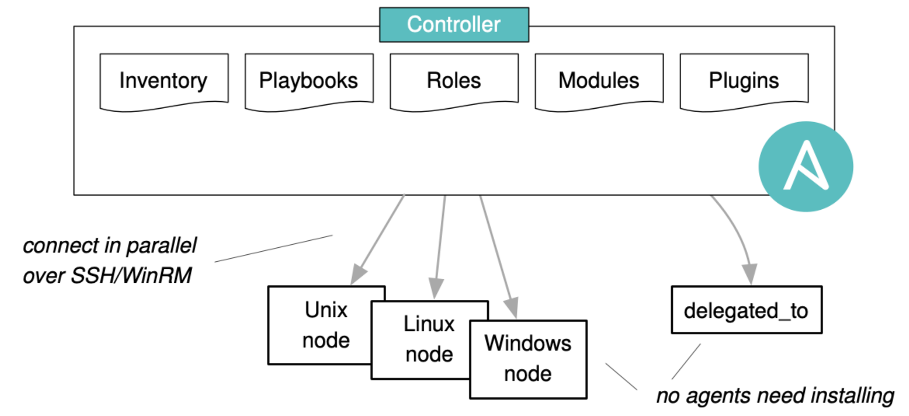
Ansible



Inventory

Ansible works against multiple hosts (also called nodes) in your infrastructure using a list or group of lists known as *inventory*. Inventories can be YAML or INI (shown here), a directory containing multiple inventory files, or dynamic in which case a program creates JSON which Ansible consumes.

```
mailhost    # ungrouped host

[web]
www01.example.com tz=Europe/Berlin ansible_port=222
www02 ansible_host=192.0.2.42 ansible_user=jane

[barcelona:children]
web

[web:vars]
ansible_user=ansible
port_no=8080
```

This inventory has an ungrouped node called `mailhost` and defines two groups, `barcelona` includes all hosts from the `web` group. The node `www02` is accessible at the specified address and requires a specific user to login. Ansible will login to all other `web` nodes as user `ansible`.

Once defined, you use inventory *patterns* to select the hosts or groups you want Ansible to run against. Specify `all` to run against all hosts in the inventory, a single host (`www12`) or group (`web`) to address just those, several hosts or groups (`alice,webservers`) for all of them (using colons instead of commas if you prefer). Exclude a host from a group (`web:!www01`) or target hosts with wildcards (`*.example.com`) if these patterns exist in the inventory. Create intersections (`web:&staging`) signifying any hosts in `web` which are also in the group `staging`, and finally use regular expressions if needed. Use `--limit` on the CLI to limit further.

Playbooks

Use playbooks (a.k.a. Ansible scripts) to execute tasks in order and more than once over multiple machines in parallel. Playbooks are text and well-suited to be placed under version control. They declare configurations and orchestrate steps on multiple machines in a defined order, and they can launch tasks synchronously or asynchronously. Playbooks consist of one or more plays (a list of plays) each of which target specific nodes or groups of nodes and execute a list of tasks.

Typically tasks use a module (Python code) to define the desired state and avoid changes if the state has been reached (idempotence). Whether you run the playbook once or many times the outcome should be identical.

Ansible offers four re-usable artifacts: variables files, task files, playbooks, and roles. Variables files contain only variables, tasks files only tasks, playbooks contain at least one play, and a role contains a set of related tasks, handlers, variables in a defined directory. Roles are easy to share and let you store tasks, variables, handlers, templates, and files in separate directories instead of in one long document.

Use one of `include_role`, `include_tasks`, or `include_vars` for dynamic (i.e. can use variable names in the filename) use, or `import_role` or `import_tasks` for static use. `include_*` statements can also loop.

Variables

A variable name can only include letters, numbers, and underscores and must not begin with a number. Python or playbook keywords are not valid variable names. Variables can be defined using YAML syntax and I recommend quoting strings:

```
username: "Jane {{ surname }}"
zip_code: 5601
```

Lists (also called arrays or sequences) can be written in square brackets or as an itemized list:

```
countries: [ "ES", "PT", "NO" ]
countries:
 - "ES"
 - "PT"
 - "NO" # 🇳🇴
```

The value of `countries[2]` is `"NO"` (lists are zero-based), but if the country code of Norway hadn't been quoted, its value would be `0` because `No` equates `False`!

Dictionary variables (hashes) store data in key/value pairs.

```
person:
  givenname: "Jane"
  surname: "Jolie"
```

Access specific fields from a dictionary by using the dot operator (`person.surname`) or the bracket (`person['surname']`) syntax.

Variables can be defined in the inventory (e.g. `tz`, `port_no`) or in `host_vars/` or `group_vars/` files – YAML files containing variable definitions, adjacent to either playbook or inventory. Variables can be included from files or specified in a play globally or per task, and variables can be prompted for and created from metadata returned from tasks with the `register` keyword:

```
- hosts: dbservers
  vars_files:
   - myvars.yml
  vars:
    username: "johnd"
  vars_prompt:
    - name: beverage
      prompt: What is your favorite beverage?
      private: no
      default: "h2o"
  tasks:
    - name: Print the value of this local variable
      debug: var=mylocal
      vars:
        mylocal: "task local"

    - name: Create a user and determine home directory
      user: name="{{ username }}"
      register: udata

    - debug: msg="Their home is at {{ udata.home }}"   #  => /home/johnd
```

In addition to the behavioral inventory parameters (variables beginning with `ansible_`) shown in *inventory* above, the following also exist (consider using Vault for sensitive data):

`ansible_connection`	connection type (`ssh`, `local`, `winrm`)
`ansible_password`	the SSH user's password
`ansible_ssh_private_key_file`	when not using SSH agent
`ansible_become_method`	privilege escalation method (`sudo`, `doas`, ...)
`ansible_become_user`	defaults to `root`
`ansible_become_password`	the password to use
`ansible_python_interpreter`	target host Python path

Block scalars

Use literal style, indicated by (`|`), to keep newlines inside a block of multiple lines, or have them replaced by spaces using folded style, indicated by (`>`). Add a dash (`-`) to strip trailing newlines.

```
- copy:
    content: |-
       Hey diddle diddle, the cat and the fiddle,
       The cow jumped over the moon.
    dest: /tmp/moo
  when: >
     some condition or
     another condition
```

Roles

The utility `ansible-galaxy role init myrole` assists in creating the correct directory structure for a role. The more common directories are:

`myrole/tasks/main.yml`	tasks file can `include` others
`myrole/handlers/main.yml`	handlers file
`myrole/files/`	`src` files for `copy`, `assemble`, etc.
`myrole/templates/`	`src` files for `template` module
`myrole/vars/main.yml`	variables associated with this role
`myrole/defaults/main.yml`	defaults for this role

Tasks / blocks

A playbook contains a list of tasks that are executed in order. We show the two most used syntaxes here. The more YAML-like second syntax is preferred.

```
- name: My first playbook
  hosts: www01,psql.example.com
  tasks:
    - name: Install a utility
      package: name="figlet" state=present

    - name: Copy a configuration file
      copy:
        src: myfile
        dest: /home/user/their/file
```

Blocks create logical groups of tasks which inherit directives (not loops) applied at the block level.

```
tasks:
  - block:
       -name: Copy a special file
        copy:
          src: myfile
          dest: thatfile
    rescue:
        .... variables: ansible_failed_task and ansible_failed_result
    always:
        .... more tasks
    when: datacenter == 'Paris'
    become: true
```

Handlers

Handlers run tasks on change, for instance, restarting a service after package upgrade.

```
tasks:
  - name: Ensure Mosquitto is at newest version
    dnf:
      name: mosquitto
      state: latest
    notify:
      - Restart Mosquitto

  - name: Install configuration
    template:
      src: mosquitto.j2
      dest: /etc/mosquitto/mosquitto.conf
    notify:
      - Restart Mosquitto
      - Alert admin

handlers:
  - name: Alert admin
    mail:
      subject: "Mosquitto config changed"

  - name: Restart Mosquitto
    service:
      name: mosquitto
      state: restarted
```

Handlers are run in the order that they are defined in the `handlers` section, not in the order they are notified, so if the configuration file changes in the above example, first an email is sent and only then is the Mosquitto service restarted. Handlers run once, irrespective of the number of times they were notified.

Loops

Ansible uses loops to execute tasks multiple times; beware that each task run conceptually means a new connection to the managed nodes. Constructs like `with_xxx` are also loops and use the lookup plugin type `xxx` to provide data for the loop.

```
- debug: msg="Hello {{ item }}"
  loop: [ "jane", "john" ]
```

Loop with a list of dicts (hashes):

```
- copy: src="{{ item.source }}" dest="{{ item.dest }}"
  loop:
    - { source: "/etc/passwd", dest: "/tmp/pw" }
    - { source: "exim.cf",     dest: "/etc/exim.cf" }
  loop_control:
    label: "{{ item.dest }}"
```

To limit the displayed output in a loop, use the `label` directive as above. (The default label is the full `item`.)

Variables `register`ed in loops will contain a `results` attribute which is a list of all responses from the module; this differs from not using loops. Avoid loops with the packaging modules which typically support using arrays to install packages.

Loops need lists, but `lookup()` produces lists only when `wantlist=True` is specified. Suggestion: use `query()` instead.

Conditionals

Ansible uses Jinja2 tests and filters in conditionals, e.g. based on facts:

```
- name: Reboot Debian flavored systems
  reboot: test_command=whoami
  when: ansible_os_family == "Debian"
```

Use `and` / `or` with parentheses to group expressions:

```
when: (variable == "aa" and other == "bb") or (number > 4)
```

Specify conditions as a list for a logical `and`:

```
when:
 - ansible_distribution == "MacOSX"
 - ansible_distribution_version is version_compare('10.15.7', '>=')
```

Filters

Jinja2 filters are bits of Python code which extend Jinja2 and run on the controller. They're used wherever a value is templated:

```
{{ servername | default('127.0.0.1') }}       127.0.0.1
{{ 59|random }} * * * * prog/in/cron          45 * * * * /prog/in/cron
{{ countries | join(" | ") }}                 ES | PT | NO
{{ "/etc/profile" | basename }}               profile
{{ "httpd.conf" | splitext }}                 ('httpd', '.conf')
{{ "hello world" | b64encode }}               aGVsbG8gd29ybGQ=
{{ "aGVsbG8gd29ybGQ=" | b64decode }}          hello world
{{ "ansible" | regex_replace('^.', 'A') }}    Ansible
```

Lookup plugins

Lookups extend Jinja2 to access data from outside sources within your playbooks, and they execute and are evaluated on the Ansible controller. Use the `file` lookup plugin to read a file:

```
file_contents: "{{ lookup('file', 'path/to/file') }}"
```

The first parameter to `lookup()` is the type of plugin, here "`file`". The other parameters are specific to the plugin. `file` expects the path to a file. The `ini` lookup reads from Windows INI style files, and the `csvfile` from CSV files. `fileglob` lists files matching shell expressions, `lines` reads lines from stdout of processes created on the controller. `password` generates a random password in a file or retrieves its content, `pipe` reads from a Unix pipe on the controller, and `url` returns content from a URL via HTTP or HTTPS. More standard plugins exist and custom lookups can be written to augment Ansible.

Variable substitution is performed *on use* in Ansible, so the following url lookup will actually cause two HTTP connections, one on each task.

```
vars:
  data = "{{lookup('url', 'http://localhost/')}}"
tasks:
  - debug: msg="{{ data.name }} {{ data.name }}"
  - debug: msg="{{ data.name }}"
```

Delegation

Tasks can be performed on a host with reference to others using delegation. If there are five hosts in the webservers group, both these tasks will run five times, the first once for each host in the group on `localhost` (the Ansible controller).

```
- hosts: webservers
  tasks:
  - name: Remove from monitoring during upgrade
    command: /opt/disable_alerts "{{inventory_hostname}}"
    delegate_to: localhost

  - name: Install required software on Web hosts
    apt: name="apache2" state=latest
```

Use the shorthand `local_action` if you prefer, with the following syntax for when more arguments are required:

```
- name: Submit syslog notification
  local_action:
    module: syslogger
    facility: "local0"
    priority: "info"
    msg: "setup for user done"
```

Environment

The remote environment can be set on play and on task level:

```
- hosts: brokers
  environment:
    username: "Jane"
    http_proxy: http://proxy.example.com:8080
  tasks:
    - name: Override global environment with "John"
      shell: echo $username
      environment:
        username: John
      register: out1    # out1.stdout  => John

    - name: Expect "Jane" as output
      shell: echo $username
      register: out2    # out2.stdout => Jane
```

CLI

Playbooks are run with the `ansible-playbook` utility, whereas ad hoc commands use `ansible` to automate a single task on one or more managed nodes.

```
$ ansible-playbook [playbook name]
$ ansible [inventory pattern] -m [module] -a '[module arguments]'
```

Both utilities understand these options:

`--user REMOTE_USER`	connect over SSH using this username `(-u)`
`--ask-pass`	prompt for the SSH user's password `(-k)`
`--become`	run operations as a privileged user `(-b)`
`--ask-become-pass`	ask for privilege escalation password `(-K)`
`--ask-vault-pass`	ask for vault password
`--extra-vars`	pass additional variables `(-e var=value), (-e @file.json)`
`--inventory INVENTORY`	specify path or comma separated host list `(-i)`
`--limit SUBSET`	further limit selected hosts to this pattern `(-l)`
`--list-hosts`	outputs a list of matching hosts without executing
`--list-tasks`	list all tasks that would be executed
`--start-at-task`	start the playbook at the task with this name
`--step`	one-step-at-a-time
`--tags TAG`	only run plays/tasks tagged with these values `(-t)`
`--verbose`	verbose mode (`-vvvv` for connection debugging)
`--version`	show program's version information

Documentation on installed modules and some types of plugins can be listed and a manual-page-type documentation shown with `ansible-doc` which has these options amongst others:

`-l`	list available plugins
`-t TYPE`	chose plugin type `module` (default), `lookup`, `callback`, `inventory,` `become`, `cache`, `connection`, `keyword`
`PLUGIN`	show documentation for this plugin

Collections

Collections are a namespaced distribution format for Ansible content which can include roles, modules, plugins, and playbooks. Installed with `ansible-galaxy collection install`, refer to their content in plays by their fully qualified collection name (FQCN) or their short name if the collection is defined in the play. Plugins (e.g. lookups or filters) in collections require FQCN.

```
- hosts: alice
  collections:
    - ansilab.demo
  tasks:
    - debug: msg="hello"

    - ansilab.demo.welcome:
      register: out

    - welcome:
      register: out
```

Run playbooks from a collection's `playbooks/` directory with an FQCN (`ansilab.demo.pb`)

Vault

Ansible Vault uses AES encryption to protect sensitive content. Once decrypted, play authors are responsible for avoiding secret disclosure. The `ansible-vault` CLI can `create`, `decrypt`, `edit`, `view`, `encrypt`, and `rekey` a file, opening `$EDITOR` to edit content if needed. Add individually encrypted strings to any `vars` file, copying the output of

```
$ ansible-vault encrypt_string "secret" --name "myvar"
```

Then run `ansible` or `ansible-playbook` with `--ask-vault-pass` to be prompted for vault password for on-the-fly decryption or specify a password in a file and use `--vault-password-file=FILENAME` for automating. Use vault IDs (see documentation) if you require multiple passwords.

Windows

Configure *Windows Remote Management* (`C:\>winrm quickconfig`), install PowerShell, create admin user. On Linux/Unix, `pip install pywinrm requests-ntlm`, and add variables to, say, inventory, for the Windows nodes:

```
[win:vars]
ansible_user=jjolie
ansible_password=sup3rs1kr3t
ansible_connection=winrm
ansible_port=5985
ansible_winrm_server_cert_validation=ignore
ansible_winrm_transport=ntlm
```

ansible.cfg

Configuration settings are read from the first found file in `$ANSIBLE_CONFIG`, `./ansible.cfg`, `~/.ansible.cfg`, or `/etc/ansible/ansible.cfg`. A few example settings:

```
[defaults]
nocows = 1
inventory = /path/to/inventory
vault_password_file = /path/to/vault/password
ansible_managed = This file is managed by Ansible on {host}
```

Configure individual settings in the environment (e.g. `export ANSIBLE_NOCOWS=1`).

~/.vimrc

```
filetype plugin on
autocmd FileType yaml setlocal ts=3 sts=3 sw=3 expandtab
```

Join the Ansible Community at
ansible.com/community