Ansible

Ansible, created by Michael DeHaan, is a radically simple model-driven configuration management, deployment, and command execution framework. Other than Python 2.6 and a working SSH infrastructure, Ansible requires no setup, no daemons, no PKI, no nothing (Fig. 1).

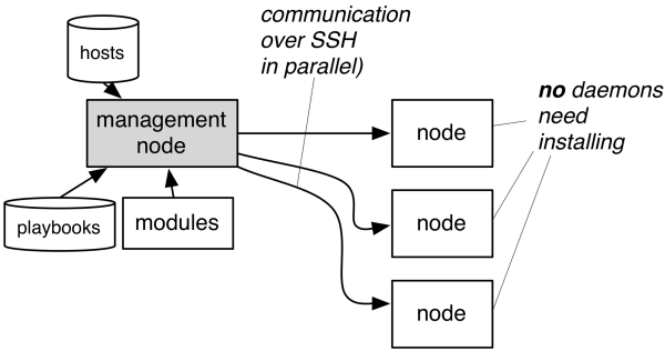

Figure 1: Ansible architecture

Problems?

If you need help or want to report a problem Ansible has a mailing-list at groups.google.com/group/ansible-project. The latest and greatest code is on github.com/ansible/ansible where you can track issues and submit ideas. There are also lots of people willing to assist on IRC: log in to `#ansible` on FreeNode.

Getting started

In this document we'll call the machine you run Ansible on (i.e. the machine from which you deploy) the *master*; machines onto which you deploy (i.e. your clients), we'll call *nodes*. On the *master* you don't necessarily require `root` permission: Ansible can run as any user. Similarly, on *nodes* you don't need root permissions either (well, maybe not): Ansible can connect to *nodes* as a "normal" user and then (if required) `sudo` to root.
Download Ansible and unpack. You don't need to install it ... running from a git checkout is fine. Create your *inventory* file and test: See the section on SSH for setting up SSH.
Ansible requires Python 2.6 though nodes only 2.4

Inventory

Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in the inventory file, which defaults to `/etc/ansible/hosts`.

```
localhost

[webs]
www.example.com
web[09-12].example.com
192.168.8.9

[devservers]
box1.example.com
jo.example.com ntpserver=127.0.0.1
```

That last entry has what is called a *host var*; ignore that for now.

Modules

Ansible ships with a number of modules[1] (called the "module library") that can be executed directly on remote hosts or through Playbooks. Users can also write their own modules. These modules can control system resources, like services, packages, or files (anything really), or handle executing system commands. The following is a list of modules in the core library with supported options (• is mandatory, ◦ optional). (If you're viewing this in a PDF reader, click on the module name for its official documentation.)

```
ansible 192.168.8.9 -m ping
ansible webs -m copy -a "src=/tmp/f dest=/etc/conf"
```

`file`

Sets attributes of files, symlinks, and directories, or removes files/symlinks/directories. Many other modules support the same options as the file module - including `copy`, `template`, and `assmeble`.

• `dest`	defines the file being managed, unless when used with *state=link*, and then sets the destination to create a symbolic link to using *src*
◦ `state`	If directory, all immediate subdirectories will be created if they do not exist. If *file*, the file will NOT be created if it does not exist, see the `copy` or `template` module if you want that behavior. If *link*, the symbolic link will be created or changed. If absent, directories will be recursively deleted, and files or symlinks will be unlinked. **Choices**: `file`, `link`, `directory`, `absent`. (default `file`)
◦ `mode`	mode the file or directory should be, such as 0644 as would be fed to *chmod*. English modes like **g+x** are not yet supported

See also `copy`, `template`, `assemble`

`get_url`

Downloads files from HTTP, HTTPS, or FTP to the remote server. The remote server must have direct access to the remote resource.

• `url`	HTTP, HTTPS, or FTP URL

[1] `http://ansible.github.com/modules.html`

• dest	absolute path of where to download the file to. If *dest* is a directory, the basename of the file on the remote server will be used. If a directory, *thirsty=yes* must also be set.
◦ thirsty	if yes, will download the file every time and replace the file if the contents change. if no, the file will only be downloaded if the destination does not exist. Generally should be yes only for small local files. prior to 0.6, acts if yes by default. **Choices**: yes, no. (default no) (** version 0.7*)
◦ others	all arguments accepted by the file module also work here

This module doesn't support proxies or passwords. Also see the template *module.*

raw

Executes a low-down and dirty SSH command, not going through the module subsystem. This is useful and should only be done in two cases. The first case is installing python-simplejson on older (Python 2.4 and before) hosts that need it as a dependency to run modules, since nearly all core modules require it. Another is speaking to any devices such as routers that do not have any Python installed. In any other case, using the shell or command module is much more appropriate. Arguments given to raw are run directly through the configured remote shell and only output is returned. There is no error detection or change handler support for this module

```
ansible host -m raw -a "yum -y install python-simplejson"
```

setup

This module is automatically called by playbooks to gather useful variables about remote hosts that can be used in playbooks. It can also be executed directly by /usr/bin/ansible to check what variables are available to a host. Ansible provides many *facts* about the system, automatically.
More ansible facts will be added with successive releases. If facter or ohai are installed, variables from these programs will also be snapshotted into the JSON file for usage in templating. These variables are prefixed with facter_ *and* ohai_ *so it's easy to tell their source. All variables are bubbled up to the caller. Using the ansible facts and choosing to not install facter and ohai means you can avoid Ruby-dependencies on your remote systems.*

```
"ansible_architecture": "x86_64",
"ansible_distribution": "CentOS",
"ansible_distribution_release": "Final",
"ansible_distribution_version": "6.2",
"ansible_eth0": {
    "ipv4": {
        "address": "REDACTED",
        "netmask": "255.255.255.0"
    },
    "ipv6": [
        {
            "address": "REDACTED",
            "prefix": "64",
            "scope": "link"
        }
    ],
    "macaddress": "REDACTED"
},
"ansible_form_factor": "Other",
"ansible_fqdn": "localhost.localdomain",
"ansible_hostname": "localhost",
"ansible_interfaces": [
    "lo",
    "eth0"
],
```

Playbooks

Simply put, Playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications. Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders. They can launch tasks synchronously or asynchronously. Playbooks are expressed in YAML[2] format and have a minimum of syntax. (Tip: use the *Online YAML Parser*[3] to experiment.) Each playbook is composed of one or more *plays* in a list. Here's a playbook that contains just one play:

```
---
- hosts: devservers
  vars:
    http_port: 80
    conf: httpd.j2
  user: root
  tasks:
  - name: ensure apache is at the latest version
    action: yum pkg=httpd state=latest
  - name: write the apache config file
    action: template src=/srv/${conf} dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    action: service name=httpd state=started
  handlers:
    - name: restart apache
      action: service name=apache state=restarted
```

```
ansible-playbook -u jpm mini.yaml
```

Use the -verbose flag for more information.

```
only_if: "not '$ansible_cmdline.BOOT_IMAGE'.startswith('$')"
```

Handlers & Notification

Modules are written to be *idempotent* and can relay when they have made a change on the remote system. Playbooks recognize this and have a basic event system that can be used to respond to change. These *notify* actions are triggered at the end of each *play* in a playbook, and trigger only once each. For instance, multiple resources may indicate that apache needs to be restarted, but apache will only be bounced once. Here's an example of restarting two services when the contents of a file change, but only if the file changes:

```
- name: template configuration file
  action: template src=template.j2 dest=/etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

[2] http://www.yaml.org/
[3] http://yaml-online-parser.appspot.com/

The things listed in the *notify* section of a task are called *handlers*. Handlers are lists of tasks, not really any different from regular tasks, that are referenced by name. Handlers are what notifiers notify. If nothing notifies a handler, it will not run. Regardless of how many things notify a handler, it will run only once, after all of the tasks complete in a particular play. Handlers are best used to restart services and trigger reboots. You probably won¿t need them for much else. Here¿s an example handlers section:

```
handlers:
    - name: restart memcached
      action: service name=memcached state=restarted
    - name: restart apache
      action: service name=apache state=restarted
```

Notify handlers are always run in the order written.

Templates

FIXME: short description of Jinja2 templates with one or two short examples using some vars from playbook and `setup`.
show: expansion if, else, endif switch / case ???? loops

Delegation

FIXME: what delegation is. mention localaction needs SSH to localhost

Facts

refer to `setup`; maybe show short module example?

Variables

Host vars

Group vars

SSH

Paramiko is a native python implementation of the SSH protocol. It's the default transport method used by Ansible, this method should work for 99% of people by default. The native ssh transport method is necessary in more complicated infrastructures where support for bastion ('proxy') hosts is required, or if GSSAPI (kerberos) is used for authentication. The native ssh transport will recognize your `/.ssh/config` file because it uses the 'ssh' command installed on the local system.
[somegroup] foo ansible_ssh_port=1234 bar ansible_ssh_port=1235

Shell variables used by Ansible

You might want to stick to just mentioning $ANSIBLE_CONFIG and giving reference to http://ansible.github.com/examples.html#configuration-defaults

ANSIBLE_SSH_ARGS

ANSIBLE_REMOTE_USER

Extending Ansible

Ansible is extensible: you can use the *Ansible* Python API to control nodes, you can extend Ansible to respond to various python events, and you can plug in inventory data from external data sources. *Ansible* is written in its own API so you have a considerable amount of power across the board[4].

Your own modules

Ansible modules are reusable units which can be used by the *Ansible* API, or by the `ansible` or `ansible-playbook` programs. Modules can be written in any language supported by *nodes* (e.g. shell scripts[5]) and are found in the path specified by `ANSIBLE_LIBRARY_PATH` or the `-module-path` command-line option[6].
The following listing illustrates what an *Ansible* module looks like in Python; it accepts a single parameter (`name`) with the name of a file on a node for which the file size should be retrieved:

```
#!/usr/bin/python
import os

DOCUMENTATION = '''
---
module: mini
short_description: Determine file size of remote file
description:
     - Determines the size of a specified file.
version_added: "0.0"
options:
  - name:
      description:
        - Absolute path to the file name on the remote node.
      required: true
      default: null
      aliases: [dest, destfile]
'''

def main():
    module = AnsibleModule(
        argument_spec = dict(
            name=dict(required=True, aliases=['dest', 'destfile'])
        ),
    )

    params = module.params
    filename = params['name']

    try:
        stat = os.stat(filename)
    except:
        module.fail_json(msg="Can't stat file: %s" % filename)

    changed = False
    msg = "Filename %s has size %s" % (filename, stat.st_size)
    module.exit_json(changed=changed, msg=msg)

# this is magic, see lib/ansible/module_common.py
#<<INCLUDE_ANSIBLE_MODULE_COMMON>>

main()
```

Use this module in a playbook or from the command-line:

[4] http://ansible.github.com/api.html
[5] http://mens.de/:/ansshell
[6] http://ansible.github.com/moduledev.html

```
ansible 127.0.0.1 -c local -m mini -a dest=/tmp/xx
127.0.0.1 | success >> {
    "changed": false,
    "msg": "Filename /tmp/xx has size 1233"
}
```

You should look at some of the modules in *Ansible*'s `library/*` for inspiration before writing your own.
Modules you write can also return *facts* like the `setup` module does, but you have to call your modules explicitly, whereas `setup` is invoked automatically from a playbook. (Fig. 2) A fact-gathering module can be written as trivially as this example in a shell script[7].

```
#!/bin/sh

COUNT=`who | wc -l`
cat <<EOF
{
   "ansible_facts" : {
      "users_logged_in" : $COUNT
    }
}
EOF
```

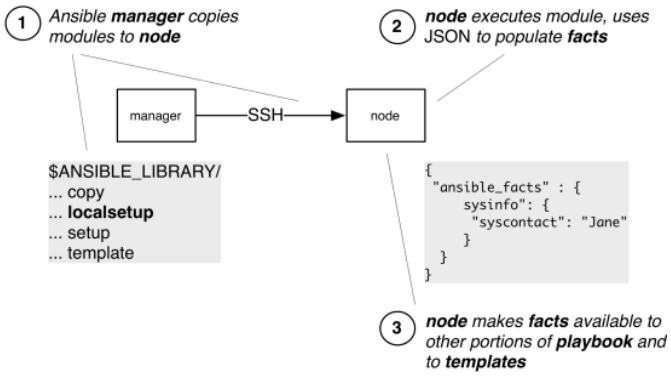

Figure 2: Ansible copies modules to nodes

Callback plugins

Action plugins

Pull mode

Instead of pushing configuration from a master to nodes it may be advantageous to pull from the master onto the nodes. *Ansible* can run a playbook on a node if the node has a full installation of *Ansible* and its dependencies. The example in Fig. 3 shows how to accomplish this using a repository available to all nodes from which they obtain current copies of Playbooks and supporting files [8].

Tips and tricks: fun with Ansible

1. If your *nodes* have a version of Python which doesn't meet Ansible's requirements, install Python non-destructively in, say, `/usr/local/Python`, and configure your inventory to use that path on nodes (e.g. with a group variable).

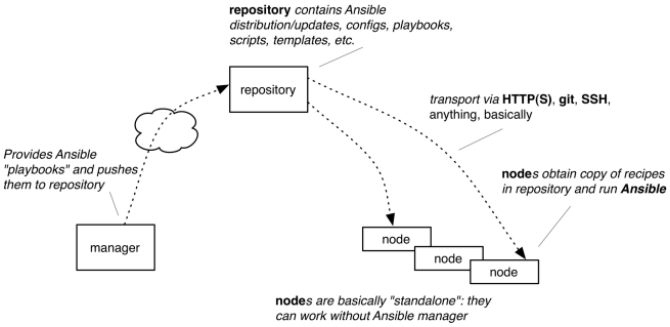

Figure 3: Ansible pull-mode

```
ansible_python_interpreter: /usr/local/Python
```

2. Install Cowsay. You must! (And make sure it's (symlinked) in `/usr/bin/cowsay`.)

```
 _____________________________
< TASK: [Ansible says mooooo] >
 -----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

3. Other "hidden" ansible variables FIXME

[7]

[8] `http://mens.de/:/anspull`