eBook Gratuit

APPRENEZ ansible

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#ansible

Table des matières

À propos

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

Chapitre 1: Démarrer avec ansible

Remarques

Cette section fournit une vue d'ensemble de ce qu'est ansible et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans ansible, et établir un lien avec les sujets connexes. La documentation de ansible étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Bonjour le monde

Créez un répertoire appelé `ansible-helloworld-playbook`

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

Créez des `hosts` fichiers et ajoutez des systèmes distants à la manière dont vous souhaitez les gérer. Comme Ansible s'appuie sur ssh pour connecter les machines, vous devez vous assurer qu'elles vous sont déjà accessibles depuis SSH depuis votre ordinateur.

```
192.168.1.1
192.168.1.2
```

Testez la connexion à vos systèmes distants à l'aide du module ping Ansible.

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

En cas de succès, il devrait retourner quelque chose comme ça

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

En cas d'erreur, il devrait retourner

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

```
}
```

Testez l'accès à sudo avec

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

Tester la connexion et la configuration avec ping

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

`-i hosts` définit le chemin d'accès au fichier d'inventaire
`targethost` est le nom de l'hôte dans le fichier `hosts`

Inventaire

Inventory est le moyen Ansible de suivre tous les systèmes de votre infrastructure. Voici un fichier d'inventaire statique simple contenant un seul système et les identifiants de connexion pour Ansible.

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

Écrivez ces lignes par exemple dans le fichier `hosts` et transmettez le fichier à la commande `ansible` ou `ansible-playbook` avec l' `--inventory-file -i` / `--inventory-file` .

Voir l'inventaire statique et l'inventaire dynamique pour plus de détails.

Mise à disposition de machines distantes avec Ansible

Nous pouvons fournir des systèmes distants avec Ansible. Vous devriez avoir une paire de clés SSH et vous devez apporter votre clé publique SSH au fichier machine ~ / .ssh / authorized_keys. Le porpuse est que vous pouvez vous connecter sans aucune autorisation.

Conditions préalables:

- Ansible

Vous avez besoin d'un fichier d'inventaire (par exemple: development.ini) où vous déterminez l'hôte que vous voulez utiliser:

```
[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 - le nom d'hôte de la machine distante
- ansible_ssh_host - l'ip ou le domaine de l'hôte distant
- ansible_port - le port de l'hôte distant qui est généralement 22

- ansible_connection - la connexion où nous définissons, nous voulons nous connecter avec ssh
- ansible_user - l'utilisateur ssh
- ansible_ssh_extra_args - arguments supplémentaires de ce que vous voulez spécifier pour la connexion ssh

Arguments supplémentaires requis pour ssh:

- StrictHostKeyChecking - Il peut demander à une clé de vérifier ce qui attend un oui ou un non. Ansible ne peut pas répondre à cette question, puis lance une erreur, l'hôte n'est pas disponible.
- UserKnownHostsFile - Nécessaire pour l'option StrictHostKeyChecking.

Si vous avez ce fichier d'inventaire, vous pouvez écrire un test playbook.yml:

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

alors vous pouvez commencer la provision:

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

ansible.cfg

C'est le fichier par défaut ansible.cfg d' 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
```

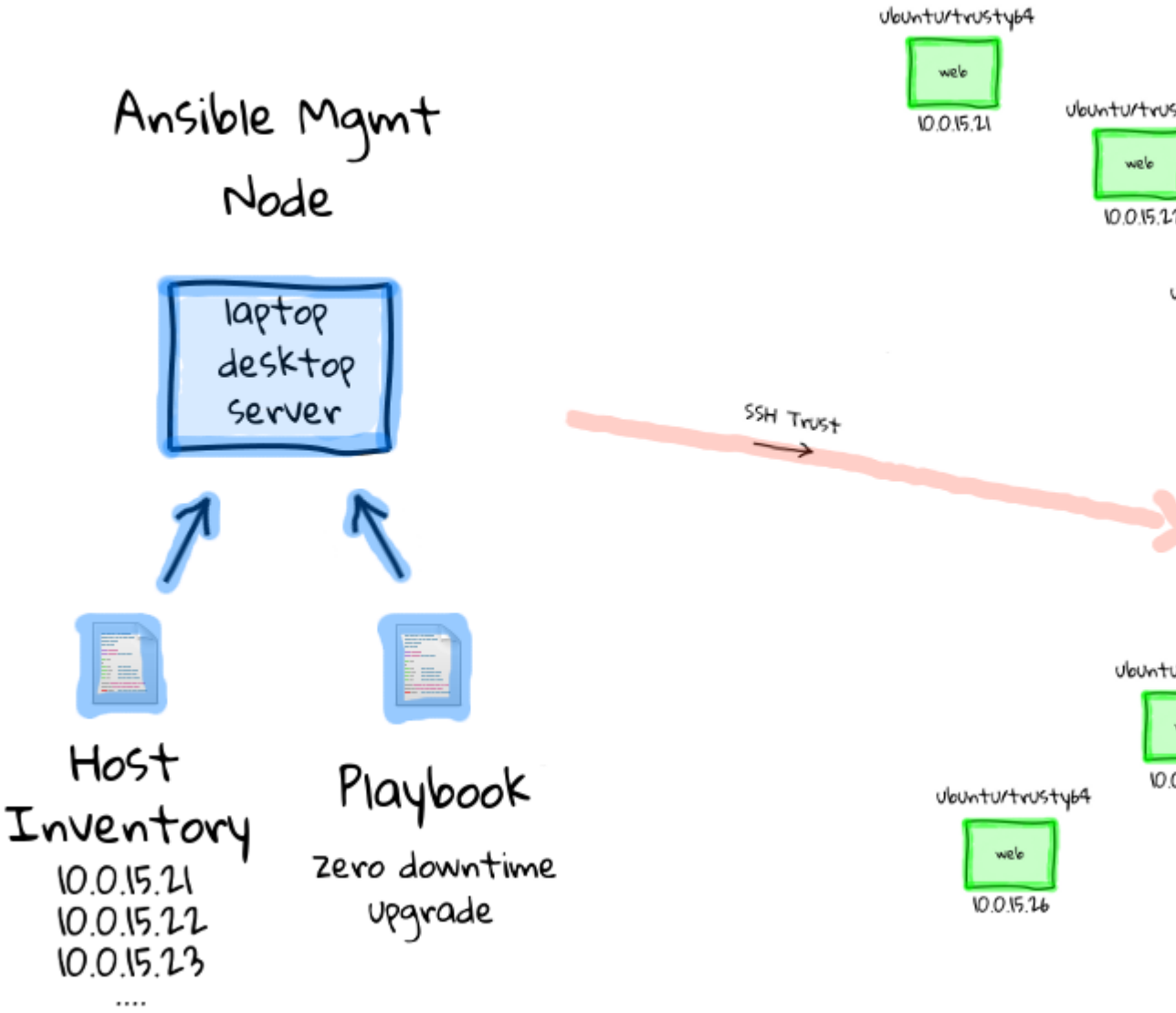
Placez cette configuration dans la racine de vos répertoires de rôles pour modifier le comportement d'Ansible lors de l'utilisation de ce rôle. Par exemple, vous pouvez le configurer pour arrêter la création de `playbook.retry` sur les exécutions échouées du playbook ou pour pointer vers des vars secrets que vous ne voulez pas inclure dans votre dépôt git.

Lire Démarrer avec ansible en ligne: https://riptutorial.com/fr/ansible/topic/826/demarrer-avec-ansible

Chapitre 2: Ansible Architecture

Examples

Comprendre l'architecture Ansible



L'idée est d'avoir une ou plusieurs machines de contrôle à partir desquelles vous pouvez émettre des commandes ad hoc vers des machines distantes (via l'outil `ansible`) ou exécuter un jeu d'instructions séquentielles via des playbooks (via l'outil `ansible-playbook`).

Fondamentalement, nous utilisons la machine de contrôle Ansible, qui sera généralement votre ordinateur de bureau, votre ordinateur portable ou votre serveur. Ensuite, à partir de là, vous utilisez Ansible pour sortir les modifications de configuration, via ssh.

Le fichier d'inventaire de l'hôte détermine les machines cibles sur lesquelles ces lectures seront exécutées. Le fichier de configuration Ansible peut être personnalisé pour refléter les paramètres

de votre environnement.

Lire Ansible Architecture en ligne: https://riptutorial.com/fr/ansible/topic/7659/ansible-architecture

Chapitre 3: Ansible Group Vars

Examples

Exemple group_vars / development et pourquoi

Structure du projet

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

Ces variables seront appliquées aux hôtes du groupe de développement en raison du nom de fichier.

```
---
## 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
```

Lire Ansible Group Vars en ligne: https://riptutorial.com/fr/ansible/topic/6226/ansible-group-vars

Chapitre 4: Ansible install mysql

Introduction

Comment utiliser ansible pour installer le fichier binaire mysql

Examples

Comment utiliser ansible pour installer le fichier binaire mysql

---

- hosts: tâches mysql:
    - name: Ajouter un utilisateur mysql user: name: shell mysql: / sbin / nologin
    - name: installe la dernière version de libselinux-python yum: nom: libselinux-python state: latest
    - nom: install perl yum: nom: perl état: dernier
    - name: supprime le paquet mysql-libs yum: nom: 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
```

Lire Ansible install mysql en ligne: https://riptutorial.com/fr/ansible/topic/10920/ansible-install-mysql

Chapitre 5: Ansible: Boucles et conditions

Remarques

Les documents officiels expliquent les conditions de jeu.

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

Ansible (github)

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

Examples

Quels types de conditionnels à utiliser?

Utilisez Conditionals via (la syntaxe est `[brackets]`):

- quand [**quand:**]

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

- boucles [**with_items:**]
- boucles [**with_dicts:**]
- Facts personnalisés [**lorsque:** my_custom_facts == '1234']
- Importations conditionnelles
- Sélectionner des fichiers et des modèles en fonction de variables

[When] Condition: Listes 'ansible_os_family`

Usage commun

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

Toutes les listes

sur la base de discuter ici 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'
        )
```

Quand condition

Utilisation de base

Utilisez la condition when pour contrôler si une tâche ou un rôle s'exécute ou est ignoré. Ceci est normalement utilisé pour modifier le comportement de jeu en fonction des faits du système de destination. Considérez ce playbook:

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

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

Où `Ubuntu.yml` et `RHEL.yml` incluent une logique spécifique à la distribution.

Une autre utilisation courante consiste à limiter les résultats à ceux de certains groupes d'inventaire Ansible. Considérez ce fichier d'inventaire:

```
[dbs]
mydb01

[webservers]
myweb01
```

Et ce playbook:

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

Cela utilise la variable magique `group_names` .

Syntaxe conditionnelle et logique

Seul état

Syntaxe

```
when: (condition)
```

Exemple

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

Filtre booléen

Exemple

```
when: result|failed
```

Conditions multiples

Syntaxe

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

Exemple (simple)

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

Exemple (complexe)

Utilisez des parenthèses pour plus de clarté ou pour contrôler la priorité. "ET" a une priorité plus élevée que "OU".

Les clauses peuvent couvrir des lignes:

```
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', '>=')
```

Notez l'utilisation des parenthèses pour regrouper le "ou" dans la première vérification de distribution.

Obtenez `ansible_os_family` et` ansible_pkg_mgr` avec la configuration

Nous pouvons obtenir des faits (`ansible_os_family` , `ansible_pkg_mgr`) avec la commande Ad-Hoc du module de configuration et du filtre.

- 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" Exemple (s)

Donné:

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

Ensuite, ces tâches avec toujours exécuter.

```
- 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
```

Cette tâche ne fonctionnera jamais.

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

Utilisation de jusqu'à une tentative de vérification en boucle

Ceci est un exemple d'utilisation jusqu'à / retries / delay pour implémenter une vérification active pour une application Web en cours de démarrage. Cela suppose qu'il y aura une période de temps (jusqu'à 3 minutes) où l'application Web refusera les connexions socket. Après cela, il vérifie la page / alive pour le mot "OK". Il délègue également la récupération de l'URL au localhost exécutant ansible. Cela est logique en tant que tâche finale dans un playbook de déploiement.

```
---
- 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
```

Le modèle de relance peut être utilisé avec n'importe quelle action; La documentation d'Ansible fournit un exemple de l'attente qu'une certaine commande shell renvoie un résultat souhaité: http://docs.ansible.com/ansible/playbooks_loops.html#do-until-loops .

Lire Ansible: Boucles et conditions en ligne: https://riptutorial.com/fr/ansible/topic/3555/ansible--boucles-et-conditions

Chapitre 6: Ansible: Looping

Examples

with_items - liste simple

Une boucle `with_items` dans ansible peut être utilisée pour boucler facilement les valeurs.

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

with_items - liste prédéfinie

Vous pouvez également parcourir une liste de variables.

De vars:

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

et puis la boucle:

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

Si vous utilisez Ansible 2.0+, vous devez utiliser des guillemets autour de l'appel à la variable.

with_items - dictionnaire prédéfini

Il est possible de créer des boucles plus complexes avec des dictionnaires.

De vars:

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

alors la boucle:

```
- name: manage packages
```

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

Ou, si vous n'aimez pas utiliser la valeur clé:

vars:

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

alors la boucle:

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

with_items - dictionnaire

Vous pouvez utiliser un dictionnaire pour une boucle légèrement plus complexe.

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

Boucles imbriquées

Vous pouvez créer des boucles imbriquées à l'aide de `with_nested` .

de vars:

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

alors la boucle:

```
- 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 }}'
```

Cette tâche fera une boucle sur chaque utilisateur et remplira son fichier `authorized_keys` avec les 4 clés définies dans la liste.

Lire Ansible: Looping en ligne: https://riptutorial.com/fr/ansible/topic/6414/ansible--looping

Chapitre 7: Boucles

Examples

Copier plusieurs fichiers en une seule tâche

```
- 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
```

Installer plusieurs packages en une seule tâche

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

Lire Boucles en ligne: https://riptutorial.com/fr/ansible/topic/6095/boucles

Chapitre 8: Comment créer un serveur Cloud DreamHost à partir d'un Playbook Ansible

Examples

Installer la bibliothèque d'ombre

Shade est une bibliothèque développée par OpenStack pour simplifier les interactions avec les clouds OpenStack, comme DreamHost.

$ installer la teinte

Ecrire une Playbook pour lancer un serveur

Créez un fichier nommé `launch-server.yaml` , qui sera notre playbook.

La première partie du playbook est une liste des hôtes sur lesquels votre playbook sera exécuté, nous n'en avons qu'une, localhost.

```
- hosts: localhost
```

Ensuite, nous devons définir une liste de tâches à effectuer dans ce playbook. Nous en aurons seulement un qui lancera un serveur Ubuntu Xenial sur DreamCompute.

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

La partie suivante du playbook utilise le `os_server` (OpenStack Server). Cela définit l'apparence du serveur dans DreamCompute.

```
os_server:
```

La première étape consiste à s'authentifier sur DreamCompute; remplacez `{username}` par votre nom d'utilisateur DreamCompute, `{password}` par votre mot de passe DreamCompute et par `{project}` par votre projet DreamCompute. Vous les trouverez dans le fichier OpenStack RC .

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

Les lignes suivantes définissent certains éléments du nouveau serveur.

```
  state: present
```

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

Permet de décomposer les quelques lignes précédentes:

- `state` est l'état du serveur, les valeurs possibles sont `present` ou `absent`
- `name` est le nom du serveur à créer; peut être n'importe quelle valeur
- `image` est l'image depuis laquelle démarrer le serveur; les valeurs possibles sont visibles sur le panneau Web de DreamHost Cloud ; la variable accepte soit le nom de l'image, soit l'UUID
- `key_name` est le nom de la clé publique à ajouter au serveur une fois créé; Cela peut être n'importe quelle clé a déjà été ajoutée à DreamCompute.
- `flavor` est la saveur du serveur à démarrer; Ceci définit la quantité de mémoire vive et de processeur que votre serveur aura; la variable accepte soit le nom d'une saveur (gp1.semisonic) ou l'ID (50, 100, 200, etc.)
- `network` est le réseau sur lequel mettre votre serveur. Dans DreamHost Cloud, c'est le réseau `public` .
- `wait` set to yes force le bookbook à attendre la création du serveur avant de continuer.

Lancer le Playbook

Exécutez le playbook Ansible:

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

Vous devriez voir la sortie comme

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

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

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

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

Maintenant, si vous vérifiez le tableau de bord de DreamHost Cloud, vous devriez voir une nouvelle instance nommée «ansible-vm1»

Lire Comment créer un serveur Cloud DreamHost à partir d'un Playbook Ansible en ligne: https://riptutorial.com/fr/ansible/topic/4689/comment-creer-un-serveur-cloud-dreamhost-a-partir-d-

un-playbook-ansible

Chapitre 9: Cryptage secret

Remarques

Ansible propose Vault (à ne pas confondre avec HashiCorp Vault !) Pour gérer le cryptage des données sensibles. Vault vise principalement à chiffrer toutes les données structurées telles que les variables, les tâches et les gestionnaires.

Examples

Cryptage de données structurées sensibles

Tout d'abord, créez un fichier de clés, par exemple `vault_pass_file` , qui contient idéalement une longue séquence de caractères aléatoires. Dans les systèmes Linux, vous pouvez utiliser `pwgen` pour créer un fichier de mot de passe aléatoire:

```
pwgen 256 1 > vault_pass_file
```

Ensuite, utilisez ce fichier pour chiffrer les données sensibles, par exemple, `groups_vars/group.yml` :

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

A partir de maintenant, pour lancer un playbook, vous avez besoin du `vault_pass_file` :

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

Notez que vous pouvez également utiliser l' `--vault-password-file vault_pass_file` au lieu de définir la variable d'environnement `ANSIBLE_VAULT_PASSWORD_FILE` .

Pour éditer ou déchiffrer le secret sur le disque, vous pouvez utiliser respectivement `ansible-vault edit` et `ansible-vault decrypt` respectivement.

Utilisation de canaux de recherche pour déchiffrer des données chiffrées non structurées

Avec Vault, vous pouvez également crypter des données non structurées, telles que des fichiers de clés privées, tout en étant en mesure de les décrypter dans votre jeu avec le module de `lookup` .

```
---

- 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')
```

Utilisation de local_action pour déchiffrer les modèles chiffrés dans un coffre

Vous pouvez exécuter un jeu qui repose sur des modèles chiffrés à l'aide d'un `local_action` en utilisant le module `local_action` .

```
---

- 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
```

S'il vous plaît noter le `changed_when: False` . Ceci est important dans le cas où vous exécutez des tests d'idempotence avec vos rôles de réponse - sinon, chaque fois que vous exécutez le livre de jeu, un changement est signalé. Dans `group_vars/all.yml` vous pouvez définir une commande globale decrypt pour la réutiliser, par exemple, `view_encrypted_file_cmd` .

group_vars / all.yml

```
---

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

Maintenant, lorsque vous exécutez une lecture, vous devez définir la variable d'environnement `ANSIBLE_VAULT_PASSWORD_FILE` pour qu'elle pointe vers votre fichier de mot de passe de coffre-fort (idéalement avec un chemin absolu).

Lire Cryptage secret en ligne: https://riptutorial.com/fr/ansible/topic/3355/cryptage-secret

Chapitre 10: Devenir (Escalade de privilèges)

Introduction

Souvent, vous devez exécuter des commandes sous un utilisateur différent ou obtenir *des* privilèges *root* . Ces options vous permettent de **devenir** un autre utilisateur du système invité.

Syntaxe

- `become` : peut être défini sur true ou yes et déclenche les paramètres d'escalade de l'utilisateur.
- `become_user` : défini sur l'utilisateur souhaité dans l'hôte distant.
- `become_method` : spécifie la commande utilisée pour établir la connexion et modifier l'utilisateur.
- `become_flags` : modifie les paramètres de connexion. Principalement utilisé lorsque vous souhaitez passer à un utilisateur du système sans privilèges de shell.

Examples

Seulement dans une tâche

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

Exécuter toutes les tâches de rôle en tant que root

```
- hosts: all
  become: true

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

Exécuter un rôle en tant que root

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

Lire Devenir (Escalade de privilèges) en ligne: https://riptutorial.com/fr/ansible/topic/8328/devenir--escalade-de-privileges-

Chapitre 11: Galaxie

Examples

Partager des rôles avec Ansible Galaxy

Il est également possible de partager facilement des rôles avec la communauté ou de télécharger des rôles créés par d'autres membres de la communauté avec Ansible Galaxy .

Ansible est livré avec un outil de ligne de commande appelé `ansible-galaxy` qui peut être utilisé pour installer des rôles dans le répertoire de rôles défini dans le fichier `ansible.cfg` :

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

Vous pouvez également utiliser l'outil Ansible Galaxy pour télécharger des rôles depuis d'autres emplacements tels que GitHub en créant un fichier texte avec l'emplacement défini comme `src` :

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

Et puis installez les rôles dans le fichier texte comme suit:

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

Vous pouvez également utiliser l'outil `ansible-galaxy` pour créer le rôle "échafaudage":

```
ansible-galaxy init rolename
```

Une fois que vous avez créé un rôle et l'a téléchargé sur GitHub, vous pouvez le partager sur Ansible Galaxy en vous connectant à votre repo GitHub dans Ansible Galaxy après la connexion.

Plus d'exemples sous le sujet Galaxy .

Lire Galaxie en ligne: https://riptutorial.com/fr/ansible/topic/6599/galaxie

Chapitre 12: Galaxie

Examples

Commandes de base

Rôle de recherche dans Ansible Galaxy

```
ansible-galaxy search role_name
```

Installez le rôle de Ansible Galaxy

```
ansible-galaxy install role_name
```

Plus d'aide

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

Lire Galaxie en ligne: https://riptutorial.com/fr/ansible/topic/6656/galaxie

Chapitre 13: Installation

Introduction

Installation d'Ansible dans n'importe quel système d'exploitation, y compris Windows avec Virtual Box et Vagrant. Une autre solution est également disponible si vous souhaitez simplement mettre en pratique des commandes et des playbooks ad hoc et ne souhaitez pas configurer l'environnement local.

Examples

Installer Ansible sur Ubuntu

Ansible gère un référentiel PPA pouvant être utilisé pour installer les binaires Ansible:

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

Pour installer une version spécifique, utilisez `pip` . Le PPA peut être obsolète.

Installation d'Ansible sur MacOS

Il y a deux manières principales d'installer Ansible sur OS X, en utilisant le gestionnaire de paquets Homebrew ou Pip.

Si vous avez un homebrew, la dernière Ansible peut être installée en utilisant la commande suivante:

```
brew install ansible
```

Pour installer Ansible 1.9.X branche, utilisez la commande suivante:

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

Pour installer Ansible 2.0.X branche, utilisez la commande suivante:

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

Pour installer en utilisant pip, utilisez la commande suivante: `pip install ansible` .

Pour installer une version spécifique, utilisez `pip install ansible=<required version>` .

Installation sur des systèmes basés sur Red Hat

Ansible peut être installé sur CentOS ou d'autres systèmes basés sur Red Hat. Tout d'abord, vous

devez installer les prérequis:

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

puis installez Ansible avec pip:

```
sudo pip install ansible
```

Je peux vous recommander de mettre à jour les setuptools après l'installation:

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

Vous pouvez également utiliser le gestionnaire de packages local:

```
yum install ansible
```

Installation à partir de la source

Ansible est **mieux utilisé à** partir d'une caisse.

Il fonctionne comme vous (pas root) et il a des dépendances python minimales.

Installation de la dépendance Python pip avec pip:

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

Ensuite, clonez le repo Ansible de GitHub:

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

Enfin, ajoutez la ligne de script d'initialisation ansible à votre ~ / .bashrc ou ~ / .zshrc:

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

Redémarrez votre session de terminal et testez avec

```
ansible --version
```

Installation sur Amazon Linux depuis git repo

Amazon Linux est une variante de RHEL, de sorte que les instructions Red Hat devraient pour la plupart fonctionner. Il y a cependant au moins une divergence.

Il y avait une instance où le **paquet python27-devel** , par opposition à **python-devel** , était

explicitement nécessaire.

Ici, nous allons installer à partir des sources.

```
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
```

Installation d'Ansible On sur n'importe quel système d'exploitation (Windows) à l'aide de Virtual Box + Vagrant

Mon ordinateur portable est équipé de Windows 10. Ici, je vous donne les étapes à suivre pour tester et apprendre Ansible.

QUELQUES THÉORIES

Pour Ansible, vous avez besoin d'une machine de contrôle et d'un hôte (ou hôtes) pour exécuter le Playbook.

- **Control Machine** doit être basé sur Linux ou MacOS (Windows n'est pas autorisé) et nécessite Python (version 2.6 ou ultérieure). Ici, Ansible sera installé.
- **La machine cible** (hôte / noeud) peut être Linux / MacOS / windows. Cela nécessite seulement que Python soit installé. Aucun logiciel d'agent requis.

INSTALLER

Étape 1: **Installez Virtual Box**

La boîte virtuelle est un logiciel permettant de créer des ordinateurs virtuels de systèmes d'exploitation différents. C'est comme avoir plusieurs ordinateurs chacun ou différents OS et différentes versions.

Téléchargez Virtual Box selon le système d'exploitation de votre système et installez-le.

Étape 2: **Installez Vagrant**

Vagrant est une interface de ligne de commande pour créer des machines virtuelles dans une boîte virtuelle. Cela facilite les choses. Vous devez apprendre les commandes de base Vagrant.

Étape 3: **Créez un dossier dans lequel vous voulez que votre machine virtuelle**

Étape 4: **Créer une machine virtuelle en utilisant Vagrant**

Ouvrez le terminal et accédez au chemin où vous avez créé le dossier et exécutez les deux commandes suivantes.

Vous devez sélectionner **Virtual Box** . J'installe Ubuntu par exemple. Vous pouvez choisir n'importe quoi dans la liste. Vous devez exécuter ces deux commandes sous **la** catégorie " **boîte virtuelle** ": `vagrant init ubuntu/trusty64` et `vagrant up --provider virtualbox` . Les autres catégories peuvent être: hyperv, vmware_desktop, etc. (cela prendra du temps, car cela téléchargera les fichiers nécessaires)

Étape 4: Installer Ansible

Pour UbuntuOS: `sudo apt-get install ansible`

Solution alternative :

Vous pouvez utiliser **Katacoda** pour pratiquer l'ansible. Pas besoin d'installer ou de configurer quoi que ce soit. Exécutez deux commandes données à l'étape 2 et après cela, vous êtes prêt à partir.

Lire Installation en ligne: https://riptutorial.com/fr/ansible/topic/4906/installation

Chapitre 14: Introduction aux playbooks

Examples

Vue d'ensemble

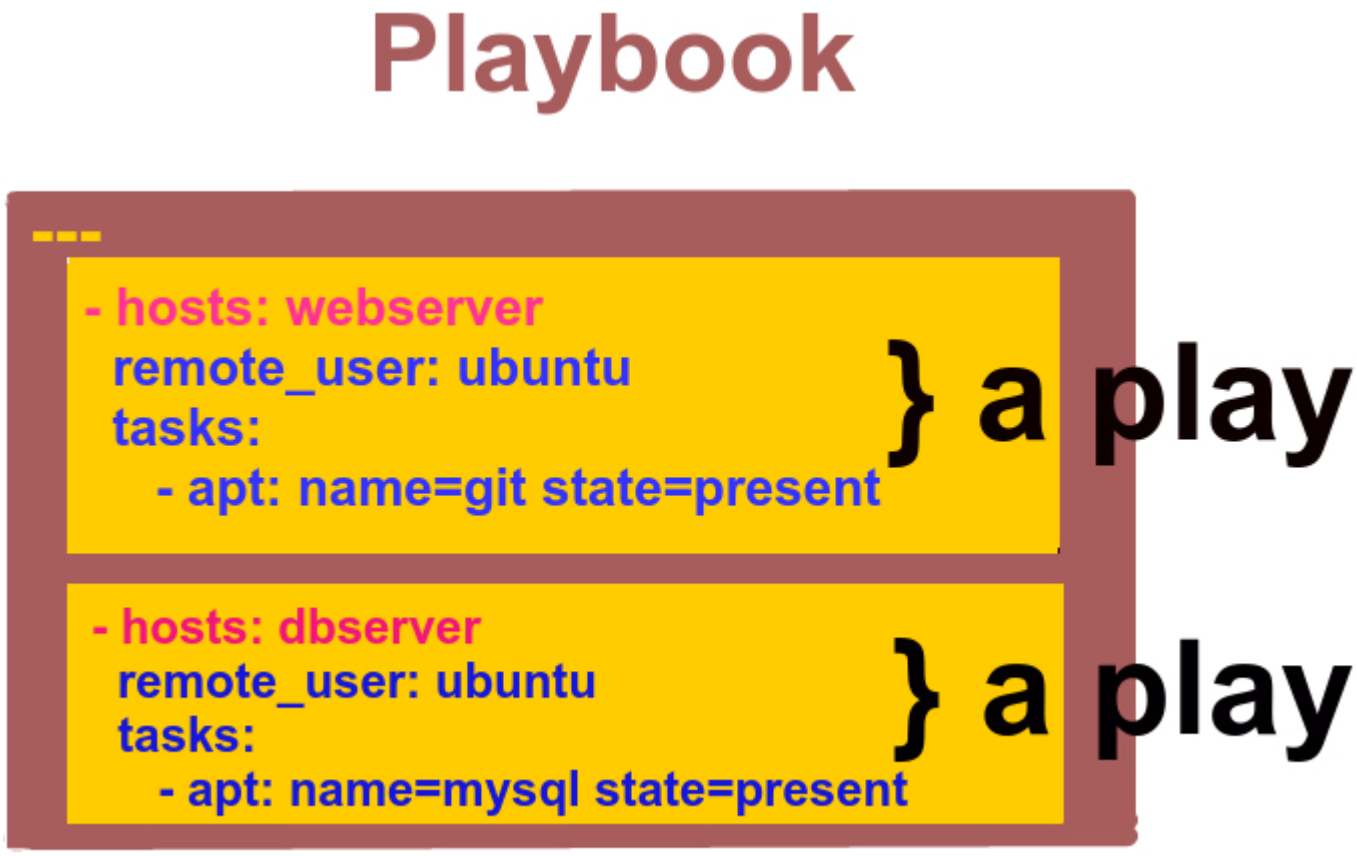
Dans Ansible, un playbook est un fichier YAML contenant la définition de l'apparence d'un serveur. Dans un playbook, vous définissez les actions que Ansible doit entreprendre pour obtenir le serveur dans l'état souhaité. Seul ce que vous définissez se fait.

Ceci est un playbook Ansible de base qui installe git sur chaque hôte appartenant au groupe `web` :

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

Structure du Playbook

Le format d'un playbook est assez simple, mais strict en termes d'espacement et de disposition. Un livre de jeu se compose de jeux. Un jeu est une combinaison d'hôtes cibles et des tâches que nous souhaitons appliquer à ces hôtes. Le dessin d'un playbook est le suivant:



Pour exécuter ce playbook, nous exécutons simplement:

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

Structure de jeu

Voici un jeu simple:

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

Comme nous l'avons dit plus tôt, chaque pièce doit contenir:

- Un ensemble d'hôtes à configurer
- Une liste des tâches à exécuter sur ces hôtes

Considérez un jeu comme la chose qui connecte les hôtes aux tâches. Outre la spécification des hôtes et des tâches, les jeux prennent également en charge un certain nombre de paramètres facultatifs. Les deux plus courants sont:

- `name` : un commentaire décrivant le sujet de la pièce. Ansible imprimera ceci lorsque le jeu commence à courir
- `vars` : une liste de variables et de valeurs

Mots clés

Play contient plusieurs tâches, qui peuvent être balisées:

```
- 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
```

La tâche avec la balise 'vim' sera exécutée lorsque 'vim' est spécifié dans les balises. Vous pouvez spécifier autant de balises que vous le souhaitez. Il est utile d'utiliser des tags tels que "install" ou "config". Ensuite, vous pouvez exécuter playbook en spécifiant des balises ou des balises. Pour

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

Par défaut, Ansible exécute toutes les balises

Lire Introduction aux playbooks en ligne: https://riptutorial.com/fr/ansible/topic/3343/introduction-aux-playbooks

Chapitre 15: Inventaire

Paramètres

Paramètre	Explication
ansible_connection	Type de connexion à l'hôte. Cela peut être le nom de l'un des plugins de connexion d'ansible. Les types de protocole SSH sont `smart` , `ssh` ou `paramiko` . La valeur par défaut est intelligente. Les types non basés sur SSH sont décrits dans la section suivante.
ansible_host	Le nom de l'hôte auquel se connecter, si différent de l'alias que vous souhaitez lui donner.
ansible_port	Le numéro de port ssh, sinon 22
ansible_user	Le nom d'utilisateur ssh par défaut à utiliser.
ansible_ssh_pass	Le mot de passe ssh à utiliser (ceci n'est pas sûr, nous vous recommandons fortement d'utiliser les `--ask-pass` ou SSH)
ansible_ssh_private_key_file	Fichier de clé privée utilisé par ssh. Utile si vous utilisez plusieurs clés et que vous ne voulez pas utiliser l'agent SSH.
ansible_ssh_common_args	Ce paramètre est toujours ajouté à la ligne de commande par défaut pour **sftp** , **scp** et **ssh** . Utile pour configurer un `ProxyCommand` pour un certain hôte (ou groupe).
ansible_sftp_extra_args	Ce paramètre est toujours ajouté à la ligne de commande **sftp** par défaut.
ansible_scp_extra_args	Ce paramètre est toujours ajouté à la ligne de commande **scp** par défaut.
ansible_ssh_extra_args	Ce paramètre est toujours ajouté à la ligne de commande **ssh** par défaut.
ansible_ssh_pipelining	Détermine s'il faut ou non utiliser le traitement en pipeline SSH. Cela peut remplacer le paramètre de `pipelining` dans `ansible.cfg` .
ansible_become	Équivalent à `ansible_sudo` ou `ansible_su` , permet de forcer l'escalade de privilèges
ansible_become_method	Permet de définir une méthode d'élévation de privilèges

Paramètre	Explication
ansible_become_user	Équivalent à `ansible_sudo_user` ou `ansible_su_user` , permet de définir l'utilisateur que vous devenez grâce à l'escalade de privilèges
ansible_become_pass	Équivalent à `ansible_sudo_pass` ou `ansible_su_pass` , vous permet de définir le mot de passe d' `ansible_su_pass` privilèges
ansible_shell_type	Le type de shell du système cible. Vous ne devez pas utiliser ce paramètre, sauf si vous avez défini `ansible_shell_executable` sur un shell non compatible Bourne (sh). Par défaut, les commandes sont formatées à l'aide de la syntaxe `sh` -style. Si vous `csh` cette `csh` sur `csh` ou `fish` , les commandes exécutées sur les systèmes cibles suivront plutôt la syntaxe de ces shell.
ansible_python_interpreter	Le chemin python de l'hôte cible. Ceci est utile pour les systèmes avec plusieurs Python ou non situés dans **/ usr / bin / python** tels que * BSD, ou lorsque **/ usr / bin / python** n'est pas une série Python 2.X. Nous ne pas utiliser le **/ usr / bin / env** mécanisme qui nécessite que le chemin d' un utilisateur distant à définir et à droite suppose également l'exécutable **python** est nommé python, où l'exécutable peut être nommé quelque chose comme **python2.6.**
ansible _ * _ interprète	Fonctionne comme `ansible_python_interpreter` ou Perl et fonctionne comme `ansible_python_interpreter` . Cela remplace le shebang des modules qui fonctionneront sur cet hôte.
ansible_shell_executable	Cela définit le shell que le contrôleur ansible utilisera sur la machine cible, et remplace l' `executable` dans `ansible.cfg` qui est par défaut **/ bin / sh** . Vous ne devriez vraiment le changer que si vous ne pouvez pas utiliser **/ bin / sh** (c.-à **-** d **. Que / bin / sh** n'est pas installé sur la machine cible ou ne peut pas être exécuté depuis sudo). Nouveau dans la version 2.1.

Examples

Inventaire avec nom d'utilisateur et mot de passe

Inventory est le moyen Ansible de suivre tous les systèmes de votre infrastructure. Voici un fichier d'inventaire simple contenant un seul système et les identifiants de connexion pour Ansible.

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

Inventaire avec clé privée personnalisée

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

Inventaire avec port SSH personnalisé

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

Passer l'inventaire statique à ansible-playbook

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

Passer l'inventaire dynamique à ansible-playbook

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

Voir l'inventaire dynamique pour plus de détails.

Inventaire, groupe Vars et vous

structure du projet (ansible Best Practice).

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

tout commence avec 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
```

qui vous permet de créer un lien vers group_vars. Tenez les données «spécifiques» à cet environnement ...

```
---
app_name: NewApp_Dev
```

```
app_url: https://dev.fakename.io
app_key: f2390f23f01233f23f
```

cela permet d'exécuter le playbook suivant CONTRE le fichier d'inventaire:

```
---
- name: Install api.
  hosts: api
  gather_facts: true
  sudo: true
  tags:
    - api
  roles:
    - { role: api,         tags: ["api"]         }
```

avec la piste suivante:

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

Fichier Hosts

Le fichier hôte est utilisé pour stocker les connexions pour les playbooks Anisble. Il existe des options pour définir les paramètres de connexion:

`ansible_host` est le nom d'hôte ou l'adresse IP

`ansible_port` est le port que la machine utilise pour SSH

`ansible_user` est l'utilisateur distant qui se connecte en tant que

`ansible_ssh_pass` si vous utilisez un mot de passe pour SSH

`ansible_ssh_private_key_file` si vous devez utiliser plusieurs clés spécifiques aux hôtes

Ce sont les options les plus couramment utilisées. Vous en trouverez plus dans la documentation officielle d'Ansible .

Voici un exemple de fichier `hosts` :

```
# 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
```

Lire Inventaire en ligne: https://riptutorial.com/fr/ansible/topic/1764/inventaire

Chapitre 16: Inventaire dynamique

Remarques

Les variables d'environnement dans l'inventaire dynamique ne fonctionnent pas

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

Si le côté serveur d'inventaire dynamique transmet `$HOME` par exemple, remplacez la variable dans le code client (Python):

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

Examples

Inventaire dynamique avec identifiants de connexion

Passer l'inventaire dynamique à `ansible-playbook` :

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

`python inventory/dyn.py` devrait imprimer quelque chose comme ceci:

```
{
  "_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"
  ]
}
```

Lire Inventaire dynamique en ligne: https://riptutorial.com/fr/ansible/topic/1758/inventaire-dynamique

Chapitre 17: Les rôles

Examples

Utiliser des rôles

Ansible utilise le concept de rôles pour mieux autoriser le code modulaire et éviter de se répéter.

Un rôle est simplement une structure de dossier dans laquelle Ansible sait où charger les fichiers vars, les tâches et les gestionnaires. Un exemple pourrait ressembler à ceci:

```
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
```

Vous pouvez ensuite utiliser le rôle avec un livret de base qui ressemble à ceci:

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

Lorsque vous exécutez Ansible sur ce playbook, il cible tous les hôtes du groupe de `webservers` et exécute le rôle `apache` défini ci-dessus, en chargeant automatiquement les variables par défaut du rôle et en exécutant toutes les tâches incluses dans `tasks/main.yml` . Ansible sait également rechercher certains types de fichiers dans des emplacements adaptés aux rôles:

- Si roles / x / tasks / main.yml existe, les tâches qui y sont répertoriées seront ajoutées au jeu.
- Si roles / x / handlers / main.yml existe, les gestionnaires listés seront ajoutés à la lecture
- Si roles / x / vars / main.yml existe, les variables listées seront ajoutées à la lecture
- Si roles / x / meta / main.yml existe, toutes les dépendances de rôles qui y sont répertoriées seront ajoutées à la liste des rôles (version 1.3 et ultérieure).

- Toute tâche de copie, de script, de modèle ou d'inclusion (dans le rôle) peut référencer des fichiers dans des rôles / x / {fichiers, modèles, tâches} / (dir dépend de la tâche) sans avoir à les tracer relativement ou absolument

Les dépendances de rôle

Les rôles vous permettent également de définir d'autres rôles en tant que dépendance en créant un fichier `meta/main.yml` avec un bloc de `dependencies` :

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

Il est également possible de passer une valeur à un paramètre / variable dans le rôle dépendant:

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

Ou même exécuter le rôle dépendant conditionnellement:

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

Les rôles dépendants sont toujours exécutés avant les rôles qui en dépendent. En outre, ils ne sont exécutés qu'une seule fois. Si deux rôles sont identiques à ceux de leur dépendance, ils ne sont exécutés que la première fois.

Imaginez les rôles role1, role2 et role3 avec les `meta/main.yml` :
role1 / meta / main.yml:

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

role2 / meta / main.yml:

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

Lors de l'exécution du rôle1 et du rôle2 dans le même répertoire (avec le rôle1 appelé avant le rôle2), l'ordre d'exécution serait le suivant:

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

Vous pouvez remplacer ce comportement en spécifiant `allow_duplicates: yes` dans `meta/main.yml` de role1 et role2. L'ordre d'exécution résultant serait le:

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

Séparer les tâches et les variables spécifiques à la distribution dans un rôle

Nous pouvons facilement séparer les tâches et les variables spécifiques à la distribution en différents fichiers .yml dédiés. Ansible nous aide à identifier automatiquement la distribution des hôtes cibles via `{{ ansible_distribution }}` et `{{ ansible_distribution_version }}` . Il nous suffit donc de nommer les fichiers .yml dédiés à la distribution.

Pour Ubuntu Xenial, le répertoire de base des rôles dirait alors:

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

A l'intérieur de `tasks/main.yml` nous pouvons désormais inclure automatiquement les variables et les tâches appropriées pour la distribution des hôtes cibles.

tâches / 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"
```

Dans les `tasks/Ubuntu16.06.yml` et `vars/Ubuntu16.04.yml` nous pouvons maintenant définir des tâches et des variables pour Ubuntu Xenial respectivement.

Lire Les rôles en ligne: https://riptutorial.com/fr/ansible/topic/3396/les-roles

Chapitre 18: Utiliser Ansible avec Amazon Web Services

Remarques

example-2: Ceci sert d'exemple, alors ne le copiez / passez pas. Au lieu de cela, pour répondre à vos besoins, vous devez personnaliser ses variables; ansible_key, règles de groupe de sécurité, etc.

exemple-1: pour désactiver la vérification de la clé hôte stricte ssh, un comportement que nous ne souhaitons pas lors de l'automatisation des tâches, nous le définissons sur `no` dans le fichier `ansible.cfg` . ie: `StrictHostKeyChecking=no`

Le fichier `ec2.py` est un script python qui exécute et renvoie vos ressources AWS en fonction de `ec2.ini` qui est le fichier de configuration à personnaliser si vous souhaitez limiter la portée de votre projet à certaines régions, balises spécifiques, etc. ...

Examples

Comment démarrer l'instance EC2 à partir des AMI Amazon officielles, la modifier et la stocker en tant que nouvelle AMI

Il s'agit d'un workflow très courant lorsque vous utilisez Ansible pour provisionner une instance AWS EC2. Cet article suppose une compréhension de base d'Ansible et, surtout, suppose que vous l'avez correctement configuré pour se connecter à AWS.

Comme le souligne la documentation officielle d'Ansible , nous allons utiliser quatre rôles:

1- **ami_find** pour obtenir l'identifiant ami sur lequel nous allons lancer notre instance EC2.

2- **ec2_ami_creation** pour lancer efficacement l'instance EC2.

3- **code_deploy** pour modifier l'instance; Cela pourrait être n'importe quoi, nous transférerons donc simplement un fichier sur la machine cible.

4- **build_ami** pour construire notre nouvelle image basée sur l'instance ec2 en cours d'exécution. Ce post suppose que vous êtes au plus haut niveau de votre projet Ansible: `my_ansible_project`

Le premier rôle: **ami_find**

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

Dans ce rôle, nous allons utiliser le module ec2_ami_find et, par exemple, nous rechercherons une machine Ubuntu et obtiendrons son **ami_id** (ami-xxxxxxxx). Maintenant, éditez le `my_ansible_project/roles/ami_find/tasks/main.yml` :

```
---
- 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 }}"
```

Le second rôle: **ec2_ami_creation**

Ici, nous allons utiliser l' `ami_id` nous avons obtenu du premier rôle, puis lancer notre nouvelle instance basée sur celle-ci:

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

Dans ce rôle, nous allons surtout utiliser ec2_module pour lancer notre instance. Maintenant, éditez le `my_ansible_project/roles/ec2_ami_creation/tasks/main.yml` :

```
---
- 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
```

Le troisième rôle: **code_deploy**

Ici, nous allons provisionner cette instance, qui a été ajoutée à un groupe appelé `just_created`

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

Dans ce rôle, nous allons utiliser le template_module pour transférer un fichier et y écrire le nom d'hôte de la machine. Maintenant, éditez le `my_ansible_project/roles/code_deploy/tasks/main.yml` :

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

puis déplacez-vous dans le dossier des modèles dans votre rôle:

`cd my_ansible_project/roles/templates` et ajouter un fichier nommé `my_file.txt.j2` contenant:

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

Le quatrième rôle: `build_ami`

Nous allons maintenant créer une image de l'instance en cours d'exécution à l'aide du module ec2_ami . Déplacer vers votre dossier de projet et:

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

Maintenant, éditez le `my_ansible_project/roles/build_ami/tasks/main.yml` :

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

Maintenant, je pense que vous vous demandez comment orchestrer tous ces rôles. Ai-je raison? Si oui, continuez à lire.

Nous allons écrire un playbook, composé de trois jeux: le premier jeu applicable sur `localhost` appellera nos deux premiers rôles, le deuxième jeu s'appliquant à notre groupe **just_created** . Le dernier rôle sera applicable sur `localhost` . Pourquoi `localhost` ? Lorsque nous voulons **gérer** des ressources AWS, nous utilisons notre machine locale, aussi simple que cela. Ensuite, nous utiliserons un fichier vars dans lequel nous `ansible_key` nos variables: `ansible_key` , `aws_region` , etc ...

créer un dossier d'infrastructure en haut de votre projet et ajouter un fichier appelé `aws.yml` :

```
---
aws_region: ap-southeast-2
```

```
ansible_key: ansible
instance_type: t2.small
```

Donc, en haut de votre projet, créez `build_base_image.yml` et ajoutez ceci:

```
---
   - 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 }}"}
```

Ça y est, n'oubliez pas de supprimer vos ressources après avoir testé ceci, ou pourquoi ne pas créer un rôle pour supprimer l'instance en cours d'exécution :-)

Comment configurer correctement Ansible pour se connecter à Amazon Web Services

La gestion des ressources AWS qui évoluent dans les limites du fichier hôte d'inventaire statique, c'est pourquoi nous avons besoin de quelque chose de dynamique. Et c'est à cela que servent les inventaires dynamiques . Commençons:

Téléchargez ces fichiers ec2.ini et ec2.py dans le dossier de votre projet:

```
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
```

Une fois terminé, rendez le fichier `ec2.py` exécutable:

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

Maintenant, exportez votre clé AWS Secret et Access en tant que variables d'environnement:

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

Pour utiliser le script `ec2.py` , nous avons besoin du SDK Python AWS, `boto` , vous devez donc l'installer:

```
sudo pip install boto
```

Pour tester si tout va bien, essayez d'exécuter `ec2.py` en listant vos ressources:

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

vous devriez voir quelque chose de similaire à:

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

Maintenant, nous voulons utiliser l'inventaire dynamique avec notre fichier d'hôtes statique. Tout d'abord, créez un dossier appelé « `inventory` , ajoutez-lui « `ec2.py` , « `ec2.ini` et notre fichier `hosts` , puis indiquez à Ansible d'utiliser ce dossier comme fichier d'inventaire:

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

Ensuite, nous devons définir la configuration au niveau du projet pour Ansible en créant un fichier de configuration Ansible dans votre dossier de projet appelé `ansible.cfg` et en ajoutant ceci:

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

Ensuite, nous devons configurer Ansible pour utiliser une clé SSH pour authentifier l'accès à nos instances EC2. L'utilisation d'un agent SSH est le meilleur moyen de s'authentifier avec les ressources, car cela facilite la gestion des clés:

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

C'est tout! Si vous avez suivi cela, vous pouvez le tester en utilisant le module `ping` , puis vous verrez vos instances en cours d'exécution configurées pour utiliser votre clé en réponse avec pong:

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

Lire Utiliser Ansible avec Amazon Web Services en ligne:
https://riptutorial.com/fr/ansible/topic/3302/utiliser-ansible-avec-amazon-web-services

Chapitre 19: Utiliser Ansible avec OpenStack

Introduction

OpenStack est une plate-forme logicielle open source pour le cloud computing. Les instances Linux peuvent être lancées / arrêtées manuellement à l'aide de l'interface Web graphique ou automatisées grâce au module cloud openstack d'ansible.

La configuration d'ansible peut être délicate, mais une fois bien configurée, son utilisation est vraiment simple et efficace pour les tests et l'environnement d'intégration continue.

Paramètres

paramètres	commentaires
hôtes: localhost	Les commandes OpenStack sont lancées depuis notre localhost
gather_facts: Faux	Nous n'avons pas besoin de rassembler des informations sur notre localhost
auth_url: https://openstack-identity.mycompany.com/v2.0	utiliser l'URL V2.0
état: présent	'present' / 'absent' pour créer / supprimer l'instance
validate_certs: Faux	utile si https utilise des certificats auto-signés
réseau: "{{NetworkName}}"	(optionnel)
auto_ip: oui	(optionnel)

Remarques

- Nous mettons l'URL d'authentification directement dans le playbook, pas dans une variable. L'URL utilisée dans vars doit être échappée.
- Attention, avec la version d'authentification URL, utilisez V2.0 au lieu de V3 dans https://openstack-identity.mycompany.com/v2.0 .
- Dans les fichiers yml, soyez très prudent lorsque vous copiez / collez du navigateur. Vérifiez deux fois les espaces pris en compte.
- Plus de détails sur: http://docs.ansible.com/ansible/list_of_cloud_modules.html#openstack

Examples

Vérifiez votre version d'Ansible

Vérifiez que les versions correctes du logiciel sont installées:

- ansible> = 2.0
- python> = 2.6
- module d'ombre pour python

```
$ansible --version
ansible 2.2.0.0

$python --version
Python 2.7.5
```

Installez 'shade' le composant python utilisé pour piloter openstack.

```
$pip install shade
```

Remarque: si vous utilisez un proxy d'entreprise, il est toujours utile de connaître le bon synt syntax

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

Rassembler des informations à partir de l'interface graphique OpenStack pour configurer Ansible

---

Notre client OpenStack est déjà configuré:

- un réseau virtuel donne des instances IP privées
- un routeur virtuel mappe l'adresse IP publique en IP privée
- une clé de sécurité a été générée
- nous avons une configuration de pare-feu par défaut pour ssh et le port 80
- nous sommes en mesure de lancer une instance grâce à l'interface Web OpenStack

Laissez rassembler toutes les informations nécessaires à partir de cette interface Web.

Les informations d'authentification peuvent être trouvées dans le fichier openstack.rc. Ce fichier peut être téléchargé à l'aide de l'interface Web OpenStack dans [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
```

Nous obtenons OS_AUTH_URL, OS_TENANT_NAME, OS_USERNAME.

Version de l'API d'authentification: OS_AUTH_URL

Attention à la version de l'API d'authentification. Par défaut, v3 est activé, mais ansible a besoin de la v2.0. Nous obtenons l'URL et définissons V2.0 au lieu de V3: https://openstack-identity.mycompany.com/v2.0

Informations sur la VM

Créez une instance à l'aide de l'interface Web OpenStack et obtenez le nom de l'image, de la saveur, de la clé, du réseau et du groupe de sécurité.

Créez un fichier ./group_vars/all avec toutes les informations nécessaires.

```
$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
```

Écrivez le playbook Ansible pour créer l'instance

Soit la commande 'os_server' du module 'Cloud' [http://docs.ansible.com/ansible/os_server_module.html] . Les variables sont définies dans ./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
```

Rassembler des informations sur notre nouvelle instance

Utilisez la commande 'os_server_facts' du module 'Cloud' [http://docs.ansible.com/ansible/os_server_module.html] . Les variables sont définies dans ./group_vars/all et le nom de l'instance est dans le serveur: "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
```

C'est très verbeux. Beaucoup d'informations sont affichées. Habituellement, seule l'adresse IP est nécessaire pour accéder à la nouvelle instance via SSH.

Obtenez votre nouvelle adresse IP publique

Au lieu d'imprimer toutes les informations, nous imprimons uniquement l'adresse IP de la première instance dont le nom est "MyOwnPersonalInstance". C'est généralement tout ce dont nous avons besoin.

```
$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
```

Supprimer notre instance

Pour supprimer notre instance, réutilisez la commande os_server avec toutes les informations d'authentification et remplacez simplement 'state: present' par '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
```

Lire Utiliser Ansible avec OpenStack en ligne: https://riptutorial.com/fr/ansible/topic/8712/utiliser-ansible-avec-openstack

Chapitre 20: Variables du groupe Ansible

Examples

Variables de groupe avec inventaire statique

Il est suggéré de définir des groupes en fonction de l'objet de l'hôte (rôles) ainsi que de la géographie ou de l'emplacement du centre de données (le cas échéant):

`inventory/production` fichiers

```
[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
```

Fichier `group_vars/all`

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

Fichier `group_vars/atlanta-webservers`

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

Fichier `group_vars/boston-webservers`

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

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

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

Après avoir lancé `ansible-playbook -i inventory/hosts install-apache.yml` (les hôtes du playbook seraient des `hosts: all`)

Les ports seraient

Adresse	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

Lire Variables du groupe Ansible en ligne: https://riptutorial.com/fr/ansible/topic/6544/variables-du-groupe-ansible

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec 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 Vars	Nick, Peter Mortensen
4	Ansible install mysql	Fernando
5	Ansible: Boucles et conditions	A K, Chu-Siang Lai, Jordan Anderson, marx, Mike, mrtuovinen, Nick, Rob H, wolfaviators
6	Ansible: Looping	calvinmclean
7	Boucles	marx, mrtuovinen
8	Comment créer un serveur Cloud DreamHost à partir d'un Playbook Ansible	Stefano Maffulli
9	Cryptage secret	fishi
10	Devenir (Escalade de privilèges)	Jordan Anderson, Willian Paixao
11	Galaxie	mrtuovinen, ydaetskcoR
12	Installation	ca2longoria, Jake Amey, Michael Duffy, mrtuovinen, Nick, PumpkinSeed, Raj, tedder42, ydaetskcoR
13	Introduction aux playbooks	32cupo, Abdelaziz Dabebi, ydaetskcoR
14	Inventaire	calvinmclean, mrtuovinen, Nick
15	Inventaire dynamique	mrtuovinen
16	Les rôles	Chu-Siang Lai, fishi, mrtuovinen, winston, ydaetskcoR

17	Utiliser Ansible avec Amazon Web Services	Abdelaziz Dabebi, another geek, ydaetskcoR
18	Utiliser Ansible avec OpenStack	BANANENMANNFRAU, Sebastien Josset
19	Variables du groupe Ansible	mrtuovinen