Ansible architecture

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

Being designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

In this section, we'll give you a really quick overview of how Ansible works so you can see how the pieces fit together.

Modules

Ansible works by connecting to your nodes and pushing out scripts called "Ansible modules" to them. Most modules accept parameters that describe the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.

You can :ref:`write your own modules <developing_modules_general>`, though you should first consider :ref:`whether you should <developing_modules>`. Typically you'll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content. You may write specialized modules in any language that can return JSON (Ruby, Python, bash, and so on).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 22); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 22); *backlink*

Unknown interpreted text role "ref".

Module utilities

When multiple modules use the same code, Ansible stores those functions as module utilities to minimize duplication and maintenance. For example, the code that parses URLs is `lib/ansible/module_utils/url.py`. You can :ref:`write your own module utilities <developing_module_utilities>` as well. Module utilities may only be written in Python or in PowerShell.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 27); *backlink*

Unknown interpreted text role "ref".

Plugins

:ref:`Plugins <plugins_lookup>` augment Ansible's core functionality. While modules execute on the target system in separate processes (usually that means on a remote system), plugins execute on the control node within the `/usr/bin/ansible` process. Plugins offer options and extensions for the core features of Ansible - transforming data, logging output, connecting to inventory, and more. Ansible ships with a number of handy plugins, and you can easily :ref:`write your own <developing_plugins>`. For example, you can write an :ref:`inventory plugin <developing_inventory>` to connect to any datasource that returns JSON. Plugins must be written in Python.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 32); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 32); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 32); *backlink*

Unknown interpreted text role "ref".

Inventory

By default, Ansible represents the machines it manages in a file (INI, YAML, and so on) that puts all of your managed machines in groups of your own choosing.

To add new machines, there is no additional SSL signing server involved, so there's never any hassle deciding why a particular machine didn't get linked up due to obscure NTP or DNS issues.

If there's another source of truth in your infrastructure, Ansible can also connect to that. Ansible can draw inventory, group, and variable information from sources like EC2, Rackspace, OpenStack, and more.

Here's what a plain text inventory file looks like:

```
---
[webservers]
www1.example.com
www2.example.com

[dbservers]
db0.example.com
db1.example.com
```

Once inventory hosts are listed, variables can be assigned to them in simple text files (in a subdirectory called 'group_vars/' or 'host_vars/' or directly in the inventory file.

Or, as already mentioned, use a dynamic inventory to pull your inventory from data sources like EC2, Rackspace, or OpenStack.

Playbooks

Playbooks can finely orchestrate multiple slices of your infrastructure topology, with very detailed control over how many machines to tackle at a time. This is where Ansible starts to get most interesting.

Ansible's approach to orchestration is one of finely-tuned simplicity, as we believe your automation code should make perfect sense to you years down the road and there should be very little to remember about special syntax or features.

Here's what a simple playbook looks like:

```
---
- hosts: webservers
  serial: 5 # update 5 machines at a time
  roles:
  - common
  - webapp

- hosts: content_servers
  roles:
  - common
  - content
```

The Ansible search path

Modules, module utilities, plugins, playbooks, and roles can live in multiple locations. If you write your own code to extend Ansible's core features, you may have multiple files with similar or the same names in different locations on your Ansible control node. The search path determines which of these files Ansible will discover and use on any given playbook run.

Ansible's search path grows incrementally over a run. As Ansible finds each playbook and role included in a given run, it appends any directories related to that playbook or role to the search path. Those directories remain in scope for the duration of the run, even after the playbook or role has finished executing. Ansible loads modules, module utilities, and plugins in this order:

1. Directories adjacent to a playbook specified on the command line. If you run Ansible with `ansible-playbook /path/to/play.yml`, Ansible appends these directories if they exist:

```
/path/to/modules
/path/to/module_utils
/path/to/plugins
```

2. Directories adjacent to a playbook that is statically imported by a playbook specified on the command line. If `play.yml` includes `- import_playbook: /path/to/subdir/play1.yml`, Ansible appends these directories if they exist:

```
/path/to/subdir/modules
/path/to/subdir/module_utils
/path/to/subdir/plugins
```

3. Subdirectories of a role directory referenced by a playbook. If `play.yml` runs `myrole`, Ansible appends these directories if they exist:

```
/path/to/roles/myrole/modules
/path/to/roles/myrole/module_utils
/path/to/roles/myrole/plugins
```

4. Directories specified as default paths in `ansible.cfg` or by the related environment variables, including the paths for the various plugin types. See :ref:`ansible_configuration_settings` for more information. Sample `ansible.cfg` fields:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\[ansible-devel][docs][docsite][rst][dev_guide]overview_architecture.rst, line 124); *backlink***
>
> Unknown interpreted text role "ref".

```
DEFAULT_MODULE_PATH
DEFAULT_MODULE_UTILS_PATH
DEFAULT_CACHE_PLUGIN_PATH
DEFAULT_FILTER_PLUGIN_PATH
```

   Sample environment variables:

```
ANSIBLE_LIBRARY
ANSIBLE_MODULE_UTILS
ANSIBLE_CACHE_PLUGINS
ANSIBLE_FILTER_PLUGINS
```

5. The standard directories that ship as part of the Ansible distribution.

> **Caution!**
>
> Modules, module utilities, and plugins in user-specified directories will override the standard versions. This includes some files with generic names. For example, if you have a file named `basic.py` in a user-specified directory, it will override the standard `ansible.module_utils.basic`.
>
> If you have more than one module, module utility, or plugin with the same name in different user-specified directories, the order of commands at the command line and the order of includes and roles in each play will affect which one is found and used on that particular play.