FREE eBook

LEARNING ansible

Free unaffiliated eBook created from
Stack Overflow contributors.

#ansible

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ansible

It is an unofficial and free ansible ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ansible.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with ansible

Remarks

This section provides an overview of what ansible is, and why a developer might want to use it.

It should also mention any large subjects within ansible, and link out to the related topics. Since the Documentation for ansible is new, you may need to create initial versions of those related topics.

Examples

Hello, World

Create a directory called `ansible-helloworld-playbook`

```
mkdir ansible-helloworld-playbook
```

Create a file `hosts` and add remote systems how want to manage. As ansible relies on ssh to connect the machines, you should make sure they are already accessible to you in ssh from your computer.

```
192.168.1.1
192.168.1.2
```

Test connection to your remote systems using the Ansible ping module.

```
ansible all -m ping -k
```

In case of success it should return something like that

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

In case of error it should return

```
192.168.1.1| UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh.",
    "unreachable": true
}
```

Test sudo access with

```
ansible all -m ping -k -b
```

Test connection and configuration with ping

```
ansible -i hosts -m ping targethost
```

`-i hosts` defines the path to inventory file
`targethost` is the name of the host in the `hosts` file

Inventory

Inventory is the Ansible way to track all the systems in your infrastructure. Here is a simple static inventory file containing a single system and the login credentials for Ansible.

```
[targethost]
192.168.1.1 ansible_user=mrtuovinen ansible_ssh_pass=PassW0rd
```

Write these lines for example to `hosts` file and pass the file to `ansible` or `ansible-playbook` command with `-i`/`--inventory-file` flag.

See static inventory and dynamic inventory for more details.

Provisioning remote machines with Ansible

We can provision remote systems with Ansible. You should have an SSH key-pair and you should take your SSH public key to the machine ~/.ssh/authorized_keys file. The porpuse is you can login without any authorization.

Prerequisites:

- Ansible

You need an Inventory file (for ex.: development.ini) where you determine the host what you want to use:

```
[MACHINE_NAME]
MACHINE_NAME hostname=MACHINE_NAME ansible_ssh_host=IP_ADDRESS ansible_port=SSH_PORT
ansible_connection=ssh ansible_user=USER ansible_ssh_extra_args="-o StrictHostKeyChecking=no -
o UserKnownHostsFile=/dev/null"
```

- hostname - the hostname of the remote machine
- ansible_ssh_host - the ip or domain of the remote host
- ansible_port - the port of the remote host which is usually 22
- ansible_connection - the connection where we set, we want to connect with ssh
- ansible_user - the ssh user
- ansible_ssh_extra_args - extra argumentums what you want to specify for the ssh

connection

Required extra args for ssh:

- StrictHostKeyChecking - It can ask a key checking what waiting for a yes or no. The Ansible can't answer this question then throw an error, the host not available.
- UserKnownHostsFile - Needed for StrictHostKeyChecking option.

If you have this inventory file you can write a test playbook.yml:

```
---
- hosts: MACHINE_NAME
  tasks:
    - name: Say hello
      debug:
        msg: 'Hello, World'
```

then you can start the provision:

```
ansible-playbook -i development.ini playbook.yml
```

ansible.cfg

This is the default ansible.cfg from Ansible github.

```
# config file for ansible -- http://ansible.com/
# ==============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#remote_tmp     = $HOME/.ansible/tmp
#local_tmp      = $HOME/.ansible/tmp
#forks          = 5
#poll_interval  = 15
#sudo_user      = root
#ask_sudo_pass = True
#ask_pass      = True
#transport      = smart
#remote_port    = 22
#module_lang    = C
#module_set_locale = False

# plays will gather facts by default, which contain information about
# the remote system.
#
```

```
# smart - gather by default, but don't regather if already gathered
# implicit - gather by default, turn off with gather_facts: False
# explicit - do not gather by default, must say gather_facts: True
#gathering = implicit

# by default retrieve all facts subsets
# all - gather all subsets
# network - gather min and network facts
# hardware - gather hardware facts (longest facts to retrieve)
# virtual - gather min and virtual facts
# facter - import facts from facter
# ohai - import facts from ohai
# You can combine them using comma (ex: network,virtual)
# You can negate them using ! (ex: !hardware,!facter,!ohai)
# A minimal set of facts is always gathered.
#gather_subset = all

# some hardware related facts are collected
# with a maximum timeout of 10 seconds. This
# option lets you increase or decrease that
# timeout to something more suitable for the
# environment.
# gather_timeout = 10

# additional paths to search for roles in, colon separated
#roles_path    = /etc/ansible/roles

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

# change the default callback
#stdout_callback = skippy
# enable additional callbacks
#callback_whitelist = timer, mail

# Determine whether includes in tasks and handlers are "static" by
# default. As of 2.0, includes are dynamic by default. Setting these
# values to True will make includes behave more like they did in the
# 1.x versions.
#task_includes_static = True
#handler_includes_static = True

# change this for alternative sudo implementations
#sudo_exe = sudo

# What flags to pass to sudo
# WARNING: leaving out the defaults might create unexpected behaviours
#sudo_flags = -H -S -n

# SSH timeout
#timeout = 10

# default user to use for playbooks if user is not specified
# (/usr/bin/ansible will use current user as default)
#remote_user = root

# logging is off by default unless this path is defined
# if so defined, consider logrotate
#log_path = /var/log/ansible.log

# default module name for /usr/bin/ansible
```

```
#module_name = command

# use this shell for commands executed under sudo
# you may need to change this to bin/bash in rare instances
# if sudo is constrained
#executable = /bin/sh

# if inventory variables overlap, does the higher precedence one win
# or are hash values merged together?  The default is 'replace' but
# this can also be set to 'merge'.
#hash_behaviour = replace

# by default, variables from roles will be visible in the global variable
# scope. To prevent this, the following option can be enabled, and only
# tasks and handlers within the role will see the variables there
#private_role_vars = yes

# list any Jinja2 extensions to enable here:
#jinja2_extensions = jinja2.ext.do,jinja2.ext.i18n

# if set, always use this private key file for authentication, same as
# if passing --private-key to ansible or ansible-playbook
#private_key_file = /path/to/file

# If set, configures the path to the Vault password file as an alternative to
# specifying --vault-password-file on the command line.
#vault_password_file = /path/to/vault_password_file

# format of string {{ ansible_managed }} available within Jinja2
# templates indicates to users editing templates files will be replaced.
# replacing {file}, {host} and {uid} and strftime codes with proper values.
#ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on {host}
# This short version is better used in templates as it won't flag the file as changed every
run.
#ansible_managed = Ansible managed: {file} on {host}

# by default, ansible-playbook will display "Skipping [host]" if it determines a task
# should not be run on a host.  Set this to "False" if you don't want to see these "Skipping"
# messages. NOTE: the task header will still be shown regardless of whether or not the
# task is skipped.
#display_skipped_hosts = True

# by default, if a task in a playbook does not include a name: field then
# ansible-playbook will construct a header that includes the task's action but
# not the task's args.  This is a security feature because ansible cannot know
# if the *module* considers an argument to be no_log at the time that the
# header is printed.  If your environment doesn't have a problem securing
# stdout from ansible-playbook (or you have manually specified no_log in your
# playbook on all of the tasks where you have secret information) then you can
# safely set this to True to get more informative messages.
#display_args_to_stdout = False

# by default (as of 1.3), Ansible will raise errors when attempting to dereference
# Jinja2 variables that are not set in templates or action lines. Uncomment this line
# to revert the behavior to pre-1.3.
#error_on_undefined_vars = False

# by default (as of 1.6), Ansible may display warnings based on the configuration of the
# system running ansible itself. This may include warnings about 3rd party packages or
# other conditions that should be resolved if possible.
# to disable these warnings, set the following value to False:
```

```
#system_warnings = True

# by default (as of 1.4), Ansible may display deprecation warnings for language
# features that should no longer be used and will be removed in future versions.
# to disable these warnings, set the following value to False:
#deprecation_warnings = True

# (as of 1.8), Ansible can optionally warn when usage of the shell and
# command module appear to be simplified by using a default Ansible module
# instead.  These warnings can be silenced by adjusting the following
# setting or adding warn=yes or warn=no to the end of the command line
# parameter string.  This will for example suggest using the git module
# instead of shelling out to the git command.
# command_warnings = False


# set plugin path directories here, separate with colons
#action_plugins     = /usr/share/ansible/plugins/action
#cache_plugins      = /usr/share/ansible/plugins/cache
#callback_plugins   = /usr/share/ansible/plugins/callback
#connection_plugins = /usr/share/ansible/plugins/connection
#lookup_plugins     = /usr/share/ansible/plugins/lookup
#inventory_plugins  = /usr/share/ansible/plugins/inventory
#vars_plugins       = /usr/share/ansible/plugins/vars
#filter_plugins     = /usr/share/ansible/plugins/filter
#test_plugins       = /usr/share/ansible/plugins/test
#strategy_plugins   = /usr/share/ansible/plugins/strategy

# by default callbacks are not loaded for /bin/ansible, enable this if you
# want, for example, a notification or logging callback to also apply to
# /bin/ansible runs
#bin_ansible_callbacks = False


# don't like cows?  that's unfortunate.
# set to 1 if you don't want cowsay support or export ANSIBLE_NOCOWS=1
#nocows = 1

# set which cowsay stencil you'd like to use by default. When set to 'random',
# a random stencil will be selected for each task. The selection will be filtered
# against the `cow_whitelist` option below.
#cow_selection = default
#cow_selection = random

# when using the 'random' option for cowsay, stencils will be restricted to this list.
# it should be formatted as a comma-separated list with no spaces between names.
# NOTE: line continuations here are for formatting purposes only, as the INI parser
#       in python does not support them.
#cow_whitelist=bud-frogs,bunny,cheese,daemon,default,dragon,elephant-in-snake,elephant,eyes,\
#              hellokitty,kitty,luke-
koala,meow,milk,moofasa,moose,ren,sheep,small,stegosaurus,\
#              stimpy,supermilker,three-eyes,turkey,turtle,tux,udder,vader-koala,vader,www

# don't like colors either?
# set to 1 if you don't want colors, or export ANSIBLE_NOCOLOR=1
#nocolor = 1

# if set to a persistent type (not 'memory', for example 'redis') fact values
# from previous runs in Ansible will be stored.  This may be useful when
# wanting to use, for example, IP information from one group of servers
# without having to talk to them in the same playbook run to get their
```

```
# current IP information.
#fact_caching = memory


# retry files
# When a playbook fails by default a .retry file will be created in ~/
# You can disable this feature by setting retry_files_enabled to False
# and you can change the location of the files by setting retry_files_save_path

#retry_files_enabled = False
#retry_files_save_path = ~/.ansible-retry

# squash actions
# Ansible can optimise actions that call modules with list parameters
# when looping. Instead of calling the module once per with_ item, the
# module is called once with all items at once. Currently this only works
# under limited circumstances, and only with parameters named 'name'.
#squash_actions = apk,apt,dnf,package,pacman,pkgng,yum,zypper

# prevents logging of task data, off by default
#no_log = False

# prevents logging of tasks, but only on the targets, data is still logged on the
master/controller
#no_target_syslog = False

# controls whether Ansible will raise an error or warning if a task has no
# choice but to create world readable temporary files to execute a module on
# the remote machine.  This option is False by default for security.  Users may
# turn this on to have behaviour more like Ansible prior to 2.1.x.  See
# https://docs.ansible.com/ansible/become.html#becoming-an-unprivileged-user
# for more secure ways to fix this than enabling this option.
#allow_world_readable_tmpfiles = False

# controls the compression level of variables sent to
# worker processes. At the default of 0, no compression
# is used. This value must be an integer from 0 to 9.
#var_compression_level = 9

# controls what compression method is used for new-style ansible modules when
# they are sent to the remote system.  The compression types depend on having
# support compiled into both the controller's python and the client's python.
# The names should match with the python Zipfile compression types:
# * ZIP_STORED (no compression. available everywhere)
# * ZIP_DEFLATED (uses zlib, the default)
# These values may be set per host via the ansible_module_compression inventory
# variable
#module_compression = 'ZIP_DEFLATED'

# This controls the cutoff point (in bytes) on --diff for files
# set to 0 for unlimited (RAM may suffer!).
#max_diff_size = 1048576

[privilege_escalation]
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False

[paramiko_connection]
```

```
# uncomment this line to cause the paramiko connection plugin to not record new host
# keys encountered.  Increases performance on new host additions.  Setting works independently
of the
# host key checking setting above.
#record_host_keys=False

# by default, Ansible requests a pseudo-terminal for commands executed under sudo. Uncomment
this
# line to disable this behaviour.
#pty=False

[ssh_connection]

# ssh arguments to use
# Leaving off ControlPersist will result in poor performance, so use
# paramiko on older platforms rather than removing it, -C controls compression use
#ssh_args = -C -o ControlMaster=auto -o ControlPersist=60s

# The path to use for the ControlPath sockets. This defaults to
# "%(directory)s/ansible-ssh-%%h-%%p-%%r", however on some systems with
# very long hostnames or very long path names (caused by long user names or
# deeply nested home directories) this can exceed the character limit on
# file socket names (108 characters for most platforms). In that case, you
# may wish to shorten the string below.
#
# Example:
# control_path = %(directory)s/%%h-%%r
#control_path = %(directory)s/ansible-ssh-%%h-%%p-%%r

# Enabling pipelining reduces the number of SSH operations required to
# execute a module on the remote server. This can result in a significant
# performance improvement when enabled, however when using "sudo:" you must
# first disable 'requiretty' in /etc/sudoers
#
# By default, this option is disabled to preserve compatibility with
# sudoers configurations that have requiretty (the default on many distros).
#
#pipelining = False

# if True, make ansible use scp if the connection type is ssh
# (default is sftp)
#scp_if_ssh = True

# if False, sftp will not use batch mode to transfer files. This may cause some
# types of file transfer failures impossible to catch however, and should
# only be disabled if your sftp version has problems with batch mode
#sftp_batch_mode = False

[accelerate]
#accelerate_port = 5099
#accelerate_timeout = 30
#accelerate_connect_timeout = 5.0

# The daemon timeout is measured in minutes. This time is measured
# from the last activity to the accelerate daemon.
#accelerate_daemon_timeout = 30

# If set to yes, accelerate_multi_key will allow multiple
# private keys to be uploaded to it, though each user must
# have access to the system via SSH to add a new key. The default
# is "no".
```

```
#accelerate_multi_key = yes

[selinux]
# file systems that require special treatment when dealing with security context
# the default behaviour that copies the existing context or uses the user default
# needs to be changed to use the file system dependent context.
#special_context_filesystems=nfs,vboxsf,fuse,ramfs

# Set this to yes to allow libvirt_lxc connections to work without SELinux.
#libvirt_lxc_noseclabel = yes

[colors]
#highlight = white
#verbose = blue
#warn = bright purple
#error = red
#debug = dark gray
#deprecate = purple
#skip = cyan
#unreachable = red
#ok = green
#changed = yellow
#diff_add = green
#diff_remove = red
#diff_lines = cyan
```

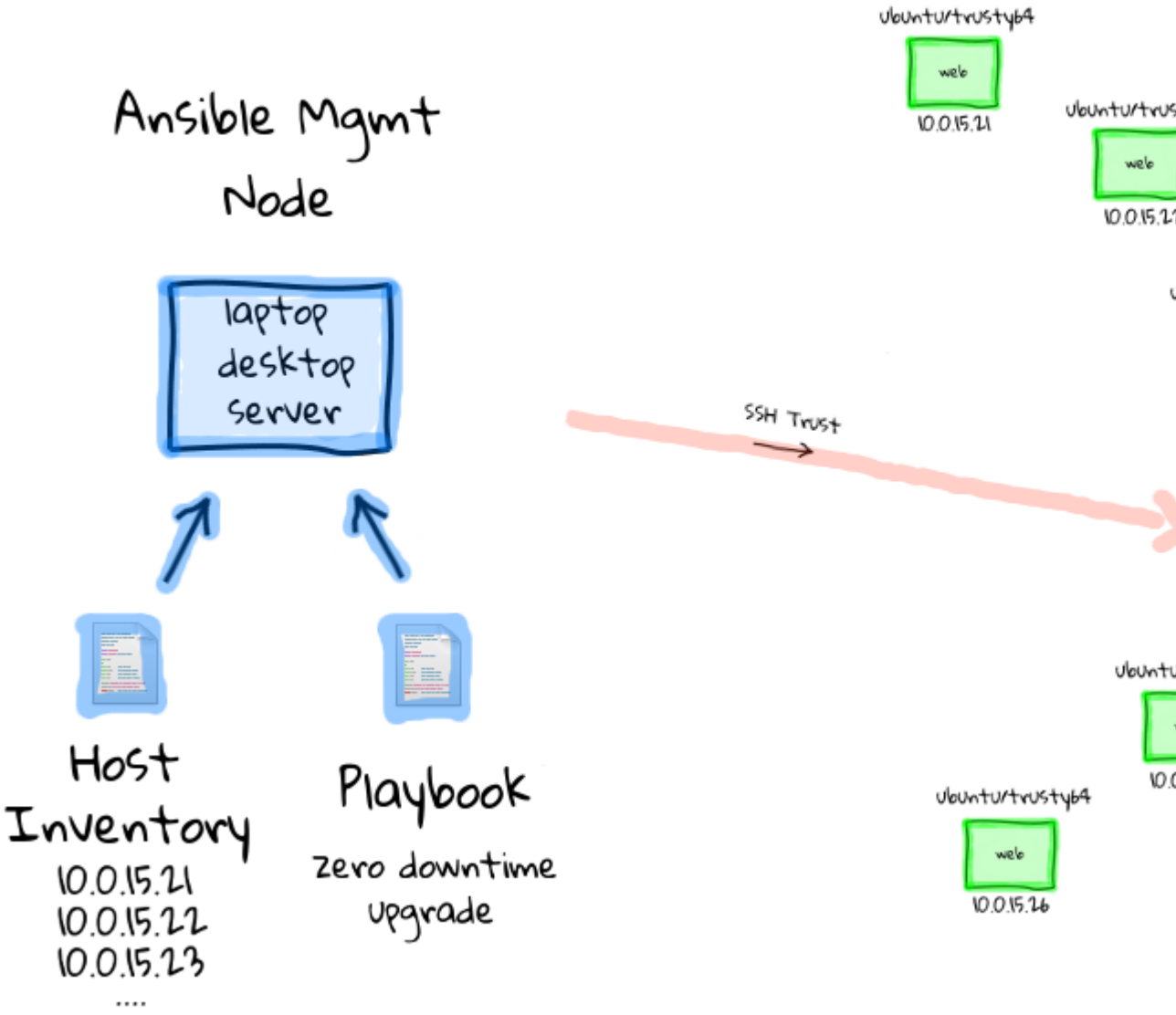
Put this configuration in the root of your role directories to change the behavior of Ansible when using that role. For example, you can set it to stop create `playbook.retry` on failed playbook runs or to point to secret vars that you don't want in your git repo.

Read Getting started with ansible online: https://riptutorial.com/ansible/topic/826/getting-started-with-ansible

Chapter 2: Ansible Architecture

Examples

Understanding Ansible Architecture



The idea is to have one or more control machines from where you can issue ad-hoc commands to remote machines (via `ansible` tool) or run a sequenced instruction set via playbooks (via `ansible-playbook` tool).

Basically, we use Ansible control machine, this will typically be your desktop, laptop or server. Then from there, you use Ansible to push configuration changes out, via ssh.

The host inventory file determines the target machines where these plays will be executed. The Ansible configuration file can be customized to reflect the settings in your environment.

Read Ansible Architecture online: https://riptutorial.com/ansible/topic/7659/ansible-architecture

Chapter 3: Ansible group variables

Examples

Group variables with static inventory

It is suggested that you define groups based on purpose of the host (roles) and also geography or datacenter location (if applicable):

File `inventory/production`

```
[rogue-server]
192.168.1.1

[atlanta-webservers]
www-atl-1.example.com
www-atl-2.example.com

[boston-webservers]
www-bos-1.example.com
www-bos-2.example.com

[atlanta-dbservers]
db-atl-1.example.com
db-atl-2.example.com

[boston-dbservers]
db-bos-1.example.com

# webservers in all geos
[webservers:children]
atlanta-webservers
boston-webservers

# dbservers in all geos
[dbservers:children]
atlanta-dbservers
boston-dbservers

# everything in the atlanta geo
[atlanta:children]
atlanta-webservers
atlanta-dbservers

# everything in the boston geo
[boston:children]
boston-webservers
boston-dbservers
```

File `group_vars/all`

```
---
apache_port: 80
```

File `group_vars/atlanta-webservers`

```
---
apache_port: 1080
```

File `group_vars/boston-webservers`

```
---
apache_port: 8080
```

File `host_vars/www-bos-2.example.com`

```
---
apache_port: 8111
```

After running `ansible-playbook -i inventory/hosts install-apache.yml` (hosts in the playbook would be `hosts: all`)

The ports would be

Address	Port
`192.168.1.1`	80
`www-atl-1.example.com`	1080
`www-atl-2.example.com`	1080
`www-bos-1.example.com`	8080
`www-bos-2.example.com`	8111

Read Ansible group variables online: https://riptutorial.com/ansible/topic/6544/ansible-group-variables

Chapter 4: Ansible Group Vars

Examples

Example group_vars/development, and why

Project structure

```
project/
  group_vars/
     development
  inventory.development
  playbook.yaml
```

These variables will be applied to hosts under the development group due to the filename.

```
---
## Application
app_name: app
app_url: app.io
web_url: cdn.io
app_friendly: New App
env_type: production
app_debug: false

## SSL
ssl: true
ev_ssl: false

## Database
database_host: 127.0.0.1
database_name: app
database_user: sql

## Elasticsearch
elasticsearch_host: 127.0.0.1
```

Read Ansible Group Vars online: https://riptutorial.com/ansible/topic/6226/ansible-group-vars

Chapter 5: Ansible install mysql

Introduction

How use ansible to install mysql binary file

Examples

How use ansible to install mysql binary file

- hosts: mysql tasks:
    - name: Add mysql user user: name: mysql shell: /sbin/nologin
    - name: install the latest version of libselinux-python yum: name: libselinux-python state: latest
    - name: install perl yum: name: perl state: latest
    - name: remove the mysql-libs package yum: name: mysql-libs state: absent

```
- name: download and unarchive tar
  unarchive:
     src=/tmp/mysql-5.6.35-linux-glibc2.5-x86_64.tar.gz
     dest=/tmp
     copy=yes


- name: Move mysql paceage to specified directory
  command: creates="/usr/local/mysql" mv /tmp/mysql-5.6.35-linux-glibc2.5-x86_64
/usr/local/mysql

- name: chown mysql mysql /usr/local/mysql
  file: path=/usr/local/mysql owner=mysql group=mysql recurse=yes


- name: Add lib to ld.so.conf
  lineinfile: dest=/etc/ld.so.conf line="/usr/local/mysql/lib/"

- name: ldconfig
  command: /sbin/ldconfig

- name: Mkdir mysql_data_dir
  file: path=/data/mysql/3306/{{ item }} state=directory owner=mysql group=mysql
  with_items:
    - data
    - logs
    - tmp

- name: Copy mysql my.cnf
  copy: src=/etc/my.cnf dest=/etc/my.cnf
```

```
- name: Copy mysql my.cnf
  copy: src=/etc/my.cnf dest=/usr/local/mysql/my.cnf


- name: Init mysql db
  command: /usr/local/mysql/scripts/mysql_install_db  \
    --user=mysql  \
    --basedir=/usr/local/mysql \
    --datadir=/data/mysql/3306/data

- name: Add mysql bin to profile
  lineinfile: dest=/etc/profile line="export PATH=$PATH:/usr/local/mysql/bin/"

- name: Source profile
  shell: executable=/bin/bash source /etc/profile

- name: Copy mysqld to init when system start
  command: cp -f /usr/local/mysql/support-files/mysql.server /etc/init.d/mysqld

- name: Add mysqld to system start
  command: /sbin/chkconfig --add mysqld

- name: Add mysql to system start when init 345
  command: /sbin/chkconfig --level 345 mysqld on

- name: Retart mysql
  service: name=mysqld state=restarted
```

Read Ansible install mysql online: https://riptutorial.com/ansible/topic/10920/ansible-install-mysql

Chapter 6: Ansible: Looping

Examples

with_items - simple list

A `with_items` loop in ansible can be used to easily loop over values.

```
- name: Add lines to this file
  lineinfile: dest=/etc/file line={{ item }} state=present
  with_items:
    - Line 1
    - Line 2
    - Line 3
```

with_items - predefined list

You can also loop over a variable list.

From vars:

```
favorite_snacks:
  - hotdog
  - ice cream
  - chips
```

and then the loop:

```
- name: create directories for storing my snacks
  file: path=/etc/snacks/{{ item }} state=directory
  with_items: '{{ favorite_snacks }}'
```

If you are using Ansible 2.0+ you must use quotes around the call to the variable.

with_items - predefined dictionary

It is possible to create more complex loops with dictionaries.

From vars:

```
packages:
  - present: tree
  - present: nmap
  - absent: apache2
```

then the loop:

```
- name: manage packages
```

```
  package: name={{ item.value }} state={{ item.key }}
  with_items: '{{ packages }}'
```

Or, if you don't like to use the key value:

vars:

```
packages:
  - name: tree
    state: present
  - name: nmap
    state: present
  - name: apache2
    state: absent
```

then the loop:

```
- name: manage packages
  package: name={{ item.name }} state={{ item.state }}
  with_items: '{{ packages }}'
```

with_items - dictionary

You can use a dictionary for a slightly more complex loop.

```
- name: manage packages
  package: name={{ item.name }} state={{ item.state }}
  with_items:
    - { name: tree, state: present }
    - { name: nmap, state: present }
    - { name: apache2, state: absent }
```

Nested loops

You can create nested loops using `with_nested`.

from vars:

```
keys:
  - key1
  - key2
  - key3
  - key4
```

then the loop:

```
- name: Distribute SSH keys among multiple users
  lineinfile: dest=/home/{{ item[0] }}/.ssh/authorized_keys line={{ item[1] }} state=present
  with_nested:
    - [ 'calvin', 'josh', 'alice' ]
    - '{{ keys }}'
```

This task will loop over each user and populate their `authorized_keys` file with the 4 keys defined in the list.

Read Ansible: Looping online: https://riptutorial.com/ansible/topic/6414/ansible--looping

Chapter 7: Ansible: Loops and Conditionals

Remarks

Official docs explains playbook conditionals.

- http://docs.ansible.com/ansible/playbooks_conditionals.html

Ansible (github)

- https://github.com/marxwang/ansible-learn-resources

Examples

What kinds of conditionals to use?

Use Conditionals via (syntax is in `[brackets]`):

- when [**when:**]

```
Task:
   - name: run if operating system is debian
     command: echo "I am a Debian Computer"
     when: ansible_os_family == "Debian"
```

- loops [**with_items:**]
- loops [**with_dicts:**]
- Custom Facts [**when:** my_custom_facts == '1234']
- Conditional imports
- Select files and Templates based on variables

[When] Condition: `ansible_os_family` Lists

Common use

- when: ansible_os_family == "CentOS"
- when: ansible_os_family == "Redhat"
- when: ansible_os_family == "Darwin"
- when: ansible_os_family == "Debian"
- when: ansible_os_family == "Windows"

All Lists

based on discuss here http://comments.gmane.org/gmane.comp.sysutils.ansible/4685

```
OS_FAMILY = dict(
            RedHat = 'RedHat',
            Fedora = 'RedHat',
            CentOS = 'RedHat',
            Scientific = 'RedHat',
            SLC = 'RedHat',
            Ascendos = 'RedHat',
            CloudLinux = 'RedHat',
            PSBM = 'RedHat',
            OracleLinux = 'RedHat',
            OVS = 'RedHat',
            OEL = 'RedHat',
            Amazon = 'RedHat',
            XenServer = 'RedHat',
            Ubuntu = 'Debian',
            Debian = 'Debian',
            SLES = 'Suse',
            SLED = 'Suse',
            OpenSuSE = 'Suse',
            SuSE = 'Suse',
            Gentoo = 'Gentoo',
            Archlinux = 'Archlinux',
            Mandriva = 'Mandrake',
            Mandrake = 'Mandrake',
            Solaris = 'Solaris',
            Nexenta = 'Solaris',
            OmniOS = 'Solaris',
            OpenIndiana = 'Solaris',
            SmartOS = 'Solaris',
            AIX = 'AIX',
            Alpine = 'Alpine',
            MacOSX = 'Darwin',
            FreeBSD = 'FreeBSD',
            HPUX = 'HP-UX'
        )
```

When Condition

Basic Usage

Use the when condition to control whether a task or role runs or is skipped. This is normally used to change play behavior based on facts from the destination system. Consider this playbook:

```
- hosts: all
  tasks:
    - include: Ubuntu.yml
      when: ansible_os_family == "Ubuntu"

    - include: RHEL.yml
      when: ansible_os_family == "RedHat"
```

Where `Ubuntu.yml` and `RHEL.yml` include some distribution-specific logic.

Another common usage is to limit results to those in certain Ansible inventory groups. Consider this inventory file:

```
[dbs]
mydb01

[webservers]
myweb01
```

And this playbook:

```
- hosts: all
  tasks:
    - name: Restart Apache on webservers
      become: yes
      service:
        name: apache2
        state: restarted
      when: webservers in group_names
```

This is using the `group_names` magic variable.

Conditional Syntax and Logic

Single condition

Syntax

```
when: (condition)
```

Example

- `when: ansible_os_family == "Debian"`
- `when: ansible_pkg_mgr == "apt"`
- `when: myvariablename is defined`

Boolean Filter

Example

```
when: result|failed
```

Multiple Conditions

Syntax

```
When: condition1 and/or condition2
```

Example (simple)

```
when: ansible_os_family == "Debian" and ansible_pkg_mgr == "apt"
```

Example (complex)

Use parentheses for clarity or to control precedence. "AND" has a higher precedence than "OR".

Clauses can span lines:

```
when:
  ansible_distribution in ['RedHat', 'CentOS', 'ScientificLinux'] and
  (ansible_distribution_version|version_compare('7', '<') or
  ansible_distribution_version|version_compare('8', '>='))
  or
  ansible_distribution == 'Fedora'
  or
  ansible_distribution == 'Ubuntu' and
  ansible_distribution_version|version_compare('15.04', '>=')
```

Note the use of parentheses to group the "or" in the first distribution check.

Get `ansible_os_family` and `ansible_pkg_mgr` with setup

We can get facts (`ansible_os_family`, `ansible_pkg_mgr`) with Ad-Hoc command of setup module and filter.

- ansible_os_family:

```
$ ansible all -m setup -a 'filter=ansible_os_family'
ra.local | SUCCESS => {
    "ansible_facts": {
        "ansible_os_family": "Debian"
    },
    "changed": false
}
```

- ansible_pkg_mgr:

```
$ ansible all -m setup -a 'filter=ansible_pkg_mgr'
debian.local | SUCCESS => {
    "ansible_facts": {
        "ansible_pkg_mgr": "apt"
    },
    "changed": false
}
```

Simple "When" Example(s)

Given:

```
---
variable_name: True
```

Then, these tasks with always run.

```
- name: This is a conditional task
  module: src=/example/ dest=/example
  when: variable_name

- name: This is a conditional task
  module: src=/example/ dest=/example
  when: True
```

This task will never run.

```
- name: This is a conditional task
  module: src=/example/ dest=/example
  when: False
```

Using until for a retry looping alive check

This is an example of using until/retries/delay to implement an alive check for a webapp that is starting up. It assumes that there will be some period of time (up to 3 minutes) where the webapp is refusing socket connections. After that, it checks the /alive page for the word "OK". It also delegates the retrieval of the URL to the localhost running ansible. This makes sense as the final task in a deployment playbook.

```
---
- hosts: my-hosts
  tasks:
  - action: uri url=http://{{ ansible_all_ipv4_addresses }}:8080/alive return_content=yes
    delegate_to: localhost
    register: result
    until: "'failed' not in result and result.content.find('OK') != -1"
    retries: 18
    delay: 10
```

The until retry pattern can be used with any action; Ansible documentation provides an example of waiting until a certain shell command returns a desired result: http://docs.ansible.com/ansible/playbooks_loops.html#do-until-loops.

Read Ansible: Loops and Conditionals online: https://riptutorial.com/ansible/topic/3555/ansible--loops-and-conditionals

Chapter 8: Become (Privilege Escalation)

Introduction

Often you need to execute commands under a different user or get *root* privileges. Those options allow you to **become** another user in the guest system.

Syntax

- `become`: can be set to true or yes and triggers the user escalation settings.
- `become_user`: set to the desired user in the remote host.
- `become_method`: specify the command used to make login and change user.
- `become_flags`: change login parameters. Mostly used when you want to change to a system user without shell privileges.

Examples

Only in a task

```
- name: Run script as foo user
  command: bash.sh
  become: true
  become_user: foo
```

Run all role tasks as root

```
- hosts: all
  become: true

- name: Start apache
  service: apache2
  state: started
```

Run a role as root

```
- hosts: all
  roles:
    - { role: myrole, become: yes }
    - myrole2
```

Read Become (Privilege Escalation) online: https://riptutorial.com/ansible/topic/8328/become--privilege-escalation-

Chapter 9: Dynamic inventory

Remarks

Environment variables in dynamic inventory won't work, f.e.

```
"ansible_ssh_private_key_file": $HOME/.ssh/key.pem"
```

If the dynamic inventory server side passes `$HOME` for example, replace the variable in the client code (Python):

```
json_input.replace("$HOME", os.environ.get("HOME"))
```

Examples

Dynamic inventory with login credentials

Pass dynamic inventory to `ansible-playbook`:

```
ansible-playbook -i inventory/dyn.py -l targethost my_playbook.yml
```

`python inventory/dyn.py` should print out something like this:

```
{
  "_meta": {
    "hostvars": {
      "10.1.0.10": {
        "ansible_user": "vagrant",
        "ansible_ssh_private_key_file": "/home/mrtuovinen/.ssh/id_rsa",
        "ansible_port": 22
      },
      "10.1.0.11": {
        "ansible_user": "ubuntu",
        "ansible_ssh_private_key_file": "/home/mrtuovinen/.ssh/id_rsa",
        "ansible_port": 22
      },
      "10.1.0.12": {
        "ansible_user": "steve",
        "ansible_ssh_private_key_file": "/home/mrtuovinen/.ssh/key.pem",
        "ansible_port": 2222
      }
    }
  },
  "vagrantbox": [
    "10.1.0.10"
  ],
  "ubuntubox": [
    "10.1.0.11"
  ],
  "osxbox": [
```

```
    "10.1.0.12"
  ]
}
```

Read Dynamic inventory online: https://riptutorial.com/ansible/topic/1758/dynamic-inventory

Chapter 10: Galaxy

Examples

Sharing roles with Ansible Galaxy

It's also possible to easily share roles with the community or download roles that have been created by other members of the community with Ansible Galaxy.

Ansible ships with a command line tool called `ansible-galaxy` that can be used to install roles in the role directory defined in the `ansible.cfg` file:

```
ansible-galaxy install username.rolename
```

You can also use the Ansible Galaxy tool to download roles from other locations such as GitHub by creating a text file with the location defined as `src`:

```
- src: https://github.com/username/rolename
```

And then install the roles in the text file like so:

```
ansible-galaxy install -r requirements.txt
```

You can also use the `ansible-galaxy` tool to create role "scaffolding":

```
ansible-galaxy init rolename
```

Once you've created a role and uploaded it to GitHub you can share it on Ansible Galaxy by linking to your GitHub repo in Ansible Galaxy after signing in.

More examples under Galaxy topic.

Read Galaxy online: https://riptutorial.com/ansible/topic/6599/galaxy

Chapter 11: Galaxy

Examples

Basic commands

Search role in Ansible Galaxy

```
ansible-galaxy search role_name
```

Install role from Ansible Galaxy

```
ansible-galaxy install role_name
```

More help

```
ansible-galaxy --help
```

Read Galaxy online: https://riptutorial.com/ansible/topic/6656/galaxy

Chapter 12: How To Create A DreamHost Cloud Server From An Ansible Playbook

Examples

Install Shade library

Shade is a library developed by OpenStack to simplify interactions with OpenStack clouds, like DreamHost.

$ pip install shade

Write a Playbook to Launch a Server

Create a file named `launch-server.yaml`, that will be our playbook.

The first part of the playbook is a list of hosts that your playbook will run on, we only have one, localhost.

```
- hosts: localhost
```

Then we need to define a list of tasks to perform in this playbook. We will only have one that launches an Ubuntu Xenial server on DreamCompute.

```
tasks:
  - name: launch an Ubuntu server
```

Next part of the playbook uses the `os_server` (OpenStack Server) module. This defines what the server has to look like in DreamCompute.

```
os_server:
```

First step is to authenticate to DreamCompute; substitute `{username}` with your DreamCompute username, `{password}` with your DreamCompute password, and `{project}` with your DreamCompute project. You'll find those in the OpenStack RC file.

```
  auth:
    auth_url: https://iad2.dream.io:5000
    username: {username}
    password: {password}
    project_name: {project}
```

Next lines define some elements of the new server.

```
  state: present
```

```
  name: ansible-vm1
  image: Ubuntu-16.04
  key_name: {keyname}
  flavor: 50
  network: public
  wait: yes
```

Lets break down the previous few lines:

- `state` is the state of the server, possible values are `present` or `absent`
- `name` is the name of the server to create; can be any value
- `image` is the image to boot the server from; possible values are visible on DreamHost Cloud web panel; the variable accepts either image name or UUID
- `key_name` is the name of the public key to add to the server once it is created; this can be any key has already been added to DreamCompute.
- `flavor` is the flavor of server to boot; this defines how much RAM and CPU your server will have; the variable accepts either the name of a flavor (gp1.semisonic) or the ID (50, 100, 200, etc)
- `network` is the network to put your server on. In DreamHost Cloud case it is the `public` network.
- `wait` set to yes forces the playbook to wait for the server to be created before continuing.

Running the Playbook

Run the Ansible playbook:

```
$ ansible-playbook launch-server.yaml
```

You should see output like

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

TASK [setup]
*****************************************************************
ok: [localhost]

TASK [launch an Ubuntu server]
*********************************************
changed: [localhost]

PLAY RECAP
*****************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0
```

Now if you check the DreamHost Cloud dashboard you should see a new instance named “ansible-vm1”

Read How To Create A DreamHost Cloud Server From An Ansible Playbook online: https://riptutorial.com/ansible/topic/4689/how-to-create-a-dreamhost-cloud-server-from-an-ansible-playbook

Chapter 13: Installation

Introduction

Installing Ansible in any OS, including Windows using Virtual Box and Vagrant. An alternate solution is also available if you just want to practice ansible ad-hoc commands and playbooks and do not wish to set up the local environment.

Examples

Installing Ansible on Ubuntu

Ansible maintains a PPA repository that can be used to install the Ansible binaries:

```
sudo apt-add-repository ppa:ansible/ansible -y
sudo apt-get update && sudo apt-get install ansible -y
```

To install a specific version, use `pip`. The PPA may be out of date.

Installing Ansible on MacOS

There are two main ways way to install Ansible on OS X, either using the Homebrew or Pip package manager.

If you have homebrew, the latest Ansible can be installed using the following command:

```
brew install ansible
```

To install Ansible 1.9.X branch use following command:

```
brew install homebrew/versions/ansible19
```

To install Ansible 2.0.X branch use following command:

```
brew install homebrew/versions/ansible20
```

To install using pip, use the following command: `pip install ansible`.

To install a specific version, use `pip install ansible=<required version>`.

Installation on Red Hat based systems

Ansible can be installed on CentOS or other Red Hat based systems. Firstly you should install the prerequisites:

```
sudo yum -y update
sudo yum -y install gcc libffi-devel openssl-devel python-pip python-devel
```

then install Ansible with pip:

```
sudo pip install ansible
```

I can recommend for you to upgrade the setuptools after the installation:

```
sudo pip install --upgrade setuptools
```

You can also use the local Package Manager as well:

```
yum install ansible
```

Installing from source

Ansible is **best used** from a checkout.

It runs as you (not root) and it has minimal python dependencies.

Python pip dependency install with pip:

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

Next, clone the Ansible repo from GitHub:

```
cd ~/Documents
git clone git://github.com/ansible/ansible.git --recursive
cd ansible
```

Finally, add the ansible initialization script line to your ~/.bashrc or ~/.zshrc :

```
source ~/Documents/ansible/hacking/env-setup
```

Restart your terminal session, and test with

```
ansible --version
```

Installation on Amazon Linux from git repo

Amazon Linux is a RHEL variant, so the Red Hat instructions should work for the most part. There is, however, at least one discrepancy.

There was an instance where the **python27-devel** package, as opposed to **python-devel**, was explicitly necessary.

Here, we will install from source.

```
sudo yum -y update
sudo yum -y install python27 python27-devel openssl-devel libffi-devel gcc git

git clone https://github.com/ansible/ansible/<search the github for a preferable branch>

cd ansible
sudo python setup.py build
sudo python setup.py install
```

Installing Ansible On Any OS(windows) Machine Using Virtual Box+Vagrant

My laptop is having Windows 10. Here i am giving steps that you can follow to test and learn Ansible.

SOME THEORY

For Ansible you need a Control Machine and a host(or hosts) to run the Playbook.

- **Control Machine** should be Linux based or MacOS(windows not allowed) and need Python (2.6 or higher version). Here Ansible will be installed.
- **Target machine** (host/node) can be Linux/MacOS/windows. This needs only Python to be installed. No agent software required.

SETUP

Step 1: **Install Virtual Box**

Virtual box is a software to create virtual computers of different OS. It is like having multiple computers each or different OS and different versions.

Download Virtual Box according to the OS in your system and install it.

Step 2: **Install Vagrant**

Vagrant is Command Line Interface to create virtual machines in virtual box. This makes things easy. You need to learn basic Vagrant commands.

Step 3: **Create a folder where you want your virtual machine**

Step 4: **Create Virtual Machine using Vagrant**

Open terminal and go to the path where you created folder, and run the following two commands.

You need to select **Virtual Box**. I am installing Ubuntu for example. You can choose anything from the list. You need to run these two commands under "**virtual box**" category: `vagrant init ubuntu/trusty64` and `vagrant up --provider virtualbox`. Other categories might be: hyperv, vmware_desktop etc. (this will take some time, as it will download the necessary files)

Step 4: Install Ansible

For UbuntuOS: `sudo apt-get install ansible`

Alternative solution:

You can use **Katacoda** to practice ansible. No need to install or setup anything. Run two commands given in step 2 and after that, you are good to go.

Read Installation online: https://riptutorial.com/ansible/topic/4906/installation

Chapter 14: Introduction to playbooks

Examples

Overview

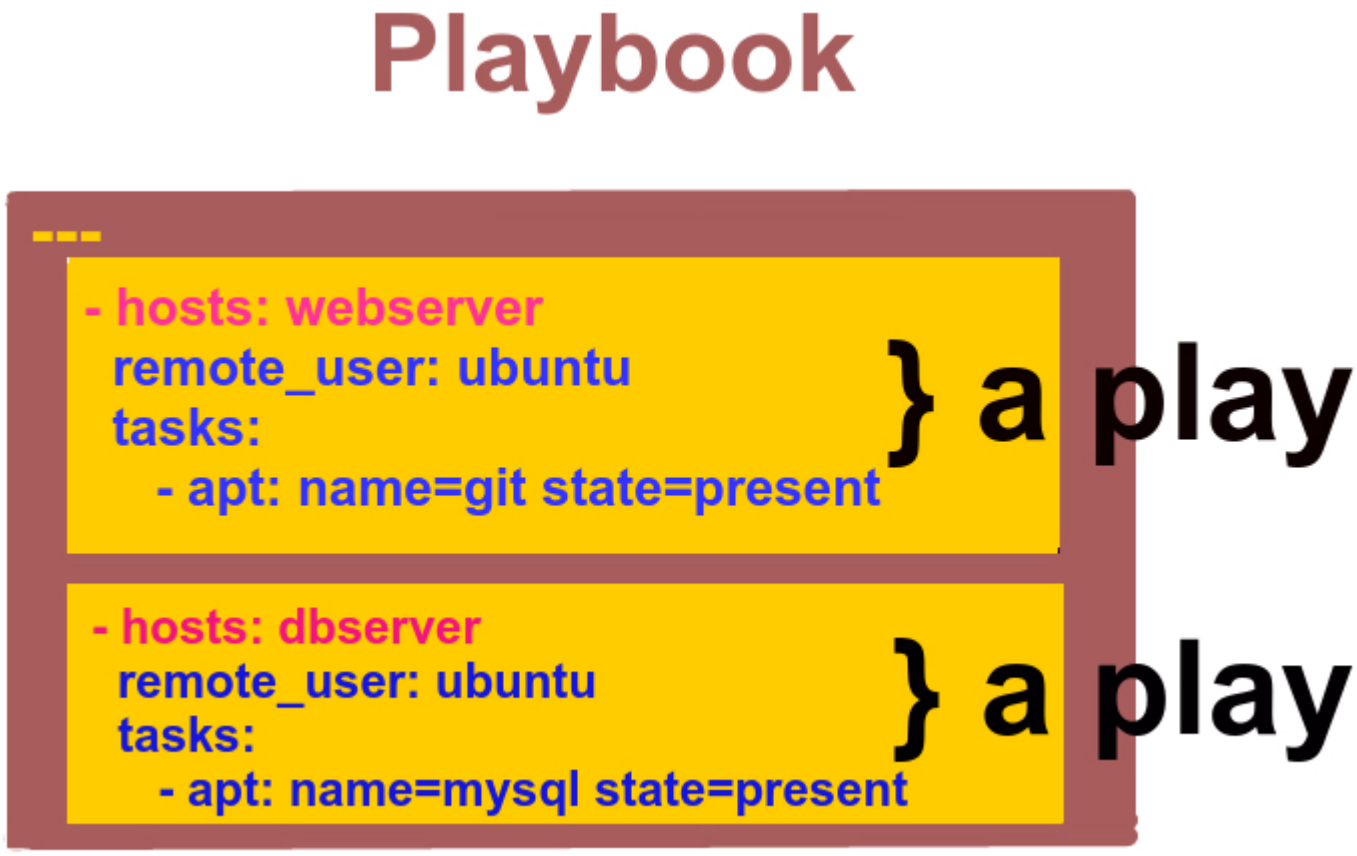
In Ansible, a playbook is is a YAML file containing the definition of how a server should look. In a playbook you define what actions Ansible should take to get the server in the state you want. Only what you define gets done.

This is a basic Ansible playbook that installs git on every host belonging to the `web` group:

```
---
- name: Git installation
  hosts: web
  remote_user: root
  tasks:
    - name: Install Git
      apt: name=git state=present
```

Playbook's structure

The format of a playbook is quite straightforward, but strict in terms of spacing and layout. A playbook consists of plays. A play is a combination of targets hosts and the tasks we want to apply on these hosts, so a drawing of a playbook is this:



To execute this playbook, we simply run:

```
ansible-playbook -i hosts my_playbook.yml
```

Play's structure

Here's a simple play:

```
- name: Configure webserver with git
  hosts: webserver
  become: true
  vars:
    package: git
  tasks:
    - name: install git
      apt: name={{ package }} state=present
```

As we said earlier, every play must contain:

- A set of hosts to configure
- A list of tasks to be executed on those hosts

Think of a play as the thing that connects hosts to tasks. In addition to specifying hosts and tasks, plays also support a number of optional settings. Two common ones are:

- `name`: a comment that describes what the play is about. Ansible will print this out when the

play starts to run
- `vars`: a list of variables and values

Tags

Play contains several tasks, which can be tagged:

```
- name: Install applications
  hosts: all
  become: true
  tasks:
    - name: Install vim
      apt: name=vim state=present
      tags:
        - vim
    - name: Install screen
      apt: name=screen state=present
      tags:
        - screen
```

Task with tag 'vim' will run when 'vim' is specified in tags. You can specify as many tags as you want. It is useful to use tags like 'install' or 'config'. Then you can run playbook with specifying tags or skip-tags. For

```
ansible-playbook my_playbook.yml --tags "tag1,tag2"
ansible-playbook my_playbook.yml --tags "tag2"
ansible-playbook my_playbook.yml --skip-tags "tag1"
```

By default Ansible run all tags

Read Introduction to playbooks online: https://riptutorial.com/ansible/topic/3343/introduction-to-playbooks

Chapter 15: Inventory

Parameters

Parameter	Explanation
ansible_connection	Connection type to the host. This can be the name of any of ansible's connection plugins. SSH protocol types are `smart`, `ssh` or `paramiko`. The default is smart. Non-SSH based types are described in the next section.
ansible_host	The name of the host to connect to, if different from the alias you wish to give to it.
ansible_port	The ssh port number, if not 22
ansible_user	The default ssh user name to use.
ansible_ssh_pass	The ssh password to use (this is insecure, we strongly recommend using `--ask-pass` or SSH keys)
ansible_ssh_private_key_file	Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.
ansible_ssh_common_args	This setting is always appended to the default command line for **sftp**, **scp**, and **ssh**. Useful to configure a `ProxyCommand` for a certain host (or group).
ansible_sftp_extra_args	This setting is always appended to the default **sftp** command line.
ansible_scp_extra_args	This setting is always appended to the default **scp** command line.
ansible_ssh_extra_args	This setting is always appended to the default **ssh** command line.
ansible_ssh_pipelining	Determines whether or not to use SSH pipelining. This can override the `pipelining` setting in `ansible.cfg`.
ansible_become	Equivalent to `ansible_sudo` or `ansible_su`, allows to force privilege escalation
ansible_become_method	Allows to set privilege escalation method
ansible_become_user	Equivalent to `ansible_sudo_user` or `ansible_su_user`, allows to set the user you become through privilege escalation

Parameter	Explanation
ansible_become_pass	Equivalent to `ansible_sudo_pass` or `ansible_su_pass`, allows you to set the privilege escalation password
ansible_shell_type	The shell type of the target system. You should not use this setting unless you have set the `ansible_shell_executable` to a non-Bourne (sh) compatible shell. By default commands are formatted using `sh`-style syntax. Setting this to `csh` or `fish` will cause commands executed on target systems to follow those shell's syntax instead.
ansible_python_interpreter	The target host python path. This is useful for systems with more than one Python or not located at **/usr/bin/python** such as *BSD, or where **/usr/bin/python** is not a 2.X series Python. We do not use the **/usr/bin/env** mechanism as that requires the remote user's path to be set right and also assumes the **python** executable is named python, where the executable might be named something like **python2.6**.
ansible_*_interpreter	Works for anything such as ruby or perl and works just like `ansible_python_interpreter`. This replaces shebang of modules which will run on that host.
ansible_shell_executable	This sets the shell the ansible controller will use on the target machine, overrides `executable` in `ansible.cfg` which defaults to **/bin/sh**. You should really only change it if is not possible to use **/bin/sh** (i.e. **/bin/sh** is not installed on the target machine or cannot be run from sudo.). New in version 2.1.

Examples

Inventory with username and password

Inventory is the Ansible way to track all the systems in your infrastructure. Here is a simple inventory file containing a single system and the login credentials for Ansible.

```
[targethost]
192.168.1.1 ansible_user=mrtuovinen ansible_ssh_pass=PassW0rd
```

Inventory with custom private key

```
[targethost]
192.168.1.1 ansible_user=mrtuovinen ssh_private_key_file=~/.ssh/custom_key
```

Inventory with custom SSH port

```
[targethost]
192.168.1.1 ansible_user=mrtuovinen ansible_port=2222
```

Pass static inventory to ansible-playbook

```
ansible-playbook -i path/to/static-inventory-file -l myhost myplaybook.yml
```

Pass dynamic inventory to ansible-playbook

```
ansible-playbook -i path/to/dynamic-inventory-script.py -l myhost myplaybook.yml
```

See dynamic inventory for more details.

Inventory, Group Vars, and You

project structure (ansible best practice).

```
project/
  group_vars/
     development
  inventory.development
  playbook.yaml
```

it all starts with inventory.development

```
[development]
dev.fakename.io

[development:vars]
ansible_host: 192.168.0.1
ansible_user: dev
ansible_pass: pass
ansible_port: 2232

[api:children]
development
```

which lets you link to group_vars. Hold data 'specific' to that environment ...

```
---
app_name: NewApp_Dev
app_url: https://dev.fakename.io
app_key: f2390f23f01233f23f
```

that lets one run the following playbook AGAINST the inventory file:

```
---
- name: Install api.
  hosts: api
  gather_facts: true
  sudo: true
```

```
  tags:
    - api
  roles:
    - { role: api,         tags: ["api"]         }
```

with the following runline:

```
ansible-playbook playbook.yaml -i  inventory.development
```

Hosts file

The host file is used to store connections for Anisble playbooks. There are options to define connection parameters:

`ansible_host` is the hostname or IP address

`ansible_port` is the port the machine uses for SSH

`ansible_user` is the remote user to connect as

`ansible_ssh_pass` if using a password to SSH

`ansible_ssh_private_key_file` if you need to use multiple keys that are specific to hosts

These are the most commonly used options. More can be found in the Ansible official documentation.

Here is an example `hosts` file:

```
# Consolidation of all groups
[hosts:children]
web-servers
offsite
onsite
backup-servers

[web-servers]
server1 ansible_host=192.168.0.1 ansible_port=1600
server2 ansible_host=192.168.0.2 ansible_port=1800

[offsite]
server3 ansible_host=10.160.40.1 ansible_port=22 ansible_user=root
server4 ansible_host=10.160.40.2 ansible_port=4300 ansible_user=root

# You can make groups of groups
[offsite:children]
backup-servers

[onsite]
server5 ansible_host=10.150.70.1 ansible_ssh_pass=password

[backup-servers]
server6 ansible_host=10.160.40.3 ansible_port=77
```

Read Inventory online: https://riptutorial.com/ansible/topic/1764/inventory

Chapter 16: Loops

Examples

Copy multiple files in a single task

```
- name: copy ssl key/cert/ssl_include files
  copy: src=files/ssl/{{ item }} dest=/etc/apache2/ssl/
  with_items:
    - g_chain.crt
    - server.crt
    - server.key
    - ssl_vhost.inc
```

Install multiple packages in a single task

```
- name: Installing Oracle Java and support libs
  apt: pkg={{ item }}
  with_items:
    - python-software-properties
    - oracle-java8-installer
    - oracle-java8-set-default
    - libjna-java
```

Read Loops online: https://riptutorial.com/ansible/topic/6095/loops

Chapter 17: Roles

Examples

Using roles

Ansible uses the concept of roles to better allow modular code and avoid repeating yourself.

A role is simply a folder structure that Ansible knows where to load vars files, tasks and handlers from. An example might look something like this:

```
apache/
├── defaults
│   └── main.yml
├── files
│   ├── mod-pagespeed-stable_current_i386.deb
│   ├── mod-pagespeed-stable_current_i386.rpm
│   ├── mod-pagespeed-stable_current_amd64.deb
│   └── mod-pagespeed-stable_current_x86_64.rpm
├── tasks
│   ├── debian.yml
│   ├── main.yml
│   └── redhat.yml
├── templates
│   ├── httpd.conf.j2
│   └── sites-available
│       └── virthualhost.conf.j2
└── vars
    ├── debian
    └── redhat
```

You can then use the role with a basic playbook that just looks like this:

```
- hosts: webservers
  roles:
     - apache
```

When you run Ansible against this playbook it will target all the hosts in the `webservers` group and run the `apache` role defined above against it, automatically loading any default variables for the role and running all the tasks included in `tasks/main.yml`. Ansible also knows to look for certain types of files in role friendly locations:

- If roles/x/tasks/main.yml exists, tasks listed therein will be added to the play
- If roles/x/handlers/main.yml exists, handlers listed therein will be added to the play
- If roles/x/vars/main.yml exists, variables listed therein will be added to the play
- If roles/x/meta/main.yml exists, any role dependencies listed therein will be added to the list of roles (1.3 and later)

- Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely

Role dependencies

Roles also enable you to define other roles as a dependency by creating a `meta/main.yml` file with a `dependencies` block:

```
dependencies:
  - role: common
```

It's also possible to pass a value to a parameter/variable in the dependent role:

```
dependencies:
  - { role: common, some_parameter: 3 }
```

Or even execute the dependent role conditionally:

```
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: sshd, enable_sshd: false,
                        when: environment == 'production' }
```

Dependent roles are always executed before the roles that depend on them. Also, they are only executed once. If two roles state the same one as their dependency, it is only executed the first time.

Imagine the roles role1, role2 and role3 with the folling `meta/main.yml`'s:
role1/meta/main.yml:

```
dependencies:
    - role: role3
```

role2/meta/main.yml:

```
dependencies:
    - role: role3
```

When executing role1 and role2 in the same playbook (with role1 called before role2), the execution order would be the following:

```
role3 -> role1 -> role2
```

You may override this behaviour by specifying `allow_duplicates: yes` in `meta/main.yml` of role1 and role2. The resulting execution order would the be:

```
role3 -> role1 -> role3 -> role2
```

Separating distribution specific tasks and variables inside a role

We can easily separate distribution specific tasks and variables into different dedicated .yml files. Ansible helps us to automatically identify the target hosts distribution via `{{ ansible_distribution }}` and `{{ ansible_distribution_version }}`, so we just have to name the distribution dedicated .yml files accordingly.

For Ubuntu Xenial the basic role dir tree would then look something like that:

```
role
├── tasks
│   ├── main.yml
│   └── Ubuntu16.04.yml
└── vars
    └── Ubuntu16.04.yml
```

Inside the `tasks/main.yml` we can now automatically include the proper variables and tasks for the target hosts distribution.

tasks/main.yml

```
---

- name: include distribution specific vars
  include_vars: "{{ ansible_distribution }}{{ ansible_distribution_version }}.yml"

- name: include distribution specific install
  include: "{{ ansible_distribution }}{{ ansible_distribution_version }}.yml"
```

Inside `tasks/Ubuntu16.06.yml` and `vars/Ubuntu16.04.yml` we can now define tasks and variables for Ubuntu Xenial respectively.

Read Roles online: https://riptutorial.com/ansible/topic/3396/roles

Chapter 18: Secret encryption

Remarks

Ansible offers Vault (not to be mistaken with HashiCorp Vault!) to handle sensitive data encryption. Vault primarily targets to encrypt any structured data such as variables, tasks, handlers.

Examples

Encrypting sensitive structured data

First, create a key file, e.g., `vault_pass_file`, which ideally contains a long sequence of random characters. In linux systems you could use `pwgen` to create a random password file:

```
pwgen 256 1 > vault_pass_file
```

Then, use this file to encrypt sensitive data, e.g., `groups_vars/group.yml`:

```
ANSIBLE_VAULT_PASSWORD_FILE=vault_pass_file ansible-vault encrypt group_vars/group.yml
```

From now on, in order to run a playbook you need the `vault_pass_file`:

```
ANSIBLE_VAULT_PASSWORD_FILE=vault_pass_file ansible-playbook -i inventories/nodes my-
playbook.yml
```

Note, you could also use the flag `--vault-password-file vault_pass_file` instead of setting the `ANSIBLE_VAULT_PASSWORD_FILE` environment variable.

In order to edit or decrypt the secret on disk you can use `ansible-vault edit` and `ansible-vault decrypt` respectively.

Using lookup pipes to decrypt non-structured vault-encrypted data

With Vault you can also encrypt non-structured data, such as private key files and still be able to decrypt them in your play with the `lookup` module.

```
---

- name: Copy private key to destination
  copy:
    dest=/home/user/.ssh/id_rsa
    mode=0600
    content=lookup('pipe', 'ANSIBLE_VAULT_PASSWORD_FILE=vault_pass_file ansible-vault view
keys/private_key.enc')
```

Using local_action to decrypt vault-encrypted templates

You can run a play which relies on vault-encrypted templates by using the `local_action` module.

```
---

- name: Decrypt template
  local_action: "shell {{ view_encrypted_file_cmd }} {{ role_path }}/templates/template.enc >
{{ role_path }}/templates/template"
  changed_when: False

- name: Deploy template
  template:
    src=templates/template
    dest=/home/user/file

- name: Remove decrypted template
  local_action: "file path={{ role_path }}/templates/template state=absent"
  changed_when: False
```

Please note the `changed_when: False`. This is important in case you run idempotence tests with your ansible roles - otherwise each time you run the playbook a change is signaled. In `group_vars/all.yml` you could set a global decrypt command for reuse, e.g., as `view_encrypted_file_cmd`.

group_vars/all.yml

```
---

view_encrypted_file_cmd: "ansible-vault --vault-password-file {{ lookup('env',
'ANSIBLE_VAULT_PASSWORD_FILE') }} view"
```

Now, when running a play you need to set the `ANSIBLE_VAULT_PASSWORD_FILE` environment variable to point to your vault password file (ideally with an absolute path).

Read Secret encryption online: https://riptutorial.com/ansible/topic/3355/secret-encryption

Chapter 19: Using Ansible with Amazon Web Services

Remarks

example-2: This is serves as an example so just don't copy/past it. Instead, to suit your needs, you should customize its variables; ansible_key, security group rules etc..

example-1: To disable the ssh strict host key checking, a behavior we don't want when automating tasks, we set it to `no` in `ansible.cfg` file. ie: `StrictHostKeyChecking=no`

The `ec2.py` file is a python script that executes and returns your AWS ressources based on the `ec2.ini` which is the configuration file that you need to customize if you want to limit the scope of your project to some particular regions, specific tags etc...

Examples

How to start EC2 instance from official Amazon AMIs, modify it and store it as new AMI

This is a very common workflow when using Ansible for provisioning an AWS EC2 instance. This post assumes a basic understand of Ansible and most importantly, assumes you've properly configured it to connect to AWS.

As Ansible official documentation insists, we are going to use four roles:

1- **ami_find** to get the ami id based on which we will launch our EC2 instance.

2- **ec2_ami_creation** to effectively launch the EC2 instance.

3- **code_deploy** for modifying the instance; this could be anything so we will simply transfer a file to the target machine.

4- **build_ami** to build our new image based on the running ec2 instance. This post assumes you are at the top level of your Ansible project: `my_ansible_project`

The first role: **ami_find**

```
cd my_ansible_project/roles && ansible-galaxy init ami_find
```

In this role we are going to use the ec2_ami_find module and as an example, we will search for the an Ubuntu machine and get its **ami_id** (ami-xxxxxxxx). Now edit `my_ansible_project/roles/ami_find/tasks/main.yml` file:

```
---
```

```
- ec2_ami_find:
    name: "ubuntu/images/hvm-ssd/ubuntu-trusty-14.04-amd64-server-*"
    sort: name
    sort_order: descending
    sort_end: 1
    region: "{{ aws_region }}"
  register: ami_find
- set_fact: ami_ubuntu="{{ ami_find.results[0].ami_id }}"
```

The second role: **ec2_ami_creation**

Here, we will use the `ami_id` we got from the first role and then launch our new instance based on it:

```
cd my_ansible_project/roles && ansible-galaxy init ec2_ami_creation
```

In this role we are going to use most importantly the ec2_module to launch our instance. Now edit `my_ansible_project/roles/ec2_ami_creation/tasks/main.yml` file:

```
---
- ec2_vpc_subnet_facts:
    region: "{{aws_region}}"
  register: vpc
- name: creation of security group of the ec2 instance
  ec2_group:
    name: example
    description: an example EC2 group
    region: "{{ aws_region }}"
    rules:
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: 0.0.0.0/0
    state: present
  register: ec2_sg

- name: create instance using Ansible
  ec2:
    key_name: "{{ ansible_key }}"
    group: example
    vpc_subnet_id: "{{vpc.subnets[0].id}}"
    instance_type: "{{ instance_type }}"
    ec2_region: "{{ aws_region }}"
    image: "{{ base_image }}"
    assign_public_ip: yes
    wait: yes
  register: ec2

- set_fact: id={{ec2.instances[0].id}}

- name: adding the newly created instance to a temporary group in order to access it later
from another play
  add_host: name={{ item.public_ip }} groups=just_created
  with_items: ec2.instances

- name: Wait for SSH to come up
  wait_for: host={{ item.public_dns_name }} port=22 delay=10 timeout=640 state=started
  with_items: ec2.instances
```

The third role: **code_deploy**

Here, we will provision this instance, which was added to a group called `just_created`

```
cd my_ansible_project/roles && ansible-galaxy init code_deploy
```

In this role we are going to use the template_module to transfer a file & write the machine hostname in it. Now edit `my_ansible_project/roles/code_deploy/tasks/main.yml` file:

```
---
- template: src=my_file.txt.j2 dest=/etc/my_file.txt
```

then move to templates folder inside your role:

`cd my_ansible_project/roles/templates` and add a file called `my_file.txt.j2` containing:

```
my name is {{ ansible_hostname }}`
```

The fourth role: `build_ami`

We will now create an image of the running instance using the ec2_ami module. Move to you project folder and:

```
 cd my_ansible_project/roles && ansible-galaxy init build_ami
```

Now edit `my_ansible_project/roles/build_ami/tasks/main.yml` file:

```
---
- ec2_ami:
    instance_id: "{{ instance_id }}"
    wait: yes
    name: Base_Image
```

Now, I think you have been wondering how to orchestrate all of these roles. Am I right? If so, continue reading.

We will write a playbook, composed of three plays: first play applicable on `localhost` will call our first two roles, second play applicable on our **just_created** group. last role will be applicable on `localhost`. Why `localhost`? When we want to **manage** some AWS ressources, we use our local machine, as simple as that. Next, we will use a vars file in which we will put our variables: `ansible_key`, `aws_region`, etc...

create infrastructure folder at the top of your project and add a file inside it called `aws.yml`:

```
---
aws_region: ap-southeast-2
ansible_key: ansible
instance_type: t2.small
```

So at the top of your project create `build_base_image.yml` and add this:

```
---
   - hosts: localhost
     connection: local
     gather_facts: False
     vars_files:
       - infrastructure/aws.yml
     roles:
       - ami_find
       - { role: ec2_creation, base_image: "{{ ami_ubuntu }}"}

   - hosts: just_created
     connection: ssh
     gather_facts: True
     become: yes
     become_method: sudo
     roles:
       - code_deploy

   - hosts: localhost
     connection: local
     gather_facts: False
     vars_files:
       - infrastructure/aws.yml
     roles:
       - { role: new_image, instance_id: "{{ id }}"}
```

That's it, Dont forget to delete your ressources after testing this, or why not create a role to delete the running instance :-)

How to properly configure Ansible to connect to Amazon Web Services

Managing AWS resources that scale up and down runs into the limits of the static inventory host file, that's why we need something dynamic. And that's what the dynamic inventories are for. Let's start:

Download these ec2.ini and ec2.py files to the your project folder:

```
cd my_ansible_project
wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.py
wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.ini
```

Once done, make the `ec2.py` file executable:

```
chmod +x ec2.py
```

Now, export your AWS Secret and Access key as environnment variables:

```
export AWS_ACCESS_KEY_ID='ABCDEFGHIJKLM'
export AWS_SECRET_ACCESS_KEY='NOPQRSTUVWXYZ'
```

To use the `ec2.py` script we need the Python AWS SDK, `boto` so you need to install it:

```
sudo pip install boto
```

To test if everything is good, try executing the `ec2.py` by listing your resources:

```
./ec2.py --list
```

you should see something similar to:

```
{
  "_meta": {
    "hostvars": {}
  }
}
```

Now we want to use the dynamic inventory along with our static hosts file. First, create a folder called `inventory`, add `ec2.py`, `ec2.ini` and our `hosts` file to it then tell Ansible to use that folder as an inventory file:

```
mkdir inventory
mv ec2.py inventory/ec2.py
mv ec2.ini inventory/ec2.ini
mv hosts inventory/hosts
```

Next we should define project level configuration for Ansible by creating an Ansible config file in your project folder called `ansible.cfg` and adding this:

```
[defaults]
hostfile = inventory
[ssh_connection]
pipelining = False
ssh_args = -o ControlMaster=auto -o ControlPersist=30m -o StrictHostKeyChecking=no
```

Next we need to configure Ansible to use an SSH key to authenticate access to our EC2 instances. Using an SSH agent is the best way to authenticate with resources, as this makes it easier to manage keys:

```
ssh-agent bash
ssh-add ~/.ssh/keypair.pem
```

That's it! If you followed this, you can test it by using the `ping` module and then, you will see your running instances that have been configured to use your key responding with pong:

```
ansible -m ping all
11.22.33.44 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Read Using Ansible with Amazon Web Services online:
https://riptutorial.com/ansible/topic/3302/using-ansible-with-amazon-web-services

Chapter 20: Using Ansible with OpenStack

Introduction

OpenStack is an open-source software platform for cloud computing. Linux instances can be launched/stopped manualy using the graphical web interface or automated thanks to ansible's openstack cloud module.

Configuring ansible can be tricky, but once well configured using it is really easy and powerfull for testing and Continuous Integration environment.

Parameters

parameters	Comments
hosts: localhost	OpenStack commands are launched from our localhost
gather_facts: False	We dont need to gather information on our localhost
auth_url: https://openstack-identity.mycompany.com/v2.0	use V2.0 URL
state: present	'present' / 'absent' to create/delete the instance
validate_certs: False	usefull if https uses self signed certificates
network: "{{ NetworkName }}"	(optional)
auto_ip: yes	(optional)

Remarks

- We put the authentication URL directly in the playbook, not in a variable. URL used in in vars must be escaped.
- Be carefull with authentication URL version use V2.0 instead of V3 in https://openstack-identity.mycompany.com/v2.0.
- In yml files, be very carefull when copy/paste from browser. Check twice the spaces as they are taken into account.
- More details at: http://docs.ansible.com/ansible/list_of_cloud_modules.html#openstack

Examples

Check your Ansible version

Check the right software versions are installed:

- ansible >=2.0
- python >=2.6
- shade module for python

```
$ansible --version
ansible 2.2.0.0

$python --version
Python 2.7.5
```

Install 'shade' the python component used to pilot openstack.

```
$pip install shade
```

Note : if you use a company proxy, it's always useful to know the right pip synthax

```
$pip install --proxy proxy_ip:proxy_port shade
```

Gather informations from OpenStack GUI to configure Ansible

Our openstack tenant is already set:

- a virtual lan gives instances private IP
- a virtual router map public IP to private IP
- a security key has been generated
- we have default firewall configuration for ssh and port 80
- we are able to launch an instance thanks to the OpenStack web interface

Let gather all needed informations from this web interface.

Authentication informations can be found in the openstack.rc file. this file can be downloaded using the OpenStack webinterface in [access and security/API Access].

```
$cat openstack.rc
#!/bin/bash

# To use an OpenStack cloud you need to authenticate against the Identity
# service named keystone, which returns a **Token** and **Service Catalog**.
# The catalog contains the endpoints for all services the user/tenant has
# access to - such as Compute, Image Service, Identity, Object Storage, Block
# Storage, and Networking (code-named nova, glance, keystone, swift,
# cinder, and neutron).
#
# *NOTE*: Using the 2.0 *Identity API* does not necessarily mean any other
# OpenStack API is version 2.0. For example, your cloud provider may implement
# Image API v1.1, Block Storage API v2, and Compute API v2.0. OS_AUTH_URL is
```

```
# only for the Identity API served through keystone.
export OS_AUTH_URL=https://openstack-identity.mycompany.com/v3

# With the addition of Keystone we have standardized on the term **tenant**
# as the entity that owns the resources.
export OS_TENANT_ID=1ac99fef77ee40148d7d5ba3e070caae
export OS_TENANT_NAME="TrainingIC"
export OS_PROJECT_NAME="TrainingIC"

# In addition to the owning entity (tenant), OpenStack stores the entity
# performing the action as the **user**.
export OS_USERNAME="UserTrainingIC"

# With Keystone you pass the keystone password.
echo "Please enter your OpenStack Password: "
read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT

# If your configuration has multiple regions, we set that information here.
# OS_REGION_NAME is optional and only valid in certain environments.
export OS_REGION_NAME="fr"
# Don't leave a blank variable, unset it if it was empty
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
```

We get OS_AUTH_URL, OS_TENANT_NAME, OS_USERNAME.

Authentication API version : OS_AUTH_URL

Beware of authentication API version. By default v3 is activated, but ansible needs the v2.0. We get the url and set V2.0 instead of V3 : https://openstack-identity.mycompany.com/v2.0

VM informations

Create an instance using the OpenStack web interface and get the name for image, flavor, key, network, security group.

Create a ./group_vars/all file with all the needed informations.

```
$vi ./group_vars/all
# Authentication
AuthUserName: UserTrainingIC
AuthPassword: PasswordTrainingIC
TenantName: TrainingIC

# VM infos
ImageName: CentOS-7-x86_64-GenericCloud-1607
FlavorName: m1.1cpu.1gb
InfraKey: KeyTrainingIC
NetworkName: NetPrivateTrainingIC
SecurityGroup: default
```

Write the ansible playbook to create the instance

Let use 'os_server' command from module 'Cloud' [http://docs.ansible.com/ansible/os_server_module.html]. Variables are defined in ./group_vars/all.

```
$vi launch_compute.yml
- name: launch a compute instance
  hosts: localhost
  gather_facts: False
  tasks:
  - name: Create and launch the VM
    os_server:
      auth:
        auth_url: https://openstack-identity.mycompany.com/v2.0
        username: "{{ AuthUserName }}"
        password: "{{ AuthPassword }}"
        project_name: "{{ TenantName }}"
      state: present
      validate_certs: False
      name: "MyOwnPersonalInstance"
      image: "{{ ImageName }}"
      key_name: "{{ InfraKey }}"
      timeout: 200
      flavor:   "{{ FlavorName }}"
      security_groups: "{{ SecurityGroup }}"
      network: "{{ NetworkName }}"
      auto_ip: yes
```

```
$ ansible-playbook  -s launch_compute.yml
[WARNING]: provided hosts list is empty, only localhost is available
PLAY [launch a compute instance] ***********************************************
TASK [Create and launch the VM] ************************************************
changed: [localhost]
PLAY RECAP *********************************************************************
localhost                  : ok=1    changed=1    unreachable=0    failed=0
```

Gather informations about our new instance

Use the 'os_server_facts' command from module 'Cloud' [http://docs.ansible.com/ansible/os_server_module.html]. Variables are defined in ./group_vars/all and the instance name is in server: "MyOwnPersonalInstance".

```
$vi get_compute_info.yml
- name: Get and print instance IP
  hosts: localhost
  gather_facts: False
  tasks:
  - name: Get VM infos
    os_server_facts:
      auth:
        auth_url: https://openstack-identity.mygroup/v2.0
        username: "{{ AuthUserName }}"
        password: "{{ AuthPassword }}"
        project_name: "{{ TenantName }}"
      validate_certs: False
      server: "MyOwnPersonalInstance"

  - name: Dump all
    debug:
      var: openstack_servers
```

```
$ansible-playbook  -s get_compute_info.yml
```

```
[WARNING]: provided hosts list is empty, only localhost is available
PLAY [Get and print instance IP] ***********************************************
TASK [Get VM IP] ***************************************************************
ok: [localhost]
TASK [Affichage] ***************************************************************
ok: [localhost] => {
    "openstack_servers": [
        {
            "OS-DCF:diskConfig": "MANUAL",
            "OS-EXT-AZ:availability_zone": "fr",
            "OS-EXT-STS:power_state": 1,
            "OS-EXT-STS:task_state": null,
[...]
            "volumes": []
        }
    ]
}

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0
```

This is very verbose. Lots of information is displayed. Usually only the IP address is needed to access the new instance via SSH.

Get your new instance public IP

Instead of printing all the informations, we print only IP address of the first instance whose name is "MyOwnPersonalInstance". It's usually all we need.

```
$vi get_compute_ip.yml
- name: Get and print instance IP
  hosts: localhost
  gather_facts: False
  tasks:
  - name: Get VM infos
    os_server_facts:
      auth:
        auth_url: https://openstack-identity.mycompany.com/v2.0
        username: "{{ AuthUserName }}"
        password: "{{ AuthPassword }}"
        project_name: "{{ TenantName }}"
      validate_certs: False
      server: "MyOwnPersonalInstance"

  - name: Dump IP
    debug:
      var: openstack_servers[0].interface_ip
```

Delete our instance

To delete our instance, reuse the os_server command with all authentication information and simply replace ' state: present' by ' state: absent'.

```
$vi stop_compute.yml
- name: launch a compute instance
```

```
hosts: localhost
gather_facts: False
tasks:
- name: Create and launch the VM
  os_server:
    auth:
      auth_url: https://openstack-identity.mygroup/v2.0
      username: "{{ AuthUserName }}"
      password: "{{ AuthPassword }}"
      project_name: "{{ ProjectName }}"
    state: absent
    validate_certs: False
    name: "{{ TPUser }}"
    timeout: 200
```

Read Using Ansible with OpenStack online: https://riptutorial.com/ansible/topic/8712/using-ansible-with-openstack

Credits

S. No	Chapters	Contributors
1	Getting started with ansible	activatedgeek, Alex, baptistemm, calvinmclean, Community, Jake Amey, jasonz, jscott, Michael Duffy, mrtuovinen, Pants, PumpkinSeed, tedder42, thisguy123, ydaetskcoR
2	Ansible Architecture	Jordan Anderson, Yogesh Darji
3	Ansible group variables	mrtuovinen
4	Ansible Group Vars	Nick, Peter Mortensen
5	Ansible install mysql	Fernando
6	Ansible: Looping	calvinmclean
7	Ansible: Loops and Conditionals	A K, Chu-Siang Lai, Jordan Anderson, marx, Mike, mrtuovinen, Nick, Rob H, wolfaviators
8	Become (Privilege Escalation)	Jordan Anderson, Willian Paixao
9	Dynamic inventory	mrtuovinen
10	Galaxy	mrtuovinen, ydaetskcoR
11	How To Create A DreamHost Cloud Server From An Ansible Playbook	Stefano Maffulli
12	Installation	ca2longoria, Jake Amey, Michael Duffy, mrtuovinen, Nick, PumpkinSeed, Raj, tedder42, ydaetskcoR
13	Introduction to playbooks	32cupo, Abdelaziz Dabebi, ydaetskcoR
14	Inventory	calvinmclean, mrtuovinen, Nick
15	Loops	marx, mrtuovinen
16	Roles	Chu-Siang Lai, fishi, mrtuovinen, winston, ydaetskcoR
17	Secret encryption	fishi

18	Using Ansible with Amazon Web Services	Abdelaziz Dabebi, another geek, ydaetskcoR
19	Using Ansible with OpenStack	BANANENMANNFRAU, Sebastien Josset