Ansible tutorial

EuroBSDCon
created by Jan-Piet Mens
@jpmens

Ansible

- Created by Michael DeHaan
- Automation
- Configuration management and deployment
- Orchestration
- Rolling updates
- Ad-hoc tasks

Goals

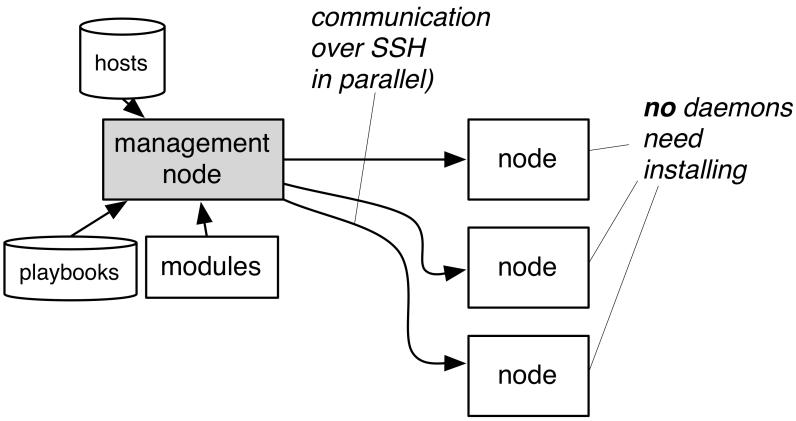
- Simplicity and ease of use
- Reliability
- Readability (YAML Ain't Markup Language)
- Minimal dependencies

"Automation SHOULD [RFC 2119] not require programming experience"

More than just shell scripts

- Idempotence
- No daemons, no agents on nodes
- Not another PKI
- No special "management" server
- No additional open firewall ports
- Push-based; pull is possible

How Ansible works



No root (maybe)

- Ansible doesn't require root access on the nodes, but some modules do
- Ansible can login as any user and escalate privileges (`sudo`, `doas`, `su`, ...)
- SSH keys (SSH agent)

What we want is password-less logins:

```
$ ssh -l ansible web doas id
uid=0(root) gid=0(wheel) groups=0(wheel), 2(kmem), 3(sys) ...
```

Installing Ansible: Requirements

Control machine

- Python
- OpenBSD, FreeBSD, NetBSD, *BSD, RHEL, CentOS, Debian, OS/X, macOS, etc.

some modules/plugins have additional requirements

Nodes

- Unix/Linux with Python 2.6+ or 3.5+, and SSH / SFTP (SCP possible)
- Windows with WinRM and Powershell 3

Installing Ansible

From source

```
$ pip install paramiko PyYAML Jinja2 httplib2 six
$ git clone git://github.com/ansible/ansible.git --recursive
$ cd ./ansible
$ source ./hacking/env-setup
```

Via package manager

```
# pkg install py36-ansible26
# pkg_add ansible
# yum install ansible   # requires epel-release
# apt-get install ansible
```

Via Pip

```
$ python3 -mvenv ansib
$ source ansib/bin/activate
(ansib) $ pip install ansible
```

Configuration file: `ansible.cfg`

- Know about it; Tune / change defaults
- Disable annoying features ;-)

```
 _____________________________________
< PLAY [JP doesn't much like cows] >
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
[defaults]
nocows = 1
```

```
$ export ANSIBLE_NOCOWS=true
```

`ansible.cfg` precedence

These are searched for in the following order; first found wins (no merge):

- `$ANSIBLE_CONFIG`
- `./ansible.cfg`
- `~/.ansible.cfg`
- `/etc/ansible/ansible.cfg`

Modules

Modules

- Modules do the actual work in Ansible
- Modules are executed in playbook tasks, and can be run ad-hoc
- Many modules are idempotent

```
$ ansible dbservers -m ping
$ ansible alice -m copy -a 'src=/etc/passwd dest=/tmp/test mode=0400'
```

Modules in playbooks

Within a playbook, modules are executed similarly (choose a syntax):

```
- name: Copy some file somewhere else
  copy: src=/etc/passwd dest=/tmp/pw mode=0400

- name: Copy some file somewhere else
  copy: src=/etc/passwd
        dest=/tmp/pw
        mode=0400

- name: Copy some file somewhere else
  copy:
    src: "/etc/passwd"
    dest: "/tmp/pw"
    mode: 0400
```

Built-in documentation

Modules contain documentation:

```
$ ansible-doc -l
apt               Manages apt-packages
apt_key           Add or remove an apt key
```


```
$ ansible-doc copy
> COPY

  The [copy] module copies a file on the local box to remote locations. Use the
  [fetch] module to copy files from remote locations to the local box. If you
  need variable interpolation in copied files, use the [template] module.

Options (= is mandatory):

- backup
        Create a backup file including the timestamp information so you can get
        the original file back if you somehow clobbered it incorrectly.
        (Choices: yes, no) [Default: no]

...
```

Inventory

Inventory

- Defaults to `/etc/ansible/hosts` in INI-format
- (don't confuse with `/etc/hosts`)
- Possible to override default in `ansible.cfg` (`inventory =`)
- Can be executable, have multiple sources / directory

```
[web]
www01
www02 tz=Europe/Berlin

[dbservers]
psql.example.com
p2 ansible_host=192.0.2.42 ansible_port=333 ansible_user=jane

[dbservers:vars]
spooldir=/var
logging=yes

[openbsd:vars]
ansible_become_method=doas
```

Inventory patterns

How to specify inventory hosts when using Ansible utilities:

What	Example
A host or group	`myhost`, `webservers`
All hosts	`all` or `*`
Intersection (all Web in staging)	`staging:&webservers`
Exclusion	`webservers:!www01`
Wildcard	`*.example.net`
Range numbered	`www[5:10]` and `www[05:10]`
Regexp	`~www\d\.example\.net`

Hostvars & Groupvars

Variables for a particular host in their own file (`host_vars/www02.yaml`):

```
---
database_port: 3106
ansible_user: jane
ansible_port: 444
```

Likewise for a group of hosts (`group_vars/freebsdmachines.yml`):

```
---
spooldir: /var
logging: yes
ansible_python_interpreter: /usr/local/bin/python2.7
```

Variables

Variables can be defined in a play:

```
- hosts: dbservers
  vars:
    spooldir: /var
    logging: true
```

Variables can also be read from vars files:

```
- hosts: localhost
  vars_files:
    - myvars.yml
```

Registered variables

```
- shell: /bin/pwd; echo Oops >&2;  whoami; echo "Hello"
  register: out

- debug: var=out
```

```
TASK [debug] *******************************************
ok: [localhost] => {
    "out": {
        "changed": true,
        "cmd": "/bin/pwd; echo Oops >&2;  whoami; echo \"Hello\"",
        "delta": "0:00:00.021167",
        "end": "2018-11-13 13:08:38.940511",
        "rc": 0,
        "start": "2018-11-13 13:08:38.919344",
        "stderr": "Oops",
        "stdout": "/private/tmp\njpm\nHello",
        "stdout_lines": [
            "/private/tmp",
            "jpm",
            "Hello"
        ]
    }
}
```

Variables from the command-line

Pay attention to white space and quoting:

```
$ ansible alice -e userid="Jane" -m debug -a 'msg="Hello {{userid}}"'
alice | SUCCESS => {
   "msg": "Hello Jane"
}
```

Complex variables can also be passed as an option:

```
--extra-vars '{ "userid" : "Jane", "numbers" : [1, 2, 3] }'
--extra-vars @file.json  --extra-vars @file.yml
```

Information about other hosts

`hostvars`, `group_names`, `groups` are maintained by Ansible.

`{{ hostvars['www02']['ansible_distribution'] }}`

```
{% for host in groups['webservers'] %}
   {{ hostvars[host]['ansible_default_ipv4']['address'] }}
{% endfor %}
```

Variable precedence

From lower to higher:

- role defaults
- inventory vars
- inventory group_vars
- inventory host_vars
- playbook group_vars
- playbook host_vars
- host facts
- play vars
- play vars_prompt
- play vars_files
- registered vars
- set_facts
- role and include vars
- block vars (only for tasks in block)
- extra vars (always win precedence)

Lab

- Create an inventory file `hosts`

```
localhost conf=EuroBSDcon location=Lillehammer

[all:vars]
ansible_connection=local
; ansible_python_interpreter=/usr/local/bin/python3 ; openbsd
ansible_python_interpreter=/usr/local/bin/python3.6 ; freebsd
```

- Run an ad-hoc ping on localhost

```
$ export ANSIBLE_INVENTORY=./hosts
$ ansible localhost -m ping
```

Playbooks

YAML

- Start of document `---`, Comments start with `#`
- Booleans: `True`, `Yes`, `On`, Strings: `hello world` vs. `"hello world"`
- Lists

```
- uno
- dos
```

- Dictionaries

```
name: Jane Jolie
locality: Paris
country: France
```


Dave Anderson
@dave_universetf
Anyway, after much research into workplace safety signs, here's something I made.
WARNING
Raw YAML.
Meaning may change suddenly.
Check whitespace before pushing.
346 5:24 AM - Sep 14, 2019

Playbooks

- A playbook is an Ansible configuration management recipe
- It contains a list of plays and is written in YAML
- YAML (YAML Ain't Markup Language) <-> JSON
- A play must have a set of hosts to configure and a list of tasks.
- A task is the most granular thing you do:
    - install an authorized key
    - copy a file
    - create a user
- Tasks are linear
- Tasks can loop
- The `name` attributes are optional but highly recommended

Playbooks

```
---
- name: Deploy fortune generator
  hosts: alice
  become: true
  tasks:
  - name: Ensure pip package is available
    package: name=python-pip state=present

  - name: Ensure required Python modules are available
    pip: name="{{ item }}"
    loop:
           - paho-mqtt
           - fortune

  - name: Ensure fortune data file is installed
    copy: src=../files/fortunes/fortunes dest=/etc/fortunes mode=0444

  - name: Initialize binary fortunes if not yet done
    command: fortune -u /etc/fortunes creates=/etc/fortunes.dat
```

Multiple plays

Playbooks can contain more than one play:

```
- name: Play1 deploy to Web servers
  hosts: webservers
  tasks:
  - yum: ...

- name: Play2 add to monitoring
  hosts: monitoringhosts
  tasks:
  - uri: ...
```

Hosts and users (1)

Each play describes the hosts it should target and as which user it should do so

```
---
- hosts:
    - webservers
    - dbs
  remote_user: ansible
```

Remote users can also be defined per task:

```
---
- hosts: webservers
  remote_user: root
  tasks:
   - name: Test connection
     ping:
     remote_user: jane
```

Hosts and users (2)

Run modules as another user (sudo, su, pbrun, pfexec, doas, dzdo, ksu)

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
```

You can also use become just for a particular task:

```
- hosts: webservers
  remote_user: yourname
  tasks:
    - service: name=httpd state=started
      become: yes
      become_method: sudo
      become_user: root
```

Use --ask-become-pass on the CLI if necessary

Handlers

```
- hosts: all
  become: yes
  tasks:
  - name: Install nginx
    yum: name=nginx state=latest
    notify:
     - kick_nginx

  - name: Configure nginx
    template: src=nginx.j2 dest=/etc/nginx/nginx.conf
    notify:
     - kick_nginx

  handlers:
  - name: kick_nginx
    service: name=nginx state=restarted
```

Blocks for logical grouping of tasks

```
tasks:
- block:
     - package: name={{ item }} state=present
       loop:
         - httpd
         - memcached

     - template: src=templates/src.j2 dest=/etc/foo.conf

     - service: name=example state=started enabled=True

  when: ansible_distribution == 'CentOS'
  become: true
  become_user: root
```

Tags

Plays and tasks supports a "tags:" attribute:

```
- openbsd_pkg:
     name: [ mosquitto, rsync, ttyd, jo ]
     state: present
  tags:
    - pkgs

- copy:
     src: mosquitto.conf
     dest: /etc/mosquitto/mosquitto.conf
  tags:
    - config
```

Run only certain parts of a long playbook:

```
$ ansible-playbook long.yml --tags "config"

$ ansible-playbook long.yml --skip-tags "config nginx"
```

Conditionals

When

```
tasks:
  - name: "shut down Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
```

```
when: (v == "CentOS" and major == "6") or
      (v == "Debian" and major == "7")

when:
      - ansible_distribution == "CentOS"
      - ansible_distribution_major_version == "6"

when: number > 4
```

Loops with with and with loop (1)

```
- group: name="{{ item }}" state=present
  loop:
     - "ops"
     - "dev"
```

```
- user: name="{{ item }}" shell="/bin/zsh" create_home=true
  with_lines:
     - "cut -d: -f1 /etc/passwd"
```

```
 - copy: src="{{ item }}" dest="d/"
   with_fileglob:
    - /etc/p*
```

Loops with with and with loop (2)

```
- copy: src={{ item.src }} dest={{ item.dest }}
  loop:
      - { src: httpd.conf, dest: /etc/httpd.conf }
      - { src: master.cf,  dest: /etc/postfix/master.cf }
```

Facts and information

Facts

Facts are bits of information obtained by speaking with your remote nodes. For example IPv4 address, configured swap space, etc.

```
$ ansible fox -m setup
"ansible_facts": {
    "ansible_all_ipv4_addresses": [
        "10.0.2.15",
        "192.168.56.118"
    ],
    "ansible_all_ipv6_addresses": [],
    "ansible_apparmor": {
        "status": "disabled"
    },
    "ansible_architecture": "amd64",
    "ansible_bios_date": "NA",
    "ansible_bios_version": "NA",
    "ansible_date_time": {
        "date": "2019-08-14",
        "day": "14",
        "epoch": "1565788541",
        "hour": "13",
        "iso8601": "2019-08-14T13:15:41Z",
        "iso8601_basic": "20190814T131541158042",
        "iso8601_basic_short": "20190814T131541",
```

Facts: networking

IPv4 and IPv6 addresses, interfaces


```
"ansible_default_ipv4": {
  "address": "10.0.2.15",
  "broadcast": "10.0.2.255",
  "device": "em0",
  "flags": [
      "UP",
      "BROADCAST",
      "RUNNING",
      "SIMPLEX",
      "MULTICAST"
  ],
  "gateway": "10.0.2.2",
  "interface": "em0",
  "macaddress": "08:00:27:a9:28:81",
  "media": "Ethernet",
  "media_options": [
      "full-duplex"
  ],
  "media_select": "autoselect",
  "media_type": "1000baseT",
  "metric": "0",
  "mtu": "1500",
  "netmask": "255.255.255.0",
  "network": "10.0.2.0",
  "options": [
      "PERFORMNUD",
      "IFDISABLED",
      "AUTO_LINKLOCAL"
  ],
  "status": "active",
  "type": "ether"
},
```

Facts: sundry


```
"ansible_distribution": "FreeBSD",
"ansible_distribution_major_version": "12",
"ansible_distribution_release": "12.0-RELEASE",
"ansible_distribution_version": "12.0",
"ansible_dns": {
    "nameservers": [
        "192.168.1.82",
        "192.168.1.113"
    ],
    "search": [
        "ww.mens.de"
    ]
},
"ansible_os_family": "FreeBSD",
"ansible_pkg_mgr": "pkgng",
```

Using facts

- IP address: `{{ ansible_default_ipv4.address }}`
- Hostname as reported by system: `{{ ansible_nodename }}`
- Fully qualified name: `{{ ansible_fqdn }}`
- Configured swap size: `{{ ansible_swaptotal_mb }}`

Disable facts

```
- hosts: webservers
  gather_facts: no
```

Fact caching

Ansible can cache facts in Redis or JSON files.

```
[defaults]
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/factcache
# fact_caching = redis
# fact_caching_connection = localhost:6379:0
fact_caching_timeout = 3600
```

Caching in Redis requires the Python `redis` library be installed via pip.

Gathering

- *smart* – gather by default, but don't regather if already cached
- *implicit* – gather by default; disable with `gather_facts: False`
- *explicit* – do not gather by default; must say `gather_facts: True`

Fact caching example

```
$ ansible-playbook t.yml
$ jq -r .ansible_date_time.iso8601 < /tmp/factcache/localhost
2018-10-29T15:01:01Z

$ ansible-playbook t.yml
$ jq -r .ansible_date_time.iso8601 < /tmp/factcache/localhost
2018-10-29T15:01:01Z

$ rm /tmp/factcache/localhost
$ ansible-playbook t.yml
$ jq -r .ansible_date_time.iso8601 < /tmp/factcache/localhost
2018-10-29T15:03:42Z
```

Local facts (`facts.d`)

- Obtained from remote `/etc/ansible/facts.d`
- Files ending in `.fact` (JSON, INI, or executables emitting JSON)

Example:

```
$ cat /etc/ansible/facts.d/beverage.fact
[favorite]
tonic=yes
```

Fact gathering

```
"ansible_local": {
    "beverage": {
        "favorite": {
            "tonic": "yes"
        }
    }
},
"ansible_machine": "x86_64",
...
```

In a template or playbook: `{{ ansible_local.beverage.favorite.tonic }}`

Local facts: executable

```
$ cat /etc/ansible/facts.d/serial.fact
#!/bin/sh

jo currybeer=true dish=vindaloo epoch=$(date +%s)
```

```
{"currybeer":true,"dish":"vindaloo","epoch":1512401355}
```

```
$ ansible alice -m setup -a filter=ansible_local
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "serial": {
                "currybeer": true,
                "dish": "vindaloo",
                "epoch": 1512401355
            }
        }
    }
}
```

Lab

- Create a playbook `fact.yml` with which you print out, using `debug`, the operating system of your workstation

```
- name: My First Playbook
  hosts: localhost
  tasks:
  - name: Which OS am I on?
    debug: var=ansible_distribution
```

- Run the playbook

```
$ export ANSIBLE_INVENTORY=./hosts
$ ansible-playbook fact.yml
PLAY [My First Playbook] ******************

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

TASK [Which OS am I on?] ******************
ok: [localhost] => {
    "ansible_distribution": "MacOSX"
}
```

{{ Templates }}

{{ Templates }}

- Templates are processed by Jinja2 (on the control machine)

```
- name: Template out httpd.conf for Apache
  template: src=httpd.conf.in dest=/etc/httpd/httpd.conf
```

```
# Listen on addr

Listen {{ http_port }}

# Name of user
User www

ServerAdmin {{ admin_email }}
```

```
# Listen on addr

Listen 80

# Name of user
User www

ServerAdmin jp@mens.de
```

{% if %}

```
# sshd_config

{# set SFTP binary
   depending on distro #}

{% if ansible_distribution == "OpenBSD" %}
Subsystem       sftp    /usr/libexec/sftp-server
{% else %}
Subsystem       sftp    /usr/lib64/ssh/sftp-server
{% endif %}
```

Managing users (1)

```
vars_files:
  - users.yml
tasks:
- name: Add users to system
  user: name={{ item.username }} shell=/bin/bash createhome=yes
  loop: '{{ users }}'

- name: Create authorized keys
  authorized_key:
       user={{ item.username }}
       key="{{ lookup('file', 'pubkeys/{{ item.username}}.pub') }}"
  loop: '{{ users }}'

- name: Template sudoers
  template: src=sudoers.in dest=./output.txt
```

users.yml

```
users:
- username: jane
  sudo: true
- username: john
  sudo: false
```

Managing users (2)

sudoers.in

```
# Individual users

{% if users is iterable %}
{%    for u in users %}
{%       if u.sudo == True %}
{{ u.username }} ALL=(ALL) NOPASSWD: ALL
{%       endif %}
{%    endfor %}
{% endif %}
```

or

```
User_Alias      ELECTRIC={{ users | join(',', attribute='username') }}
```

sudoers

```
# Individual users

jane ALL=(ALL) NOPASSWD: ALL
```

Validation of file content is important!


```
- hosts: bindservers
  user: root
  vars:
    checkconf: /usr/sbin/named-checkconf
  tasks:
  - name: Install and validate named.conf
    template:
        src=named.conf.j2
        dest=/etc/named.conf
        validate='{{ checkconf }} %s'
```

Filters

- Filters transform template expressions (think Unix pipe)
- Jinja2 has many built-in filters; Ansible adds more
- Filters can be used within `{{ }}` in playbooks

```
{{ dbserver | default('127.0.0.1') }}
```

```
- name: Checkout repository
  git: dest="{{ target_dir }}" repo="{{ url }}"
  when: do_checkout|default('')
```

Filter examples

```
{% set mylist = [ "one", "two", "three" ] -%}

{{ 'secret' | password_hash('sha256') }}    $5$rounds=535000...vM0HGLdrmZ1
{{ [3, 2, 5] | min }}                       2
{{ 59|random }} * * * * /program/in/cron    14 * * * * /program/in/cron
{{ '192.0.2.1/24' | ipaddr('address') }}    192.0.2.1
{{ 'test1'|hash('md5') }}                   5a105e8b9d40e1329780d62ea2265d8a
{{ mylist | join(" | ") }}                  one | two | three
{{ "/etc/profile" | basename }}             profile
{{ "/etc/profile" | dirname }}              /etc
{{ "~jpm" | expanduser }}                   /Users/jpm
{{ "httpd.conf" | splitext }}               ('httpd', '.conf')
{{ "hello world" | b64encode }}             aGVsbG8gd29ybGQ=
{{ "aGVsbG8gd29ybGQ=" | b64decode }}        hello world
{{ "ansible" | regex_replace('a', 'A') }}   Ansible
```

Creating a custom filter

```
{{ "Hello world" | stars() }}
```

```
*** Hello world ***
```

```
def star_this(var):
    return "*** %s ***" % (var)

class FilterModule(object):
    def filters(self):
        return { 'stars' : star_this }
```

Install the filter file in `filter_plugins/stars.py`

$LOOKUP

- Access data from foreign sources
- Evaluated on control machine
- Results available in templating engine
    - file
    - password
    - csv, ini
    - credstash (AWS's KMS and DynamoDB)
    - DNS (`dig`)
    - env
    - passwordstore
    - pipe
    - redis, mongodb
    - template
    - shelvefile
    - etcd
    - url

file lookup

```
- hosts: localhost
  vars:
    - sentence: "{{ lookup('file', 'data') }}"
  tasks:
  - name: Show data as read via file lookup
    debug: var=sentence
```

```
ok: [localhost] => {
    "sentence": "The quick brown fox\njumps over the lazy dog."
}
```

CSV lookup

```
Code,Country
NL,Netherlands
DE,Germany
ES,Spain
FR,France
```

```
- hosts: localhost
  vars:
  - cc: FR
  - s: "{{ lookup('csvfile', cc + ' file=europe.csv delimiter=,') }}"
  tasks:
  - debug: msg="country code {{ cc }} == {{ s }}"
```

```
ok: [localhost] => {
    "msg": "country code FR == France"
}
```

INI lookup

```
[params]
curry = Vindaloo
drink = lassi
```

```
- hosts: localhost
  vars:
    - food: "{{ lookup('ini', 'curry section=params file=info.ini') }}"
  tasks:
    - debug: msg="a favorite dish is {{ food }}"
```

```
ok: [localhost] => {
    "msg": "a favorite dish is Vindaloo"
}
```

password lookup (1)

The *password* lookup generates a random password and stores it in a file

```
- hosts: localhost
  vars:
     password: "{{ lookup('password', 'pw.file length=20 chars=xxx') }}"
  tasks:
  - debug: msg="{{ password }}"
```

Output:

```
ok: [localhost] => {
    "msg": "Iq_arENe,1-Rt069hCGt"
}

$ cat pw.file
Iq_arENe,1-Rt069hCGt
```

password lookup (2)

```
ascii_letters='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
ascii_lowercase='abcdefghijklmnopqrstuvwxyz'
ascii_uppercase='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
digits='0123456789'
hexdigits='0123456789abcdefABCDEF'
letters='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
lowercase='abcdefghijklmnopqrstuvwxyz'
octdigits='01234567'
printable='0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\
          "#$%&()*+,-./:;<=>?@[\]^_`{|}~ \t\n\r\x0b\x0c'
punctuation='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~'
uppercase='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
whitespace='\t\n\x0b\x0c\r '
```

dns lookup

The *dig* lookup performs all sorts of DNS queries; it can return flat results or lists:

```
- debug: msg="IPv4 of example.net. is {{ lookup('dig', 'example.net.')}}"
- debug: msg="NL is for {{ lookup('dig', 'nl.cc.jpmens.net/TXT')}}"
```

Output:

```
ok: [localhost] => {
    "msg": "IPv4 of example.net. is 192.0.2.34"
}

ok: [localhost] => {
    "msg": "NL is for NETHERLANDS"
}
```

custom lookup plugins (1)

```
vars:
  userlist: [ jjolie, ev00 ]
tasks:
- name: Create user if required
  user:
      name: "{{ item }}"
      comment: "{{ lookup('ldapget', item) }}"
      home: "/home/{{ item }}"
      create_home: true
      password: "{{ '123456' | password_hash('sha512') }}"
      update_password: on_create
  loop: "{{ userlist }}"
```

```
$ tail -2 /etc/passwd
jjolie:x:1003:1003:Jane Jolie:/home/jjolie:/bin/bash
ev00:x:1004:1004:E. Valleri:/home/ev00:/bin/bash
```

custom lookup plugins (2)

```
from ansible.plugins.lookup import LookupBase
import ldap

class LookupModule(LookupBase):
    def run(self, terms, variables, **kwargs):

        ld = ldap.initialize("ldap://192.168.33.101")
        ld.bind_s(None, None)

        ret = []
        for term in terms:
            filter = "(uid={uid})".format(uid=term)
            cn = "unknown"
            res = ld.search_s("dc=example,dc=net",
                              ldap.SCOPE_SUBTREE, filter, None)
            if len(res) > 0:
                dn, entry = res[0]
                if "cn" in entry:
                    cn = entry["cn"][0].decode("utf-8")
            ret.append(cn)
        return ret
```

Install the filter file in `lookup_plugins/ldapget.py`.

Lab

- Create a template source with the following content:

```
I'm currently at the {{ conf }} in {{ location }}.
```

- Create a playbook `conference.yml` with which you template out that template to a file on your local workstation

```
- name: Testing templates
  hosts: localhost
  gather_facts: false
  tasks:
  - name: Template out a greeting
    template: src=in.put dest=/tmp/out.put mode="0444"
```

- Run the playbook and verify content and permissions of the target file.

Packaging

Module categories

Cloud, Clustering, Commands, Database, Files, Identity, Inventory, Messaging, Monitoring, Network, Notification, Packaging, Remote Management, Source Control, Storage, System, Univention, Utilities, Web Infrastructure, Windows

Why yum, apt, zypper, ... ?

Looking at Packaging, we see *apk*, *apt*, *dnf*, *homebrew*, *layman*, *macports*, *openbsd_pkg*, *opkg*, *package*, *pacman*, *pkg5*, *pkgin*, *pkgng*, *pkgutil*, *portage*, *portinstall*, *slackpkg*, *svr4pkg*, *urpmi*, *yum*, *zypper*.

- No additional abstraction required
- Use existing knowhow for your infrastructure
- Playbooks can be written to use actions dependent on node OS
- The `package` module uses the OS package manager

> ⚠ The `package` module doesn't translate package *names*

Code reuse

Import Playbooks

- Includes a file with a list of plays
- Can only be included at the top level; you cannot use this action inside a play.

```
---
- import_playbook: one.yml
- import_playbook: two.yml
```

Import / include tasks

```
vars:
  username: jane
  keys:
    - "{{ lookup('file', 'xxx') }}"
tasks:
- include_tasks: something.yml
- copy: src=file.in dest=file.out
- import_tasks: db.yml dbtype=psql
```

- Includes are dynamic which allows loops etc. as well as constructs like `include_tasks: "{{ hostname }}.yml"`

Roles (1)

- Modularization of playbooks
- Simplification, reuse and share
- Roles have name (`webserver`) and associated files

Element	Directory
Tasks	`roles/webserver/tasks/main.yml`
Files	`roles/webserver/files/`
Templates	`roles/webserver/templates/`
Handlers	`roles/webserver/handlers/main.yml`
Variables	`roles/webserver/vars/main.yml`
Defaults	`roles/webserver/defaults/main.yml`
Dependency info	`roles/webserver/meta/main.yml`
Tests	`roles/webserver/tests/{inventory,test.yml}`
Library	`roles/webserver/library/`

```
[defaults]
roles_path = /etc/ansible/roles
```

Roles (2)

Invoking roles

```
- hosts: webservers
  roles:
    - ntp
    - webserver
    - { role: appli, dir: "/var/apps/1", port: 6000 }
```

Roles (3)

We can instruct Ansible to run certain tasks before and/or after roles

```
- hosts: webservers
  pre_tasks:
    - name: Disable the frobnicator
      command: frob --disable

  roles:
    - ntp

  post_tasks:
    - name: Startup the frobnicator
      command: frob --launch
```

Role example (1)

```
$ tree
.
├── roles
│   └── tiny
│       ├── tasks
│       │   └── main.yml
│       ├── templates
│       │   └── hosts.in
│       └── vars
│           └── main.yml
└── roletest.yml
```

Role example (2)

roles/tiny/tasks/main.yml

```
- name: Install hosts from template
  template: src=hosts.in dest={{ directory }}/hosts mode=0444
```

roles/tiny/templates/hosts.in

```
127.0.0.1               localhost
{{ ansible_default_ipv4.address }}  box
```

roles/tiny/vars/main.yml

```
directory: /tmp
```

roletest.yml

```
- hosts: localhost
  roles:
    - tiny
```

Boilerplate roles

```
$ ansible-galaxy init rolename

$ tree
.
└── rolename
    ├── README.md
    ├── defaults
    │   └── main.yml
    ├── handlers
    │   └── main.yml
    ├── meta
    │   └── main.yml
    ├── tasks
    │   └── main.yml
    ├── tests
    │   ├── inventory
    │   └── test.yml
    └── vars
        └── main.yml
```

Ansible Galaxy

- Browse and search for roles at https://galaxy.ansible.com.

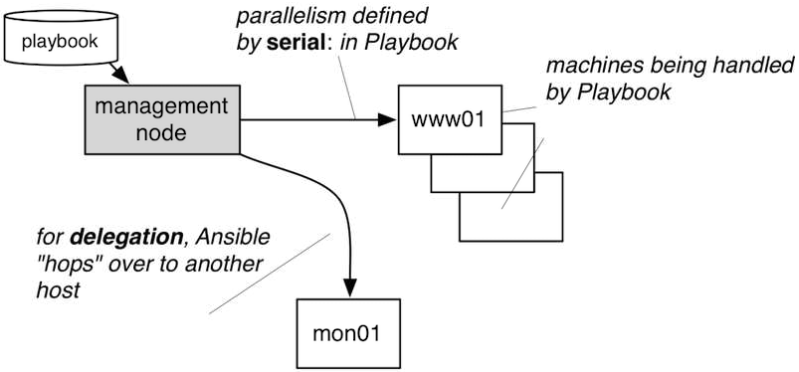
```
$ ansible-galaxy install username.rolename
```

```
$ ansible-galaxy install geerlingguy.apache \
    geerlingguy.mysql geerlingguy.php
```

```
---
- hosts: lampservers
  roles:
    - geerlingguy.mysql
    - geerlingguy.apache
    - geerlingguy.php
```

Delegation

Delegation




```
- hosts: www[01:10]
  tasks:
  - package: name=httpd state=latest

  - copy: src=httpd.conf dest=/etc/somewhere/httpd.conf

  - uri: url="http://{{ monitor }}/create?host={{ inventory_hostname }}"
    delegate_to: mon01
```

Local action

Basically a `delegate_to: localhost`

```
- name: Run the migration process
  local_action: command /usr/bin/migrate_it
```

Run play locally:

```
- hosts: 127.0.0.1
  connection: local
```

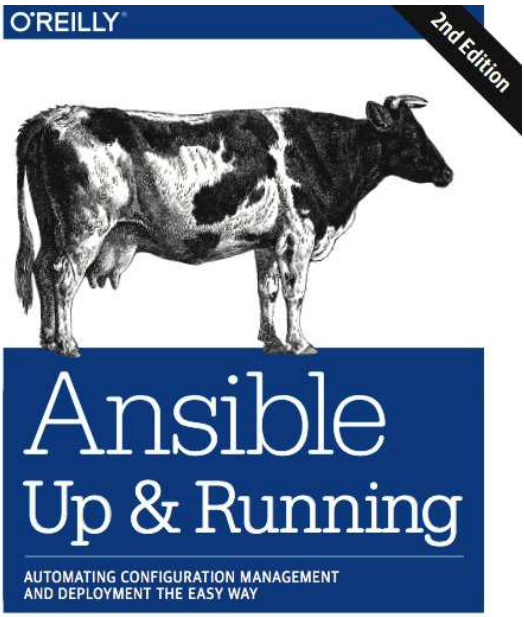
```
$ ansible-playbook playbook.yml --connection=local
```

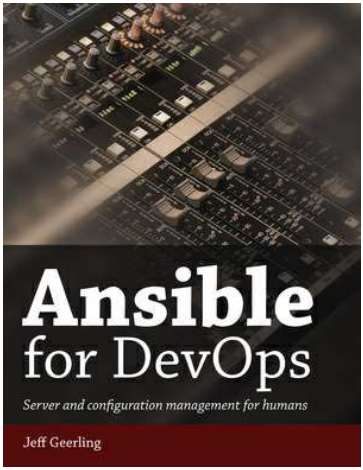
Further study

Ansible features to study

- Pull mode
- Vault
- Creating custom modules
- Module facts

Books


O'REILLY
2nd Edition
Ansible
Up & Running
AUTOMATING CONFIGURATION MANAGEMENT
AND DEPLOYMENT THE EASY WAY
Lorin Hochstein & René Moser


Ansible
for DevOps
Server and configuration management for humans
Jeff Geerling

Dynamic inventory

Inventory scripts

- Inventory scripts
  - Cobbler, Foreman, Spacewalk
  - AWS EC2, Azure, Cloudstack, Consul, OpenVZ
  - OpenStack, docker, VMware
  - Digital Ocean, Linode, ...
  - Nagios, Zabbix
  - https://github.com/ansible/ansible/tree/devel/contrib/inventory
- Build your own
  - LDAP
  - CMDB
  - ...

Your own dynamic inventory

- Program in any language
- produce JSON

Configuration:

```
$ grep inventory ansible.cfg
inventory = ./invent.sh
```

Scripts can be invoked multiple times in one Ansible run:

```
--list

--host host1
--host host2
--host alice
--host bob
--host db1
```

Dynamic inventory with `_meta`

```
{
  "webservers": {
     "hosts": [ "127.0.0.1", "www01" ],
     "vars": { "http_port": 80, "spooldir" : "/dump" }
  },
  "_meta": {
    "hostvars": {
       "www01": { "regno": "A001", "location" : "Spain" },
       "127.0.0.1": { "regno": "X094" }
    }
  }
}
```

```
$ ansible webservers -i inventory.sh -m debug -a var=regno
127.0.0.1 | SUCCESS => {
    "regno": "X094"
}
www01 | SUCCESS => {
    "regno": "A001"
}
```