Notes for Ansible

Liability Warning

The information contained in this document and the examples are provided “as-is” with no warrantee implied or otherwise about the accuracy or functionality of the examples.

You use them at your own risk. If anything results to your machine or environment as a result of ignoring this warning, then the fault is yours only.

Notes

Ansible works with YAML files that describe the actions to be done. It supports roles, custom modules and playbooks. It runs as an agentless system, pushing changes from the master out to any affected nodes. A host file (called an inventory) – usually “ansible_host” – is used to drive the selection of nodes to be processed. All communication is done using SSH (the default) and a valid SSH connection (with PEM preferably) must exist between the master and slave.

Format of the Host file

The host file is a flat text file that lists the names and/or IP addresses of nodes that are going to be taking part in a deployment. The hosts that are listed are grouped together in groups that are then referred to by the ansible command to drive the provisioning process.

A simple example of the file follows: -

```
cat ./ansible_hosts
[myhosts]
127.0.0.1
localhost
```

This file can also be used to define variables, hostname patterns, specific ports etc. An example of some of these is…

```
cat ./ansible_hosts
[myhosts]
127.0.0.1
localhost

[webservers1-10]
webhost[1:10]

[dbhostsWithDifferentDefaults]
```

```
host1 ansible_connection=ssh ansible_ssh_user=userx
host2:30
host3 ansible_ssh_host=192.168.20.1

[hostsWithVars]
host3 port_var=10 var2=10

[hostGroupWithVars]
hostg1
hostg2
hostg3

[hostGroupWithVars:vars]
var1=10
var2=10
```

Other keys can also be used like [host:children], but I leave these for those who are interested in them.

Playbook format

Playbook is the terminology used to Ansible to describe the deployment logic that will be performed during a run. These playbooks use a YAML format and have simple keywords to describe the actions that should be carried out.

An example of a simple playbook can be found below…

```
---
- hosts: myhosts
  tasks:
  - name: Run a test shell
    shell: echo This is a command
  - name: Install a package
    gem: name=rake state=latest
```

The keys are as follows: -

- Hosts: <hostGroup> - List of hosts in the inventory this process is to apply to
- Tasks – key word used to indicate the start of a list of tasks
- - The '-' represents a list in YAML and is used as a delimiter for tasks and other lists
- name: - Keyword used to indicate a label for the task to do. This name is optional, but highly useful.
- Module/action – The next keyword indicates the module or action to do. This can be many things and relates to the modules that are available. In the example provides above these are – "shell" and "gem".

An example of a playbook with repeating lists is shown below…

```
---
- include: playbook1.yml
- hosts: myhosts
  tasks:
  - name: Run a command
    command: ls -l
    register: outputVar
  - name: Clear dirs
    file: name=/tmp/{{ item }} state=absent
    with_items:
    - dir1
    - dir2
    - dir3
    - dir4
    - dir5
  - name: Make dirs
    file: name=/tmp/{{ item }} state=directory
    with_items:
    - dir1
    - dir2
    - dir3
    - dir4
    - dir5
```

The additional keys are as follows: -

- include: <playbookName> - include another playbook to run
- register: - Used to store the stdout/stderr from a command into a variable for further processing later on
- with_items: - Used to provide a list of "items" that will be processed by that command. In this case the items are substituted where `{{ item }}` is specified

Running Ansible

There are two main ways of running Ansible. The first is to use the `ansible` command that is primarily used to run single commands. The second way is to use ansible-playbook that is primarily used to run a playbook file.

Examples of `ansible` commands are shown below…

```
Rhiannon-mac:playbooks alexgray$ ansible -m ping -k -i
ansible_hosts all
SSH password:
127.0.0.1 | success >> {
    "changed": false,
    "ping": "pong"
}
```

```
localhost | success >> {
    "changed": false,
    "ping": "pong"
}

Rhiannon-mac:playbooks alexgray$ ansible -m command -k -i
ansible_hosts all -a "echo This is a command run"
SSH password:
127.0.0.1 | success | rc=0 >>
This is a command run

localhost | success | rc=0 >>
This is a command run
```

Output is shown in JSON format. The main parameters are: -

- The –m parameter refers to the module to run,
- the –a parameter refers to the command string to run using that module,
- -k refers to asking the SSH password (not using PEM files),
- all refers to running it on all nodes in the inventory file which is specified by –i

Examples of `ansible-playbook` commands are shown below…

```
Rhiannon-mac:playbooks alexgray$ ansible-playbook -i ansible_hosts
playbook1.yml -k
SSH password:

PLAY [myhosts]
****************************************************************

GATHERING FACTS
****************************************************************
ok: [127.0.0.1]
ok: [localhost]

TASK: [Run a test shell]
*****************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Install a package]
*****************************************************
ok: [127.0.0.1]
ok: [127.0.0.1]

PLAY RECAP
********************************************************************
*
127.0.0.1                  : ok=3    changed=1    unreachable=0
failed=0
localhost                  : ok=1    changed=0    unreachable=0
failed=0


Rhiannon-mac:playbooks alexgray$ ansible-playbook -i ansible_hosts
```

```
playbook2.yml -k
SSH password:

PLAY [myhosts]
*********************************************************************

GATHERING FACTS
*********************************************************************
ok: [127.0.0.1]
ok: [localhost]

TASK: [Run a test shell]
*****************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Install a package]
*****************************************************
ok: [127.0.0.1]
ok: [127.0.0.1]

PLAY [myhosts]
*********************************************************************

GATHERING FACTS
*********************************************************************
ok: [localhost]
ok: [127.0.0.1]

TASK: [Run a command]
*****************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Clear dirs]
*********************************************************
ok: [127.0.0.1] => (item=dir1)
ok: [127.0.0.1] => (item=dir1)
ok: [127.0.0.1] => (item=dir2)
ok: [127.0.0.1] => (item=dir2)
ok: [127.0.0.1] => (item=dir3)
ok: [127.0.0.1] => (item=dir3)
ok: [127.0.0.1] => (item=dir4)
ok: [127.0.0.1] => (item=dir4)
ok: [127.0.0.1] => (item=dir5)
ok: [127.0.0.1] => (item=dir5)

TASK: [Make dirs]
*********************************************************
changed: [127.0.0.1] => (item=dir1)
ok: [127.0.0.1] => (item=dir1)
changed: [127.0.0.1] => (item=dir2)
ok: [127.0.0.1] => (item=dir2)
changed: [127.0.0.1] => (item=dir3)
ok: [127.0.0.1] => (item=dir3)
changed: [127.0.0.1] => (item=dir4)
ok: [127.0.0.1] => (item=dir4)
```

```
changed: [127.0.0.1] => (item=dir5)
ok: [127.0.0.1] => (item=dir5)

PLAY RECAP
********************************************************************
*
127.0.0.1                  : ok=7    changed=2    unreachable=0
failed=0
localhost                  : ok=2    changed=0    unreachable=0
failed=0
```

The main phases that are run are: -

- Gathering facts – a phase which gathers facts about the target host
- Task – list of tasks it is running

Common Modules

Assuming you have a valid playbook, the following are a set of common modules that you might want to run to perform tasks.

- *Installing a package can be done by…*

```
- name: install the latest version of Apache
  yum: name=httpd state=latest

- name: remove the Apache package
  yum: name=httpd state=absent

- name: install foo
  apt: name=foo state=present

- name: remove the foo package
  apt: name=foo state=absent
```

- *Starting a service can be done by…*

```
# Example action to start service httpd, if not
running
- service: name=httpd state=started

# Example action to stop service httpd, if running
- service: name=httpd state=stopped

# Example action to restart service httpd, in all
cases
- service: name=httpd state=restarted

# Example action to reload service httpd, in all cases
- service: name=httpd state=reloaded
```

Note – these are missing the name:, but it can be added if wanted

- *Managing a file/directory by…*

```
- file: path=/etc/foo.conf owner=foo group=foo mode=0644
- file: src=/file/to/link/to dest=/path/to/symlink owner=foo
group=foo state=link
- file: src=/tmp/{{ item.path }} dest={{ item.dest }} state=link
  with_items:
    - { path: 'x', dest: 'y' }
    - { path: 'z', dest: 'k' }
```

Note – these are missing the name:, but it can be added if wanted

- *Running a command by…*

```
- name: Run a command
  command: ls -l
  register: outputVar

- name: Run a test shell
  shell: echo This is a command

# You can also use the 'args' form to provide the options. This
command
# will change the working directory to somedir/ and will only run
when
# /path/to/database doesn't exist.
- command: /usr/bin/make_database.sh arg1 arg2
  args:
    chdir: somedir/
    creates: /path/to/database
```

Roles

Roles allow you to create “modules” or “packages” which contain specific files, variables and logic that you can share between playbooks. In terms of Puppet roles are like modules.

You can invoke 1:N roles in your playbook by using the syntax like…

```
---
- hosts: myhosts
  vars:
    hitomi: anime
    silent_mobius: Yuki
  roles:
  -  aRole
```

This will then look for a role directory in the current path and for each role specified, include the logic from `main.yml` file(s) and run the logic in them.

The file structure of roles is like the following…

```
Rhiannon-mac:playbooks alexgray$ ls -laR roles
total 16
drwxr-xr-x  3 alexgray  staff   136 30 Oct 11:14 .
drwxr-xr-x  4 alexgray  staff   408 30 Oct 09:47 ..
-rw-r--r--@ 1 alexgray  staff  6148 28 Oct 16:08 .DS_Store
drwxr-xr-x  9 alexgray  staff   340 30 Oct 09:47 aRole

roles/aRole:
total 24
drwxr-xr-x  9 alexgray  staff   340 30 Oct 09:47 .
drwxr-xr-x  3 alexgray  staff   136 30 Oct 11:14 ..
-rw-r--r--@ 1 alexgray  staff  8196 28 Oct 16:25 .DS_Store
drwxr-xr-x  2 alexgray  staff    68 28 Oct 16:06 defaults
drwxr-xr-x  2 alexgray  staff    68 28 Oct 16:06 files
drwxr-xr-x  2 alexgray  staff    68 28 Oct 16:06 handlers
drwxr-xr-x  2 alexgray  staff    68 28 Oct 16:06 meta
drwxr-xr-x  2 alexgray  staff   102 28 Oct 16:10 tasks
drwxr-xr-x  2 alexgray  staff   102 28 Oct 16:26 templates
drwxr-xr-x  2 alexgray  staff    68 28 Oct 16:06 vars

roles/aRole/defaults:
total 0
drwxr-xr-x  2 alexgray  staff   68 28 Oct 16:06 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..

roles/aRole/files:
total 0
drwxr-xr-x  2 alexgray  staff   68 28 Oct 16:06 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..

roles/aRole/handlers:
total 0
drwxr-xr-x  2 alexgray  staff   68 28 Oct 16:06 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..

roles/aRole/meta:
total 0
drwxr-xr-x  2 alexgray  staff   68 28 Oct 16:06 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..

roles/aRole/tasks:
total 8
drwxr-xr-x  2 alexgray  staff  102 28 Oct 16:10 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..
-rw-r--r--  1 alexgray  staff  178 28 Oct 16:29 main.yml

roles/aRole/templates:
total 8
drwxr-xr-x  2 alexgray  staff  102 28 Oct 16:26 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..
-rw-r--r--  1 alexgray  staff   76 28 Oct 16:31 conf.j2

roles/aRole/vars:
total 0
drwxr-xr-x  2 alexgray  staff   68 28 Oct 16:06 .
drwxr-xr-x  9 alexgray  staff  340 30 Oct 09:47 ..
Rhiannon-mac:playbooks alexgray$
```

The purpose of these directories is as follows: -

- defaults – Default variables
- files – Any static files
- handlers – Any handlers
- meta – Metadata associated with the role
- tasks – The main list of tasks (main.yml)
- templates – Any templates (using Jinga2 syntax)
- vars – Any variable files

An example `main.yml` might be the following…

```
---
- name: Hello-goodbye
  debug: msg="Hello and goodbye"
- name: Install GEM
  gem: name=rake state=latest
- name: Install template
  template: src=conf.j2 dest=/tmp/config.deployed
```

The template file used is…

```
This is a template document for the {{ hitomi }} called {{
silent_mobius }}.
```

…with `{{ hitomi }}` and `{{ silent_mobius }}` being variables defined in a playbook that uses this role.

Custom Modules

Custom modules are a way of allowing you to define/code custom modules (not packages/modules in the Puppet sense) and invoke these from within the playbook. Most commonly, these modules are written in Python, but can be written using any language.

Modules are located by looking for a "library" directory under the current working directory, but can be located from other sources as well. They can be invoked by...

```
---
- hosts: myhosts
  vars:
    hitomi: anime
    silent_mobius: Yuki
  roles:
  - aRole
  tasks:
  - name: Exterminate-1
    action: exterminate.py darlek=the-controller
            exterminate_mode=bychainsaw
    register: exterminatedBy
```

```
  - name: Externed-by
    debug: msg="{{exterminatedBy}}"
  - name: Exterminate-2
    action: exterminate.py darlek=the-controller
            exterminate_mode=bydarlek
    register: exterminatedBy
  - name: Externed-by
    debug: msg="{{exterminatedBy}}"
```

...specifying an action tag as shown in the example above.

For Python, a simple example of a custom (Doctor Who) module is...

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

DOCUMENTATION = '''
---
module: exterminate
short_description: Invoking this module will exterminate you
# Exterminate... Exterminate...
# Exterminate... Exterminate...
# Exterminate... Exterminate...
# Exterminate... Exterminate...
# Exterminate... Exterminate...
# Exterminate... Exterminate...
# Exterminate... Exterminate...
'''

import ConfigParser
import os
import warnings

def main():
    module = AnsibleModule(
            argument_spec = dict(
            darlek=dict(default=None),
            exterminate_mode=dict(default="bydarlek",
                choices=["bydarlek", "bycyberman",
                         "bychainsaw"]),
        )
    )

    creature = module.params["darlek"]
    mode = module.params["exterminate_mode"]
    messE=[]
    if mode in "bydarlek":
      messE.append("killed by a Darlek")
    elif mode in "bycyberman":
      messE.append("killed by a Cyberman")
    else:
      messE.append("killed by an insane chainsaw")

    mess=[]
    mess.append("You have been exterminated " + ''.join(messE))
```

```
    mess.append(" via " + creature)

    if mode in "bychainsaw":
      module.exit_json(msg=mess, changed=False)
    else:
      module.exit_json(msg=mess, changed=True)

from ansible.module_utils.basic import *
main()
```

The parts in bold are the main ones you require for Ansible. The **AnsibleModule** definition allows you to define the parameters coming in which will be different per module. The "**`from ansible.module_utils.basic import * main()`**" is generic and should be included "as is". It is simply present to include Ansible functions.

The **`module.exit_json(msg=mess, changed=False)`** is used to determine what the module returns in the JSON. There are other JSON exit procedures as well, but that is the standard one.

Once the playbook is invoked it will act as a normal module, e.g.

```
Rhiannon-mac:playbooks alexgray$ ansible-playbook -i ansible_hosts
playbook5.yml -k
SSH password:

PLAY [myhosts]
*********************************************************************

GATHERING FACTS
*********************************************************************
ok: [localhost]
ok: [127.0.0.1]

TASK: [aRole | Hello-Goodbye]
*************************************************
ok: [127.0.0.1] => {
    "msg": "Hello and goodbye"
}
ok: [127.0.0.1] => {
    "msg": "Hello and goodbye"
}

TASK: [aRole | Install GEM]
***************************************************
ok: [127.0.0.1]
ok: [127.0.0.1]

TASK: [aRole | Install template]
*****************************************
ok: [127.0.0.1]
ok: [127.0.0.1]

TASK: [Exterminate-1]
*********************************************************
```

```
ok: [127.0.0.1]
ok: [127.0.0.1]

TASK: [Extermined-by]
*********************************************************
ok: [127.0.0.1] => {
    "msg": "{u'msg': [u'You have been exterminated killed by an
insane chainsaw', u' via the-controller'], 'invocation':
{'module_name': u'exterminate.py', 'module_args': u'darlek=the-
controller exterminate_mode=bychainsaw'}, u'changed': False}"
}
ok: [127.0.0.1] => {
    "msg": "{u'msg': [u'You have been exterminated killed by an
insane chainsaw', u' via the-controller'], 'invocation':
{'module_name': u'exterminate.py', 'module_args': u'darlek=the-
controller exterminate_mode=bychainsaw'}, u'changed': False}"
}

TASK: [Exterminate-2]
*********************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Extermined-by]
*********************************************************
ok: [127.0.0.1] => {
    "msg": "{u'msg': [u'You have been exterminated killed by a
Darlek', u' via the-controller'], 'invocation': {'module_name':
u'exterminate.py', 'module_args': u'darlek=the-controller
exterminate_mode=bydarlek'}, u'changed': True}"
}
ok: [127.0.0.1] => {
    "msg": "{u'msg': [u'You have been exterminated killed by a
Darlek', u' via the-controller'], 'invocation': {'module_name':
u'exterminate.py', 'module_args': u'darlek=the-controller
exterminate_mode=bydarlek'}, u'changed': True}"
}

PLAY RECAP
********************************************************************
*
127.0.0.1                  : ok=8    changed=1    unreachable=0
failed=0
localhost                  : ok=1    changed=0    unreachable=0
failed=0
```

Handlers

Handlers are designed to provide something like exception handlers that can be invoked when certain conditions result from a task being run. You can define and invoke many different handlers as shown in the following example.

```
---
- include: playbook1.yml
- hosts: myhosts
  handlers:
```

```
  - name: debug-something
    debug: msg='All is debug...'
  - name: Start-service
    service: state=started name=nginx
  - name: opps-worked
    debug: msg='All is not well'
  tasks:
  - name: Run a command
    command: ls -l
    register: outputVar
  - name: Show something special
    shell: echo "This is a debugging task. What
happens when it ends?"
    notify:
    - debug-something
  - name: Clear dirs
    file: name=/tmp/{{ item }} state=absent
    with_items:
    - dir1
    - dir2
    - dir3
    - dir4
    - dir5
  - name: Make dirs
    file: name=/tmp/{{ item }} state=directory
    with_items:
    - dir1
    - dir2
    - dir3
    - dir4
    - dir5
  - name: Start-service
    service: state=started name=nginx
    register: outputTxt
    ignore_errors: yes
  - name: Print-task
    debug: msg="{{ outputTxt }}"
  - name: It-failed
    debug: msg="It failed"
    when: outputTxt|failed
```

This will invoke the handler “debug-something” sometime after “Show something special” is run.

```
Rhiannon-mac:playbooks alexgray$ ansible-playbook -i ansible_hosts
playbook3.yml -k
SSH password:
```

```
PLAY [myhosts]
*********************************************************************

GATHERING FACTS
*********************************************************************
ok: [127.0.0.1]
ok: [localhost]

TASK: [Run a test shell]
*****************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Install a package]
*****************************************************
ok: [127.0.0.1]
ok: [127.0.0.1]

PLAY [myhosts]
*********************************************************************

GATHERING FACTS
*********************************************************************
ok: [127.0.0.1]
ok: [localhost]

TASK: [Run a command]
*****************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Show something special]
************************************************
changed: [127.0.0.1]
changed: [127.0.0.1]

TASK: [Clear dirs]
************************************************************
changed: [127.0.0.1] => (item=dir1)
ok: [127.0.0.1] => (item=dir1)
changed: [127.0.0.1] => (item=dir2)
ok: [127.0.0.1] => (item=dir2)
changed: [127.0.0.1] => (item=dir3)
ok: [127.0.0.1] => (item=dir3)
changed: [127.0.0.1] => (item=dir4)
ok: [127.0.0.1] => (item=dir4)
changed: [127.0.0.1] => (item=dir5)
ok: [127.0.0.1] => (item=dir5)

TASK: [Make dirs]
************************************************************
changed: [127.0.0.1] => (item=dir1)
ok: [127.0.0.1] => (item=dir1)
changed: [127.0.0.1] => (item=dir2)
ok: [127.0.0.1] => (item=dir2)
changed: [127.0.0.1] => (item=dir3)
ok: [127.0.0.1] => (item=dir3)
```

```
changed: [127.0.0.1] => (item=dir4)
ok: [127.0.0.1] => (item=dir4)
changed: [127.0.0.1] => (item=dir5)
ok: [127.0.0.1] => (item=dir5)

TASK: [Start-service]
*********************************************************
failed: [127.0.0.1] => {"failed": true}
msg: get_service_tools not implemented on target platform
...ignoring
failed: [127.0.0.1] => {"failed": true}
msg: get_service_tools not implemented on target platform
...ignoring

TASK: [Print-task]
*************************************************************
ok: [127.0.0.1] => {
    "msg": "{u'msg': u'get_service_tools not implemented on target
platform', u'failed': True, 'invocation': {'module_name':
u'service', 'module_args': u'state=started name=nginx'}}"
}
ok: [127.0.0.1] => {
    "msg": "{u'msg': u'get_service_tools not implemented on target
platform', u'failed': True, 'invocation': {'module_name':
u'service', 'module_args': u'state=started name=nginx'}}"
}

TASK: [It-failed]
*************************************************************
ok: [127.0.0.1] => {
    "msg": "It failed"
}
ok: [127.0.0.1] => {
    "msg": "It failed"
}

NOTIFIED: [debug-something]
***************************************************
ok: [127.0.0.1] => {
    "msg": "All is debug..."
}
ok: [127.0.0.1] => {
    "msg": "All is debug..."
}

PLAY RECAP
********************************************************************
*
127.0.0.1                  : ok=12   changed=3    unreachable=0
failed=0
localhost                  : ok=2    changed=0    unreachable=0
failed=0
```

Module Search Paths

Module search paths are used to see where modules are. The following shows the set of default module paths. Ansible-doc is used to show documented modules in that search path.

```
% ansible-doc oracle
module oracle not found in
/opt/local/share/ansible:/opt/local/share/ansible/cloud:/opt/loca
l/share/ansible/commands:/opt/local/share/ansible/database:/opt/l
ocal/share/ansible/files:/opt/local/share/ansible/internal:/opt/l
ocal/share/ansible/inventory:/opt/local/share/ansible/messaging:/
opt/local/share/ansible/monitoring:/opt/local/share/ansible/net_i
nfrastructure:/opt/local/share/ansible/network:/opt/local/share/a
nsible/notification:/opt/local/share/ansible/packaging:/opt/local
/share/ansible/source_control:/opt/local/share/ansible/system:/op
t/local/share/ansible/utilities:/opt/local/share/ansible/web_infr
astructure:/opt/local/share/ansible/windows
```