Ansible and BSD

Managing BSD machines is different from managing other Unix-like machines. If you have managed nodes running BSD, review these topics.

Connecting to BSD nodes

Ansible connects to managed nodes using OpenSSH by default. This works on BSD if you use SSH keys for authentication. However, if you use SSH passwords for authentication, Ansible relies on sshpass. Most versions of sshpass do not deal well with BSD login prompts, so when using SSH passwords against BSD machines, use `paramiko` to connect instead of OpenSSH. You can do this in ansible.cfg globally or you can set it as an inventory/group/host variable. For example:

```
[freebsd]
mybsdhost1 ansible_connection=paramiko
```

Bootstrapping BSD

Ansible is agentless by default, however, it requires Python on managed nodes. Only the :ref:`raw <raw_module>` module will operate without Python. Although this module can be used to bootstrap Ansible and install Python on BSD variants (see below), it is very limited and the use of Python is required to make full use of Ansible's features.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 27); *backlink***
>
> Unknown interpreted text role "ref".

The following example installs Python which includes the json library required for full functionality of Ansible. On your control machine you can execute the following for most versions of FreeBSD:

```
ansible -m raw -a "pkg install -y python" mybsdhost1
```

Or for OpenBSD:

```
ansible -m raw -a "pkg_add python%3.8"
```

Once this is done you can now use other Ansible modules apart from the `raw` module.

> **Note**
>
> This example demonstrated using pkg on FreeBSD and pkg_add on OpenBSD, however you should be able to substitute the appropriate package tool for your BSD; the package name may also differ. Refer to the package list or documentation of the BSD variant you are using for the exact Python package name you intend to install.

Setting the Python interpreter

To support a variety of Unix-like operating systems and distributions, Ansible cannot always rely on the existing environment or `env` variables to locate the correct Python binary. By default, modules point at `/usr/bin/python` as this is the most common location. On BSD variants, this path may differ, so it is advised to inform Ansible of the binary's location. See :ref:`INTERPRETER_PYTHON`. For example, set `ansible_python_interpreter` inventory variable:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 52); *backlink***
>
> Unknown interpreted text role "ref".

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python
[openbsd:vars]
ansible_python_interpreter=/usr/local/bin/python3.8
```

FreeBSD packages and ports

In FreeBSD, there is no guarantee that either `/usr/local/bin/python` executable file or a link to an executable file is installed by default. The best practice for a remote host, with respect to Ansible, is to install at least the Python version supported by Ansible, for example, `lang/python38`, and both meta ports `lang/python3` and `lang/python`. Quoting from */usr/ports/lang/python3/pkg-descr*:

```
This is a meta port to the Python 3.x interpreter and provides symbolic links
to bin/python3, bin/pydoc3, bin/idle3 and so on to allow compatibility with
minor version agnostic python scripts.
```

Quoting from */usr/ports/lang/python/pkg-descr*:

```
This is a meta port to the Python interpreter and provides symbolic links
to bin/python, bin/pydoc, bin/idle and so on to allow compatibility with
version agnostic python scripts.
```

As a result, the following packages are installed:

```
shell> pkg info | grep python
python-3.8_3,2                "meta-port" for the default version of Python interpreter
python3-3_3                   Meta-port for the Python interpreter 3.x
python38-3.8.12_1             Interpreted object-oriented programming language
```

and the following executables and links

```
shell> ll /usr/local/bin/ | grep python
lrwxr-xr-x  1 root  wheel       7 Jan 24 08:30 python@ -> python3
lrwxr-xr-x  1 root  wheel      14 Jan 24 08:30 python-config@ -> python3-config
lrwxr-xr-x  1 root  wheel       9 Jan 24 08:29 python3@ -> python3.8
lrwxr-xr-x  1 root  wheel      16 Jan 24 08:29 python3-config@ -> python3.8-config
-r-xr-xr-x  1 root  wheel    5248 Jan 13 01:12 python3.8*
-r-xr-xr-x  1 root  wheel    3153 Jan 13 01:12 python3.8-config*
```

INTERPRETER_PYTHON_FALLBACK

Since version 2.8 Ansible provides a useful variable `ansible_interpreter_python_fallback` to specify a list of paths to search for Python. See :ref:`INTERPRETER_PYTHON_FALLBACK`. This list will be searched and the first item found will be used. For example, the configuration below would make the installation of the meta-ports in the previous section redundant, that is, if you don't install the Python meta ports the first two items in the list will be skipped and `/usr/local/bin/python3.8` will be discovered.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 106); *backlink***
>
> Unknown interpreted text role "ref".

```
ansible_interpreter_python_fallback=['/usr/local/bin/python', '/usr/local/bin/python3', '/usr/local/bin/pytho
```

You can use this variable, prolonged by the lower versions of Python, and put it, for example, into the `group_vars/all`. Then, override it for specific groups in `group_vars/{group1, group2, ...}` and for specific hosts in `host_vars/{host1, host2, ...}` if needed. See :ref:`ansible_variable_precedence`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 113); *backlink***
>
> Unknown interpreted text role "ref".

Debug the discovery of Python

For example, given the inventory

```
shell> cat hosts
[test]
test_11
test_12
test_13

[test:vars]
ansible_connection=ssh
ansible_user=admin
ansible_become=yes
ansible_become_user=root
ansible_become_method=sudo
ansible_interpreter_python_fallback=['/usr/local/bin/python', '/usr/local/bin/python3', '/usr/local/bin/pytho
ansible_perl_interpreter=/usr/local/bin/perl
```

The playbook below

```
shell> cat playbook.yml
- hosts: test_11
  gather_facts: false
```

```
  tasks:
    - command: which python
      register: result
    - debug:
        var: result.stdout
    - debug:
        msg: |-
          {% for i in _vars %}
          {{ i }}:
            {{ lookup('vars', i)|to_nice_yaml|indent(2) }}
          {% endfor %}
      vars:
        _vars: "{{ query('varnames', '.*python.*') }}"
```

displays the details

```
shell> ansible-playbook -i hosts playbook.yml

PLAY [test_11] *****************************************************************************

TASK [command] *****************************************************************************
[WARNING]: Platform freebsd on host test_11 is using the discovered Python interpreter at
/usr/local/bin/python, but future installation of another Python interpreter could change the
meaning of that path. See https://docs.ansible.com/ansible-
core/2.12/reference_appendices/interpreter_discovery.html for more information.
changed: [test_11]

TASK [debug] *******************************************************************************
ok: [test_11] =>
  result.stdout: /usr/local/bin/python

TASK [debug] *******************************************************************************
ok: [test_11] =>
  msg: |-
    ansible_interpreter_python_fallback:
      - /usr/local/bin/python
      - /usr/local/bin/python3
      - /usr/local/bin/python3.8

    discovered_interpreter_python:
      /usr/local/bin/python

    ansible_playbook_python:
      /usr/bin/python3
```

You can see that the first item from the list `ansible_interpreter_python_fallback` was discovered at the FreeBSD remote host. The variable `ansible_playbook_python` keeps the path to Python at the Linux controller that ran the playbook.

Regarding the warning, quoting from :ref:`INTERPRETER_PYTHON`

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 194); *backlink*
>
> Unknown interpreted text role "ref".

```
The fallback behavior will issue a warning that the interpreter
should be set explicitly (since interpreters installed later may
change which one is used). This warning behavior can be disabled by
setting auto_silent or auto_legacy_silent. ...
```

You can either ignore it or get rid of it by setting the variable `ansible_python_interpreter=auto_silent` because this is, actually, what you want by using `/usr/local/bin/python` (*"interpreters installed later may change which one is used"*). For example

```
shell> cat hosts
[test]
test_11
test_12
test_13

[test:vars]
ansible_connection=ssh
ansible_user=admin
ansible_become=yes
ansible_become_user=root
ansible_become_method=sudo
ansible_interpreter_python_fallback=['/usr/local/bin/python', '/usr/local/bin/python3', '/usr/local/bin/pytho
ansible_python_interpreter=auto_silent
ansible_perl_interpreter=/usr/local/bin/perl
```

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 224)
>
> Unknown directive type "seealso".

```
.. seealso::

   * :ref:`interpreter_discovery`
   * `FreeBSD Wiki: Ports/DEFAULT_VERSIONS <https://wiki.freebsd.org/Ports/DEFAULT_VERSIONS>`_
```

Additional variables

If you use additional plugins beyond those bundled with Ansible, you can set similar variables for `bash`, `perl` or `ruby`, depending on how the plugin is written. For example:

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python
ansible_perl_interpreter=/usr/local/bin/perl
```

Which modules are available?

The majority of the core Ansible modules are written for a combination of Unix-like machines and other generic services, so most should function well on the BSDs with the obvious exception of those that are aimed at Linux-only technologies (such as LVG).

Using BSD as the control node

Using BSD as the control machine is as simple as installing the Ansible package for your BSD variant or by following the `pip` or 'from source' instructions.

BSD facts

Ansible gathers facts from the BSDs in a similar manner to Linux machines, but since the data, names and structures can vary for network, disks and other devices, one should expect the output to be slightly different yet still familiar to a BSD administrator.

BSD efforts and contributions

BSD support is important to us at Ansible. Even though the majority of our contributors use and target Linux we have an active BSD community and strive to be as BSD-friendly as possible. Please feel free to report any issues or incompatibilities you discover with BSD; pull requests with an included fix are also welcome!

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)intro_bsd.rst, line 267)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`intro_adhoc`
       Examples of basic commands
   :ref:`working_with_playbooks`
       Learning ansible's configuration management language
   :ref:`developing_modules`
       How to write modules
   `Mailing List <https://groups.google.com/group/ansible-project>`_
       Questions? Help? Ideas?  Stop by the list on Google Groups
   :ref:`communication_irc`
       How to join Ansible chat channels
```