Getting Started with Ansible for Linux on z

David Gross

Abstract

This paper addresses the use of Ansible to help with automation of server deployment and management using Ansible v1.9.2 with RedHat Enterprise Linux 7. It assumes knowledge of SSH, Linux, Git, virtual machines and basic programming.

The Challenge

Deploying programs and configurations across new servers while maintaining existing servers in an enterprise environment poses a couple of challenges.

When deploying new servers, rolling out updates, or making sure all existing servers stay configured properly, the amount of time required to maintain them grows quickly as the number of servers increases. When only managing a few servers minimal effort is required to issue SSH commands and complete your tasks. However when you scale out your systems it will eventually become massively time consuming to keep everything in check. Grow large enough and it will become completely unmanageable.

Another challenge which can be addressed with the same solution is rolling out rapid updates for test and production environments. You can automate the patch/update process such that as soon as the developer has finished a patch it will start rolling out to the test servers. Once tested you can approve the patch to production and let automation work its magic.

The Solution

The solution is to automate the deployment and management of the servers. With appropriately configured automation 90-100% automation is possible. Making updates and testing significantly easier and less time consuming. There are several options that exist to address these needs however the focus of this paper is Ansible.

Why Ansible?

Ansible allows for the automated deployment of programs, files and configurations. By the nature of how Ansible works it will also recognize if something was changed from the prescribed state and restore it to compliance. For instance if someone removes a package or changes a configuration file it will put it back to the defined state.

One shining feature of Ansible is that it is a lightweight solution which can manage multiple platforms regardless of processor architecture. Ansible uses only SSH to access managed nodes making the requirements for the control machine and the managed nodes very minimal. Other solutions require a client to be installed on the managed nodes whereas Ansible does not. No client means less work to initialize control and easy integration with existing infrastructure. Additionally Ansible at its core is as simple as defining what you want it to be and Ansible will make it so. (With caveats: some tasks are easier to complete than others. However - in principle and practice many tasks are really that easy.)

High-level view of how Ansible works

Ansible supports the ability to send a shell command in the same way you issue one over SSH but Ansible has the ability to do it much more efficiently. In Ansible you will define how you want the server to be in a playbook. When Ansible is executed it will then check the condition of the managed node. (called “gathering facts”) If the configuration of the managed node is different than how the playbook defines it Ansible will make the necessary changes to bring it back to it's defined state. Checking before doing work allows Ansible to limit redundant work done by the systems. There is no reason to copy a 2GB file again if nothing has changed since the last time it was copied.

Example: If you have a folder than contains several configuration files for a program Ansible will check to see if those files exist and if they are the correct files. If they match everything, Ansible will skip it and move on to the next task. If the files have be altered or removed Ansible will then replace the incorrect files with the correct ones.

Concepts

- **Playbooks** are the ‘blueprints’ that get executed. You define how you want the servers deployed and configured and then Ansible makes it happen.

• **Roles** are smaller playbooks in essence. It allows you to break the playbook up into modular parts instead of having one non-modular/non-reusable playbook. For instance a role to install git can be used in many different playbooks.
• **Templates** allow you to dynamically generate files such as config files. They conform to the structure of the file but you can generate unique information for each node. Such as IP addresses and host names.
• **Host file** host files are used to define the IP addresses of the servers you are managing. You can group the IP addresses and have groups of groups as well as using 'all' to execute on all servers.
• **Modules** are the "methods" (from a java perspective) that actually get executed when you run a playbook. They are what contain the instructions for the various tasks you can do.
• **Control Machines** are the machine you have ansible installed on and are executing the playbook from.
• **Managed Nodes** are the servers that you are managing from the control machine.
• **Ansible.cfg** is a file you have in the same directory as your playbook to define configurations for that playbook.
• **Playbook.yaml** The file that gets executed to trigger everything else. It can be called anything but it has to be a .yaml file.

Advantages and Disadvantages

Advantages

• **Lightweight**
  • Uses only SSH to control the servers.
  • Does not require a client to be installed on the servers it is managing.
  • Only requires installation on control machine.

• **Powerful**
  • Able to do anything that can be done over SSH and for many actions it is more efficient than standard shell commands.

• **Modular**
  • Changing what servers are controlled by what parts of the playbook is as simple as adding or taking away IP addresses from the host file.
  • Playbooks can be broken up into roles and templates allowing different parts of the playbook to be used in other playbooks.

• **Scales well**
  • Ansible can manage a theoretically infinite number of servers. The host file contains the IP addresses of the servers you are managing in groups and to add to it you simply need to add the IP address to the group you want that new server to be a part of and the next time the playbook runs it will just add that IP to what it is working on. Removing is just as easy and Ansible won't complain when servers are added and removed.

• **Simple and easy to understand**
  • Playbooks are written in yaml (yet another markup language) which is a fairly 'plain English' type of syntax. Most of the syntax is fairly easy to understand even if you do not know the module you are looking at directly.
  • There is minimal extra syntax surrounding your code, line breaks and indents are used instead of brackets and other such notation.
  • SSH key usage allows your server to be able to manage without requiring username/password to be entered or stored.

• **Almost totally platform agnostic**
  • Ansible can be run from a variety of platforms to issue commands. You can run Ansible on Red Hat, Debian, CentOS, OS X, and BSD.
  • Any of those platforms can control any mixture of ssh enabled servers. You can manage most distros of Linux all from one playbook on one machine. In addition to linux you can manage OSX and AIX systems.

Disadvantages

• Cannot be run from windows without a linux VM.
  • Ansible has the ability to manage windows to some extent however you cannot use windows as a control machine. (The place where Ansible is installed.)
• Perhaps not as robust as certain programs like chef for certain needs.

Playbooks and ad hoc commands

When using Ansible you can issue a single command using an ad hoc command or a series of commands using a playbook.

Ad hoc is a very useful way to issue out a single command to all servers, servers in a group, or several groups. For example, if you wanted to restart all services for vsftpd in group1 as defined in the host file "hosts" you can use:

```
ansible group1 -m service -a "name=vsftpd state=restarted" -i hosts
```

Ansible will then restart the service vsftpd for every ip in group1.
If your command was successful you will get something like this back for each server:

```
10.20.80.121 | success >> {
    "changed": true,
    "name": "vsftpd",
    "state": "started"
}
```

Playbooks are executed and they are used to perform more than one command.
For example, you want to install vsftpd on group2 as defined in the host file "hosts" and make sure the service is running.

The command to execute would be similar to this:

```
# ansible-playbook playbook.yaml -i hosts
```

The playbook would look something like this:

```
---
- hosts: group2
  tasks:
    - name: Install vsftpd
      yum: name=vsftpd state=latest

    - name: Check vsftpd service started
      service: name=vsftpd state=started
```

"-hosts: group2" defines what group within the host file is used.

"tasks:" is the start of the commands being used for group2, after you execute commands you can use "-hosts:" again to choose a new group from the hosts file followed by more tasks.

"-name:" while you can execute commands without names it helps to make both the playbook and the log easier to read, understand, and debug.

"yum:" is one of the built in modules and will handle the various states a package can be in. If vsftpd is already installed it will check the version, and if "state=latest" is used it will always make sure to update to the current version. If vsftpd is not installed it will install the version as defined in "state." Obviously, this is a RHEL specific module. There is also a module for working with SUSE zypper repositories, but you have to test for the distro flavor and set a parameter based on the result to call the correct module – it is not automatic.

Ad hoc is useful for those times when you might want to issue out a command to a number of servers but is not something you do all the time. The playbooks are useful for when you have repetitive commands or more than one. Once you have Ansible available to you, doing a quick restart of a service or of servers becomes very easy to do. As well as the deployment of whole new servers that are the same as other servers your already have. It takes the user the same amount of effort to deploy 1 or 100 once you have the server defined in a playbook and you just simply add entries to the hosts file.

Conditionals
In addition to checks built into a module to skip redundant work, you can put conditionals into the playbook based on the outcome of a previous module. Below is an example of copying a repository file to a managed node. The next command only needs to be run if the repository actually changes.

“register: repo” creates a local variable called 'repo' that stores information about the task that it was created in. In the next task “when: repo.changed” is asking did ‘repo’ change and if it did the current task will execute. This allows your to further reduce redundant/unnecessary work on your managed nodes.

```
- hosts: yumGuest
  tasks:
    - name: Copy sandbx02.repo
      copy: src=sandbx02.repo
            dest=/etc/yum.repos.d/sandbx02.repo
            force=yes
      register: repo

    - name: Yum Clean
      shell: yum clean all
      when: repo.changed
```

Variables
Variables can be created and used in play books. They are used in standard ways and in some slightly more unique ways. As seen in the above you ‘register’ the variable and then within that group of tasks it is available. See Ansible documentation for more details and the many different ways variables occur.

Best Practices

While there are many best practices I am going to only list a few to get you started.

- The device that is running the playbook should have the ssh keys to all of the servers that it is managing.
  - This allows a much easier/seamless management of the server. By having the ssh keys distributed it saves you the extra effort of writing the username and password for each and every server.
  - In addition to the writing being saved it is more secure. If you do not use ssh keys you will have to include username/password in the host file. Which is plain text and a very poor way to store such information.
- Create roles instead of putting all of your commands into one playbook. By having roles it allows you to reuse parts of the code for other playbooks. For instance, installing git using yum is something you might want to do frequently whereas installing createrepo is much more infrequent. So you can reuse the install git role in other play books without needing to rewrite the code for it.
- Do OS and version checks. Even in the same distro there can be variances in how something is configured between versions. So always test for each version of the OS and account for it with a check.
- Always name your tasks descriptively. This will help you when you are trying to debug a playbook. Letting you know where it failed with ease.
- More can be found http://docs.ansible.com/ansible/playbooks_best_practices.html

Reasons to have a dedicated Linux server
When deciding where to run Ansible it is important to keep in mind you need SSH keys distributed to all managed nodes (ideally). Having a dedicated control machine will help keep the effort and complexity to a minimum.
In addition to only needing to copy the SSH keys to one control machine you can also have all of your playbooks and relevant files in one location. (I do however suggest having backups either on your systems or on git.)
When managing a large number of nodes running a playbook can take a long time. Having a dedicated control machine allows your to initiate a job and not worry about your personal machine being unusable for you.
Ansible has changed their syntax between versions, having a dedicated server everyone uses will allow for less conflict when it comes to versions of Ansible being used.

Scenario and example architecture

Scenario: Installing and configuring the Ansible control server.

1. If you have Ansible and it’s dependencies in your yum repository skip this step.

Installing Ansible from the git source repository:

```
git clone https://github.com/ansible/ansible.git --recursive
```

```
#The git clone pulled down a functional ansible environment, which the
#following source command simply adds to your shell PATH variable

source ./ansible/hacking/env-setup

#You might need the c compiler to get the rest of these packages
#installed.

yum install gcc

#These next 3 lines are to install pip – the Python Package manager
#If you have your own way or you can easy_install you can skip these 3
wget https://bootstrap.pypa.io/get-pip.py
chmod +x get-pip.py
python get-pip.py

pip install paramiko
pip install PyYAML
pip install Jinja2
pip install httplib2
pip install six
```

See ansible documentation for latest instructions: http://docs.ansible.com/ansible/intro_installation.html#installation

2. After Ansible is installed you need to make your .yaml file, hosts file and your ansible.cfg. Every time you execute ansible it will use the files you have given it. So if you have two sets of these files in different directories they will not interfere with each other. Ansible will automatically look for the ansible.cfg in the location it is executed, but you can specify all 3 files by giving a specific path to each one.

Example files:
Ansible.cfg allows you to define things such as: the transport protocol which is ssh in this example and the "ForwardAgent=yes" is telling it to use the host machines ssh keys.
ansible.cfg

```
[defaults]
transport = ssh

[ssh_connection]
ssh_args = -o ForwardAgent=yes
```

hosts (file)
The host file contains the groups and ip addresses, however you can add in extra information such as connection type and user/password if you need to be specific.

```
[yumServer]
10.20.80.121 ansible_connection=ssh  ansible_user=root

[yumGuest]
10.20.81.117
```

3. Once ansible is installed and the config files are in place it is a good idea to copy your ssh keys out to the servers that are being managed.
   a. Generate the ssh keys, there are several ways to do this so use whatever you deem best.
   b. Copy the ssh keys to your servers by using "ssh-copy-id root@DestinationIP"

Scenario: Deploying an in house built rpm to your nodes using a local yum repository.

For my example I will be using several servers.
- Ansible server
- Git server
- Yum server
- Managed nodes rpm is being deployed too. (Can be one or legion)

Purpose of each server
- **Ansible server (aka control machine)**: This is the server we will execute the playbooks on.
- By having a dedicated Ansible server you only have to copy the SSH keys of this server to all the managed nodes. It makes it easier to ensure all the SSH keys have been added.
- It also allows you to execute long playbooks overnight or over the weekend. As well as setting up

automated execution.
• **Git server**: storage and versioning of playbooks, host files, config files, rpms and more.
  • By having the git server it allows a centralized and redundant place to keep all of your items. If you ever need to rebuild your Ansible server it is very easy and you minimize the chance of loss of files.
• **Yum server**: custom local repository for deployment of rpms not found on the official repository.
  • By using yum instead of copying the rpm to the server and manually installing the rpm it allows your to have multiple versions of the same program and manage dependencies easier.
  • Ansible also has built in yum modules so interacting with yum is very easy and it makes more advanced playbook usage simpler.
• **Managed nodes**: The servers being deployed to. Ansible has virtually infinite scalability for number of nodes it can manage. You can use it to manage a single server or thousands.

Note: mongodb-org is just a name of an in house compiled rpm of mongodb.

Playbook in parts with explanations
Achieving all of the above can be accomplished using a single playbook or a playbook with multiple roles. For this example I will be using a single playbook.
Below is an example playbook that will install the required programs for a yum server, copy the rpms onto the server and create the yum repository. Then it will configure the managed nodes with the appropriate .repo file and clean/update yum so the new repo is accessible.

```
# Note: Every yaml file needs to start with the three dashes to
designate the beginning. However you can add comments before them.
---
# "- host:" defines what the host group that is going to be use from
the host file.
- hosts: yumServer
# tasks now define tasks that are going to be done for this host group.
Each task is designated by – name.
  tasks:
#"– Name" while not required (can be left blank) is very useful for
user readability and for debugging later. As the playbook is executed
it will display the name of the task that it is running to let you know
if it was completed, changed, or failed.
    - name: Install vsftpd
      yum: name=vsftpd state=latest

    - name: Install createrepo
      yum: name=createrepo state=latest

    - name: Check vsftpd service started
      service: name=vsftpd state=started

    - name: Copy rpms
      copy: src=/root/mongoPackages/
            dest=/var/ftp/pub/rhel/Packages/
            force=yes
      register: rpms

#Instead of copy you can use a git server.
    - name: Install Git
      yum: name=git state=latest

    - name: Git Clone
      git: repo=GitAddress
        accept_hostkey=yes
        dest=/var/ftp/pub/rhel/Packages/
        force=yes
       register: rpms
#End of git block

    - name: Create repo
      shell: createrepo /var/ftp/pub/rhel/
      when: rpms.changed


- hosts: yumGuest
  tasks:
    - name: Copy sandbx02.repo
```

```
      copy: src=sandbx02.repo
            dest=/etc/yum.repos.d/sandbx02.repo
            force=yes
      register: repo

    - name: Yum Clean
      shell: yum clean all
      when: repo.changed

    - name: Yum Update
      shell: yum -y update

  #mongodb-org is the package name here.
    - name: Install mongodb
      yum: name=mongodb-org state=latest

    - name: Setting process limit for user mongod
      lineinfile: dest=/etc/security/limits.d/20-nproc.conf
                  line="mongod soft nproc 64000"

    - name: Disable SELinux
      selinux: state=disabled
      register: selin

    - name: Reboot
      shell: shutdown -r +10
      when: selin.changed
```

EXTRA EXAMPLES

The ansible .cfg file sets certain configuration for Ansible when executed. You can have multiple ansible.cfg files so you can make them project specific if needed. It allows you to define things like the transport protocol, ssh in this example. the “ForwardAgent=yes” is telling it to use the host machines ssh keys.

ansible.cfg

```
[defaults]
transport = ssh

[ssh_connection]
ssh_args = -o ForwardAgent=yes
```

hosts

The host file contains the groups and ip addresses, however you can add in extra information such as connection type and user/password if you need to be specific.

```
[yumServer]
10.20.30.121 ansible_connection=ssh  ansible_user=root

[yumGuest]
10.20.31.117
10.20.31.118
10.20.31.119
10.20.31.120
10.20.31.180
```

Example playbook run

Files:
customRepo.repo is my in house repository. Use whatever your repository is for your own project.

ansible.cfg

```
[defaults]
transport = ssh
```

```
[ssh_connection]
ssh_args = -o ForwardAgent=yes
```

hosts

```
[ansibleServer]
10.20.80.121

[managedNodes]
10.20.81.119
10.20.81.120
10.20.81.121
10.20.81.122
10.20.81.123
10.20.81.115
10.20.81.116
```

installAnsible.yaml
This playbook will install ansible on the ansible server (ansible-ception) and then ping the managed nodes to see if they are alive. (This install is using rpms instead of building from source as shown above) You can use something other than ping or use this as an opportunity to configure them but for this example I am just going to ping them. In this case I am adding a custom repository I have built which contains the ansible rpm as well as its dependencies.

```
---
- hosts: ansibleServer
  tasks:
    - name: Copy customRepo.repo
      copy: src=customRepo.repo
            dest=/etc/yum.repos.d/customRepo.repo
            force=yes
      register: repo

    - name: Yum Clean
      shell: yum clean all
      when: repo.changed

    - name: Yum Update
      shell: yum -y update

    - name: Install Ansible
      yum: name=ansible state=latest

-hosts: managedNodes
    - name: Check if servers are alive
      ping:
```

Ansible was installed but there was an error in my code(the group name was spelled incorrectly in the host file) so it skipped the last command and never gathered facts on those servers:

Output:

```
[root@ansible1 ansibleExample]# ansible-playbook installAnsible.yaml -i hosts

PLAY [ansibleServer] **********************************************************

GATHERING FACTS ***************************************************************
ok: [10.20.80.121]

TASK: [Copy customRepo.repo] **************************************************
changed: [10.20.80.121]

TASK: [Yum Clean] *************************************************************
changed: [10.20.80.121]

TASK: [Yum Update] ************************************************************
changed: [10.20.80.121]

TASK: [Install Ansible] *******************************************************
changed: [10.20.80.121]
```

```
PLAY [managedNodes] ************************************************************
skipping: no hosts matched

PLAY RECAP *********************************************************************
10.20.80.121               : ok=5    changed=4    unreachable=0    failed=0

====
```

So if we run it again (with the host file fixed) we will see that it has no reason to update the .repo file or install ansible because that was successful the last run. Also because it did not change the repo file it skips the “yum clean.”

Output:

```
[root@ansible1 ansibleExample]# ansible-playbook installAnsible.yaml -i hosts

PLAY [ansibleServer] ***********************************************************

GATHERING FACTS ****************************************************************
ok: [10.20.80.121]

TASK: [Copy customRepo.repo] ***************************************************
ok: [10.20.80.121]

TASK: [Yum Clean] **************************************************************
skipping: [10.20.80.121]

TASK: [Yum Update] *************************************************************
changed: [10.20.80.121]

TASK: [Install Ansible] ********************************************************
ok: [10.20.80.121]

PLAY [managedNodes] ************************************************************

GATHERING FACTS ****************************************************************
ok: [10.20.81.120]
ok: [10.20.81.121]
ok: [10.20.81.123]
ok: [10.20.81.122]
ok: [10.20.81.119]
ok: [10.20.81.116]
ok: [10.20.81.115]

TASK: [Check if servers are alive] *********************************************
ok: [10.20.81.121]
ok: [10.20.81.120]
ok: [10.20.81.123]
ok: [10.20.81.122]
ok: [10.20.81.119]
ok: [10.20.81.116]
ok: [10.20.81.115]

PLAY RECAP *********************************************************************
10.20.80.121               : ok=4    changed=1    unreachable=0    failed=0
10.20.81.115               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.116               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.119               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.120               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.121               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.122               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.123               : ok=2    changed=0    unreachable=0    failed=0

====
```

Now I am going to go in and delete the repo file and ansible will see this the next run and correct it. Ansible is still installed however so it won’t try to install it again.

Output:

```
[root@ansible1 ansibleExample]# ansible-playbook installAnsible.yaml -i hosts

PLAY [ansibleServer] ***********************************************************

GATHERING FACTS ****************************************************************
```



```
ok: [10.20.80.121]

TASK: [Copy customRepo.repo] ***************************************************
changed: [10.20.80.121]

TASK: [Yum Clean] **************************************************************
changed: [10.20.80.121]

TASK: [Yum Update] *************************************************************
changed: [10.20.80.121]

TASK: [Install Ansible] ********************************************************
ok: [10.20.80.121]

PLAY [managedNodes] ************************************************************

GATHERING FACTS ****************************************************************
ok: [10.20.81.120]
ok: [10.20.81.121]
ok: [10.20.81.123]
ok: [10.20.81.122]
ok: [10.20.81.119]
ok: [10.20.81.116]
ok: [10.20.81.115]

TASK: [Check if servers are alive] *********************************************
ok: [10.20.81.120]
ok: [10.20.81.121]
ok: [10.20.81.123]
ok: [10.20.81.122]
ok: [10.20.81.119]
ok: [10.20.81.116]
ok: [10.20.81.115]

PLAY RECAP *********************************************************************
10.20.80.121               : ok=5    changed=3    unreachable=0    failed=0
10.20.81.115               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.116               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.119               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.120               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.121               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.122               : ok=2    changed=0    unreachable=0    failed=0
10.20.81.123               : ok=2    changed=0    unreachable=0    failed=0

===
```

You can continue to remove or change things to break your settings and every time ansible is executed again it will fix whatever is broken and leave all the working tasks alone.

Links
Ansible documentation: http://docs.ansible.com/ansible/index.html

IBM Systems
Route 100 Somers, New York 10589
U.S.A.
Produced in the United States of America,
01/2016

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.