would be if we are trying to push a file to multiple servers and some servers already have the same file with the same content then it’s not going to overwrite the file, will simply skip it and push it to their servers where there is a mismatch.

Ansible Introduction

Ansible can control large number of servers and eases administration and operations tasks. Ansible can do simple configuration management and complex orchestration, it has all the features that config tools have plus it's very easy to learn and implement.

It communicates over normal SSH channels in order to retrieve information from remote machines, issue commands, and copy files. For windows node it uses winrm.



Since it uses SSH for Linux and winrm for windows there no need to setup anything at the agent side as ssh comes by default in Linux servers and so as winrm in windows server.

Installing Ansible

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.

Latest Release Via Yum

RPMs are available from yum for EPEL 6, 7, and currently supported Fedora distributions. Ansible itself can manage earlier operating systems that contain Python 2.6 or higher (so also EL6). Fedora users can install Ansible directly, though if you are using RHEL or CentOS and have not already done so, configure EPEL

install the epel-release RPM if needed on CentOS, RHEL, or Scientific Linux

$ sudo yum install ansible

Latest Releases Via Apt (Ubuntu) : Refer Ansible Documentation installing on Ubuntu to install in our course now.

Ubuntu builds are available in a PPA here.

(Windows isn't supported for the control machine). To configure the PPA on your machine and install ansible run these commands:

$ sudo apt-get install software-properties-common

$ sudo apt-add-repository ppa:ansible/ansible

$ sudo apt-get update

$ sudo apt-get install ansible

Some Quick Notes:

- Machine, where ansible is installed is called as Control Machine.
- Ansible is written in Python Language.
- You should have python 2.6/2.7 to install ansible on control machine.
- Windows isn't supported for the control machine.
- Ansible can automate tasks on Linux and windows Machines

Inventory

Inventory is a text file where you define the host information that you want to manage with ansible. The default inventory file location is /etc/ansible/hosts. You can specify a different inventory file using the -i <path> option on the command line.

For this exercise we need two Linux servers, you can spin two centos vm or ec2 instance for practice.

Hosts and Groups

Create a file named inventory-dev(name can be anything) and add below mentioned entry.

web1 ansible_ssh_host=192.168.1.13 ansible_ssh_user=vagrant ansible_ssh_pass='vagrant'

db1 ansible_ssh_host=192.168.1.14 ansible_ssh_user=vagrant ansible_ssh_pass='vagrant'

[webservers]

web1

[dbservers]

db1

Explanation

- web1 and db1 are the names that we have given to the hosts.
- ansible_ssh_host is the variable and its value is the IP address of the server.
- ansible_ssh_user variable holds the username
- ansible_ssh_password holds the password
- [webserver] & [dbservers] is the name of the group which can contain n number hosts. Groupnames are enclosed in square brackets [] .

Note: Mentioning password in the inventory file is not recommended, it's just for initial learning later we will do ssh key exchange. 6. Inventory for Production systems/Real Time.

As we have seen above we put the password in clear text and IP address information also in the inventory. This is a real concern for security, you cannot share this inventory with anyone and also cannot track it in VCS like git. We have better ways to deal with this situation.

1. Since ansible uses SSH, its always recommended to do SSH key exchange and authorize ansible server login to the nodes its managing.
   Note : This way we don't need to mention username and password in the inventory file.

2. Next thing is the IP address, we can manage that with the /etc/hosts file. Map IP to hostname in /etc/hosts file and you can then mention the hostname directly in the inventory.

Hosts file

$ cat /etc/hosts

192.168.1.13 web1

192.168.1.14 db1

INVENTORY File

$ cat inventory-dev

[webservers]

web1

[dbservers]

db1

So now our inventory is very simple and just contain the group and hostname, which is safe.

```
root@control:~# ansible --version
ansible 2.4.2.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansib
le/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.6 (default, Nov 23 2017, 15:49:48) [GCC 4.8.4]
root@control:~# mkdir ansible-repo
root@control:~# cd ansible-repo/
root@control:~/ansible-repo# ls
root@control:~/ansible-repo# mkdir exercise1
root@control:~/ansible-repo# cd exercise1/
root@control:~/ansible-repo/exercise1# vi int-dev
root@control:~/ansible-repo/exercise1# cat int-dev
web01 ansible_ssh_host=192.168.1.9 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant
db01 ansible_ssh_host=192.168.1.10 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant
```

```
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m ping web01
web01 | FAILED! => {
    "msg": "Using a SSH password instead of a key is not possible because Host Key check
ing is enabled and sshpass does not support this.  Please add this host's fingerprint to
 your known_hosts file to manage this host."
}
root@control:~/ansible-repo/exercise1#
```

```
root@control:~/ansible-repo/exercise1# ssh 192.168.1.9
The authenticity of host '192.168.1.9 (192.168.1.9)' can't be established.
RSA key fingerprint is cb:8e:2e:e9:d0:0d:5d:7b:48:d3:25:eb:18:50:b7:22.
Are you sure you want to continue connecting (yes/no)? no
Host key verification failed.
root@control:~/ansible-repo/exercise1#
```

vi /etc/ansible/ansible.cfg

we have to uncomment host_key_checking value like below

```
# uncomment this to disable SSH key host checking
host_key_checking = False
```

After this we have to login all nodes and we have to change PasswordAuthentication should be yes in /etc/ssh/sshd_config

sudo -i

vi /etc/ssh/sshd_config

```
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no
PasswordAuthentication no
```

Then we wil go to our control server and do below steps to connect all nodes.

```
root@control:~/ansible-repo/exercise1# vi /etc/ansible/ansible.cfg
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m ping web01
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
```

```
root@control:~/ansible-repo/exercise1# cat int-dev
web01 ansible_ssh_host=192.168.1.9 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant
db01 ansible_ssh_host=192.168.1.10 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant

[websrvs]
web01


[dbsrvs]
db01


[webdbgrp:children]
websrvs
dbsrvs

root@control:~/ansible-repo/exercise1# ansible -i int-dev -m ping websrvs
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m ping dbsrvs
```

```
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m ping all
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m ping '*'
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
```

```
web01 ansible_ssh_host=192.168.1.9
db01 ansible_ssh_host=192.168.1.10

[websrvs]
web01


[dbsrvs]
db01


[webdbgrp:children]
websrvs
dbsrvs

[webdbgrp:vars]
ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant
~
~
~
~
~
~
~
~
:wq
```

```
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m yum -a "name=nginx state=installed" websrvs
web01 | FAILED! => {
    "changed": false,
    "msg": "No package matching 'nginx' found available, installed or updated",
    "rc": 126,
    "results": [
        "No package matching 'nginx' found available, installed or updated"
    ]
}
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m yum -a "name=epel-release state=installed" websr
vs
web01 | FAILED! => {
    "changed": false,
    "msg": "You need to be root to perform this command.\n",
    "rc": 1,
    "results": [
        "Loaded plugins: fastestmirror\n"
    ]
}
```

```
root@control:~/ansible-repo/exercise1# ansible -i int-dev -m yum -a "name=epel-release state=installed" websr
vs --sudo
```