Automation with Ansible

Find the latest, print-friendly version of this presentation and tutorial materials at https://christopherdemarco.com/ansible

The opinions and mistakes that follow are my own and do not represent my employer, Red Hat, USENIX, or anyone else.

All code samples were believed correct at runtime.
Your mileage may vary.

To my grandfather, who taught me how to write.
To my father, who taught me why.

This tutorial is interactive.

Please interrupt me!

Join us on Slack!
#m8-ansible

http://lisainvite.herokuapp.com/

Who has used Ansible before?

Lightweight configuration management

Stop managing your tools, start using them.

Possibly pay Red Hat to help.

Agentless

~~Install and maintain the client.~~

~~Strange firewall ports?~~

Got SSH?

Got Python >= 2.4? (and maybe not even that!)

Serverless

~~Admin SPOF / Yet Another Cluster~~

Install Ansible, pull configuration codebase, & run locally.

No daemons or databases.

Laptop? Jenkins? On the node itself?

Stateless

Ship bytecode to the nodes being configured—load is in the targets, not the controller.

Keep static host inventories in source control.

Generate dynamic host inventories; scripts simply output JSON.

Small DSL

YAML

Jinja2

Python

OTOH

SSH and Python are slow.

YAML is *too* easy.

Ansible is procedural, not declarative.

Ansible’s DSL does not incorporate a general-purpose programming language.

There’s lots of infrastructure we can’t demo!

AWS

Google Compute Engine

OpenStack

Azure

Kubernetes

Cisco

F5

NetApp

Windows

. . .

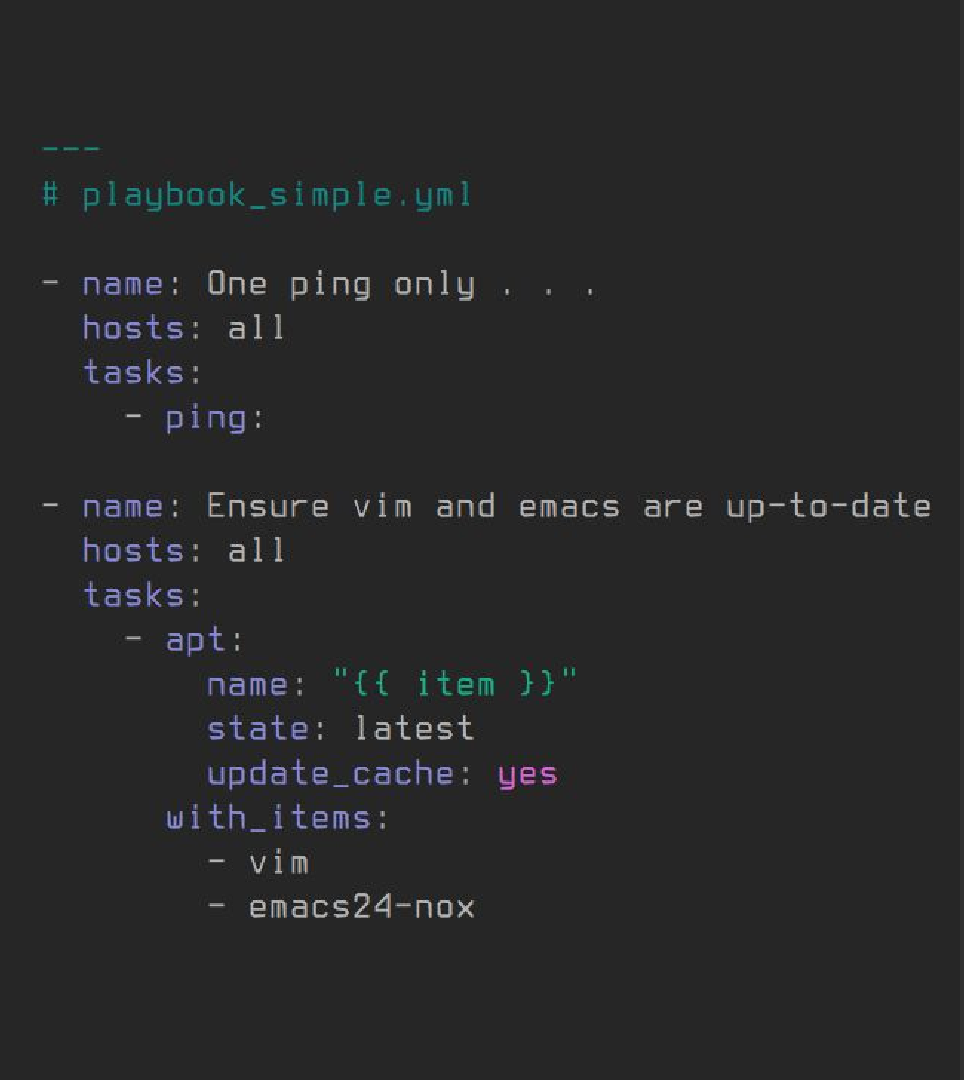
Hello Docker

Playbooks

A play consists of tasks.

A groups of plays is called a playbook.

Playbooks are structured as YAML lists and dictionaries.



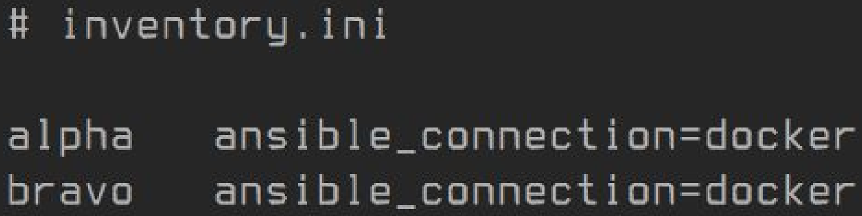
```yaml
---
# playbook_simple.yml

- name: One ping only . . .
  hosts: all
  tasks:
    - ping:

- name: Ensure vim and emacs are up-to-date
  hosts: all
  tasks:
    - apt:
        name: "{{ item }}"
        state: latest
        update_cache: yes
      with_items:
        - vim
        - emacs24-nox
```

`hosts` and `tasks` are required parameters of the play.

The `tasks` parameter contains a list of modules. (`ping` and `apt`)

`state` is a parameter of a module.
`with_items` is a parameter of a task. Do not confuse them!

Inventory



```
# inventory.ini

alpha   ansible_connection=docker
bravo   ansible_connection=docker
```

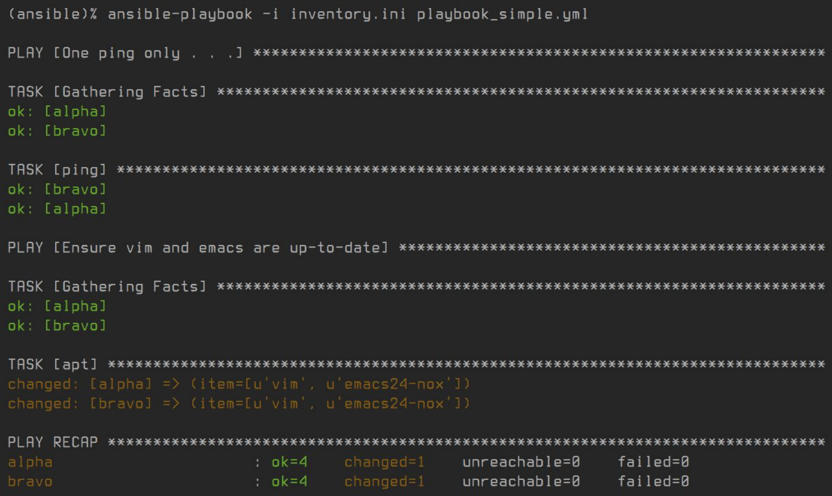
Use INI or YAML.

Or use a script for dynamic inventory.

Set variables.

Set groups.

Run it!

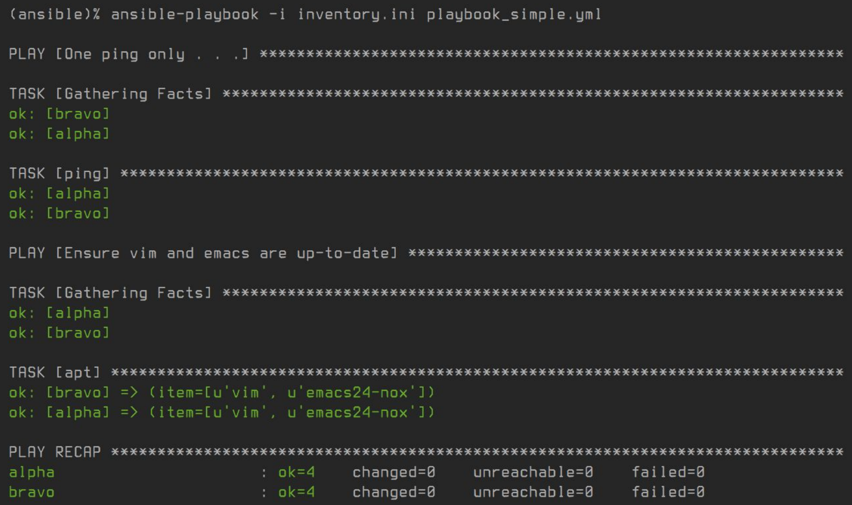


```
(ansible)% ansible-playbook -i inventory.ini playbook_simple.yml

PLAY [One ping only . . .] ***************************************************************

TASK [Gathering Facts] *******************************************************************
ok: [alpha]
ok: [bravo]

TASK [ping] ******************************************************************************
ok: [bravo]
ok: [alpha]

PLAY [Ensure vim and emacs are up-to-date] ***********************************************

TASK [Gathering Facts] *******************************************************************
ok: [alpha]
ok: [bravo]

TASK [apt] *******************************************************************************
changed: [alpha] => (item=[u'vim', u'emacs24-nox'])
changed: [bravo] => (item=[u'vim', u'emacs24-nox'])

PLAY RECAP *******************************************************************************
alpha                      : ok=4    changed=1    unreachable=0    failed=0
bravo                      : ok=4    changed=1    unreachable=0    failed=0
```

Ansible runs against all hosts in parallel.

Show changes as they’re made, and summarize overall results.

Re-run it!



```
(ansible)% ansible-playbook -i inventory.ini playbook_simple.yml

PLAY [One ping only . . .] ***************************************************************

TASK [Gathering Facts] *******************************************************************
ok: [bravo]
ok: [alpha]

TASK [ping] ******************************************************************************
ok: [alpha]
ok: [bravo]

PLAY [Ensure vim and emacs are up-to-date] ***********************************************

TASK [Gathering Facts] *******************************************************************
ok: [alpha]
ok: [bravo]

TASK [apt] *******************************************************************************
ok: [bravo] => (item=[u'vim', u'emacs24-nox'])
ok: [alpha] => (item=[u'vim', u'emacs24-nox'])

PLAY RECAP *******************************************************************************
alpha                      : ok=4    changed=0    unreachable=0    failed=0
bravo                      : ok=4    changed=0    unreachable=0    failed=0
```

Ansible will only do what’s necessary.

Now you try it . . .

Accessing your lab workstation

Your paper tokens expire.

Do not break your instances.

Everything disappears afterwards.

Beware USENIX Code of Conduct.

Ansible and class materials are installed.

API credentials expire after this session, don’t try anything funny.

Code samples are provided.

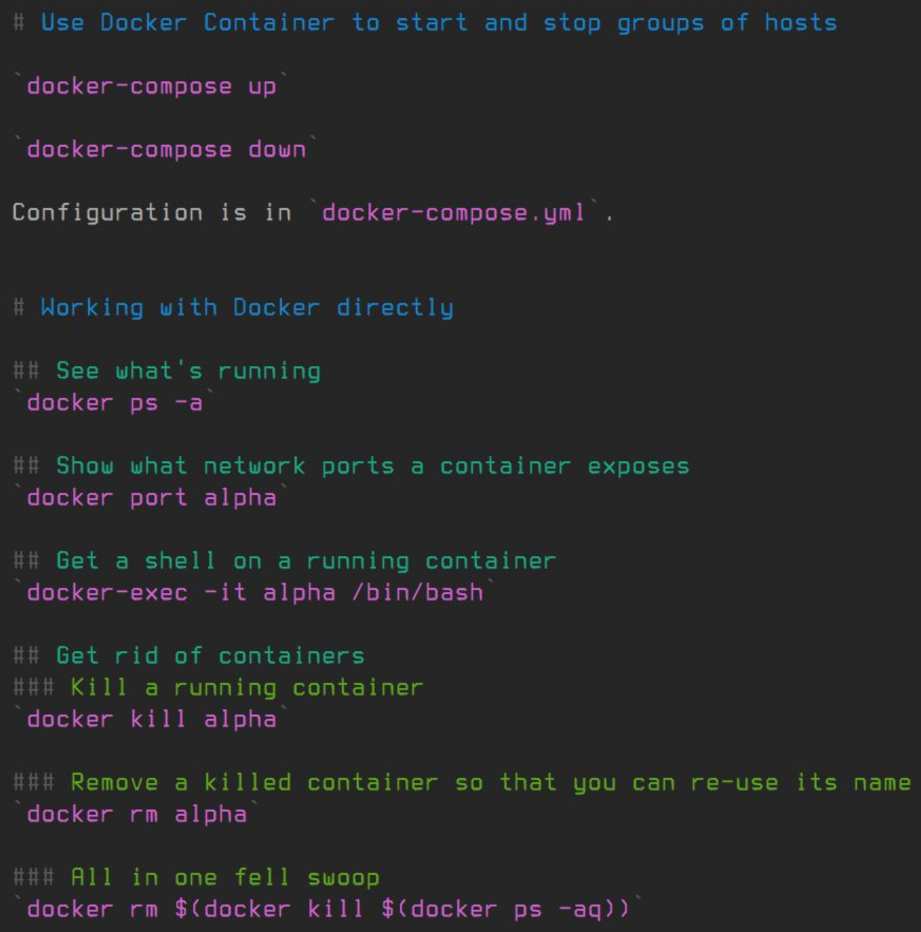
The first “hello docker” example is in `~/class/1330_hello`.

Background `docker-compose up`, use tmux/screen, or open a second SSH session.

Once you’ve provisioned the container, connect to it and play around.

Docker 101

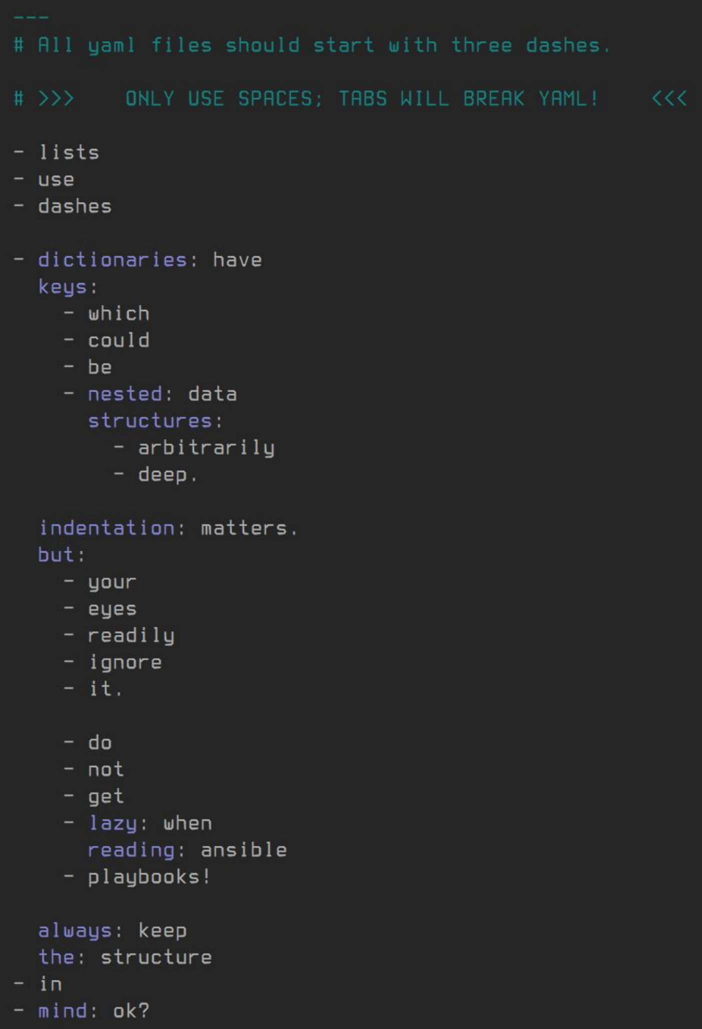
~/class/docker_101.md



```
# Use Docker Container to start and stop groups of hosts

`docker-compose up`

`docker-compose down`

Configuration is in `docker-compose.yml`.


# Working with Docker directly

## See what's running
`docker ps -a`

## Show what network ports a container exposes
`docker port alpha`

## Get a shell on a running container
`docker-exec -it alpha /bin/bash`

## Get rid of containers
### Kill a running container
`docker kill alpha`

### Remove a killed container so that you can re-use its name
`docker rm alpha`

### All in one fell swoop
`docker rm $(docker kill $(docker ps -aq))`
```

YAML sucks.

~/class/yaml_sucks.yml



```
---
# All yaml files should start with three dashes.

# >>>    ONLY USE SPACES; TABS WILL BREAK YAML!    <<<

- lists
- use
- dashes

- dictionaries: have
  keys:
    - which
    - could
    - be
    - nested: data
      structures:
        - arbitrarily
        - deep.

  indentation: matters.
  but:
    - your
    - eyes
    - readily
    - ignore
    - it.

    - do
    - not
    - get
    - lazy: when
      reading: ansible
    - playbooks!

  always: keep
  the: structure
- in
- mind: ok?
```

DON’T USE TABS IN YAML!

http://fortunes.example

Let’s build it.

What do we need to do?

Copy source.

Install requirements.

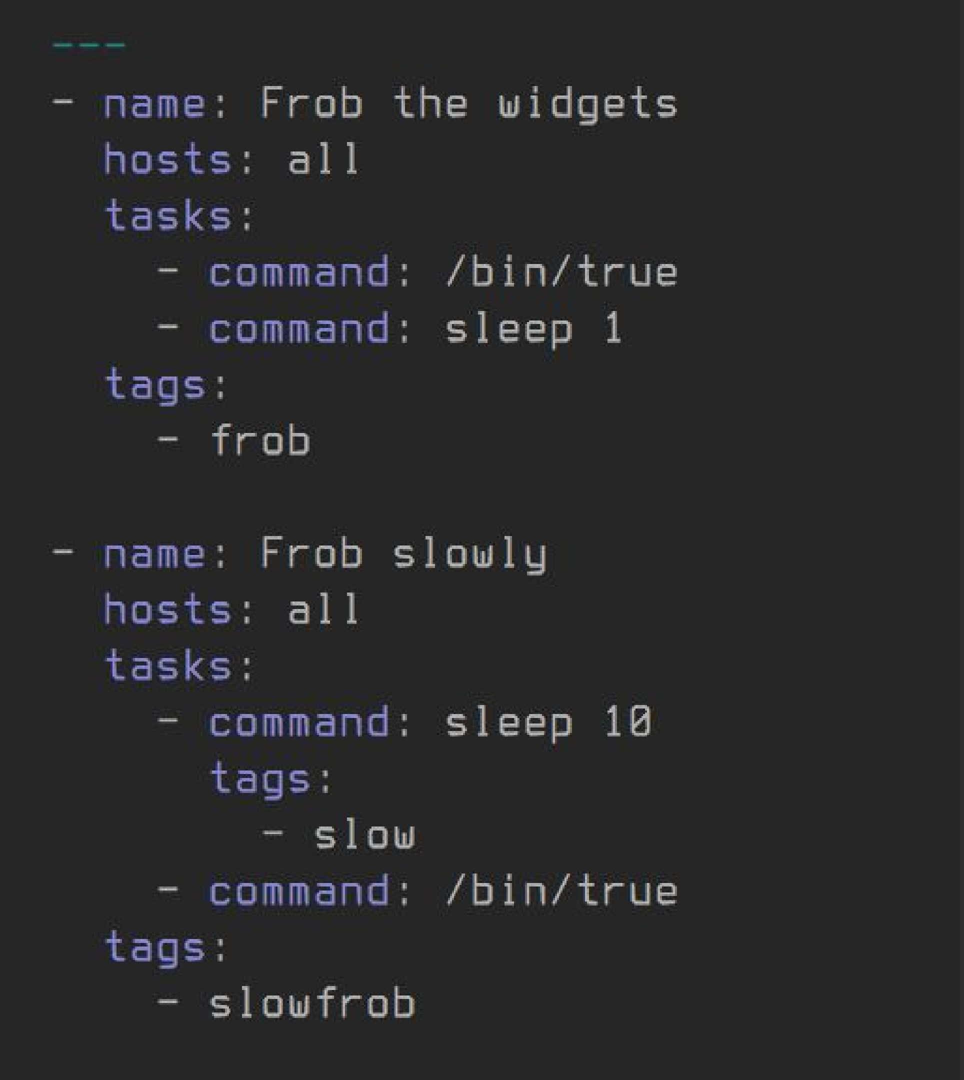
Set up app.

Does it work?

This docker-compose configuration exposes tcp/80.

Test with curl.

Tag things to skip them.



```yaml
---
- name: Frob the widgets
  hosts: all
  tasks:
    - command: /bin/true
    - command: sleep 1
  tags:
    - frob

- name: Frob slowly
  hosts: all
  tasks:
    - command: sleep 10
      tags:
        - slow
    - command: /bin/true
  tags:
    - slowfrob
```

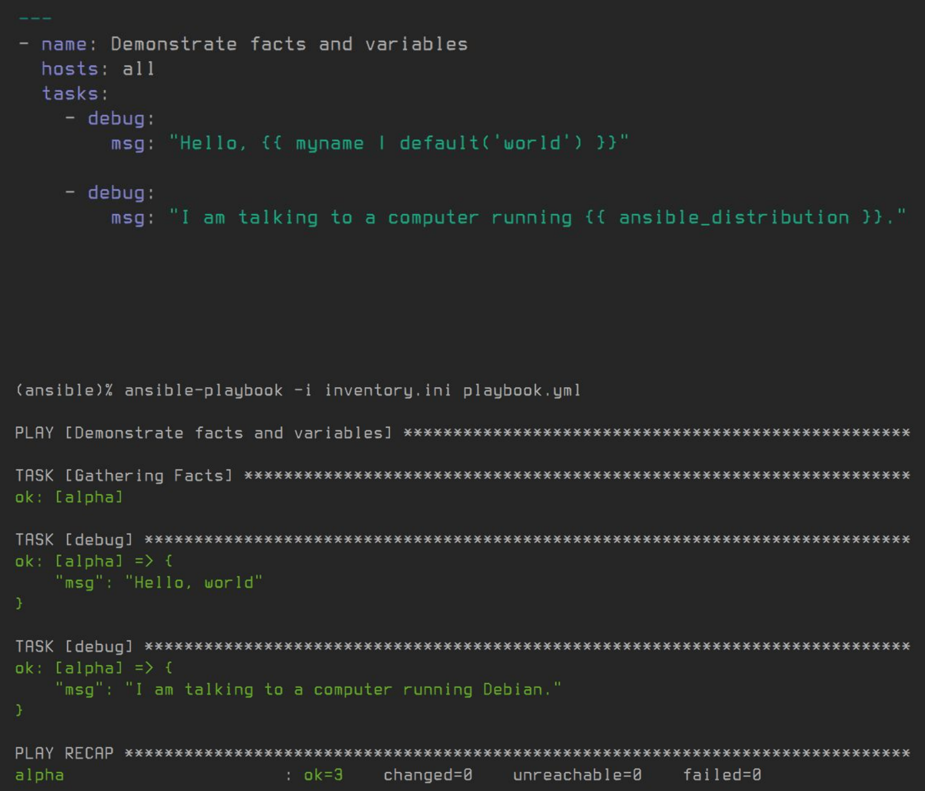
Use the `ansible-playbook --tags=` option to run only selected tags.

Use the `--skip-tags=` option to exclude selected tags.

Separate multiple tags with a comma [and no space].

Use the `ansible-playbook --list-tasks` option to see what tags are defined.

Variables



```
---
- name: Demonstrate facts and variables
  hosts: all
  tasks:
    - debug:
        msg: "Hello, {{ myname | default('world') }}"

    - debug:
        msg: "I am talking to a computer running {{ ansible_distribution }}."
```

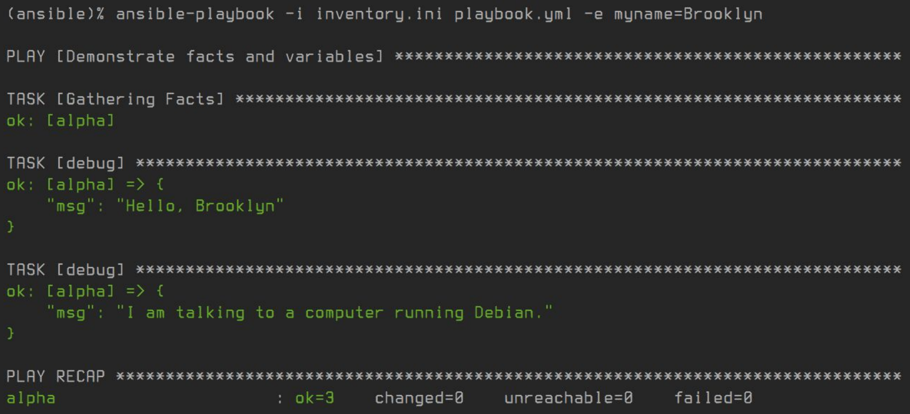
```
(ansible)% ansible-playbook -i inventory.ini playbook.yml

PLAY [Demonstrate facts and variables] ***************************************************

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

TASK [debug] *****************************************************************************
ok: [alpha] => {
    "msg": "Hello, world"
}

TASK [debug] *****************************************************************************
ok: [alpha] => {
    "msg": "I am talking to a computer running Debian."
}

PLAY RECAP *******************************************************************************
alpha                      : ok=3    changed=0    unreachable=0    failed=0
```

Variables can have default values.

Facts are variables discovered automatically by Ansible.

Interpolate variables using Jinja2 syntax.

Use the `debug` module to print.



```
(ansible)% ansible-playbook -i inventory.ini playbook.yml -e myname=Brooklyn

PLAY [Demonstrate facts and variables] ****************************************************

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

TASK [debug] ******************************************************************************
ok: [alpha] => {
    "msg": "Hello, Brooklyn"
}

TASK [debug] ******************************************************************************
ok: [alpha] => {
    "msg": "I am talking to a computer running Debian."
}

PLAY RECAP ********************************************************************************
alpha                      : ok=3    changed=0    unreachable=0    failed=0
```

Set variables on the command line with the `-e` argument.

Or define them in inventory.

Or use `include_vars` to load them from a file.

Yeah, but Docker . . .

Let’s build it on a real host.

Managing / accessing your lab SSH host

alpha.<your-host>.foam.ninja

Use `~/class/inventory.py` as your Ansible inventory.

Username is `ubuntu`. Password login is not permitted; use the SSH key at `~/.ssh` .

Recreate alpha by running the `~/recreate_alpha.sh` script; don't forget to remove stale keys from `~/.ssh/known_hosts`.

Docker vs. a real host

How do we switch these on?

sudo

initscript/service

How do we become a different user?

`become`

`become_user`

Gotcha: `become` applies to the play—not to the task list, or to individual tasks.

How do we branch on platform type?

List the available facts: `ansible -m setup -i <inventory> all`.

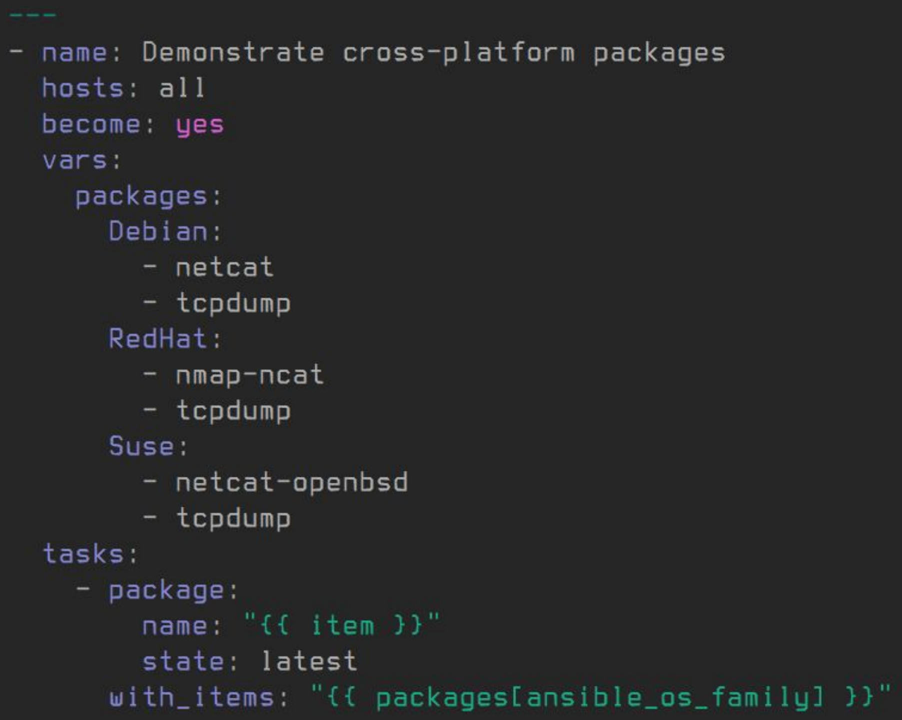
Use a task’s `when` parameter to restrict its execution.

Gotcha: no Jinja braces in `when`!

Point your browser at http://alpha.your-workstation.foam.ninja!

It sure is annoying to set `become_bool` in inventory . . .

Yeah, but Debian . . .



```yaml
---
- name: Demonstrate cross-platform packages
  hosts: all
  become: yes
  vars:
    packages:
      Debian:
        - netcat
        - tcpdump
      RedHat:
        - nmap-ncat
        - tcpdump
      Suse:
        - netcat-openbsd
        - tcpdump
  tasks:
    - package:
        name: "{{ item }}"
        state: latest
      with_items: "{{ packages[ansible_os_family] }}"
```

You can set variables in a play.

`packages` is a dictionary. Index using Python syntax.

The `package` module is cross-platform.

Roles

Roles make things modular.

Use them as much as possible!

Install and configure an application.

Apply common configuration.

Bundle assets and resources.

Share code.

Userland

Let’s build it.

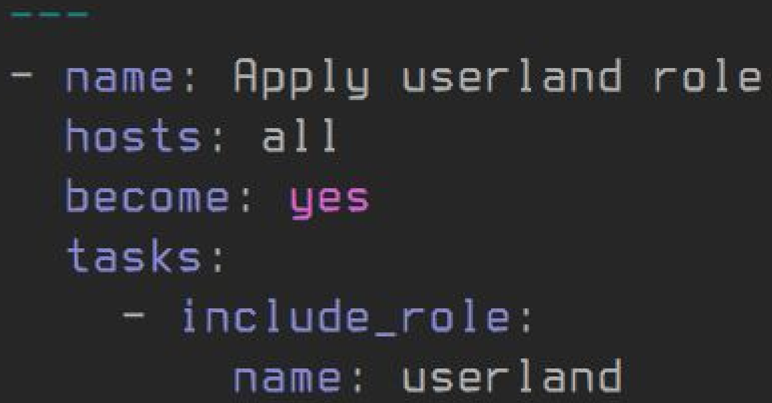
What do we need to do?

Provide my preferred username and shell.

Authenticate using my GitHub keypair.

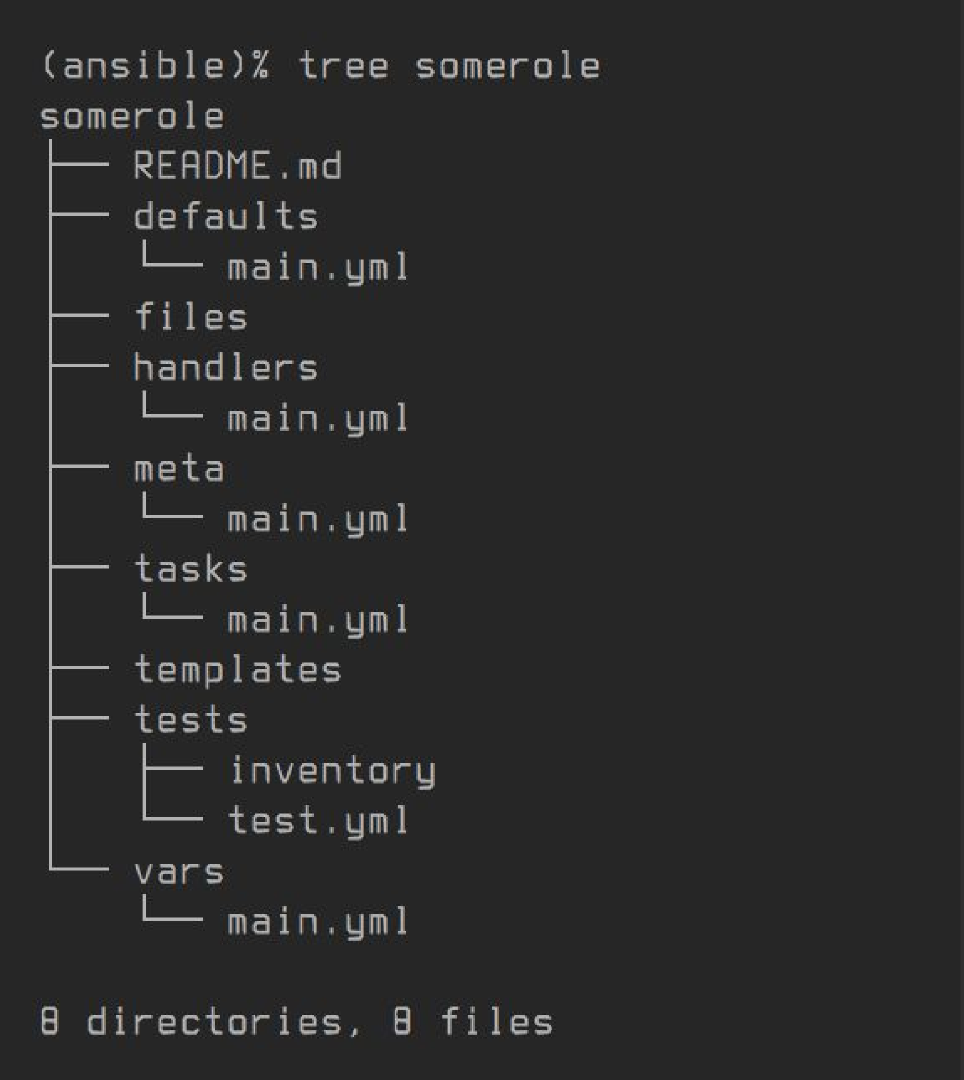
Portably install the essentials.

Setup a convenience alias in `~/.ssh/config`.



```yaml
---
- name: Apply userland role
  hosts: all
  become: yes
  tasks:
    - include_role:
        name: userland
```

Use a role with the `include_role` task.



```
(ansible)% tree somerole
somerole
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

Set `roles_path` in `ansible.cfg`.

`ansible-galaxy init <rolename>`.

Put tasks in `tasks/main.yml`. (Note that this is a task list—like you'd put in a play—not a playbook.) Similarly for `handlers/main.yml`.

Put files/templates in their respective directories, and you can use them within the role without an explicit path.

Set role defaults.

Note that my defaults are probably not what you want!

As a role author, it’s your responsibility to set sane defaults.

Tasks return data structures.

Provide status, stdout/stderr, etc.

`register` them, then use them like variables.

Output them using `debug`.

Work locally.

`delegate_to: localhost`

Let’s set a login password.
How can we store it securely?

Keep secrets in `ansible-vault`.

Use the `import_vars` task to pull vars into the current play.

`ansible-vault [create | edit | view]`

It’s encrypted, therefore safe to store in version control.

`ansible-playbook --ask-vault-pass`

Gotcha!
sshd doesn’t permit password logins!

Handlers

Handlers `register` listeners.

Tasks can `notify` handlers.

Put a handler in a play, or in the `handlers` list of a role.

Use third-party roles.

But are they any good?

http://galaxy.ansible.com

`ansible-galaxy install user.role`

Gotcha! `ansible-galaxy` ignores `ansible.cfg` and installs to ~/.ansible/roles` unless you set `ANSIBLE_ROLES_PATH`.

Check them into source control?

Read them!

OK for prototyping . . . but you'll probably rewrite 'em . . .

Keep things organized.

Playbooks go in the top level.

Directories for inventory, roles, variables, etc.

`roles_path`

Use version control!

Manage third-party modules!

Password-protect fortunes.

Let’s build it.

Templates

`template` is just another module. Syntax is basically like `copy`.

Use the familiar variable interpolation syntax—including filters.

Wrap Python code with `{%  %}`.

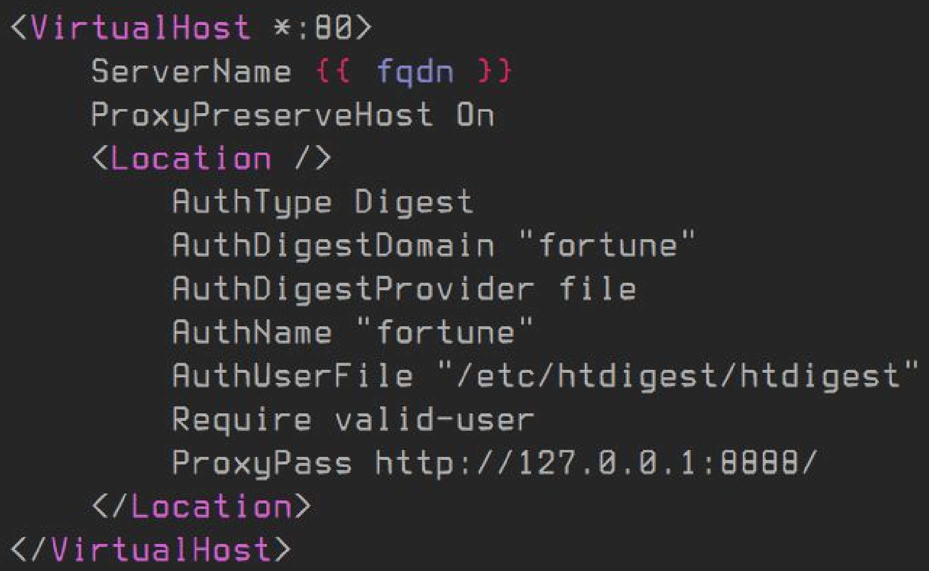
What do we need to do?

Install Apache2.

Template the config.

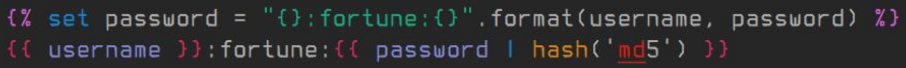
Enable the config and modules, restart `apache2`.

Set up htpasswd.



```
<VirtualHost *:80>
    ServerName {{ fqdn }}
    ProxyPreserveHost On
    <Location />
        AuthType Digest
        AuthDigestDomain "fortune"
        AuthDigestProvider file
        AuthName "fortune"
        AuthUserFile "/etc/htdigest/htdigest"
        Require valid-user
        ProxyPass http://127.0.0.1:8080/
    </Location>
</VirtualHost>
```

The Apache template is simple with just a single variable expansion.



```
{% set password = "{}:fortune:{}".format(username, password) %}
{{ username }}:fortune:{{ password | hash('md5') }}
```

The htpasswd template shows an inline Python code block, and a filter.

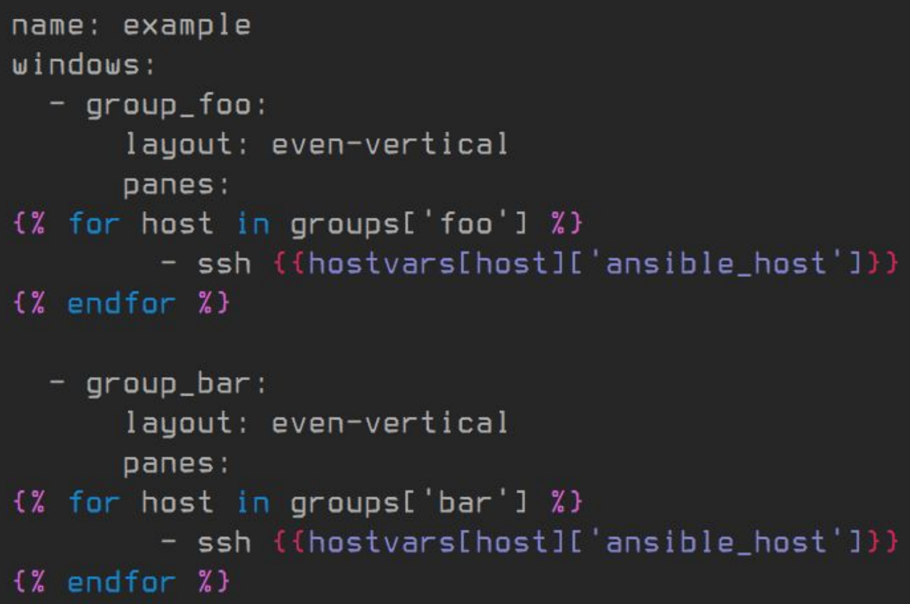
Automate tmuxinator.

```
---
- name: Write tmuxinator manifest
  become: no
  hosts: localhost
  tasks:
    - file:
        name: ~/.tmuxinator
        state: directory
    - template:
        src: tmuxinator.yml.j2
        dest: "{{ ansible_env.HOME }}/.tmuxinator/example.yml"
      delegate_to: localhost
```

Because this is a bogus example, we need to limit execution to localhost.

The magic `ansible_env` variable is a dictionary containing the env vars of the host *on which the play was run*.

The `delegate_to` parameter specifies which host will run the task.



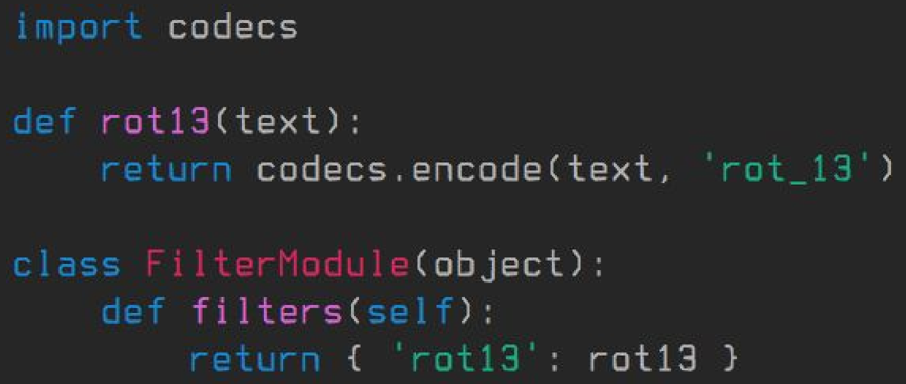
```
name: example
windows:
  - group_foo:
      layout: even-vertical
      panes:
{% for host in groups['foo'] %}
        - ssh {{hostvars[host]['ansible_host']}}
{% endfor %}

  - group_bar:
      layout: even-vertical
      panes:
{% for host in groups['bar'] %}
        - ssh {{hostvars[host]['ansible_host']}}
{% endfor %}
```

`groups` is a variable like any other.

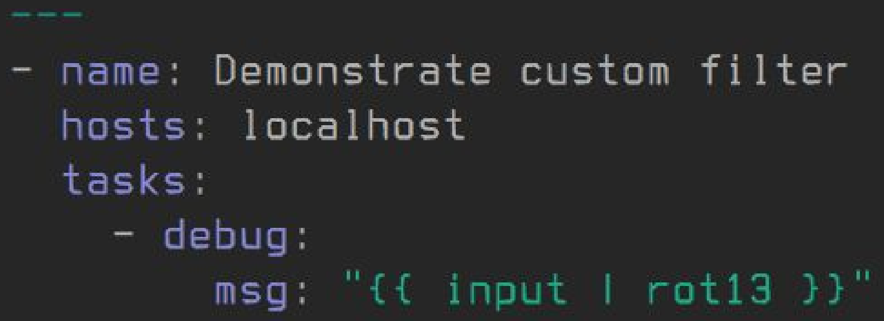
Note that interpolation braces are not required within a Python block.

`hostvars` lets you reference an arbitrary host’s variables!

Jevgr phfgbz svygref.



```python
import codecs

def rot13(text):
    return codecs.encode(text, 'rot_13')

class FilterModule(object):
    def filters(self):
        return { 'rot13': rot13 }
```



```yaml
---
- name: Demonstrate custom filter
  hosts: localhost
  tasks:
    - debug:
        msg: "{{ input | rot13 }}"
```

Write plugins in Python. Each plugin implements a class whose `filters` method returns a function implementing the actual filter.

Put your Python in a `filter_plugins/` directory adjacent to your playbook or inside your role.

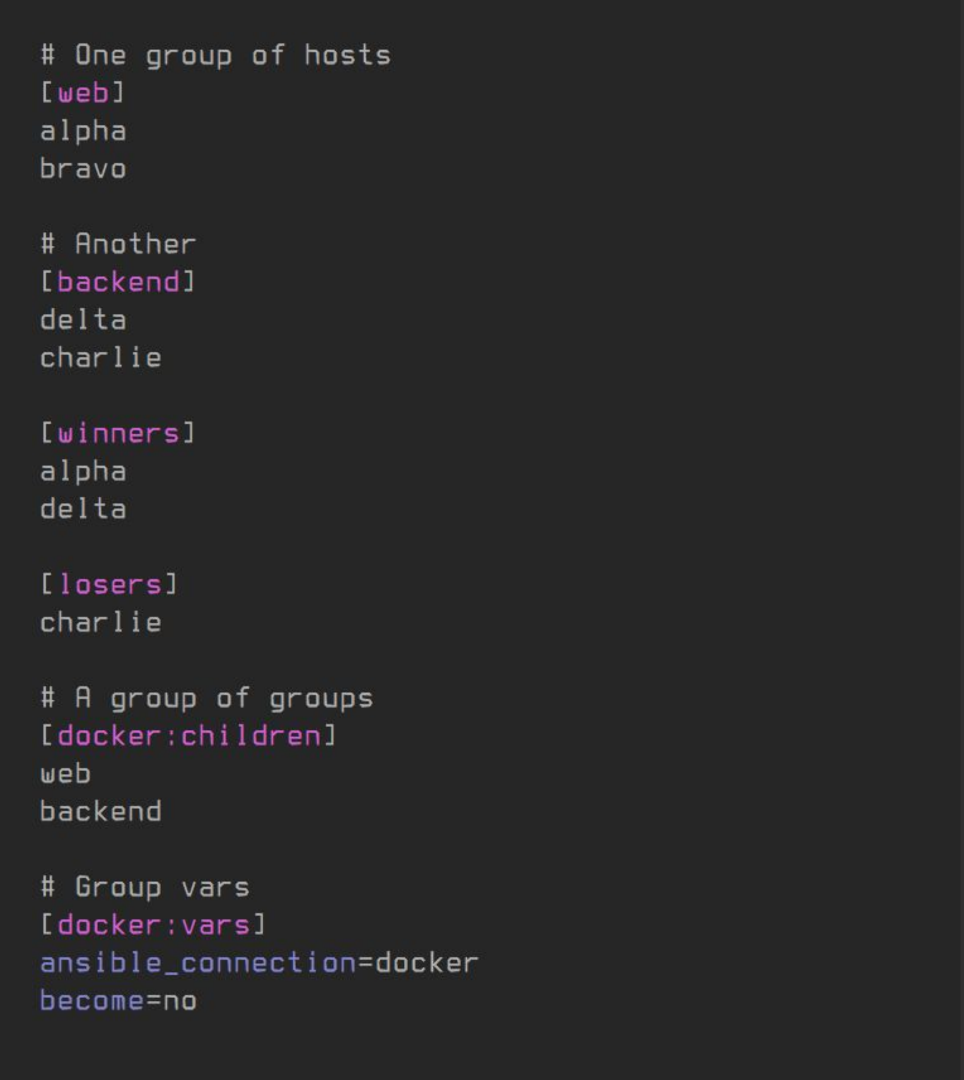
Documentation tour

Beware the bright light . . .

break

return at 1530

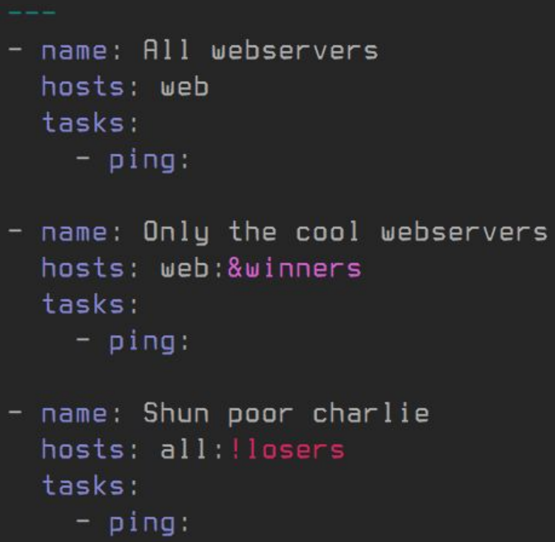
Grouping and limiting



```
# One group of hosts
[web]
alpha
bravo

# Another
[backend]
delta
charlie

[winners]
alpha
delta

[losers]
charlie

# A group of groups
[docker:children]
web
backend

# Group vars
[docker:vars]
ansible_connection=docker
become=no
```

Inventory groups can include other groups.

Variables are inherited via group membership.

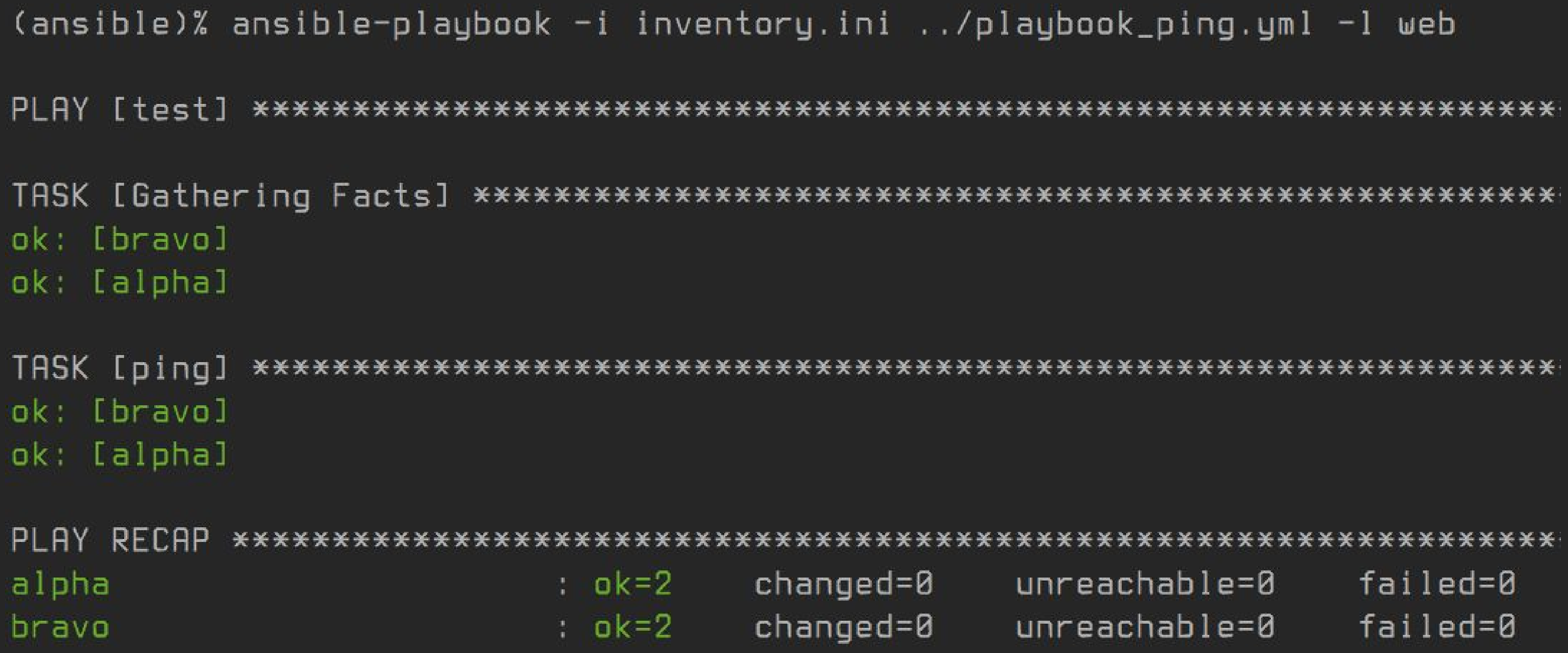


```yaml
---
- name: All webservers
  hosts: web
  tasks:
    - ping:

- name: Only the cool webservers
  hosts: web:&winners
  tasks:
    - ping:

- name: Shun poor charlie
  hosts: all:!losers
  tasks:
    - ping:
```

Group membership can use complex pattern syntax.

Limit what will be operated on with the `-l` argument to `ansible-playbook`, or in a play’s `hosts` attribute.

Use the same pattern syntax on the command line or in playbooks.



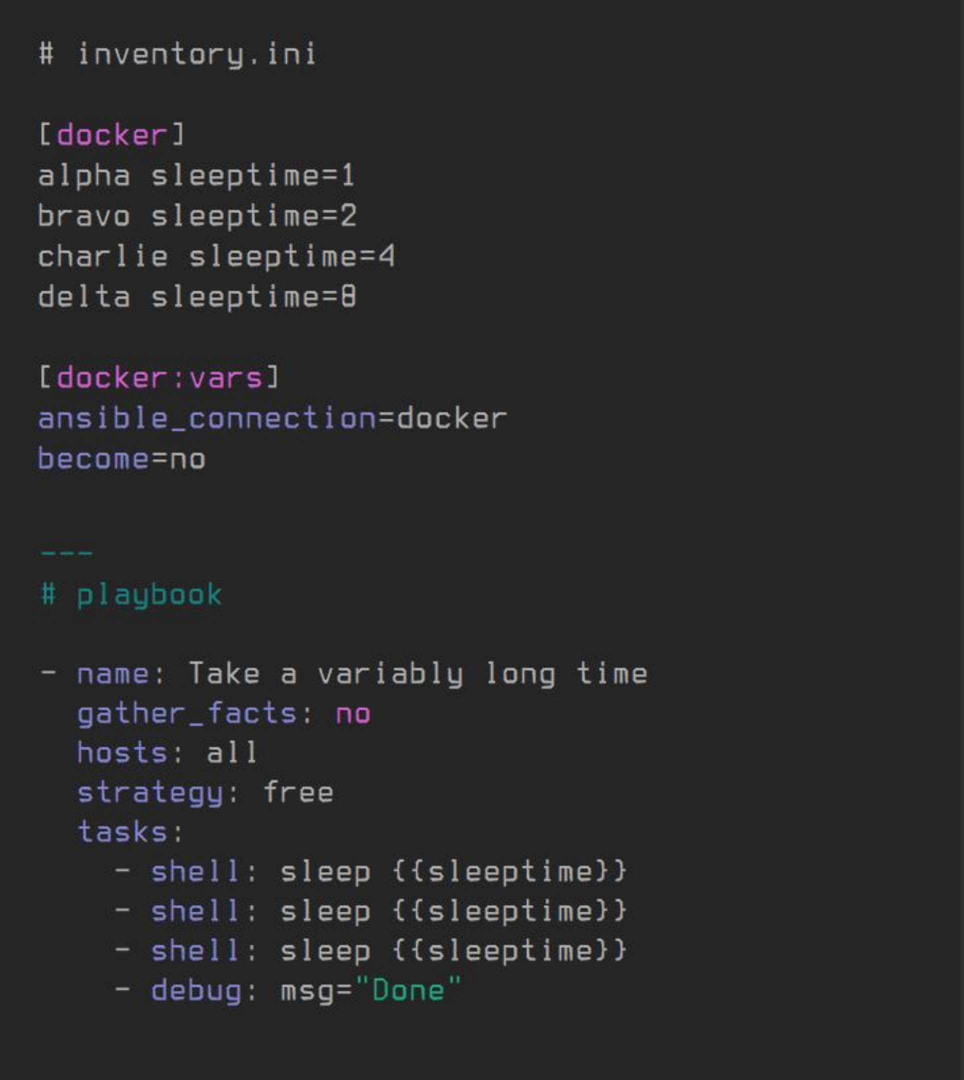
```
(ansible)% ansible-playbook -i inventory.ini ../playbook_ping.yml -l web

PLAY [test] *****************************************************************

TASK [Gathering Facts] ******************************************************
ok: [bravo]
ok: [alpha]

TASK [ping] *****************************************************************
ok: [bravo]
ok: [alpha]

PLAY RECAP ******************************************************************
alpha                      : ok=2    changed=0    unreachable=0    failed=0
bravo                      : ok=2    changed=0    unreachable=0    failed=0
```

Dynamic inventory

Let’s build it.

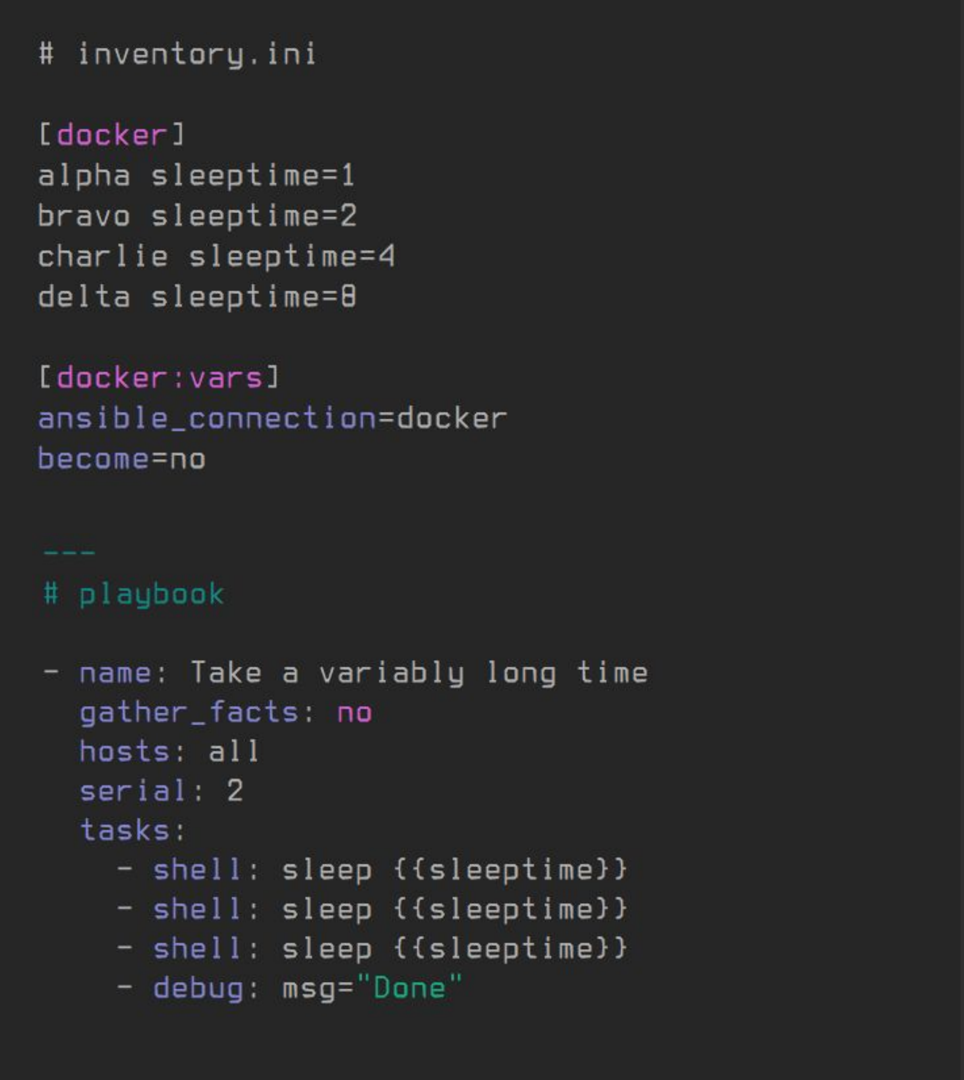
Dynamic inventory: AWS

Scale



```
# inventory.ini

[docker]
alpha sleeptime=1
bravo sleeptime=2
charlie sleeptime=4
delta sleeptime=8

[docker:vars]
ansible_connection=docker
become=no


---
# playbook

- name: Take a variably long time
  gather_facts: no
  hosts: all
  strategy: free
  tasks:
    - shell: sleep {{sleeptime}}
    - shell: sleep {{sleeptime}}
    - shell: sleep {{sleeptime}}
    - debug: msg="Done"
```

Use `strategy: free` to keep hosts from waiting for each other.



```
# inventory.ini

[docker]
alpha sleeptime=1
bravo sleeptime=2
charlie sleeptime=4
delta sleeptime=8

[docker:vars]
ansible_connection=docker
become=no
```

```
---
# playbook

- name: Take a variably long time
  gather_facts: no
  hosts: all
  serial: 2
  tasks:
    - shell: sleep {{sleeptime}}
    - shell: sleep {{sleeptime}}
    - shell: sleep {{sleeptime}}
    - debug: msg="Done"
```

Use the default strategy with the `serial` directive to define batches of hosts.

Melt your laptop.

Set `forks` in `ansible.cfg` to scale the number of remote connections.

Gotchas

Gotcha!
Install & setup

Install your OS’s package.

Scatter stuff all over your machine?

How stale is the version?

Instead, run from a git repo.

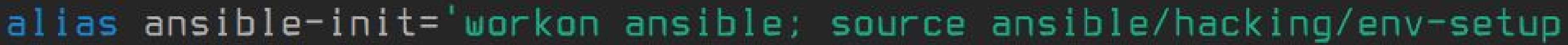
https://github.com/ansible/ansible.git
But don’t commit it to your source control!

Better yet, use a git submodule.

But you’ll need a virtualenv.

http://docs.python-guide.org/en/latest/dev/virtualenvs/

Prefer OS packages for the various virtualenv methods; don’t `pip install` system-wide.



```
alias ansible-init='workon ansible; source ansible/hacking/env-setup
```

Gotcha!
SSH details

Use SSH-specific variables in inventory and on the command line.

`host_key_checking`

`remote_port`

`remote_user`

`ssh_args`

`--extra-vars @aws.var`

Gotcha!
It won’t work?

Can’t auth? Can’t sudo?

Use `-vvvv` to watch the SSH stream.

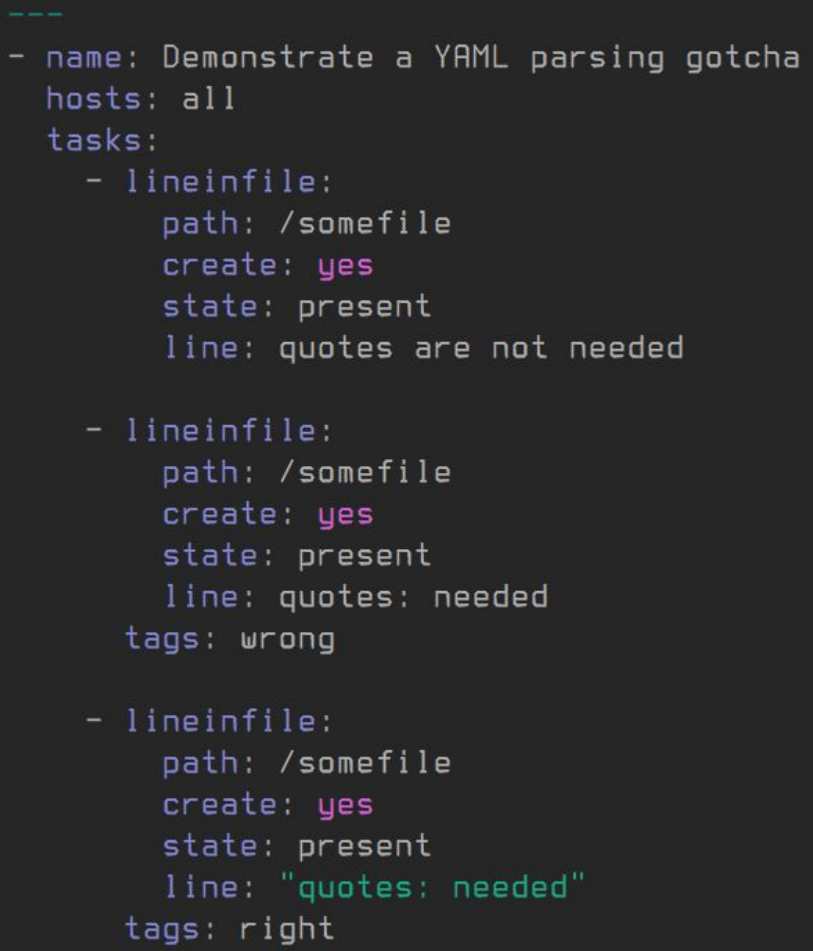
Is SSH doing what you expect?

Gotcha!
So many `.retry`!

`retry_files_enabled`

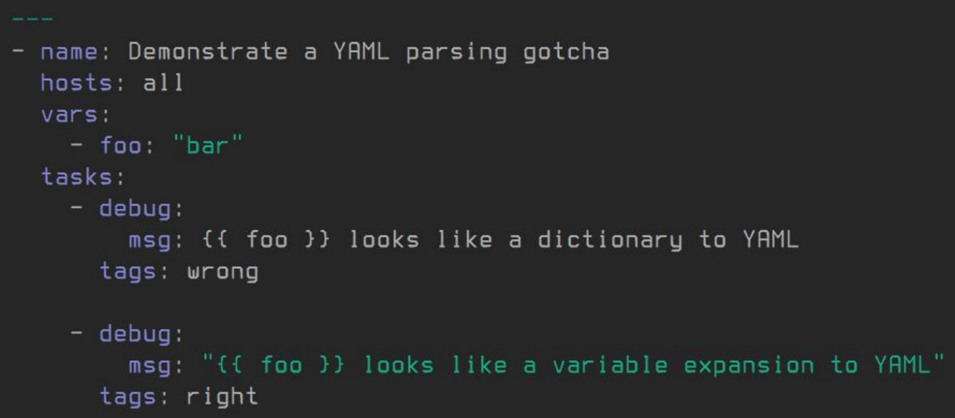
`retry_files_save_path`

Gotcha!
YAML parsing



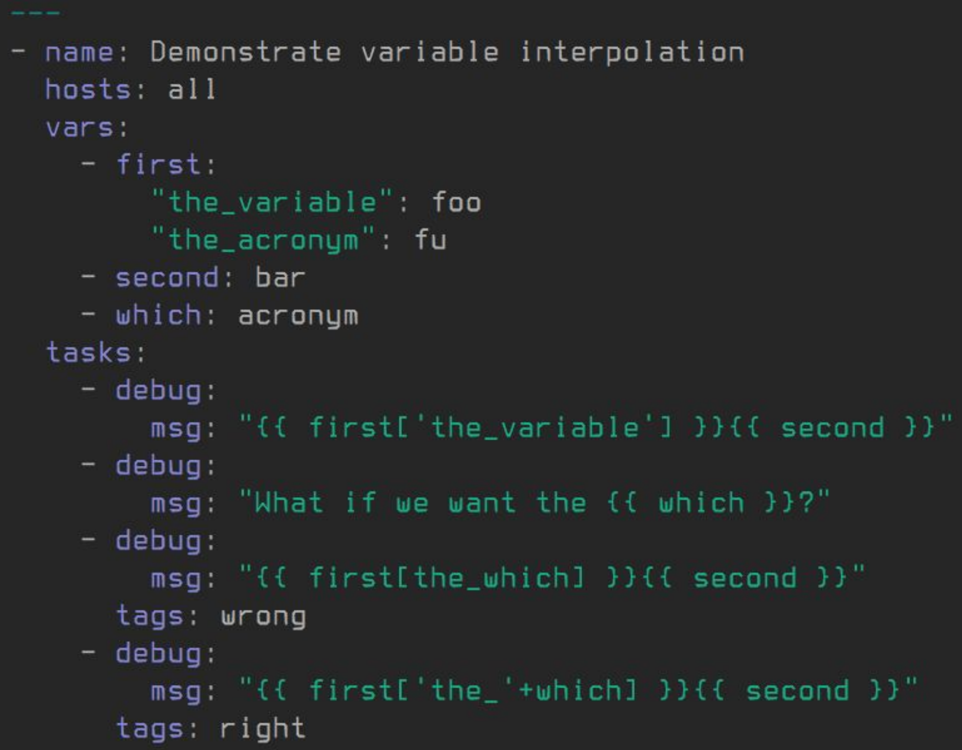
```
---
- name: Demonstrate a YAML parsing gotcha
  hosts: all
  tasks:
    - lineinfile:
        path: /somefile
        create: yes
        state: present
        line: quotes are not needed

    - lineinfile:
        path: /somefile
        create: yes
        state: present
        line: quotes: needed
      tags: wrong

    - lineinfile:
        path: /somefile
        create: yes
        state: present
        line: "quotes: needed"
      tags: right
```

Gotcha!
YAML parsing



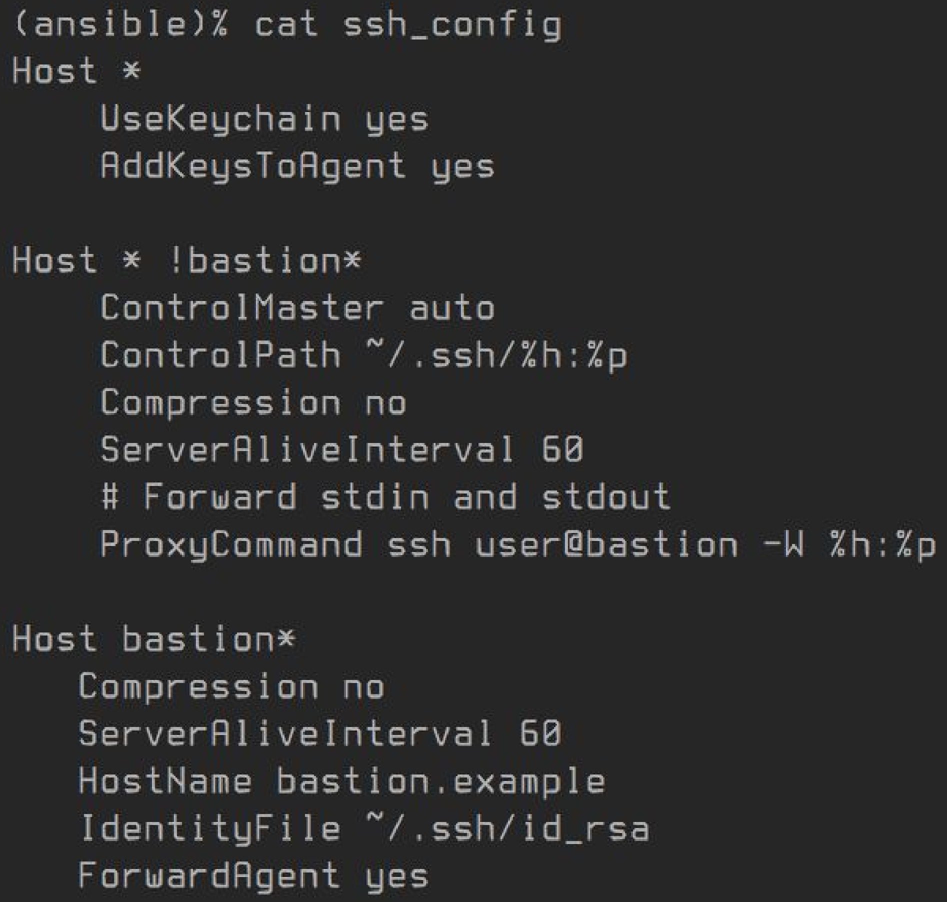
```yaml
---
- name: Demonstrate a YAML parsing gotcha
  hosts: all
  vars:
    - foo: "bar"
  tasks:
    - debug:
        msg: {{ foo }} looks like a dictionary to YAML
      tags: wrong

    - debug:
        msg: "{{ foo }} looks like a variable expansion to YAML"
      tags: right
```

Gotcha! Interpolation



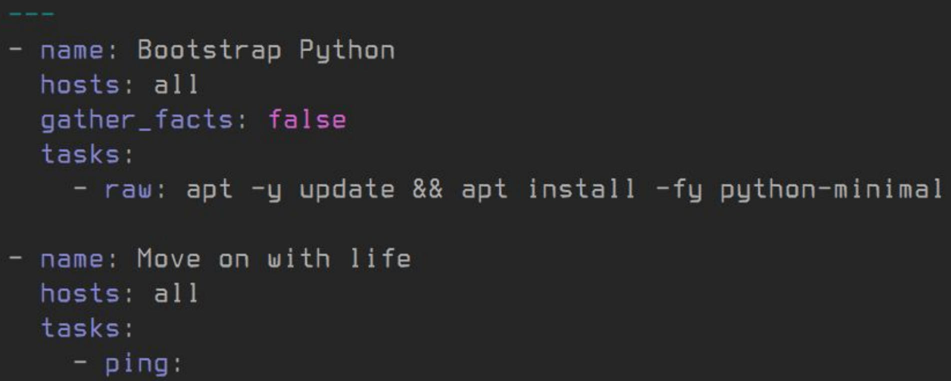
```yaml
---
- name: Demonstrate variable interpolation
  hosts: all
  vars:
    - first:
        "the_variable": foo
        "the_acronym": fu
    - second: bar
    - which: acronym
  tasks:
    - debug:
        msg: "{{ first['the_variable'] }}{{ second }}"
    - debug:
        msg: "What if we want the {{ which }}?"
    - debug:
        msg: "{{ first[the_which] }}{{ second }}"
      tags: wrong
    - debug:
        msg: "{{ first['the_'+which] }}{{ second }}"
      tags: right
```

Gotcha! Firewall



```
(ansible)% cat ssh_config
Host *
    UseKeychain yes
    AddKeysToAgent yes

Host * !bastion*
    ControlMaster auto
    ControlPath ~/.ssh/%h:%p
    Compression no
    ServerAliveInterval 60
    # Forward stdin and stdout
    ProxyCommand ssh user@bastion -W %h:%p

Host bastion*
   Compression no
   ServerAliveInterval 60
   HostName bastion.example
   IdentityFile ~/.ssh/id_rsa
   ForwardAgent yes
```

Gotcha!
No Python!



```yaml
---
- name: Bootstrap Python
  hosts: all
  gather_facts: false
  tasks:
    - raw: apt -y update && apt install -fy python-minimal

- name: Move on with life
  hosts: all
  tasks:
    - ping:
```

Gotcha!
Python3

:-\

Gotcha! GitHub keys

Thou shalt not transport thine public key.

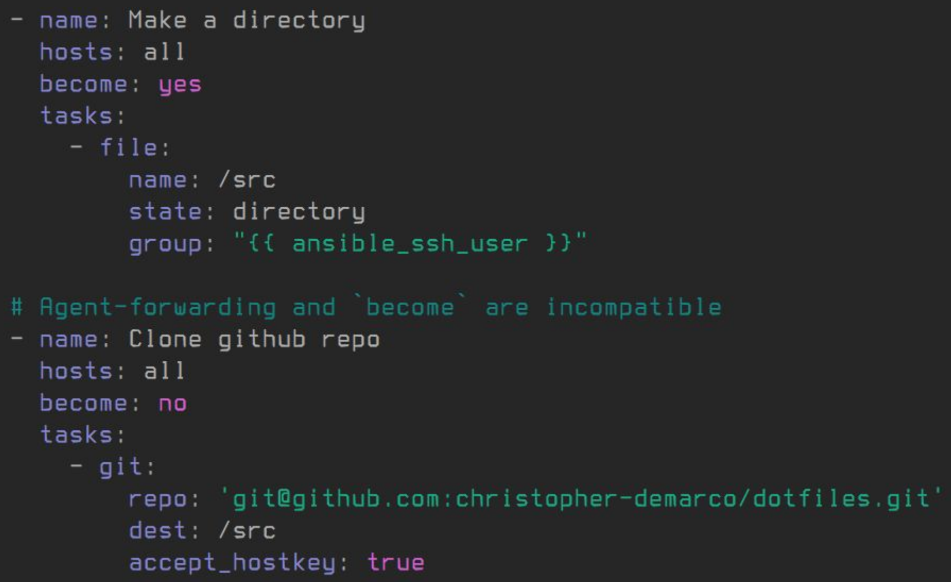
```
% ansible-playbook --ssh-extra-args=-A
```

```
# ansible.cfg
[ssh_connection]
ssh_args=-o ForwardAgent=yes
```

```
# ~/.ssh/config
Host *
    ForwardAgent yes
    UseKeychain yes
    AddKeysToAgent yes
```

```
---
# make sure you're running ssh-agent
# and that you've run ssh-add

- hosts: all
  tasks:
    - name: Test ssh-agent forwarding for github
      command: ssh -T git@github.com
```

Gotcha! GitHub keys (cont.)



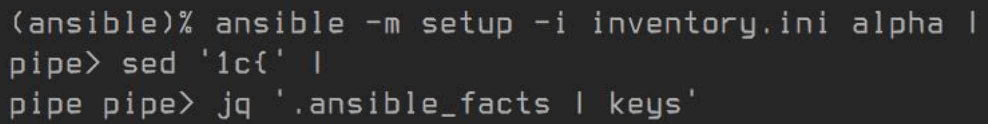
```yaml
- name: Make a directory
  hosts: all
  become: yes
  tasks:
    - file:
        name: /src
        state: directory
        group: "{{ ansible_ssh_user }}"

# Agent-forwarding and `become` are incompatible
- name: Clone github repo
  hosts: all
  become: no
  tasks:
    - git:
        repo: 'git@github.com:christopher-demarco/dotfiles.git'
        dest: /src
        accept_hostkey: true
```

Gotcha! `sudo` password?!

Add the `-K` flag to make Ansible prompt for a sudo password.

Store the `ansible_become_pass` variable in Vault.

Gotcha!
JSON



```
(ansible)% ansible -m setup -i inventory.ini alpha |
pipe> sed '1c{' |
pipe pipe> jq '.ansible_facts | keys'
```

Gotcha! Variable precedence!

Don’t get into a position where you need to know this! KISS!

Defaults are lowest priority: role defaults and inventory vars.

Then facts, play vars, task vars.

Then include_vars.

Then set facts.

Then `-e` extra vars.

Basically, the narrowest scope wins.

Q & A

Thank you.

Lab systems are being destroyed now.
Please fill out your surveys.