Ansible Provisioning

Stefan Hornburg (Racke)

Contents

Controller Installation

Virtual environment / ansible core

Check support matrix to determine whether you have a supported Python version.
Install Python venv module, e.g. system-wide for Debian/Ubuntu:

```
$ sudo install python3-venv
```

Create virtual environment for Ansible core:

```
$ python3 -m venv ansible2.18
```

Activate virtual environment:

```
$ source ansible2.18/bin/activate
(ansible2.18) $
```

Install Ansible core:

```
$ python3 -m pip install ansible-core==2.18
...
```

Target hosts

Preparing target host

The only inevitable requirements for the target are a SSH server and Python.
Modern distributions are using Python 3, therefore we recommend to force Ansible using it:

```
# group_vars/all.yml
ansible_python_interpreter: /usr/bin/python3
```

Otherwise Ansible uses Python 2 if available on the system.
This applies to:

- Debian 9 (Stretch) and later
- CentOS 8

Alpine

Install Python3:

```
~ apk update
~ apk add --quiet python3
```

Arch Linux

Synchronizing package databases:

```
$ sudo pacman -Fy
```

Install Python3:

```
$ sudo pacman -S python3
```

Debian

On a Debian host with minimal setup you can install Python as follows:

```
% apt-get update
% apt-get install python3
```

It is also useful to install *python3-apt* on Debian hosts. This allows you to run the playbook in check mode on a pristine system.

Gentoo

Installing packages requires the *equery* program:

```
$ sudo emerge app-portage/gentoolkit
```

FreeBSD

Install Python3:

```
~# pkg install python3
```

Adjust *ansible_python_interpreter* variable:

```
ansible_python_interpreter: /usr/local/bin/python3
```

RedHat

RedHat 8 and derivatives like CentOS and Alma Linux OS don't come with Python pre-installed,

```
~ dnf update
~ dnf install -y python3
```

Setup SSH connection details for all hosts

In the file `group_vars/all`:

```
---
ansible_connection: ssh
ansible_user: ansible
```

Connection variables

ansible_host IP address of the target host

ansible_user Ansible user on target host

ansible_connection Connection method (defaults to `ssh`)

ansible_ansible_ssh_private_key Private key file on the controller

Example:

```
test_vml ansible_host=192.168.6.66 ansible_user=ansible ansible_ssh_private_key_file=/home/mynames/k
```

Troubleshooting

Shared connection to 10.11.12.13

The *ansible_user* needs to be in the group sudo on Ubuntu host.

```
Oct  9 17:59:25 mytestbox sudo: nevairbe : command not allowed ; TTY=pts/1 ; PWD=/home/nevairbe ; US
```

Development and Debugging

Check mode

The Check mode is very useful to simulate what will be applied to the host without actually changing anything.

However, some tasks don't work in check mode. You can choose either to ignore errors in check mode or skip these tasks altogether. Please note that the `command` module automatically skip its tasks in check mode.

Ignore errors in check mode:

```
- name: Check if the elasticsearch package is installed
  shell: dpkg-query -W -f'${Status}' elasticsearch
  ignore_errors: "{{ ansible_check_mode }}"
```

Skip tasks in check mode:

```
- name: Check if the elasticsearch package is installed
  shell: dpkg-query -W -f'${Status}' elasticsearch
  when: not ansible_check_mode
```

YAML callback plugin

The default output for errors is hard to read, thus it is recommended to configure the YAML callback plugin in *ansible.cfg*:

```
[defaults]
...
# Use the YAML callback plugin.
stdout_callback = yaml
# Use the stdout_callback when running ad-hoc commands.
bin_ansible_callbacks = True
```

Thanks to Jeff Geerling for his blog post about this feature.

This callback can also be turned out by an environment variable:

```
export ANSIBLE_STDOUT_CALLBACK=yaml
```

Reference: `https://docs.ansible.com/ansible/latest/plugins/callback.html`

Inventory

Group names

The following characters are invalid group names: spaces, hyphens, and preceding numbers.

Tasks

A task calls an Ansible module and can be modified by task keywords, e.g. conditions or delegations.

Common Tasks

Directories and Files

Create directory:

```
- name: Create DKIM directory
  file:
    state: directory
    path: "/etc/exim4/dkim"
    owner: root
```

Copy file:

```
- name: Install DKIM domain setup script
  copy:
    src: "dkimsetup"
    dest: "/usr/local/sbin/dkimsetup"
    owner: root
    mode: 0755
```

Create file from template:

```
- name: Add DKIM settings
  template:
    src: "dkim_macros.j2"
    dest: "/etc/exim4/conf.d/main/00_dkim_macros"
    owner: root
```

Add or change line in a file:

```
- name: Register Foo's certificate chain in LDAP configuration file
  lineinfile:
    dest: /etc/openldap/ldap.conf
    regex: '^TLS_CACERT\s+'
    line: 'TLS_CACERT /etc/ssl/foo-chain.pem'
```

Packages

You can install distribution packages with the generic *package* module or with the distribution specific module.

Install Git package:

```
- name: Install Git package
  package:
    name: git
```

Cronjobs

Example for creating a cronjob:

```
- name: Create cronjob
  cron:
    name: Mrad backup
    minute: 12
    hour: 22
    user: root
    job: "/usr/local/bin/mrad /etc/mrad.cfg"
    cron_file: mrad
```

This creates a file *mrad* in the directory `/etc/cron.d` as per the *cron_file* attribute. The contents are:

```
#Ansible: Mrad backup
12 22 * * * root /usr/local/bin/mrad /etc/mrad.cfg
```

Use Cases

Replace content in files

For configuration and other text files you can use either *lineinfile* or *blockinfile* module. There also other modules for specific formats, e.g. the *xml* module for XML files.

Keywords

environment

```
- name: Install sensor package for Ubuntu
  apt:
    name: sensor
  environment:
    SERVER_ADDRESS: "{{ cyber_server_address }}"
    CUSTOMER_ID: "{{ cyber_customer_id }"
```

Jinja2 templates

Jinja2 templates are used all over space. First in mind are template files for the *template* task, but they can used in other places like in *when* conditions.

Jinja has nice features, e.g. template inheritance.

You might consider to add a header line, which makes system administrators aware of the origin on the file, e.g.:

```
{{ ansible_managed | comment }}
```

Default output is:

```
# Ansible managed
```

Filters

Reference: `https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html`

In addition to the Ansible filters you can also use Jinja filters.

Various Filters

default

This filter returns the value `443` if the variable *https_port* is not defined.

```
"{{ https_port | default(443) }}"
```

If you use the *default* filter with the special variable *omit*, the parameter would not be passed to the ansible module. That allows us to gracefully fall back to the default of the module parameter.

```
- name: Create PostgreSQL database
  postgresql_db:
    name: "{{ dbname }}"
    owner: "{{ dbuser | default(omit) }}"
```

Alias for this filter: *d*

basename

Returns the file name from a file path. Useful with *with_fileglob*.

reverse (Jinja filter)

This reverses a string, a list or any other Python object.

List filters

See also how to use Python syntax to manipulate lists.

first

Returns first element of a list.

join

Joins list members into a string:

```
 {{ monitoring_server_ips | join(' ') }}
```

This is filter is also useful to print the output from a command into a file, line by line:

```
- copy:
    content: "{{ grep_results.stdout_lines | join('\n') }}"
    dest: /tmp/grep.txt
```

length

Counts the members of the list which is convenient in a condition:

```
when: mylist | length
```

map

The *map* filter has a number of different usages.
It can be used to apply any of the string filters to all elements of the list, e.g.

```
"{{ services.split(',') | map('trim') | list }}"
```

Example using *basename* filter
Extract
If you have a list of dictionaries and you want a single value from each dictionary, apply *attribute* map:

```
{{ nginx_vhosts | map(attribute='server_name') | list }}
```

select

This filter selects any elements with matches a given condition:

```
{{ omd_versions.stdout_lines | select('search', '\\s+\\(default\\)$') }}
```

The difference between select('search') and select('match') is that the latter requires the whole element to match.

selectattr

This select all entries from a list of dictionaries where one attribute from the dictionary fulfils a certain condition.

```
{{ rest_api_fields | selectattr('type', 'equalto', 'string') | list }}
```

reject

This filter removes elements from a list of strings.
In the following example we remove all filenames which contain the string `junk`:

```
{{ filenames | reject('search', 'junk') | list }}
```

sort

This filter sorts a list, which can be useful to achieve a stable output. For example, the list of hosts in a group doesn't maintain an order.

```
{% for host in groups['all'] | sort %}
{{ hostvars[host]['inventory_hostname_short'] }} {{ hostvars[host['ansible_facts']['eth0']['ipv4']['
{% endfor %}
```

For list consisting of dictionaries, you can also sort by an attribute.
Here we are taking the list of files registered by the *find* module and sorting them by their modification time (oldest first):

```
{{ find_result.files | sort(attribute='mtime') | list }}
```

You can also sort in reverse order (newest first):

```
{{ find_result.files | sort(attribute='mtime',reverse=True) | list }}
```

unique

Reduces a list to unique items by omitting duplicate ones.

Dictionary filters

dict2items

Turns a dictionary in a list of dictionaries. Each entry in the new list has a *key* attribute with the key in the orginal dictionary and a *value* attribute with value from the original dictionary.
This is handy to loop over a dictionary:

```
{% for vg in ansible_lvm.vgs | dict2items %}
{{ vg.key }} {{ vg.value.free_g }} GB
{% endfor %}
```

String filters

capitalize

Uppercases first character of the string.

```
{{ "bullseye" | capitalize }}
# => 'Bullseye'
```

See *filter* title for uppercasing the first character of every **word**.

comment

Turns string into a comment. This is especially useful for multi-line strings. By default the filter uses the # sign.

lower (Jinja filter)

Converts whole string to lowercase.

password_hash

Encrypts string with given method:

```
{{ user_password | password_hash('bcrypt') }}
```

regex_findall

This filter can be used to extract strings, e.g. a form value from HTML retrieved by the uri module.
Form element:

```
<input type="hidden" name="csrftoken" value="db788e6feb8a4927db84d6c0da1dfe67" />
```

Filter to extract the value:

```
 "{{ sympa_response.content | regex_findall('name=\"csrftoken\" value=\"(.*)\"') | first }}"
```

Result: `db788e6feb8a4927db84d6c0da1dfe67`

regex_replace

Replaces match of regular expression.
If you want to replace a multiline string, e.g. a comment:

```
/*
Get rid of me
*/

Keep me
```

You can't use `.*` here as the dot doesn't match the newline. It works if you specify a character class with whitespace and non whitespace:

```
regex_replace('/\\*([\\s\\S]*)\\*/', '')
```

regex_search

Compares string with a regular expression and returns the match.
For example to get the numeric part from the hostname of the target:

```
- hosts: fedora33-text-box
  tasks:
    debug:
      msg: "{{ inventory_hostname | regex_search ('[0-9]+') }}"
```

This results in the following output:

```
TASK [debug] *******************************************************************
ok: [fedora33-test-box] =>
  msg: '33'
```

You can also capture part of the match:

```
- name: Determine version of RPM package to install on the server
  set_fact:
    rpm_package_version: "{{ rhn_ssl_tool.stdout | regex_search(regexp,'\\1') | first }}"
   vars:
     regexp: 'rhn-org-httpd-ssl-key-pair-spacewalk7-(.*).noarch.rpm'
```

title Converts first character of every word to uppercase.

```
{{ "foo bar" | title }}
# => 'Foo Bar'
```

trim

Removes leading and trailing whitespace.

upper (Jinja filter)

Converts whole string to uppercase.

Numeric filters

pow

Calculate 1 GB:

```
1024 | pow(3)
```

round

Rounds a number with or without decimal points:

```
10.5 | round => 11
10.444 | round(2) => 10.44
```

Data filters

json_ query

As a JSON query can result in multiple matches, the *json_ query* filter returns a list.
Uses JSON Matching Expression paths (JMESPATH).
It requires the jmespath Python library to be installed on the **controller**.

Type filters

int

Converts to integer type. Useful for numerical comparison between a variable and an integer:

```
ansible_distribution_major_version|int >= 10
```

string

Converts to string type.

```
- name: Configure IPv6
  ansible.posix.sysctl:
    name: "{{ item }}"
    value: "{{ ipv6_disabled | string }}"
  loop:
    - net.ipv6.conf.default.disable_ipv6
    - net.ipv6.conf.lo.disable_ipv6
    - net.ipv6.conf.all.disable_ipv6
```

Tests

Reference for tests builtin into Jinja: `https://jinja.palletsprojects.com/en/2.11.x/templates/#list-of-buil`

defined

Whether a variable is defined or not.

divisibleby

Whether a number is divisble without fraction. Can be useful in loops to create batches:

```
{% for email in addresses %}{{ email }}{% if loop.index is divisibleby 10 %}{{ '\n -}}{% else %};{%
```

sameas

Check if variable *internet_ access* is true:

```
- name: Ensure that APT cache is up-to-date
  apt:
    update_cache: true
    cache_valid_time: 14400
  when: internet_access is sameas true
```

success

Determines if a task was successful by looking at a variable registered by the task. Useful in combination with the *until* task keyword.

Conditions

Don't use curly braces in the conditions.

Regular expressions

Regular expressions are used by the *regex_findall* and *regex_replace* filters.

Variables and facts

Facts are information gathered by Ansible, usually by running a setup task before executing other tasks.

```
TASK [Gathering Facts] *********************************************************
ok: [buster-test-box]
```

Facts are accessible as variables.

Magic variables

Magic variables are automatically set by the Ansible but can't be overridden.
Commonly used magic variables are:

groups map with all groups and the corresponding hosts

inventory_hostname full hostname as specified in the inventory, e.g. *foo.example.com*

inventory_hostname_short short version of *inventory_hostname*, e.g. *foo*

inventory_dir useful to locate resources relative to the inventory directory

The complete list is available in the Ansible documentation.

Facts: OS variables

ansible_os_family and *ansible_distribution*

A list of common OS families and their distributions:

Debian Debian, Ubuntu, Kali

RedHat RedHat, CentOS, Fedora

Suse SLES, OpenSUSE

FreeBSD FreeBSD

Gentoo Gentoo

Alpine Alpine

Archlinux Archlinux

ansible_ distribution_ release

As code names for releases are used only by Debian based distributions, this variable makes only sense for distributions in the *Debian* ansible_os_family.

Debian 12	bookworm
Debian 11	bullseye
Debian 10	buster
Debian 9	stretch
Ubuntu 22.04	jammy
Ubuntu 20.04	focal
Ubuntu 18.04	bionic
Ubuntu 16.04	xenial

ansible_ distribution_ version

This is the distribution version number, e.g *10.5* for Debian buster.

ansible_ distribution_ major_ version

Debian Jessie	`8`
Debian Stretch	`9`
Debian Buster	`10`
Ubuntu Xenial	`16`
Ubuntu Bionic	`18`
Ubuntu Focal + Ubuntu Groovy	`20`
Fedora 31	31
SLES15	15
Gentoo	2
Kali	2020 (current year)

You can use this variable to install a Debian package only on releases that comes with that package:

```
- name: Install certbot package(s)
  apt:
    name:
      - certbot
  when:
    - ansible_distribution == 'Debian'
    - ansible_distribution_major_version|int >= 10
```

ansible_ architecture

This shows you the main architecture for your OS.

Family	Architecture
Alpine	x86_64
Archlinux	`x86_64`
Debian	`x86_64`
FreeBSD	`amd64`
Gentoo	x86_64
RedHat	`x86_64`
Suse	`x86_64`

ansible_ kernel

This shows you the kernel version, e.g. `5.10.0-9-amd64`.

ansible_ machine_ id

The machine id (UUID) for the target, which is stored in `/etc/machine_id`, e.g. `1ff77447cd174f4a9d7e37aed637c38`

ansible_ service_ mgr

Service manager on the target. Most modern distributions are using *systemd*.

Distribution	Service manager	Using it since
Alpine	service	
Archlinux	systemd	
CentOS	systemd	CentOS 7
Debian	systemd	Debian 8
Fedora	systemd	
FreeBSD	bsdinit	
Gentoo	openrc	
Suse	systemd	

Facts: Virtualization

ansible_facts['virtualization_ role'] has the value `guest` for virtual machines.

Facts: Memory

ansible_ memory_ mb provides memory information in Megabytes:

```
nocache:
  free: 828
  used: 1154
real:
  free: 82
  total: 1982
  used: 1900
swap:
  cached: 0
  free: 0
  total: 0
  used: 0
```

Facts: Networks

ansible_ interfaces are a list of the network interfaces on the target host.
Example contents:

```
ansible_interfaces:
  - lo
  - enp2s0
  - wlp0s20f3
```

ansible_ default_ ipv4 points to the interface used for the default route. In most cases it can be used to determine the "main IP address" for a server.
Example contents of the *ansible_ default_ ipv4* variable:

```
ansible_default_ipv4:
    address: 10.0.2.15
    alias: eth0
    broadcast: 10.0.2.255
    gateway: 10.0.2.2
    interface: eth0
    macaddress: 08:00:27:8d:c0:4d
    mtu: 1500
    netmask: 255.255.255.0
    network: 10.0.2.0
    type: ether
```

The variable *ansible_ all_ ipv4_ addresses* is list of all IPv4 addresses on the target host with the exception of addresses from the loopback interface (`127.0.0.0/8` address block).
Example contents:

```
["192.168.2.130", "141.57.69.174", "192.168.2.112"]
```

Facts: SELinux

ansible_ selinux

Facts: Service Manager

In order to determine whether the host uses Systemd you can check the *ansible_service_mgr" variable.

Connection variables

ansible_user user name used for connecting to the target host

Reference: `https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-ho`

Playbook variables

ansible_run_tags

List of tags passed to ansible-playbook. Using `--tags=certbot,nginx` results in:

```
ansible_run_tags:
  - nginx
  - certbot
```

Without `--tags` the list contains one element `all`:

```
ansible_run_tags:
  - all
```

Dynamic variables with *set_fact*

The *set_fact* module allows you to generate variables from other variables (from inventory, register, ...).

Extract output from a command in a loop

It can be a challenge to find out how to use *set_fact* based on a previous command in a loop, but it is quiet simple if you use the *sum* filter:

```
- set_fact:
    modules_list:
      "{{ command_output.results | sum(attribute='stdout_lines', start=[]) }}"
```

Access facts of other hosts

You can access variables for other hosts from the current inventory through the *hostvars* array.

Python

You can apply standard Python methods to variables.

String methods

Useful methods are *join*, *split*, *startswith* and *endswith*.

You can use a regular expression instead of *startswith* or *endswith*, but these methods improve the readablility.

In the following we check if the listen address is a Unix socket. If not we configure the allowed clients for TCP address.

```
{% if item.pool_listen.startswith('/') == false %}
listen.allowed_clients = {{ item.pool_listen_allowed_clients | default('127.0.0.1', true) }}
{% endif %}
```

See also: `https://docs.python.org/3/library/stdtypes.html#string-methods`.

Slicing

Slicing can be used for strings and lists with the same syntax [*start*:*end*]. The first element is 0, so `[1:]` removes the first character from a string respective the first element of an array.

Returns the string represented by the variable *backup_ base_ directory* with the first character removed:

```
{{ backup_base_directory[1:] }}
```

Removes subdomain to use as cookie domain:

```
cookie_domain: ".{{ app_domain.split('.')[1:] | join('.') }}"
```

Lists

Lists can be combined with the + operator:

```
- set_fact: Combine two arrays
    myarray: "{{ myarray + youarray }}"
```

This can also used to add element(s) to a list, just wrap them in a new list `[]`:

```
- set_fact: Add element to array
    myarray: "{{ myarray + ['element'] }}"
```

Dicts

This is good example to prevent variable nesting while inside a Jinja template:

```
Environment="FCGI_CHILDREN={{ sympa.web.get(unit_name + '_procs') }}"
```

Precedence

`https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#ansible-variable-prec`

Lookups

Lookups can be used for file contents, templates, environment variable and various other things. Lookups are always executed on the **controller**.

Files

Ansible will lookup the file in multiple directories if you are using a relative path. Run the playbook with -vvvvv to see the paths to these directories.

In this case the APT key is stored in the *files* directory of the role.

```
- name: Ensure key for Docker repository is imported
  apt_key:
    data: "{{ lookup('ansible.builtin.file', 'docker_apt_pgp.asc') }}"
    id: 0EBFCD88
    state: present
```

The file lookup **removes newlines** at the end of the file **which breaks SSH private keys** (`https://github.com/ansible/ansible/issues/30829`).

Here comes a neat trick to avoid it by utilizing YAML syntax:

```
- name: Add SSH private key
  copy:
    content: |
      {{ lookup(''ansible.builtin.file', users_inventory_dir + '/files/ssh-keys/' + ssh_id ) }}
    dest: "/home/{{ item.username }}/.ssh/id_rsa"
    owner: "{{ item.username }}"
    mode: 0600
  no_log: true
```

If your file is located in the inventory, you can use the *inventory_dir* variable inside the lookup:

```
"{{ lookup(''ansible.builtin.file', inventory_dir + '/../files/ssh-keys/checkmk-ssh.pub') }}"
```

Note: you can use *with_file* as alternative to the file lookup, see *authorized_key* example.

Encrypted files

Read a value from a vaulted YAML file:

```
"{% set secrets = lookup('ansible.builtin.unvault', 'vault/' + lookup('env', 'USER') + '-secrets.yml
```

Environment variables

This example shows how to determine the user on the controller:

```
"{{ lookup('ansible.builtin.env', 'USER') }}"
```

DNS records

Query DNS A record for *example.com*:

```
"{{ lookup('community.general.dig', 'example.com.') }}"
```

Ouery DNS PTR record for *example.com*:

```
"{{ lookup(community.general.dig,'96.7.128.198/PTR') }}"
```

Specific name servers can be used as follows:

```
"{{ lookup('community.general.dig', 'example.com.', '@199.43.135.53,199.43.133.53' )}}"
```

Requirements

Needs **dnspython** Python module.

DNS TXT records

DNS TXT records are used for various things. For example common email authentication methods like SPF, DKIM and DMARC are using TXT records.

You can use the *dnstxt* lookup plugin to check the DKIM record for the domain *linuxia.de* and selector *mail*:

```
- name: Check whether DKIM DNS entry exists
  debug:
    msg: "{{lookup('community.general.dnstxt', 'mail._domainkey.linuxia.de')}}"
```

The output looks like:

```
msg: v=DKIM1; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDywbZfxszCUIV3WsMWChd+8iergHXcsWNR/vPtc4gwtbR+
```

Requirements

Needs **dnspython** Python module.

Conditions

Conditions determine whether a task is executed.

Arguments to conditions are automatically passed through templating. Don't wrap them in curly braces.

Examples:

Whether the key `owner` exist in the *item* dict:

```
when: "'owner' in item"
```

Whether list is not empty:

```
when: mylist | length
```

Loops

The traditional way for loops in Ansible using *with_ items*, *with_ dict*, *with_ subelements* etc. has been replaced by *loop*: `https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#migrating-from` This has been introduced with Ansible 2.5. Of course you can still use the *with_* loops.

Documentation for *with_ subelements*: `https://docs.ansible.com/ansible/latest/plugins/lookup/subelement`

with_ subelements example

Data:

```
users:
  - username: backuptransfer
    ssh_keys:
      - backup
    ssh_keys_to_remove: []
```

Task:

```
- name: Add SSH keys
  authorized_key:
    user: "{{ item.0.username }}"
    key: "{{ lookup('file', inventory_dir + '/files/ssh-keys/' + item.1 + '-ssh.pub') }}"
    state: present
  with_subelements:
    - "{{ users }}"
    - ssh_keys
```

Loop over comma separated list

```
loop: "{{ services.split(',') | map('trim') | list }}"
```

Handlers

Handlers are only running if one of the triggering tasks is in the *changed* status. Also they are executed at the end of the playbook. So if multiple tasks triggering a handler, it is only run once.

A typical task executed by a handler is the restart of a service:

```
- name: Restart Exim4
  service:
    name: exim4
    state: restarted
```

Flushing handlers

It is also possible to run handlers in the middle of playbook:

```
- name: Flush handlers
  meta: flush_handlers
```

Commandline

Ad-hoc mode

With the ad-hoc mode you can execute arbitrary commands on the target hosts.

You can execute commands with `ansible -a` for all hosts in a group, e.g a simple call of the `id` program, which doesn't need superuser permissions:

```
$ ansible shopserver -i production -a id

www1.linuxia.de | CHANGED | rc=0 >>
uid=1001(ansible) gid=1001(ansible) Gruppen=1001(ansible),1002(sysadmin)
```

Add `--become` for commands which need to be executed as superuser:

```
$ ansible shopserver -i production --become -a "grep Failed.password /var/log/auth.log"
```

Use the pseudo group *all* to execute the command on all hosts in the given inventory.

Tags

You can run tasks for certain tags respective skip them.

```
ansible-playbook --tags=letsencrypt site.yml
```

```
ansible-playbook --skip-tags=letsencrypt site.yml
```

Tasks

List all tasks which would be executed by the playbook `site.yml`.

```
ansible-playbook --tags=letsencrypt --list-tasks site.yml
```

List tasks tagged with *letsencrypt* which would be executed by the playbook `site.yml`.

```
ansible-playbook --tags=letsencrypt --list-tasks site.yml
```

Limit

Limit the execution of the playbook to one or more host groups:

```
ansible-playbook -i prod --limit myshop,mydb plays/ecommerce.yml
```

Vault

Password file

The location of the Vault password file can be set by:

commandline argument `-vault-password-file`

environment variable *ANSIBLE_VAULT_PASSWORD_FILE*

ansible configuration *vault_password_file*

Encrypt file

```
$ ansible-vault encrypt repo-git-id
Encryption successful
```

Encrypt single variable

```
$ ansible-vault encrypt_string --stdin-name mysql_root_password
...
```

The result looks as follows and you can put that as is into a YAML inventory file:

```
mysql_root_password: !vault |
  $ANSIBLE_VAULT;1.1;AES256
  62373730376636323437633965356564366233326266663238343765353634646362323234663664
  6239663063343861656234663035323765386366333632630a336363396562653437366339316538
  37383639366532633839313961383465646164626638316238386239303264303463633936316539
  3237356539366266640a306363393136656636346435303862653335333564326638336539313161
  3134
```

Decrypt single variable

Unfortunately Ansible doesn't offer a command to decrypt single variables in a YAML file.
Here the `yq` command comes to rescue.
Sample `yq` installation on Debian:

```
apt-get install jq
pip install yq
```

Now you can select the variable with `yp` and output it as raw string instead of json (`-r` option):

```
yq -r .mysql_root_password production/production.yml
```

This output can be piped into the `ansible-vault` command as follows:

```
yq -r .mysql_root_password production/production.yml | ansible-vault decrypt
```

This also works for variables on deeper levels of the YAML file. Please note that some keys with ”special characters” like `.` and `-` needs to be quoted.

```
yq -r  '."nginx-servers".hosts."example.linuxia.de".htpasswd_credentials[0].password' production/pro
```

Troubleshooting

Hangs while gathering facts

Check connection first:

```
$ ansible -m ping -i live DEBIAN
```

Exclude hardware from facts:

```
$ ansible -m setup -i live DEBIAN -a 'gather_subset=!hardware'
```

Containers

Ansible Docker Reference: `https://docs.ansible.com/ansible/latest/scenario_guides/guide_docker.html`

SELinux

Ansible provides the fact *ansible_selinux* with information about the SELinux status. It is a dict with the fields *config_mode*, *mode*, *policyvers*, *status* and *type*.

Typical values are:

Distribution / Release	Status	Mode	Type
Debian	disabled	-	-
Ubuntu	disabled	-	-
Centos	enabled	permissive	targeted
Fedora	enabled	enforcing	targeted

However this fact requires installation of a SELinux Python library. For recent releases these are *libselinux-python3* for RedHat OS family and *python3-selinux* for the Debian OS family.

You can refresh the *ansible_selinux* fact after the installation of the library with the *setup* module:

```
- name: Refresh SELinux fact
  setup:
    filter: 'ansible_selinux'
```

Modules

apache2_module

```
- name: Enable fcgid module
  apache2_module:
    name: fcgid
    state: present
```

apt

The *apt* module is used to install APT packages from repositories and from files. To install multiple packages, pass a list to *name*:

```
- name: Install Perl packages needed by mrad
  apt:
    name:
      - dar
      - libappconfig-perl
      - libdate-manip-perl
      - libmime-lite-perl
      - libfilesys-df-perl
```

Installing from a file requires to transfer the file to the target first:

```
- name: Copy Sympa package to target
  copy:
    src: "{{ sympa.package_file }}"
    dest: /var/cache/apt/archives
```

Now you can install the package with the help of the *deb* parameter:

```
- name: Install Sympa package from a file
  apt:
    deb: /home/racke/sympa-community/sympa-6.2.48~dfsg-1_amd64.deb
```

Install a Debian package only on releases that comes with that package:

```
- name: Install certbot package(s)
  apt:
    name:
      - certbot
  when:
    - ansible_distribution == 'Debian'
    - ansible_distribution_major_version|int >= 10
```

Purging a package is triggered by the *purge* parameter:

```
- name: Purge Sympa package
  apt:
    name: sympa
    state: absent
    purge: yes
```

To update the APT cache without installing a package:

```
- name: Update APT cache
  apt:
    update_cache: yes
  when: ansible_os_family == 'Debian'
```

apt-key

The *apt_key* module manages APT keys for authenticating packages.
Example:

```
- name: Install APT key for MySQL repository
  apt_key:
    data: "{{ lookup('file', 'mysql-apt-repo.key') }}"
    state: present
```

This requires `gpg` installed on the target machine:

```
- name: Install GnuPG
  apt:
    name:
      - gpg
```

apt-repository

Add Ubuntu PPA repository:

```
- name: Add Linbit DRBD PPA repository
  apt_repository:
    repo: ppa:linbit/linbit-drbd9-stack
```

In most cases you will also need to update the APT cache when adding the repository. Otherwise you wouldn't be able to install packages from the repository.

```
- name: Add Linbit DRBD PPA repository
  apt_repository:
    repo: ppa:linbit/linbit-drbd9-stack
    codename: focal
  register: drbd_linbit_apt

- name: Update APT cache
  apt:
    update_cache: yes
  when: drbd_linbit_apt.changed
```

You can also select the filename for the APT sources file configuring the repository:

```
- name: Install apt repository for Gitlab
  apt_repository:
    repo: "deb https://packages.gitlab.com/gitlab/gitlab-ce/ubuntu/ focal main"
    filename: gitlab-repo
```

The resulting file name would be `/etc/apt/sources.list.d/gitlab-repo.list`.

assert

The *assert* module checks whether given conditions are met. It fails if not.

```
- name: Ensure that role is executed on Debian Jessie or Debian Stretch
  assert:
    that:
      - ansible_distribution == 'Debian'
      - ansible_distribution_major_version == '8' or ansible_distribution_major_version == '9'
    msg: "Debian Jessie or Debian Stretch required for docker role"
```

authorized_key (SSH authorized keys)

The *authorized_key* module allows you to add or remove keys from user accounts.

```
- name: "Add SSH keys"
  authorized_key:
    user: interch
    state: present
    key: "{{ item }}"
  with_file:
    - racke-ssh.pub
    - linuxia-ssh.pub
```

blockinfile

The *blockinfile* module is similar to the *lineinfile module*, but allows you to manipulate multiple lines in a file.

Example:

```
- name: Ensure that SSH port is in user's config
  blockinfile:
    path: "/home/{{ username }}/.ssh/config"
    block: |
      Host *.example.com
        Port 7494
    create: yes
    mode: 0644
  become_user: "{{ username }}"
```

If the file doesn't exist already, it be will created (because of *create* set to *yes*). In that case the content of the SSH config file would look like:

```
# BEGIN ANSIBLE MANAGED BLOCK
Host *.example.com
  Port 7494
# END ANSIBLE MANAGED BLOCK
```

If you want to put multiple entries in a single file (either through separate tasks or by running the *blockinfile* task in a loop), you need to add an unique marker to the task, e.g.:

```
- name: Ensure that SSH port is in user's config
  blockinfile:
    path: "/home/{{ username }}/.ssh/config"
    block: |
      Host *.example.com
        Port 7494
    marker: "# {mark} ANSIBLE MANAGED BLOCK FOR EXAMPLE.COM"
    create: yes
    mode: 0644
  become_user: "{{ username }}"
```

Now the output would look like:

```
# BEGIN ANSIBLE MANAGED BLOCK FOR EXAMPLE.COM
Host *.example.com
  Port 7494
# END ANSIBLE MANAGED BLOCK FOR EXAMPLE.COM
```

copy

The *copy* module copies files to the target.

cpanm

The *cpanm* module allows you to manage Perl modules.

```
- name: Install Dancer2 module with cpanm
  cpanm:
    name: Dancer2
```

You can also specify a minimum version:

```
- name: Install Dancer2 module with cpanm
  cpanm:
    name: Dancer2
    version: '0.301000'
```

It is not possible to use a list of Perl modules in the *name* parameter, so you need to run the task in a loop:

```
- name: Install Dancer2 plugins with cpanm
  cpanm:
    name: "{{ item }}"
  loop:
    - Dancer2::Plugin::Email
    - Dancer2::Plugin::Auth::Extensible
    - Dancer2::Plugin::GraphQL
```

debconf

The *debconf* module allows you to preseed values for the debconf configuration.

```
- name: Preseed Debconf values for Sympa (wwsympa_url)
  debconf:
     name: sympa
     question: wwsympa/wwsympa_url
     value: "https://{{ common.web.domain }}/sympa"
     vtype: string
```

docker_container

The *docker_container* module
Start Elasticsearch container with image from GitHub:

```
- name: Ensure that Elasticsearch container is running
  docker_container:
    name: "esdemo"
    image: "elasticsearch:7.10.1"
    state: 'started'
    restart_policy: always
```

You can also use registries other than Docker hub for the images, e.g. the Elasticsearch registry at `https://www.docker.elastic.co/r/elasticsearch`:

```
- name: Ensure that Open Source Elasticsearch container is running
  docker_container:
    name: "esdemo"
    image: "docker.elastic.co/elasticsearch/elasticsearch-oss:7.10.1-amd64"
    state: 'started'
    restart_policy: always
```

fetch

The *fetch* module retrieves a file from a remote host.
This module doesn't support check mode. Recursive fetching is not supported.
Example:

```
- name: Upload ~/.gnupg archive to Ansible Controller
  fetch:
    src: '{{ reprepro_home + "/" + reprepro_gpg_snapshot_name }}'
    dest: '{{ reprepro_gpg_snapshot_path + "/" + reprepro_gpg_snapshot_name }}'
    flat: True
```

file

The *file* module manages files and directories on the target.
Create a directory:

```
- name: Ensure that installation directory exists
  file:
    state: directory
    path: /usr/local/sympa
    owner: sympa
    group: sympa
    mode: 0755
```

find

The *find* module locates files and directories. It supports a subset of the functionality of the Unix *find* command.
Example:

```
 - name: Get list of available Apache modules
      find:
        file_type: 'file'
        paths: '/etc/apache2/mods-available/'
        patterns: '*.load'
      register: apache_mods_available
```

You can specify multiple paths and patterns by passing a list to these parameters.
In order to extract all file paths from the result use the map filter:

```
- name: Paths for Apache modules
  set_fact:
    apache_mods_paths: "{{ apache_mods_available.files | map(attribute='path') | list }}"
```

This gives you a list of full paths:

```
- /etc/apache2/mods-available/fcgid.conf
- /etc/apache2/mods-available/fcgid.load
- /etc/apache2/mods-available/http2.conf
- /etc/apache2/mods-available/http2.load
```

In order to get only the filenames, apply the basename filter as well:

```
- name: Filenames for Apache modules
  set_fact:
    apache_mods_files: "{{ apache_mods_available.files | map(attribute='path') | map('basename') | l
```

Now the resulting list is:

```
- fcgid.conf
- fcgid.load
- http2.conf
- http2.load
```

get_url

The *get_url* module download files to the target.

```
- name: Download Debian 10 image for KVM
  get_url:
    url: https://cdimage.debian.org/cdimage/openstack/current-10/debian-10-openstack-amd64.qcow2
    checksum: sha256:85c43e90a13f5c1021afd07f843ace498b4bca4ff71b8e5c50d70e2566a304aa
    dest: /var/lib/libvirt/images/debian10.qcow2
```

getent

Determine the home directory of the remote user:

```
- name: Retrieve account information for remote user
  getent:
    database: passwd
    key: "{{ ansible_ssh_user }}"
    split: ":"

- name: Set fact for home directory
  set_fact:
    user_home: "{{ getent_passwd[ansible_ssh_user][4] }}"
```

git

The git module clones a Git repository.
Notable parameters are:

depth if set, creates a shallow clone

git_config

Updates Git configuration files, e.g user name and email:

```
- name: Configure Git username for etckeeper commits
  git_config:
    name: user.name
    scope: global
    value: "{{ git_user_name }}"

- name: Configure Git email for etckeeper commits
  git_config:
    name: user.email
    scope: global
    value: "{{ git_user_email }}"
```

The *scope* setting might be a bit confusing - *global* refers to the user's global configuration `~/.gitconfig`, while *system* refers to `/etc/gitconfig`.

group

Creates a user group.

```
- name: Create unix group for Sympa
  group:
    name: sympa
```

hostname

The *hostname* module sets the hostname of the target system, e.g:

```
- name: Set hostname
  hostname:
    name: "{{ inventory_hostname_short }}"
```

The variable *inventory_ hostname_ short* contains the first part of the fully qualified domain name (FQDN), e.g. `foo` for `foo.linuxia.de`.

This adjusts the hostname in `/etc/hostname` and creates or updates `/etc/machine-info`:

```
$ cat /etc/hostname
foo
$ cat /etc/machine-info
PRETTY_HOSTNAME=foo
```

import_ role, include_ role

The *import_ role* module loads the given role similar to roles specified with the *roles:* keyword.

With *import_ role* Ansible checks first whether the role is available before executing the playbook.

The *include_ role* loads a role *dynamically*.

Instead of executing the role as usual you can run a specific task file:

```
- name: Run task file dkim-key-pair.yml from exim-dkim role
  import_role:
    name: exim-dkim
    tasks_from: dkim-key-pair.yml
```

import_ tasks

The *import_ tasks* module allows you to import tasks from another task file than `main.yml`:

```
- name: Install systemd timers
  import_tasks: timers.yml
```

import_ tasks can not run in a loop.
This is useful to group related tasks and keep the main task file lean.
In addition you can restrict the scope to a subset of the targets, e.g. based on the distribution.

```
- name: Initialize PostgreSQL cluster
  import_tasks: initialize.yml
  when: ansible_os_family in ['Alpine', 'RedHat', 'FreeBSD', 'Suse']
```

known_hosts

Ansible provides the *known_hosts* module for adding or removing host keys from the `~/.ssh/known_hosts` file.

```
- name: Add Git remote to known hosts
  known_hosts:
    name: 'git.linuxia.de'
    key: '[git.linuxia.de] ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBE
    hash_host: yes
```

The ECDSA host key is located in the file `/etc/ssh/ssh_host_ecdsa_key.pub` on the server. Don't use the RSA host key, which might not accepted.

Another use case is a backup server which pulls backups from a number of clients with SSH. The SSH public key of any client needs to be added to the `~/.ssh/known_hosts` file of the server:

```
- name: Add known hosts entries for backup clients
  known_hosts:
    name: "{{ item }}"
    key: "{{ item }}: {{ hostvars[item].ssh_host_key }}"
    hash_host: yes
  with_items: "{{ groups['backup_clients'] }}"
  when:
    - "'backup_servers' in group_names"
```

lineinfile

The *lineinfile* module add or updates a **particular** line in a **text** file.
It isn't applicable to the following use cases:

- Replacing multiple occurrences of the same line (see *replace*)
- Replacing a block with multiple lines blockinfile (see *blockinfile*)
- Manipulating files with structured data such as XML and JSON

Configure proxy

```
- name: Configure proxy (Debian, Ubuntu)
  lineinfile:
    dest: /etc/environment
    regexp: '(?i)^no_proxy='
    line: "no_proxy=example.com,localhost"
  when: ansible_os_family == 'Debian'
```

Unattended upgrades

This is an example where we add the report email address from the variable *unattended_ upgrades_ report_ email* to the configuration of the *unattended-upgrades* Debian package:

```
- name: Add email address for reports on upgrades
  lineinfile:
    dest: "/etc/apt/apt.conf.d/50unattended-upgrades"
    regexp: '^(//)?Unattended-Upgrade::Mail\s+"(.*)";'
    line: "Unattended-Upgrade::Mail \"{{ unattended_upgrades_report_email }}\";"
```

The orignal line after installation looks like that:

```
//Unattended-Upgrade::Mail "root";
```

So the regular expression needs to cover both the option with or without the `//` at the beginning of the line.

Adjust PHP FPM configuration

```
- name: Adjust PHP FPM configuration
  lineinfile:
    path: "/etc/php-fpm.d/www.conf"
    line: "catch_workers_output = yes"
    state: present
    regexp: "^;?catch_workers_output"
```

This makes sure that the line `catch_workers_output = yes` appears in the configuration file. It replaces an existing line by regular expression, so it would work whether the configuration directive is commented out or not:

- `catch_workers_output = no`
- `;catch_workers_output = yes`

Cluster nodes (lineinfile in a loop)

In some cases it may be warranted to add multiple lines to a file by using *lineinfile* in a loop.

Inserting a list of cluster nodes into `/etc/hosts` could be such a case:

```
- name: Ensure that the node hostnames can be resolved (essential for DRBD/OCFS2 clustering)
  lineinfile:
    path: /etc/hosts
    line: '{{ hostvars[item].ansible_default_ipv4.address }} {{ item }}'
    regexp: '^{{ hostvars[item].ansible_default_ipv4.address | regex_escape }}'
    insertbefore: '^$'
    firstmatch: yes
  loop: "{{ groups['mycluster'] }}"
```

The combination of `insertbefore: '^$'` and `firstmatch: yes` instructs *lineinfile* to add the new lines before the first empty line in `/etc/hosts`.

The pristine file for host *debian* after initial installation may look like that:

```
127.0.0.1    localhost
127.0.1.1    debian.localdomain debian

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

And after you added *node1*, *node2*, *node3* from the group *mycluster*:

```
127.0.0.1    localhost
127.0.1.1    debian.localdomain debian
10.1.118.11 node1
10.1.118.12 node2
10.1.118.13 node3

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Write a file on the controller

```
- name: Update Python interpreter
  ansible.builtin.lineinfile:
    path: "{{ playbook_dir }}/host_vars/{{ inventory_hostname }}.yml"
    regex: '^ansible_python_interpreter:'
    line: "ansible_python_interpreter: /usr/bin/python3"
    create: yes
  delegate_to: localhost
  become: false
```

lineinfile uses root user by default. If you want to write files as the user on the controller change that with `become: false`.

locale_gen

Debian only enables the basic locales by default. You can use the *locale_gen* module to add needed locales:

```
- name: Ensure that German locale exists
  locale_gen:
    name: de_DE.UTF-8
    state: present
```

mysql_db, mysql_user

Ansible provides the *mysql_user* module and the *mysql_db* module for creating MySQL roles and databases.

Create database:

```
- name: Create database for Sympa
  mysql_db:
    name: sympa
    state: present
```

Create user for the Sympa database:

```
- name: Create MySQL user for Sympa
  mysql_user:
    name: sympa
    priv: 'sympa.*:ALL'
    password: nevairbe
    state: present
```

Create remote user for replicating the Sympa database:

```
- name: Add remote replication user
  mysql_user:
    name: replicator
    host: '10.0.2.20'
    password: nevairbe
    priv: "sympa:*:REPLICATION SLAVE"
    state: present
```

Dump database:

```
- name: Create a dump from Sympa database
  mysql_db:
    name: sympa
    state: dump
    target: "/home/backup/sympa-{{ ansible_date_time.date }}.sql"
```

Please note that the dump file is world readable, so you need to protect sensitive data with restricted permissions for the dump directory.

mysql_query

```
- name: Update robots
  community.mysql.mysql_query:
    login_db: sympa
    query: 'UPDATE list_table SET robot_list = "example.org"'
```

mysql_replication

Determine MySQL slave status.

```
- name: Get slave status
  mysql_replication:
    mode: getslave
  register: mysql_slave_info

- set_fact:
    mysql_slave_health: |-
      {% if mysql_slave_info.Slave_IO_Running == 'Yes' and mysql_slave_info.Slave_SQL_Running == 'Ye
```

package

The *package* module is a generic package manager for supported distributions.

Simple tasks like installing Git can be accomplished with it, but software might be packaged with different names in the distributions.

```
- name: Ensure that Git is installed
  package:
    name:
      - git
```

Specific modules by distributions are:

Distributions	Ansible module
Alpine	apk
ArchLinux	pacman
Debian	apt
Gentoo	portage
RedHat	yum
Suse/SLES	zypper
Ubuntu	apt

pam_limits

This module changes the Linux PAM limits in the file `/etc/security/limits.conf` (or the file specified by the *dest* parameter).

To impose soft and hard limits for the number of open files to user *foo*:

```
- name: ulimits - Add soft value of maximum open files to user foo
  pam_limits:
    domain: foo
    limit_type: soft
    limit_item: nofile
    value: "1024"
```

```
- name: ulimits - Add hard value of maximum open files to user foo
  pam_limits:
    domain: foo
    limit_type: hard
    limit_item: nofile
    value: "2048"
```

Please note that this module doesn't indicate possible changes in *check mode*.
Available limit types are:

core	limits the core file size (KB)
data	max data size (KB)
fsize	maximum filesize (KB)
memlock	max locked-in-memory address space (KB)
nofile	max number of open file descriptors
rss	max resident set size (KB)
stack	max stack size (KB)
cpu	max CPU time (MIN)
nproc	max number of processes
as	address space limit (KB)
maxlogins	max number of logins for this user
maxsyslogins	max number of logins on the system
priority	the priority to run user process with
locks	max number of file locks the user can hold
sigpending	max number of pending signals
msgqueue	max memory used by POSIX message queues (bytes)
nice	max nice priority allowed to raise to values: [-20, 19]
rtprio	max realtime priority
chroot	change root to directory (Debian-specific)

pip

The *pip* module install and removes Python packages.

```
- name: Ensure that SSLyze is installed
  pip:
    name: sslyze
```

portage

The *portage* module manages packages on Gentoo through the *emerge* command.
In general the "packages" are compiled from sources, so it can take a **long time** to install a Gentoo package.

```
- name: Ensure that Git is installed
  portage:
    name:
      - git
```

postgresql_db, postgresql_user

Ansible provides the *postgresql_ user* module and the *postgresql_ db* module for creating PostgreSQL roles and databases.

Creating a role can be as simple as in the following example:

```
- name: Create PostgreSQL role for Sympa
  postgresql_user:
    name: sympa
```

Now you can create a database for this role:

```
- name: Create PostgreSQL database for Sympa
  postgresql_db:
    name: sympa
    encoding: UTF-8
    lc_collate: en_US.UTF-8
    lc_ctype: en_US.UTF-8
    template: template0
    owner: sympa
    state: present
```

postgresql_query

You can ensure that a column exists in a table with using this module:

```
- name: Add unsubscribe_link field to subscriber table
  postgresql_query:
    db: sympa
    query: 'ALTER TABLE subscriber_table ADD COLUMN IF NOT EXISTS
  unsubscribe_link text'
  become: true
  become_user: postgres
  vars:
    ansible_ssh_pipelining: true
```

reboot

The *reboot* module reboots the target host.

The output for a successful reboot looks like that:

```
ok: [buster-test-box] =>
  msg:
    changed: true
    elapsed: 30
    failed: false
    rebooted: true
```

replace

The *replace* module replace all instances of a pattern within a file.

E.g. if you want to prepend every line in a file with #:

```
- replace:
    path: "/home/racke/example.txt"
    regexp: '^(.*)$'
    replace: '# \1'
```

service

The *service* module start and stops services.

On hosts with systemd, the task is delegated to the *systemd* module.

Services are not always enabled and started when you install the corresponding package, so add a task to ensure that this is the case:

```
- name: Ensure that rsyslog is installed
  package:
    name: rsyslog

- name: Ensure that rsyslog is enabled and running
  service:
    name: rsyslog
    state: started
    enabled: yes
```

sleep parameter

Some services have stop scripts that terminate immediately even before the actual process is vanished. This can prevent a successful start when you restart the service.

To alleviate this problem you can add a value for the sleep parameter:

```
- name: restart mysql
  service:
    name: {{ mysql_daemon }}
    state: restarted
    sleep: 5
```

selinux

Disable SELinux:

```
- name: Disable SELinux (Fedora)
  selinux:
    state: disabled
   when: ansible_distribution == 'Fedora'
```

setup

The *setup* module allows you to gather facts on the remote hosts. This is done automatically at the beginning of a play unless you disable it with setting *gather_facts* to `False`.

In some cases you might need to refresh the facts. You can find one example in the SELinux section.

Another one follows:

```
# Vagrant box debian/buster64 has "buster/sid" in /etc/debian_version
- name: Ensure that base-files package is up-to-date for the Debian buster assertion
  apt:
    name:
      - base-files
    state: latest
  register: basefiles

- name: Reread facts
  setup:
  when:
    basefiles.changed
```

Installing `base-files` changes the content of `/etc/debian_version` from `buster/sid` to `10.0`, but this is not reflected in the *ansible_distribution_major_version* variable.

Without refreshing the facts the following assertion would fail in a fresh buster VM:

```
- name: Role requires Debian Buster
  assert:
    that:
      - ansible_distribution == 'Debian'
      - ansible_distribution_major_version == '10'
    fail_msg: "Distribution {{ ansible_distribution }}, major version: {{ ansible_distribution_major
```

service_facts

You sometimes need to find out the status of the services, especially in a heterogeneous environment. The *service_facts* sets a dict with all existing services, whether they are running or not.

So you can use that to determine whether the service exists at all and and is in a certain state:

```
- name: Populate service facts
  ansible.builtin.service_facts:

- name: Adjust firewalld settings
  import_role:
    name: firewalld
  when:
    - "'firewalld.service' in ansible_facts.services"
    - ansible_facts.services['firewalld.service']['state'] == 'running'
```

synchronize

Reference: *synchronize* module

This can be used to synchronize backups from server A (*sync_src_server*) to server B (*sync_dest_server*). Both need to be part of the inventory:

```
- name: Synchronize
  synchronize:
    mode: push
    src: "/data/backups/"
    dest: "/srv/backups/server-a/"
   delegate_to: "{{ sync_src_server }}"
   when: inventory_hostname == sync_dest_server
```

Ensure that *rsync* is installed on both hosts:

```
- name: Ensure that rsync is installed on both hosts
  package:
    name: rsync
  when: inventory_hostname in [sync_src_server, sync_dest_server]
```

systemd

The *systemd* module is the similar to the *service* module, but is specifically used to manage systemd services.

After changing an unit file, let systemd know about it:

```
- name: Reload systemd daemon on unit file changes
  systemd:
    daemon_reload: true
```

This is the equivalent to the `systemctl daemon-reload` command.

template

The *template* module generates files from templates.

The template needs to exists on the Ansible controller. There is *remote_src* parameter as for the *copy* module.

timezone

```
- name: Set timezone to Europe/Berlin
  timezone:
    name: Europe/Berlin
```

ufw

Manages ufw firewall rules.

```
- name: Open firewall port for IMAP/TLS
  ufw:
    rule: allow
    port: '993'
    proto: tcp
```

unarchive

The *unarchive* module unpacks an archive.

Please make sure that the file permissions are correct inside the archive. Although there is a *mode* parameter, it is pretty much useless for software archives as the same mode is applied to files **and** directories.

As the *unzip* binary is not always installed by default (e.g. Debian), make sure it is installed first when dealing with zip archives.

For example:

```
- name: Install unzip
  apt:
    name:
      - unzip
  when:
    - software_archive is match('.*\.zip')
```

Alpine Linux uses tar provided by busybox, which is not sufficient to unpack tar archives with this module.

Install GNU tar on Alpine Linux:

```
- name: Install GNU tar on Alpine Linux
  apk:
    name: tar
```

FreeBSD has a similar problem:

```
fatal: [freebsd-test-box]: FAILED! => changed=false
  msg: Failed to find handler for "/usr/local/src/sympa/sympa-6.2.59b.1.tar.gz". Make sure the requi
```

Install GNU tar on FreeBSD:

```
- name: Install GNU tar on FreeBSD
  pkgng:
    name: gtar
```

uri

The *uri* module interacts with web services.

Example:

```
- name: Create schema fields
  uri:
    url: "http://localhost:{{ solr_port }}/solr/{{ solr_core }}/schema"
    method: "POST"
    body_format: json
    body: "{{ { 'add-field' : solr_rest_fields} | to_json }}"
```

It can also be used to test correctness of web server configuration:

```
- name: Test redirection of / to /sympa
  uri:
    url: "https://lists.linuxia.de"
  register: sympa_redirection
  failed_when: not sympa_redirection.redirected or sympa_redirection.url != "https://lists.linuxia.d
```

HTTP headers can be added as dict:

```
- name: Trigger list overview
  uri:
    url: "https://lists.linuxia.de/sympa/lists"
    method: "GET"
    headers:
      referer: "https://lists.linuxia.de/sympa"
```

user

The *user* module creates users on the target system.
Create an user:

```
- name: Create unix user for Sympa
  user:
    name: sympa
    group: sympa
    shell: /bin/bash
    createhome: yes
    password_lock: yes
```

xml

The *xml* module manipulates XML files and strings.

zypper

The *zypper* module manages packages on Suse distributions (OpenSUSE and SLES).
The syntax is similar to the *package* module.

zypper_repository

The *zypper_repository* module manages repositories on Suse distributions (OpenSUSE and SLES).

Roles

The location of the parent directory for the roles can be configured through the *roles_path* variable in the Ansible configuration file:

```
[defaults]
roles_path = roles
```

`ansible-galaxy` command

Install role from Github repository:

```
$ ansible-galaxy roles install git+https://github.com/racke/ansible-role-clamav.git
Starting galaxy role install process
- extracting ansible-role-clamav to /home/racke/provisioning/linuxia-inventory/roles/ansible-role-cl
- ansible-role-clamav was installed successfully
```

External Roles

Roles from Ansible Galaxy can be integrated as follows:

- Eintrag in `roles/requirements.yml` z.B.

```
- src: geerlingguy.memcached
  version: "1.0.8"
```

- Ansible Galaxy Installation:

```
 ansible-galaxy install -p roles -r roles/requirements.yml
```

- Git commit, e.g.:

```
git add roles/requirements.yml roles/geerlingguy.memcached
git commit -m "Add external role for installing memcached."
```

- Git tag, e.g.:

```
git tag -a -s -m "Role geerlingguy.memcached, version 1.0.8" ROLE_GEERLINGGUY_MEMCACHED_1_0_8
```

Upgrades

If you want to upgrade that role later, you need to bump up the version in `roles/requirements.yml` and rerun the `ansible-galaxy` command with the `-f` or `--force` flag:

```
ansible-galaxy install -f -p roles -r roles/requirements.yml
```

Wrapping external roles

It is quite useful to wrap external roles into your own role with the *import_ role* module. The following role demonstrates that:

1. Ensure that the default password has been overridden
2. Import the role with variables specific for your servers
3. Execute additional tasks (here we install backup package)

```
- name: Assert that default root password has been changed
  assert:
    that: "mysql_root_password != 'root'"
    fail_msg: 'Default value for mysql_root_password'

- name: Run external MySQL role
  import_role:
    name: geerlingguy.mysql
  vars:
    mysql_packages:
      - mariadb-server
      - mariadb-client

- name: Install automysqlbackup
  apt:
    name: automysqlbackup
```

You can also check in the wrapping role that the tasks in the imported role are not executed if a certain condition is met. For an example, it doesn't make sense to install *htpasswd* when you don't have credentials configured:

```
- name: Run external HTTP basic authentication role
  import_role:
    name: geerlingguy.htpasswd
  when:
   - htpasswd_credentials
```

Note: the role will be still imported, but all tasks are going to be skipped.

Load variables from a role

If you need the variables from a role without actually executing the role, you can use the following workaround:

```
roles:
  # including sympa role for defaults, but skipping it
  - role: sympa
    when: false
```

Monitoring with OMD

With this role we install a check_mk agent on the system, which runs as systemd service instead of as xinetd.

The agent is accessed remotely by SSH from the monitoring host, specified by the variables *omd_url* and *omd_version*.

`roles/monitoring/tasks/main.yml`

```
---

- name: Install check-mk-agent
  apt:
    deb: "{{ omd_url }}/check_mk/agents/check-mk-agent_{{ omd_version }}-1_all.deb"
  tags:
    - monitoring

- name: Enable and start systemd socket
  systemd:
    name: "check_mk.socket"
    enabled: yes
    state: started
  tags:
    - monitoring

- name: Add SSH key for monitoring
  authorized_key:
    user: root
    key: "{{ lookup('file', 'checkmk-ssh.pub') }}"
    key_options: 'command="/usr/bin/check_mk_agent"'
    state: present
```

Sample playbook:

```
---
- hosts: all

  vars:
    omd_version: "1.5.0p7"
    omd_url: "https://monitor.linuxia.de/omd"

  roles:
    - monitoring
```

Best Practices

- Use YAML callback plugin

Linuxia Wiki

Stefan Hornburg (Racke)
Ansible Provisioning

wiki.linuxia.de