IT Automation with

Agenda

- Ansible installation
- Ansible configuration
- Introduction to YAML

Ansible installation

Basics / What will be installed

Ansible by default manages machines over SSH protocol.

Once Ansible is installed, it won't add add a database and there won't be any daemons to start or keep running.

You only need to install it on one machine, e.g: a laptop.

When Ansible manages remote machines, it doesn't leave software installed or running on them, so there's no real question about how to upgrade Ansible when moving to a new version.

Ansible installation

Control node requirements

Currently Ansible can be run from any machine with Python 2 (2.6 or 2.7) or Python 3 (versions 3.5 and higher) installed. Windows isn't supported for the control machine.

This includes Red Hat, Debian, CentOS, OSX, any of the BSDs, etc.

Note: Ansible 2.2 introduces a tech preview of support for Python 3.

Ansible installation

```
# Debian & Ubuntu (apt).
$ sudo apt-get install ansible

# RHEL & CentOS (yum).
$ sudo yum install ansible

# Mac OS X (homebrew).
$ brew install ansible

# Python (pip).
$ sudo pip install ansible
```

Verifying Ansible installation

```
[quicklab@master-0 ~]$ ansible --version
ansible 2.4.3.0
  config file = /home/quicklab/ansible.cfg
  configured module search path = [u'/home/quicklab/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Feb 20 2018, 09:19:12) [GCC 4.8.5 20150623 (Red Hat 4.8.5-28)]
```

```
[quicklab@master-0 ~]$ ansible localhost -m ping
 [WARNING]: provided hosts list is empty, only localhost is available. Note
that the implicit localhost does not match 'all'

localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

SSH key-based authentication

- Users can authenticate through ssh by using *public key authentication.*
- The private key must be kept secret and secure.
- The public key is put into the hosts the user wants to log in.
- Key generation is performed by **ssh-keygen** command.
- The file permissions for ~/.ssh directory must be 0700
- The file permissions for private key file must be 0600 or 0400
- In order to copy public key to destination hosts it can be done through ssh-copy-id command: *ssh-copy-id foo@<destination-host>*

Demo ssh-keys

Configuring Ansible

Configuring Ansible

Where does the config comes from?

- Ansible configuration file can be specified in several paths:
  - System wide: /etc/ansible/ansible.cfg
  - User's $HOME: ~/.ansible.cfg
  - Current directory: ./ansible *# recommended way*
  - Custom location: $ANSIBLE_CONFIG
- The search order that Ansible follows is the **reverse** of above list.
- Ansible will take settings for first file that it finds as per its precedence order and will not combine settings from lower precedence levels.
- To check from where Ansible is taking its config the **ansible** command can be invoked either with --version or with -v.

Configuring Ansible

Configuration file

- The Ansible configuration file is split into several sections, each one enclosed in square brackets and settings specified as key/value pairs.

Name	Purpose
[default]	most config options are set here
[privilege_escalation]	Settings for configuring privilege escalation on managed hosts
[paramiko_connection]	Parameters for old managed hosts
[ssh_connection]	Parameters for customizing ssh connection
[selinux]	Parameters concerning SELinux

- Some settings are pre-defined with default values and will still apply even if they are commented out in global */etc/ansible/ansible.cfg.*
  To identify current values: *ansible-config dump*

Configuring Ansible

Configuration file

- Among the most common options to modify there are:

Setting	Description
remote_user	The user Ansible will use to connect to remote managed hosts.
inventory	Specifies the inventory Ansible is currently using by default.
become	Enable / disable privileges escalation.
become_user	The privileged account will be used for privilege escalation
become_method	The method to use for privilege escalation, e.g: sudo

**https://raw.githubusercontent.com/ansible/ansible/devel/examples/ansible.cfg*

Introduction to YAML

YAML Basics

Lists and dictionaries

- For Ansible, nearly every YAML file starts with a list.
- Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”.
- All YAML files can optionally begin with --- and end with ....
- All members of a list are lines beginning at the same indentation level starting with a "- " (a dash and a space):

```
---
# A list of tasty fruits
fruits:
    - Apple
    - Orange
    - Strawberry
    - Mango
...
```

- A dictionary is represented in a simple key: value form (the colon must be followed by a space):

```
---
# A list of tasty fruits
fruits:
    - Apple
    - Orange
    - Strawberry
    - Mango
...
```

YAML Basics

Lists of dicts and abbreviated form

- More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

```
# Employee records
-  martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
```

- Dictionaries and lists can also be represented in an abbreviated form if you really want to:

```
---
martin: {name: Martin D'vloper, job: Developer, skill: Elite}
fruits: ['Apple', 'Orange', 'Strawberry', 'Mango']
```

- In YAML, comments are initiated by a hash symbol (#) and can exist at the end of any line.

YAML Basics

Boolean values

- Although Ansible doesn't use them too much, but you can also specify a boolean value (true/false) in several forms:

```
create_key: yes
needs_agent: no
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

- Ansible is pretty flexible in how you represent truthy and falsey values in playbooks. Strictly speaking, module arguments (for example, **update_cache=yes**) are treated differently from values elsewhere in playbooks (for example, **sudo: True**).

- YAML truthy: true, True, TRUE, yes, Yes, YES, on, On, ON, y, Y
- YAML falsey: false, False, FALSE, no, No, NO, off, Off, OFF, n, N
- Module arg truthy: yes, on, 1, true
- Module arg falsey: no, off, 0, false

Ansible docs typically use **yes** and **no** when passing args to modules and **True** and **False** elsewhere

YAML Basics

multi-line values

- Multi-line strings can be expressed using | or >.
- Spanning multiple lines using "|" will include newlines characters as well as any trailing spaces.
- In the other hand, spanning multiple lines using ">" will convert newline characters in spaces.

```
include_newlines: |
            exactly as you see
            will appear these three
            lines of poetry

fold_newlines: >
            this is really a
            single line of text
            despite appearances
```

YAML Basics

Boolean values and multi-line values

Let's test this out in this simple playbook:

```yaml
- hosts: localhost
  vars:
    include_newlines: |
     exactly as you see
        will appear these three
     lines of poetry

    fold_newlines: >
     this is really a
     single line of text
     despite appearances
  tasks:
    - debug: var=include_newlines
    - debug: var=fold_newlines
```

Now let's run it to see its output:

```
[quicklab@master-0 ~]$ ansible-playbook yaml-multiline-value.yml
 [WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'


PLAY [localhost] ***************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************
ok: [localhost]

TASK [debug] *******************************************************************************************************************
ok: [localhost] => {
    "include_newlines": "exactly as you see\n   will appear these three\nlines of poetry\n"
}

TASK [debug] *******************************************************************************************************************
ok: [localhost] => {
    "fold_newlines": "this is really a single line of text despite appearances\n"
}

PLAY RECAP *********************************************************************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0
```

YAML Basics

Verifying YAML Syntax using Python

- **Python way:** Requires installation of *PyYAML* package.

  $ python -c 'import yaml, sys; print yaml.load(sys.stdin)' < file.yml

- If no syntax error exists, Python prints the contents of the YAML file to stdout in JSON format:

```
[jrosental@example ~] cat file.yml
---
- first item
- second item
[jrosental@example ~] python -c ‘import yaml,sys; print
yaml.load(sys.stdin) < file.yml’
[‘first item’,’second item’]
```

YAML Basics

Verifying YAML Syntax using Python

The following example shows what output is shown when a file contains Invalid YAML syntax:

```
[jrosental@example ~] cat file.yml
---
- first item
-second item
[jrosental@example ~] python -c 'import yaml,sys; print yaml.load(sys.stdin) <
file.yml'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib64/python2.7/site-packages/yaml/__init__.py", line 71, in load
    return loader.get_single_data()
...output omitted...
yaml.scanner.ScannerError: while scanning a simple key
  in "<stdin>", line 4, column 1
could not found expected ':'
  in "<stdin>", line 5, column 1
```

YAML Basics

Verifying YAML Syntax using YAML lint

- **YAML Lint:** http://www.yamllint.com

Will return Valid YAML! on valid YAML syntax and a message like

(<unknown>): could not find expected ':' while scanning a simple key at line 4 column 1

If the previous malformed YAML is passed.

May be an alternative for people that are not familiar with Python.

YAML Basics

Verifying YAML Syntax using ansible command line tools

- Ansible offers a native way to validate YAML syntax in a playbook through the **ansible-playbook** command when invoked with **--syntax-check** option.
- It is the preferred way for validate YAML syntax for a playbook since It performs a more rigorous review and ensures any element specific to Ansible playbook is present.
- In case of error, it gives information such as where the error might be (similar to what any other parser / compiler will do)

YAML Basics

Verifying YAML Syntax using ansible command line tools

The following example demostrate that although the purely YAML syntax is valid, we get an error as it doesn't seem to be a valid playbook:

```
[quicklab@master-0 ~]$ ansible-playbook --syntax-check file.yml
 [WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

ERROR! playbook entries must be either a valid play or an include statement

The error appears to have been in '/home/quicklab/file.yml': line 2, column 3, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

---
- first item
  ^ here
```

YAML Basics

Gotchas

- A colon followed by a space (or newline) starts a comment.
  This will result in a YAML syntax error:

```
foo: somebody said I should put a colon here: so I did

windows_drive: c:
```

But this will work:

```
windows_path: c:\windows
```

You will want to quote hash values using colons following by a space or the end of the line using single or double quotes:

```
foo: 'somebody said I should put a colon here: so I did'

windows_drive: 'c:'
```

YAML Basics

Gotchas

- Ansible uses “{{ var }}” for variables. If a value after a colon starts with a “{“ YAML will think it’s a dictionary, so you must quote it, like so:

```
foo: "{{ variable }}"
```

  If your value starts with a quote the entire value must be quoted:

```
foo: "{{ variable }}/additional/string/literal"
foo2: "{{ variable }}\\backslashes\\are\\also\\special\\characters"
foo3: "even if it's just a string literal it must all be quoted"
```

- In addition to ‘ and “ there are a number of characters that are reserved As the first character of an unquotes scalar: [] {} > | * & ! % # ` @ ,

YAML Basics

Exercise

How would you define a YAML file from the following information?

“Martin Dubois works a developer for ACME, Inc company . In his CV he states His Perl and Python skills are ‘Elite’ while his Pascal ones are not so good (‘Lame’).

His favorite fruits are: apple, orange, strawberry and mango.

He lives in:

Melvin Porter
P.O. Box 132 1599 Curabitur Rd.
Bandera South Dakota 45149
(959) 119-8364”

Questions?

References

- YAML version 1.2: http://www.yaml.org/spec/1.2/spec.html
- YAML syntax: http://docs.ansible.com/ansible/YAMLSyntax.html