Ansible Documentation

Release 1.4.5

Ansible

February 23, 2014

1 About Ansible **1**
1.1 Introduction . . . 1
1.2 Quickstart Video . . . 30
1.3 Playbooks . . . 30
1.4 Playbooks: Special Topics . . . 76
1.5 About Modules . . . 90
1.6 Module Index . . . 91
1.7 Detailed Guides . . . 393
1.8 Developer Information . . . 415
1.9 Ansible Tower . . . 428
1.10 Community Information . . . 428
1.11 Ansible Galaxy . . . 428
1.12 Frequently Asked Questions . . . 428
1.13 Glossary . . . 433
1.14 YAML Syntax . . . 441
1.15 Ansible Guru . . . 443

CHAPTER

ONE

ABOUT ANSIBLE

Welcome to the Ansible documentation!

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Ansible's goals are foremost those of simplicity and maximum ease of use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with an accelerated socket mode and pull modes as alternatives), and a language that is designed around auditability by humans – even those not familiar with the program.

We believe simplicity is relevant to all sizes of environments and design for busy users of all types – whether this means developers, sysadmins, release engineers, IT managers, and everywhere in between. Ansible is appropriate for managing small setups with a handful of instances as well as enterprise environments with many thousands.

Ansible manages machines in an agentless manner. There is never a question of how to upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled. As OpenSSH is one of the most peer reviewed open source components, the security exposure of using the tool is greatly reduced. Ansible is decentralized – it relies on your existing OS credentials to control access to remote machines; if needed it can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

This documentation covers the current released version of Ansible (1.4.5) and also some development version features (1.5). For recent features, in each section, the version of Ansible where the feature is added is indicated. Ansible, Inc releases a new major release of Ansible approximately every 2 months. The core application evolves somewhat conservatively, valuing simplicity in language design and setup, while the community around new modules and plugins being developed and contributed moves very very quickly, typically adding 20 or so new modules in each release.

1.1 Introduction

Before we dive into the really fun parts – playbooks, configuration management, deployment, and orchestration, we'll learn how to get Ansible installed and some basic concepts. We'll go over how to execute ad-hoc commands in parallel across your nodes using /usr/bin/ansible. We'll also see what sort of modules are available in Ansible's core (though you can also write your own, which we'll also show later).

1.1.1 Installation

Topics

- Installation
  - Getting Ansible
  - Basics / What Will Be Installed
  - What Version To Pick?
  - Control Machine Requirements
  - Managed Node Requirements
  - Installing the Control Machine
    - Running From Source
    - Latest Release Via Yum
    - Latest Releases Via Apt (Ubuntu)
    - Latest Releases Via pkg (FreeBSD)
    - Latest Releases Via Pip
    - Tarballs of Tagged Releases

Getting Ansible

You may also wish to follow the Github project if you have a github account. This is also where we keep the issue tracker for sharing bugs and feature ideas.

Basics / What Will Be Installed

Ansible by default manages machines over the SSH protocol.

Once Ansible is installed, it will not add a database, and there will be no daemons to start or keep running. You only need to install it on one machine (which could easily be a laptop) and it can manage an entire fleet of remote machines from that central point. When Ansible manages remote machines, it does not leave software installed or running on them, so there's no real question about how to upgrade Ansible when moving to a new version.

What Version To Pick?

Because it runs so easily from source and does not require any installation of software on remote machines, many users will actually track the development version.

Ansible's release cycles are usually about two months long. Due to this short release cycle, minor bugs will generally be fixed in the next release versus maintaining backports on the stable branch. Major bugs will still have maintenance releases when needed, though these are infrequent.

If you are wishing to run the latest released version of Ansible and you are running Red Hat Enterprise Linux (TM), CentOS, Fedora, Debian, or Ubuntu, we recommend using the OS package manager.

For other installation options, we recommend installing via "pip", which is the Python package manager, though other options are also available.

If you wish to track the development release to use and test the latest features, we will share information about running from source. It's not necessary to install the program to run from source.

Control Machine Requirements

Currently Ansible can be run from any machine with Python 2.6 installed (Windows isn't supported for the control machine).

This includes Red Hat, Debian, CentOS, OS X, any of the BSDs, and so on.

Managed Node Requirements

On the managed nodes, you only need Python 2.4 or later, but if you are running less than Python 2.5 on the remotes, you will also need:

- `python-simplejson`

Note: Ansible's "raw" module (for executing commands in a quick and dirty way) and the script module don't even need that. So technically, you can use Ansible to install python-simplejson using the raw module, which then allows you to use everything else. (That's jumping ahead though.)

Note: If you have SELinux enabled on remote nodes, you will also want to install libselinux-python on them before using any copy/file/template related functions in Ansible. You can of course still use the yum module in Ansible to install this package on remote systems that do not have it.

Note: Python 3 is a slightly different language than Python 2 and most Python programs (including Ansible) are not switching over yet. However, some Linux distributions (Gentoo, Arch) may not have a Python 2.X interpreter installed by default. On those systems, you should install one, and set the 'ansible_python_interpreter' variable in inventory (see *Inventory*) to point at your 2.X Python. Distributions like Red Hat Enterprise Linux, CentOS, Fedora, and Ubuntu all have a 2.X interpreter installed by default and this does not apply to those distributions. This is also true of nearly all Unix systems. If you need to bootstrap these remote systems by installing Python 2.X, using the 'raw' module will be able to do it remotely.

Installing the Control Machine

Running From Source

Ansible is trivially easy to run from a checkout, root permissions are not required to use it and there is no software to actually install for Ansible itself. No daemons or database setup are required. Because of this, many users in our community use the development version of Ansible all of the time, so they can take advantage of new features when they are implemented, and also easily contribute to the project. Because there is nothing to install, following the development version is significantly easier than most open source projects.

To install from source.

```
$ git clone git://github.com/ansible/ansible.git
$ cd ./ansible
$ source ./hacking/env-setup
```

If you don't have pip installed in your version of Python, install pip:

```
$ sudo easy_install pip
```

Ansible also uses the following Python modules that need to be installed:

```
$ sudo pip install paramiko PyYAML jinja2 httplib2
```

Once running the env-setup script you'll be running from checkout and the default inventory file will be /etc/ansible/hosts. You can optionally specify an inventory file (see *Inventory*) other than /etc/ansible/hosts:

```
$ echo "127.0.0.1" > ~/ansible_hosts
$ export ANSIBLE_HOSTS=~/ansible_hosts
```

You can read more about the inventory file in later parts of the manual.

Now let's test things with a ping command:

```
$ ansible all -m ping --ask-pass
```

You can also use "sudo make install" if you wish.

Latest Release Via Yum

RPMs are available from yum for EPEL 6 and currently supported Fedora distributions.

Ansible itself can manage earlier operating systems that contain Python 2.4 or higher (so also EL5).

Fedora users can install Ansible directly, though if you are using RHEL or CentOS and have not already done so, configure EPEL

```
# install the epel-release RPM if needed on CentOS, RHEL, or Scientific Linux
$ sudo yum install ansible
```

You can also build an RPM yourself. From the root of a checkout or tarball, use the `make rpm` command to build an RPM you can distribute and install. Make sure you have `rpm-build`, `make`, and `python2-devel` installed.

```
$ git clone git://github.com/ansible/ansible.git
$ cd ./ansible
$ make rpm
$ sudo rpm -Uvh ~/rpmbuild/ansible-*.noarch.rpm
```

Latest Releases Via Apt (Ubuntu)

Ubuntu builds are available in a PPA here.

Once configured,

```
$ sudo add-apt-repository ppa:rquillo/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

Debian/Ubuntu packages can also be built from the source checkout, run:

```
$ make deb
```

You may also wish to run from source to get the latest, which is covered above.

Latest Releases Via pkg (FreeBSD)

```
$ sudo pkg install ansible
```

You may also wish to install from ports, run:

```
$ sudo make -C /usr/ports/sysutils/ansible install
```

Latest Releases Via Pip

Ansible can be installed via "pip", the Python package manager. If 'pip' isn't already available in your version of Python, you can get pip by:

```
$ sudo easy_install pip
```

Then install Ansible with:

```
$ sudo pip install ansible
```

Readers that use virtualenv can also install Ansible under virtualenv, though we'd recommend to not worry about it and just install Ansible globally. Do not use easy_install to install ansible directly.

Tarballs of Tagged Releases

Packaging Ansible or wanting to build a local package yourself, but don't want to do a git checkout? Tarballs of releases are available on the Ansible downloads page.

These releases are also tagged in the git repository with the release version.

See also:

Introduction To Ad-Hoc Commands Examples of basic commands

Playbooks Learning ansible's configuration management language

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.1.2 Getting Started

Topics

- Getting Started
  - Foreword
  - Remote Connection Information
  - Your first commands
  - Host Key Checking

Foreword

Now that you've read *Installation* and installed Ansible, it's time to dig in and get started with some commands.

What we are showing first are not the powerful configuration/deployment/orchestration of Ansible, called playbooks. Playbooks are covered in a separate section.

This section is about how to get going initially. Once you have these concepts down, read *Introduction To Ad-Hoc Commands* for some more detail, and then you'll be ready to dive into playbooks and explore the most interesting parts!

Remote Connection Information

Before we get started, it's important to understand how Ansible is communicating with remote machines over SSH.

By default, Ansible 1.3 and later will try to use native OpenSSH for remote communication when possible. This enables both ControlPersist (a performance feature), Kerberos, and options in ~/.ssh/config such as Jump Host setup. When using Enterprise Linux 6 operating systems as the control machine (Red Hat Enterprise Linux and derivatives

such as CentOS), however, the version of OpenSSH may be too old to support ControlPersist. On these operating systems, Ansible will fallback into using a high-quality Python implementation of OpenSSH called 'paramiko'. If you wish to use features like Kerberized SSH and more, consider using Fedora, OS X, or Ubuntu as your control machine until a newer version of OpenSSH is available for your platform – or engage 'accelerated mode' in Ansible. See *Accelerated Mode*.

In Ansible 1.2 and before, the default was strictly paramiko and native SSH had to be explicitly selected with -c ssh or set in the configuration file.

Occasionally you'll encounter a device that doesn't do SFTP. This is rare, but if talking with some remote devices that don't support SFTP, you can switch to SCP mode in *The Ansible Configuration File*.

When speaking with remote machines, Ansible will by default assume you are using SSH keys – which we encourage – but passwords are fine too. To enable password auth, supply the option `--ask-pass` where needed. If using sudo features and when sudo requires a password, also supply `--ask-sudo-pass` as appropriate.

While it may be common sense, it is worth sharing: Any management system benefits from being run near the machines being managed. If running in a cloud, consider running Ansible from a machine inside that cloud. It will work better than on the open internet in most cases.

As an advanced topic, Ansible doesn't just have to connect remotely over SSH. The transports are pluggable, and there are options for managing things locally, as well as managing chroot, lxc, and jail containers. A mode called 'ansible-pull' can also invert the system and have systems 'phone home' via scheduled git checkouts to pull configuration directives from a central repository.

Your first commands

Now that you've installed Ansible, it's time to get started with some basics.

Edit (or create) /etc/ansible/hosts and put one or more remote systems in it, for which you have your SSH key in `authorized_keys`:

```
192.168.1.50
aserver.example.org
bserver.example.org
```

This is an inventory file, which is also explained in greater depth here: *Inventory*.

We'll assume you are using SSH keys for authentication. To set up SSH agent to avoid retyping passwords, you can do:

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

(Depending on your setup, you may wish to use Ansible's `--private-key` option to specify a pem file instead)

Now ping all your nodes:

```
$ ansible all -m ping
```

Ansible will attempt to remote connect to the machines using your current user name, just like SSH would. To override the remote user name, just use the '-u' parameter.

If you would like to access sudo mode, there are also flags to do that:

```
# as bruce
$ ansible all -m ping -u bruce
# as bruce, sudoing to root
$ ansible all -m ping -u bruce --sudo
# as bruce, sudoing to batman
$ ansible all -m ping -u bruce --sudo --sudo-user batman
```

(The sudo implementation is changeable in Ansible's configuration file if you happen to want to use a sudo replacement. Flags passed to sudo (like -H) can also be set there.)

Now run a live command on all of your nodes:

```
$ ansible all -a "/bin/echo hello"
```

Congratulations. You've just contacted your nodes with Ansible. It's soon going to be time to read some of the more real-world *Introduction To Ad-Hoc Commands*, and explore what you can do with different modules, as well as the Ansible *Playbooks* language. Ansible is not just about running commands, it also has powerful configuration management and deployment features. There's more to explore, but you already have a fully working infrastructure!

Host Key Checking

Ansible 1.2.1 and later have host key checking enabled by default.

If a host is reinstalled and has a different key in 'known_hosts', this will result in a error message until corrected. If a host is not initially in 'known_hosts' this will result in prompting for confirmation of the key, which results in a interactive experience if using Ansible, from say, cron. You might not want this.

If you wish to disable this behavior and understand the implications, you can do so by editing /etc/ansible/ansible.cfg or ~/.ansible.cfg:

```
[defaults]
host_key_checking = False
```

Alternatively this can be set by an environment variable:

```
$ export ANSIBLE_HOST_KEY_CHECKING=False
```

Also note that host key checking in paramiko mode is reasonably slow, therefore switching to 'ssh' is also recommended when using this feature. Ansible will log some information about module arguments on the remote system in the remote syslog. To enable basic logging on the control machine see *The Ansible Configuration File* document and set the 'log_path' configuration file setting. Enterprise users may also be interested in *Ansible Tower*. Tower provides a very robust database logging feature where it is possible to drill down and see history based on hosts, projects, and particular inventories over time – explorable both graphically and through a REST API.

See also:

Inventory More information about inventory

Introduction To Ad-Hoc Commands Examples of basic commands

Playbooks Learning Ansible's configuration management language

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.1.3 Inventory

Topics

- Inventory
  - Hosts and Groups
  - Host Variables
  - Group Variables
  - Groups of Groups, and Group Variables
  - Splitting Out Host and Group Specific Data
  - List of Behavioral Inventory Parameters

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

Not only is this inventory configurable, but you can also use multiple inventory files at the same time (explained below) and also pull inventory from dynamic or cloud sources, as described in *Dynamic Inventory*.

Hosts and Groups

The format for /etc/ansible/hosts is an INI format and looks like this:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

The things in brackets are group names, which are used in classifying systems and deciding what systems you are controlling at what times and for what purpose.

It is ok to put systems in more than one group, for instance a server could be both a webserver and a dbserver. If you do, note that variables will come from all of the groups they are a member of, and variable precedence is detailed in a later chapter.

If you have hosts that run on non-standard SSH ports you can put the port number after the hostname with a colon. Ports listed in your SSH config file won't be used, so it is important that you set them if things are not running on the default port:

```
badwolf.example.com:5309
```

Suppose you have just static IPs and want to set up some aliases that don't live in your host file, or you are connecting through tunnels. You can do things like this:

```
jumper ansible_ssh_port=5555 ansible_ssh_host=192.168.1.50
```

In the above example, trying to ansible against the host alias "jumper" (which may not even be a real hostname) will contact 192.168.1.50 on port 5555. Note that this is using a feature of the inventory file to define some special variables. Generally speaking this is not the best way to define variables that describe your system policy, but we'll share suggestions on doing this later. We're just getting started.

Adding a lot of hosts? If you have a lot of hosts following similar patterns you can do this rather than listing each hostname:

```
[webservers]
www[01:50].example.com
```

For numeric patterns, leading zeros can be included or removed, as desired. Ranges are inclusive. You can also define alphabetic ranges:

```
[databases]
db-[a:f].example.com
```

You can also select the connection type and user on a per host basis:

```
[targets]

localhost              ansible_connection=local
other1.example.com     ansible_connection=ssh        ansible_ssh_user=mpdehaan
other2.example.com     ansible_connection=ssh        ansible_ssh_user=mdehaan
```

As mentioned above, setting these in the inventory file is only a shorthand, and we'll discuss how to store them in individual files in the 'host_vars' directory a bit later on.

Host Variables

As alluded to above, it is easy to assign variables to hosts that will be used later in playbooks:

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

Group Variables

Variables can also be applied to an entire group at once:

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

Groups of Groups, and Group Variables

It is also possible to make groups of groups and assign variables to groups. These variables can be used by /usr/bin/ansible-playbook, but not /usr/bin/ansible:

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
```

```
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

If you need to store lists or hash data, or prefer to keep host and group specific variables separate from the inventory file, see the next section.

Splitting Out Host and Group Specific Data

The preferred practice in Ansible is actually not to store variables in the main inventory file.

In addition to the storing variables directly in the INI file, host and group variables can be stored in individual files relative to the inventory file.

These variable files are in YAML format. See *YAML Syntax* if you are new to YAML.

Assuming the inventory file path is:

```
/etc/ansible/hosts
```

If the host is named 'foosball', and in groups 'raleigh' and 'webservers', variables in YAML files at the following locations will be made available to the host:

```
/etc/ansible/group_vars/raleigh
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/foosball
```

For instance, suppose you have hosts grouped by datacenter, and each datacenter uses some different servers. The data in the groupfile '/etc/ansible/group_vars/raleigh' for the 'raleigh' group might look like:

```
---
ntp_server: acme.example.org
database_server: storage.example.org
```

It is ok if these files do not exist, as this is an optional feature.

Tip: In Ansible 1.2 or later the group_vars/ and host_vars/ directories can exist in either the playbook directory OR the inventory directory. If both paths exist, variables in the playbook directory will be loaded second.

Tip: Keeping your inventory file and variables in a git repo (or other version control) is an excellent way to track changes to your inventory and host variables.

List of Behavioral Inventory Parameters

As alluded to above, setting the following variables controls how ansible interacts with remote hosts. Some we have already mentioned:

```
ansible_ssh_host
  The name of the host to connect to, if different from the alias you wish to give to it.
ansible_ssh_port
  The ssh port number, if not 22
ansible_ssh_user
  The default ssh user name to use.
ansible_ssh_pass
  The ssh password to use (this is insecure, we strongly recommend using --ask-pass or SSH keys)
ansible_sudo_pass
  The sudo password to use (this is insecure, we strongly recommend using --ask-sudo-pass)
ansible_connection
  Connection type of the host. Candidates are local, ssh or paramiko.  The default is paramiko before
ansible_ssh_private_key_file
  Private key file used by ssh.  Useful if using multiple keys and you don't want to use SSH agent.
ansible_python_interpreter
  The target host python path. This is useful for systems with more
  than one Python or not located at "/usr/bin/python" such as \*BSD, or where /usr/bin/python
  is not a 2.X series Python.  We do not use the "/usr/bin/env" mechanism as that requires the remote
  path to be set right and also assumes the "python" executable is named python, where the executable
  be named something like "python26".
ansible\_\*\_interpreter
  Works for anything such as ruby or perl and works just like ansible_python_interpreter.
  This replaces shebang of modules which will run on that host.
```

Examples from a host file:

```
some_host         ansible_ssh_port=2222     ansible_ssh_user=manager
aws_host          ansible_ssh_private_key_file=/home/example/.ssh/aws.pem
freebsd_host      ansible_python_interpreter=/usr/local/bin/python
ruby_module_host  ansible_ruby_interpreter=/usr/bin/ruby.1.9.3
```

See also:

Dynamic Inventory Pulling inventory from dynamic sources, such as cloud providers

Introduction To Ad-Hoc Commands Examples of basic commands

Playbooks Learning ansible's configuration management language

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.1.4 Dynamic Inventory

Topics

- Dynamic Inventory
  - Example: The Cobbler External Inventory Script
  - Example: AWS EC2 External Inventory Script
  - Other inventory scripts
  - Using Multiple Inventory Sources

Often a user of a configuration management system will want to keep inventory in a different software system. Ansible provides a basic text-based system as described in *Inventory* but what if you want to use something else?

Frequent examples include pulling inventory from a cloud provider, LDAP, Cobbler, or a piece of expensive enterprisey CMDB software.

Ansible easily supports all of these options via an external inventory system. The plugins directory contains some of these already – including options for EC2/Eucalyptus, Rackspace Cloud, and OpenStack, examples of some of which will be detailed below.

doc:*tower* also provides a database to store inventory results that is both web and REST Accessible. Tower syncs with all Ansible dynamic inventory sources you might be using, and also includes a graphical inventory editor. By having a database record of all of your hosts, it's easy to correlate past event history and see which ones have had failures on their last playbook runs.

For information about writing your own dynamic inventory source, see *Developing Dynamic Inventory Sources*.

Example: The Cobbler External Inventory Script

It is expected that many Ansible users with a reasonable amount of physical hardware may also be Cobbler users. (note: Cobbler was originally written by Michael DeHaan and is now lead by James Cammarata, who also works for Ansible, Inc).

While primarily used to kickoff OS installations and manage DHCP and DNS, Cobbler has a generic layer that allows it to represent data for multiple configuration management systems (even at the same time), and has been referred to as a 'lightweight CMDB' by some admins. This particular script will communicate with Cobbler using Cobbler's XMLRPC API.

To tie Ansible's inventory to Cobbler (optional), copy this script to /etc/ansible and *chmod +x* the file. cobblerd will now need to be running when you are using Ansible and you'll need to use Ansible's `-i` command line option (e.g. `-i /etc/ansible/cobbler.py`).

First test the script by running `/etc/ansible/cobbler.py` directly. You should see some JSON data output, but it may not have anything in it just yet.

Let's explore what this does. In cobbler, assume a scenario somewhat like the following:

```
cobbler profile add --name=webserver --distro=CentOS6-x86_64
cobbler profile edit --name=webserver --mgmt-classes="webserver" --ksmeta="a=2 b=3"
cobbler system edit --name=foo --dns-name="foo.example.com" --mgmt-classes="atlanta" --ksmeta="c=4"
cobbler system edit --name=bar --dns-name="bar.example.com" --mgmt-classes="atlanta" --ksmeta="c=5"
```

In the example above, the system 'foo.example.com' will be addressable by ansible directly, but will also be addressable when using the group names 'webserver' or 'atlanta'. Since Ansible uses SSH, we'll try to contract system foo over 'foo.example.com', only, never just 'foo'. Similarly, if you try "ansible foo" it wouldn't find the system... but "ansible 'foo*"' would, because the system DNS name starts with 'foo'.

The script doesn't just provide host and group info. In addition, as a bonus, when the 'setup' module is run (which happens automatically when using playbooks), the variables 'a', 'b', and 'c' will all be auto-populated in the templates:

```
# file: /srv/motd.j2
Welcome, I am templated with a value of a={{ a }}, b={{ b }}, and c={{ c }}
```

Which could be executed just like this:

```
ansible webserver -m setup
ansible webserver -m template -a "src=/tmp/motd.j2 dest=/etc/motd"
```

Note: The name 'webserver' came from cobbler, as did the variables for the config file. You can still pass in your own variables like normal in Ansible, but variables from the external inventory script will override any that have the same name.

So, with the template above (motd.j2), this would result in the following data being written to /etc/motd for system 'foo':

```
Welcome, I am templated with a value of a=2, b=3, and c=4
```

And on system 'bar' (bar.example.com):

```
Welcome, I am templated with a value of a=2, b=3, and c=5
```

And technically, though there is no major good reason to do it, this also works too:

```
ansible webserver -m shell -a "echo {{ a }}"
```

So in other words, you can use those variables in arguments/actions as well.

Example: AWS EC2 External Inventory Script

If you use Amazon Web Services EC2, maintaining an inventory file might not be the best approach, because hosts may come and go over time, be managed by external applications, or you might even be using AWS autoscaling. For this reason, you can use the EC2 external inventory script.

You can use this script in one of two ways. The easiest is to use Ansible's `-i` command line option and specify the path to the script after marking it executable:

```
ansible -i ec2.py -u ubuntu us-east-1d -m ping
```

The second option is to copy the script to */etc/ansible/hosts* and *chmod +x* it. You will also need to copy the ec2.ini file to */etc/ansible/ec2.ini*. Then you can run ansible as you would normally.

To successfully make an API call to AWS, you will need to configure Boto (the Python interface to AWS). There are a variety of methods available, but the simplest is just to export two environment variables:

```
export AWS_ACCESS_KEY_ID='AK123'
export AWS_SECRET_ACCESS_KEY='abc123'
```

You can test the script by itself to make sure your config is correct:

```
cd plugins/inventory
./ec2.py --list
```

After a few moments, you should see your entire EC2 inventory across all regions in JSON.

Since each region requires its own API call, if you are only using a small set of regions, feel free to edit `ec2.ini` and list only the regions you are interested in. There are other config options in `ec2.ini` including cache control, and destination variables.

At their heart, inventory files are simply a mapping from some name to a destination address. The default `ec2.ini` settings are configured for running Ansible from outside EC2 (from your laptop for example) – and this is not the most efficient way to manage EC2.

If you are running Ansible from within EC2, internal DNS names and IP addresses may make more sense than public DNS names. In this case, you can modify the `destination_variable` in `ec2.ini` to be the private DNS name of an instance. This is particularly important when running Ansible within a private subnet inside a VPC, where the only way to access an instance is via its private IP address. For VPC instances, *vpc_destination_variable* in `ec2.ini` provides a means of using which ever boto.ec2.instance variable makes the most sense for your use case.

The EC2 external inventory provides mappings to instances from several groups:

Instance ID These are groups of one since instance IDs are unique. e.g. `i-00112233` `i-a1b1c1d1`

Region A group of all instances in an AWS region. e.g. `us-east-1` `us-west-2`

Availability Zone A group of all instances in an availability zone. e.g. `us-east-1a` `us-east-1b`

Security Group Instances belong to one or more security groups. A group is created for each security group, with all characters except alphanumerics, dashes (-) converted to underscores (_). Each group is prefixed by `security_group_` e.g. `security_group_default security_group_webservers security_group_Pete_s_Fancy_Group`

Tags Each instance can have a variety of key/value pairs associated with it called Tags. The most common tag key is 'Name', though anything is possible. Each key/value pair is its own group of instances, again with special characters converted to underscores, in the format `tag_KEY_VALUE` e.g. `tag_Name_Web tag_Name_redis-master-001 tag_aws_cloudformation_logical-id_WebServerGroup`

When the Ansible is interacting with a specific server, the EC2 inventory script is called again with the `--host HOST` option. This looks up the HOST in the index cache to get the instance ID, and then makes an API call to AWS to get information about that specific instance. It then makes information about that instance available as variables to your playbooks. Each variable is prefixed by `ec2_`. Here are some of the variables available:

- ec2_architecture
- ec2_description
- ec2_dns_name
- ec2_id
- ec2_image_id
- ec2_instance_type
- ec2_ip_address
- ec2_kernel
- ec2_key_name
- ec2_launch_time
- ec2_monitored
- ec2_ownerId
- ec2_placement
- ec2_platform
- ec2_previous_state
- ec2_private_dns_name
- ec2_private_ip_address
- ec2_public_dns_name
- ec2_ramdisk
- ec2_region
- ec2_root_device_name
- ec2_root_device_type
- ec2_security_group_ids
- ec2_security_group_names
- ec2_spot_instance_request_id
- ec2_state
- ec2_state_code

- ec2_state_reason
- ec2_status
- ec2_subnet_id
- ec2_tag_Name
- ec2_tenancy
- ec2_virtualization_type
- ec2_vpc_id

Both `ec2_security_group_ids` and `ec2_security_group_names` are comma-separated lists of all security groups. Each EC2 tag is a variable in the format `ec2_tag_KEY`.

To see the complete list of variables available for an instance, run the script by itself:

```
cd plugins/inventory
./ec2.py --host ec2-12-12-12-12.compute-1.amazonaws.com
```

Note that the AWS inventory script will cache results to avoid repeated API calls, and this cache setting is configurable in ec2.ini. To explicitly clear the cache, you can run the ec2.py script with the `--refresh-cache` parameter.

Other inventory scripts

In addition to Cobbler and EC2, inventory scripts are also available for:

```
BSD Jails
Digital Ocean
Linode
OpenShift
OpenStack Nova
Red Hat's SpaceWalk
Vagrant (not to be confused with the provisioner in vagrant, which is preferred)
Zabbix
```

Sections on how to use these in more detail will be added over time, but by looking at the "plugins/" directory of the Ansible checkout it should be very obvious how to use them. The process for the AWS inventory script is the same.

If you develop an interesting inventory script that might be general purpose, please submit a pull request – we'd likely be glad to include it in the project.

Using Multiple Inventory Sources

If the location given to -i in Ansible is a directory (or as so configured in ansible.cfg), Ansible can use multiple inventory sources at the same time. When doing so, it is possible to mix both dynamic and statically managed inventory sources in the same ansible run. Instant hybrid cloud!

See also:

Inventory All about static inventory files

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.1.5 Patterns

Topics

- Patterns

Patterns in Ansible are how we decide which hosts to manage. This can mean what hosts to communicate with, but in terms of *Playbooks* it actually means what hosts to apply a particular configuration or IT process to.

We'll go over how to use the command line in *Introduction To Ad-Hoc Commands* section, however, basically it looks like this:

```
ansible <pattern_goes_here> -m <module_name> -a <arguments>
```

Such as:

```
ansible webservers -m service -a "name=httpd state=restarted"
```

A pattern usually refers to a set of groups (which are sets of hosts) – in the above case, machines in the "webservers" group.

Anyway, to use Ansible, you'll first need to know how to tell Ansible which hosts in your inventory to talk to. This is done by designating particular host names or groups of hosts.

The following patterns are equivalent and target all hosts in the inventory:

```
all
*
```

It is also possible to address a specific host or set of hosts by name:

```
one.example.com
one.example.com:two.example.com
192.168.1.50
192.168.1.*
```

The following patterns address one or more groups. Groups separated by a colon indicate an "OR" configuration. This means the host may be in either one group or the other:

```
webservers
webservers:dbservers
```

You can exclude groups as well, for instance, all machines must be in the group webservers but not in the group phoenix:

```
webservers:!phoenix
```

You can also specify the intersection of two groups. This would mean the hosts must be in the group webservers and the host must also be in the group staging:

```
webservers:&staging
```

You can do combinations:

```
webservers:dbservers:&staging:!phoenix
```

The above configuration means "all machines in the groups 'webservers' and 'dbservers' are to be managed if they are in the group 'staging' also, but the machines are not to be managed if they are in the group 'phoenix' ... whew!

You can also use variables if you want to pass some group specifiers via the "-e" argument to ansible-playbook, but this is uncommonly used:

```
webservers:!{{excluded}}:&{{required}}
```

You also don't have to manage by strictly defined groups. Individual host names, IPs and groups, can also be referenced using wildcards:

```
*.example.com
*.com
```

It's also ok to mix wildcard patterns and groups at the same time:

```
one*.com:dbservers
```

Most people don't specify patterns as regular expressions, but you can. Just start the pattern with a '~':

```
~(web|db).*\.example\.com
```

While we're jumping a bit ahead, additionally, you can add an exclusion criteria just by supplying the `--limit` flag to /usr/bin/ansible or /usr/bin/ansible-playbook:

```
ansible-playbook site.yml --limit datacenter2
```

Easy enough. See *Introduction To Ad-Hoc Commands* and then *Playbooks* for how to apply this knowledge.

See also:

Introduction To Ad-Hoc Commands Examples of basic commands

Playbooks Learning ansible's configuration management language

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.1.6 Introduction To Ad-Hoc Commands

Topics

- Introduction To Ad-Hoc Commands
  - Parallelism and Shell Commands
  - File Transfer
  - Managing Packages
  - Users and Groups
  - Deploying From Source Control
  - Managing Services
  - Time Limited Background Operations
  - Gathering Facts

The following examples show how to use */usr/bin/ansible* for running ad hoc tasks.

What's an ad-hoc command?

An ad-hoc command is something that you might type in to do something really quick, but don't want to save for later.

This is a good place to start to understand the basics of what Ansible can do prior to learning the playbooks language – ad-hoc commands can also be used to do quick things that you might not necessarily want to write a full playbook for.

Generally speaking, the true power of Ansible lies in playbooks. Why would you use ad-hoc tasks versus playbooks?

For instance, if you wanted to power off all of your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook.

For configuration management and deployments, though, you'll want to pick up on using '/usr/bin/ansible-playbook' – the concepts you will learn here will port over directly to the playbook language.

(See *Playbooks* for more information about those)

If you haven't read *Inventory* already, please look that over a bit first and then we'll get going.

Parallelism and Shell Commands

Arbitrary example.

Let's use Ansible's command line tool to reboot all web servers in Atlanta, 10 at a time. First, let's set up SSH-agent so it can remember our credentials:

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

If you don't want to use ssh-agent and want to instead SSH with a password instead of keys, you can with `--ask-pass` (`-k`), but it's much better to just use ssh-agent.

Now to run the command on all servers in a group, in this case, *atlanta*, in 10 parallel forks:

```
$ ansible atlanta -a "/sbin/reboot" -f 10
```

/usr/bin/ansible will default to running from your user account. If you do not like this behavior, pass in "-u username". If you want to run commands as a different user, it looks like this:

```
$ ansible atlanta -a "/usr/bin/foo" -u username
```

Often you'll not want to just do things from your user account. If you want to run commands through sudo:

```
$ ansible atlanta -a "/usr/bin/foo" -u username --sudo [--ask-sudo-pass]
```

Use `--ask-sudo-pass` (`-K`) if you are not using passwordless sudo. This will interactively prompt you for the password to use. Use of passwordless sudo makes things easier to automate, but it's not required.

It is also possible to sudo to a user other than root using `--sudo-user` (`-U`):

```
$ ansible atlanta -a "/usr/bin/foo" -u username -U otheruser [--ask-sudo-pass]
```

Note: Rarely, some users have security rules where they constrain their sudo environment to running specific command paths only. This does not work with ansible's no-bootstrapping philosophy and hundreds of different modules. If doing this, use Ansible from a special account that does not have this constraint. One way of doing this without sharing access to unauthorized users would be gating Ansible with *Ansible Tower*, which can hold on to an SSH credential and let members of certain organizations use it on their behalf without having direct access.

Ok, so those are basics. If you didn't read about patterns and groups yet, go back and read *Patterns*.

The `-f 10` in the above specifies the usage of 10 simultaneous processes to use. You can also set this in *The Ansible Configuration File* to avoid setting it again. The default is actually 5, which is really small and conservative. You are probably going to want to talk to a lot more simultaneous hosts so feel free to crank this up. If you have more hosts than the value set for the fork count, Ansible will talk to them, but it will take a little longer. Feel free to push this value as high as your system can handle it!

You can also select what Ansible "module" you want to run. Normally commands also take a `-m` for module name, but the default module name is 'command', so we didn't need to specify that all of the time. We'll use `-m` in later examples to run some other *About Modules*.

Note: The *command - Executes a command on a remote node* module does not support shell variables and things like piping. If we want to execute a module using a shell, use the 'shell' module instead. Read more about the differences on the *About Modules* page.

Using the *shell - Execute commands in nodes.* module looks like this:

```
$ ansible raleigh -m shell -a 'echo $TERM'
```

When running any command with the Ansible *ad hoc* CLI (as opposed to *Playbooks*), pay particular attention to shell quoting rules, so the local shell doesn't eat a variable before it gets passed to Ansible. For example, using double vs single quotes in the above example would evaluate the variable on the box you were on.

So far we've been demoing simple command execution, but most Ansible modules usually do not work like simple scripts. They make the remote system look like you state, and run the commands necessary to get it there. This is commonly referred to as 'idempotence', and is a core design goal of Ansible. However, we also recognize that running arbitrary commands is equally important, so Ansible easily supports both.

File Transfer

Here's another use case for the */usr/bin/ansible* command line. Ansible can SCP lots of files to multiple machines in parallel.

To transfer a file directly to many different servers:

```
$ ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

If you use playbooks, you can also take advantage of the `template` module, which takes this another step further. (See module and playbook documentation).

The `file` module allows changing ownership and permissions on files. These same options can be passed directly to the `copy` module as well:

```
$ ansible webservers -m file -a "dest=/srv/foo/a.txt mode=600"
$ ansible webservers -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan group=mdehaan"
```

The `file` module can also create directories, similar to `mkdir -p`:

```
$ ansible webservers -m file -a "dest=/path/to/c mode=755 owner=mdehaan group=mdehaan state=directory
```

As well as delete directories (recursively) and delete files:

```
$ ansible webservers -m file -a "dest=/path/to/c state=absent"
```

Managing Packages

There are modules available for yum and apt. Here are some examples with yum.

Ensure a package is installed, but don't update it:

```
$ ansible webservers -m yum -a "name=acme state=installed"
```

Ensure a package is installed to a specific version:

```
$ ansible webservers -m yum -a "name=acme-1.5 state=installed"
```

Ensure a package is at the latest version:

```
$ ansible webservers -m yum -a "name=acme state=latest"
```

Ensure a package is not installed:

```
$ ansible webservers -m yum -a "name=acme state=removed"
```

Ansible has modules for managing packages under many platforms. If your package manager does not have a module available for it, you can install for other packages using the command module or (better!) contribute a module for other package managers. Stop by the mailing list for info/details.

Users and Groups

The 'user' module allows easy creation and manipulation of existing user accounts, as well as removal of user accounts that may exist:

```
$ ansible all -m user -a "name=foo password=<crypted password here>"
```

```
$ ansible all -m user -a "name=foo state=absent"
```

See the *About Modules* section for details on all of the available options, including how to manipulate groups and group membership.

Deploying From Source Control

Deploy your webapp straight from git:

```
$ ansible webservers -m git -a "repo=git://foo.example.org/repo.git dest=/srv/myapp version=HEAD"
```

Since Ansible modules can notify change handlers it is possible to tell Ansible to run specific tasks when the code is updated, such as deploying Perl/Python/PHP/Ruby directly from git and then restarting apache.

Managing Services

Ensure a service is started on all webservers:

```
$ ansible webservers -m service -a "name=httpd state=started"
```

Alternatively, restart a service on all webservers:

```
$ ansible webservers -m service -a "name=httpd state=restarted"
```

Ensure a service is stopped:

```
$ ansible webservers -m service -a "name=httpd state=stopped"
```

Time Limited Background Operations

Long running operations can be backgrounded, and their status can be checked on later. The same job ID is given to the same task on all hosts, so you won't lose track. If you kick hosts and don't want to poll, it looks like this:

```
$ ansible all -B 3600 -a "/usr/bin/long_running_operation --do-stuff"
```

If you do decide you want to check on the job status later, you can:

```
$ ansible all -m async_status -a "jid=123456789"
```

Polling is built-in and looks like this:

```
$ ansible all -B 1800 -P 60 -a "/usr/bin/long_running_operation --do-stuff"
```

The above example says "run for 30 minutes max (`-B`: 30*60=1800), poll for status (`-P`) every 60 seconds".

Poll mode is smart so all jobs will be started before polling will begin on any machine. Be sure to use a high enough `--forks` value if you want to get all of your jobs started very quickly. After the time limit (in seconds) runs out (`-B`), the process on the remote nodes will be terminated.

Typically you'll be only be backgrounding long-running shell commands or software upgrades only. Backgrounding the copy module does not do a background file transfer. *Playbooks* also support polling, and have a simplified syntax for this.

Gathering Facts

Facts are described in the playbooks section and represent discovered variables about a system. These can be used to implement conditional execution of tasks but also just to get ad-hoc information about your system. You can see all facts via:

```
$ ansible all -m setup
```

Its also possible to filter this output to just export certain facts, see the "setup" module documentation for details.

Read more about facts at *Variables* once you're ready to read up on *Playbooks*.

See also:

The Ansible Configuration File All about the Ansible config file

About Modules A list of available modules

Playbooks Using Ansible for configuration management & deployment

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.1.7 The Ansible Configuration File

Topics

- The Ansible Configuration File
  - Getting the latest configuration
  - Environmental configuration
  - Explanation of values by section
    - General defaults
      - action_plugins
      - ansible_managed
      - ask_pass
      - ask_sudo_pass
      - callback_plugins
      - connection_plugins
      - deprecation_warnings
      - display_skipped_hosts
      - error_on_undefined_vars
      - executable
      - filter_plugins
      - forks
      - hash_behaviour
      - hostfile
      - host_key_checking
      - jinja2_extensions
      - legacy_playbook_variables
      - library
      - log_path
      - lookup_plugins
      - module_name
      - nocolor
      - nocows
      - pattern
      - poll_interval
      - private_key_file
      - remote_port
      - remote_tmp
      - remote_user
      - roles_path
      - sudo_exe
      - sudo_flags
      - sudo_user
      - timeout
      - transport
      - vars_plugins
    - Paramiko Specific Settings
      - record_host_keys
    - OpenSSH Specific Settings
      - ssh_args
      - control_path
      - scp_if_ssh
      - pipelining
    - Accelerate Mode Settings
      - accelerate_port
      - accelerate_timeout
      - accelerate_connect_timeout

Certain settings in Ansible are adjustable via a configuration file. The stock configuration should be sufficient for most users, but there may be reasons you would want to change them.

Changes can be made and used in a configuration file which will be processed processed in the following order:

```
* ANSIBLE_CONFIG (an environment variable)
* ansible.cfg (in the current directory)
* .ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg
```

Prior to 1.5 the order was:

```
* ansible.cfg (in the current directory)
* ANSIBLE_CONFIG (an environment variable)
* .ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg
```

Ansible will process the above list and use the first file found. Settings in files are not merged together.

Getting the latest configuration

If installing ansible from a package manager, the latest ansible.cfg should be present in /etc/ansible, possibly as a ".rpmnew" file (or other) as appropriate in the case of updates.

If you have installed from pip or from source, however, you may want to create this file in order to override default settings in Ansible.

You may wish to consult the ansible.cfg in source control for all of the possible latest values.

Environmental configuration

Ansible also allows configuration of settings via environment variables. If these environment variables are set, they will override any setting loaded from the configuration file. These variables are for brevity not defined here, but look in 'constants.py' in the source tree if you want to use these. They are mostly considered to be a legacy system as compared to the config file, but are equally valid.

Explanation of values by section

The configuration file is broken up into sections. Most options are in the "general" section but some sections of the file are specific to certain connection types.

General defaults

In the [defaults] section of ansible.cfg, the following settings are tunable:

action_plugins Actions are pieces of code in ansible that enable things like module execution, templating, and so forth.

This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
action_plugins = /usr/share/ansible_plugins/action_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details.

ansible_managed Ansible-managed is a string that can be inserted into files written by Ansible's config templating system, if you use a string like:

```
{{ ansible_managed }}
```

The default configuration shows who modified a file and when:

```
ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on {host}
```

This is useful to tell users that a file has been placed by Ansible and manual changes are likely to be overwritten.

Note that if using this feature, and there is a date in the string, the template will be reported changed each time as the date is updated.

ask_pass This controls whether an Ansible playbook should prompt for a password by default. The default behavior is no:

```
#ask_pass=True
```

If using SSH keys for authentication, it's probably not needed to change this setting.

ask_sudo_pass Similar to ask_pass, this controls whether an Ansible playbook should prompt for a sudo password by default when sudoing. The default behavior is also no:

```
#ask_sudo_pass=True
```

Users on platforms where sudo passwords are enabled should consider changing this setting.

callback_plugins This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
callback_plugins = /usr/share/ansible_plugins/callback_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

connection_plugins This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
connection_plugins = /usr/share/ansible_plugins/connection_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

deprecation_warnings New in version 1.3.

Allows disabling of deprecating warnings in ansible-playbook output:

```
deprecation_warnings = True
```

Deprecation warnings indicate usage of legacy features that are slated for removal in a future release of Ansible.

display_skipped_hosts If set to *False*, ansible will not display any status for a task that is skipped. The default behavior is to display skipped tasks:

```
#display_skipped_hosts=True
```

Note that Ansible will always show the task header for any task, regardless of whether or not the task is skipped.

error_on_undefined_vars On by default since Ansible 1.3, this causes ansible to fail steps that reference variable names that are likely typoed:

```
#error_on_undefined_vars=True
```

If set to False, any '{{ template_expression }}' that contains undefined variables will be rendered in a template or ansible action line exactly as written.

executable This indicates the command to use to spawn a shell under a sudo environment. Users may need to change this in rare instances to /bin/bash in rare instances when sudo is constrained, but in most cases it may be left as is:

```
#executable = /bin/bash
```

filter_plugins This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
filter_plugins = /usr/share/ansible_plugins/filter_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

forks This is the default number of parallel processes to spawn when communicating with remote hosts. Since Ansible 1.3, the fork number is automatically limited to the number of possible hosts, so this is really a limit of how much network and CPU load you think you can handle. Many users may set this to 50, some set it to 500 or more. If you have a large number of hosts, higher values will make actions across all of those hosts complete faster. The default is very very conservative:

```
forks=5
```

hash_behaviour Ansible by default will override variables in specific precedence orders, as described in *Variables*. When a variable of higher precedence wins, it will replace the other value.

Some users prefer that variables that are hashes (aka 'dictionaries' in Python terms) are merged together. This setting is called 'merge'. This is not the default behavior and it does not affect variables whose values are scalars (integers, strings) or arrays. We generally recommend not using this setting unless you think you have an absolute need for it, and playbooks in the official examples repos do not use this setting:

```
#hash_behaviour=replace
```

The valid values are either 'replace' (the default) or 'merge'.

hostfile This is the default location of the inventory file, script, or directory that Ansible will use to determine what hosts it has available to talk to:

```
hostfile = /etc/ansible/hosts
```

host_key_checking As described in *Getting Started*, host key checking is on by default in Ansible 1.3 and later. If you understand the implications and wish to disable it, you may do so here by setting the value to False:

```
host_key_checking=True
```

jinja2_extensions This is a developer-specific feature that allows enabling additional Jinja2 extensions:

```
jinja2_extensions = jinja2.ext.do,jinja2.ext.i18n
```

If you do not know what these do, you probably don't need to change this setting :)

legacy_playbook_variables Ansible prefers to use Jinja2 syntax '{{ like_this }}' to indicate a variable should be substituted in a particular string. However, older versions of playbooks used a more Perl-style syntax. This syntax was undesirable as it frequently conflicted with bash and was hard to explain to new users when referencing complicated variable hierarchies, so we have standardized on the '{{ jinja2 }}' way.

To ensure a string like '$foo' is not inadvertently replaced in a Perl or Bash script template, the old form of templating (which is still enabled as of Ansible 1.4) can be disabled like so

```
legacy_playbook_variables = no
```

library This is the default location Ansible looks to find modules:

```
library = /usr/share/ansible
```

Ansible knows how to look in multiple locations if you feed it a colon separated path, and it also will look for modules in the "./library" directory alongside a playbook.

log_path If present and configured in ansible.cfg, Ansible will log information about executions at the designated location. Be sure the user running Ansible has permissions on the logfile:

```
log_path=/var/log/ansible.log
```

This behavior is not on by default. Note that ansible will, without this setting, record module arguments called to the syslog of managed machines. Password arguments are excluded.

For Enterprise users seeking more detailed logging history, you may be interested in *Ansible Tower*.

lookup_plugins This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
lookup_plugins = /usr/share/ansible_plugins/lookup_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

module_name This is the default module name (-m) value for /usr/bin/ansible. The default is the 'command' module. Remember the command module doesn't support shell variables, pipes, or quotes, so you might wish to change it to 'shell':

```
module_name = command
```

nocolor By default ansible will try to colorize output to give a better indication of failure and status information. If you dislike this behavior you can turn it off by setting 'nocolor' to 1:

```
nocolor=0
```

nocows By default ansible will take advantage of cowsay if installed to make /usr/bin/ansible-playbook runs more exciting. Why? We believe systems management should be a happy experience. If you do not like the cows, you can disable them by setting 'nocows' to 1:

```
nocows=0
```

pattern This is the default group of hosts to talk to in a playbook if no "hosts:" stanza is supplied. The default is to talk to all hosts. You may wish to change this to protect yourself from surprises:

```
hosts=*
```

Note that /usr/bin/ansible always requires a host pattern and does not use this setting, only /usr/bin/ansible-playbook.

poll_interval For asynchronous tasks in Ansible (covered in *Asynchronous Actions and Polling*), this is how often to check back on the status of those tasks when an explicit poll interval is not supplied. The default is a reasonably moderate 15 seconds which is a tradeoff between checking in frequently and providing a quick turnaround when something may have completed:

```
poll_interval=15
```

private_key_file If you are using a pem file to authenticate with machines rather than SSH agent or passwords, you can set the default value here to avoid re-specifying `--ansible-private-keyfile` with every invocation:

```
private_key_file=/path/to/file.pem
```

remote_port This sets the default SSH port on all of your systems, for systems that didn't specify an alternative value in inventory. The default is the standard 22:

```
remote_port = 22
```

remote_tmp Ansible works by transferring modules to your remote machines, running them, and then cleaning up after itself. In some cases, you may not wish to use the default location and would like to change the path. You can do so by altering this setting:

```
remote_tmp = $HOME/.ansible/tmp
```

The default is to use a subdirectory of the user's home directory. Ansible will then choose a random directory name inside this location.

remote_user This is the default username ansible will connect as for /usr/bin/ansible-playbook. Note that /usr/bin/ansible will always default to the current user:

```
remote_user = root
```

roles_path The roles path indicate additional directories beyond the 'roles/' subdirectory of a playbook project to search to find Ansible roles. For instance, if there was a source control repository of common roles and a different repository of playbooks, you might choose to establish a convention to checkout roles in /opt/mysite/roles like so:

```
roles_path = /opt/mysite/roles
```

Roles will be first searched for in the playbook directory. Should a role not be found, it will indicate all the possible paths that were searched.

sudo_exe If using an alternative sudo implementation on remote machines, the path to sudo can be replaced here provided the sudo implementation is matching CLI flags with the standard sudo:

```
sudo_exe=sudo
```

sudo_flags Additional flags to pass to sudo when engaging sudo support. The default is '-H' which preserves the environment of the original user. In some situations you may wish to add or remote flags, but in general most users will not need to change this setting:

```
sudo_flags=-H
```

sudo_user This is the default user to sudo to if `--sudo-user` is not specified or 'sudo_user' is not specified in an Ansible playbook. The default is the most logical: 'root':

```
sudo_user=root
```

timeout This is the default SSH timeout to use on connection attempts:

```
timeout = 10
```

transport This is the default transport to use if "-c <transport_name>" is not specified to /usr/bin/ansible or /usr/bin/ansible-playbook. The default is 'smart', which will use 'ssh' (OpenSSH based) if the local operating system is new enough to support ControlPersist technology, and then will otherwise use 'paramiko'. Other transport options include 'local', 'chroot', 'jail', and so on.

Users should usually leave this setting as 'smart' and let their playbooks choose an alternate setting when needed with the 'connection:' play parameter.

vars_plugins This is a developer-centric feature that allows low-level extensions around Ansible to be loaded from different locations:

```
vars_plugins = /usr/share/ansible_plugins/vars_plugins
```

Most users will not need to use this feature. See *Developing Plugins* for more details

Paramiko Specific Settings

Paramiko is the default SSH connection implementation on Enterprise Linux 6 or earlier, and is not used by default on other platforms. Settings live under the [paramiko] header.

record_host_keys The default setting of yes will record newly discovered and approved (if host key checking is enabled) hosts in the user's hostfile. This setting may be inefficient for large numbers of hosts, and in those situations, using the ssh transport is definitely recommended instead. Setting it to False will improve performance and is recommended when host key checking is disabled:

```
record_host_keys=True
```

OpenSSH Specific Settings

Under the [ssh_connection] header, the following settings are tunable for SSH connections. OpenSSH is the default connection type for Ansible on OSes that are new enough to support ControlPersist. (This means basically all operating systems except Enterprise Linux 6 or earlier).

ssh_args If set, this will pass a specific set of options to Ansible rather than Ansible's usual defaults:

```
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

In particular, users may wish to raise the ControlPersist time to encourage performance. A value of 30 minutes may be appropriate.

control_path This is the location to save ControlPath sockets. This defaults to:

```
control_path=%(directory)s/ansible-ssh-%%h-%%p-%%r
```

On some systems with very long hostnames or very long path names (caused by long user names or deeply nested home directories) this can exceed the character limit on file socket names (108 characters for most platforms). In that case, you may wish to shorten the string to something like the below:

```
control_path = %(directory)s/%%h-%%r
```

Ansible 1.4 and later will instruct users to run with "-vvvv" in situations where it hits this problem and if so it is easy to tell there is too long of a Control Path filename. This may be frequently encountered on EC2.

scp_if_ssh Occasionally users may be managing a remote system that doesn't have SFTP enabled. If set to True, we can cause scp to be used to transfer remote files instead:

```
scp_if_ssh=False
```

There's really no reason to change this unless problems are encountered, and then there's also no real drawback to managing the switch. Most environments support SFTP by default and this doesn't usually need to be changed.

pipelining Enabling pipelining reduces the number of SSH operations required to execute a module on the remote server, by executing many ansible modules without actual file transfer. This can result in a very significant performance improvement when enabled, however when using "sudo:" operations you must first disable 'requiretty' in /etc/sudoers on all managed hosts.

By default, this option is disabled to preserve compatibility with sudoers configurations that have requiretty (the default on many distros), but is highly recommended if you can enable it, eliminating the need for *Accelerated Mode*:

```
pipelining=False
```

Accelerate Mode Settings

Under the [accelerate] header, the following settings are tunable for *Accelerated Mode*. Acceleration is a useful performance feature to use if you cannot enable *ssh_pipelining* in your environment, but is probably not needed if you can.

accelerate_port New in version 1.3.

This is the port to use for accelerate mode:

```
accelerate_port = 5099
```

accelerate_timeout New in version 1.4.

This setting controls the timeout for receiving data from a client. If no data is received during this time, the socket connection will be closed. A keepalive packet is sent back to the controller every 15 seconds, so this timeout should not be set lower than 15 (by default, the timeout is 30 seconds):

```
accelerate_timeout = 30
```

accelerate_connect_timeout New in version 1.4.

This setting controls the timeout for the socket connect call, and should be kept relatively low. The connection to the *accelerate_port* will be attempted 3 times before Ansible will fall back to ssh or paramiko (depending on your default connection setting) to try and start the accelerate daemon remotely. The default setting is 1.0 seconds:

```
accelerate_connect_timeout = 1.0
```

Note, this value can be set to less than one second, however it is probably not a good idea to do so unless you're on a very fast and reliable LAN. If you're connecting to systems over the internet, it may be necessary to increase this timeout.

1.2 Quickstart Video

We've recorded a short video that shows how to get started with Ansible that you may like to use alongside the documentation.

The quickstart video is about 20 minutes long and will show you some of the basics about your first steps with Ansible.

Enjoy, and be sure to visit the rest of the documentation to learn more.

1.3 Playbooks

Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

If Ansible modules are the tools in your workshop, playbooks are your design plans.

At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

While there's a lot of information here, there's no need to learn everything at once. You can start small and pick up more features over time as you need them.

Playbooks are designed to be human-readable and are developed in a basic text language. There are multiple ways to organize playbooks and the files they include, and we'll offer up some suggestions on that and making the most out of Ansible.

It is recommended to look at Example Playbooks while reading along with the playbook documentation. These illustrate best practices as well as how to put many of the various concepts together.

1.3.1 Intro to Playbooks

About Playbooks

Playbooks are a completely different way to use ansible than in adhoc task execution mode, and are particularly powerful.

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.

While you might run the main /usr/bin/ansible program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

There are also some full sets of playbooks illustrating a lot of these techniques in the ansible-examples repository. We'd recommend looking at these in another tab as you go along.

There are also many jumping off points after you learn playbooks, so hop back to the documentation index after you're done with this section.

Playbook Language Example

Playbooks are expressed in YAML format (see *YAML Syntax*) and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.

Each playbook is composed of one or more 'plays' in a list.

The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module, which you should have learned about in earlier chapters.

By composing a playbook of multiple 'plays', it is possible to orchestrate multi-machine deployments, running certain steps on all machines in the webservers group, then certain steps on the database server group, then more commands back on the webservers group, etc.

"plays" are more or less a sports analogy. You can have quite a lot of plays that affect your systems to do different things. It's not as if you were just defining one particular state or model, and you can run different plays at different times.

For starters, here's a playbook that contains just one play:

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
```

```
    - name: restart apache
      service: name=httpd state=restarted
```

Below, we'll break down what the various features of the playbook language are.

Basics

Hosts and Users

For each play in a playbook, you get to choose which machines in your infrastructure to target and what remote user to complete the steps (called tasks) as.

The *hosts* line is a list of one or more groups or host patterns, separated by colons, as described in the *Patterns* documentation. The *remote_user* is just the name of the user account:

```
---
- hosts: webservers
  remote_user: root
```

Note: The *remote_user* parameter was formerly called just *user*. It was renamed in Ansible 1.4 to make it more distinguishable from the *user* module (used to create users on remote systems).

Remote users can also be defined per task:

```
---
- hosts: webservers
  remote_user: root
  tasks:
    - name: test connection
      ping:
      remote_user: yourname
```

Note: The *remote_user* parameter for tasks was added in 1.4.

Support for running things from sudo is also available:

```
---
- hosts: webservers
  remote_user: yourname
  sudo: yes
```

You can also use sudo on a particular task instead of the whole play:

```
---
- hosts: webservers
  remote_user: yourname
  tasks:
    - service: name=nginx state=started
      sudo: yes
```

You can also login as you, and then sudo to different users than root:

```
---
- hosts: webservers
  remote_user: yourname
  sudo: yes
  sudo_user: postgres
```

If you need to specify a password to sudo, run *ansible-playbook* with `--ask-sudo-pass` (*-K*). If you run a sudo playbook and the playbook seems to hang, it's probably stuck at the sudo prompt. Just *Control-C* to kill it and run it again with *-K*.

Important: When using *sudo_user* to a user other than root, the module arguments are briefly written into a random tempfile in /tmp. These are deleted immediately after the command is executed. This only occurs when sudoing from a user like 'bob' to 'timmy', not when going from 'bob' to 'root', or logging in directly as 'bob' or 'root'. If this concerns you that this data is briefly readable (not writable), avoid transferring uncrypted passwords with *sudo_user* set. In other cases, '/tmp' is not used and this does not come into play. Ansible also takes care to not log password parameters.

Tasks list

Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task. It is important to understand that, within a play, all hosts are going to get the same task directives. It is the purpose of a play to map a selection of hosts to tasks.

When running the playbook, which runs top to bottom, hosts with failed tasks are taken out of the rotation for the entire playbook. If things fail, simply correct the playbook file and rerun.

The goal of each task is to execute a module, with very specific arguments. Variables, as mentioned above, can be used in arguments to modules.

Modules are 'idempotent', meaning if you run them again, they will make only the changes they must in order to bring the system to the desired state. This makes it very safe to rerun the same playbook multiple times. They won't change things unless they have to change things.

The *command* and *shell* modules will typically rerun the same command again, which is totally ok if the command is something like 'chmod' or 'setsebool', etc. Though there is a 'creates' flag available which can be used to make these modules also idempotent.

Every task should have a *name*, which is included in the output from running the playbook. This is output for humans, so it is nice to have reasonably good descriptions of each task step. If the name is not provided though, the string fed to 'action' will be used for output.

Tasks can be declared using the legacy "action: module options" format, but it is recommended that you use the more conventional "module: options" format. This recommended format is used throughout the documentation, but you may encounter the older format in some playbooks.

Here is what a basic task looks like, as with most modules, the service module takes key=value arguments:

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=running
```

The *command* and *shell* modules are the only modules that just take a list of arguments and don't use the key=value form. This makes them work as simply as you would expect:

```
tasks:
  - name: disable selinux
    command: /sbin/setenforce 0
```

The command and shell module care about return codes, so if you have a command whose successful exit code is not zero, you may wish to do this:

```
tasks:
  - name: run this command and ignore the result
    shell: /usr/bin/somecommand || /bin/true
```

Or this:

```
tasks:
  - name: run this command and ignore the result
    shell: /usr/bin/somecommand
    ignore_errors: True
```

If the action line is getting too long for comfort you can break it on a space and indent any continuation lines:

```
tasks:
  - name: Copy ansible inventory file to client
    copy: src=/etc/ansible/hosts dest=/etc/ansible/hosts
            owner=root group=root mode=0644
```

Variables can be used in action lines. Suppose you defined a variable called 'vhost' in the 'vars' section, you could do this:

```
tasks:
  - name: create a virtual host file for {{ vhost }}
    template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}
```

Those same variables are usable in templates, which we'll get to later.

Now in a very basic playbook all the tasks will be listed directly in that play, though it will usually make more sense to break up tasks using the 'include:' directive. We'll show that a bit later.

Action Shorthand

New in version 0.8.

Ansible prefers listing modules like this in 0.8 and later:

```
template: src=templates/foo.j2 dest=/etc/foo.conf
```

You will notice in earlier versions, this was only available as:

```
action: template src=templates/foo.j2 dest=/etc/foo.conf
```

The old form continues to work in newer versions without any plan of deprecation.

Handlers: Running Operations On Change

As we've mentioned, 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 block of tasks in a playbook, and will only be triggered once even if notified by multiple different tasks.

For instance, multiple resources may indicate that apache needs to be restarted because they have changed a config file, but apache will only be bounced once to avoid unnecessary restarts.

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

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.

Here's an example handlers section:

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

Handlers are best used to restart services and trigger reboots. You probably won't need them for much else.

Note: Notify handlers are always run in the order written.

Roles are described later on. It's worthwhile to point out that handlers are automatically processed between 'pre_tasks', 'roles', 'tasks', and 'post_tasks' sections. If you ever want to flush all the handler commands immediately though, in 1.2 and later, you can:

```
tasks:
   - shell: some tasks go here
   - meta: flush_handlers
   - shell: some other tasks
```

In the above example any queued up handlers would be processed early when the 'meta' statement was reached. This is a bit of a niche case but can come in handy from time to time.

Executing A Playbook

Now that you've learned playbook syntax, how do you run a playbook? It's simple. Let's run a playbook using a parallelism level of 10:

```
ansible-playbook playbook.yml -f 10
```

Ansible-Pull

Should you want to invert the architecture of Ansible, so that nodes check in to a central location, instead of pushing configuration out to them, you can.

Ansible-pull is a small script that will checkout a repo of configuration instructions from git, and then run ansible-playbook against that content.

Assuming you load balance your checkout location, ansible-pull scales essentially infinitely.

Run `ansible-pull --help` for details.

There's also a clever playbook available to using ansible in push mode to configure ansible-pull via a crontab!

Tips and Tricks

Look at the bottom of the playbook execution for a summary of the nodes that were targeted and how they performed. General failures and fatal "unreachable" communication attempts are kept separate in the counts.

If you ever want to see detailed output from successful modules as well as unsuccessful ones, use the `--verbose` flag. This is available in Ansible 0.5 and later.

Ansible playbook output is vastly upgraded if the cowsay package is installed. Try it!

To see what hosts would be affected by a playbook before you run it, you can do this:

```
ansible-playbook playbook.yml --list-hosts.
```

See also:

YAML Syntax Learn about YAML syntax

Best Practices Various tips about managing playbooks in the real world

Ansible Documentation Hop back to the documentation index for a lot of special topics about playbooks

About Modules Learn about available modules

Developing Modules Learn how to extend Ansible by writing your own modules

Patterns Learn about how to select hosts

Github examples directory Complete end-to-end playbook examples

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

1.3.2 Playbook Roles and Include Statements

Topics

- Playbook Roles and Include Statements
  - Introduction
  - Task Include Files And Encouraging Reuse
  - Roles
  - Role Default Variables
  - Role Dependencies
  - Ansible Galaxy

Introduction

While it is possible to write a playbook in one very large file (and you might start out learning playbooks this way), eventually you'll want to reuse files and start to organize things.

At a basic level, including task files allows you to break up bits of configuration policy into smaller files. Task includes pull in tasks from other files. Since handlers are tasks too, you can also include handler files from the 'handlers:' section.

See *Playbooks* if you need a review of these concepts.

Playbooks can also include plays from other playbook files. When that is done, the plays will be inserted into the playbook to form a longer list of plays.

When you start to think about it – tasks, handlers, variables, and so on – begin to form larger concepts. You start to think about modeling what something is, rather than how to make something look like something. It's no longer "apply this handful of THINGS to these hosts", you say "these hosts are dbservers" or "these hosts are webservers". In programming, we might call that "encapsulating" how things work. For instance, you can drive a car without knowing how the engine works.

Roles in Ansible build on the idea of include files and combine them to form clean, reusable abstractions – they allow you to focus more on the big picture and only dive down into the details when needed.

We'll start with understanding includes so roles make more sense, but our ultimate goal should be understanding roles – roles are great and you should use them every time you write playbooks.

See the ansible-examples repository on GitHub for lots of examples of all of this put together. You may wish to have this open in a separate tab as you dive in.

Task Include Files And Encouraging Reuse

Suppose you want to reuse lists of tasks between plays or playbooks. You can use include files to do this. Use of included task lists is a great way to define a role that system is going to fulfill. Remember, the goal of a play in a playbook is to map a group of systems into multiple roles. Let's see what this looks like...

A task include file simply contains a flat list of tasks, like so:

```
---
# possibly saved as tasks/foo.yml
- name: placeholder foo
  command: /bin/foo
- name: placeholder bar
  command: /bin/bar
```

Include directives look like this, and can be mixed in with regular tasks in a playbook:

```
tasks:
 - include: tasks/foo.yml
```

You can also pass variables into includes. We call this a 'parameterized include'.

For instance, if deploying multiple wordpress instances, I could contain all of my wordpress tasks in a single wordpress.yml file, and use it like so:

```
tasks:
  - include: wordpress.yml user=timmy
  - include: wordpress.yml user=alice
  - include: wordpress.yml user=bob
```

If you are running Ansible 1.4 and later, include syntax is streamlined to match roles, and also allows passing list and dictionary parameters:

```
tasks:
 - { include: wordpress.yml, user: timmy, ssh_keys: [ 'keys/one.txt', 'keys/two.txt' ] }
```

Using either syntax, variables passed in can then be used in the included files. We've already covered them a bit in *Variables*. You can reference them like this:

```
{{ user }}
```

(In addition to the explicitly passed-in parameters, all variables from the vars section are also available for use here as well.)

Starting in 1.0, variables can also be passed to include files using an alternative syntax, which also supports structured variables:

```
tasks:

  - include: wordpress.yml
    vars:
```

```
        remote_user: timmy
        some_list_variable:
          - alpha
          - beta
          - gamma
```

Playbooks can include other playbooks too, but that's mentioned in a later section.

Note: As of 1.0, task include statements can be used at arbitrary depth. They were previously limited to a single level, so task includes could not include other files containing task includes.

Includes can also be used in the 'handlers' section, for instance, if you want to define how to restart apache, you only have to do that once for all of your playbooks. You might make a handlers.yml that looks like:

```
---
# this might be in a file like handlers/handlers.yml
- name: restart apache
  service: name=apache state=restarted
```

And in your main playbook file, just include it like so, at the bottom of a play:

```
handlers:
  - include: handlers/handlers.yml
```

You can mix in includes along with your regular non-included tasks and handlers.

Includes can also be used to import one playbook file into another. This allows you to define a top-level playbook that is composed of other playbooks.

For example:

```
- name: this is a play at the top level of a file
  hosts: all
  remote_user: root
  tasks:
  - name: say hi
    tags: foo
    shell: echo "hi..."

- include: load_balancers.yml
- include: webservers.yml
- include: dbservers.yml
```

Note that you cannot do variable substitution when including one playbook inside another.

Note: You can not conditionally path the location to an include file, like you can with 'vars_files'. If you find yourself needing to do this, consider how you can restructure your playbook to be more class/role oriented. This is to say you cannot use a 'fact' to decide what include file to use. All hosts contained within the play are going to get the same tasks. ('*when*' provides some ability for hosts to conditionally skip tasks).

Roles

New in version 1.2.

Now that you have learned about vars_files, tasks, and handlers, what is the best way to organize your playbooks? The short answer is to use roles! Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.

Roles are just automation around 'include' directives as described above, and really don't contain much additional magic beyond some improvements to search path handling for referenced files. However, that can be a big thing!

Example project structure:

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     meta/
   webservers/
     files/
     templates/
     tasks/
     handlers/
     vars/
     meta/
```

In a playbook, it would look like this:

```
---
- hosts: webservers
  roles:
     - common
     - webservers
```

This designates the following behaviors, for each role 'x':

- If roles/x/tasks/main.yml exists, tasks listed therein will be added to the play
- If roles/x/handlers/main.yml exists, handlers listed therein will be added to the play
- If roles/x/vars/main.yml exists, variables listed therein will be added to the play
- If roles/x/meta/main.yml exists, any role dependencies listed therein will be added to the list of roles (1.3 and later)
- Any copy tasks can reference files in roles/x/files/ without having to path them relatively or absolutely
- Any script tasks can reference scripts in roles/x/files/ without having to path them relatively or absolutely
- Any template tasks can reference files in roles/x/templates/ without having to path them relatively or absolutely
- Any include tasks can reference files in roles/x/tasks/ without having to path them relatively or absolutely

In Ansible 1.4 and later you can configure a roles_path to search for roles. Use this to check all of your common roles out to one location, and share them easily between multiple playbook projects. See *The Ansible Configuration File* for details about how to set this up in ansible.cfg.

Note: Role dependencies are discussed below.

If any files are not present, they are just ignored. So it's ok to not have a 'vars/' subdirectory for the role, for instance.

Note, you are still allowed to list tasks, vars_files, and handlers "loose" in playbooks without using roles, but roles are a good organizational feature and are highly recommended. if there are loose things in the playbook, the roles are evaluated first.

Also, should you wish to parameterize roles, by adding variables, you can do so, like this:

```
---
- hosts: webservers
  roles:
    - common
    - { role: foo_app_instance, dir: '/opt/a',  port: 5000 }
    - { role: foo_app_instance, dir: '/opt/b',  port: 5001 }
```

While it's probably not something you should do often, you can also conditionally apply roles like so:

```
---
- hosts: webservers
  roles:
    - { role: some_role, when: "ansible_os_family == 'RedHat'" }
```

This works by applying the conditional to every task in the role. Conditionals are covered later on in the documentation.

Finally, you may wish to assign tags to the roles you specify. You can do so inline::

```
---
- hosts: webservers
  roles:
    - { role: foo, tags: ["bar", "baz"] }
```

If the play still has a 'tasks' section, those tasks are executed after roles are applied.

If you want to define certain tasks to happen before AND after roles are applied, you can do this:

```
---
- hosts: webservers
  pre_tasks:
    - shell: echo 'hello'
  roles:
    - { role: some_role }
  tasks:
    - shell: echo 'still busy'
  post_tasks:
    - shell: echo 'goodbye'
```

Note: If using tags with tasks (described later as a means of only running part of a playbook), be sure to also tag your pre_tasks and post_tasks and pass those along as well, especially if the pre and post tasks are used for monitoring outage window control or load balancing.

Role Default Variables

New in version 1.3.

Role default variables allow you to set default variables for included or dependent roles (see below). To create defaults, simply add a *defaults/main.yml* file in your role directory. These variables will have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.

Role Dependencies

New in version 1.3.

Role dependencies allow you to automatically pull in other roles when using a role. Role dependencies are stored in the *meta/main.yml* file contained within the role directory. This file should contain a list of roles and parameters to insert before the specified role, such as the following in an example *roles/myapp/meta/main.yml*:

```
---
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: apache, port: 80 }
  - { role: postgres, dbname: blarg, other_parameter: 12 }
```

Role dependencies can also be specified as a full path, just like top level roles:

```
---
dependencies:
   - { role: '/path/to/common/roles/foo', x: 1 }
```

Roles dependencies are always executed before the role that includes them, and are recursive. By default, roles can also only be added as a dependency once - if another role also lists it as a dependency it will not be run again. This behavior can be overridden by adding *allow_duplicates: yes* to the *meta/main.yml* file. For example, a role named 'car' could add a role named 'wheel' to its dependencies as follows:

```
---
dependencies:
- { role: wheel, n: 1 }
- { role: wheel, n: 2 }
- { role: wheel, n: 3 }
- { role: wheel, n: 4 }
```

And the *meta/main.yml* for wheel contained the following:

```
---
allow_duplicates: yes
dependencies:
- { role: tire }
- { role: brake }
```

The resulting order of execution would be as follows:

```
tire(n=1)
brake(n=1)
wheel(n=1)
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

Note: Variable inheritance and scope are detailed in the *Variables*.

Ansible Galaxy

Ansible Galaxy, is a free site for finding, downloading, rating, and reviewing all kinds of community developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

You can sign up with social auth, and the download client 'ansible-galaxy' is included in Ansible 1.4.2 and later.

Read the "About" page on the Galaxy site for more information.

See also:

YAML Syntax Learn about YAML syntax

Playbooks Review the basic Playbook language features

Best Practices Various tips about managing playbooks in the real world

Variables All about variables in playbooks

Conditionals Conditionals in playbooks

Loops Loops in playbooks

About Modules Learn about available modules

Developing Modules Learn how to extend Ansible by writing your own modules

GitHub Ansible examples Complete playbook files from the GitHub project source

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

1.3.3 Variables

Topics

- Variables
  - What Makes A Valid Variable Name
  - Variables Defined in Inventory
  - Variables Defined in a Playbook
  - Variables defined from included files and roles
  - Using Variables: About Jinja2
  - Jinja2 Filters
    - Filters For Formatting Data
    - Filters Often Used With Conditionals
    - Forcing Variables To Be Defined
    - Defaulting Undefined Variables
    - Set Theory Filters
    - Other Useful Filters
  - Hey Wait, A YAML Gotcha
  - Information discovered from systems: Facts
  - Turning Off Facts
  - Local Facts (Facts.d)
  - Registered Variables
  - Accessing Complex Variable Data
  - Magic Variables, and How To Access Information About Other Hosts
  - Variable File Separation
  - Passing Variables On The Command Line
  - Conditional Imports
  - Variable Precedence: Where Should I Put A Variable?

While automation exists to make it easier to make things repeatable, all of your systems are likely not exactly alike.

All of your systems are likely not the same. On some systems you may want to set some behavior or configuration that is slightly different from others.

Also, some of the observed behavior or state of remote systems might need to influence how you configure those systems. (Such as you might need to find out the IP address of a system and even use it as a configuration value on another system).

You might have some templates for configuration files that are mostly the same, but slightly different based on those variables.

Variables in Ansible are how we deal with differences between systems.

Once understanding variables you'll also want to dig into *Conditionals* and *Loops*. Useful things like the "group_by" module and the "when" conditional can also be used with variables, and to help manage differences between systems.

It's highly recommended that you consult the ansible-examples github repository to see a lot of examples of variables put to use.

What Makes A Valid Variable Name

Before we start using variables it's important to know what are valid variable names.

Variable names should be letters, numbers, and underscores. Variables should always start with a letter.

"foo_port" is a great variable. "foo5" is fine too.

"foo-port", "foo port", "foo.port" and "12" are not valid variable names.

Easy enough, let's move on.

Variables Defined in Inventory

We've actually already covered a lot about variables in another section, so so far this shouldn't be terribly new, but a bit of a refresher.

Often you'll want to set variables based on what groups a machine is in. For instance, maybe machines in Boston want to use 'boston.ntp.example.com' as an NTP server.

See the *Inventory* document for multiple ways on how to define variables in inventory.

Variables Defined in a Playbook

In a playbook, it's possible to define variables directly inline like so:

```
- hosts: webservers
  vars:
    http_port: 80
```

This can be nice as it's right there when you are reading the playbook.

Variables defined from included files and roles

It turns out we've already talked about variables in another place too.

As described in *Playbook Roles and Include Statements*, variables can also be included in the playbook via include files, which may or may not be part of an "Ansible Role". Usage of roles is preferred as it provides a nice organizational system.

Using Variables: About Jinja2

It's nice enough to know about how to define variables, but how do you use them?

Ansible allows you to reference variables in your playbooks using the Jinja2 templating system. While you can do a lot of complex things in Jinja, only the basics are things you really need to learn at first.

For instance, in a simple template, you can do something like:

```
My amp goes to {{ max_amp_value }}
```

And that will provide the most basic form of variable substitution.

This is also valid directly in playbooks, and you'll occasionally want to do things like:

```
template: src=foo.cfg.j2 dest={{ remote_install_path}}/foo.cfg
```

In the above example, we used a variable to help decide where to place a file.

Inside a template you automatically have access to all of the variables that are in scope for a host. Actually it's more than that – you can also read variables about other hosts. We'll show how to do that in a bit.

Note: ansible allows Jinja2 loops and conditionals in templates, but in playbooks, we do not use them. Ansible templates are pure machine-parseable YAML. This is an rather important feature as it means it is possible to code-generate pieces of files, or to have other ecosystem tools read Ansible files. Not everyone will need this but it can unlock possibilities.

Jinja2 Filters

Note: These are infrequently utilized features. Use them if they fit a use case you have, but this is optional knowledge.

Filters in Jinja2 are a way of transforming template expressions from one kind of data into another. Jinja2 ships with many of these. See builtin filters in the official Jinja2 template documentation.

In addition to those, Ansible supplies many more.

Filters For Formatting Data

The following filters will take a data structure in a template and render it in a slightly different format. These are occasionally useful for debugging:

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

Filters Often Used With Conditionals

The following tasks are illustrative of how filters can be used with conditionals:

```
tasks:
  - shell: /usr/bin/foo
    register: result
    ignore_errors: True

  - debug: msg="it failed"
    when: result|failed

  # in most cases you'll want a handler, but if you want to do something right now, this is nice
  - debug: msg="it changed"
    when: result|changed

  - debug: msg="it succeeded"
    when: result|success
```

```
- debug: msg="it was skipped"
  when: result|skipped
```

Forcing Variables To Be Defined

The default behavior from ansible and ansible.cfg is to fail if variables are undefined, but you can turn this off.

This allows an explicit check with this feature off:

```
{{ variable | mandatory }}
```

The variable value will be used as is, but the template evaluation will raise an error if it is undefined.

Defaulting Undefined Variables

Jinja2 provides a useful 'default' filter, that is often a better approach to failing if a variable is not defined.

```
{{ some_variable | default(5) }}
```

In the above example, if the variable 'some_variable' is not defined, the value used will be 5, rather than an error being raised.

Set Theory Filters

All these functions return a unique set from sets or lists.

New in version 1.4.

To get a unique set from a list:

```
{{ list1 | unique }}
```

To get a union of two lists:

```
{{ list1 | union(list2) }}
```

To get the intersection of 2 lists (unique list of all items in both):

```
{{ list1 | intersect(list2) }}
```

To get the difference of 2 lists (items in 1 that don't exist in 2):

```
{{ list1 | difference(list2) }}
```

To get the symmetric difference of 2 lists (items exclusive to each list):

```
{{ list1 | symmetric_difference(list2) }}
```

Other Useful Filters

To get the last name of a file path, like 'foo.txt' out of '/etc/asdf/foo.txt':

```
{{ path | basename }}
```

To get the directory from a path:

```
{{ path | dirname }}
```

To expand a path containing a tilde (~) character:

{{ path | expanduser }}

To work with Base64 encoded strings:

```
{{ encoded | b64decode }}
{{ decoded | b64encode }}
```

To take an md5sum of a filename:

```
{{ filename | md5 }}
```

To cast values as certain types, such as when you input a string as "True" from a vars_prompt and the system doesn't know it is a boolean value:

```
- debug: msg=test
  when: some_string_value | bool
```

A few useful filters are typically added with each new Ansible release. The development documentation shows how to extend Ansible filters by writing your own as plugins, though in general, we encourage new ones to be added to core so everyone can make use of them.

Hey Wait, A YAML Gotcha

YAML syntax requires that if you start a value with {{ foo }} you quote the whole line, since it wants to be sure you aren't trying to start a YAML dictionary. This is covered on the *YAML Syntax* page.

This won't work:

```
- hosts: app_servers
  vars:
      app_path: {{ base_path }}/22
```

Do it like this and you'll be fine:

```
- hosts: app_servers
  vars:
       app_path: "{{ base_path }}/22"
```

Information discovered from systems: Facts

There are other places where variables can come from, but these are a type of variable that are discovered, not set by the user.

Facts are information derived from speaking with your remote systems.

An example of this might be the ip address of the remote host, or what the operating system is.

To see what information is available, try the following:

```
ansible hostname -m setup
```

This will return a ginormous amount of variable data, which may look like this, as taken from Ansible 1.4 on a Ubuntu 12.04 system:

```
"ansible_all_ipv4_addresses": [
    "REDACTED IP ADDRESS"
],
"ansible_all_ipv6_addresses": [
    "REDACTED IPV6 ADDRESS"
],
"ansible_architecture": "x86_64",
"ansible_bios_date": "09/20/2012",
"ansible_bios_version": "6.00",
"ansible_cmdline": {
    "BOOT_IMAGE": "/boot/vmlinuz-3.5.0-23-generic",
    "quiet": true,
    "ro": true,
    "root": "UUID=4195bff4-e157-4e41-8701-e93f0aec9e22",
    "splash": true
},
"ansible_date_time": {
    "date": "2013-10-02",
    "day": "02",
    "epoch": "1380756810",
    "hour": "19",
    "iso8601": "2013-10-02T23:33:30Z",
    "iso8601_micro": "2013-10-02T23:33:30.036070Z",
    "minute": "33",
    "month": "10",
    "second": "30",
    "time": "19:33:30",
    "tz": "EDT",
    "year": "2013"
},
"ansible_default_ipv4": {
    "address": "REDACTED",
    "alias": "eth0",
    "gateway": "REDACTED",
    "interface": "eth0",
    "macaddress": "REDACTED",
    "mtu": 1500,
    "netmask": "255.255.255.0",
    "network": "REDACTED",
    "type": "ether"
},
"ansible_default_ipv6": {},
"ansible_devices": {
    "fd0": {
        "holders": [],
        "host": "",
        "model": null,
        "partitions": {},
        "removable": "1",
        "rotational": "1",
        "scheduler_mode": "deadline",
        "sectors": "0",
        "sectorsize": "512",
        "size": "0.00 Bytes",
        "support_discard": "0",
        "vendor": null
    },
    "sda": {
```

```
        "holders": [],
        "host": "SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ult
        "model": "VMware Virtual S",
        "partitions": {
            "sda1": {
                "sectors": "39843840",
                "sectorsize": 512,
                "size": "19.00 GB",
                "start": "2048"
            },
            "sda2": {
                "sectors": "2",
                "sectorsize": 512,
                "size": "1.00 KB",
                "start": "39847934"
            },
            "sda5": {
                "sectors": "2093056",
                "sectorsize": 512,
                "size": "1022.00 MB",
                "start": "39847936"
            }
        },
        "removable": "0",
        "rotational": "1",
        "scheduler_mode": "deadline",
        "sectors": "41943040",
        "sectorsize": "512",
        "size": "20.00 GB",
        "support_discard": "0",
        "vendor": "VMware,"
    },
    "sr0": {
        "holders": [],
        "host": "IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)",
        "model": "VMware IDE CDR10",
        "partitions": {},
        "removable": "1",
        "rotational": "1",
        "scheduler_mode": "deadline",
        "sectors": "2097151",
        "sectorsize": "512",
        "size": "1024.00 MB",
        "support_discard": "0",
        "vendor": "NECVMWar"
    }
},
"ansible_distribution": "Ubuntu",
"ansible_distribution_release": "precise",
"ansible_distribution_version": "12.04",
"ansible_domain": "",
"ansible_env": {
    "COLORTERM": "gnome-terminal",
    "DISPLAY": ":0",
    "HOME": "/home/mdehaan",
    "LANG": "C",
    "LESSCLOSE": "/usr/bin/lesspipe %s %s",
    "LESSOPEN": "| /usr/bin/lesspipe %s",
```

```
    "LOGNAME": "root",
    "LS_COLORS": "rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=
    "MAIL": "/var/mail/root",
    "OLDPWD": "/root/ansible/docsite",
    "PATH": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "PWD": "/root/ansible",
    "SHELL": "/bin/bash",
    "SHLVL": "1",
    "SUDO_COMMAND": "/bin/bash",
    "SUDO_GID": "1000",
    "SUDO_UID": "1000",
    "SUDO_USER": "mdehaan",
    "TERM": "xterm",
    "USER": "root",
    "USERNAME": "root",
    "XAUTHORITY": "/home/mdehaan/.Xauthority",
    "_": "/usr/local/bin/ansible"
},
"ansible_eth0": {
    "active": true,
    "device": "eth0",
    "ipv4": {
        "address": "REDACTED",
        "netmask": "255.255.255.0",
        "network": "REDACTED"
    },
    "ipv6": [
        {
            "address": "REDACTED",
            "prefix": "64",
            "scope": "link"
        }
    ],
    "macaddress": "REDACTED",
    "module": "e1000",
    "mtu": 1500,
    "type": "ether"
},
"ansible_form_factor": "Other",
"ansible_fqdn": "ubuntu2",
"ansible_hostname": "ubuntu2",
"ansible_interfaces": [
    "lo",
    "eth0"
],
"ansible_kernel": "3.5.0-23-generic",
"ansible_lo": {
    "active": true,
    "device": "lo",
    "ipv4": {
        "address": "127.0.0.1",
        "netmask": "255.0.0.0",
        "network": "127.0.0.0"
    },
    "ipv6": [
        {
            "address": "::1",
            "prefix": "128",
```

```
            "scope": "host"
        }
    ],
    "mtu": 16436,
    "type": "loopback"
},
"ansible_lsb": {
    "codename": "precise",
    "description": "Ubuntu 12.04.2 LTS",
    "id": "Ubuntu",
    "major_release": "12",
    "release": "12.04"
},
"ansible_machine": "x86_64",
"ansible_memfree_mb": 74,
"ansible_memtotal_mb": 991,
"ansible_mounts": [
    {
        "device": "/dev/sda1",
        "fstype": "ext4",
        "mount": "/",
        "options": "rw,errors=remount-ro",
        "size_available": 15032406016,
        "size_total": 20079898624
    }
],
"ansible_os_family": "Debian",
"ansible_pkg_mgr": "apt",
"ansible_processor": [
    "Intel(R) Core(TM) i7 CPU         860  @ 2.80GHz"
],
"ansible_processor_cores": 1,
"ansible_processor_count": 1,
"ansible_processor_threads_per_core": 1,
"ansible_processor_vcpus": 1,
"ansible_product_name": "VMware Virtual Platform",
"ansible_product_serial": "REDACTED",
"ansible_product_uuid": "REDACTED",
"ansible_product_version": "None",
"ansible_python_version": "2.7.3",
"ansible_selinux": false,
"ansible_ssh_host_key_dsa_public": "REDACTED KEY VALUE"
"ansible_ssh_host_key_ecdsa_public": "REDACTED KEY VALUE"
"ansible_ssh_host_key_rsa_public": "REDACTED KEY VALUE"
"ansible_swapfree_mb": 665,
"ansible_swaptotal_mb": 1021,
"ansible_system": "Linux",
"ansible_system_vendor": "VMware, Inc.",
"ansible_user_id": "root",
"ansible_userspace_architecture": "x86_64",
"ansible_userspace_bits": "64",
"ansible_virtualization_role": "guest",
"ansible_virtualization_type": "VMware"
```


In the above the model of the first harddrive may be referenced in a template or playbook as:

```
{{ ansible_devices.sda.model }}
```

Similarly, the hostname as the system reports it is:

```
{{ ansible_hostname }}
```

Facts are frequently used in conditionals (see *Conditionals*) and also in templates.

Facts can be also used to create dynamic groups of hosts that match particular criteria, see the *About Modules* documentation on 'group_by' for details, as well as in generalized conditional statements as discussed in the *Conditionals* chapter.

Turning Off Facts

If you know you don't need any fact data about your hosts, and know everything about your systems centrally, you can turn off fact gathering. This has advantages in scaling Ansible in push mode with very large numbers of systems, mainly, or if you are using Ansible on experimental platforms. In any play, just do this:

```
- hosts: whatever
  gather_facts: no
```

Local Facts (Facts.d)

New in version 1.3.

As discussed in the playbooks chapter, Ansible facts are a way of getting data about remote systems for use in playbook variables. Usually these are discovered automatically by the 'setup' module in Ansible. Users can also write custom facts modules, as described in the API guide. However, what if you want to have a simple way to provide system or user provided data for use in Ansible variables, without writing a fact module?

For instance, what if you want users to be able to control some aspect about how their systems are managed? "Facts.d" is one such mechanism.

Note: Perhaps "local facts" is a bit of a misnomer, it means "locally supplied user values" as opposed to "centrally supplied user values", or what facts are – "locally dynamically determined values".

If a remotely managed system has an "/etc/ansible/facts.d" directory, any files in this directory ending in ".fact", can be JSON, INI, or executable files returning JSON, and these can supply local facts in Ansible.

For instance assume a /etc/ansible/facts.d/preferences.fact:

```
[general]
asdf=1
bar=2
```

This will produce a hash variable fact named "general" with 'asdf' and 'bar' as members. To validate this, run the following:

```
ansible <hostname> -m setup -a "filter=ansible_local"
```

And you will see the following fact added:

```
"ansible_local": {
        "preferences": {
            "general": {
                "asdf" : "1",
                "bar"  : "2"
            }
        }
 }
```

And this data can be accessed in a template/playbook as:

```
{{ ansible_local.preferences.general.asdf }}
```

The local namespace prevents any user supplied fact from overriding system facts or variables defined elsewhere in the playbook.

Registered Variables

Another major use of variables is running a command and using the result of that command to save the result into a variable. Results will vary from module to module. Use of -v when executing playbooks will show possible values for the results.

The value of a task being executed in ansible can be saved in a variable and used later. See some examples of this in the *Conditionals* chapter.

While it's mentioned elsewhere in that document too, here's a quick syntax example:

```
- hosts: web_servers

  tasks:

     - shell: /usr/bin/foo
       register: foo_result
       ignore_errors: True

     - shell: /usr/bin/bar
       when: foo_result.rc == 5
```

Registered variables are valid on the host the remainder of the playbook run, which is the same as the lifetime of "facts" in Ansible. Effectively registered variables are just like facts.

Accessing Complex Variable Data

We already talked about facts a little higher up in the documentation.

Some provided facts, like networking information, are made available as nested data structures. To access them a simple {{ foo }} is not sufficient, but it is still easy to do. Here's how we get an IP address:

```
{{ ansible_eth0["ipv4"]["address"] }}
```

OR alternatively:

```
{{ ansible_eth0.ipv4.address }}
```

Similarly, this is how we access the first element of an array:

```
{{ foo[0] }}
```

Magic Variables, and How To Access Information About Other Hosts

Even if you didn't define them yourself, Ansible provides a few variables for you automatically. The most important of these are 'hostvars', 'group_names', and 'groups'. Users should not use these names themselves as they are reserved. 'environment' is also reserved.

Hostvars lets you ask about the variables of another host, including facts that have been gathered about that host. If, at this point, you haven't talked to that host yet in any play in the playbook or set of playbooks, you can get at the variables, but you will not be able to see the facts.

If your database server wants to use the value of a 'fact' from another node, or an inventory variable assigned to another node, it's easy to do so within a template or even an action line:

```
{{ hostvars['test.example.com']['ansible_distribution'] }}
```

Additionally, *group_names* is a list (array) of all the groups the current host is in. This can be used in templates using Jinja2 syntax to make template source files that vary based on the group membership (or role) of the host:

```
{% if 'webserver' in group_names %}
   # some part of a configuration file that only applies to webservers
{% endif %}
```

groups is a list of all the groups (and hosts) in the inventory. This can be used to enumerate all hosts within a group. For example:

```
{% for host in groups['app_servers'] %}
   # something that applies to all app servers.
{% endfor %}
```

A frequently used idiom is walking a group to find all IP addresses in that group:

```
{% for host in groups['app_servers'] %}
   {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
{% endfor %}
```

An example of this could include pointing a frontend proxy server to all of the app servers, setting up the correct firewall rules between servers, etc.

Additionally, *inventory_hostname* is the name of the hostname as configured in Ansible's inventory host file. This can be useful for when you don't want to rely on the discovered hostname *ansible_hostname* or for other mysterious reasons. If you have a long FQDN, *inventory_hostname_short* also contains the part up to the first period, without the rest of the domain.

play_hosts is available as a list of hostnames that are in scope for the current play. This may be useful for filling out templates with multiple hostnames or for injecting the list into the rules for a load balancer.

Don't worry about any of this unless you think you need it. You'll know when you do.

Also available, *inventory_dir* is the pathname of the directory holding Ansible's inventory host file, *inventory_file* is the pathname and the filename pointing to the Ansible's inventory host file.

Variable File Separation

It's a great idea to keep your playbooks under source control, but you may wish to make the playbook source public while keeping certain important variables private. Similarly, sometimes you may just want to keep certain information in different files, away from the main playbook.

You can do this by using an external variables file, or files, just like this:

```
---
- hosts: all
  remote_user: root
  vars:
    favcolor: blue
  vars_files:
    - /vars/external_vars.yml
```

```
  tasks:
  - name: this is just a placeholder
    command: /bin/echo foo
```

This removes the risk of sharing sensitive data with others when sharing your playbook source with them.

The contents of each variables file is a simple YAML dictionary, like this:

```
---
# in the above example, this would be vars/external_vars.yml
somevar: somevalue
password: magic
```

Note: It's also possible to keep per-host and per-group variables in very similar files, this is covered in *Patterns*.

Passing Variables On The Command Line

In addition to *vars_prompt* and *vars_files*, it is possible to send variables over the Ansible command line. This is particularly useful when writing a generic release playbook where you may want to pass in the version of the application to deploy:

```
ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
```

This is useful, for, among other things, setting the hosts group or the user for the playbook.

Example:

```
---
- remote_user: '{{ user }}'
  hosts: '{{ hosts }}'
  tasks:
     - ...

ansible-playbook release.yml --extra-vars "hosts=vipers user=starbuck"
```

As of Ansible 1.2, you can also pass in extra vars as quoted JSON, like so:

```
--extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'
```

The key=value form is obviously simpler, but it's there if you need it!

As of Ansible 1.3, extra vars can be loaded from a JSON file with the "@" syntax:

```
--extra-vars "@some_file.json"
```

Also as of Ansible 1.3, extra vars can be formatted as YAML, either on the command line or in a file as above.

Conditional Imports

Note: This behavior is infrequently used in Ansible. You may wish to skip this section. The 'group_by' module as described in the module documentation is a better way to achieve this behavior in most cases.

Sometimes you will want to do certain things differently in a playbook based on certain criteria. Having one playbook that works on multiple platforms and OS versions is a good example.

As an example, the name of the Apache package may be different between CentOS and Debian, but it is easily handled with a minimum of syntax in an Ansible Playbook:

```
---
- hosts: all
  remote_user: root
  vars_files:
    - "vars/common.yml"
    - [ "vars/{{ ansible_os_family }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: make sure apache is running
    service: name={{ apache }} state=running
```

Note: The variable 'ansible_os_family' is being interpolated into the list of filenames being defined for vars_files.

As a reminder, the various YAML files contain just keys and values:

```
---
# for vars/CentOS.yml
apache: httpd
somethingelse: 42
```

How does this work? If the operating system was 'CentOS', the first file Ansible would try to import would be 'vars/CentOS.yml', followed by '/vars/os_defaults.yml' if that file did not exist. If no files in the list were found, an error would be raised. On Debian, it would instead first look towards 'vars/Debian.yml' instead of 'vars/CentOS.yml', before falling back on 'vars/os_defaults.yml'. Pretty simple.

To use this conditional import feature, you'll need facter or ohai installed prior to running the playbook, but you can of course push this out with Ansible if you like:

```
# for facter
ansible -m yum -a "pkg=facter ensure=installed"
ansible -m yum -a "pkg=ruby-json ensure=installed"

# for ohai
ansible -m yum -a "pkg=ohai ensure=installed"
```

Ansible's approach to configuration – separating variables from tasks, keeps your playbooks from turning into arbitrary code with ugly nested ifs, conditionals, and so on - and results in more streamlined & auditable configuration rules – especially because there are a minimum of decision points to track.

Variable Precedence: Where Should I Put A Variable?

A lot of folks may ask about how variables override another. Ultimately it's Ansible's philosophy that it's better you know where to put a variable, and then you have to think about it a lot less.

Avoid defining the variable "x" in 47 places and then ask the question "which x gets used". Why? Because that's not Ansible's Zen philosophy of doing things.

There is only one Empire State Building. One Mona Lisa, etc. Figure out where to define a variable, and don't make it complicated.

However, let's go ahead and get precedence out of the way! It exists. It's a real thing, and you might have a use for it.

If multiple variables of the same name are defined in different places, they win in a certain order, which is:

```
* -e variables always win
* then comes "most everything else"
* then comes variables defined in inventory
* then "role defaults", which are the most "defaulty" and lose in priority to everything.
```

That seems a little theoretical. Let's show some examples and where you would choose to put what based on the kind of control you might want over values.

First off, group variables are super powerful.

Site wide defaults should be defined as a 'group_vars/all' setting. Group variables are generally placed alongside your inventory file. They can also be returned by a dynamic inventory script (see *Dynamic Inventory*) or defined in things like *Ansible Tower* from the UI or API:

```
---
# file: /etc/ansible/group_vars/all
# this is the site wide default
ntp_server: default-time.example.com
```

Regional information might be defined in a 'group_vars/region' variable. If this group is a child of the 'all' group (which it is, because all groups are), it will override the group that is higher up and more general:

```
---
# file: /etc/ansible/group_vars/boston
ntp_server: boston-time.example.com
```

If for some crazy reason we wanted to tell just a specific host to use a specific NTP server, it would then override the group variable!:

```
---
# file: /etc/ansible/host_vars/xyz.boston.example.com
ntp_server: override.example.com
```

So that covers inventory and what you would normally set there. It's a great place for things that deal with geography or behavior. Since groups are frequently the entity that maps roles onto hosts, it is sometimes a shortcut to set variables on the group instead of defining them on a role. You could go either way.

Remember: Child groups override parent groups, and hosts always override their groups.

Next up: learning about role variable precedence.

We'll pretty much assume you are using roles at this point. You should be using roles for sure. Roles are great. You are using roles aren't you? Hint hint.

Ok, so if you are writing a redistributable role with reasonable defaults, put those in the 'roles/x/defaults/main.yml' file. This means the role will bring along a default value but ANYTHING in Ansible will override it. It's just a default. That's why it says "defaults" :) See *Playbook Roles and Include Statements* for more info about this:

```
---
# file: roles/x/defaults/main.yml
# if not overriden in inventory or as a parameter, this is the value that will be used
http_port: 80
```

if you are writing a role and want to ensure the value in the role is absolutely used in that role, and is not going to be overridden by inventory, you should but it in roles/x/vars/main.yml like so, and inventory values cannot override it. -e however, still will:

```
---
# file: roles/x/vars/main.yml
# this will absolutely be used in this role
http_port: 80
```

So the above is a great way to plug in constants about the role that are always true. If you are not sharing your role with others, app specific behaviors like ports is fine to put in here. But if you are sharing roles with others, putting variables in here might be bad. Nobody will be able to override them with inventory, but they still can by passing a parameter to the role.

Parameterized roles are useful.

If you are using a role and want to override a default, pass it as a parameter to the role like so:

```
roles:
   - { name: apache, http_port: 8080 }
```

This makes it clear to the playbook reader that you've made a conscious choice to override some default in the role, or pass in some configuration that the role can't assume by itself. It also allows you to pass something site-specific that isn't really part of the role you are sharing with others.

This can often be used for things that might apply to some hosts multiple times, like so:

```
roles:
   - { role: app_user, name: Ian    }
   - { role: app_user, name: Terry  }
   - { role: app_user, name: Graham }
   - { role: app_user, name: John   }
```

That's a bit arbitrary, but you can see how the same role was invoked multiple Times. In that example it's quite likely there was no default for 'name' supplied at all. Ansible can yell at you when variables aren't defined – it's the default behavior in fact.

So that's a bit about roles.

There are a few bonus things that go on with roles.

Generally speaking, variables set in one role are available to others. This means if you have a "roles/common/vars/main.yml" you can set variables in there and make use of them in other roles and elsewhere in your playbook:

```
roles:
   - { role: common_settings }
   - { role: something, foo: 12 }
   - { role: something_else }
```

Note: There are some protections in place to avoid the need to namespace variables. In the above, variables defined in common_settings are most definitely available to 'app_user' and 'something_else' tasks, but if "something's" guaranteed to have foo set at 12, even if somewhere deep in common settings it set foo to 20.

So, that's precedence, explained in a more direct way. Don't worry about precedence, just think about if your role is defining a variable that is a default, or a "live" variable you definitely want to use. Inventory lies in precedence right in the middle, and if you want to forcibly override something, use -e.

If you found that a little hard to understand, take a look at the ansible-examples repo on our github for a bit more about how all of these things can work together.

See also:

Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Loops Looping in playbooks

Playbook Roles and Include Statements Playbook organization by roles

Best Practices Best practices in playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.3.4 Conditionals

Topics

- Conditionals
  - The When Statement
  - Loading in Custom Facts
  - Applying 'when' to roles and includes
  - Conditional Imports
  - Selecting Files And Templates Based On Variables
  - Register Variables

Often the result of a play may depend on the value of a variable, fact (something learned about the remote system), or previous task result. In some cases, the values of variables may depend on other variables. Further, additional groups can be created to manage hosts based on whether the hosts match other criteria. There are many options to control execution flow in Ansible.

Let's dig into what they are.

Contents

- Conditionals
  - The When Statement
  - Loading in Custom Facts
  - Applying 'when' to roles and includes
  - Conditional Imports
  - Selecting Files And Templates Based On Variables
  - Register Variables

The When Statement

Sometimes you will want to skip a particular step on a particular host. This could be something as simple as not installing a certain package if the operating system is a particular version, or it could be something like performing some cleanup steps if a filesystem is getting full.

This is easy to do in Ansible, with the *when* clause, which contains a Jinja2 expression (see *Variables*). It's actually pretty simple:

```
tasks:
  - name: "shutdown Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
```

A number of Jinja2 "filters" can also be used in when statements, some of which are unique and provided by Ansible. Suppose we want to ignore the error of one statement and then decide to do something conditionally based on success or failure:

```
tasks:
  - command: /bin/false
    register: result
    ignore_errors: True
  - command: /bin/something
    when: result|failed
  - command: /bin/something_else
```

```
    when: result|success
  - command: /bin/still/something_else
    when: result|skipped
```

Note that was a little bit of foreshadowing on the 'register' statement. We'll get to it a bit later in this chapter.

As a reminder, to see what facts are available on a particular system, you can do:

```
ansible hostname.example.com -m setup
```

Tip: Sometimes you'll get back a variable that's a string and you'll want to do a math operation comparison on it. You can do this like so:

```
tasks:
  - shell: echo "only on Red Hat 6, derivatives, and later"
    when: ansible_os_family == "RedHat" and ansible_lsb.major_release|int >= 6
```

Note: the above example requires the lsb_release package on the target host in order to return the ansible_lsb.major_release fact.

Variables defined in the playbooks or inventory can also be used. An example may be the execution of a task based on a variable's boolean value:

```
vars:
  epic: true
```

Then a conditional execution might look like:

```
tasks:
    - shell: echo "This certainly is epic!"
      when: epic
```

or:

```
tasks:
    - shell: echo "This certainly isn't epic!"
      when: not epic
```

If a required variable has not been set, you can skip or fail using Jinja2's *defined* test. For example:

```
tasks:
    - shell: echo "I've got '{{ foo }}' and am not afraid to use it!"
      when: foo is defined

    - fail: msg="Bailing out. this play requires 'bar'"
      when: bar is not defined
```

This is especially useful in combination with the conditional import of vars files (see below).

Note that when combining *when* with *with_items* (see *Loops*), be aware that the *when* statement is processed separately for each item. This is by design:

```
tasks:
    - command: echo {{ item }}
      with_items: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

Loading in Custom Facts

It's also easy to provide your own facts if you want, which is covered in *Developing Modules*. To run them, just make a call to your own custom fact gathering module at the top of your list of tasks, and variables returned there will be accessible to future tasks:

```
tasks:
    - name: gather site specific fact data
      action: site_facts
    - command: /usr/bin/thingy
      when: my_custom_fact_just_retrieved_from_the_remote_system == '1234'
```

Applying 'when' to roles and includes

Note that if you have several tasks that all share the same conditional statement, you can affix the conditional to a task include statement as below. Note this does not work with playbook includes, just task includes. All the tasks get evaluated, but the conditional is applied to each and every task:

```
- include: tasks/sometasks.yml
  when: "'reticulating splines' in output"
```

Or with a role:

```
- hosts: webservers
  roles:
     - { role: debian_stock_config, when: ansible_os_family == 'Debian' }
```

You will note a lot of 'skipped' output by default in Ansible when using this approach on systems that don't match the criteria. Read up on the 'group_by' module in the *About Modules* docs for a more streamlined way to accomplish the same thing.

Conditional Imports

Note: This is an advanced topic that is infrequently used. You can probably skip this section.

Sometimes you will want to do certain things differently in a playbook based on certain criteria. Having one playbook that works on multiple platforms and OS versions is a good example.

As an example, the name of the Apache package may be different between CentOS and Debian, but it is easily handled with a minimum of syntax in an Ansible Playbook:

```
---
- hosts: all
  remote_user: root
  vars_files:
    - "vars/common.yml"
    - [ "vars/{{ ansible_os_family }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: make sure apache is running
    service: name={{ apache }} state=running
```

Note: The variable 'ansible_os_family' is being interpolated into the list of filenames being defined for vars_files.

As a reminder, the various YAML files contain just keys and values:

```
---
# for vars/CentOS.yml
apache: httpd
somethingelse: 42
```

How does this work? If the operating system was 'CentOS', the first file Ansible would try to import would be 'vars/CentOS.yml', followed by '/vars/os_defaults.yml' if that file did not exist. If no files in the list were found, an error would be raised. On Debian, it would instead first look towards 'vars/Debian.yml' instead of 'vars/CentOS.yml', before falling back on 'vars/os_defaults.yml'. Pretty simple.

To use this conditional import feature, you'll need facter or ohai installed prior to running the playbook, but you can of course push this out with Ansible if you like:

```
# for facter
ansible -m yum -a "pkg=facter ensure=installed"
ansible -m yum -a "pkg=ruby-json ensure=installed"

# for ohai
ansible -m yum -a "pkg=ohai ensure=installed"
```

Ansible's approach to configuration – separating variables from tasks, keeps your playbooks from turning into arbitrary code with ugly nested ifs, conditionals, and so on - and results in more streamlined & auditable configuration rules – especially because there are a minimum of decision points to track.

Selecting Files And Templates Based On Variables

Note: This is an advanced topic that is infrequently used. You can probably skip this section.

Sometimes a configuration file you want to copy, or a template you will use may depend on a variable. The following construct selects the first available file appropriate for the variables of a given host, which is often much cleaner than putting a lot of if conditionals in a template.

The following example shows how to template out a configuration file that was very different between, say, CentOS and Debian:

```
- name: template a file
  template: src={{ item }} dest=/etc/myapp/foo.conf
  with_first_found:
    - files:
       - {{ ansible_distribution }}.conf
       - default.conf
      paths:
       - search_location_one/somedir/
       - /opt/other_location/somedir/
```

Register Variables

Often in a playbook it may be useful to store the result of a given command in a variable and access it later. Use of the command module in this way can in many ways eliminate the need to write site specific facts, for instance, you could test for the existence of a particular program.

The 'register' keyword decides what variable to save a result in. The resulting variables can be used in templates, action lines, or *when* statements. It looks like this (in an obviously trivial example):

```
- name: test play
  hosts: all

  tasks:

      - shell: cat /etc/motd
        register: motd_contents

      - shell: echo "motd contains the word hi"
        when: motd_contents.stdout.find('hi') != -1
```

As shown previously, the registered variable's string contents are accessible with the 'stdout' value. The registered result can be used in the "with_items" of a task if it is converted into a list (or already is a list) as shown below. "stdout_lines" is already available on the object as well though you could also call "home_dirs.stdout.split()" if you wanted, and could split by other fields:

```
- name: registered variable usage as a with_items list
  hosts: all

  tasks:

      - name: retrieve the list of home directories
        command: ls /home
        register: home_dirs

      - name: add home dirs to the backup spooler
        file: path=/mnt/bkspool/{{ item }} src=/home/{{ item }} state=link
        with_items: home_dirs.stdout_lines
        # same as with_items: home_dirs.stdout.split()
```

See also:

Playbooks An introduction to playbooks

Playbook Roles and Include Statements Playbook organization by roles

Best Practices Best practices in playbooks

Conditionals Conditional statements in playbooks

Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.3.5 Loops

Often you'll want to do many things in one task, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached.

This chapter is all about how to use loops in playbooks.

Topics

- Loops
  - Standard Loops
  - Nested Loops
  - Looping over Fileglobs
  - Looping over Parallel Sets of Data
  - Looping over Subelements
  - Looping over Integer Sequences
  - Random Choices
  - Do-Until Loops
  - Finding First Matched Files
  - Iterating Over The Results of a Program Execution
  - Looping Over A List With An Index
  - Flattening A List
  - Using register with a loop
  - Writing Your Own Iterators

Standard Loops

To save some typing, repeated tasks can be written in short-hand like so:

```
- name: add several users
  user: name={{ item }} state=present groups=wheel
  with_items:
     - testuser1
     - testuser2
```

If you have defined a YAML list in a variables file, or the 'vars' section, you can also do:

```
with_items: somelist
```

The above would be the equivalent of:

```
- name: add user testuser1
  user: name=testuser1 state=present groups=wheel
- name: add user testuser2
  user: name=testuser2 state=present groups=wheel
```

The yum and apt modules use with_items to execute fewer package manager transactions.

Note that the types of items you iterate over with 'with_items' do not have to be simple lists of strings. If you have a list of hashes, you can reference subkeys using things like:

```
- name: add several users
  user: name={{ item.name }} state=present groups={{ item.groups }}
  with_items:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }
```

Nested Loops

Loops can be nested as well:

```
- name: give users access to multiple databases
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:ALL append_privs=yes password=foo
  with_nested:
    - [ 'alice', 'bob', 'eve' ]
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

As with the case of 'with_items' above, you can use previously defined variables. Just specify the variable's name without templating it with '{{ }}':

```
- name: here, 'users' contains the above list of employees
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:ALL append_privs=yes password=foo
  with_nested:
    - users
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

Looping over Fileglobs

`with_fileglob` matches all files in a single directory, non-recursively, that match a pattern. It can be used like this:

```
---
- hosts: all

  tasks:

    # first ensure our target directory exists
    - file: dest=/etc/fooapp state=directory

    # copy each file over that matches the given pattern
    - copy: src={{ item }} dest=/etc/fooapp/ owner=root mode=600
      with_fileglob:
        - /playbooks/files/fooapp/*
```

Looping over Parallel Sets of Data

Note: This is an uncommon thing to want to do, but we're documenting it for completeness. You probably won't be reaching for this one often.

Suppose you have the following variable data was loaded in via somewhere:

```
---
alpha: [ 'a', 'b', 'c', 'd' ]
numbers:  [ 1, 2, 3, 4 ]
```

And you want the set of '(a, 1)' and '(b, 2)' and so on. Use 'with_together' to get this:

```
tasks:
    - debug: msg="{{ item.0 }} and {{ item.1 }}"
      with_together:
        - alpha
        - numbers
```

Looping over Subelements

Suppose you want to do something like loop over a list of users, creating them, and allowing them to login by a certain set of SSH keys.

How might that be accomplished? Let's assume you had the following defined and loaded in via "vars_files" or maybe a "group_vars/all" file:

```
---
users:
  - name: alice
    authorized:
      - /tmp/alice/onekey.pub
      - /tmp/alice/twokey.pub
  - name: bob
    authorized:
      - /tmp/bob/id_rsa.pub
```

It might happen like so:

```
- user: name={{ item.name }} state=present generate_ssh_key=yes
  with_items: users

- authorized_key: "user={{ item.0.name }} key='{{ lookup('file', item.1) }}'"
  with_subelements:
     - users
     - authorized
```

Subelements walks a list of hashes (aka dictionaries) and then traverses a list with a given key inside of those records.

The authorized_key pattern is exactly where it comes up most.

Looping over Integer Sequences

`with_sequence` generates a sequence of items in ascending numerical order. You can specify a start, end, and an optional step value.

Arguments should be specified in key=value pairs. If supplied, the 'format' is a printf style string.

Numerical values can be specified in decimal, hexadecimal (0x3f8) or octal (0600). Negative numbers are not supported. This works as follows:

```
---
- hosts: all

  tasks:

    # create groups
    - group: name=evens state=present
    - group: name=odds state=present

    # create some test users
    - user: name={{ item }} state=present groups=evens
      with_sequence: start=0 end=32 format=testuser%02x

    # create a series of directories with even numbers for some reason
    - file: dest=/var/stuff/{{ item }} state=directory
      with_sequence: start=4 end=16 stride=2

    # a simpler way to use the sequence plugin
```

```
    # create 4 groups
    - group: name=group{{ item }} state=present
      with_sequence: count=4
```

Random Choices

The 'random_choice' feature can be used to pick something at random. While it's not a load balancer (there are modules for those), it can somewhat be used as a poor man's loadbalancer in a MacGyver like situation:

```
- debug: msg={{ item }}
  with_random_choice:
     - "go through the door"
     - "drink from the goblet"
     - "press the red button"
     - "do nothing"
```

One of the provided strings will be selected at random.

At a more basic level, they can be used to add chaos and excitement to otherwise predictable automation environments.

Do-Until Loops

Sometimes you would want to retry a task until a certain condition is met. Here's an example:

```
- action: shell /usr/bin/foo
  register: result
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

The above example run the shell module recursively till the module's result has "all systems go" in it's stdout or the task has been retried for 5 times with a delay of 10 seconds. The default value for "retries" is 3 and "delay" is 5.

The task returns the results returned by the last task run. The results of individual retries can be viewed by -vv option. The registered variable will also have a new key "attempts" which will have the number of the retries for the task.

Finding First Matched Files

Note: This is an uncommon thing to want to do, but we're documenting it for completeness. You probably won't be reaching for this one often.

This isn't exactly a loop, but it's close. What if you want to use a reference to a file based on the first file found that matches a given criteria, and some of the filenames are determined by variable names? Yes, you can do that as follows:

```
- name: INTERFACES | Create Ansible header for /etc/network/interfaces
  template: src={{ item }} dest=/etc/foo.conf
  with_first_found:
    - "{{ansible_virtualization_type}}_foo.conf"
    - "default_foo.conf"
```

This tool also has a long form version that allows for configurable search paths. Here's an example:

```
- name: some configuration template
  template: src={{ item }} dest=/etc/file.cfg mode=0444 owner=root group=root
  with_first_found:
```

```
  - files:
     - "{{inventory_hostname}}/etc/file.cfg"
    paths:
     - ../../../templates.overwrites
     - ../../../templates
  - files:
      - etc/file.cfg
    paths:
      - templates
```

Iterating Over The Results of a Program Execution

Note: This is an uncommon thing to want to do, but we're documenting it for completeness. You probably won't be reaching for this one often.

Sometimes you might want to execute a program, and based on the output of that program, loop over the results of that line by line. Ansible provides a neat way to do that, though you should remember, this is always executed on the control machine, not the local machine:

```
- name: Example of looping over a command result
  shell: /usr/bin/frobnicate {{ item }}
  with_lines: /usr/bin/frobnications_per_host --param {{ inventory_hostname }}
```

Ok, that was a bit arbitrary. In fact, if you're doing something that is inventory related you might just want to write a dynamic inventory source instead (see *Dynamic Inventory*), but this can be occasionally useful in quick-and-dirty implementations.

Should you ever need to execute a command remotely, you would not use the above method. Instead do this:

```
- name: Example of looping over a REMOTE command result
  shell: /usr/bin/something
  register: command_result

- name: Do something with each result
  shell: /usr/bin/something_else --param {{ item }}
  with_items: command_result.stdout_lines
```

Looping Over A List With An Index

Note: This is an uncommon thing to want to do, but we're documenting it for completeness. You probably won't be reaching for this one often.

If you want to loop over an array and also get the numeric index of where you are in the array as you go, you can also do that. It's uncommonly used:

```
- name: indexed loop demo
  debug: msg="at array position {{ item.0 }} there is a value {{ item.1 }}"
  with_indexed_items: some_list
```

Flattening A List

Note: This is an uncommon thing to want to do, but we're documenting it for completeness. You probably won't be

reaching for this one often.

In rare instances you might have several lists of lists, and you just want to iterate over every item in all of those lists. Assume a really crazy hypothetical datastructure:

```
---
# file: roles/foo/vars/main.yml
packages_base:
  - [ 'foo-package', 'bar-package' ]
packages_apps:
  - [ ['one-package', 'two-package' ]]
  - [ ['red-package'], ['blue-package']]
```

As you can see the formatting of packages in these lists is all over the place. How can we install all of the packages in both lists?:

```
- name: flattened loop demo
  yum: name={{ item }} state=installed
  with_flattened:
     - packages_base
     - packages_apps
```

That's how!

Using register with a loop

When using `register` with a loop the data structure placed in the variable during a loop, will contain a `results` attribute, that is a list of all responses from the module.

Here is an example of using `register` with `with_items`:

```
- shell: echo "{{ item }}"
  with_items:
    - one
    - two
  register: echo
```

This differs from the data structure returned when using `register` without a loop:

```
{
    "changed": true,
    "msg": "All items completed",
    "results": [
        {
            "changed": true,
            "cmd": "echo \"one\" ",
            "delta": "0:00:00.003110",
            "end": "2013-12-19 12:00:05.187153",
            "invocation": {
                "module_args": "echo \"one\"",
                "module_name": "shell"
            },
            "item": "one",
            "rc": 0,
            "start": "2013-12-19 12:00:05.184043",
            "stderr": "",
            "stdout": "one"
        },
```

```
        {
            "changed": true,
            "cmd": "echo \"two\" ",
            "delta": "0:00:00.002920",
            "end": "2013-12-19 12:00:05.245502",
            "invocation": {
                "module_args": "echo \"two\"",
                "module_name": "shell"
            },
            "item": "two",
            "rc": 0,
            "start": "2013-12-19 12:00:05.242582",
            "stderr": "",
            "stdout": "two"
        }
    ]
}
```


Subsequent loops over the registered variable to inspect the results may look like:

```
- name: Fail if return code is not 0
  fail:
    msg: "The command ({{ item.cmd }}) did not have a 0 return code"
  when: item.rc != 0
  with_items: echo.results
```

Writing Your Own Iterators

While you ordinarily shouldn't have to, should you wish to write your own ways to loop over arbitrary datastructures, you can read *Developing Plugins* for some starter information. Each of the above features are implemented as plugins in ansible, so there are many implementations to reference.

See also:

Playbooks An introduction to playbooks

Playbook Roles and Include Statements Playbook organization by roles

Best Practices Best practices in playbooks

Conditionals Conditional statements in playbooks

Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.3.6 Best Practices

Here are some tips for making the most of Ansible playbooks.

You can find some example playbooks illustrating these best practices in our ansible-examples repository. (NOTE: These may not use all of the features in the latest release, but are still an excellent reference!).

Topics

- Best Practices
  - Content Organization
    - Directory Layout
    - How to Arrange Inventory, Stage vs Production
    - Group And Host Variables
    - Top Level Playbooks Are Separated By Role
    - Task And Handler Organization For A Role
    - What This Organization Enables (Examples)
    - Deployment vs Configuration Organization
  - Stage vs Production
  - Rolling Updates
  - Always Mention The State
  - Group By Roles
  - Operating System and Distribution Variance
  - Bundling Ansible Modules With Playbooks
  - Whitespace and Comments
  - Always Name Tasks
  - Keep It Simple
  - Version Control

Content Organization

The following section shows one of many possible ways to organize playbook content. Your usage of Ansible should fit your needs, however, not ours, so feel free to modify this approach and organize as you see fit.

(One thing you will definitely want to do though, is use the "roles" organization feature, which is documented as part of the main playbooks page. See *Playbook Roles and Include Statements*).

Directory Layout

The top level of the directory would contain files and directories like so:

```
production                # inventory file for production servers
stage                     # inventory file for stage environment

group_vars/
   group1                 # here we assign variables to particular groups
   group2                 # ""
host_vars/
   hostname1              # if systems need specific variables, put them here
   hostname2              # ""

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier

roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
```

```
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

How to Arrange Inventory, Stage vs Production

In the example below, the *production* file contains the inventory of all of your production hosts. Of course you can pull inventory from an external data source as well, but this is just a basic example.

It is suggested that you define groups based on purpose of the host (roles) and also geography or datacenter location (if applicable):

```
# file: production

[atlanta-webservers]
www-atl-1.example.com
www-atl-2.example.com

[boston-webservers]
www-bos-1.example.com
www-bos-2.example.com

[atlanta-dbservers]
db-atl-1.example.com
db-atl-2.example.com

[boston-dbservers]
db-bos-1.example.com

# webservers in all geos
[webservers:children]
atlanta-webservers
boston-webservers

# dbservers in all geos
[dbservers:children]
atlanta-dbservers
boston-dbservers

# everything in the atlanta geo
[atlanta:children]
atlanta-webservers
atlanta-dbservers

# everything in the boston geo
[boston:children]
boston-webservers
boston-dbservers
```

Group And Host Variables

Now, groups are nice for organization, but that's not all groups are good for. You can also assign variables to them! For instance, atlanta has its own NTP servers, so when setting up ntp.conf, we should use them. Let's set those now:

```
---
# file: group_vars/atlanta
ntp: ntp-atlanta.example.com
backup: backup-atlanta.example.com
```

Variables aren't just for geographic information either! Maybe the webservers have some configuration that doesn't make sense for the database servers:

```
---
# file: group_vars/webservers
apacheMaxRequestsPerChild: 3000
apacheMaxClients: 900
```

If we had any default values, or values that were universally true, we would put them in a file called group_vars/all:

```
---
# file: group_vars/all
ntp: ntp-boston.example.com
backup: backup-boston.example.com
```

We can define specific hardware variance in systems in a host_vars file, but avoid doing this unless you need to:

```
---
# file: host_vars/db-bos-1.example.com
foo_agent_port: 86
bar_agent_port: 99
```

Top Level Playbooks Are Separated By Role

In site.yml, we include a playbook that defines our entire infrastructure. Note this is SUPER short, because it's just including some other playbooks. Remember, playbooks are nothing more than lists of plays:

```
---
# file: site.yml
- include: webservers.yml
- include: dbservers.yml
```

In a file like webservers.yml (also at the top level), we simply map the configuration of the webservers group to the roles performed by the webservers group. Also notice this is incredibly short. For example:

```
---
# file: webservers.yml
- hosts: webservers
  roles:
    - common
    - webtier
```

Task And Handler Organization For A Role

Below is an example tasks file that explains how a role works. Our common role here just sets up NTP, but it could do more if we wanted:

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum: pkg=ntp state=installed
  tags: ntp

- name: be sure ntp is configured
  template: src=ntp.conf.j2 dest=/etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp

- name: be sure ntpd is running and enabled
  service: name=ntpd state=running enabled=yes
  tags: ntp
```

Here is an example handlers file. As a review, handlers are only fired when certain tasks report changes, and are run at the end of each play:

```
---
# file: roles/common/handlers/main.yml
- name: restart ntpd
  service: name=ntpd state=restarted
```

See *Playbook Roles and Include Statements* for more information.

What This Organization Enables (Examples)

Above we've shared our basic organizational structure.

Now what sort of use cases does this layout enable? Lots! If I want to reconfigure my whole infrastructure, it's just:

```
ansible-playbook -i production site.yml
```

What about just reconfiguring NTP on everything? Easy.:

```
ansible-playbook -i production site.yml --tags ntp
```

What about just reconfiguring my webservers?:

```
ansible-playbook -i production webservers.yml
```

What about just my webservers in Boston?:

```
ansible-playbook -i production webservers.yml --limit boston
```

What about just the first 10, and then the next 10?:

```
ansible-playbook -i production webservers.yml --limit boston[0-10]
ansible-playbook -i production webservers.yml --limit boston[10-20]
```

And of course just basic ad-hoc stuff is also possible.:

```
ansible -i production -m ping
ansible -i production -m command -a '/sbin/reboot' --limit boston
```

And there are some useful commands to know (at least in 1.1 and higher):

```
# confirm what task names would be run if I ran this command and said "just ntp tasks"
ansible-playbook -i production webservers.yml --tags ntp --list-tasks

# confirm what hostnames might be communicated with if I said "limit to boston"
ansible-playbook -i production webservers.yml --limit boston --list-hosts
```

Deployment vs Configuration Organization

The above setup models a typical configuration topology. When doing multi-tier deployments, there are going to be some additional playbooks that hop between tiers to roll out an application. In this case, 'site.yml' may be augmented by playbooks like 'deploy_exampledotcom.yml' but the general concepts can still apply.

Consider "playbooks" as a sports metaphor – you don't have to just have one set of plays to use against your infrastructure all the time – you can have situational plays that you use at different times and for different purposes.

Ansible allows you to deploy and configure using the same tool, so you would likely reuse groups and just keep the OS configuration in separate playbooks from the app deployment.

Stage vs Production

As also mentioned above, a good way to keep your stage (or testing) and production environments separate is to use a separate inventory file for stage and production. This way you pick with -i what you are targeting. Keeping them all in one file can lead to surprises!

Testing things in a stage environment before trying in production is always a great idea. Your environments need not be the same size and you can use group variables to control the differences between those environments.

Rolling Updates

Understand the 'serial' keyword. If updating a webserver farm you really want to use it to control how many machines you are updating at once in the batch.

See *Delegation, Rolling Updates, and Local Actions*.

Always Mention The State

The 'state' parameter is optional to a lot of modules. Whether 'state=present' or 'state=absent', it's always best to leave that parameter in your playbooks to make it clear, especially as some modules support additional states.

Group By Roles

A system can be in multiple groups. See *Inventory* and *Patterns*. Having groups named after things like *webservers* and *dbservers* is repeated in the examples because it's a very powerful concept.

This allows playbooks to target machines based on role, as well as to assign role specific variables using the group variable system.

See *Playbook Roles and Include Statements*.

Operating System and Distribution Variance

When dealing with a parameter that is different between two different operating systems, the best way to handle this is by using the group_by module.

This makes a dynamic group of hosts matching certain criteria, even if that group is not defined in the inventory file:

```
---

# talk to all hosts just so we can learn about them

- hosts: all
  tasks:
     - group_by: key={{ ansible_distribution }}

# now just on the CentOS hosts...

- hosts: CentOS
  gather_facts: False
  tasks:
     - # tasks that only happen on CentOS go here
```

If group-specific settings are needed, this can also be done. For example:

```
---
# file: group_vars/all
asdf: 10

---
# file: group_vars/CentOS
asdf: 42
```

In the above example, CentOS machines get the value of '42' for asdf, but other machines get '10'.

Bundling Ansible Modules With Playbooks

New in version 0.5.

If a playbook has a "./library" directory relative to its YAML file, this directory can be used to add ansible modules that will automatically be in the ansible module path. This is a great way to keep modules that go with a playbook together.

Whitespace and Comments

Generous use of whitespace to break things up, and use of comments (which start with '#'), is encouraged.

Always Name Tasks

It is possible to leave off the 'name' for a given task, though it is recommended to provide a description about why something is being done instead. This name is shown when the playbook is run.

Keep It Simple

When you can do something simply, do something simply. Do not reach to use every feature of Ansible together, all at once. Use what works for you. For example, you will probably not need `vars`, `vars_files`, `vars_prompt` and `--extra-vars` all at once, while also using an external inventory file.

Version Control

Use version control. Keep your playbooks and inventory file in git (or another version control system), and commit when you make changes to them. This way you have an audit trail describing when and why you changed the rules that are automating your infrastructure.

See also:

YAML Syntax Learn about YAML syntax

Playbooks Review the basic playbook features

About Modules Learn about available modules

Developing Modules Learn how to extend Ansible by writing your own modules

Patterns Learn about how to select hosts

Github examples directory Complete playbook files from the github project source

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

1.4 Playbooks: Special Topics

Here are some playbook features that not everyone may need to learn, but can be quite useful for particular applications. Browsing these topics is recommended as you may find some useful tips here, but feel free to learn the basics of Ansible first and adopt these only if they seem relevant or useful to your environment.

1.4.1 Accelerated Mode

New in version 1.3.

While OpenSSH using the ControlPersist feature is quite fast and scalable, there is a certain small amount of overhead involved in using SSH connections. While many people will not encounter a need, if you are running on a platform that doesn't have ControlPersist support (such as an EL6 control machine), you'll probably be even more interested in tuning options.

Accelerate mode is there to help connections work faster, but still uses SSH for initial secure key exchange. There is no additional public key infrastructure to manage, and this does not require things like NTP or even DNS.

Accelerated mode can be anywhere from 2-6x faster than SSH with ControlPersist enabled, and 10x faster than paramiko.

Accelerated mode works by launching a temporary daemon over SSH. Once the daemon is running, Ansible will connect directly to it via a socket connection. Ansible secures this communication by using a temporary AES key that is exchanged during the SSH connection (this key is different for every host, and is also regenerated periodically).

By default, Ansible will use port 5099 for the accelerated connection, though this is configurable. Once running, the daemon will accept connections for 30 minutes, after which time it will terminate itself and need to be restarted over SSH.

Accelerated mode offers several improvements over the original fireball mode from which it was based:

- No bootstrapping is required, only a single line needs to be added to each play you wish to run in accelerated mode.
- Support for sudo commands (see below for more details and caveats) is available.
- There are fewer requirements. ZeroMQ is no longer required, nor are there any special packages beyond python-keyczar

- python 2.5 or higher is required.

In order to use accelerated mode, simply add *accelerate: true* to your play:

```
---
- hosts: all
  accelerate: true
  tasks:
  - name: some task
    command: echo {{ item }}
    with_items:
    - foo
    - bar
    - baz
```

If you wish to change the port Ansible will use for the accelerated connection, just add the *accelerated_port* option:

```
---
- hosts: all
  accelerate: true
  # default port is 5099
  accelerate_port: 10000
```

The *accelerate_port* option can also be specified in the environment variable ACCELERATE_PORT, or in your *ansible.cfg* configuration:

```
[accelerate]
accelerate_port = 5099
```

As noted above, accelerated mode also supports running tasks via sudo, however there are two important caveats:

- You must remove requiretty from your sudoers options.
- Prompting for the sudo password is not yet supported, so the NOPASSWD option is required for sudo'ed commands.

Fireball Mode

New in version 0.8: (deprecated as of 1.3)

Note: The following section has been deprecated as of Ansible 1.3 in favor of the accelerated mode described above. This documentation is here for users who may still be using the original fireball connection method only, and should not be used for any new deployments.

Ansible's core connection types of 'local', 'paramiko', and 'ssh' are augmented in version 0.8 and later by a new extra-fast connection type called 'fireball'. It can only be used with playbooks and does require some additional setup outside the lines of Ansible's normal "no bootstrapping" philosophy. You are not required to use fireball mode to use Ansible, though some users may appreciate it.

Fireball mode works by launching a temporary 0mq daemon from SSH that by default lives for only 30 minutes before shutting off. Fireball mode, once running, uses temporary AES keys to encrypt a session, and requires direct communication to given nodes on the configured port. The default is 5099. The fireball daemon runs as any user you set it down as. So it can run as you, root, or so on. If multiple users are running Ansible as the same batch of hosts, take care to use unique ports.

Fireball mode is roughly 10 times faster than paramiko for communicating with nodes and may be a good option if you have a large number of hosts:

```
---

# set up the fireball transport
- hosts: all
  gather_facts: no
  connection: ssh # or paramiko
  sudo: yes
  tasks:
      - action: fireball

# these operations will occur over the fireball transport
- hosts: all
  connection: fireball
  tasks:
      - shell: echo "Hello {{ item }}"
        with_items:
            - one
            - two
```

In order to use fireball mode, certain dependencies must be installed on both ends. You can use this playbook as a basis for initial bootstrapping on any platform. You will also need gcc and zeromq-devel installed from your package manager, which you can of course also get Ansible to install:

```
---
- hosts: all
  sudo: yes
  gather_facts: no
  connection: ssh
  tasks:
      - easy_install: name=pip
      - pip: name={{ item }} state=present
        with_items:
          - pyzmq
          - pyasn1
          - PyCrypto
          - python-keyczar
```

Fedora and EPEL also have Ansible RPM subpackages available for fireball-dependencies.

Also see the module documentation section.

See also:

Playbooks Introductory playbook information

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.2 Asynchronous Actions and Polling

By default tasks in playbooks block, meaning the connections stay open until the task is done on each node. This may not always be desirable, or you may be running operations that take longer than the SSH timeout.

The easiest way to do this is to kick them off all at once and then poll until they are done.

You will also want to use asynchronous mode on very long running operations that might be subject to timeout.

To launch a task asynchronously, specify its maximum runtime and how frequently you would like to poll for status. The default poll value is 10 seconds if you do not specify a value for *poll*:

```
---
- hosts: all
  remote_user: root
  tasks:
  - name: simulate long running op (15 sec), wait for up to 45, poll every 5
    command: /bin/sleep 15
    async: 45
    poll: 5
```

Note: There is no default for the async time limit. If you leave off the 'async' keyword, the task runs synchronously, which is Ansible's default.

Alternatively, if you do not need to wait on the task to complete, you may "fire and forget" by specifying a poll value of 0:

```
---
- hosts: all
  remote_user: root
  tasks:
  - name: simulate long running op, allow to run for 45, fire and forget
    command: /bin/sleep 15
    async: 45
    poll: 0
```

Note: You shouldn't "fire and forget" with operations that require exclusive locks, such as yum transactions, if you expect to run other commands later in the playbook against those same resources.

Note: Using a higher value for `--forks` will result in kicking off asynchronous tasks even faster. This also increases the efficiency of polling.

See also:

Playbooks An introduction to playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.3 Check Mode ("Dry Run")

New in version 1.1.

Topics

- Check Mode ("Dry Run")
  - Running a task in check mode
  - Showing Differences with `--diff`

When ansible-playbook is executed with `--check` it will not make any changes on remote systems. Instead, any module instrumented to support 'check mode' (which contains most of the primary core modules, but it is not required that all modules do this) will report what changes they would have made rather than making them. Other modules that do not support check mode will also take no action, but just will not report what changes they might have made.

Check mode is just a simulation, and if you have steps that use conditionals that depend on the results of prior commands, it may be less useful for you. However it is great for one-node-at-time basic configuration management

use cases.

Example:

```
ansible-playbook foo.yml --check
```

Running a task in check mode

New in version 1.3.

Sometimes you may want to have a task to be executed even in check mode. To achieve this, use the *always_run* clause on the task. Its value is a Jinja2 expression, just like the *when* clause. In simple cases a boolean YAML value would be sufficient as a value.

Example:

```
tasks:

  - name: this task is run even in check mode
    command: /something/to/run --even-in-check-mode
    always_run: yes
```

As a reminder, a task with a *when* clause evaluated to false, will still be skipped even if it has a *always_run* clause evaluated to true.

Showing Differences with `--diff`

New in version 1.1.

The `--diff` option to ansible-playbook works great with `--check` (detailed above) but can also be used by itself. When this flag is supplied, if any templated files on the remote system are changed, and the ansible-playbook CLI will report back the textual changes made to the file (or, if used with `--check`, the changes that would have been made). Since the diff feature produces a large amount of output, it is best used when checking a single host at a time, like so:

```
ansible-playbook foo.yml --check --diff --limit foo.example.com
```

1.4.4 Delegation, Rolling Updates, and Local Actions

Topics

- Delegation, Rolling Updates, and Local Actions
  - Rolling Update Batch Size
  - Maximum Failure Percentage
  - Delegation
  - Local Playbooks

Being designed for multi-tier deployments since the beginning, Ansible is great at doing things on one host on behalf of another, or doing local steps with reference to some remote hosts.

This in particular this is very applicable when setting up continuous deployment infrastructure or zero downtime rolling updates, where you might be talking with load balancers or monitoring systems.

Additional features allow for tuning the orders in which things complete, and assigning a batch window size for how many machines to process at once during a rolling update.

This section covers all of these features. For examples of these items in use, please see the ansible-examples repository. There are quite a few examples of zero-downtime update procedures for different kinds of applications.

You should also consult the *About Modules* section, various modules like 'ec2_elb', 'nagios', and 'bigip_pool', and 'netscaler' dovetail neatly with the concepts mentioned here.

You'll also want to read up on *Playbook Roles and Include Statements*, as the 'pre_task' and 'post_task' concepts are the places where you would typically call these modules.

Rolling Update Batch Size

New in version 0.7.

By default, Ansible will try to manage all of the machines referenced in a play in parallel. For a rolling updates use case, you can define how many hosts Ansible should manage at a single time by using the ''serial" keyword:

```
- name: test play
  hosts: webservers
  serial: 3
```

In the above example, if we had 100 hosts, 3 hosts in the group 'webservers' would complete the play completely before moving on to the next 3 hosts.

Maximum Failure Percentage

New in version 1.3.

By default, Ansible will continue executing actions as long as there are hosts in the group that have not yet failed. In some situations, such as with the rolling updates described above, it may be desirable to abort the play when a certain threshold of failures have been reached. To achieve this, as of version 1.3 you can set a maximum failure percentage on a play as follows:

```
- hosts: webservers
  max_fail_percentage: 30
  serial: 10
```

In the above example, if more than 3 of the 10 servers in the group were to fail, the rest of the play would be aborted.

Note: The percentage set must be exceeded, not equaled. For example, if serial were set to 4 and you wanted the task to abort when 2 of the systems failed, the percentage should be set at 49 rather than 50.

Delegation

New in version 0.7.

This isn't actually rolling update specific but comes up frequently in those cases.

If you want to perform a task on one host with reference to other hosts, use the 'delegate_to' keyword on a task. This is ideal for placing nodes in a load balanced pool, or removing them. It is also very useful for controlling outage windows. Using this with the 'serial' keyword to control the number of hosts executing at one time is also a good idea:

```
---
- hosts: webservers
  serial: 5

  tasks:
  - name: take out of load balancer pool
```

```
    command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
    delegate_to: 127.0.0.1

  - name: actual steps would go here
    yum: name=acme-web-stack state=latest

  - name: add back to load balancer pool
    command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
    delegate_to: 127.0.0.1
```

These commands will run on 127.0.0.1, which is the machine running Ansible. There is also a shorthand syntax that you can use on a per-task basis: 'local_action'. Here is the same playbook as above, but using the shorthand syntax for delegating to 127.0.0.1:

```
---
# ...
  tasks:
  - name: take out of load balancer pool
    local_action: command /usr/bin/take_out_of_pool {{ inventory_hostname }}

# ...

  - name: add back to load balancer pool
    local_action: command /usr/bin/add_back_to_pool {{ inventory_hostname }}
```

A common pattern is to use a local action to call 'rsync' to recursively copy files to the managed servers. Here is an example:

```
---
# ...
  tasks:
  - name: recursively copy files from management server to target
    local_action: command rsync -a /path/to/files {{ inventory_hostname }}:/path/to/target/
```

Note that you must have passphrase-less SSH keys or an ssh-agent configured for this to work, otherwise rsync will need to ask for a passphrase.

Local Playbooks

It may be useful to use a playbook locally, rather than by connecting over SSH. This can be useful for assuring the configuration of a system by putting a playbook on a crontab. This may also be used to run a playbook inside a OS installer, such as an Anaconda kickstart.

To run an entire playbook locally, just set the "hosts:" line to "hosts:127.0.0.1" and then run the playbook like so:

```
ansible-playbook playbook.yml --connection=local
```

Alternatively, a local connection can be used in a single playbook play, even if other plays in the playbook use the default remote connection type:

```
- hosts: 127.0.0.1
  connection: local
```

See also:

Playbooks An introduction to playbooks

Ansible Examples on GitHub Many examples of full-stack deployments

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.5 Setting the Environment (and Working With Proxies)

New in version 1.1.

It is quite possible that you may need to get package updates through a proxy, or even get some package updates through a proxy and access other packages not through a proxy. Or maybe a script you might wish to call may also need certain environment variables set to run properly.

Ansible makes it easy for you to configure your environment by using the 'environment' keyword. Here is an example:

```
- hosts: all
  remote_user: root

  tasks:

    - apt: name=cobbler state=installed
      environment:
        http_proxy: http://proxy.example.com:8080
```

The environment can also be stored in a variable, and accessed like so:

```
- hosts: all
  remote_user: root

  # here we make a variable named "env" that is a dictionary
  vars:
    proxy_env:
      http_proxy: http://proxy.example.com:8080

  tasks:

    - apt: name=cobbler state=installed
      environment: proxy_env
```

While just proxy settings were shown above, any number of settings can be supplied. The most logical place to define an environment hash might be a group_vars file, like so:

```
---
# file: group_vars/boston

ntp_server: ntp.bos.example.com
backup: bak.bos.example.com
proxy_env:
  http_proxy: http://proxy.bos.example.com:8080
  https_proxy: http://proxy.bos.example.com:8080
```

See also:

Playbooks An introduction to playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.6 Error Handling In Playbooks

Topics

- Error Handling In Playbooks
  - Ignoring Failed Commands
  - Controlling What Defines Failure
  - Overriding The Changed Result

Ansible normally has defaults that make sure to check the return codes of commands and modules and it fails fast – forcing an error to be dealt with unless you decide otherwise.

Sometimes a command that returns 0 isn't an error. Sometimes a command might not always need to report that it 'changed' the remote system. This section describes how to change the default behavior of Ansible for certain tasks so output and error handling behavior is as desired.

Ignoring Failed Commands

New in version 0.6.

Generally playbooks will stop executing any more steps on a host that has a failure. Sometimes, though, you want to continue on. To do so, write a task that looks like this:

```
- name: this will not be counted as a failure
  command: /bin/false
  ignore_errors: yes
```

Note that the above system only governs the failure of the particular task, so if you have an undefined variable used, it will still raise an error that users will need to address.

Controlling What Defines Failure

New in version 1.4.

Suppose the error code of a command is meaningless and to tell if there is a failure what really matters is the output of the command, for instance if the string "FAILED" is in the output.

Ansible in 1.4 and later provides a way to specify this behavior as follows:

```
- name: this command prints FAILED when it fails
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
```

In previous version of Ansible, this can be still be accomplished as follows:

```
- name: this command prints FAILED when it fails
  command: /usr/bin/example-command -x -y -z
  register: command_result
  ignore_errors: True

- name: fail the play if the previous command did not succeed
  fail: msg="the command failed"
  when: "'FAILED' in command_result.stderr"
```

Overriding The Changed Result

New in version 1.3.

When a shell/command or other module runs it will typically report "changed" status based on whether it thinks it affected machine state.

Sometimes you will know, based on the return code or output that it did not make any changes, and wish to override the "changed" result such that it does not appear in report output or does not cause handlers to fire:

```
tasks:

  - shell: /usr/bin/billybass --mode="take me to the river"
    register: bass_result
    changed_when: "bass_result.rc != 2"

  # this will never report 'changed' status
  - shell: wall 'beep'
    changed_when: False
```

See also:

Playbooks An introduction to playbooks

Best Practices Best practices in playbooks

Conditionals Conditional statements in playbooks

Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.7 Using Lookups

Lookup plugins allow access of data in Ansible from outside sources. This can include the filesystem but also external datastores. These values are then made available using the standard templating system in Ansible, and are typically used to load variables or templates with information from those systems.

Note: This is considered an advanced feature, and many users will probably not rely on these features.

Topics

- Using Lookups
  - Intro to Lookups: Getting File Contents
  - The Password Lookup
  - More Lookups

Intro to Lookups: Getting File Contents

The file lookup is the most basic lookup type.

Contents can be read off the filesystem as follows:

```
- hosts: all
  vars:
     contents: "{{ lookup('file', '/etc/foo.txt') }}"
  tasks:
     - debug: msg="the value of foo.txt is {{ contents }}"
```

The Password Lookup

`password` generates a random plaintext password and store it in a file at a given filepath. Support for crypted save modes (as with vars_prompt) is pending. If the file exists previously, it will retrieve its contents, behaving just like with_file. Usage of variables like "{{ inventory_hostname }}" in the filepath can be used to set up random passwords per host (what simplifies password management in 'host_vars' variables).

Generated passwords contain a random mix of upper and lowercase ASCII letters, the numbers 0-9 and punctuation (". , : - _"). The default length of a generated password is 20 characters. This length can be changed by passing an extra parameter:

```
---
- hosts: all

  tasks:

    # create a mysql user with a random password:
    - mysql_user: name={{ client }}
                  password="{{ lookup('password', 'credentials/' + client + '/' + tier + '/' + role +
                  priv={{ client }}_{{ tier }}_{{ role }}.*:ALL

    (...)
```

Note: If the file already exists, no data will be written to it. If the file has contents, those contents will be read in as the password. Empty files cause the password to return as an empty string

---

Starting in version 1.4, password accepts a "chars" parameter to allow defining a custom character set in the generated passwords. It accepts comma separated list of names that are either string module attributes (ascii_letters,digits, etc) or are used literally:

```
---
- hosts: all

  tasks:

    # create a mysql user with a random password using only ascii letters:
    - mysql_user: name={{ client }}
                  password="{{ lookup('password', '/tmp/passwordfile chars=ascii') }}"
                  priv={{ client }}_{{ tier }}_{{ role }}.*:ALL

    # create a mysql user with a random password using only digits:
    - mysql_user: name={{ client }}
                  password="{{ lookup('password', '/tmp/passwordfile chars=digits') }}"
                  priv={{ client }}_{{ tier }}_{{ role }}.*:ALL

    # create a mysql user with a random password using many different char sets:
    - mysql_user: name={{ client }}
                  password="{{ lookup('password', '/tmp/passwordfile chars=ascii,numbers,digits,hexdi
                  priv={{ client }}_{{ tier }}_{{ role }}.*:ALL

    (...)
```

To enter comma use two commas ‘„’ somewhere - preferably at the end. Quotes and double quotes are not supported.

More Lookups

Note: This feature is very infrequently used in Ansible. You may wish to skip this section.

New in version 0.8.

Various *lookup plugins* allow additional ways to iterate over data. In *Loops* you will learn how to use them to walk over collections of numerous types. However, they can also be used to pull in data from remote sources, such as shell commands or even key value stores. This section will cover lookup plugins in this capacity.

Here are some examples:

```
---
- hosts: all

  tasks:

     - debug: msg="{{ lookup('env','HOME') }} is an environment variable"

     - debug: msg="{{ item }} is a line from the result of this command"
       with_lines:
         - cat /etc/motd

     - debug: msg="{{ lookup('pipe','date') }} is the raw result of running this command"

     - debug: msg="{{ lookup('redis_kv', 'redis://localhost:6379,somekey') }} is value in Redis for s

     - debug: msg="{{ lookup('dnstxt', 'example.com') }} is a DNS TXT record for example.com"

     - debug: msg="{{ lookup('template', './some_template.j2') }} is a value from evaluation of this
```

As an alternative you can also assign lookup plugins to variables or use them elsewhere. This macros are evaluated each time they are used in a task (or template):

```
vars:
  motd_value: "{{ lookup('file', '/etc/motd') }}"

tasks:
  - debug: msg="motd value is {{ motd_value }}"
```

See also:

Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Variables All about variables

Loops Looping in playbooks

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.8 Prompts

When running a playbook, you may wish to prompt the user for certain input, and can do so with the 'vars_prompt' section.

A common use for this might be for asking for sensitive data that you do not want to record.

This has uses beyond security, for instance, you may use the same playbook for all software releases and would prompt for a particular release version in a push-script.

Here is a most basic example:

```
---
- hosts: all
  remote_user: root
  vars:
    from: "camelot"
  vars_prompt:
    name: "what is your name?"
    quest: "what is your quest?"
    favcolor: "what is your favorite color?"
```

If you have a variable that changes infrequently, it might make sense to provide a default value that can be overridden. This can be accomplished using the default argument:

```
vars_prompt:
  - name: "release_version"
    prompt: "Product release version"
    default: "1.0"
```

An alternative form of vars_prompt allows for hiding input from the user, and may later support some other options, but otherwise works equivalently:

```
vars_prompt:
  - name: "some_password"
    prompt: "Enter password"
    private: yes
  - name: "release_version"
    prompt: "Product release version"
    private: no
```

If Passlib is installed, vars_prompt can also crypt the entered value so you can use it, for instance, with the user module to define a password:

```
vars_prompt:
  - name: "my_password2"
    prompt: "Enter password2"
    private: yes
    encrypt: "md5_crypt"
    confirm: yes
    salt_size: 7
```

You can use any crypt scheme supported by 'Passlib':

- *des_crypt* - DES Crypt
- *bsdi_crypt* - BSDi Crypt
- *bigcrypt* - BigCrypt
- *crypt16* - Crypt16
- *md5_crypt* - MD5 Crypt

- *bcrypt* - BCrypt
- *sha1_crypt* - SHA-1 Crypt
- *sun_md5_crypt* - Sun MD5 Crypt
- *sha256_crypt* - SHA-256 Crypt
- *sha512_crypt* - SHA-512 Crypt
- *apr_md5_crypt* - Apache's MD5-Crypt variant
- *phpass* - PHPass' Portable Hash
- *pbkdf2_digest* - Generic PBKDF2 Hashes
- *cta_pbkdf2_sha1* - Cryptacular's PBKDF2 hash
- *dlitz_pbkdf2_sha1* - Dwayne Litzenberger's PBKDF2 hash
- *scram* - SCRAM Hash
- *bsd_nthash* - FreeBSD's MCF-compatible nthash encoding

However, the only parameters accepted are 'salt' or 'salt_size'. You can use your own salt using 'salt', or have one generated automatically using 'salt_size'. If nothing is specified, a salt of size 8 will be generated.

See also:

Playbooks An introduction to playbooks

Conditionals Conditional statements in playbooks

Variables All about variables

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.4.9 Tags

If you have a large playbook it may become useful to be able to run a specific part of the configuration without running the whole playbook.

Both plays and tasks support a "tags:" attribute for this reason.

Example:

```
tasks:

    - yum: name={{ item }} state=installed
      with_items:
         - httpd
         - memcached
      tags:
         - packages

    - template: src=templates/src.j2 dest=/etc/foo.conf
      tags:
         - configuration
```

If you wanted to just run the "configuration" and "packages" part of a very long playbook, you could do this:

```
ansible-playbook example.yml --tags "configuration,packages"
```

On the other hand, if you want to run a playbook *without* certain tasks, you could do this:

```
ansible-playbook example.yml --skip-tags "notification"
```

You may also apply tags to roles:

```
roles:
  - { role: webserver, port: 5000, tags: [ 'web', 'foo' ] }
```

And you may also tag basic include statements:

```
- include: foo.yml tags=web,foo
```

Both of these have the function of tagging every single task inside the include statement.

See also:

Playbooks An introduction to playbooks

Playbook Roles and Include Statements Playbook organization by roles

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.5 About Modules

1.5.1 Introduction

Ansible ships with a number of modules (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.

Let's review how we execute three different modules from the command line:

```
ansible webservers -m service -a "name=httpd state=running"
ansible webservers -m ping
ansible webservers -m command -a "/sbin/reboot -t now"
```

Each module supports taking arguments. Nearly all modules take `key=value` arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.

From playbooks, Ansible modules are executed in a very similar way:

```
- name: reboot the servers
  action: command /sbin/reboot -t now
```

Which can be abbreviated to:

```
- name: reboot the servers
  command: /sbin/reboot -t now
```

All modules technically return JSON format data, though if you are using the command line or playbooks, you don't really need to know much about that. If you're writing your own module, you care, and this means you do not have to write modules in any particular language – you get to choose.

Modules are *idempotent*, meaning they will seek to avoid changes to the system unless a change needs to be made. When using Ansible playbooks, these modules can trigger 'change events' in the form of notifying 'handlers' to run additional tasks.

Documentation for each module can be accessed from the command line with the ansible-doc tool:

```
ansible-doc yum
```

See also:

Introduction To Ad-Hoc Commands Examples of using modules in /usr/bin/ansible

Playbooks Examples of using modules with /usr/bin/ansible-playbook

Developing Modules How to write your own modules

Python API Examples of using modules with the Python API

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.6 Module Index

1.6.1 All Modules

accelerate - Enable accelerated mode on remote node

Author James Cammarata

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

This modules launches an ephemeral *accelerate* daemon on the remote node which Ansible can use to communicate with nodes at high speed. The daemon listens on a configurable port for a configurable amount of time. Fireball mode is AES encrypted

Options

Note: Requires python-keyczar

Examples

```
# To use accelerate mode, simply add "accelerate: true" to your play. The initial
# key exchange and starting up of the daemon will occur over SSH, but all commands and
# subsequent actions will be conducted over the raw socket connection using AES encryption

- hosts: devservers
  accelerate: true
```

```
tasks:
    - command: /usr/bin/anything
```

Note: See the advanced playbooks chapter for more about using accelerated mode.

acl - Sets and retrieves file ACL information.

Author Brian Coca

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Sets and retrieves file ACL information.

Options

Examples

```
# Grant user Joe read access to a file
- acl: name=/etc/foo.conf entity=joe etype=user permissions="r" state=present

# Removes the acl for Joe on a specific file
- acl: name=/etc/foo.conf entity=joe etype=user state=absent

# Sets default acl for joe on foo.d
- acl: name=/etc/foo.d entity=joe etype=user permissions=rw default=yes state=present

# Same as previous but using entry shorthand
- acl: name=/etc/foo.d entrty="default:user:joe:rw-" state=present

# Obtain the acl for a specific file
- acl: name=/etc/foo.conf
  register: acl_info
```

Note: The "acl" module requires that acls are enabled on the target filesystem and that the setfacl and getfacl binaries are installed.

add_host - add a host (and alternatively a group) to the ansible-playbook in-memory inventory

Author Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

Use variables to create new hosts and groups in inventory for use in later plays of the same playbook. Takes variables so you can define the new hosts more fully.

Options

Examples

```
# add host to group 'just_created' with variable foo=42
- add_host: name={{ ip_from_ec2 }} groups=just_created foo=42

# add a host with a non-standard port local to your machines
- add_host: name={{ new_ip }}:{{ new_port }}

# add a host alias that we reach through a tunnel
- add_host: hostname={{ new_ip }}
            ansible_ssh_host={{ inventory_hostname }}
            ansible_ssh_port={{ new_port }}
```

airbrake_deployment - Notify airbrake about app deployments

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Notify airbrake about app deployments (see http://help.airbrake.io/kb/api-2/deploy-tracking)

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- airbrake_deployment: token=AAAAAA
                       environment='staging'
                       user='ansible'
                       revision=4.2
```

apt - Manages apt-packages

Author Matthew Williams

- Synopsis
- Options
- Examples

Synopsis

Manages *apt* packages (such as for Debian/Ubuntu).

Options

Note: Requires python-apt

Note: Requires aptitude

Examples

```
# Update repositories cache and install "foo" package
- apt: pkg=foo update_cache=yes

# Remove "foo" package
- apt: pkg=foo state=absent

# Install the package "foo"
- apt: pkg=foo state=present

# Install the version '1.00' of package "foo"
- apt: pkg=foo=1.00 state=present

# Update the repository cache and update package "nginx" to latest version using default release sque
- apt: pkg=nginx state=latest default_release=squeeze-backports update_cache=yes

# Install latest version of "openjdk-6-jdk" ignoring "install-recommends"
- apt: pkg=openjdk-6-jdk state=latest install_recommends=no

# Update all packages to the latest version
- apt: upgrade=dist
```

```
# Run the equivalent of "apt-get update" as a separate step
- apt: update_cache=yes

# Only run "update_cache=yes" if the last one is more than more than 3600 seconds ago
- apt: update_cache=yes cache_valid_time=3600

# Pass options to dpkg on run
- apt: upgrade=dist update_cache=yes dpkg_options='force-confold,force-confdef'
```

Note: Three of the upgrade modes (`full`, `safe` and its alias `yes`) require `aptitude`, otherwise `apt-get` suffices.

apt_key - Add or remove an apt key

Author Jayson Vantuyl & others

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Add or remove an *apt* key, optionally downloading it

Options

Examples

```
# Add an Apt signing key, uses whichever key is at the URL
- apt_key: url=https://ftp-master.debian.org/keys/archive-key-6.0.asc state=present

# Add an Apt signing key, will not download if present
- apt_key: id=473041FA url=https://ftp-master.debian.org/keys/archive-key-6.0.asc state=present

# Remove an Apt signing key, uses whichever key is at the URL
- apt_key: url=https://ftp-master.debian.org/keys/archive-key-6.0.asc state=absent

# Remove a Apt specific signing key, leading 0x is valid
- apt_key: id=0x473041FA state=absent

# Add a key from a file on the Ansible server
- apt_key: data="{{ lookup('file', 'apt.gpg') }}" state=present

# Add an Apt signing key to a specific keyring file
- apt_key: id=473041FA url=https://ftp-master.debian.org/keys/archive-key-6.0.asc keyring=/etc/apt/t
```

Note: doesn't download the key unless it really needs it

Note: as a sanity check, downloaded key id must match the one specified

Note: best practice is to specify the key id and the url

apt_repository - Add and remove APT repositores

Author Alexander Saltanov

- Synopsis
- Options
- Examples

Synopsis

Add or remove an APT repositories in Ubuntu and Debian.

Options

Note: Requires python-apt

Note: Requires python-pycurl

Examples

```
# Add specified repository into sources list.
apt_repository: repo='deb http://archive.canonical.com/ubuntu hardy partner' state=present

# Add source repository into sources list.
apt_repository: repo='deb-src http://archive.canonical.com/ubuntu hardy partner' state=present

# Remove specified repository from sources list.
apt_repository: repo='deb http://archive.canonical.com/ubuntu hardy partner' state=absent

# On Ubuntu target: add nginx stable repository from PPA and install its signing key.
# On Debian target: adding PPA is not available, so it will fail immediately.
apt_repository: repo='ppa:nginx/stable'
```

Note: This module works on Debian and Ubuntu and requires `python-apt` and `python-pycurl` packages.

Note: This module supports Debian Squeeze (version 6) as well as its successors.

Note: This module treats Debian and Ubuntu distributions separately. So PPA could be installed only on Ubuntu machines.

arista_interface - Manage physical Ethernet interfaces

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage physical Ethernet interface resources on Arista EOS network devices

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_interface module to manage resource
state.  Note that interface names must be the full interface name not shortcut
names (ie Ethernet, not Et1)

    tasks:
    - name: enable interface Ethernet 1
      action: arista_interface interface_id=Ethernet1 admin=up speed=10g duplex=full logging=true

    - name: set mtu on Ethernet 1
      action: arista_interface interface_id=Ethernet1 mtu=1600 speed=10g duplex=full logging=true

    - name: reset changes to Ethernet 1
      action: arista_interface interface_id=Ethernet1 admin=down mtu=1500 speed=10g duplex=full loggi
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

arista_l2interface - Manage layer 2 interfaces

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage layer 2 interface resources on Arista EOS network devices

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_l2interface module to manage resource
state. Note that interface names must be the full interface name not shortcut
names (ie Ethernet, not Et1)

    tasks:
    - name: create switchport ethernet1 access port
      action: arista_l2interface interface_id=Ethernet1 logging=true

    - name: create switchport ethernet2 trunk port
      action: arista_l2interface interface_id=Ethernet2 vlan_tagging=enable logging=true

    - name: add vlans to red and blue switchport ethernet2
      action: arista_l2interface interface_id=Ethernet2 tagged_vlans=red,blue logging=true

    - name: set untagged vlan for Ethernet1
      action: arista_l2interface interface_id=Ethernet1 untagged_vlan=red logging=true

    - name: convert access to trunk
      action: arista_l2interface interface_id=Ethernet1 vlan_tagging=enable tagged_vlans=red,blue log

    - name: convert trunk to access
      action: arista_l2interface interface_id=Ethernet2 vlan_tagging=disable untagged_vlan=blue loggi

    - name: delete switchport ethernet1
      action: arista_l2interface interface_id=Ethernet1 state=absent logging=true
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

arista_lag - Manage port channel (lag) interfaces

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage port channel interface resources on Arista EOS network devices

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_lag module to manage resource
state.  Note that interface names must be the full interface name not shortcut
names (ie Ethernet, not Et1)

    tasks:
    - name: create lag interface
      action: arista_lag interface_id=Port-Channel1 links=Ethernet1,Ethernet2 logging=true

    - name: add member links
      action: arista_lag interface_id=Port-Channel1 links=Ethernet1,Ethernet2,Ethernet3 logging=true

    - name: remove member links
      action: arista_lag interface_id=Port-Channel1 links=Ethernet2,Ethernet3 logging=true

    - name: remove lag interface
      action: arista_lag interface_id=Port-Channel1 state=absent logging=true
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

arista_vlan - Manage VLAN resources

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage VLAN resources on Arista EOS network devices. This module requires the Netdev EOS extension to be installed in EOS. For detailed instructions for installing and using the Netdev module please see [link]

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_vlan module to manage resource
state.

  tasks:
  - name: create vlan 999
    action: arista_vlan vlan_id=999 logging=true

  - name: create / edit vlan 999
    action: arista_vlan vlan_id=999 name=test logging=true

  - name: remove vlan 999
    action: arista_vlan vlan_id=999 state=absent logging=true
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

assemble - Assembles a configuration file from fragments

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Assembles a configuration file from fragments. Often a particular program will take a single configuration file and does not support a `conf.d` style structure where it is easy to build up the configuration from multiple sources. `assemble` will take a directory of files that can be local or have already been transferred to the system, and concatenate them together to produce a destination file. Files are assembled in string sorting order. Puppet calls this idea *fragments*.

Options

Examples

```
# Example from Ansible Playbooks
- assemble: src=/etc/someapp/fragments dest=/etc/someapp/someapp.conf

# When a delimiter is specified, it will be inserted in between each fragment
- assemble: src=/etc/someapp/fragments dest=/etc/someapp/someapp.conf delimiter='### START FRAGMENT
```

assert - Fail with custom message

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module asserts that a given expression is true and can be a simpler alternative to the 'fail' module in some cases.

Options

Examples

```
- assert: ansible_os_family != "RedHat"
- assert: "'foo' in some_command_result.stdout"
```

async_status - Obtain status of asynchronous task

Author Michael DeHaan

- Synopsis
- Options

Synopsis

This module gets the status of an asynchronous task.

Options

Note: See also http://docs.ansible.com/playbooks_async.html

at - Schedule the execution of a command or scripts via the at command.

Author Richard Isaacson

- Synopsis
- Options
- Examples

Synopsis

Use this module to schedule a command or script to run once in the future. All jobs are executed in the a queue.

Options

Note: Requires at

Examples

```
# Schedule a command to execute in 20 minutes as root.
- at: command="ls -d / > /dev/null" unit_count=20 unit_type="minutes"

# Schedule a script to execute in 1 hour as the neo user.
- at: script_file="/some/script.sh" user="neo" unit_count=1 unit_type="hours"

# Match a command to an existing job and delete the job.
- at: command="ls -d / > /dev/null" action="delete"

# Schedule a command to execute in 20 minutes making sure it is unique in the queue.
- at: command="ls -d / > /dev/null" action="unique" unit_count=20 unit_type="minutes"
```

authorized_key - Adds or removes an SSH authorized key

Author Brad Olson

- Synopsis
- Options
- Examples

Synopsis

Adds or removes authorized keys for particular user accounts

Options

Examples

```
# Example using key data from a local file on the management machine
- authorized_key: user=charlie key="{{ lookup('file', '/home/charlie/.ssh/id_rsa.pub') }}"

# Using alternate directory locations:
- authorized_key: user=charlie
                  key="{{ lookup('file', '/home/charlie/.ssh/id_rsa.pub') }}"
                  path='/etc/ssh/authorized_keys/charlie'
                  manage_dir=no

# Using with_file
- name: Set up authorized_keys for the deploy user
  authorized_key: user=deploy
                  key="{{ item }}"
  with_file:
    - public_keys/doe-jane
    - public_keys/doe-john

# Using key_options:
- authorized_key: user=charlie
                  key="{{ lookup('file', '/home/charlie/.ssh/id_rsa.pub') }}"
                  key_options='no-port-forwarding,host="10.0.1.1"'
```

bigip_monitor_http - Manages F5 BIG-IP LTM http monitors

Author Serge van Ginderachter

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM monitors via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
- name: BIGIP F5 | Create HTTP Monitor
  local_action:
    module:             bigip_monitor_http
    state:              present
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ item.monitorname }}"
    send:               "{{ item.send }}"
    receive:            "{{ item.receive }}"
  with_items: f5monitors
- name: BIGIP F5 | Remove HTTP Monitor
  local_action:
    module:             bigip_monitor_http
    state:              absent
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ monitorname }}"
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

Note: Monitor API documentation: https://devcentral.f5.com/wiki/iControl.LocalLB__Monitor.ashx

bigip_monitor_tcp - Manages F5 BIG-IP LTM tcp monitors

Author Serge van Ginderachter

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM tcp monitors via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
- name: BIGIP F5 | Create TCP Monitor
  local_action:
    module:             bigip_monitor_tcp
    state:              present
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ item.monitorname }}"
    type:               tcp
    send:               "{{ item.send }}"
    receive:            "{{ item.receive }}"
  with_items: f5monitors-tcp
- name: BIGIP F5 | Create TCP half open Monitor
  local_action:
    module:             bigip_monitor_tcp
    state:              present
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ item.monitorname }}"
    type:               tcp
    send:               "{{ item.send }}"
    receive:            "{{ item.receive }}"
  with_items: f5monitors-halftcp
- name: BIGIP F5 | Remove TCP Monitor
  local_action:
    module:             bigip_monitor_tcp
    state:              absent
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ monitorname }}"
  with_flattened:
  - f5monitors-tcp
  - f5monitors-halftcp
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

Note: Monitor API documentation: https://devcentral.f5.com/wiki/iControl.LocalLB__Monitor.ashx

bigip_node - Manages F5 BIG-IP LTM nodes

Author Matt Hite

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM nodes via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
## playbook task examples:

---
# file bigip-test.yml
# ...
- hosts: bigip-test
  tasks:
  - name: Add node
    local_action: >
      bigip_node
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      name="{{ ansible_default_ipv4["address"] }}"

# Note that the BIG-IP automatically names the node using the
# IP address specified in previous play's host parameter.
# Future plays referencing this node no longer use the host
# parameter but instead use the name parameter.
# Alternatively, you could have specified a name with the
# name parameter when state=present.

  - name: Modify node description
    local_action: >
      bigip_node
      server=lb.mydomain.com
      user=admin
      password=mysecret
```

```
      state=present
      partition=matthite
      name="{{ ansible_default_ipv4["address"] }}"
      description="Our best server yet"

  - name: Delete node
    local_action: >
      bigip_node
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=absent
      partition=matthite
      name="{{ ansible_default_ipv4["address"] }}"
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

bigip_pool - Manages F5 BIG-IP LTM pools

Author Matt Hite

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manages F5 BIG-IP LTM pools via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
## playbook task examples:

---
# file bigip-test.yml
# ...
- hosts: localhost
```

```
  tasks:
  - name: Create pool
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      name=matthite-pool
      partition=matthite
      lb_method=least_connection_member
      slow_ramp_time=120

  - name: Modify load balancer method
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      name=matthite-pool
      partition=matthite
      lb_method=round_robin

- hosts: bigip-test
  tasks:
  - name: Add pool member
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      name=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80

  - name: Remove pool member from pool
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=absent
      name=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80

- hosts: localhost
  tasks:
  - name: Delete pool
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
```

```
      state=absent
      name=matthite-pool
      partition=matthite
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

bigip_pool_member - Manages F5 BIG-IP LTM pool members

Author Matt Hite

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM pool members via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
## playbook task examples:

---
# file bigip-test.yml
# ...
- hosts: bigip-test
  tasks:
  - name: Add pool member
    local_action: >
      bigip_pool_member
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      pool=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
```

```
      port=80
      description="web server"
      connection_limit=100
      rate_limit=50
      ratio=2

  - name: Modify pool member ratio and description
    local_action: >
      bigip_pool_member
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      pool=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80
      ratio=1
      description="nginx server"

  - name: Remove pool member from pool
    local_action: >
      bigip_pool_member
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=absent
      pool=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

Note: Supersedes bigip_pool for managing pool members

boundary_meter - Manage boundary meters

Author curtis@serverascode.com

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

This module manages boundary meters

Options

Note: Requires Boundary API access

Note: Requires bprobe is required to send data, but not to register a meter

Note: Requires Python urllib2

Examples

```
- name: Create meter
  boundary_meter: apiid=AAAAAA api_key=BBBBBB state=present name={{ inventory_hostname }}"

- name: Delete meter
  boundary_meter: apiid=AAAAAA api_key=BBBBBB state=absent name={{ inventory_hostname }}"
```

Note: This module does not yet support boundary tags.

bzr - Deploy software (or files) from bzr branches

Author André Paramés

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage *bzr* branches to deploy files or software.

Options

Examples

```
# Example bzr checkout from Ansible Playbooks
- bzr: name=bzr+ssh://foosball.example.org/path/to/branch dest=/srv/checkout version=22
```

campfire - Send a message to Campfire

Author Adam Garside <adam.garside@gmail.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to Campfire. Messages with newlines will result in a "Paste" message being sent.

Options

Note: Requires urllib2

Note: Requires cgi

Examples

```
- campfire: subscription=foo token=12345 room=123 msg="Task completed."

- campfire: subscription=foo token=12345 room=123 notify=loggins
        msg="Task completed ... with feeling."
```

cloudformation - create a AWS CloudFormation stack

Author James S. Martin

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Launches an AWS CloudFormation stack and waits for it complete.

Options

Note: Requires boto

Examples

```
# Basic task example
tasks:
- name: launch ansible cloudformation example
  action: cloudformation >
    stack_name="ansible-cloudformation" state=present
    region=us-east-1 disable_rollback=yes
    template=files/cloudformation-example.json
  args:
    template_parameters:
      KeyName: jmartin
      DiskType: ephemeral
      InstanceType: m1.small
      ClusterSize: 3
    tags:
      Stack: ansible-cloudformation
```

command - Executes a command on a remote node

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `command` module takes the command name followed by a list of space-delimited arguments. The given command will be executed on all selected nodes. It will not be processed through the shell, so variables like `$HOME` and operations like `"<"`, `">"`, `"|"`, and `"&"` will not work (use the `shell` module if you need these features).

Options

Examples

```
# Example from Ansible Playbooks
- command: /sbin/shutdown -t now

# Run the command if the specified file does not exist
- command: /usr/bin/make_database.sh arg1 arg2 creates=/path/to/database
```

Note: If you want to run a command through the shell (say you are using <, >, |, etc), you actually want the `shell` module instead. The `command` module is much more secure as it's not affected by the user's environment.

Note: `creates`, `removes`, and `chdir` can be specified after the command. For instance, if you only want to run a command if a certain file does not exist, use this.

---

copy - Copies files to remote locations.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `copy` module copies a file on the local box to remote locations.

Options

Examples

```
# Example from Ansible Playbooks
- copy: src=/srv/myfiles/foo.conf dest=/etc/foo.conf owner=foo group=foo mode=0644

# Copy a new "ntp.conf file into place, backing up the original if it differs from the copied versio
- copy: src=/mine/ntp.conf dest=/etc/ntp.conf owner=root group=root mode=644 backup=yes

# Copy a new "sudoers" file into place, after passing validation with visudo
- copy: src=/mine/sudoers dest=/etc/sudoers validate='visudo -cf %s'
```

Note: The "copy" module recursively copy facility does not scale to lots (>hundreds) of files. For alternative, see synchronize module, which is a wrapper around rsync.

cron - Manage cron.d and crontab entries.

Author Dane Summers

- Synopsis
- Options
- Examples

Synopsis

Use this module to manage crontab entries. This module allows you to create named crontab entries, update, or delete them. The module includes one line with the description of the crontab entry `"#Ansible: <name>"` corresponding to the "name" passed to the module, which is used by future ansible/module calls to find/check the state.

Options

Note: Requires cron

Examples

```
# Ensure a job that runs at 2 and 5 exists.
# Creates an entry like "* 5,2 * * ls -alh > /dev/null"
- cron: name="check dirs" hour="5,2" job="ls -alh > /dev/null"

# Ensure an old job is no longer present. Removes any job that is prefixed
# by "#Ansible: an old job" from the crontab
- cron: name="an old job" state=absent

# Creates an entry like "@reboot /some/job.sh"
- cron: name="a job for reboot" special_time=reboot job="/some/job.sh"

# Creates a cron file under /etc/cron.d
- cron: name="yum autoupdate" weekday="2" minute=0 hour=12
        user="root" job="YUMINTERACTIVE=0 /usr/sbin/yum-autoupdate"
        cron_file=ansible_yum-autoupdate

# Removes a cron file from under /etc/cron.d
- cron: cron_file=ansible_yum-autoupdate state=absent
```

datadog_event - Posts events to DataDog service

Author Artras 'arturaz' Šlajus <x11@arturaz.net>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Allows to post events to DataDog (www.datadoghq.com) service. Uses http://docs.datadoghq.com/api/#events API.

Options

Note: Requires urllib2

Examples

```
# Post an event with low priority
datadog_event: title="Testing from ansible" text="Test!" priority="low"
               api_key="6873258723457823548234234234"
# Post an event with several tags
datadog_event: title="Testing from ansible" text="Test!"
```

```
api_key="6873258723457823548234234234"
tags=aa,bb,cc
```

debug - Print statements during execution

Author Dag Wieers, Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This module prints statements during execution and can be useful for debugging variables or expressions without necessarily halting the playbook. Useful for debugging together with the 'when:' directive.

Options

Examples

```
# Example that prints the loopback address and gateway for each host
- debug: msg="System {{ inventory_hostname }} has uuid {{ ansible_product_uuid }}"

- debug: msg="System {{ inventory_hostname }} has gateway {{ ansible_default_ipv4.gateway }}"
  when: ansible_default_ipv4.gateway is defined

- shell: /usr/bin/uptime
  register: result
- debug: var=result
```

digital_ocean - Create/delete a droplet/SSH_key in DigitalOcean

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Create/delete a droplet in DigitalOcean and optionally waits for it to be 'running', or deploy an SSH key.

Options

Note: Requires dopy

Examples

```
# Ensure a SSH key is present
# If a key matches this name, will return the ssh key id and changed = False
# If no existing key matches this name, a new key is created, the ssh key id is returned and changed

- digital_ocean: >
      state=present
      command=ssh
      name=my_ssh_key
      ssh_pub_key='ssh-rsa AAAA...'
      client_id=XXX
      api_key=XXX

# Create a new Droplet
# Will return the droplet details including the droplet id (used for idempotence)

- digital_ocean: >
      state=present
      command=droplet
      name=mydroplet
      client_id=XXX
      api_key=XXX
      size_id=1
      region_id=2
      image_id=3
      wait_timeout=500
  register: my_droplet
- debug: msg="ID is {{ my_droplet.droplet.id }}"
- debug: msg="IP is {{ my_droplet.droplet.ip_address }}"

# Ensure a droplet is present
# If droplet id already exist, will return the droplet details and changed = False
# If no droplet matches the id, a new droplet will be created and the droplet details (including the

- digital_ocean: >
      state=present
      command=droplet
      id=123
      name=mydroplet
      client_id=XXX
      api_key=XXX
      size_id=1
      region_id=2
      image_id=3
      wait_timeout=500

# Create a droplet with ssh key
# The ssh key id can be passed as argument at the creation of a droplet (see ssh_key_ids).
# Several keys can be added to ssh_key_ids as id1,id2,id3
# The keys are used to connect as root to the droplet.
```

```
- digital_ocean: >
      state=present
      ssh_key_ids=id1,id2
      name=mydroplet
      client_id=XXX
      api_key=XXX
      size_id=1
      region_id=2
      image_id=3
```

Note: Two environment variables can be used, DO_CLIENT_ID and DO_API_KEY.

django_manage - Manages a Django application.

Author Scott Anderson

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages a Django application using the *manage.py* application frontend to *django-admin*. With the *virtualenv* parameter, all management commands will be executed by the given *virtualenv* installation.

Options

Note: Requires virtualenv

Note: Requires django

Examples

```
# Run cleanup on the application installed in 'django_dir'.
- django_manage: command=cleanup app_path={{ django_dir }}

# Load the initial_data fixture into the application
- django_manage: command=loaddata app_path={{ django_dir }} fixtures={{ initial_data }}

#Run syncdb on the application
- django_manage: >
      command=syncdb
      app_path={{ django_dir }}
      settings={{ settings_app_name }}
      pythonpath={{ settings_dir }}
```

```
      virtualenv={{ virtualenv_dir }}

#Run the SmokeTest test case from the main app. Useful for testing deploys.
- django_manage: command=test app_path=django_dir apps=main.SmokeTest
```

Note: *virtualenv* (http://www.virtualenv.org) must be installed on the remote host if the virtualenv parameter is specified.

Note: This module will create a virtualenv if the virtualenv parameter is specified and a virtualenv does not already exist at the given location.

Note: This module assumes English error messages for the 'createcachetable' command to detect table existence, unfortunately.

Note: To be able to use the migrate command, you must have south installed and added as an app in your settings

Note: To be able to use the collectstatic command, you must have enabled staticfiles in your settings

dnsmadeeasy - Interface with dnsmadeeasy.com (a DNS hosting service).

Author Brice Burgess

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages DNS records via the v2 REST API of the DNS Made Easy service. It handles records only; there is no manipulation of domains or monitor/account support yet. See: http://www.dnsmadeeasy.com/services/rest-api/

Options

Note: Requires urllib

Note: Requires urllib2

Note: Requires hashlib

Note: Requires hmac

Examples

```
# fetch my.com domain records
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present
  register: response

# create / ensure the presence of a record
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present record_name="test" r

# update the previously created record
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present record_name="test" r

# fetch a specific record
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present record_name="test"
  register: response

# delete a record / ensure it is absent
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=absent record_name="test"
```

Note: The DNS Made Easy service requires that machines interacting with the API have the proper time and timezone set. Be sure you are within a few seconds of actual time by using NTP.

Note: This module returns record(s) in the "result" element when 'state' is set to 'present'. This value can be be registered and used in your playbooks.

docker - manage docker containers

Author Cove Schneider, Pavel Antonov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage the life cycle of docker containers.

Options

Note: Requires docker-py

Examples

```
Start one docker container running tomcat in each host of the web group and bind tomcat's listening p
on the host:
```

```
- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker: image=centos command="service tomcat6 start" ports=:8080

The tomcat server's port is NAT'ed to a dynamic port on the host, but you can determine which port th
mapped to using docker_containers:

- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker: image=centos command="service tomcat6 start" ports=8080 count=5
  - name: Display IP address and port mappings for containers
    debug: msg={{inventory_hostname}}:{{item.NetworkSettings.Ports['8080/tcp'][0].HostPort}}
    with_items: docker_containers

Just as in the previous example, but iterates over the list of docker containers with a sequence:

- hosts: web
  sudo: yes
  vars:
    start_containers_count: 5
  tasks:
  - name: run tomcat servers
    docker: image=centos command="service tomcat6 start" ports=8080 count={{start_containers_count}}
  - name: Display IP address and port mappings for containers
    debug: msg={{inventory_hostname}}:{{docker_containers[{{item}}].NetworkSettings.Ports['8080/tcp']
    with_sequence: start=0 end={{start_containers_count - 1}}

Stop, remove all of the running tomcat containers and list the exit code from the stopped containers

- hosts: web
  sudo: yes
  tasks:
  - name: stop tomcat servers
    docker: image=centos command="service tomcat6 start" state=absent
  - name: Display return codes from stopped containers
    debug: msg="Returned {{inventory_hostname}}:{{item}}"
    with_items: docker_containers
```

docker_image - manage docker images

Author Pavel Antonov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Create, check and remove docker images

Options

Note: Requires docker-py

Examples

```
Build docker image if required. Path should contains Dockerfile to build image:

- hosts: web
  sudo: yes
  tasks:
  - name: check or build image
    docker_image: path="/path/to/build/dir" name="my/app" state=present

Build new version of image:

- hosts: web
  sudo: yes
  tasks:
  - name: check or build image
    docker_image: path="/path/to/build/dir" name="my/app" state=build

Remove image from local docker storage:

- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker_image: name="my/app" state=absent
```

easy_install - Installs Python libraries

Author Matt Wright

- Synopsis
- Options
- Examples

Synopsis

Installs Python libraries, optionally in a *virtualenv*

Options

Note: Requires virtualenv

Examples

```
# Examples from Ansible Playbooks
- easy_install: name=pip

# Install Bottle into the specified virtualenv.
- easy_install: name=bottle virtualenv=/webapps/myapp/venv
```

Note: Please note that the `easy_install` module can only install Python libraries. Thus this module is not able to remove libraries. It is generally recommended to use the `pip` module which you can first install using `easy_install`.

Note: Also note that *virtualenv* must be installed on the remote host if the `virtualenv` parameter is specified.

ec2 - create, terminate, start or stop an instance in ec2, return instanceid

Author Seth Vidal, Tim Gerla, Lester Wade

- Synopsis
- Options
- Examples

Synopsis

Creates or terminates ec2 instances. When created optionally waits for it to be 'running'. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic provisioning example
- local_action:
    module: ec2
    keypair: mykey
    instance_type: c1.medium
    image: emi-40603AD1
    wait: yes
    group: webserver
    count: 3

# Advanced example with tagging and CloudWatch
```

```
- local_action:
    module: ec2
    keypair: mykey
    group: databases
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    count: 5
    instance_tags:
       db: postgres
    monitoring: yes

# Single instance with additional IOPS volume from snapshot
local_action:
    module: ec2
    keypair: mykey
    group: webserver
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    volumes:
    - device_name: /dev/sdb
      snapshot: snap-abcdef12
      device_type: io1
      iops: 1000
      volume_size: 100
    monitoring: yes

# Multiple groups example
local_action:
    module: ec2
    keypair: mykey
    group: ['databases', 'internal-services', 'sshable', 'and-so-forth']
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    count: 5
    instance_tags:
        db: postgres
    monitoring: yes

# Multiple instances with additional volume from snapshot
local_action:
    module: ec2
    keypair: mykey
    group: webserver
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    count: 5
    volumes:
    - device_name: /dev/sdb
      snapshot: snap-abcdef12
      volume_size: 10
```

```
    monitoring: yes

# VPC example
- local_action:
    module: ec2
    keypair: mykey
    group_id: sg-1dc53f72
    instance_type: m1.small
    image: ami-6e649707
    wait: yes
    vpc_subnet_id: subnet-29e63245
    assign_public_ip: yes

# Launch instances, runs some tasks
# and then terminate them


- name: Create a sandbox instance
  hosts: localhost
  gather_facts: False
  vars:
    keypair: my_keypair
    instance_type: m1.small
    security_group: my_securitygroup
    image: my_ami_id
    region: us-east-1
  tasks:
    - name: Launch instance
      local_action: ec2 keypair={{ keypair }} group={{ security_group }} instance_type={{ instance_ty
      register: ec2
    - name: Add new instance to host group
      local_action: add_host hostname={{ item.public_ip }} groupname=launched
      with_items: ec2.instances
    - name: Wait for SSH to come up
      local_action: wait_for host={{ item.public_dns_name }} port=22 delay=60 timeout=320 state=start
      with_items: ec2.instances

- name: Configure instance(s)
  hosts: launched
  sudo: True
  gather_facts: True
  roles:
    - my_awesome_role
    - my_awesome_test

- name: Terminate instances
  hosts: localhost
  connection: local
  tasks:
    - name: Terminate instances that were previously launched
      local_action:
        module: ec2
        state: 'absent'
        instance_ids: '{{ ec2.instance_ids }}'

# Start a few existing instances, run some tasks
# and stop the instances
```

```
- name: Start sandbox instances
  hosts: localhost
  gather_facts: false
  connection: local
  vars:
    instance_ids:
      - 'i-xxxxxx'
      - 'i-xxxxxx'
      - 'i-xxxxxx'
    region: us-east-1
  tasks:
    - name: Start the sandbox instances
      local_action:
        module: ec2
        instance_ids: '{{ instance_ids }}'
        region: '{{ region }}'
        state: running
        wait: True
  role:
    - do_neat_stuff
    - do_more_neat_stuff

- name: Stop sandbox instances
  hosts: localhost
  gather_facts: false
  connection: local
  vars:
    instance_ids:
      - 'i-xxxxxx'
      - 'i-xxxxxx'
      - 'i-xxxxxx'
    region: us-east-1
  tasks:
    - name: Stop the sanbox instances
      local_action:
      module: ec2
      instance_ids: '{{ instance_ids }}'
      region: '{{ region }}'
      state: stopped
      wait: True

#
# Enforce that 5 instances with a tag "foo" are running
#

- local_action:
    module: ec2
    keypair: mykey
    instance_type: c1.medium
    image: emi-40603AD1
    wait: yes
    group: webserver
    instance_tags:
        foo: bar
    exact_count: 5
    count_tag: foo

#
```

```
# Enforce that 5 running instances named "database" with a "dbtype" of "postgres"
#

- local_action:
    module: ec2
    keypair: mykey
    instance_type: c1.medium
    image: emi-40603AD1
    wait: yes
    group: webserver
    instance_tags:
        Name: database
        dbtype: postgres
    exact_count: 5
    count_tag:
        Name: database
        dbtype: postgres

#
# count_tag complex argument examples
#

    # instances with tag foo
    count_tag:
        foo:

    # instances with tag foo=bar
    count_tag:
        foo: bar

    # instances with tags foo=bar & baz
    count_tag:
        foo: bar
        baz:

    # instances with tags foo & bar & baz=bang
    count_tag:
        - foo
        - bar
        - baz: bang
```

ec2_ami - create or destroy an image in ec2, return imageid

Author Evan Duffield <eduffield@iacquire.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates or deletes ec2 images. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
# Basic AMI Creation
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    instance_id: i-xxxxxx
    wait: yes
    name: newtest
  register: instance

# Basic AMI Creation, without waiting
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    region: xxxxxx
    instance_id: i-xxxxxx
    wait: no
    name: newtest
  register: instance

# Deregister/Delete AMI
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    region: xxxxxx
    image_id: ${instance.image_id}
    delete_snapshot: True
    state: absent

# Deregister AMI
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    region: xxxxxx
    image_id: ${instance.image_id}
    delete_snapshot: False
    state: absent
```

ec2_eip - associate an EC2 elastic IP with an instance.

Author Lorin Hochstein <lorin@nimbisservices.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module associates AWS EC2 elastic IP addresses with instances

Options

Note: Requires boto

Examples

```
- name: associate an elastic IP with an instance
  ec2_eip: instance_id=i-1212f003 ip=93.184.216.119

- name: disassociate an elastic IP from an instance
  ec2_eip: instance_id=i-1212f003 ip=93.184.216.119 state=absent

- name: allocate a new elastic IP and associate it with an instance
  ec2_eip: instance_id=i-1212f003

- name: allocate a new elastic IP without associating it to anything
  ec2_eip:
  register: eip
- name: output the IP
  debug: msg="Allocated IP is {{ eip.public_ip }}"

- name: provision new instances with ec2
  ec2: keypair=mykey instance_type=c1.medium image=emi-40603AD1 wait=yes group=webserver count=3
  register: ec2
- name: associate new elastic IPs with each of the instances
  ec2_eip: "instance_id={{ item }}"
  with_items: ec2.instance_ids

- name: allocate a new elastic IP inside a VPC in us-west-2
  ec2_eip: region=us-west-2 in_vpc=yes
  register: eip
- name: output the IP
  debug: msg="Allocated IP inside a VPC is {{ eip.public_ip }}"
```

Note: This module will return `public_ip` on success, which will contain the public IP address associated with the instance.

Note: There may be a delay between the time the Elastic IP is assigned and when the cloud instance is reachable via the new address. Use wait_for and pause to delay further playbook execution until the instance is reachable, if necessary.

ec2_elb - De-registers or registers instances from EC2 ELBs

Author John Jarvis

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module de-registers or registers an AWS EC2 instance from the ELBs that it belongs to. Returns fact "ec2_elbs" which is a list of elbs attached to the instance if state=absent is passed as an argument. Will be marked changed when called only if there are ELBs found to operate on.

Options

Note: Requires boto

Examples

```
# basic pre_task and post_task example
pre_tasks:
  - name: Gathering ec2 facts
    ec2_facts:
  - name: Instance De-register
    local_action: ec2_elb
    args:
      instance_id: "{{ ansible_ec2_instance_id }}"
      state: 'absent'
roles:
  - myrole
post_tasks:
  - name: Instance Register
    local_action: ec2_elb
    args:
      instance_id: "{{ ansible_ec2_instance_id }}"
      ec2_elbs: "{{ item }}"
      state: 'present'
    with_items: ec2_elbs
```

ec2_elb_lb - Creates or destroys Amazon ELB. - Returns information about the load balancer. - Will be marked changed when called only if state is changed.

Author Jim Dalton

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

C r e a t e s

o r

d e s t r o y s

A m a z o n

E L B .

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic provisioning example
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1a
      - us-east-1d
    listeners:
      - protocol: http # options are http, https, ssl, tcp
        load_balancer_port: 80
        instance_port: 80
      - protocol: https
        load_balancer_port: 443
        instance_protocol: http # optional, defaults to value of protocol setting
        instance_port: 80
        # ssl certificate required for https or ssl
        ssl_certificate_id: "arn:aws:iam::123456789012:server-certificate/company/servercerts/ProdSe

# Configure a health check
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1d
```

```
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
    health_check:
        ping_protocol: http # options are http, https, ssl, tcp
        ping_port: 80
        ping_path: "/index.html" # not required for tcp or ssl
        response_timeout: 5 # seconds
        interval: 30 # seconds
        unhealthy_threshold: 2
        healthy_threshold: 10

# Ensure ELB is gone
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: absent

# Normally, this module will purge any listeners that exist on the ELB
# but aren't specified in the listeners parameter. If purge_listeners is
# false it leaves them alone
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1a
      - us-east-1d
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
    purge_listeners: no

# Normally, this module will leave availability zones that are enabled
# on the ELB alone. If purge_zones is true, then any extreneous zones
# will be removed
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1a
      - us-east-1d
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
    purge_zones: yes
```

ec2_facts - Gathers facts about remote hosts within ec2 (aws)

Author Silviu Dicu <silviudicu@gmail.com>

- Synopsis
- Examples

Synopsis

New in version 1.0.

This module fetches data from the metadata servers in ec2 (aws). Eucalyptus cloud provides a similar service and this module should work this cloud provider as well.

Examples

```
# Conditional example
- name: Gather facts
  action: ec2_facts

- name: Conditional
  action: debug msg="This instance is a t1.micro"
  when: ansible_ec2_instance_type == "t1.micro"
```

Note: Parameters to filter on ec2_facts may be added later.

ec2_group - maintain an ec2 VPC security group.

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

maintains ec2 security groups. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
- name: example ec2 group
  local_action:
    module: ec2_group
    name: example
    description: an example EC2 group
```

```
    vpc_id: 12345
    region: eu-west-1a
    ec2_secret_key: SECRET
    ec2_access_key: ACCESS
    rules:
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: 10.0.0.0/8
      - proto: udp
        from_port: 10050
        to_port: 10050
        cidr_ip: 10.0.0.0/8
      - proto: udp
        from_port: 10051
        to_port: 10051
        group_id: sg-12345678
      - proto: all
        # the containing group name may be specified here
        group_name: example
```

ec2_key - maintain an ec2 key pair.

Author Vincent Viallet

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

maintains ec2 key pairs. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Creates a new ec2 key pair named `example` if not present, returns generated
# private key
```

```
- name: example ec2 key
  local_action:
    module: ec2_key
    name: example

# Creates a new ec2 key pair named `example` if not present using provided key
# material
- name: example2 ec2 key
  local_action:
    module: ec2_key
    name: example2
    key_material: 'ssh-rsa AAAAxyz...== me@example.com'
    state: present

# Creates a new ec2 key pair named `example` if not present using provided key
# material
- name: example3 ec2 key
  local_action:
    module: ec2_key
    name: example3
    key_material: "{{ item }}"
  with_file: /path/to/public_key.id_rsa.pub

# Removes ec2 key pair by name
- name: remove example key
  local_action:
    module: ec2_key
    name: example
    state: absent
```

ec2_tag - create and remove tag(s) to ec2 resources.

Author Lester Wade

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates and removes tags from any EC2 resource. The resource is referenced by its resource id (e.g. an instance being i-XXXXXXX). It is designed to be used with complex args (tags), see the examples. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Basic example of adding tag(s)
tasks:
- name: tag a resource
  local_action: ec2_tag resource=vol-XXXXXX region=eu-west-1 state=present
  args:
    tags:
      Name: ubervol
      env: prod

# Playbook example of adding tag(s) to spawned instances
tasks:
- name: launch some instances
  local_action: ec2 keypair={{ keypair }} group={{ security_group }} instance_type={{ instance_type }
  register: ec2

- name: tag my launched instances
  local_action: ec2_tag resource={{ item.id }} region=eu-west-1 state=present
  with_items: ec2.instances
  args:
    tags:
      Name: webserver
      env: prod
```

ec2_vol - create and attach a volume, return volume id and device map

Author Lester Wade

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

creates an EBS volume and optionally attaches it to an instance. If both an instance ID and a device name is given and the instance has a device at the device name, then no volume is created and no attachment is made. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Simple attachment action
- local_action:
    module: ec2_vol
    instance: XXXXXX
    volume_size: 5
    device_name: sdd

# Example using custom iops params
- local_action:
    module: ec2_vol
    instance: XXXXXX
    volume_size: 5
    iops: 200
    device_name: sdd

# Example using snapshot id
- local_action:
    module: ec2_vol
    instance: XXXXXX
    snapshot: "{{ snapshot }}"

# Playbook example combined with instance launch
- local_action:
    module: ec2
    keypair: "{{ keypair }}"
    image: "{{ image }}"
    wait: yes
    count: 3
    register: ec2
- local_action:
    module: ec2_vol
    instance: "{{ item.id }} "
    volume_size: 5
    with_items: ec2.instances
    register: ec2_vol
```

ec2_vpc - configure AWS virtual private clouds

Author Carson Gee

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Create or terminates AWS virtual private clouds. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic creation example:
      local_action:
        module: ec2_vpc
        state: present
        cidr_block: 172.23.0.0/16
        region: us-west-2
# Full creation example with subnets and optional availability zones.
# The absence or presense of subnets deletes or creates them respectively.
      local_action:
        module: ec2_vpc
        state: present
        cidr_block: 172.22.0.0/16
        subnets:
          - cidr: 172.22.1.0/24
            az: us-west-2c
          - cidr: 172.22.2.0/24
            az: us-west-2b
          - cidr: 172.22.3.0/24
            az: us-west-2a
        internet_gateway: True
        route_tables:
          - subnets:
              - 172.22.2.0/24
              - 172.22.3.0/24
            routes:
              - dest: 0.0.0.0/0
                gw: igw
          - subnets:
              - 172.22.1.0/24
            routes:
              - dest: 0.0.0.0/0
                gw: igw
        region: us-west-2
      register: vpc

# Removal of a VPC by id
      local_action:
        module: ec2_vpc
        state: absent
        vpc_id: vpc-aaaaaaa
        region: us-west-2
If you have added elements not managed by this module, e.g. instances, NATs, etc then
the delete will fail until those dependencies are removed.
```

ejabberd_user - Manages users for ejabberd servers

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module provides user management for ejabberd servers

Options

Note: Requires ejabberd

Examples

```
Example playbook entries using the ejabberd_user module to manage users state.

    tasks:

    - name: create a user if it does not exists
      action: ejabberd_user username=test host=server password=password

    - name: delete a user if it exists
      action: ejabberd_user username=test host=server state=absent
```

Note: Password parameter is required for state == present only

Note: Passwords must be stored in clear text for this release

elasticache - Manage cache clusters in Amazon Elasticache.

Author Jim Dalton

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage cache clusters in Amazon Elasticache. Returns information about the specified cache cluster.

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic example
- local_action:
    module: elasticache
    name: "test-please-delete"
    state: present
    engine: memcached
    cache_engine_version: 1.4.14
    node_type: cache.m1.small
    num_nodes: 1
    cache_port: 11211
    cache_security_groups:
      - default
    zone: us-east-1d


# Ensure cache cluster is gone
- local_action:
    module: elasticache
    name: "test-please-delete"
    state: absent

# Reboot cache cluster
- local_action:
    module: elasticache
    name: "test-please-delete"
    state: rebooted
```

facter - Runs the discovery program *facter* on the remote system

Author Michael DeHaan

- Synopsis
- Examples

Synopsis

Runs the *facter* discovery program (https://github.com/puppetlabs/facter) on the remote system, returning JSON data that can be useful for inventory purposes.

Note: Requires facter

Note: Requires ruby-json

Examples

```
# Example command-line invocation
ansible www.example.net -m facter
```

fail - Fail with custom message

Author Dag Wieers

- Synopsis
- Options
- Examples

Synopsis

This module fails the progress with a custom message. It can be useful for bailing out when a certain condition is met using `when`.

Options

Examples

```
# Example playbook using fail and when together
- fail: msg="The system may not be provisioned according to the CMDB status."
  when: cmdb_status != "to-be-staged"
```

fetch - Fetches a file from remote nodes

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This module works like `copy`, but in reverse. It is used for fetching files from remote machines and storing them locally in a file tree, organized by hostname. Note that this module is written to transfer log files that might not be present, so a missing remote file won't be an error unless fail_on_missing is set to 'yes'.

Options

Examples

```
# Store file into /tmp/fetched/host.example.com/tmp/somefile
- fetch: src=/tmp/somefile dest=/tmp/fetched

# Specifying a path directly
- fetch: src=/tmp/somefile dest=/tmp/prefix-{{ ansible_hostname }} flat=yes

# Specifying a destination path
- fetch: src=/tmp/uniquefile dest=/tmp/special/ flat=yes

# Storing in a path relative to the playbook
- fetch: src=/tmp/uniquefile dest=special/prefix-{{ ansible_hostname }} flat=yes
```

file - Sets attributes of files

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

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 `assemble`.

Options

Examples

```
- 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
```

Note: See also `copy`, `template`, `assemble`

filesystem - Makes file system on block device

Author Alexander Bulimov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module creates file system.

Options

Examples

```
# Create a ext2 filesystem on /dev/sdb1.
- filesystem: fstype=ext2 dev=/dev/sdb1

# Create a ext4 filesystem on /dev/sdb1 and check disk blocks.
- filesystem: fstype=ext4 dev=/dev/sdb1 opts="-cc"
```

Note: uses mkfs command

fireball - Enable fireball mode on remote node

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This modules launches an ephemeral *fireball* ZeroMQ message bus daemon on the remote node which Ansible can use to communicate with nodes at high speed. The daemon listens on a configurable port for a configurable amount of time. Starting a new fireball as a given user terminates any existing user fireballs. Fireball mode is AES encrypted

Options

Note: Requires zmq

Note: Requires keyczar

Examples

```
# This example playbook has two plays: the first launches 'fireball' mode on all hosts via SSH, and
# the second actually starts using it for subsequent management over the fireball connection

- hosts: devservers
  gather_facts: false
  connection: ssh
```

```
  sudo: yes
  tasks:
      - action: fireball

- hosts: devservers
  connection: fireball
  tasks:
      - command: /usr/bin/anything
```

Note: See the advanced playbooks chapter for more about using fireball mode.

firewalld - Manage arbitrary ports/services with firewalld

Author Adam Miller <maxamillion@fedoraproject.org>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module allows for addition or deletion of services and ports either tcp or udp in either running or permanent firewalld rules

Options

Note: Requires firewalld >= 0.2.11

Examples

```
- firewalld: service=https permanent=true state=enabled
- firewalld: port=8081/tcp permanent=true state=disabled
- firewalld: zone=dmz service=http permanent=true state=enabled
- firewalld: rich_rule='rule service name="ftp" audit limit value="1/m" accept' permanent=true state=
```

Note: Not tested on any debian based system

flowdock - Send a message to a flowdock

Author Matt Coddington

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to a flowdock team inbox or chat using the push API (see https://www.flowdock.com/api/team-inbox and https://www.flowdock.com/api/chat)

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- flowdock: type=inbox
            token=AAAAAA
            from_address=user@example.com
            source='my cool app'
            msg='test from ansible'
            subject='test subject'

- flowdock: type=chat
            token=AAAAAA
            external_user_name=testuser
            msg='test from ansible'
            tags=tag1,tag2,tag3
```

gc_storage - This module manages objects/buckets in Google Cloud Storage.

Author benno@ansible.com Note. Most of the code has been taken from the S3 module.

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module allows users to manage their objects/buckets in Google Cloud Storage. It allows upload and download operations and can set some canned permissions. It also allows retrieval of URLs for objects for use in playbooks,

and retrieval of string contents of objects. This module requires setting the default project in GCS prior to playbook usage. See https://developers.google.com/storage/docs/reference/v1/apiversion1 for information about setting the default project.

Options

Note: Requires boto 2.9+

Examples

```
# upload some content
- gc_storage: bucket=mybucket object=key.txt src=/usr/local/myfile.txt mode=put permission=public-rea

# download some content
- gc_storage: bucket=mybucket object=key.txt dest=/usr/local/myfile.txt mode=get

# Download an object as a string to use else where in your playbook
- gc_storage: bucket=mybucket object=key.txt mode=get_str

# Create an empty bucket
- gc_storage: bucket=mybucket mode=create

# Create a bucket with key as directory
- gc_storage: bucket=mybucket object=/my/directory/path mode=create

# Delete a bucket and all contents
- gc_storage: bucket=mybucket mode=delete
```

gce - create or terminate GCE instances

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Creates or terminates Google Compute Engine (GCE) instances. See https://cloud.google.com/products/compute-engine for an overview. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Basic provisioning example.  Create a single Debian 7 instance in the
# us-central1-a Zone of n1-standard-1 machine type.
- local_action:
    module: gce
    name: test-instance
    zone: us-central1-a
    machine_type: n1-standard-1
    image: debian-7

# Example using defaults and with metadata to create a single 'foo' instance
- local_action:
    module: gce
    name: foo
    metadata: '{"db":"postgres", "group":"qa", "id":500}'


# Launch instances from a control node, runs some tasks on the new instances,
# and then terminate them
- name: Create a sandbox instance
  hosts: localhost
  vars:
    names: foo,bar
    machine_type: n1-standard-1
    image: debian-6
    zone: us-central1-a
  tasks:
    - name: Launch instances
      local_action: gce instance_names={{names}} machine_type={{machine_type}}
                    image={{image}} zone={{zone}}
      register: gce
    - name: Wait for SSH to come up
      local_action: wait_for host={{item.public_ip}} port=22 delay=10
                    timeout=60 state=started
      with_items: {{gce.instance_data}}

- name: Configure instance(s)
  hosts: launched
  sudo: True
  roles:
    - my_awesome_role
    - my_awesome_tasks

- name: Terminate instances
  hosts: localhost
  connection: local
  tasks:
    - name: Terminate instances that were previously launched
      local_action:
        module: gce
        state: 'absent'
        instance_names: {{gce.instance_names}}
```

gce_lb - create/destroy GCE load-balancer resources

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module can create and destroy Google Compute Engine `loadbalancer` and `httphealthcheck` resources. The primary LB resource is the `load_balancer` resource and the health check parameters are all prefixed with *httphealthcheck*. The full documentation for Google Compute Engine load balancing is at https://developers.google.com/compute/docs/load-balancing/. However, the ansible module simplifies the configuration by following the libcloud model. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Simple example of creating a new LB, adding members, and a health check
- local_action:
    module: gce_lb
    name: testlb
    region: us-central1
    members: ["us-central1-a/www-a", "us-central1-b/www-b"]
    httphealthcheck_name: hc
    httphealthcheck_port: 80
    httphealthcheck_path: "/up"
```

gce_net - create/destroy GCE networks and firewall rules

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module can create and destroy Google Compue Engine networks and firewall rules https://developers.google.com/compute/docs/networking. The *name* parameter is reserved for referencing a network while the *fwname* parameter is used to reference firewall rules. IPv4 Address ranges must be specified using

the CIDR http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing format. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Simple example of creating a new network
- local_action:
    module: gce_net
    name: privatenet
    ipv4_range: '10.240.16.0/24'

# Simple example of creating a new firewall rule
- local_action:
    module: gce_net
    name: privatenet
    allowed: tcp:80,8080
    src_tags: ["web", "proxy"]
```

gce_pd - utilize GCE persistent disk resources

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module can create and destroy unformatted GCE persistent disks https://developers.google.com/compute/docs/disks#persistentdisks It also supports attaching and detaching disks from running instances but does not support creating boot disks from images or snapshots. The 'gce' module supports creating instances with boot disks. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Simple attachment action to an existing instance
- local_action:
    module: gce_pd
    instance_name: notlocalhost
    size_gb: 5
    name: pd
```

gem - Manage Ruby gems

Author Johan Wiren

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage installation and uninstallation of Ruby gems.

Options

Examples

```
# Installs version 1.0 of vagrant.
- gem: name=vagrant version=1.0 state=present

# Installs latest available version of rake.
- gem: name=rake state=latest

# Installs rake version 1.0 from a local gem on disk.
- gem: name=rake gem_source=/path/to/gems/rake-1.0.gem state=present
```

get_url - Downloads files from HTTP, HTTPS, or FTP to node

Author Jan-Piet Mens

- Synopsis
- Options
- Examples

Synopsis

Downloads files from HTTP, HTTPS, or FTP to the remote server. The remote server *must* have direct access to the remote resource. By default, if an environment variable `<protocol>_proxy` is set on the target host, requests will be sent through that proxy. This behaviour can be overridden by setting a variable for this task (see setting the environment), or by using the use_proxy option.

Options

Note: Requires urllib2

Note: Requires urlparse

Examples

```
- name: download foo.conf
  get_url: url=http://example.com/path/file.conf dest=/etc/foo.conf mode=0440

- name: download file with sha256 check
  get_url: url=http://example.com/path/file.conf dest=/etc/foo.conf sha256sum=b5bb9d8014a0f9b1d61e21e
```

Note: This module doesn't yet support configuration for proxies.

git - Deploy software (or files) from git checkouts

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

Manage *git* checkouts of repositories to deploy files or software.

Options

Examples

```
# Example git checkout from Ansible Playbooks
- git: repo=git://foosball.example.org/path/to/repo.git
       dest=/srv/checkout
       version=release-0.22

# Example read-write git checkout from github
```

```
- git: repo=ssh://git@github.com/mylogin/hello.git dest=/home/mylogin/hello

# Example just ensuring the repo checkout exists
- git: repo=git://foosball.example.org/path/to/repo.git dest=/srv/checkout update=no
```

Note: If the task seems to be hanging, first verify remote host is in `known_hosts`. SSH will prompt user to authorize the first contact with a remote host. To avoid this prompt, one solution is to add the remote host public key in `/etc/ssh/ssh_known_hosts` before calling the git module, with the following command: ssh-keyscan remote_host.com >> /etc/ssh/ssh_known_hosts.

github_hooks - Manages github service hooks.

Author Phillip Gentry, CX Inc

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Adds service hooks and removes service hooks that have an error status.

Options

Examples

```
# Example creating a new service hook. It ignores duplicates.
- github_hooks: action=create hookurl=http://11.111.111.111:2222 user={{ gituser }} oauthkey={{ oauth

# Cleaning all hooks for this repo that had an error on the last update. Since this works for all hoo
- local_action: github_hooks action=cleanall user={{ gituser }} oauthkey={{ oauthkey }} repo={{ repo
```

glance_image - Add/Delete images from glance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove images from the glance repository.

Options

Note: Requires glanceclient

Note: Requires keystoneclient

Examples

```
# Upload an image from an HTTP URL
- glance_image: login_username=admin
                login_password=passme
                login_tenant_name=admin
                name=cirros
                container_format=bare
                disk_format=qcow2
                state=present
                copy_from=http:launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.ir
```

group - Add or remove groups

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Manage presence of groups on a host.

Options

Note: Requires groupadd

Note: Requires groupdel

Note: Requires groupmod

Examples

```
# Example group command from Ansible Playbooks
- group: name=somegroup state=present
```

group_by - Create Ansible groups based on facts

Author Jeroen Hoekx

- Synopsis
- Options
- Examples

Synopsis

Use facts to create ad-hoc groups that can be used later in a playbook.

Options

Examples

```
# Create groups based on the machine architecture
-  group_by: key=machine_{{ ansible_machine }}
# Create groups like 'kvm-host'
-  group_by: key=virt_{{ ansible_virtualization_type }}_{{ ansible_virtualization_role }}
```

Note: Spaces in group names are converted to dashes '-'.

grove - Sends a notification to a grove.io channel

Author Jonas Pfenniger <zimbatm@zimbatm.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

The `grove` module sends a message for a service to a Grove.io channel.

Options

Examples

```
- grove: >
    channel_token=6Ph62VBBJOccmtTPZbubiPzdrhipZXtg
    service=my-app
    message=deployed {{ target }}
```

hg - Manages Mercurial (hg) repositories.

Author Yeukhon Wong

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Manages Mercurial (hg) repositories. Supports SSH, HTTP/S and local address.

Options

Examples

```
# Ensure the current working copy is inside the stable branch and deletes untracked files if any.
- hg: repo=https://bitbucket.org/user/repo1 dest=/home/user/repo1 revision=stable purge=yes
```

Note: If the task seems to be hanging, first verify remote host is in `known_hosts`. SSH will prompt user to authorize the first contact with a remote host. To avoid this prompt, one solution is to add the remote host public key in `/etc/ssh/ssh_known_hosts` before calling the hg module, with the following command: ssh-keyscan remote_host.com >> /etc/ssh/ssh_known_hosts.

---

hipchat - Send a message to hipchat

Author WAKAYAMA Shirou

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to hipchat

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- hipchat: token=AAAAAA room=notify msg="Ansible task finished"
```

homebrew - Package manager for Homebrew

Author Andrew Dunham

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages Homebrew packages

Options

Examples

```
- homebrew: name=foo state=present
- homebrew: name=foo state=present update_homebrew=yes
- homebrew: name=foo state=absent
- homebrew: name=foo,bar state=absent
- homebrew: name=foo state=present install_options=with-baz,enable-debug
```

hostname - Manage hostname

Author Hiroaki Nakamura

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Set system's hostname Currently implemented on only Debian, Ubuntu, RedHat and CentOS.

Options

Note: Requires hostname

Examples

```
- hostname: name=web01
```

htpasswd - manage user files for basic authentication

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Add and remove username/password entries in a password file using htpasswd. This is used by web servers such as Apache and Nginx for basic authentication.

Options

Examples

```
# Add a user to a password file and ensure permissions are set
- htpasswd: path=/etc/nginx/passwdfile name=janedoe password=9s36?;fyNp owner=root group=www-data mo
# Remove a user from a password file
- htpasswd: path=/etc/apache2/passwdfile name=foobar state=absent
```

Note: This module depends on the *passlib* Python library, which needs to be installed on all target systems.

Note: On Debian, Ubuntu, or Fedora: install *python-passlib*.

Note: On RHEL or CentOS: Enable EPEL, then install *python-passlib*.

include_vars - Load variables from files, dynamically within a task.

Author Benno Joy

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Loads variables from a YAML file dynamically during task runtime. It can work with conditionals, or use host specific variables to determine the path name to load from.

Options

Examples

```
# Conditionally decide to load in variables when x is 0, otherwise do not.
- include_vars: contingency_plan.yml
  when: x == 0

# Load a variable file based on the OS type, or a default if not found.
- include_vars: "{{ item }}"
  with_first_found:
   - "{{ ansible_os_distribution }}.yml"
   - "default.yml"
```

ini_file - Tweak settings in INI files

Author Jan-Piet Mens

- Synopsis
- Options
- Examples

Synopsis

Manage (add, remove, change) individual settings in an INI-style file without having to manage the file as a whole with, say, `template` or `assemble`. Adds missing sections if they don't exist. Comments are discarded when the source file is read, and therefore will not show up in the destination file.

Options

Note: Requires ConfigParser

Examples

```
# Ensure "fav=lemonade is in section "[drinks]" in specified file
- ini_file: dest=/etc/conf section=drinks option=fav value=lemonade mode=0600 backup=yes

- ini_file: dest=/etc/anotherconf
            section=drinks
            option=temperature
```

```
value=cold
backup=yes
```

Note: While it is possible to add an *option* without specifying a *value*, this makes no sense.

Note: A section named `default` cannot be added by the module, but if it exists, individual options within the section can be updated. (This is a limitation of Python's *ConfigParser*.) Either use `template` to create a base INI file with a `[default]` section, or use `lineinfile` to add the missing line.

irc - Send a message to an IRC channel

Author Jan-Piet Mens, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to an IRC channel. This is a very simplistic implementation.

Options

Note: Requires socket

Examples

```
- irc: server=irc.example.net channel="#t1" msg="Hello world"

- local_action: irc port=6669
                channel="#t1"
                msg="All finished at {{ ansible_date_time.iso8601 }}"
                color=red
                nick=ansibleIRC
```

jabber - Send a message to jabber user or chat room

Author Brian Coca

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to jabber

Options

Note: Requires xmpp

Examples

```
# send a message to a user
- jabber: user=mybot@example.net
          password=secret
          to=friend@example.net
          msg="Ansible task finished"

# send a message to a room
- jabber: user=mybot@example.net
          password=secret
          to=mychaps@conference.example.net/ansiblebot
          msg="Ansible task finished"

# send a message, specifying the host and port
- jabber user=mybot@example.net
         host=talk.example.net
         port=5223
         password=secret
         to=mychaps@example.net
         msg="Ansible task finished"
```

jboss - deploy applications to JBoss

Author Jeroen Hoekx

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Deploy applications to JBoss standalone using the filesystem

Options

Examples

```
# Deploy a hello world application
- jboss: src=/tmp/hello-1.0-SNAPSHOT.war deployment=hello.war state=present
# Update the hello world application
- jboss: src=/tmp/hello-1.1-SNAPSHOT.war deployment=hello.war state=present
# Undeploy the hello world application
- jboss: deployment=hello.war state=absent
```

Note: The JBoss standalone deployment-scanner has to be enabled in standalone.xml

Note: Ensure no identically named application is deployed through the JBoss CLI

kernel_blacklist - Blacklist kernel modules

Author Matthias Vogelgesang

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or remove kernel modules from blacklist.

Options

Examples

```
# Blacklist the nouveau driver module
- kernel_blacklist: name=nouveau state=present
```

keystone_user - Manage OpenStack Identity (keystone) users, tenants and roles

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage users,tenants, roles from OpenStack.

Options

Note: Requires python-keystoneclient

Examples

```
# Create a tenant
- keystone_user: tenant=demo tenant_description="Default Tenant"

# Create a user
- keystone_user: user=john tenant=demo password=secrete

# Apply the admin role to the john user in the demo tenant
- keystone_user: role=admin user=john tenant=demo
```

lineinfile - Ensure a particular line is in a file, or replace an existing line using a back-referenced regular expression.

Author Daniel Hokka Zakrisson

- Synopsis
- Options
- Examples

Synopsis

This module will search a file for a line, and ensure that it is present or absent. This is primarily useful when you want to change a single line in a file only. For other cases, see the `copy` or `template` modules.

Options

Examples

```
- lineinfile: dest=/etc/selinux/config regexp=^SELINUX= line=SELINUX=disabled

- lineinfile: dest=/etc/sudoers state=absent regexp="^%wheel"

- lineinfile: dest=/etc/hosts regexp='^127\.0\.0\.1' line='127.0.0.1 localhost' owner=root group=root

- lineinfile: dest=/etc/httpd/conf/httpd.conf regexp="^Listen " insertafter="^#Listen " line="Listen
```

```
- lineinfile: dest=/etc/services regexp="^# port for http" insertbefore="^www.*80/tcp" line="# port f

# Add a line to a file if it does not exist, without passing regexp
- lineinfile: dest=/tmp/testfile line="192.168.1.99 foo.lab.net foo"

# Fully quoted because of the ': ' on the line. See the Gotchas in the YAML docs.
- lineinfile: "dest=/etc/sudoers state=present regexp='^%wheel' line='%wheel ALL=(ALL) NOPASSWD: ALL'

- lineinfile: dest=/opt/jboss-as/bin/standalone.conf regexp='^(.*)Xms(\d+)m(.*)$' line='\1Xms${xms}m\

# Validate a the sudoers file before saving
- lineinfile: dest=/etc/sudoers state=present regexp='^%ADMIN ALL\=' line='%ADMIN ALL=(ALL) NOPASSWD:
```

linode - create / delete / stop / restart an instance in Linode Public Cloud

Author Vincent Viallet

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

creates / deletes a Linode Public Cloud instance and optionally waits for it to be 'running'.

Options

Note: Requires linode-python

Examples

```
# Create a server
- local_action:
     module: linode
     api_key: 'longStringFromLinodeApi'
     name: linode-test1
     plan: 1
     datacenter: 2
     distribution: 99
     password: 'superSecureRootPassword'
     ssh_pub_key: 'ssh-rsa qwerty'
     swap: 768
     wait: yes
     wait_timeout: 600
     state: present

# Ensure a running server (create if missing)
- local_action:
```

```
      module: linode
      api_key: 'longStringFromLinodeApi'
      name: linode-test1
      linode_id: 12345678
      plan: 1
      datacenter: 2
      distribution: 99
      password: 'superSecureRootPassword'
      ssh_pub_key: 'ssh-rsa qwerty'
      swap: 768
      wait: yes
      wait_timeout: 600
      state: present

# Delete a server
- local_action:
      module: linode
      api_key: 'longStringFromLinodeApi'
      name: linode-test1
      linode_id: 12345678
      state: absent

# Stop a server
- local_action:
      module: linode
      api_key: 'longStringFromLinodeApi'
      name: linode-test1
      linode_id: 12345678
      state: stopped

# Reboot a server
- local_action:
      module: linode
      api_key: 'longStringFromLinodeApi'
      name: linode-test1
      linode_id: 12345678
      state: restarted
```

Note: LINODE_API_KEY env variable can be used instead

lvg - Configure LVM volume groups

Author Alexander Bulimov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

This module creates, removes or resizes volume groups.

Options

Examples

```
# Create a volume group on top of /dev/sda1 with physical extent size = 32MB.
- lvg:  vg=vg.services pvs=/dev/sda1 pesize=32

# Create or resize a volume group on top of /dev/sdb1 and /dev/sdc5.
# If, for example, we already have VG vg.services on top of /dev/sdb1,
# this VG will be extended by /dev/sdc5.  Or if vg.services was created on
# top of /dev/sda5, we first extend it with /dev/sdb1 and /dev/sdc5,
# and then reduce by /dev/sda5.
- lvg: vg=vg.services pvs=/dev/sdb1,/dev/sdc5

# Remove a volume group with name vg.services.
- lvg: vg=vg.services state=absent
```

Note: module does not modify PE size for already present volume group

lvol - Configure LVM logical volumes

Author Jeroen Hoekx

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

This module creates, removes or resizes logical volumes.

Options

Examples

```
# Create a logical volume of 512m.
- lvol: vg=firefly lv=test size=512

# Create a logical volume of 512g.
- lvol: vg=firefly lv=test size=512g

# Create a logical volume the size of all remaining space in the volume group
- lvol: vg=firefly lv=test size=100%FREE

# Extend the logical volume to 1024m.
- lvol: vg=firefly lv=test size=1024

# Reduce the logical volume to 512m
```

```
- lvol: vg=firefly lv=test size=512

# Remove the logical volume.
- lvol: vg=firefly lv=test state=absent
```

Note: Filesystems on top of the volume are not resized.

macports - Package manager for MacPorts

Author Jimmy Tang

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages MacPorts packages

Options

Examples

```
- macports: name=foo state=present
- macports: name=foo state=present update_cache=yes
- macports: name=foo state=absent
- macports: name=foo state=active
- macports: name=foo state=inactive
```

mail - Send an email

Author Dag Wieers

- Synopsis
- Options
- Examples

Synopsis

This module is useful for sending emails from playbooks. One may wonder why automate sending emails? In complex environments there are from time to time processes that cannot be automated, either because you lack the authority to make it so, or because not everyone agrees to a common approach. If you cannot automate a specific step, but the step is non-blocking, sending out an email to the responsible party to make him perform his part of the bargain is an

elegant way to put the responsibility in someone else's lap. Of course sending out a mail can be equally useful as a way to notify one or more people in a team that a specific action has been (successfully) taken.

Options

Examples

```
# Example playbook sending mail to root
- local_action: mail msg='System {{ ansible_hostname }} has been successfully provisioned.'

# Send e-mail to a bunch of users, attaching files
- local_action: mail
                host='127.0.0.1'
                port=2025
                subject="Ansible-report"
                body="Hello, this is an e-mail. I hope you like it ;-)"
                from="jane@example.net (Jane Jolie)"
                to="John Doe <j.d@example.org>, Suzie Something <sue@example.com>"
                cc="Charlie Root <root@localhost>"
                attach="/etc/group /tmp/pavatar2.png"
                headers=Reply-To=john@example.com|X-Special="Something or other"
                charset=utf8
```

modprobe - Add or remove kernel modules

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or remove kernel modules.

Options

Examples

```
# Add the 802.1q module
- modprobe: name=8021q state=present
```

mongodb_user - Adds or removes a user from a MongoDB database.

Author Elliott Foster

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Adds or removes a user from a MongoDB database.

Options

Note: Requires pymongo

Examples

```
# Create 'burgers' database user with name 'bob' and password '12345'.
- mongodb_user: database=burgers name=bob password=12345 state=present

# Delete 'burgers' database user with name 'bob'.
- mongodb_user: database=burgers name=bob state=absent

# Define more users with various specific roles (if not defined, no roles is assigned, and the user w
- mongodb_user: database=burgers name=ben password=12345 roles='read' state=present
- mongodb_user: database=burgers name=jim password=12345 roles='readWrite,dbAdmin,userAdmin' state=pr
- mongodb_user: database=burgers name=joe password=12345 roles='readWriteAnyDatabase' state=present
```

Note: Requires the pymongo Python package on the remote host, version 2.4.2+. This can be installed using pip or the OS package manager. @see http://api.mongodb.org/python/current/installation.html

monit - Manage the state of a program monitored via Monit

Author Darryl Stoflet

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage the state of a program monitored via *Monit*

Options

Examples

```
# Manage the state of program "httpd" to be in "started" state.
- monit: name=httpd state=started
```

mount - Control active and configured mount points

Author Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

This module controls active and configured mount points in `/etc/fstab`.

Options

Examples

```
# Mount DVD read-only
- mount: name=/mnt/dvd src=/dev/sr0 fstype=iso9660 opts=ro state=present

# Mount up device by label
- mount: name=/srv/disk src='LABEL=SOME_LABEL' state=present

# Mount up device by UUID
- mount: name=/home src='UUID=b3e48f45-f933-4c8e-a700-22a159ec9077' opts=noatime state=present
```

mqtt - Publish a message on an MQTT topic for the IoT

Author Jan-Piet Mens

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Publish a message on an MQTT topic.

Options

Note: Requires mosquitto

Examples

```
- local_action: mqtt
              topic=service/ansible/{{ ansible_hostname }}
              payload="Hello at {{ ansible_date_time.iso8601 }}"
              qos=0
              retain=false
              client_id=ans001
```

Note: This module requires a connection to an MQTT broker such as Mosquitto http://mosquitto.org and the `mosquitto` Python module (http://mosquitto.org/python).

mysql_db - Add or remove MySQL databases from a remote host.

Author Mark Theunissen

- Synopsis
- Options
- Examples

Synopsis

Add or remove MySQL databases from a remote host.

Options

Note: Requires ConfigParser

Examples

```
# Create a new database with name 'bobdata'
- mysql_db: name=bobdata state=present
```

Note: Requires the MySQLdb Python package on the remote host. For Ubuntu, this is as easy as apt-get install python-mysqldb. (See `apt`.)

Note: Both *login_password* and *login_user* are required when you are passing credentials. If none are present, the module will attempt to read the credentials from `~/.my.cnf`, and finally fall back to using the MySQL default login of `root` with no password.

mysql_replication - Manage MySQL replication

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages MySQL server replication, slave, master status get and change master host.

Options

Examples

```
# Stop mysql slave thread
- mysql_replication: mode=stopslave

# Get master binlog file name and binlog position
- mysql_replication: mode=getmaster

# Change master to master server 192.168.1.1 and use binary log 'mysql-bin.000009' with position 4578
- mysql_replication: mode=changemaster master_host=192.168.1.1 master_log_file=mysql-bin.000009 maste
```

mysql_user - Adds or removes a user from a MySQL database.

Author Mark Theunissen

- Synopsis
- Options
- Examples

Synopsis

Adds or removes a user from a MySQL database.

Options

Note: Requires ConfigParser

Note: Requires MySQLdb

Examples

```
# Create database user with name 'bob' and password '12345' with all database privileges
- mysql_user: name=bob password=12345 priv=*.*:ALL state=present

# Creates database user 'bob' and password '12345' with all database privileges and 'WITH GRANT OPTIO
- mysql_user: name=bob password=12345 priv=*.*:ALL,GRANT state=present

# Ensure no user named 'sally' exists, also passing in the auth credentials.
- mysql_user: login_user=root login_password=123456 name=sally state=absent

# Example privileges string format
mydb.*:INSERT,UPDATE/anotherdb.*:SELECT/yetanotherdb.*:ALL

# Example using login_unix_socket to connect to server
- mysql_user: name=root password=abc123 login_unix_socket=/var/run/mysqld/mysqld.sock

# Example .my.cnf file for setting the root password
# Note: don't use quotes around the password, because the mysql_user module
# will include them in the password but the mysql client will not

[client]
user=root
password=n<_665{vS43y
```

Note: Requires the MySQLdb Python package on the remote host. For Ubuntu, this is as easy as apt-get install python-mysqldb.

Note: Both `login_password` and `login_username` are required when you are passing credentials. If none are present, the module will attempt to read the credentials from `~/.my.cnf`, and finally fall back to using the MySQL default login of 'root' with no password.

Note: MySQL server installs with default login_user of 'root' and no password. To secure this user as part of an idempotent playbook, you must create at least two tasks: the first must change the root user's password, without providing any login_user/login_password details. The second must drop a ~/.my.cnf file containing the new root credentials. Subsequent runs of the playbook will then succeed by reading the new credentials from the file.

mysql_variables - Manage MySQL global variables

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Query / Set MySQL variables

Options

Examples

```
# Check for sync_binary_log setting
- mysql_variables: variable=sync_binary_log

# Set read_only variable to 1
- mysql_variables: variable=read_only value=1
```

nagios - Perform common tasks in Nagios related to downtime and notifications.

Author Tim Bielawa

- Synopsis
- Options
- Examples

Synopsis

The `nagios` module has two basic functions: scheduling downtime and toggling alerts for services or hosts. All actions require the *host* parameter to be given explicitly. In playbooks you can use the `{{inventory_hostname}}` variable to refer to the host the playbook is currently running on. You can specify multiple services at once by separating them with commas, .e.g., `services=httpd,nfs,puppet`. When specifying what service to handle there is a special service value, *host*, which will handle alerts/downtime for the *host itself*, e.g., `service=host`. This keyword may not be given with other services at the same time. *Setting alerts/downtime for a host does not affect alerts/downtime for any of the services running on it.* To schedule downtime for all services on particular host use keyword "all", e.g., `service=all`. When using the `nagios` module you will need to specify your Nagios server using the `delegate_to` parameter.

Options

Note: Requires Nagios

Examples

```
# set 30 minutes of apache downtime
- nagios: action=downtime minutes=30 service=httpd host={{ inventory_hostname }}

# schedule an hour of HOST downtime
- nagios: action=downtime minutes=60 service=host host={{ inventory_hostname }}

# schedule downtime for ALL services on HOST
- nagios: action=downtime minutes=45 service=all host={{ inventory_hostname }}

# schedule downtime for a few services
- nagios: action=downtime services=frob,foobar,qeuz host={{ inventory_hostname }}
```

```
# enable SMART disk alerts
- nagios: action=enable_alerts service=smart host={{ inventory_hostname }}

# "two services at once: disable httpd and nfs alerts"
- nagios: action=disable_alerts service=httpd,nfs host={{ inventory_hostname }}

# disable HOST alerts
- nagios: action=disable_alerts service=host host={{ inventory_hostname }}

# silence ALL alerts
- nagios: action=silence host={{ inventory_hostname }}

# unsilence all alerts
- nagios: action=unsilence host={{ inventory_hostname }}

# SHUT UP NAGIOS
- nagios: action=silence_nagios

# ANNOY ME NAGIOS
- nagios: action=unsilence_nagios

# command something
- nagios: action=command command='DISABLE_FAILURE_PREDICTION'
```

netscaler - Manages Citrix NetScaler entities

Author Nandor Sivok

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages Citrix NetScaler server and service entities.

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
# Disable the server
ansible host -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass"
```

```
# Enable the server
ansible host -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass action=enable"

# Disable the service local:8080
ansible host -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass name=local:8080
```

newrelic_deployment - Notify newrelic about app deployments

Author Matt Coddington

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Notify newrelic about app deployments (see http://newrelic.github.io/newrelic_api/NewRelicApi/Deployment.html)

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- newrelic_deployment: token=AAAAAA
                       app_name=myapp
                       user='ansible deployment'
                       revision=1.0
```

nova_compute - Create/Delete VMs from OpenStack

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Create or Remove virtual machines from Openstack.

Options

Note: Requires novaclient

Examples

```
# Creates a new VM and attaches to a network and passes metadata to the instance
- nova_compute:
       state: present
       login_username: admin
       login_password: admin
       login_tenant_name: admin
       name: vm1
       image_id: 4f905f38-e52a-43d2-b6ec-754a13ffb529
       key_name: ansible_key
       wait_for: 200
       flavor_id: 4
       nics:
         - net-id: 34605f38-e52a-25d2-b6ec-754a13ffb723
       meta:
         hostname: test1
         group: uge_master
```

nova_keypair - Add/Delete key pair from nova

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove key pair from nova .

Options

Note: Requires novaclient

Examples

```
# Creates a key pair with the running users public key
- nova_keypair: state=present login_username=admin
                login_password=admin login_tenant_name=admin name=ansible_key
                public_key={{ lookup('file','~/.ssh/id_rsa.pub') }}
```

```
# Creates a new key pair and the private key returned after the run.
- nova_keypair: state=present login_username=admin login_password=admin
                login_tenant_name=admin name=ansible_key
```

npm - Manage node.js packages with npm

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage node.js packages with Node Package Manager (npm)

Options

Examples

```
description: Install "coffee-script" node.js package.
- npm: name=coffee-script path=/app/location

description: Install "coffee-script" node.js package on version 1.6.1.
- npm: name=coffee-script version=1.6.1 path=/app/location

description: Install "coffee-script" node.js package globally.
- npm: name=coffee-script global=yes

description: Remove the globally package "coffee-script".
- npm: name=coffee-script global=yes state=absent

description: Install packages based on package.json.
- npm: path=/app/location

description: Update packages based on package.json to their latest version.
- npm: path=/app/location state=latest

description: Install packages based on package.json using the npm installed with nvm v0.10.1.
- npm: path=/app/location executable=/opt/nvm/v0.10.1/bin/npm state=present
```

ohai - Returns inventory data from *Ohai*

Author Michael DeHaan

- Synopsis
- Examples

Synopsis

Similar to the `facter` module, this runs the *Ohai* discovery program (http://wiki.opscode.com/display/chef/Ohai) on the remote host and returns JSON inventory data. *Ohai* data is a bit more verbose and nested than *facter*.

Note: Requires ohai

Examples

```
# Retrieve (ohai) data from all Web servers and store in one-file per host
ansible webservers -m ohai --tree=/tmp/ohaidata
```

open_iscsi - Manage iscsi targets with open-iscsi

Author Serge van Ginderachter

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Discover targets on given portal, (dis)connect targets, mark targets to manually or auto start, return device nodes of connected targets.

Options

Note: Requires open_iscsi library and tools (iscsiadm)

Examples

openbsd_pkg - Manage packages on OpenBSD.

Author Patrik Lundin

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage packages on OpenBSD using the pkg tools.

Options

Examples

```
# Make sure nmap is installed
- openbsd_pkg: name=nmap state=present

# Make sure nmap is the latest version
- openbsd_pkg: name=nmap state=latest

# Make sure nmap is not installed
- openbsd_pkg: name=nmap state=absent
```

openvswitch_bridge - Manage Open vSwitch bridges

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage Open vSwitch bridges

Options

Note: Requires ovs-vsctl

Examples

```
# Create a bridge named br-int
- openvswitch_bridge: bridge=br-int state=present
```

openvswitch_port - Manage Open vSwitch ports

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage Open vSwitch ports

Options

Note: Requires ovs-vsctl

Examples

```
# Creates port eth2 on bridge br-ex
- openvswitch_port: bridge=br-ex port=eth2 state=present
```

opkg - Package manager for OpenWrt

Author Patrick Pelletier

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages OpenWrt packages

Options

Examples

```
- opkg: name=foo state=present
- opkg: name=foo state=present update_cache=yes
- opkg: name=foo state=absent
- opkg: name=foo,bar state=absent
```

osx_say - Makes an OSX computer to speak.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

makes an OS computer speak! Amuse your friends, annoy your coworkers!

Options

Note: Requires say

Examples

```
- local_action: osx_say msg="{{inventory_hostname}} is all done" voice=Zarvox
```

Note: If you like this module, you may also be interested in the osx_say callback in the plugins/ directory of the source checkout.

ovirt - oVirt/RHEV platform management

Author Vincent Van der Kussen

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

allows you to create new instances, either from scratch or an image, in addition to deleting or stopping instances on the oVirt/RHEV platform

Options

Note: Requires ovirt-engine-sdk

Examples

```
# Basic example provisioning from image.

action: ovirt >
    user=admin@internal
    url=https://ovirt.example.com
    instance_name=ansiblevm04
    password=secret
    image=centos_64
    zone=cluster01
    resource_type=template"

# Full example to create new instance from scratch
action: ovirt >
    instance_name=testansible
    resource_type=new
    instance_type=server
    user=admin@internal
    password=secret
    url=https://ovirt.example.com
    instance_disksize=10
    zone=cluster01
    region=datacenter1
    instance_cpus=1
    instance_nic=nic1
    instance_network=rhevm
    instance_mem=1000
    disk_alloc=thin
    sdomain=FIBER01
    instance_cores=1
    instance_os=rhel_6x64
    disk_int=virtio"

# stopping an instance
action: ovirt >
    instance_name=testansible
    state=stopped
    user=admin@internal
    password=secret
    url=https://ovirt.example.com

# starting an instance
action: ovirt >
    instance_name=testansible
    state=started
    user=admin@internal
    password=secret
    url=https://ovirt.example.com
```

pacman - Package manager for Archlinux

Author Afterburn

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Manages Archlinux packages

Options

Examples

```
# Install package foo
- pacman: name=foo state=installed

# Remove package foo
- pacman: name=foo state=absent

# Remove packages foo and bar
- pacman: name=foo,bar state=absent

# Recursively remove package baz
- pacman: name=baz state=absent recurse=yes

# Update the package database (pacman -Syy) and install bar (bar will be the updated if a newer versi
- pacman: name=bar, state=installed, update_cache=yes
```

pagerduty - Create PagerDuty maintenance windows

Author Justin Johns

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module will let you create PagerDuty maintenance windows

Options

Note: Requires PagerDuty API access

Examples

```
# List ongoing maintenance windows.
- pagerduty: name=companyabc user=example@example.com passwd=password123 state=ongoing

# Create a 1 hour maintenance window for service FOO123.
- pagerduty: name=companyabc
             user=example@example.com
             passwd=password123
             state=running
             service=FOO123

# Create a 4 hour maintenance window for service FOO123 with the description "deployment".
- pagerduty: name=companyabc
             user=example@example.com
             passwd=password123
             state=running
             service=FOO123
             hours=4
             desc=deployment
```

Note: This module does not yet have support to end maintenance windows.

pause - Pause playbook execution

Author Tim Bielawa

- Synopsis
- Options
- Examples

Synopsis

Pauses playbook execution for a set amount of time, or until a prompt is acknowledged. All parameters are optional. The default behavior is to pause with a prompt. You can use `ctrl+c` if you wish to advance a pause earlier than it is set to expire or if you need to abort a playbook run entirely. To continue early: press `ctrl+c` and then `c`. To abort a playbook: press `ctrl+c` and then `a`. The pause module integrates into async/parallelized playbooks without any special considerations (see also: Rolling Updates). When using pauses with the `serial` playbook parameter (as in rolling updates) you are only prompted once for the current group of hosts.

Options

Examples

```
# Pause for 5 minutes to build app cache.
- pause: minutes=5

# Pause until you can verify updates to an application were successful.
- pause:
```

```
# A helpful reminder of what to look out for post-update.
- pause: prompt="Make sure org.foo.FooOverload exception is not present"
```

ping - Try to connect to host and return `pong` on success.

Author Michael DeHaan

- Synopsis
- Examples

Synopsis

A trivial test module, this module always returns `pong` on successful contact. It does not make sense in playbooks, but it is useful from `/usr/bin/ansible`

Examples

```
# Test 'webservers' status
ansible webservers -m ping
```

pingdom - Pause/unpause Pingdom alerts

Author Justin Johns

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module will let you pause/unpause Pingdom alerts

Options

Note: Requires This pingdom python library: https://github.com/mbabineau/pingdom-python

Examples

```
# Pause the check with the ID of 12345.
- pingdom: uid=example@example.com
           passwd=password123
           key=apipassword123
           checkid=12345
           state=paused

# Unpause the check with the ID of 12345.
- pingdom: uid=example@example.com
           passwd=password123
           key=apipassword123
           checkid=12345
           state=running
```

Note: This module does not yet have support to add/remove checks.

pip - Manages Python library dependencies.

Author Matt Wright

- Synopsis
- Options
- Examples

Synopsis

Manage Python library dependencies. To use this module, one of the following keys is required: `name` or `requirements`.

Options

Note: Requires virtualenv

Note: Requires pip

Examples

```
# Install (Bottle) python package.
- pip: name=bottle

# Install (Bottle) python package on version 0.11.
- pip: name=bottle version=0.11

# Install (MyApp) using one of the remote protocols (bzr+,hg+,git+,svn+). You do not have to supply '
- pip: name='svn+http://myrepo/svn/MyApp#egg=MyApp'

# Install (Bottle) into the specified (virtualenv), inheriting none of the globally installed modules
```

```
- pip: name=bottle virtualenv=/my_app/venv

# Install (Bottle) into the specified (virtualenv), inheriting globally installed modules
- pip: name=bottle virtualenv=/my_app/venv virtualenv_site_packages=yes

# Install (Bottle) into the specified (virtualenv), using Python 2.7
- pip: name=bottle virtualenv=/my_app/venv virtualenv_command=virtualenv-2.7

# Install specified python requirements.
- pip: requirements=/my_app/requirements.txt

# Install specified python requirements in indicated (virtualenv).
- pip: requirements=/my_app/requirements.txt virtualenv=/my_app/venv

# Install specified python requirements and custom Index URL.
- pip: requirements=/my_app/requirements.txt extra_args='-i https://example.com/pypi/simple'

# Install (Bottle) for Python 3.3 specifically,using the 'pip-3.3' executable.
- pip: name=bottle executable=pip-3.3
```

Note: Please note that virtualenv (http://www.virtualenv.org/) must be installed on the remote host if the virtualenv parameter is specified.

pkgin - Package manager for SmartOS

Author Shaun Zinck

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Manages SmartOS packages

Options

Examples

```
# install package foo"
- pkgin: name=foo state=present

# remove package foo
- pkgin: name=foo state=absent

# remove packages foo and bar
- pkgin: name=foo,bar state=absent
```

pkgng - Package manager for FreeBSD >= 9.0

Author bleader

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage binary packages for FreeBSD using 'pkgng' which is available in versions after 9.0.

Options

Examples

```
# Install package foo
- pkgng: name=foo state=present

# Remove packages foo and bar
- pkgng: name=foo,bar state=absent
```

Note: When using pkgsite, be careful that already in cache packages won't be downloaded again.

pkgutil - Manage CSW-Packages on Solaris

Author Alexander Winkler

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages CSW packages (SVR4 format) on Solaris 10 and 11. These were the native packages on Solaris <= 10 and are available as a legacy feature in Solaris 11. Pkgutil is an advanced packaging system, which resolves dependency on installation. It is designed for CSW packages.

Options

Examples

```
# Install a package
pkgutil: name=CSWcommon state=present

# Install a package from a specific repository
pkgutil: name=CSWnrpe site='ftp://myinternal.repo/opencsw/kiel state=latest'
```

portinstall - Installing packages from FreeBSD's ports system

Author berenddeboer

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage packages for FreeBSD using 'portinstall'.

Options

Examples

```
# Install package foo
- portinstall: name=foo state=present

# Install package security/cyrus-sasl2-saslauthd
- portinstall: name=security/cyrus-sasl2-saslauthd state=present

# Remove packages foo and bar
- portinstall: name=foo,bar state=absent
```

postgresql_db - Add or remove PostgreSQL databases from a remote host.

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

Add or remove PostgreSQL databases from a remote host.

Options

Note: Requires psycopg2

Examples

```
# Create a new database with name "acme"
- postgresql_db: name=acme

# Create a new database with name "acme" and specific encoding and locale
# settings. If a template different from "template0" is specified, encoding
# and locale settings must match those of the template.
- postgresql_db: name=acme
                 encoding='UTF-8'
                 lc_collate='de_DE.UTF-8'
                 lc_ctype='de_DE.UTF-8'
                 template='template0'
```

Note: The default authentication assumes that you are either logging in as or sudo'ing to the `postgres` account on the host.

Note: This module uses *psycopg2*, a Python PostgreSQL database adapter. You must ensure that psycopg2 is installed on the host before using this module. If the remote host is the PostgreSQL server (which is the default case), then PostgreSQL must also be installed on the remote host. For Ubuntu-based systems, install the `postgresql`, `libpq-dev`, and `python-psycopg2` packages on the remote host before using this module.

postgresql_privs - Grant or revoke privileges on PostgreSQL database objects.

Author Bernhard Weitzhofer

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Grant or revoke privileges on PostgreSQL database objects. This module is basically a wrapper around most of the functionality of PostgreSQL's GRANT and REVOKE statements with detection of changes (GRANT/REVOKE *privs* ON *type objs* TO/FROM *roles*)

Options

Note: Requires psycopg2

Examples

```
# On database "library":
# GRANT SELECT, INSERT, UPDATE ON TABLE public.books, public.authors
# TO librarian, reader WITH GRANT OPTION
- postgresql_privs: >
    database=library
    state=present
    privs=SELECT,INSERT,UPDATE
    type=table
    objs=books,authors
    schema=public
    roles=librarian,reader
    grant_option=yes

# Same as above leveraging default values:
- postgresql_privs: >
    db=library
    privs=SELECT,INSERT,UPDATE
    objs=books,authors
    roles=librarian,reader
    grant_option=yes

# REVOKE GRANT OPTION FOR INSERT ON TABLE books FROM reader
# Note that role "reader" will be *granted* INSERT privilege itself if this
# isn't already the case (since state=present).
- postgresql_privs: >
    db=library
    state=present
    priv=INSERT
    obj=books
    role=reader
    grant_option=no

# REVOKE INSERT, UPDATE ON ALL TABLES IN SCHEMA public FROM reader
# "public" is the default schema. This also works for PostgreSQL 8.x.
- postgresql_privs: >
    db=library
    state=absent
    privs=INSERT,UPDATE
    objs=ALL_IN_SCHEMA
    role=reader

# GRANT ALL PRIVILEGES ON SCHEMA public, math TO librarian
- postgresql_privs: >
    db=library
    privs=ALL
    type=schema
    objs=public,math
    role=librarian
```

```
# GRANT ALL PRIVILEGES ON FUNCTION math.add(int, int) TO librarian, reader
# Note the separation of arguments with colons.
- postgresql_privs: >
    db=library
    privs=ALL
    type=function
    obj=add(int:int)
    schema=math
    roles=librarian,reader

# GRANT librarian, reader TO alice, bob WITH ADMIN OPTION
# Note that group role memberships apply cluster-wide and therefore are not
# restricted to database "library" here.
- postgresql_privs: >
    db=library
    type=group
    objs=librarian,reader
    roles=alice,bob
    admin_option=yes

# GRANT ALL PRIVILEGES ON DATABASE library TO librarian
# Note that here "db=postgres" specifies the database to connect to, not the
# database to grant privileges on (which is specified via the "objs" param)
- postgresql_privs: >
    db=postgres
    privs=ALL
    type=database
    obj=library
    role=librarian

# GRANT ALL PRIVILEGES ON DATABASE library TO librarian
# If objs is omitted for type "database", it defaults to the database
# to which the connection is established
- postgresql_privs: >
    db=library
    privs=ALL
    type=database
    role=librarian
```

Note: Default authentication assumes that postgresql_privs is run by the `postgres` user on the remote host. (Ansible's `user` or `sudo-user`).

Note: This module requires Python package *psycopg2* to be installed on the remote host. In the default case of the remote host also being the PostgreSQL server, PostgreSQL has to be installed there as well, obviously. For Debian/Ubuntu-based systems, install packages *postgresql* and *python-psycopg2*.

Note: Parameters that accept comma separated lists (*privs*, *objs*, *roles*) have singular alias names (*priv*, *obj*, *role*).

Note: To revoke only `GRANT OPTION` for a specific object, set *state* to `present` and *grant_option* to `no` (see examples).

Note: Note that when revoking privileges from a role R, this role may still have access via privileges granted to any role R is a member of including `PUBLIC`.

Note: Note that when revoking privileges from a role R, you do so as the user specified via *login*. If R has been granted the same privileges by another user also, R can still access database objects via these privileges.

Note: When revoking privileges, `RESTRICT` is assumed (see PostgreSQL docs).

postgresql_user - Adds or removes a users (roles) from a PostgreSQL database.

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

Add or remove PostgreSQL users (roles) from a remote host and, optionally, grant the users access to an existing database or tables. The fundamental function of the module is to create, or delete, roles from a PostgreSQL cluster. Privilege assignment, or removal, is an optional step, which works on one database at a time. This allows for the module to be called several times in the same module to modify the permissions on different databases, or to grant permissions to already existing users. A user cannot be removed until all the privileges have been stripped from the user. In such situation, if the module tries to remove the user it will fail. To avoid this from happening the fail_on_user option signals the module to try to remove the user, but if not possible keep going; the module will report if changes happened and separately if the user was removed or not.

Options

Note: Requires psycopg2

Examples

```
# Create django user and grant access to database and products table
- postgresql_user: db=acme name=django password=ceec4eif7ya priv=CONNECT/products:ALL

# Create rails user, grant privilege to create other databases and demote rails from super user statu
- postgresql_user: name=rails password=secret role_attr_flags=CREATEDB,NOSUPERUSER

# Remove test user privileges from acme
- postgresql_user: db=acme name=test priv=ALL/products:ALL state=absent fail_on_user=no

# Remove test user from test database and the cluster
- postgresql_user: db=test name=test priv=ALL state=absent

# Example privileges string format
INSERT,UPDATE/table:SELECT/anothertable:ALL

# Remove an existing user's password
- postgresql_user: db=test user=test password=NULL
```

Note: The default authentication assumes that you are either logging in as or sudo'ing to the postgres account on the host.

Note: This module uses psycopg2, a Python PostgreSQL database adapter. You must ensure that psycopg2 is installed on the host before using this module. If the remote host is the PostgreSQL server (which is the default case), then PostgreSQL must also be installed on the remote host. For Ubuntu-based systems, install the postgresql, libpq-dev, and python-psycopg2 packages on the remote host before using this module.

Note: If you specify PUBLIC as the user, then the privilege changes will apply to all users. You may not specify password or role_attr_flags when the PUBLIC user is specified.

quantum_floating_ip - Add/Remove floating IP from an instance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove a floating IP to an instance

Options

Note: Requires novaclient

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Assign a floating ip to the instance from an external network
- quantum_floating_ip: state=present login_username=admin login_password=admin
                       login_tenant_name=admin network_name=external_network
                       instance_name=vm1 internal_network_name=internal_network
```

quantum_floating_ip_associate - Associate or disassociate a particular floating IP with an instance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Associates or disassociates a specific floating IP with a particular instance

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Associate a specific floating IP with an Instance
- quantum_floating_ip_associate:
           state=present
           login_username=admin
           login_password=admin
           login_tenant_name=admin
           ip_address=1.1.1.1
           instance_name=vm1
```

quantum_network - Creates/Removes networks from OpenStack

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or Remove network from OpenStack.

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Create a GRE backed Quantum network with tunnel id 1 for tenant1
- quantum_network: name=t1network tenant_name=tenant1 state=present
                   provider_network_type=gre provider_segmentation_id=1
                   login_username=admin login_password=admin login_tenant_name=admin

# Create an external network
- quantum_network: name=external_network state=present
                   provider_network_type=local router_external=yes
                   login_username=admin login_password=admin login_tenant_name=admin
```

quantum_router - Create or Remove router from openstack

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Create or Delete routers from OpenStack

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Creates a router for tenant admin
- quantum_router: state=present
                  login_username=admin
                  login_password=admin
                  login_tenant_name=admin
                  name=router1"
```

quantum_router_gateway - set/unset a gateway interface for the router with the specified external network

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Creates/Removes a gateway interface from the router, used to associate a external network with a router to route external traffic.

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Attach an external network with a router to allow flow of external traffic
- quantum_router_gateway: state=present login_username=admin login_password=admin
                          login_tenant_name=admin router_name=external_router
                          network_name=external_network
```

quantum_router_interface - Attach/Dettach a subnet's interface to a router

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Attach/Dettach a subnet interface to a router, to provide a gateway for the subnet.

Options

Note: Requires quantumclient

Note: Requires keystoneclient

Examples

```
# Attach tenant1's subnet to the external router
- quantum_router_interface: state=present login_username=admin
                            login_password=admin
                            login_tenant_name=admin
                            tenant_name=tenant1
                            router_name=external_route
                            subnet_name=t1subnet
```

quantum_subnet - Add/Remove floating IP from an instance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove a floating IP to an instance

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Create a subnet for a tenant with the specified subnet
- quantum_subnet: state=present login_username=admin login_password=admin
                  login_tenant_name=admin tenant_name=tenant1
                  network_name=network1 name=net1subnet cidr=192.168.0.0/24"
```

rabbitmq_parameter - Adds or removes parameters to RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage dynamic, cluster-wide parameters for RabbitMQ

Options

Examples

```
# Set the federation parameter 'local_username' to a value of 'guest' (in quotes)
- rabbitmq_parameter: component=federation
                      name=local-username
                      value='"guest"'
                      state=present
```

rabbitmq_plugin - Adds or removes users to RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Enables or disables RabbitMQ plugins

Options

Examples

```
# Enables the rabbitmq_management plugin
- rabbitmq_plugin: names=rabbitmq_management state=enabled
```

rabbitmq_policy - Manage the state of policies in RabbitMQ.

Author John Dewey

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manage the state of a virtual host in RabbitMQ.

Options

Examples

```
- name: ensure the default vhost contains the HA policy via a dict
  rabbitmq_policy: name=HA pattern='.*'
  args:
    tags:
      "ha-mode": all

- name: ensure the default vhost contains the HA policy
  rabbitmq_policy: name=HA pattern='.*' tags="ha-mode=all"
```

rabbitmq_user - Adds or removes users to RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Add or remove users to RabbitMQ and assign permissions

Options

Examples

```
# Add user to server and assign full access control
- rabbitmq_user: user=joe
                 password=changeme
                 vhost=/
                 configure_priv=.*
                 read_priv=.*
                 write_priv=.*
                 state=present
```

rabbitmq_vhost - Manage the state of a virtual host in RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage the state of a virtual host in RabbitMQ

Options

Examples

```
# Ensure that the vhost /test exists.
- rabbitmq_vhost: name=/test state=present
```

raw - Executes a low-down and dirty SSH command

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

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. Standard output, error output and return code are returned when available. There is no change handler support for this module. This module does not require python on the remote system, much like the `script` module.

Options

Examples

```
# Bootstrap a legacy python 2.4 host
- raw: yum -y install python-simplejson
```

Note: If you want to execute a command securely and predictably, it may be better to use the `command` module instead. Best practices when writing playbooks will follow the trend of using `command` unless `shell` is explicitly required. When running ad-hoc commands, use your best judgement.

---

rax - create / delete an instance in Rackspace Public Cloud

Author Jesse Keating, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

creates / deletes a Rackspace Public Cloud instance and optionally waits for it to be 'running'.

Options

Note: Requires pyrax

Examples

```
- name: Build a Cloud Server
  gather_facts: False
  tasks:
    - name: Server build request
      local_action:
        module: rax
        credentials: ~/.raxpub
        name: rax-test1
        flavor: 5
        image: b11d9567-e412-4255-96b9-bd63ab23bcfe
```

```
        files:
          /root/.ssh/authorized_keys: /home/localuser/.ssh/id_rsa.pub
          /root/test.txt: /home/localuser/test.txt
        wait: yes
        state: present
        networks:
          - private
          - public
      register: rax

- name: Build an exact count of cloud servers with incremented names
  hosts: local
  gather_facts: False
  tasks:
    - name: Server build requests
      local_action:
        module: rax
        credentials: ~/.raxpub
        name: test%03d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        state: present
        count: 10
        count_offset: 10
        exact_count: yes
        group: test
        wait: yes
      register: rax
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_clb - create / delete a load balancer in Rackspace Public Cloud

Author Christopher H. Laco, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

creates / deletes a Rackspace Public Cloud load balancer.

Options

Note: Requires pyrax

Examples

```
- name: Build a Load Balancer
  gather_facts: False
  hosts: local
  connection: local
  tasks:
    - name: Load Balancer create request
      local_action:
        module: rax_clb
        credentials: ~/.raxpub
        name: my-lb
        port: 8080
        protocol: HTTP
        type: SERVICENET
        timeout: 30
        region: DFW
        wait: yes
        state: present
        meta:
          app: my-cool-app
      register: my_lb
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_clb_nodes - add, modify and remove nodes from a Rackspace Cloud Load Balancer

Author Lukasz Kawczynski

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Adds, modifies and removes nodes from a Rackspace Cloud Load Balancer

Options

Note: Requires pyrax

Examples

```
# Add a new node to the load balancer
- local_action:
    module: rax_clb_nodes
    load_balancer_id: 71
    address: 10.2.2.3
    port: 80
    condition: enabled
    type: primary
    wait: yes
    credentials: /path/to/credentials

# Drain connections from a node
- local_action:
    module: rax_clb_nodes
    load_balancer_id: 71
    node_id: 410
    condition: draining
    wait: yes
    credentials: /path/to/credentials

# Remove a node from the load balancer
- local_action:
    module: rax_clb_nodes
    load_balancer_id: 71
    node_id: 410
    state: absent
    wait: yes
    credentials: /path/to/credentials
```

Note: The following environment variables can be used: `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDENTIALS` and `RAX_REGION`.

rax_dns - Manage domains on Rackspace Cloud DNS

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manage domains on Rackspace Cloud DNS

Options

Note: Requires pyrax

Examples

```
- name: Create domain
  hosts: all
  gather_facts: False
  tasks:
    - name: Domain create request
      local_action:
        module: rax_dns
        credentials: ~/.raxpub
        name: example.org
        email: admin@example.org
      register: rax_dns
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_dns_record - Manage DNS records on Rackspace Cloud DNS

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manage DNS records on Rackspace Cloud DNS

Options

Note: Requires pyrax

Examples

```
- name: Create record
  hosts: all
  gather_facts: False
  tasks:
    - name: Record create request
      local_action:
        module: rax_dns_record
        credentials: ~/.raxpub
        domain: example.org
        name: www.example.org
        data: 127.0.0.1
        type: A
      register: rax_dns_record
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_facts - Gather facts for Rackspace Cloud Servers

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Gather facts for Rackspace Cloud Servers.

Options

Note: Requires pyrax

Examples

```
- name: Gather info about servers
  hosts: all
  gather_facts: False
  tasks:
    - name: Get facts about servers
      local_action:
        module: rax_facts
        credentials: ~/.raxpub
        name: "{{ inventory_hostname }}"
        region: DFW
    - name: Map some facts
      set_fact:
        ansible_ssh_host: "{{ rax_accessipv4 }}"
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_files - Manipulate Rackspace Cloud Files Containers

Author Paul Durivage

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manipulate Rackspace Cloud Files Containers

Options

Note: Requires pyrax

Examples

```
- name: "Test Cloud Files Containers"
  hosts: local
  gather_facts: no
  tasks:
    - name: "List all containers"
      rax_files: state=list

    - name: "Create container called 'mycontainer'"
      rax_files: container=mycontainer

    - name: "Create container 'mycontainer2' with metadata"
      rax_files:
        container: mycontainer2
        meta:
          key: value
          file_for: someuser@example.com

    - name: "Set a container's web index page"
      rax_files: container=mycontainer web_index=index.html

    - name: "Set a container's web error page"
      rax_files: container=mycontainer web_error=error.html

    - name: "Make container public"
      rax_files: container=mycontainer public=yes

    - name: "Make container public with a 24 hour TTL"
      rax_files: container=mycontainer public=yes ttl=86400

    - name: "Make container private"
      rax_files: container=mycontainer private=yes

- name: "Test Cloud Files Containers Metadata Storage"
  hosts: local
  gather_facts: no
  tasks:
    - name: "Get mycontainer2 metadata"
      rax_files:
        container: mycontainer2
        type: meta

    - name: "Set mycontainer2 metadata"
```

```
    rax_files:
      container: mycontainer2
      type: meta
      meta:
        uploaded_by: someuser@example.com

- name: "Remove mycontainer2 metadata"
  rax_files:
    container: "mycontainer2"
    type: meta
    state: absent
    meta:
      key: ""
      file_for: ""
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_files_objects - Upload, download, and delete objects in Rackspace Cloud Files

Author Paul Durivage

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Upload, download, and delete objects in Rackspace Cloud Files

Options

Note: Requires pyrax

Examples

```
- name: "Test Cloud Files Objects"
  hosts: local
  gather_facts: False
  tasks:
    - name: "Get objects from test container"
      rax_files_objects: container=testcont dest=~/Downloads/testcont

    - name: "Get single object from test container"
      rax_files_objects: container=testcont src=file1 dest=~/Downloads/testcont

    - name: "Get several objects from test container"
      rax_files_objects: container=testcont src=file1,file2,file3 dest=~/Downloads/testcont

    - name: "Delete one object in test container"
      rax_files_objects: container=testcont method=delete dest=file1

    - name: "Delete several objects in test container"
      rax_files_objects: container=testcont method=delete dest=file2,file3,file4

    - name: "Delete all objects in test container"
      rax_files_objects: container=testcont method=delete

    - name: "Upload all files to test container"
      rax_files_objects: container=testcont method=put src=~/Downloads/onehundred

    - name: "Upload one file to test container"
      rax_files_objects: container=testcont method=put src=~/Downloads/testcont/file1

    - name: "Upload one file to test container with metadata"
      rax_files_objects:
        container: testcont
        src: ~/Downloads/testcont/file2
        method: put
        meta:
          testkey: testdata
          who_uploaded_this: someuser@example.com

    - name: "Upload one file to test container with TTL of 60 seconds"
      rax_files_objects: container=testcont method=put src=~/Downloads/testcont/file3 expires=60

    - name: "Attempt to get remote object that does not exist"
      rax_files_objects: container=testcont method=get src=FileThatDoesNotExist.jpg dest=~/Downloads/
      ignore_errors: yes

    - name: "Attempt to delete remote object that does not exist"
      rax_files_objects: container=testcont method=delete dest=FileThatDoesNotExist.jpg
      ignore_errors: yes

- name: "Test Cloud Files Objects Metadata"
  hosts: local
  gather_facts: false
  tasks:
    - name: "Get metadata on one object"
      rax_files_objects:  container=testcont type=meta dest=file2

    - name: "Get metadata on several objects"
```

```
    rax_files_objects:  container=testcont type=meta src=file2,file1

  - name: "Set metadata on an object"
    rax_files_objects:
      container: testcont
      type: meta
      dest: file17
      method: put
      meta:
        key1: value1
        key2: value2
      clear_meta: true

  - name: "Verify metadata is set"
    rax_files_objects:  container=testcont type=meta src=file17

  - name: "Delete metadata"
    rax_files_objects:
      container: testcont
      type: meta
      dest: file17
      method: delete
      meta:
        key1: ''
        key2: ''

  - name: "Get metadata on all objects"
    rax_files_objects:  container=testcont type=meta
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_keypair - Create a keypair for use with Rackspace Cloud Servers

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Create a keypair for use with Rackspace Cloud Servers

Options

Note: Requires pyrax

Examples

```
- name: Create a keypair
  hosts: local
  gather_facts: False
  tasks:
    - name: keypair request
      local_action:
        module: rax_keypair
        credentials: ~/.raxpub
        name: my_keypair
        region: DFW
      register: keypair
    - name: Create local public key
      local_action:
        module: copy
        content: "{{ keypair.keypair.public_key }}"
        dest: "{{ inventory_dir }}/{{ keypair.keypair.name }}.pub"
    - name: Create local private key
      local_action:
        module: copy
        content: "{{ keypair.keypair.private_key }}"
        dest: "{{ inventory_dir }}/{{ keypair.keypair.name }}"
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

Note: Keypairs cannot be manipulated, only created and deleted. To "update" a keypair you must first delete and then recreate.

rax_network - create / delete an isolated network in Rackspace Public Cloud

Author Christopher H. Laco, Jesse Keating

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

creates / deletes a Rackspace Public Cloud isolated network.

Options

Note: Requires pyrax

Examples

```
- name: Build an Isolated Network
  gather_facts: False

  tasks:
    - name: Network create request
      local_action:
        module: rax_network
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        state: present
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS` points to a credentials file appropriate for pyrax

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_queue - create / delete a queue in Rackspace Public Cloud

Author Christopher H. Laco, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

creates / deletes a Rackspace Public Cloud queue.

Options

Note: Requires pyrax

Examples

```
- name: Build a Queue
  gather_facts: False
  hosts: local
  connection: local
  tasks:
    - name: Queue create request
      local_action:
        module: rax_queue
        credentials: ~/.raxpub
        client_id: unique-client-name
        name: my-queue
        region: DFW
        state: present
      register: my_queue
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rds - create, delete, or modify an Amazon rds instance

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates, deletes, or modifies rds instances. When creating an instance it can be either a new instance or a read-only replica of an existing instance. This module has a dependency on python-boto >= 2.5. The 'promote' command requires boto >= 2.18.0.

Options

Note: Requires boto

Examples

```
# Basic mysql provisioning example
- rds: >
      command=create
      instance_name=new_database
      db_engine=MySQL
      size=10
      instance_type=db.m1.small
      username=mysql_admin
      password=1nsecure

# Create a read-only replica and wait for it to become available
- rds: >
      command=replicate
      instance_name=new_database_replica
      source_instance=new_database
      wait=yes
      wait_timeout=600

# Delete an instance, but create a snapshot before doing so
- rds: >
      command=delete
      instance_name=new_database
      snapshot=new_database_snapshot

# Get facts about an instance
- rds: >
      command=facts
      instance_name=new_database
      register: new_database_facts

# Rename an instance and wait for the change to take effect
- rds: >
      command=modify
      instance_name=new_database
      new_instance_name=renamed_database
      wait=yes
```

redhat_subscription - Manage Red Hat Network registration and subscriptions using the `subscription-manager` command

Author James Laska

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage registration and subscription to the Red Hat Network entitlement platform.

Options

Note: Requires subscription-manager

Examples

```
# Register as user (joe_user) with password (somepass) and auto-subscribe to available content.
- redhat_subscription: action=register username=joe_user password=somepass autosubscribe=true

# Register with activationkey (1-222333444) and consume subscriptions matching
# the names (Red hat Enterprise Server) and (Red Hat Virtualization)
- redhat_subscription: action=register
                       activationkey=1-222333444
                       pool='^(Red Hat Enterprise Server|Red Hat Virtualization)$'
```

Note: In order to register a system, subscription-manager requires either a username and password, or an activation-key.

redis - Various redis commands, slave and flush

Author Xabier Larrakoetxea

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Unified utility to interact with redis instances. 'slave' Sets a redis instance in slave or master mode. 'flush' Flushes all the instance or a specified db.

Options

Note: Requires redis

Examples

```
# Set local redis instance to be slave of melee.island on port 6377
- redis: command=slave master_host=melee.island master_port=6377

# Deactivate slave mode
- redis: command=slave slave_mode=master

# Flush all the redis db
- redis: command=flush flush_mode=all

# Flush only one db in a redis instance
- redis: command=flush db=1 flush_mode=db
```

Note: Requires the redis-py Python package on the remote host. You can install it with pip (pip install redis) or with a package manager. https://github.com/andymccurdy/redis-py

Note: If the redis master instance we are making slave of is password protected this needs to be in the redis.conf in the masterauth variable

rhn_channel - Adds or removes Red Hat software channels

Author Vincent Van der Kussen

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Adds or removes Red Hat software channels

Options

Note: Requires none

Examples

```
- rhn_channel: name=rhel-x86_64-server-v2vwin-6 sysname=server01 url=https://rhn.redhat.com/rpc/api u
```

Note: this module fetches the system id from RHN.

rhn_register - Manage Red Hat Network registration using the `rhnreg_ks` command

Author James Laska

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage registration to the Red Hat Network.

Options

Note: Requires rhnreg_ks

Examples

```
# Unregister system from RHN.
- rhn_register: state=absent username=joe_user password=somepass

# Register as user (joe_user) with password (somepass) and auto-subscribe to available content.
- rhn_register: state=present username=joe_user password=somepass

# Register with activationkey (1-222333444) and enable extended update support.
- rhn_register: state=present activationkey=1-222333444 enable_eus=true

# Register as user (joe_user) with password (somepass) against a satellite
# server specified by (server_url).
- rhn_register:
    state=present
    username=joe_user
    password=somepass
    server_url=https://xmlrpc.my.satellite/XMLRPC

# Register as user (joe_user) with password (somepass) and enable
# channels (rhel-x86_64-server-6-foo-1) and (rhel-x86_64-server-6-bar-1).
- rhn_register: state=present username=joe_user
                password=somepass
                channels=rhel-x86_64-server-6-foo-1,rhel-x86_64-server-6-bar-1
```

Note: In order to register a system, rhnreg_ks requires either a username and password, or an activationkey.

riak - This module handles some common Riak operations

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module can be used to join nodes to a cluster, check the status of the cluster.

Options

Examples

```
# Join's a Riak node to another node
- riak: command=join target_node=riak@10.1.1.1

# Wait for handoffs to finish.  Use with async and poll.
- riak: wait_for_handoffs=yes

# Wait for riak_kv service to startup
- riak: wait_for_service=kv
```

route53 - add or delete entries in Amazons Route53 DNS service

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates and deletes DNS records in Amazons Route53 service

Options

Note: Requires boto

Examples

```
# Add new.foo.com as an A record with 3 IPs
- route53: >
      command=create
      zone=foo.com
      record=new.foo.com
      type=A
      ttl=7200
      value=1.1.1.1,2.2.2.2,3.3.3.3

# Retrieve the details for new.foo.com
- route53: >
      command=get
      zone=foo.com
      record=new.foo.com
      type=A
  register: rec

# Delete new.foo.com A record using the results from the get command
- route53: >
      command=delete
      zone=foo.com
      record={{ rec.set.record }}
      type={{ rec.set.type }}
      value={{ rec.set.value }}

# Add an AAAA record.  Note that because there are colons in the value
# that the entire parameter list must be quoted:
- route53: >
      command=create
      zone=foo.com
      record=localhost.foo.com
      type=AAAA
      ttl=7200
      value="::1"
```

rpm_key - Adds or removes a gpg key from the rpm db

Author Hector Acosta <hector.acosta@gazzang.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Adds or removes (rpm –import) a gpg key to your rpm database.

Options

Examples

```
# Example action to import a key from a url
- rpm_key: state=present key=http://apt.sw.be/RPM-GPG-KEY.dag.txt

# Example action to import a key from a file
- rpm_key: state=present key=/path/to/key.gpg

# Example action to ensure a key is not present in the db
- rpm_key: state=absent key=DEADB33F
```

s3 - idempotent S3 module putting a file into S3.

Author Lester Wade, Ralph Tice

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

This module allows the user to dictate the presence of a given file in an S3 bucket. If or once the key (file) exists in the bucket, it returns a time-expired download URL. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Simple PUT operation
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=put
# Simple GET operation
- s3: bucket=mybucket object=/my/desired/key.txt dest=/usr/local/myfile.txt mode=get
# GET/download and overwrite local file (trust remote)
- s3: bucket=mybucket object=/my/desired/key.txt dest=/usr/local/myfile.txt mode=get
# GET/download and do not overwrite local file (trust remote)
- s3: bucket=mybucket object=/my/desired/key.txt dest=/usr/local/myfile.txt mode=get force=false
# PUT/upload and overwrite remote file (trust local)
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=put
# PUT/upload and do not overwrite remote file (trust local)
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=put force=false
# Download an object as a string to use else where in your playbook
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=getstr
# Create an empty bucket
```

```
- s3: bucket=mybucket mode=create
# Create a bucket with key as directory
- s3: bucket=mybucket object=/my/directory/path mode=create
# Delete a bucket and all contents
- s3: bucket=mybucket mode=delete
```

script - Runs a local script on a remote node after transferring it

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `script` module takes the script name followed by a list of space-delimited arguments. The local script at path will be transfered to the remote node and then executed. The given script will be processed through the shell environment on the remote node. This module does not require python on the remote system, much like the `raw` module.

Options

Examples

```
# Example from Ansible Playbooks
- script: /some/local/script.sh --some-arguments 1234
```

Note: It is usually preferable to write Ansible modules than pushing scripts. Convert your script to an Ansible module for bonus points!

seboolean - Toggles SELinux booleans.

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Toggles SELinux booleans.

Options

Examples

```
# Set (httpd_can_network_connect) flag on and keep it persistent across reboots
- seboolean: name=httpd_can_network_connect state=yes persistent=yes
```

Note: Not tested on any debian based system

selinux - Change policy and state of SELinux

Author Derek Carter <goozbach@friocorte.com>

- Synopsis
- Options
- Examples

Synopsis

Configures the SELinux mode and policy. A reboot may be required after usage. Ansible will not issue this reboot but will let you know when it is required.

Options

Note: Requires libselinux-python

Examples

```
- selinux: policy=targeted state=enforcing
- selinux: policy=targeted state=permissive
- selinux: state=disabled
```

Note: Not tested on any debian based system

service - Manage services.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

Controls services on remote hosts.

Options

Examples

```
# 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

# Example action to enable service httpd, and not touch the running state
- service: name=httpd enabled=yes

# Example action to start service foo, based on running process /usr/bin/foo
- service: name=foo pattern=/usr/bin/foo state=started

# Example action to restart network service for interface eth0
- service: name=network state=restarted args=eth0
```

set_fact - Set host facts from a task

Author Dag Wieers

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module allows setting new variables. Variables are set on a host-by-host basis just like facts discovered by the setup module. These variables will survive between plays.

Options

Examples

```
# Example setting host facts using key=value pairs
- set_fact: one_fact="something" other_fact="{{ local_var * 2 }}"

# Example setting host facts using complex arguments
- set_fact:
     one_fact: something
     other_fact: "{{ local_var * 2 }}"
```

setup - Gathers facts about remote hosts

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

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.

Options

Examples

```
# Display facts from all hosts and store them indexed by I(hostname) at C(/tmp/facts).
ansible all -m setup --tree /tmp/facts

# Display only facts regarding memory found by ansible on all hosts and output them.
ansible all -m setup -a 'filter=ansible_*_mb'

# Display only facts returned by facter.
ansible all -m setup -a 'filter=facter_*'

# Display only facts about certain interfaces.
ansible all -m setup -a 'filter=ansible_eth[0-2]'
```

Note: 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. (See also `facter` and `ohai`.)

Note: The filter option filters only the first level subkey below ansible_facts.

---

shell - Execute commands in nodes.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `shell` module takes the command name followed by a list of space-delimited arguments. It is almost exactly like the `command` module but runs the command through a shell (`/bin/sh`) on the remote node.

Options

Examples

```
# Execute the command in remote shell; stdout goes to the specified
# file on the remote
- shell: somescript.sh >> somelog.txt
```

Note: If you want to execute a command securely and predictably, it may be better to use the `command` module instead. Best practices when writing playbooks will follow the trend of using `command` unless `shell` is explicitly required. When running ad-hoc commands, use your best judgement.

slurp - Slurps a file from remote nodes

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This module works like `fetch`. It is used for fetching a base64- encoded blob containing the data in a remote file.

Options

Examples

```
ansible host -m slurp -a 'src=/tmp/xx'
   host | success >> {
      "content": "aGVsbG8gQW5zaWJsZSB3b3JsZAo=",
```

```
    "encoding": "base64"
}
```

Note: See also: `fetch`

stat - retrieve file or file system status

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Retrieves facts for a file similar to the linux/unix 'stat' command.

Options

Examples

```
# Obtain the stats of /etc/foo.conf, and check that the file still belongs
# to 'root'. Fail otherwise.
- stat: path=/etc/foo.conf
  register: st
- fail: msg="Whoops! file ownership has changed"
  when: st.stat.pw_name != 'root'

# Determine if a path exists and is a directory.  Note we need to test
# both that p.stat.isdir actually exists, and also that it's set to true.
- stat: path=/path/to/something
  register: p
- debug: msg="Path exists and is a directory"
  when: p.stat.isdir is defined and p.stat.isdir == true
```

subversion - Deploys a subversion repository.

Author Dane Summers, njharman@gmail.com

- Synopsis
- Options
- Examples

Synopsis

Deploy given repository URL / revision to dest. If dest exists, update to the specified revision, otherwise perform a checkout.

Options

Examples

```
# Checkout subversion repository to specified folder.
- subversion: repo=svn+ssh://an.example.org/path/to/repo dest=/src/checkout
```

Note: Requres *svn* to be installed on the client.

supervisorctl - Manage the state of a program or group of programs running via Supervisord

Author Matt Wright

- Synopsis
- Options
- Examples

Synopsis

Manage the state of a program or group of programs running via *Supervisord*

Options

Examples

```
# Manage the state of program to be in 'started' state.
- supervisorctl: name=my_app state=started

# Restart my_app, reading supervisorctl configuration from a specified file.
- supervisorctl: name=my_app state=restarted config=/var/opt/my_project/supervisord.conf

# Restart my_app, connecting to supervisord with credentials and server URL.
- supervisorctl: name=my_app state=restarted username=test password=testpass server_url=http://localh
```

svr4pkg - Manage Solaris SVR4 packages

Author Boyd Adamson

- Synopsis
- Options
- Examples

Synopsis

Manages SVR4 packages on Solaris 10 and 11. These were the native packages on Solaris <= 10 and are available as a legacy feature in Solaris 11. Note that this is a very basic packaging system. It will not enforce dependencies on install or remove.

Options

Examples

```
# Install a package from an already copied file
- svr4pkg: name=CSWcommon src=/tmp/cswpkgs.pkg state=present

# Install a package directly from an http site
- svr4pkg: name=CSWpkgutil src=http://get.opencsw.org/now state=present

# Install a package with a response file
- svr4pkg: name=CSWggrep src=/tmp/third-party.pkg response_file=/tmp/ggrep.response state=present

# Ensure that a package is not installed.
- svr4pkg: name=SUNWgnome-sound-recorder state=absent
```

swdepot - Manage packages with swdepot package manager (HP-UX)

Author Raul Melo

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Will install, upgrade and remove packages with swdepot package manager (HP-UX)

Options

Examples

```
- swdepot: name=unzip-6.0 state=installed depot=repository:/path
- swdepot: name=unzip state=latest depot=repository:/path
- swdepot: name=unzip state=absent
```

synchronize - Uses rsync to make synchronizing file paths in your playbooks quick and easy.

Author Timothy Appnel

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This is a wrapper around rsync. Of course you could just use the command action to call rsync yourself, but you also have to add a fair number of boilerplate options and host facts. You still may need to call rsync directly via `command` or `shell` depending on your use case. The synchronize action is meant to do common things with `rsync` easily. It does not provide access to the full power of rsync, but does make most invocations easier to follow.

Options

Examples

```
# Synchronization of src on the control machine to dest on the remote hosts
synchronize: src=some/relative/path dest=/some/absolute/path

# Synchronization without any --archive options enabled
synchronize: src=some/relative/path dest=/some/absolute/path archive=no

# Synchronization with --archive options enabled except for --recursive
synchronize: src=some/relative/path dest=/some/absolute/path recursive=no

# Synchronization without --archive options enabled except use --links
synchronize: src=some/relative/path dest=/some/absolute/path archive=no links=yes

# Synchronization of two paths both on the control machine
local_action: synchronize src=some/relative/path dest=/some/absolute/path

# Synchronization of src on the inventory host to the dest on the localhost in
pull mode
synchronize: mode=pull src=some/relative/path dest=/some/absolute/path

# Synchronization of src on delegate host to dest on the current inventory host
synchronize: >
    src=some/relative/path dest=/some/absolute/path
    delegate_to: delegate.host

# Synchronize and delete files in dest on the remote host that are not found in src of localhost.
synchronize: src=some/relative/path dest=/some/absolute/path delete=yes

# Synchronize using an alternate rsync command
synchronize: src=some/relative/path dest=/some/absolute/path rsync_path="sudo rsync"

# Example .rsync-filter file in the source directory
- var       # exclude any path whose last part is 'var'
- /var      # exclude any path starting with 'var' starting at the source directory
+ /var/conf # include /var/conf even though it was previously excluded
```

Note: Inspect the verbose output to validate the destination user/host/path are what was expected.

Note: The remote user for the dest path will always be the remote_user, not the sudo_user.

Note: Expect that dest=~/x will be ~<remote_user>/x even if using sudo.

Note: To exclude files and directories from being synchronized, you may add `.rsync-filter` files to the source directory.

sysctl - Manage entries in sysctl.conf.

Author David "DaviXX" CHANIAL <david.chanial@gmail.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

This module manipulates sysctl entries and optionally performs a `/sbin/sysctl -p` after changing them.

Options

Examples

```
# Set vm.swappiness to 5 in /etc/sysctl.conf
- sysctl: name=vm.swappiness value=5 state=present

# Remove kernel.panic entry from /etc/sysctl.conf
- sysctl: name=kernel.panic state=absent sysctl_file=/etc/sysctl.conf

# Set kernel.panic to 3 in /tmp/test_sysctl.conf
- sysctl: name=kernel.panic value=3 sysctl_file=/tmp/test_sysctl.conf reload=no

# Set ip fowarding on in /proc and do not reload the sysctl file
- sysctl: name="net.ipv4.ip_forward" value=1 sysctl_set=yes

# Set ip forwarding on in /proc and in the sysctl file and reload if necessary
- sysctl: name="net.ipv4.ip_forward" value=1 sysctl_set=yes state=present reload=yes
```

template - Templates a file out to a remote server.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

Templates are processed by the Jinja2 templating language (http://jinja.pocoo.org/docs/) - documentation on the template formatting can be found in the Template Designer Documentation (http://jinja.pocoo.org/docs/templates/). Six additional variables can be used in templates: `ansible_managed` (configurable via the `defaults` section of `ansible.cfg`) contains a string which can be used to describe the template name, host, modification time of the template file and the owner uid, `template_host` contains the node name of the template's machine, `template_uid` the owner, `template_path` the absolute path of the template, `template_fullpath` is the absolute path of the template, and `template_run_date` is the date that the template was rendered. Note that including a string that uses a date in the template will resort in the template being marked 'changed' each time.

Options

Examples

```
# Example from Ansible Playbooks
- template: src=/mytemplates/foo.j2 dest=/etc/file.conf owner=bin group=wheel mode=0644

# Copy a new "sudoers file into place, after passing validation with visudo
- action: template src=/mine/sudoers dest=/etc/sudoers validate='visudo -cf %s'
```

Note: Since Ansible version 0.9, templates are loaded with `trim_blocks=True`.

Note: Also, you can override jinja2 settings by adding a special header to template file. i.e. `#jinja2:variable_start_string:'[%' , variable_end_string:'%]'` which changes the variable interpolation markers to [% var %] instead of {{ var }}. This is the best way to prevent evaluation of things that look like, but should not be Jinja2. raw/endraw in Jinja2 will not work as you expect because templates in Ansible are recursively evaluated.

---

unarchive - Copies an archive to a remote location and unpack it

Author Dylan Martin

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

The `unarchive` module copies an archive file from the local machine to a remote and unpacks it.

Options

Examples

```
# Example from Ansible Playbooks
- unarchive: src=foo.tgz dest=/var/lib/foo
```

Note: requires `tar`/`unzip` command on target host

Note: can handle *gzip*, *bzip2* and *xz* compressed as well as uncompressed tar files

Note: detects type of archive automatically

Note: uses tar's `--diff arg` to calculate if changed or not. If this `arg` is not supported, it will always unpack the archive

Note: does not detect if a .zip file is different from destination - always unzips

Note: existing files/directories in the destination which are not in the archive are not touched. This is the same behavior as a normal archive extraction

Note: existing files/directories in the destination which are not in the archive are ignored for purposes of deciding if the archive should be unpacked or not

uri - Interacts with webservices

Author Romeo Theriault

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Interacts with HTTP and HTTPS web services and supports Digest, Basic and WSSE HTTP authentication mechanisms.

Options

Note: Requires urlparse

Note: Requires httplib2

Examples

```
# Check that you can connect (GET) to a page and it returns a status 200
- uri: url=http://www.example.com

# Check that a page returns a status 200 and fail if the word AWESOME is not in the page contents.
- action: uri url=http://www.example.com return_content=yes
  register: webpage

- action: fail
  when: 'AWESOME' not in "{{ webpage.content }}"


# Create a JIRA issue.
- action: >
        uri url=https://your.jira.example.com/rest/api/2/issue/
        method=POST user=your_username password=your_pass
        body="{{ lookup('file','issue.json') }}" force_basic_auth=yes
        status_code=201 HEADER_Content-Type="application/json"

- action: >
        uri url=https://your.form.based.auth.examle.com/index.php
        method=POST body="name=your_username&password=your_password&enter=Sign%20in"
        status_code=302 HEADER_Content-Type="application/x-www-form-urlencoded"
  register: login

# Login to a form based webpage, then use the returned cookie to
# access the app in later tasks.
- action: uri url=https://your.form.based.auth.example.com/dashboard.php
            method=GET return_content=yes HEADER_Cookie="{{login.set_cookie}}"
```

urpmi - Urpmi manager

Author Philippe Makowski

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.4.

Manages packages with *urpmi* (such as for Mageia or Mandriva)

Options

Examples

```
# install package foo
- urpmi: pkg=foo state=present
# remove package foo
- urpmi: pkg=foo state=absent
# description: remove packages foo and bar
- urpmi: pkg=foo,bar state=absent
# description: update the package database (urpmi.update -a -q) and install bar (bar will be the upda
- urpmi: name=bar, state=present, update_cache=yes
```

user - Manage user accounts

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Manage user accounts and user attributes.

Options

Note: Requires useradd

Note: Requires userdel

Note: Requires usermod

Examples

```
# Add the user 'johnd' with a specific uid and a primary group of 'admin'
- user: name=johnd comment="John Doe" uid=1040

# Remove the user 'johnd'
- user: name=johnd state=absent remove=yes

# Create a 2048-bit SSH key for user jsmith
- user: name=jsmith generate_ssh_key=yes ssh_key_bits=2048
```

virt - Manages virtual machines supported by libvirt

Author Michael DeHaan, Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

Manages virtual machines supported by *libvirt*.

Options

Note: Requires libvirt

Examples

```
# a playbook task line:
- virt: name=alpha state=running

# /usr/bin/ansible invocations
ansible host -m virt -a "name=alpha command=status"
ansible host -m virt -a "name=alpha command=get_xml"
ansible host -m virt -a "name=alpha command=create uri=lxc:///"

# a playbook example of defining and launching an LXC guest
tasks:
  - name: define vm
    virt: name=foo
          command=define
          xml="{{ lookup('template', 'container-template.xml.j2') }}"
          uri=lxc:///
  - name: start vm
    virt: name=foo state=running uri=lxc:///
```

wait_for - Waits for a condition before continuing.

Author Jeroen Hoekx, John Jarvis

- Synopsis
- Options
- Examples

Synopsis

Waiting for a port to become available is useful for when services are not immediately available after their init scripts return - which is true of certain Java application servers. It is also useful when starting guests with the `virt` module and needing to pause until they are ready. This module can also be used to wait for a file to be available on the filesystem or with a regex match a string to be present in a file.

Options

Examples

```
# wait 300 seconds for port 8000 to become open on the host, don't start checking for 10 seconds
- wait_for: port=8000 delay=10

# wait until the file /tmp/foo is present before continuing
- wait_for: path=/tmp/foo

# wait until the string "completed" is in the file /tmp/foo before continuing
- wait_for: path=/tmp/foo search_regex=completed
```

xattr - set/retrieve extended attributes

Author Brian Coca

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages filesystem user defined extended attributes, requires that they are enabled on the target filesystem and that the setfattr/getfattr utilities are present.

Options

Examples

```
# Obtain the extended attributes  of /etc/foo.conf
- xattr: name=/etc/foo.conf

# Sets the key 'foo' to value 'bar'
- xattr: path=/etc/foo.conf key=user.foo value=bar

# Removes the key 'foo'
- xattr: name=/etc/foo.conf key=user.foo state=absent
```

yum - Manages packages with the *yum* package manager

Author Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

Installs, upgrade, removes, and lists packages and groups with the *yum* package manager.

Options

Note: Requires yum

Note: Requires rpm

Examples

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

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

- name: install the latest version of Apche from the testing repo
  yum: name=httpd enablerepo=testing state=installed

- name: upgrade all packages
  yum: name=* state=latest

- name: install the nginx rpm from a remote repo
  yum: name=http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-centos-6-0.el6.ngx.noarch.rp

- name: install nginx rpm from a local file
  yum: name=/usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch.rpm state=present

- name: install the 'Development tools' package group
  yum: name="@Development tools" state=present
```

zfs - Manage zfs

Author Johan Wiren

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages ZFS file systems on Solaris and FreeBSD. Can manage file systems, volumes and snapshots. See zfs(1M) for more information about the properties.

Options

Examples

```
# Create a new file system called myfs in pool rpool
- zfs: name=rpool/myfs state=present

# Create a new volume called myvol in pool rpool.
- zfs: name=rpool/myvol state=present volsize=10M

# Create a snapshot of rpool/myfs file system.
- zfs: name=rpool/myfs@mysnapshot state=present

# Create a new file system called myfs2 with snapdir enabled
- zfs: name=rpool/myfs2 state=present snapdir=enabled
```

zypper - Manage packages on SuSE and openSuSE

Author Patrick Callahan

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage packages on SuSE and openSuSE using the zypper and rpm tools.

Options

Note: Requires zypper

Note: Requires rpm

Examples

```
# Install "nmap"
- zypper: name=nmap state=present

# Remove the "nmap" package
- zypper: name=nmap state=absent
```

zypper_repository - Add and remove Zypper repositories

Author Matthias Vogelgesang

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or remove Zypper repositories on SUSE and openSUSE

Options

Note: Requires zypper

Examples

```
# Add NVIDIA repository for graphics drivers
- zypper_repository: name=nvidia-repo repo='ftp://download.nvidia.com/opensuse/12.2' state=present

# Remove NVIDIA repository
- zypper_repository: name=nvidia-repo repo='ftp://download.nvidia.com/opensuse/12.2' state=absent
```

1.6.2 Cloud Modules

cloudformation - create a AWS CloudFormation stack

Author James S. Martin

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Launches an AWS CloudFormation stack and waits for it complete.

Options

Note: Requires boto

Examples

```
# Basic task example
tasks:
- name: launch ansible cloudformation example
  action: cloudformation >
    stack_name="ansible-cloudformation" state=present
    region=us-east-1 disable_rollback=yes
    template=files/cloudformation-example.json
  args:
    template_parameters:
      KeyName: jmartin
      DiskType: ephemeral
      InstanceType: m1.small
      ClusterSize: 3
    tags:
      Stack: ansible-cloudformation
```

digital_ocean - Create/delete a droplet/SSH_key in DigitalOcean

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Create/delete a droplet in DigitalOcean and optionally waits for it to be 'running', or deploy an SSH key.

Options

Note: Requires dopy

Examples

```
# Ensure a SSH key is present
# If a key matches this name, will return the ssh key id and changed = False
# If no existing key matches this name, a new key is created, the ssh key id is returned and changed

- digital_ocean: >
      state=present
      command=ssh
      name=my_ssh_key
      ssh_pub_key='ssh-rsa AAAA...'
      client_id=XXX
      api_key=XXX

# Create a new Droplet
# Will return the droplet details including the droplet id (used for idempotence)

- digital_ocean: >
      state=present
      command=droplet
      name=mydroplet
      client_id=XXX
      api_key=XXX
      size_id=1
      region_id=2
      image_id=3
      wait_timeout=500
  register: my_droplet
- debug: msg="ID is {{ my_droplet.droplet.id }}"
- debug: msg="IP is {{ my_droplet.droplet.ip_address }}"

# Ensure a droplet is present
# If droplet id already exist, will return the droplet details and changed = False
# If no droplet matches the id, a new droplet will be created and the droplet details (including the

- digital_ocean: >
      state=present
      command=droplet
      id=123
      name=mydroplet
      client_id=XXX
      api_key=XXX
      size_id=1
      region_id=2
      image_id=3
      wait_timeout=500

# Create a droplet with ssh key
# The ssh key id can be passed as argument at the creation of a droplet (see ssh_key_ids).
# Several keys can be added to ssh_key_ids as id1,id2,id3
# The keys are used to connect as root to the droplet.

- digital_ocean: >
      state=present
      ssh_key_ids=id1,id2
      name=mydroplet
      client_id=XXX
      api_key=XXX
```

```
size_id=1
region_id=2
image_id=3
```

Note: Two environment variables can be used, DO_CLIENT_ID and DO_API_KEY.

docker - manage docker containers

Author Cove Schneider, Pavel Antonov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage the life cycle of docker containers.

Options

Note: Requires docker-py

Examples

```
Start one docker container running tomcat in each host of the web group and bind tomcat's listening p
on the host:

- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker: image=centos command="service tomcat6 start" ports=:8080

The tomcat server's port is NAT'ed to a dynamic port on the host, but you can determine which port th
mapped to using docker_containers:

- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker: image=centos command="service tomcat6 start" ports=8080 count=5
  - name: Display IP address and port mappings for containers
    debug: msg={{inventory_hostname}}:{{item.NetworkSettings.Ports['8080/tcp'][0].HostPort}}
    with_items: docker_containers

Just as in the previous example, but iterates over the list of docker containers with a sequence:
```

```
- hosts: web
  sudo: yes
  vars:
    start_containers_count: 5
  tasks:
  - name: run tomcat servers
    docker: image=centos command="service tomcat6 start" ports=8080 count={{start_containers_count}}
  - name: Display IP address and port mappings for containers
    debug: msg={{inventory_hostname}}:{{docker_containers[{{item}}].NetworkSettings.Ports['8080/tcp']
    with_sequence: start=0 end={{start_containers_count - 1}}

Stop, remove all of the running tomcat containers and list the exit code from the stopped containers

- hosts: web
  sudo: yes
  tasks:
  - name: stop tomcat servers
    docker: image=centos command="service tomcat6 start" state=absent
  - name: Display return codes from stopped containers
    debug: msg="Returned {{inventory_hostname}}:{{item}}"
    with_items: docker_containers
```

docker_image - manage docker images

Author Pavel Antonov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Create, check and remove docker images

Options

Note: Requires docker-py

Examples

```
Build docker image if required. Path should contains Dockerfile to build image:

- hosts: web
  sudo: yes
  tasks:
  - name: check or build image
    docker_image: path="/path/to/build/dir" name="my/app" state=present
```

```
Build new version of image:

- hosts: web
  sudo: yes
  tasks:
  - name: check or build image
    docker_image: path="/path/to/build/dir" name="my/app" state=build

Remove image from local docker storage:

- hosts: web
  sudo: yes
  tasks:
  - name: run tomcat servers
    docker_image: name="my/app" state=absent
```

ec2 - create, terminate, start or stop an instance in ec2, return instanceid

Author Seth Vidal, Tim Gerla, Lester Wade

- Synopsis
- Options
- Examples

Synopsis

Creates or terminates ec2 instances. When created optionally waits for it to be 'running'. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic provisioning example
- local_action:
    module: ec2
    keypair: mykey
    instance_type: c1.medium
    image: emi-40603AD1
    wait: yes
    group: webserver
    count: 3

# Advanced example with tagging and CloudWatch
```

```
- local_action:
    module: ec2
    keypair: mykey
    group: databases
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    count: 5
    instance_tags:
       db: postgres
    monitoring: yes

# Single instance with additional IOPS volume from snapshot
local_action:
    module: ec2
    keypair: mykey
    group: webserver
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    volumes:
    - device_name: /dev/sdb
      snapshot: snap-abcdef12
      device_type: io1
      iops: 1000
      volume_size: 100
    monitoring: yes

# Multiple groups example
local_action:
    module: ec2
    keypair: mykey
    group: ['databases', 'internal-services', 'sshable', 'and-so-forth']
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    count: 5
    instance_tags:
        db: postgres
    monitoring: yes

# Multiple instances with additional volume from snapshot
local_action:
    module: ec2
    keypair: mykey
    group: webserver
    instance_type: m1.large
    image: ami-6e649707
    wait: yes
    wait_timeout: 500
    count: 5
    volumes:
    - device_name: /dev/sdb
      snapshot: snap-abcdef12
      volume_size: 10
```

```
    monitoring: yes

# VPC example
- local_action:
    module: ec2
    keypair: mykey
    group_id: sg-1dc53f72
    instance_type: m1.small
    image: ami-6e649707
    wait: yes
    vpc_subnet_id: subnet-29e63245
    assign_public_ip: yes

# Launch instances, runs some tasks
# and then terminate them


- name: Create a sandbox instance
  hosts: localhost
  gather_facts: False
  vars:
    keypair: my_keypair
    instance_type: m1.small
    security_group: my_securitygroup
    image: my_ami_id
    region: us-east-1
  tasks:
    - name: Launch instance
      local_action: ec2 keypair={{ keypair }} group={{ security_group }} instance_type={{ instance_ty
      register: ec2
    - name: Add new instance to host group
      local_action: add_host hostname={{ item.public_ip }} groupname=launched
      with_items: ec2.instances
    - name: Wait for SSH to come up
      local_action: wait_for host={{ item.public_dns_name }} port=22 delay=60 timeout=320 state=start
      with_items: ec2.instances

- name: Configure instance(s)
  hosts: launched
  sudo: True
  gather_facts: True
  roles:
    - my_awesome_role
    - my_awesome_test

- name: Terminate instances
  hosts: localhost
  connection: local
  tasks:
    - name: Terminate instances that were previously launched
      local_action:
        module: ec2
        state: 'absent'
        instance_ids: '{{ ec2.instance_ids }}'

# Start a few existing instances, run some tasks
# and stop the instances
```

```
- name: Start sandbox instances
  hosts: localhost
  gather_facts: false
  connection: local
  vars:
    instance_ids:
      - 'i-xxxxxx'
      - 'i-xxxxxx'
      - 'i-xxxxxx'
    region: us-east-1
  tasks:
    - name: Start the sandbox instances
      local_action:
        module: ec2
        instance_ids: '{{ instance_ids }}'
        region: '{{ region }}'
        state: running
        wait: True
  role:
    - do_neat_stuff
    - do_more_neat_stuff

- name: Stop sandbox instances
  hosts: localhost
  gather_facts: false
  connection: local
  vars:
    instance_ids:
      - 'i-xxxxxx'
      - 'i-xxxxxx'
      - 'i-xxxxxx'
    region: us-east-1
  tasks:
    - name: Stop the sanbox instances
      local_action:
      module: ec2
      instance_ids: '{{ instance_ids }}'
      region: '{{ region }}'
      state: stopped
      wait: True

#
# Enforce that 5 instances with a tag "foo" are running
#

- local_action:
    module: ec2
    keypair: mykey
    instance_type: c1.medium
    image: emi-40603AD1
    wait: yes
    group: webserver
    instance_tags:
        foo: bar
    exact_count: 5
    count_tag: foo

#
```

```
# Enforce that 5 running instances named "database" with a "dbtype" of "postgres"
#

- local_action:
    module: ec2
    keypair: mykey
    instance_type: c1.medium
    image: emi-40603AD1
    wait: yes
    group: webserver
    instance_tags:
        Name: database
        dbtype: postgres
    exact_count: 5
    count_tag:
        Name: database
        dbtype: postgres

#
# count_tag complex argument examples
#

    # instances with tag foo
    count_tag:
        foo:

    # instances with tag foo=bar
    count_tag:
        foo: bar

    # instances with tags foo=bar & baz
    count_tag:
        foo: bar
        baz:

    # instances with tags foo & bar & baz=bang
    count_tag:
        - foo
        - bar
        - baz: bang
```

ec2_ami - create or destroy an image in ec2, return imageid

Author Evan Duffield <eduffield@iacquire.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates or deletes ec2 images. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
# Basic AMI Creation
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    instance_id: i-xxxxxx
    wait: yes
    name: newtest
  register: instance

# Basic AMI Creation, without waiting
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    region: xxxxxx
    instance_id: i-xxxxxx
    wait: no
    name: newtest
  register: instance

# Deregister/Delete AMI
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    region: xxxxxx
    image_id: ${instance.image_id}
    delete_snapshot: True
    state: absent

# Deregister AMI
- local_action:
    module: ec2_ami
    aws_access_key: xxxxxxxxxxxxxxxxxxxxx
    aws_secret_key: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    region: xxxxxx
    image_id: ${instance.image_id}
    delete_snapshot: False
    state: absent
```

ec2_eip - associate an EC2 elastic IP with an instance.

Author Lorin Hochstein <lorin@nimbisservices.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module associates AWS EC2 elastic IP addresses with instances

Options

Note: Requires boto

Examples

```
- name: associate an elastic IP with an instance
  ec2_eip: instance_id=i-1212f003 ip=93.184.216.119

- name: disassociate an elastic IP from an instance
  ec2_eip: instance_id=i-1212f003 ip=93.184.216.119 state=absent

- name: allocate a new elastic IP and associate it with an instance
  ec2_eip: instance_id=i-1212f003

- name: allocate a new elastic IP without associating it to anything
  ec2_eip:
  register: eip
- name: output the IP
  debug: msg="Allocated IP is {{ eip.public_ip }}"

- name: provision new instances with ec2
  ec2: keypair=mykey instance_type=c1.medium image=emi-40603AD1 wait=yes group=webserver count=3
  register: ec2
- name: associate new elastic IPs with each of the instances
  ec2_eip: "instance_id={{ item }}"
  with_items: ec2.instance_ids

- name: allocate a new elastic IP inside a VPC in us-west-2
  ec2_eip: region=us-west-2 in_vpc=yes
  register: eip
- name: output the IP
  debug: msg="Allocated IP inside a VPC is {{ eip.public_ip }}"
```

Note: This module will return `public_ip` on success, which will contain the public IP address associated with the instance.

Note: There may be a delay between the time the Elastic IP is assigned and when the cloud instance is reachable via the new address. Use wait_for and pause to delay further playbook execution until the instance is reachable, if necessary.

ec2_elb - De-registers or registers instances from EC2 ELBs

Author John Jarvis

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module de-registers or registers an AWS EC2 instance from the ELBs that it belongs to. Returns fact "ec2_elbs" which is a list of elbs attached to the instance if state=absent is passed as an argument. Will be marked changed when called only if there are ELBs found to operate on.

Options

Note: Requires boto

Examples

```
# basic pre_task and post_task example
pre_tasks:
  - name: Gathering ec2 facts
    ec2_facts:
  - name: Instance De-register
    local_action: ec2_elb
    args:
      instance_id: "{{ ansible_ec2_instance_id }}"
      state: 'absent'
roles:
  - myrole
post_tasks:
  - name: Instance Register
    local_action: ec2_elb
    args:
      instance_id: "{{ ansible_ec2_instance_id }}"
      ec2_elbs: "{{ item }}"
      state: 'present'
    with_items: ec2_elbs
```

ec2_elb_lb - Creates or destroys Amazon ELB. - Returns information about the load balancer. - Will be marked changed when called only if state is changed.

Author Jim Dalton

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

C r e a t e s

o r

d e s t r o y s

A m a z o n

E L B .

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic provisioning example
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1a
      - us-east-1d
    listeners:
      - protocol: http # options are http, https, ssl, tcp
        load_balancer_port: 80
        instance_port: 80
      - protocol: https
        load_balancer_port: 443
        instance_protocol: http # optional, defaults to value of protocol setting
        instance_port: 80
        # ssl certificate required for https or ssl
        ssl_certificate_id: "arn:aws:iam::123456789012:server-certificate/company/servercerts/ProdSe

# Configure a health check
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1d
```

```
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
    health_check:
        ping_protocol: http # options are http, https, ssl, tcp
        ping_port: 80
        ping_path: "/index.html" # not required for tcp or ssl
        response_timeout: 5 # seconds
        interval: 30 # seconds
        unhealthy_threshold: 2
        healthy_threshold: 10

# Ensure ELB is gone
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: absent

# Normally, this module will purge any listeners that exist on the ELB
# but aren't specified in the listeners parameter. If purge_listeners is
# false it leaves them alone
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1a
      - us-east-1d
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
    purge_listeners: no

# Normally, this module will leave availability zones that are enabled
# on the ELB alone. If purge_zones is true, then any extreneous zones
# will be removed
- local_action:
    module: ec2_elb_lb
    name: "test-please-delete"
    state: present
    zones:
      - us-east-1a
      - us-east-1d
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
    purge_zones: yes
```

ec2_facts - Gathers facts about remote hosts within ec2 (aws)

Author Silviu Dicu <silviudicu@gmail.com>

- Synopsis
- Examples

Synopsis

New in version 1.0.

This module fetches data from the metadata servers in ec2 (aws). Eucalyptus cloud provides a similar service and this module should work this cloud provider as well.

Examples

```
# Conditional example
- name: Gather facts
  action: ec2_facts

- name: Conditional
  action: debug msg="This instance is a t1.micro"
  when: ansible_ec2_instance_type == "t1.micro"
```

Note: Parameters to filter on ec2_facts may be added later.

ec2_group - maintain an ec2 VPC security group.

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

maintains ec2 security groups. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
- name: example ec2 group
  local_action:
    module: ec2_group
    name: example
    description: an example EC2 group
```

```
    vpc_id: 12345
    region: eu-west-1a
    ec2_secret_key: SECRET
    ec2_access_key: ACCESS
    rules:
      - proto: tcp
        from_port: 80
        to_port: 80
        cidr_ip: 0.0.0.0/0
      - proto: tcp
        from_port: 22
        to_port: 22
        cidr_ip: 10.0.0.0/8
      - proto: udp
        from_port: 10050
        to_port: 10050
        cidr_ip: 10.0.0.0/8
      - proto: udp
        from_port: 10051
        to_port: 10051
        group_id: sg-12345678
      - proto: all
        # the containing group name may be specified here
        group_name: example
```

ec2_key - maintain an ec2 key pair.

Author Vincent Viallet

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

maintains ec2 key pairs. This module has a dependency on python-boto >= 2.5

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Creates a new ec2 key pair named `example` if not present, returns generated
# private key
```

```
- name: example ec2 key
  local_action:
    module: ec2_key
    name: example

# Creates a new ec2 key pair named `example` if not present using provided key
# material
- name: example2 ec2 key
  local_action:
    module: ec2_key
    name: example2
    key_material: 'ssh-rsa AAAAxyz...== me@example.com'
    state: present

# Creates a new ec2 key pair named `example` if not present using provided key
# material
- name: example3 ec2 key
  local_action:
    module: ec2_key
    name: example3
    key_material: "{{ item }}"
  with_file: /path/to/public_key.id_rsa.pub

# Removes ec2 key pair by name
- name: remove example key
  local_action:
    module: ec2_key
    name: example
    state: absent
```

ec2_tag - create and remove tag(s) to ec2 resources.

Author Lester Wade

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates and removes tags from any EC2 resource. The resource is referenced by its resource id (e.g. an instance being i-XXXXXXX). It is designed to be used with complex args (tags), see the examples. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Basic example of adding tag(s)
tasks:
- name: tag a resource
  local_action: ec2_tag resource=vol-XXXXXX region=eu-west-1 state=present
  args:
    tags:
      Name: ubervol
      env: prod

# Playbook example of adding tag(s) to spawned instances
tasks:
- name: launch some instances
  local_action: ec2 keypair={{ keypair }} group={{ security_group }} instance_type={{ instance_type }
  register: ec2

- name: tag my launched instances
  local_action: ec2_tag resource={{ item.id }} region=eu-west-1 state=present
  with_items: ec2.instances
  args:
    tags:
      Name: webserver
      env: prod
```

ec2_vol - create and attach a volume, return volume id and device map

Author Lester Wade

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

creates an EBS volume and optionally attaches it to an instance. If both an instance ID and a device name is given and the instance has a device at the device name, then no volume is created and no attachment is made. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Simple attachment action
- local_action:
    module: ec2_vol
    instance: XXXXXX
    volume_size: 5
    device_name: sdd

# Example using custom iops params
- local_action:
    module: ec2_vol
    instance: XXXXXX
    volume_size: 5
    iops: 200
    device_name: sdd

# Example using snapshot id
- local_action:
    module: ec2_vol
    instance: XXXXXX
    snapshot: "{{ snapshot }}"

# Playbook example combined with instance launch
- local_action:
    module: ec2
    keypair: "{{ keypair }}"
    image: "{{ image }}"
    wait: yes
    count: 3
    register: ec2
- local_action:
    module: ec2_vol
    instance: "{{ item.id }} "
    volume_size: 5
    with_items: ec2.instances
    register: ec2_vol
```

ec2_vpc - configure AWS virtual private clouds

Author Carson Gee

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Create or terminates AWS virtual private clouds. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic creation example:
      local_action:
        module: ec2_vpc
        state: present
        cidr_block: 172.23.0.0/16
        region: us-west-2
# Full creation example with subnets and optional availability zones.
# The absence or presense of subnets deletes or creates them respectively.
      local_action:
        module: ec2_vpc
        state: present
        cidr_block: 172.22.0.0/16
        subnets:
          - cidr: 172.22.1.0/24
            az: us-west-2c
          - cidr: 172.22.2.0/24
            az: us-west-2b
          - cidr: 172.22.3.0/24
            az: us-west-2a
        internet_gateway: True
        route_tables:
          - subnets:
              - 172.22.2.0/24
              - 172.22.3.0/24
            routes:
              - dest: 0.0.0.0/0
                gw: igw
          - subnets:
              - 172.22.1.0/24
            routes:
              - dest: 0.0.0.0/0
                gw: igw
        region: us-west-2
      register: vpc

# Removal of a VPC by id
      local_action:
        module: ec2_vpc
        state: absent
        vpc_id: vpc-aaaaaaa
        region: us-west-2
If you have added elements not managed by this module, e.g. instances, NATs, etc then
the delete will fail until those dependencies are removed.
```

elasticache - Manage cache clusters in Amazon Elasticache.

Author Jim Dalton

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage cache clusters in Amazon Elasticache. Returns information about the specified cache cluster.

Options

Note: Requires boto

Examples

```
# Note: None of these examples set aws_access_key, aws_secret_key, or region.
# It is assumed that their matching environment variables are set.

# Basic example
- local_action:
    module: elasticache
    name: "test-please-delete"
    state: present
    engine: memcached
    cache_engine_version: 1.4.14
    node_type: cache.m1.small
    num_nodes: 1
    cache_port: 11211
    cache_security_groups:
      - default
    zone: us-east-1d


# Ensure cache cluster is gone
- local_action:
    module: elasticache
    name: "test-please-delete"
    state: absent

# Reboot cache cluster
- local_action:
    module: elasticache
    name: "test-please-delete"
    state: rebooted
```

gc_storage - This module manages objects/buckets in Google Cloud Storage.

Author benno@ansible.com Note. Most of the code has been taken from the S3 module.

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module allows users to manage their objects/buckets in Google Cloud Storage. It allows upload and download operations and can set some canned permissions. It also allows retrieval of URLs for objects for use in playbooks, and retrieval of string contents of objects. This module requires setting the default project in GCS prior to playbook usage. See https://developers.google.com/storage/docs/reference/v1/apiversion1 for information about setting the default project.

Options

Note: Requires boto 2.9+

Examples

```
# upload some content
- gc_storage: bucket=mybucket object=key.txt src=/usr/local/myfile.txt mode=put permission=public-rea

# download some content
- gc_storage: bucket=mybucket object=key.txt dest=/usr/local/myfile.txt mode=get

# Download an object as a string to use else where in your playbook
- gc_storage: bucket=mybucket object=key.txt mode=get_str

# Create an empty bucket
- gc_storage: bucket=mybucket mode=create

# Create a bucket with key as directory
- gc_storage: bucket=mybucket object=/my/directory/path mode=create

# Delete a bucket and all contents
- gc_storage: bucket=mybucket mode=delete
```

gce - create or terminate GCE instances

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Creates or terminates Google Compute Engine (GCE) instances. See https://cloud.google.com/products/compute-engine for an overview. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Basic provisioning example.  Create a single Debian 7 instance in the
# us-central1-a Zone of n1-standard-1 machine type.
- local_action:
    module: gce
    name: test-instance
    zone: us-central1-a
    machine_type: n1-standard-1
    image: debian-7

# Example using defaults and with metadata to create a single 'foo' instance
- local_action:
    module: gce
    name: foo
    metadata: '{"db":"postgres", "group":"qa", "id":500}'


# Launch instances from a control node, runs some tasks on the new instances,
# and then terminate them
- name: Create a sandbox instance
  hosts: localhost
  vars:
    names: foo,bar
    machine_type: n1-standard-1
    image: debian-6
    zone: us-central1-a
  tasks:
    - name: Launch instances
      local_action: gce instance_names={{names}} machine_type={{machine_type}}
                    image={{image}} zone={{zone}}
      register: gce
    - name: Wait for SSH to come up
      local_action: wait_for host={{item.public_ip}} port=22 delay=10
                    timeout=60 state=started
      with_items: {{gce.instance_data}}

- name: Configure instance(s)
  hosts: launched
  sudo: True
  roles:
    - my_awesome_role
```

```
    - my_awesome_tasks

- name: Terminate instances
  hosts: localhost
  connection: local
  tasks:
    - name: Terminate instances that were previously launched
      local_action:
        module: gce
        state: 'absent'
        instance_names: {{gce.instance_names}}
```

gce_lb - create/destroy GCE load-balancer resources

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module can create and destroy Google Compute Engine `loadbalancer` and `httphealthcheck` resources. The primary LB resource is the `load_balancer` resource and the health check parameters are all prefixed with *httphealthcheck*. The full documentation for Google Compute Engine load balancing is at https://developers.google.com/compute/docs/load-balancing/. However, the ansible module simplifies the configuration by following the libcloud model. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Simple example of creating a new LB, adding members, and a health check
- local_action:
    module: gce_lb
    name: testlb
    region: us-central1
    members: ["us-central1-a/www-a", "us-central1-b/www-b"]
    httphealthcheck_name: hc
    httphealthcheck_port: 80
    httphealthcheck_path: "/up"
```

gce_net - create/destroy GCE networks and firewall rules

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module can create and destroy Google Compue Engine networks and firewall rules https://developers.google.com/compute/docs/networking. The *name* parameter is reserved for referencing a network while the *fwname* parameter is used to reference firewall rules. IPv4 Address ranges must be specified using the CIDR http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing format. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Simple example of creating a new network
- local_action:
    module: gce_net
    name: privatenet
    ipv4_range: '10.240.16.0/24'

# Simple example of creating a new firewall rule
- local_action:
    module: gce_net
    name: privatenet
    allowed: tcp:80,8080
    src_tags: ["web", "proxy"]
```

gce_pd - utilize GCE persistent disk resources

Author Eric Johnson <erjohnso@google.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module can create and destroy unformatted GCE persistent disks https://developers.google.com/compute/docs/disks#persistentdisks It also supports attaching and detaching disks from running instances but does not support creating boot disks from images or snapshots. The 'gce' module supports creating instances with boot disks. Full install/configuration instructions for the gce* modules can be found in the comments of ansible/test/gce_tests.py.

Options

Note: Requires libcloud

Examples

```
# Simple attachment action to an existing instance
- local_action:
    module: gce_pd
    instance_name: notlocalhost
    size_gb: 5
    name: pd
```

glance_image - Add/Delete images from glance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove images from the glance repository.

Options

Note: Requires glanceclient

Note: Requires keystoneclient

Examples

```
# Upload an image from an HTTP URL
- glance_image: login_username=admin
                login_password=passme
                login_tenant_name=admin
                name=cirros
                container_format=bare
                disk_format=qcow2
                state=present
                copy_from=http:launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.im
```

keystone_user - Manage OpenStack Identity (keystone) users, tenants and roles

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage users,tenants, roles from OpenStack.

Options

Note: Requires python-keystoneclient

Examples

```
# Create a tenant
- keystone_user: tenant=demo tenant_description="Default Tenant"

# Create a user
- keystone_user: user=john tenant=demo password=secrete

# Apply the admin role to the john user in the demo tenant
- keystone_user: role=admin user=john tenant=demo
```

linode - create / delete / stop / restart an instance in Linode Public Cloud

Author Vincent Viallet

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

creates / deletes a Linode Public Cloud instance and optionally waits for it to be 'running'.

Options

Note: Requires linode-python

Examples

```
# Create a server
- local_action:
     module: linode
     api_key: 'longStringFromLinodeApi'
     name: linode-test1
     plan: 1
     datacenter: 2
     distribution: 99
     password: 'superSecureRootPassword'
     ssh_pub_key: 'ssh-rsa qwerty'
     swap: 768
     wait: yes
     wait_timeout: 600
     state: present

# Ensure a running server (create if missing)
- local_action:
     module: linode
     api_key: 'longStringFromLinodeApi'
     name: linode-test1
     linode_id: 12345678
     plan: 1
     datacenter: 2
     distribution: 99
     password: 'superSecureRootPassword'
     ssh_pub_key: 'ssh-rsa qwerty'
     swap: 768
     wait: yes
     wait_timeout: 600
     state: present

# Delete a server
- local_action:
     module: linode
     api_key: 'longStringFromLinodeApi'
     name: linode-test1
     linode_id: 12345678
     state: absent

# Stop a server
- local_action:
     module: linode
```

```
    api_key: 'longStringFromLinodeApi'
    name: linode-test1
    linode_id: 12345678
    state: stopped

# Reboot a server
- local_action:
    module: linode
    api_key: 'longStringFromLinodeApi'
    name: linode-test1
    linode_id: 12345678
    state: restarted
```

Note: LINODE_API_KEY env variable can be used instead

nova_compute - Create/Delete VMs from OpenStack

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Create or Remove virtual machines from Openstack.

Options

Note: Requires novaclient

Examples

```
# Creates a new VM and attaches to a network and passes metadata to the instance
- nova_compute:
       state: present
       login_username: admin
       login_password: admin
       login_tenant_name: admin
       name: vm1
       image_id: 4f905f38-e52a-43d2-b6ec-754a13ffb529
       key_name: ansible_key
       wait_for: 200
       flavor_id: 4
       nics:
         - net-id: 34605f38-e52a-25d2-b6ec-754a13ffb723
       meta:
         hostname: test1
         group: uge_master
```

nova_keypair - Add/Delete key pair from nova

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove key pair from nova .

Options

Note: Requires novaclient

Examples

```
# Creates a key pair with the running users public key
- nova_keypair: state=present login_username=admin
                login_password=admin login_tenant_name=admin name=ansible_key
                public_key={{ lookup('file','~/.ssh/id_rsa.pub') }}

# Creates a new key pair and the private key returned after the run.
- nova_keypair: state=present login_username=admin login_password=admin
                login_tenant_name=admin name=ansible_key
```

ovirt - oVirt/RHEV platform management

Author Vincent Van der Kussen

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

allows you to create new instances, either from scratch or an image, in addition to deleting or stopping instances on the oVirt/RHEV platform

Options

Note: Requires ovirt-engine-sdk

Examples

```
# Basic example provisioning from image.

action: ovirt >
    user=admin@internal
    url=https://ovirt.example.com
    instance_name=ansiblevm04
    password=secret
    image=centos_64
    zone=cluster01
    resource_type=template"

# Full example to create new instance from scratch
action: ovirt >
    instance_name=testansible
    resource_type=new
    instance_type=server
    user=admin@internal
    password=secret
    url=https://ovirt.example.com
    instance_disksize=10
    zone=cluster01
    region=datacenter1
    instance_cpus=1
    instance_nic=nic1
    instance_network=rhevm
    instance_mem=1000
    disk_alloc=thin
    sdomain=FIBER01
    instance_cores=1
    instance_os=rhel_6x64
    disk_int=virtio"

# stopping an instance
action: ovirt >
    instance_name=testansible
    state=stopped
    user=admin@internal
    password=secret
    url=https://ovirt.example.com

# starting an instance
action: ovirt >
    instance_name=testansible
    state=started
    user=admin@internal
    password=secret
    url=https://ovirt.example.com
```

quantum_floating_ip - Add/Remove floating IP from an instance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove a floating IP to an instance

Options

Note: Requires novaclient

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Assign a floating ip to the instance from an external network
- quantum_floating_ip: state=present login_username=admin login_password=admin
                       login_tenant_name=admin network_name=external_network
                       instance_name=vm1 internal_network_name=internal_network
```

quantum_floating_ip_associate - Associate or disassociate a particular floating IP with an instance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Associates or disassociates a specific floating IP with a particular instance

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Associate a specific floating IP with an Instance
- quantum_floating_ip_associate:
           state=present
           login_username=admin
           login_password=admin
           login_tenant_name=admin
           ip_address=1.1.1.1
           instance_name=vm1
```

quantum_network - Creates/Removes networks from OpenStack

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or Remove network from OpenStack.

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Create a GRE backed Quantum network with tunnel id 1 for tenant1
- quantum_network: name=t1network tenant_name=tenant1 state=present
                   provider_network_type=gre provider_segmentation_id=1
                   login_username=admin login_password=admin login_tenant_name=admin

# Create an external network
- quantum_network: name=external_network state=present
                   provider_network_type=local router_external=yes
                   login_username=admin login_password=admin login_tenant_name=admin
```

quantum_router - Create or Remove router from openstack

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Create or Delete routers from OpenStack

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Creates a router for tenant admin
- quantum_router: state=present
                  login_username=admin
                  login_password=admin
                  login_tenant_name=admin
                  name=router1"
```

quantum_router_gateway - set/unset a gateway interface for the router with the specified external network

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Creates/Removes a gateway interface from the router, used to associate a external network with a router to route external traffic.

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Attach an external network with a router to allow flow of external traffic
- quantum_router_gateway: state=present login_username=admin login_password=admin
                          login_tenant_name=admin router_name=external_router
                          network_name=external_network
```

quantum_router_interface - Attach/Dettach a subnet's interface to a router

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Attach/Dettach a subnet interface to a router, to provide a gateway for the subnet.

Options

Note: Requires quantumclient

Note: Requires keystoneclient

Examples

```
# Attach tenant1's subnet to the external router
- quantum_router_interface: state=present login_username=admin
                            login_password=admin
                            login_tenant_name=admin
                            tenant_name=tenant1
                            router_name=external_route
                            subnet_name=t1subnet
```

quantum_subnet - Add/Remove floating IP from an instance

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Add or Remove a floating IP to an instance

Options

Note: Requires quantumclient

Note: Requires neutronclient

Note: Requires keystoneclient

Examples

```
# Create a subnet for a tenant with the specified subnet
- quantum_subnet: state=present login_username=admin login_password=admin
                  login_tenant_name=admin tenant_name=tenant1
                  network_name=network1 name=net1subnet cidr=192.168.0.0/24"
```

rax - create / delete an instance in Rackspace Public Cloud

Author Jesse Keating, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

creates / deletes a Rackspace Public Cloud instance and optionally waits for it to be 'running'.

Options

Note: Requires pyrax

Examples

```
- name: Build a Cloud Server
  gather_facts: False
  tasks:
    - name: Server build request
      local_action:
        module: rax
        credentials: ~/.raxpub
        name: rax-test1
        flavor: 5
        image: b11d9567-e412-4255-96b9-bd63ab23bcfe
        files:
          /root/.ssh/authorized_keys: /home/localuser/.ssh/id_rsa.pub
          /root/test.txt: /home/localuser/test.txt
        wait: yes
        state: present
        networks:
          - private
          - public
      register: rax

- name: Build an exact count of cloud servers with incremented names
  hosts: local
  gather_facts: False
  tasks:
    - name: Server build requests
      local_action:
        module: rax
        credentials: ~/.raxpub
        name: test%03d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        state: present
        count: 10
        count_offset: 10
        exact_count: yes
```

```
        group: test
        wait: yes
      register: rax
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_clb - create / delete a load balancer in Rackspace Public Cloud

Author Christopher H. Laco, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

creates / deletes a Rackspace Public Cloud load balancer.

Options

Note: Requires pyrax

Examples

```
- name: Build a Load Balancer
  gather_facts: False
  hosts: local
  connection: local
  tasks:
    - name: Load Balancer create request
      local_action:
        module: rax_clb
        credentials: ~/.raxpub
        name: my-lb
        port: 8080
        protocol: HTTP
```

```
        type: SERVICENET
        timeout: 30
        region: DFW
        wait: yes
        state: present
        meta:
          app: my-cool-app
      register: my_lb
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_clb_nodes - add, modify and remove nodes from a Rackspace Cloud Load Balancer

Author Lukasz Kawczynski

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Adds, modifies and removes nodes from a Rackspace Cloud Load Balancer

Options

Note: Requires pyrax

Examples

```
# Add a new node to the load balancer
- local_action:
    module: rax_clb_nodes
    load_balancer_id: 71
    address: 10.2.2.3
    port: 80
    condition: enabled
```

```
    type: primary
    wait: yes
    credentials: /path/to/credentials

# Drain connections from a node
- local_action:
    module: rax_clb_nodes
    load_balancer_id: 71
    node_id: 410
    condition: draining
    wait: yes
    credentials: /path/to/credentials

# Remove a node from the load balancer
- local_action:
    module: rax_clb_nodes
    load_balancer_id: 71
    node_id: 410
    state: absent
    wait: yes
    credentials: /path/to/credentials
```

Note: The following environment variables can be used: `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDENTIALS` and `RAX_REGION`.

rax_dns - Manage domains on Rackspace Cloud DNS

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manage domains on Rackspace Cloud DNS

Options

Note: Requires pyrax

Examples

```
- name: Create domain
  hosts: all
  gather_facts: False
  tasks:
```

```
    - name: Domain create request
      local_action:
        module: rax_dns
        credentials: ~/.raxpub
        name: example.org
        email: admin@example.org
      register: rax_dns
```

Note: The following environment variables can be used, RAX_USERNAME, RAX_API_KEY, RAX_CREDS_FILE, RAX_CREDENTIALS, RAX_REGION.

Note: RAX_CREDENTIALS and RAX_CREDS_FILE points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: RAX_USERNAME and RAX_API_KEY obviate the use of a credentials file

Note: RAX_REGION defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_dns_record - Manage DNS records on Rackspace Cloud DNS

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manage DNS records on Rackspace Cloud DNS

Options

Note: Requires pyrax

Examples

```
- name: Create record
  hosts: all
  gather_facts: False
  tasks:
    - name: Record create request
      local_action:
        module: rax_dns_record
        credentials: ~/.raxpub
```

```
        domain: example.org
        name: www.example.org
        data: 127.0.0.1
        type: A
      register: rax_dns_record
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_facts - Gather facts for Rackspace Cloud Servers

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Gather facts for Rackspace Cloud Servers.

Options

Note: Requires pyrax

Examples

```
- name: Gather info about servers
  hosts: all
  gather_facts: False
  tasks:
    - name: Get facts about servers
      local_action:
        module: rax_facts
        credentials: ~/.raxpub
        name: "{{ inventory_hostname }}"
        region: DFW
```

```
    - name: Map some facts
      set_fact:
        ansible_ssh_host: "{{ rax_accessipv4 }}"
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_files - Manipulate Rackspace Cloud Files Containers

Author Paul Durivage

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manipulate Rackspace Cloud Files Containers

Options

Note: Requires pyrax

Examples

```
- name: "Test Cloud Files Containers"
  hosts: local
  gather_facts: no
  tasks:
    - name: "List all containers"
      rax_files: state=list

    - name: "Create container called 'mycontainer'"
      rax_files: container=mycontainer

    - name: "Create container 'mycontainer2' with metadata"
      rax_files:
```

```
        container: mycontainer2
        meta:
          key: value
          file_for: someuser@example.com

    - name: "Set a container's web index page"
      rax_files: container=mycontainer web_index=index.html

    - name: "Set a container's web error page"
      rax_files: container=mycontainer web_error=error.html

    - name: "Make container public"
      rax_files: container=mycontainer public=yes

    - name: "Make container public with a 24 hour TTL"
      rax_files: container=mycontainer public=yes ttl=86400

    - name: "Make container private"
      rax_files: container=mycontainer private=yes

- name: "Test Cloud Files Containers Metadata Storage"
  hosts: local
  gather_facts: no
  tasks:
    - name: "Get mycontainer2 metadata"
      rax_files:
        container: mycontainer2
        type: meta

    - name: "Set mycontainer2 metadata"
      rax_files:
        container: mycontainer2
        type: meta
        meta:
          uploaded_by: someuser@example.com

    - name: "Remove mycontainer2 metadata"
      rax_files:
        container: "mycontainer2"
        type: meta
        state: absent
        meta:
          key: ""
          file_for: ""
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_files_objects - Upload, download, and delete objects in Rackspace Cloud Files

Author Paul Durivage

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Upload, download, and delete objects in Rackspace Cloud Files

Options

Note: Requires pyrax

Examples

```
- name: "Test Cloud Files Objects"
  hosts: local
  gather_facts: False
  tasks:
    - name: "Get objects from test container"
      rax_files_objects: container=testcont dest=~/Downloads/testcont

    - name: "Get single object from test container"
      rax_files_objects: container=testcont src=file1 dest=~/Downloads/testcont

    - name: "Get several objects from test container"
      rax_files_objects: container=testcont src=file1,file2,file3 dest=~/Downloads/testcont

    - name: "Delete one object in test container"
      rax_files_objects: container=testcont method=delete dest=file1

    - name: "Delete several objects in test container"
      rax_files_objects: container=testcont method=delete dest=file2,file3,file4

    - name: "Delete all objects in test container"
      rax_files_objects: container=testcont method=delete

    - name: "Upload all files to test container"
      rax_files_objects: container=testcont method=put src=~/Downloads/onehundred

    - name: "Upload one file to test container"
      rax_files_objects: container=testcont method=put src=~/Downloads/testcont/file1

    - name: "Upload one file to test container with metadata"
      rax_files_objects:
        container: testcont
```

```
        src: ~/Downloads/testcont/file2
        method: put
        meta:
          testkey: testdata
          who_uploaded_this: someuser@example.com

    - name: "Upload one file to test container with TTL of 60 seconds"
      rax_files_objects: container=testcont method=put src=~/Downloads/testcont/file3 expires=60

    - name: "Attempt to get remote object that does not exist"
      rax_files_objects: container=testcont method=get src=FileThatDoesNotExist.jpg dest=~/Downloads/
      ignore_errors: yes

    - name: "Attempt to delete remote object that does not exist"
      rax_files_objects: container=testcont method=delete dest=FileThatDoesNotExist.jpg
      ignore_errors: yes

- name: "Test Cloud Files Objects Metadata"
  hosts: local
  gather_facts: false
  tasks:
    - name: "Get metadata on one object"
      rax_files_objects:  container=testcont type=meta dest=file2

    - name: "Get metadata on several objects"
      rax_files_objects:  container=testcont type=meta src=file2,file1

    - name: "Set metadata on an object"
      rax_files_objects:
        container: testcont
        type: meta
        dest: file17
        method: put
        meta:
          key1: value1
          key2: value2
        clear_meta: true

    - name: "Verify metadata is set"
      rax_files_objects:  container=testcont type=meta src=file17

    - name: "Delete metadata"
      rax_files_objects:
        container: testcont
        type: meta
        dest: file17
        method: delete
        meta:
          key1: ''
          key2: ''

    - name: "Get metadata on all objects"
      rax_files_objects:  container=testcont type=meta
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_keypair - Create a keypair for use with Rackspace Cloud Servers

Author Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Create a keypair for use with Rackspace Cloud Servers

Options

Note: Requires pyrax

Examples

```
- name: Create a keypair
  hosts: local
  gather_facts: False
  tasks:
    - name: keypair request
      local_action:
        module: rax_keypair
        credentials: ~/.raxpub
        name: my_keypair
        region: DFW
      register: keypair
    - name: Create local public key
      local_action:
        module: copy
        content: "{{ keypair.keypair.public_key }}"
        dest: "{{ inventory_dir }}/{{ keypair.keypair.name }}.pub"
    - name: Create local private key
      local_action:
        module: copy
        content: "{{ keypair.keypair.private_key }}"
        dest: "{{ inventory_dir }}/{{ keypair.keypair.name }}"
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS_FILE` points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

Note: Keypairs cannot be manipulated, only created and deleted. To "update" a keypair you must first delete and then recreate.

rax_network - create / delete an isolated network in Rackspace Public Cloud

Author Christopher H. Laco, Jesse Keating

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

creates / deletes a Rackspace Public Cloud isolated network.

Options

Note: Requires pyrax

Examples

```
- name: Build an Isolated Network
  gather_facts: False

  tasks:
    - name: Network create request
      local_action:
        module: rax_network
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        state: present
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: `RAX_CREDENTIALS` and `RAX_CREDS` points to a credentials file appropriate for pyrax

Note: `RAX_USERNAME` and `RAX_API_KEY` obviate the use of a credentials file

Note: `RAX_REGION` defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rax_queue - create / delete a queue in Rackspace Public Cloud

Author Christopher H. Laco, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

creates / deletes a Rackspace Public Cloud queue.

Options

Note: Requires pyrax

Examples

```
- name: Build a Queue
  gather_facts: False
  hosts: local
  connection: local
  tasks:
    - name: Queue create request
      local_action:
        module: rax_queue
        credentials: ~/.raxpub
        client_id: unique-client-name
        name: my-queue
        region: DFW
        state: present
      register: my_queue
```

Note: The following environment variables can be used, `RAX_USERNAME`, `RAX_API_KEY`, `RAX_CREDS_FILE`, `RAX_CREDENTIALS`, `RAX_REGION`.

Note: RAX_CREDENTIALS and RAX_CREDS_FILE points to a credentials file appropriate for pyrax. See https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#authenticating

Note: RAX_USERNAME and RAX_API_KEY obviate the use of a credentials file

Note: RAX_REGION defines a Rackspace Public Cloud region (DFW, ORD, LON, ...)

rds - create, delete, or modify an Amazon rds instance

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates, deletes, or modifies rds instances. When creating an instance it can be either a new instance or a read-only replica of an existing instance. This module has a dependency on python-boto >= 2.5. The 'promote' command requires boto >= 2.18.0.

Options

Note: Requires boto

Examples

```
# Basic mysql provisioning example
- rds: >
      command=create
      instance_name=new_database
      db_engine=MySQL
      size=10
      instance_type=db.m1.small
      username=mysql_admin
      password=1nsecure

# Create a read-only replica and wait for it to become available
- rds: >
      command=replicate
      instance_name=new_database_replica
      source_instance=new_database
      wait=yes
      wait_timeout=600
```

```
# Delete an instance, but create a snapshot before doing so
- rds: >
      command=delete
      instance_name=new_database
      snapshot=new_database_snapshot

# Get facts about an instance
- rds: >
      command=facts
      instance_name=new_database
      register: new_database_facts

# Rename an instance and wait for the change to take effect
- rds: >
      command=modify
      instance_name=new_database
      new_instance_name=renamed_database
      wait=yes
```

route53 - add or delete entries in Amazons Route53 DNS service

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Creates and deletes DNS records in Amazons Route53 service

Options

Note: Requires boto

Examples

```
# Add new.foo.com as an A record with 3 IPs
- route53: >
      command=create
      zone=foo.com
      record=new.foo.com
      type=A
      ttl=7200
      value=1.1.1.1,2.2.2.2,3.3.3.3

# Retrieve the details for new.foo.com
```

```
- route53: >
      command=get
      zone=foo.com
      record=new.foo.com
      type=A
  register: rec

# Delete new.foo.com A record using the results from the get command
- route53: >
      command=delete
      zone=foo.com
      record={{ rec.set.record }}
      type={{ rec.set.type }}
      value={{ rec.set.value }}

# Add an AAAA record.  Note that because there are colons in the value
# that the entire parameter list must be quoted:
- route53: >
      command=create
      zone=foo.com
      record=localhost.foo.com
      type=AAAA
      ttl=7200
      value="::1"
```

s3 - idempotent S3 module putting a file into S3.

Author Lester Wade, Ralph Tice

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

This module allows the user to dictate the presence of a given file in an S3 bucket. If or once the key (file) exists in the bucket, it returns a time-expired download URL. This module has a dependency on python-boto.

Options

Note: Requires boto

Examples

```
# Simple PUT operation
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=put
# Simple GET operation
- s3: bucket=mybucket object=/my/desired/key.txt dest=/usr/local/myfile.txt mode=get
```

```
# GET/download and overwrite local file (trust remote)
- s3: bucket=mybucket object=/my/desired/key.txt dest=/usr/local/myfile.txt mode=get
# GET/download and do not overwrite local file (trust remote)
- s3: bucket=mybucket object=/my/desired/key.txt dest=/usr/local/myfile.txt mode=get force=false
# PUT/upload and overwrite remote file (trust local)
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=put
# PUT/upload and do not overwrite remote file (trust local)
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=put force=false
# Download an object as a string to use else where in your playbook
- s3: bucket=mybucket object=/my/desired/key.txt src=/usr/local/myfile.txt mode=getstr
# Create an empty bucket
- s3: bucket=mybucket mode=create
# Create a bucket with key as directory
- s3: bucket=mybucket object=/my/directory/path mode=create
# Delete a bucket and all contents
- s3: bucket=mybucket mode=delete
```

virt - Manages virtual machines supported by libvirt

Author Michael DeHaan, Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

Manages virtual machines supported by *libvirt*.

Options

Note: Requires libvirt

Examples

```
# a playbook task line:
- virt: name=alpha state=running

# /usr/bin/ansible invocations
ansible host -m virt -a "name=alpha command=status"
ansible host -m virt -a "name=alpha command=get_xml"
ansible host -m virt -a "name=alpha command=create uri=lxc:///"

# a playbook example of defining and launching an LXC guest
tasks:
  - name: define vm
    virt: name=foo
          command=define
          xml="{{ lookup('template', 'container-template.xml.j2') }}"
```

```
        uri=lxc:///
  - name: start vm
    virt: name=foo state=running uri=lxc:///
```

1.6.3 Commands Modules

command - Executes a command on a remote node

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `command` module takes the command name followed by a list of space-delimited arguments. The given command will be executed on all selected nodes. It will not be processed through the shell, so variables like `$HOME` and operations like `"<"`, `">"`, `"|"`, and `"&"` will not work (use the `shell` module if you need these features).

Options

Examples

```
# Example from Ansible Playbooks
- command: /sbin/shutdown -t now

# Run the command if the specified file does not exist
- command: /usr/bin/make_database.sh arg1 arg2 creates=/path/to/database
```

Note: If you want to run a command through the shell (say you are using <, >, |, etc), you actually want the `shell` module instead. The `command` module is much more secure as it's not affected by the user's environment.

Note: `creates`, `removes`, and `chdir` can be specified after the command. For instance, if you only want to run a command if a certain file does not exist, use this.

---

raw - Executes a low-down and dirty SSH command

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

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. Standard output, error output and return code are returned when available. There is no change handler support for this module. This module does not require python on the remote system, much like the `script` module.

Options

Examples

```
# Bootstrap a legacy python 2.4 host
- raw: yum -y install python-simplejson
```

Note: If you want to execute a command securely and predictably, it may be better to use the `command` module instead. Best practices when writing playbooks will follow the trend of using `command` unless `shell` is explicitly required. When running ad-hoc commands, use your best judgement.

script - Runs a local script on a remote node after transferring it

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `script` module takes the script name followed by a list of space-delimited arguments. The local script at path will be transfered to the remote node and then executed. The given script will be processed through the shell environment on the remote node. This module does not require python on the remote system, much like the `raw` module.

Options

Examples

```
# Example from Ansible Playbooks
- script: /some/local/script.sh --some-arguments 1234
```

Note: It is usually preferable to write Ansible modules than pushing scripts. Convert your script to an Ansible module for bonus points!

shell - Execute commands in nodes.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `shell` module takes the command name followed by a list of space-delimited arguments. It is almost exactly like the `command` module but runs the command through a shell (`/bin/sh`) on the remote node.

Options

Examples

```
# Execute the command in remote shell; stdout goes to the specified
# file on the remote
- shell: somescript.sh >> somelog.txt
```

Note: If you want to execute a command securely and predictably, it may be better to use the `command` module instead. Best practices when writing playbooks will follow the trend of using `command` unless `shell` is explicitly required. When running ad-hoc commands, use your best judgement.

1.6.4 Database Modules

mongodb_user - Adds or removes a user from a MongoDB database.

Author Elliott Foster

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Adds or removes a user from a MongoDB database.

Options

Note: Requires pymongo

Examples

```
# Create 'burgers' database user with name 'bob' and password '12345'.
- mongodb_user: database=burgers name=bob password=12345 state=present

# Delete 'burgers' database user with name 'bob'.
- mongodb_user: database=burgers name=bob state=absent

# Define more users with various specific roles (if not defined, no roles is assigned, and the user w
- mongodb_user: database=burgers name=ben password=12345 roles='read' state=present
- mongodb_user: database=burgers name=jim password=12345 roles='readWrite,dbAdmin,userAdmin' state=pr
- mongodb_user: database=burgers name=joe password=12345 roles='readWriteAnyDatabase' state=present
```

Note: Requires the pymongo Python package on the remote host, version 2.4.2+. This can be installed using pip or the OS package manager. @see http://api.mongodb.org/python/current/installation.html

mysql_db - Add or remove MySQL databases from a remote host.

Author Mark Theunissen

- Synopsis
- Options
- Examples

Synopsis

Add or remove MySQL databases from a remote host.

Options

Note: Requires ConfigParser

Examples

```
# Create a new database with name 'bobdata'
- mysql_db: name=bobdata state=present
```

Note: Requires the MySQLdb Python package on the remote host. For Ubuntu, this is as easy as apt-get install python-mysqldb. (See `apt`.)

Note: Both *login_password* and *login_user* are required when you are passing credentials. If none are present, the module will attempt to read the credentials from `~/.my.cnf`, and finally fall back to using the MySQL default login of `root` with no password.

mysql_replication - Manage MySQL replication

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages MySQL server replication, slave, master status get and change master host.

Options

Examples

```
# Stop mysql slave thread
- mysql_replication: mode=stopslave

# Get master binlog file name and binlog position
- mysql_replication: mode=getmaster

# Change master to master server 192.168.1.1 and use binary log 'mysql-bin.000009' with position 4578
- mysql_replication: mode=changemaster master_host=192.168.1.1 master_log_file=mysql-bin.000009 maste
```

mysql_user - Adds or removes a user from a MySQL database.

Author Mark Theunissen

- Synopsis
- Options
- Examples

Synopsis

Adds or removes a user from a MySQL database.

Options

Note: Requires ConfigParser

Note: Requires MySQLdb

Examples

```
# Create database user with name 'bob' and password '12345' with all database privileges
- mysql_user: name=bob password=12345 priv=*.*:ALL state=present

# Creates database user 'bob' and password '12345' with all database privileges and 'WITH GRANT OPTIO
- mysql_user: name=bob password=12345 priv=*.*:ALL,GRANT state=present

# Ensure no user named 'sally' exists, also passing in the auth credentials.
- mysql_user: login_user=root login_password=123456 name=sally state=absent

# Example privileges string format
mydb.*:INSERT,UPDATE/anotherdb.*:SELECT/yetanotherdb.*:ALL

# Example using login_unix_socket to connect to server
- mysql_user: name=root password=abc123 login_unix_socket=/var/run/mysqld/mysqld.sock

# Example .my.cnf file for setting the root password
# Note: don't use quotes around the password, because the mysql_user module
# will include them in the password but the mysql client will not

[client]
user=root
password=n<_665{vS43y
```

Note: Requires the MySQLdb Python package on the remote host. For Ubuntu, this is as easy as apt-get install python-mysqldb.

Note: Both `login_password` and `login_username` are required when you are passing credentials. If none are present, the module will attempt to read the credentials from `~/.my.cnf`, and finally fall back to using the MySQL default login of 'root' with no password.

Note: MySQL server installs with default login_user of 'root' and no password. To secure this user as part of an idempotent playbook, you must create at least two tasks: the first must change the root user's password, without providing any login_user/login_password details. The second must drop a ~/.my.cnf file containing the new root credentials. Subsequent runs of the playbook will then succeed by reading the new credentials from the file.

---

mysql_variables - Manage MySQL global variables

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Query / Set MySQL variables

Options

Examples

```
# Check for sync_binary_log setting
- mysql_variables: variable=sync_binary_log

# Set read_only variable to 1
- mysql_variables: variable=read_only value=1
```

postgresql_db - Add or remove PostgreSQL databases from a remote host.

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

Add or remove PostgreSQL databases from a remote host.

Options

Note: Requires psycopg2

Examples

```
# Create a new database with name "acme"
- postgresql_db: name=acme

# Create a new database with name "acme" and specific encoding and locale
# settings. If a template different from "template0" is specified, encoding
# and locale settings must match those of the template.
- postgresql_db: name=acme
                 encoding='UTF-8'
                 lc_collate='de_DE.UTF-8'
                 lc_ctype='de_DE.UTF-8'
                 template='template0'
```

Note: The default authentication assumes that you are either logging in as or sudo'ing to the `postgres` account on the host.

Note: This module uses *psycopg2*, a Python PostgreSQL database adapter. You must ensure that psycopg2 is installed on the host before using this module. If the remote host is the PostgreSQL server (which is the default case), then PostgreSQL must also be installed on the remote host. For Ubuntu-based systems, install the `postgresql`, `libpq-dev`, and `python-psycopg2` packages on the remote host before using this module.

postgresql_privs - Grant or revoke privileges on PostgreSQL database objects.

Author Bernhard Weitzhofer

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Grant or revoke privileges on PostgreSQL database objects. This module is basically a wrapper around most of the functionality of PostgreSQL's GRANT and REVOKE statements with detection of changes (GRANT/REVOKE *privs* ON *type objs* TO/FROM *roles*)

Options

Note: Requires psycopg2

Examples

```
# On database "library":
# GRANT SELECT, INSERT, UPDATE ON TABLE public.books, public.authors
# TO librarian, reader WITH GRANT OPTION
- postgresql_privs: >
    database=library
    state=present
    privs=SELECT,INSERT,UPDATE
    type=table
    objs=books,authors
    schema=public
    roles=librarian,reader
    grant_option=yes

# Same as above leveraging default values:
- postgresql_privs: >
    db=library
    privs=SELECT,INSERT,UPDATE
    objs=books,authors
    roles=librarian,reader
    grant_option=yes

# REVOKE GRANT OPTION FOR INSERT ON TABLE books FROM reader
# Note that role "reader" will be *granted* INSERT privilege itself if this
# isn't already the case (since state=present).
- postgresql_privs: >
    db=library
    state=present
    priv=INSERT
    obj=books
```

```
    role=reader
    grant_option=no

# REVOKE INSERT, UPDATE ON ALL TABLES IN SCHEMA public FROM reader
# "public" is the default schema. This also works for PostgreSQL 8.x.
- postgresql_privs: >
    db=library
    state=absent
    privs=INSERT,UPDATE
    objs=ALL_IN_SCHEMA
    role=reader

# GRANT ALL PRIVILEGES ON SCHEMA public, math TO librarian
- postgresql_privs: >
    db=library
    privs=ALL
    type=schema
    objs=public,math
    role=librarian

# GRANT ALL PRIVILEGES ON FUNCTION math.add(int, int) TO librarian, reader
# Note the separation of arguments with colons.
- postgresql_privs: >
    db=library
    privs=ALL
    type=function
    obj=add(int:int)
    schema=math
    roles=librarian,reader

# GRANT librarian, reader TO alice, bob WITH ADMIN OPTION
# Note that group role memberships apply cluster-wide and therefore are not
# restricted to database "library" here.
- postgresql_privs: >
    db=library
    type=group
    objs=librarian,reader
    roles=alice,bob
    admin_option=yes

# GRANT ALL PRIVILEGES ON DATABASE library TO librarian
# Note that here "db=postgres" specifies the database to connect to, not the
# database to grant privileges on (which is specified via the "objs" param)
- postgresql_privs: >
    db=postgres
    privs=ALL
    type=database
    obj=library
    role=librarian

# GRANT ALL PRIVILEGES ON DATABASE library TO librarian
# If objs is omitted for type "database", it defaults to the database
# to which the connection is established
- postgresql_privs: >
    db=library
    privs=ALL
    type=database
    role=librarian
```

Note: Default authentication assumes that postgresql_privs is run by the `postgres` user on the remote host. (Ansible's `user` or `sudo-user`).

Note: This module requires Python package *psycopg2* to be installed on the remote host. In the default case of the remote host also being the PostgreSQL server, PostgreSQL has to be installed there as well, obviously. For Debian/Ubuntu-based systems, install packages *postgresql* and *python-psycopg2*.

Note: Parameters that accept comma separated lists (*privs*, *objs*, *roles*) have singular alias names (*priv*, *obj*, *role*).

Note: To revoke only `GRANT OPTION` for a specific object, set *state* to `present` and *grant_option* to `no` (see examples).

Note: Note that when revoking privileges from a role R, this role may still have access via privileges granted to any role R is a member of including `PUBLIC`.

Note: Note that when revoking privileges from a role R, you do so as the user specified via *login*. If R has been granted the same privileges by another user also, R can still access database objects via these privileges.

Note: When revoking privileges, `RESTRICT` is assumed (see PostgreSQL docs).

postgresql_user - Adds or removes a users (roles) from a PostgreSQL database.

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

Add or remove PostgreSQL users (roles) from a remote host and, optionally, grant the users access to an existing database or tables. The fundamental function of the module is to create, or delete, roles from a PostgreSQL cluster. Privilege assignment, or removal, is an optional step, which works on one database at a time. This allows for the module to be called several times in the same module to modify the permissions on different databases, or to grant permissions to already existing users. A user cannot be removed until all the privileges have been stripped from the user. In such situation, if the module tries to remove the user it will fail. To avoid this from happening the fail_on_user option signals the module to try to remove the user, but if not possible keep going; the module will report if changes happened and separately if the user was removed or not.

Options

Note: Requires psycopg2

Examples

```
# Create django user and grant access to database and products table
- postgresql_user: db=acme name=django password=ceec4eif7ya priv=CONNECT/products:ALL

# Create rails user, grant privilege to create other databases and demote rails from super user statu
- postgresql_user: name=rails password=secret role_attr_flags=CREATEDB,NOSUPERUSER

# Remove test user privileges from acme
- postgresql_user: db=acme name=test priv=ALL/products:ALL state=absent fail_on_user=no

# Remove test user from test database and the cluster
- postgresql_user: db=test name=test priv=ALL state=absent

# Example privileges string format
INSERT,UPDATE/table:SELECT/anothertable:ALL

# Remove an existing user's password
- postgresql_user: db=test user=test password=NULL
```

Note: The default authentication assumes that you are either logging in as or sudo'ing to the postgres account on the host.

Note: This module uses psycopg2, a Python PostgreSQL database adapter. You must ensure that psycopg2 is installed on the host before using this module. If the remote host is the PostgreSQL server (which is the default case), then PostgreSQL must also be installed on the remote host. For Ubuntu-based systems, install the postgresql, libpq-dev, and python-psycopg2 packages on the remote host before using this module.

Note: If you specify PUBLIC as the user, then the privilege changes will apply to all users. You may not specify password or role_attr_flags when the PUBLIC user is specified.

redis - Various redis commands, slave and flush

Author Xabier Larrakoetxea

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Unified utility to interact with redis instances. 'slave' Sets a redis instance in slave or master mode. 'flush' Flushes all the instance or a specified db.

Options

Note: Requires redis

Examples

```
# Set local redis instance to be slave of melee.island on port 6377
- redis: command=slave master_host=melee.island master_port=6377

# Deactivate slave mode
- redis: command=slave slave_mode=master

# Flush all the redis db
- redis: command=flush flush_mode=all

# Flush only one db in a redis instance
- redis: command=flush db=1 flush_mode=db
```

Note: Requires the redis-py Python package on the remote host. You can install it with pip (pip install redis) or with a package manager. https://github.com/andymccurdy/redis-py

Note: If the redis master instance we are making slave of is password protected this needs to be in the redis.conf in the masterauth variable

riak - This module handles some common Riak operations

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module can be used to join nodes to a cluster, check the status of the cluster.

Options

Examples

```
# Join's a Riak node to another node
- riak: command=join target_node=riak@10.1.1.1

# Wait for handoffs to finish.  Use with async and poll.
- riak: wait_for_handoffs=yes

# Wait for riak_kv service to startup
- riak: wait_for_service=kv
```

1.6.5 Files Modules

acl - Sets and retrieves file ACL information.

Author Brian Coca

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Sets and retrieves file ACL information.

Options

Examples

```
# Grant user Joe read access to a file
- acl: name=/etc/foo.conf entity=joe etype=user permissions="r" state=present

# Removes the acl for Joe on a specific file
- acl: name=/etc/foo.conf entity=joe etype=user state=absent

# Sets default acl for joe on foo.d
- acl: name=/etc/foo.d entity=joe etype=user permissions=rw default=yes state=present

# Same as previous but using entry shorthand
- acl: name=/etc/foo.d entrty="default:user:joe:rw-" state=present

# Obtain the acl for a specific file
- acl: name=/etc/foo.conf
  register: acl_info
```

Note: The "acl" module requires that acls are enabled on the target filesystem and that the setfacl and getfacl binaries are installed.

assemble - Assembles a configuration file from fragments

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Assembles a configuration file from fragments. Often a particular program will take a single configuration file and does not support a `conf.d` style structure where it is easy to build up the configuration from multiple sources. `assemble` will take a directory of files that can be local or have already been transferred to the system, and concatenate them together to produce a destination file. Files are assembled in string sorting order. Puppet calls this idea *fragments*.

Options

Examples

```
# Example from Ansible Playbooks
- assemble: src=/etc/someapp/fragments dest=/etc/someapp/someapp.conf

# When a delimiter is specified, it will be inserted in between each fragment
- assemble: src=/etc/someapp/fragments dest=/etc/someapp/someapp.conf delimiter='### START FRAGMENT
```

copy - Copies files to remote locations.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

The `copy` module copies a file on the local box to remote locations.

Options

Examples

```
# Example from Ansible Playbooks
- copy: src=/srv/myfiles/foo.conf dest=/etc/foo.conf owner=foo group=foo mode=0644

# Copy a new "ntp.conf file into place, backing up the original if it differs from the copied versio
- copy: src=/mine/ntp.conf dest=/etc/ntp.conf owner=root group=root mode=644 backup=yes

# Copy a new "sudoers" file into place, after passing validation with visudo
- copy: src=/mine/sudoers dest=/etc/sudoers validate='visudo -cf %s'
```

Note: The "copy" module recursively copy facility does not scale to lots (>hundreds) of files. For alternative, see synchronize module, which is a wrapper around rsync.

fetch - Fetches a file from remote nodes

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This module works like `copy`, but in reverse. It is used for fetching files from remote machines and storing them locally in a file tree, organized by hostname. Note that this module is written to transfer log files that might not be present, so a missing remote file won't be an error unless fail_on_missing is set to 'yes'.

Options

Examples

```
# Store file into /tmp/fetched/host.example.com/tmp/somefile
- fetch: src=/tmp/somefile dest=/tmp/fetched

# Specifying a path directly
- fetch: src=/tmp/somefile dest=/tmp/prefix-{{ ansible_hostname }} flat=yes

# Specifying a destination path
- fetch: src=/tmp/uniquefile dest=/tmp/special/ flat=yes

# Storing in a path relative to the playbook
- fetch: src=/tmp/uniquefile dest=special/prefix-{{ ansible_hostname }} flat=yes
```

file - Sets attributes of files

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

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 `assemble`.

Options

Examples

```
- 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
```

Note: See also `copy`, `template`, `assemble`

ini_file - Tweak settings in INI files

Author Jan-Piet Mens

- Synopsis
- Options
- Examples

Synopsis

Manage (add, remove, change) individual settings in an INI-style file without having to manage the file as a whole with, say, `template` or `assemble`. Adds missing sections if they don't exist. Comments are discarded when the source file is read, and therefore will not show up in the destination file.

Options

Note: Requires ConfigParser

Examples

```
# Ensure "fav=lemonade is in section "[drinks]" in specified file
- ini_file: dest=/etc/conf section=drinks option=fav value=lemonade mode=0600 backup=yes

- ini_file: dest=/etc/anotherconf
            section=drinks
            option=temperature
            value=cold
            backup=yes
```

Note: While it is possible to add an *option* without specifying a *value*, this makes no sense.

Note: A section named `default` cannot be added by the module, but if it exists, individual options within the section can be updated. (This is a limitation of Python's *ConfigParser*.) Either use `template` to create a base INI file with a `[default]` section, or use `lineinfile` to add the missing line.

lineinfile - Ensure a particular line is in a file, or replace an existing line using a back-referenced regular expression.

Author Daniel Hokka Zakrisson

- Synopsis
- Options
- Examples

Synopsis

This module will search a file for a line, and ensure that it is present or absent. This is primarily useful when you want to change a single line in a file only. For other cases, see the `copy` or `template` modules.

Options

Examples

```
- lineinfile: dest=/etc/selinux/config regexp=^SELINUX= line=SELINUX=disabled

- lineinfile: dest=/etc/sudoers state=absent regexp="^%wheel"

- lineinfile: dest=/etc/hosts regexp='^127\.0\.0\.1' line='127.0.0.1 localhost' owner=root group=root

- lineinfile: dest=/etc/httpd/conf/httpd.conf regexp="^Listen " insertafter="^#Listen " line="Listen

- lineinfile: dest=/etc/services regexp="^# port for http" insertbefore="^www.*80/tcp" line="# port

# Add a line to a file if it does not exist, without passing regexp
- lineinfile: dest=/tmp/testfile line="192.168.1.99 foo.lab.net foo"

# Fully quoted because of the ': ' on the line. See the Gotchas in the YAML docs.
- lineinfile: "dest=/etc/sudoers state=present regexp='^%wheel' line='%wheel ALL=(ALL) NOPASSWD: ALL'

- lineinfile: dest=/opt/jboss-as/bin/standalone.conf regexp='^(.*)Xms(\d+)m(.*)$' line='\1Xms${xms}m

# Validate a the sudoers file before saving
- lineinfile: dest=/etc/sudoers state=present regexp='^%ADMIN ALL\=' line='%ADMIN ALL=(ALL) NOPASSWD
```

stat - retrieve file or file system status

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Retrieves facts for a file similar to the linux/unix 'stat' command.

Options

Examples

```
# Obtain the stats of /etc/foo.conf, and check that the file still belongs
# to 'root'. Fail otherwise.
- stat: path=/etc/foo.conf
  register: st
- fail: msg="Whoops! file ownership has changed"
  when: st.stat.pw_name != 'root'

# Determine if a path exists and is a directory.  Note we need to test
# both that p.stat.isdir actually exists, and also that it's set to true.
- stat: path=/path/to/something
  register: p
- debug: msg="Path exists and is a directory"
  when: p.stat.isdir is defined and p.stat.isdir == true
```

synchronize - Uses rsync to make synchronizing file paths in your playbooks quick and easy.

Author Timothy Appnel

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This is a wrapper around rsync. Of course you could just use the command action to call rsync yourself, but you also have to add a fair number of boilerplate options and host facts. You still may need to call rsync directly via `command` or `shell` depending on your use case. The synchronize action is meant to do common things with `rsync` easily. It does not provide access to the full power of rsync, but does make most invocations easier to follow.

Options

Examples

```
# Synchronization of src on the control machine to dest on the remote hosts
synchronize: src=some/relative/path dest=/some/absolute/path

# Synchronization without any --archive options enabled
synchronize: src=some/relative/path dest=/some/absolute/path archive=no
```

```
# Synchronization with --archive options enabled except for --recursive
synchronize: src=some/relative/path dest=/some/absolute/path recursive=no

# Synchronization without --archive options enabled except use --links
synchronize: src=some/relative/path dest=/some/absolute/path archive=no links=yes

# Synchronization of two paths both on the control machine
local_action: synchronize src=some/relative/path dest=/some/absolute/path

# Synchronization of src on the inventory host to the dest on the localhost in
pull mode
synchronize: mode=pull src=some/relative/path dest=/some/absolute/path

# Synchronization of src on delegate host to dest on the current inventory host
synchronize: >
    src=some/relative/path dest=/some/absolute/path
    delegate_to: delegate.host

# Synchronize and delete files in dest on the remote host that are not found in src of localhost.
synchronize: src=some/relative/path dest=/some/absolute/path delete=yes

# Synchronize using an alternate rsync command
synchronize: src=some/relative/path dest=/some/absolute/path rsync_path="sudo rsync"

# Example .rsync-filter file in the source directory
- var       # exclude any path whose last part is 'var'
- /var      # exclude any path starting with 'var' starting at the source directory
+ /var/conf # include /var/conf even though it was previously excluded
```

Note: Inspect the verbose output to validate the destination user/host/path are what was expected.

Note: The remote user for the dest path will always be the remote_user, not the sudo_user.

Note: Expect that dest=~/x will be ~<remote_user>/x even if using sudo.

Note: To exclude files and directories from being synchronized, you may add `.rsync-filter` files to the source directory.

template - Templates a file out to a remote server.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

Templates are processed by the Jinja2 templating language (http://jinja.pocoo.org/docs/) - documentation on the template formatting can be found in the Template Designer Documentation (http://jinja.pocoo.org/docs/templates/). Six additional variables can be used in templates: `ansible_managed` (configurable via the `defaults` section of `ansible.cfg`) contains a string which can be used to describe the template name, host, modification time of the template file and the owner uid, `template_host` contains the node name of the template's machine, `template_uid` the owner, `template_path` the absolute path of the template, `template_fullpath` is the absolute path of the template, and `template_run_date` is the date that the template was rendered. Note that including a string that uses a date in the template will resort in the template being marked 'changed' each time.

Options

Examples

```
# Example from Ansible Playbooks
- template: src=/mytemplates/foo.j2 dest=/etc/file.conf owner=bin group=wheel mode=0644

# Copy a new "sudoers file into place, after passing validation with visudo
- action: template src=/mine/sudoers dest=/etc/sudoers validate='visudo -cf %s'
```

Note: Since Ansible version 0.9, templates are loaded with `trim_blocks=True`.

Note: Also, you can override jinja2 settings by adding a special header to template file. i.e. `#jinja2:variable_start_string:'[%' , variable_end_string:'%]'` which changes the variable interpolation markers to [% var %] instead of {{ var }}. This is the best way to prevent evaluation of things that look like, but should not be Jinja2. raw/endraw in Jinja2 will not work as you expect because templates in Ansible are recursively evaluated.

---

unarchive - Copies an archive to a remote location and unpack it

Author Dylan Martin

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

The `unarchive` module copies an archive file from the local machine to a remote and unpacks it.

Options

Examples

```
# Example from Ansible Playbooks
- unarchive: src=foo.tgz dest=/var/lib/foo
```

Note: requires `tar`/`unzip` command on target host

Note: can handle *gzip*, *bzip2* and *xz* compressed as well as uncompressed tar files

Note: detects type of archive automatically

Note: uses tar's `--diff arg` to calculate if changed or not. If this `arg` is not supported, it will always unpack the archive

Note: does not detect if a .zip file is different from destination - always unzips

Note: existing files/directories in the destination which are not in the archive are not touched. This is the same behavior as a normal archive extraction

Note: existing files/directories in the destination which are not in the archive are ignored for purposes of deciding if the archive should be unpacked or not

xattr - set/retrieve extended attributes

Author Brian Coca

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages filesystem user defined extended attributes, requires that they are enabled on the target filesystem and that the setfattr/getfattr utilities are present.

Options

Examples

```
# Obtain the extended attributes  of /etc/foo.conf
- xattr: name=/etc/foo.conf

# Sets the key 'foo' to value 'bar'
- xattr: path=/etc/foo.conf key=user.foo value=bar

# Removes the key 'foo'
- xattr: name=/etc/foo.conf key=user.foo state=absent
```

1.6.6 Internal Modules

async_status - Obtain status of asynchronous task

Author Michael DeHaan

- Synopsis
- Options

Synopsis

This module gets the status of an asynchronous task.

Options

Note: See also http://docs.ansible.com/playbooks_async.html

1.6.7 Inventory Modules

add_host - add a host (and alternatively a group) to the ansible-playbook in-memory inventory

Author Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

Use variables to create new hosts and groups in inventory for use in later plays of the same playbook. Takes variables so you can define the new hosts more fully.

Options

Examples

```
# add host to group 'just_created' with variable foo=42
- add_host: name={{ ip_from_ec2 }} groups=just_created foo=42

# add a host with a non-standard port local to your machines
- add_host: name={{ new_ip }}:{{ new_port }}

# add a host alias that we reach through a tunnel
- add_host: hostname={{ new_ip }}
            ansible_ssh_host={{ inventory_hostname }}
            ansible_ssh_port={{ new_port }}
```

group_by - Create Ansible groups based on facts

Author Jeroen Hoekx

- Synopsis
- Options
- Examples

Synopsis

Use facts to create ad-hoc groups that can be used later in a playbook.

Options

Examples

```
# Create groups based on the machine architecture
-  group_by: key=machine_{{ ansible_machine }}
# Create groups like 'kvm-host'
-  group_by: key=virt_{{ ansible_virtualization_type }}_{{ ansible_virtualization_role }}
```

Note: Spaces in group names are converted to dashes '-'.

1.6.8 Messaging Modules

rabbitmq_parameter - Adds or removes parameters to RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage dynamic, cluster-wide parameters for RabbitMQ

Options

Examples

```
# Set the federation parameter 'local_username' to a value of 'guest' (in quotes)
- rabbitmq_parameter: component=federation
                      name=local-username
                      value='"guest"'
                      state=present
```

rabbitmq_plugin - Adds or removes users to RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Enables or disables RabbitMQ plugins

Options

Examples

```
# Enables the rabbitmq_management plugin
- rabbitmq_plugin: names=rabbitmq_management state=enabled
```

rabbitmq_policy - Manage the state of policies in RabbitMQ.

Author John Dewey

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

Manage the state of a virtual host in RabbitMQ.

Options

Examples

```
- name: ensure the default vhost contains the HA policy via a dict
  rabbitmq_policy: name=HA pattern='.*'
  args:
    tags:
      "ha-mode": all

- name: ensure the default vhost contains the HA policy
  rabbitmq_policy: name=HA pattern='.*' tags="ha-mode=all"
```

rabbitmq_user - Adds or removes users to RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Add or remove users to RabbitMQ and assign permissions

Options

Examples

```
# Add user to server and assign full access control
- rabbitmq_user: user=joe
                 password=changeme
                 vhost=/
                 configure_priv=.*
                 read_priv=.*
```

```
write_priv=.*
state=present
```

rabbitmq_vhost - Manage the state of a virtual host in RabbitMQ

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage the state of a virtual host in RabbitMQ

Options

Examples

```
# Ensure that the vhost /test exists.
- rabbitmq_vhost: name=/test state=present
```

1.6.9 Monitoring Modules

airbrake_deployment - Notify airbrake about app deployments

Author Bruce Pennypacker

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Notify airbrake about app deployments (see http://help.airbrake.io/kb/api-2/deploy-tracking)

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- airbrake_deployment: token=AAAAAA
                       environment='staging'
                       user='ansible'
                       revision=4.2
```

boundary_meter - Manage boundary meters

Author curtis@serverascode.com

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

This module manages boundary meters

Options

Note: Requires Boundary API access

Note: Requires bprobe is required to send data, but not to register a meter

Note: Requires Python urllib2

Examples

```
- name: Create meter
  boundary_meter: apiid=AAAAAA api_key=BBBBBB state=present name={{ inventory_hostname }}"

- name: Delete meter
  boundary_meter: apiid=AAAAAA api_key=BBBBBB state=absent name={{ inventory_hostname }}"
```

Note: This module does not yet support boundary tags.

datadog_event - Posts events to DataDog service

Author Artras 'arturaz' Šlajus <x11@arturaz.net>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Allows to post events to DataDog (www.datadoghq.com) service. Uses http://docs.datadoghq.com/api/#events API.

Options

Note: Requires urllib2

Examples

```
# Post an event with low priority
datadog_event: title="Testing from ansible" text="Test!" priority="low"
               api_key="6873258723457823548234234234"
# Post an event with several tags
datadog_event: title="Testing from ansible" text="Test!"
               api_key="6873258723457823548234234234"
               tags=aa,bb,cc
```

monit - Manage the state of a program monitored via Monit

Author Darryl Stoflet

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage the state of a program monitored via *Monit*

Options

Examples

```
# Manage the state of program "httpd" to be in "started" state.
- monit: name=httpd state=started
```

nagios - Perform common tasks in Nagios related to downtime and notifications.

Author Tim Bielawa

- Synopsis
- Options
- Examples

Synopsis

The `nagios` module has two basic functions: scheduling downtime and toggling alerts for services or hosts. All actions require the *host* parameter to be given explicitly. In playbooks you can use the `{{inventory_hostname}}` variable to refer to the host the playbook is currently running on. You can specify multiple services at once by separating them with commas, .e.g., `services=httpd,nfs,puppet`. When specifying what service to handle there is a special service value, *host*, which will handle alerts/downtime for the *host itself*, e.g., `service=host`. This keyword may not be given with other services at the same time. *Setting alerts/downtime for a host does not affect alerts/downtime for any of the services running on it.* To schedule downtime for all services on particular host use keyword "all", e.g., `service=all`. When using the `nagios` module you will need to specify your Nagios server using the `delegate_to` parameter.

Options

Note: Requires Nagios

Examples

```
# set 30 minutes of apache downtime
- nagios: action=downtime minutes=30 service=httpd host={{ inventory_hostname }}

# schedule an hour of HOST downtime
- nagios: action=downtime minutes=60 service=host host={{ inventory_hostname }}

# schedule downtime for ALL services on HOST
- nagios: action=downtime minutes=45 service=all host={{ inventory_hostname }}

# schedule downtime for a few services
- nagios: action=downtime services=frob,foobar,qeuz host={{ inventory_hostname }}

# enable SMART disk alerts
- nagios: action=enable_alerts service=smart host={{ inventory_hostname }}

# "two services at once: disable httpd and nfs alerts"
- nagios: action=disable_alerts service=httpd,nfs host={{ inventory_hostname }}

# disable HOST alerts
- nagios: action=disable_alerts service=host host={{ inventory_hostname }}

# silence ALL alerts
- nagios: action=silence host={{ inventory_hostname }}
```

```
# unsilence all alerts
- nagios: action=unsilence host={{ inventory_hostname }}

# SHUT UP NAGIOS
- nagios: action=silence_nagios

# ANNOY ME NAGIOS
- nagios: action=unsilence_nagios

# command something
- nagios: action=command command='DISABLE_FAILURE_PREDICTION'
```

newrelic_deployment - Notify newrelic about app deployments

Author Matt Coddington

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Notify newrelic about app deployments (see http://newrelic.github.io/newrelic_api/NewRelicApi/Deployment.html)

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- newrelic_deployment: token=AAAAAA
                       app_name=myapp
                       user='ansible deployment'
                       revision=1.0
```

pagerduty - Create PagerDuty maintenance windows

Author Justin Johns

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module will let you create PagerDuty maintenance windows

Options

Note: Requires PagerDuty API access

Examples

```
# List ongoing maintenance windows.
- pagerduty: name=companyabc user=example@example.com passwd=password123 state=ongoing

# Create a 1 hour maintenance window for service FOO123.
- pagerduty: name=companyabc
             user=example@example.com
             passwd=password123
             state=running
             service=FOO123

# Create a 4 hour maintenance window for service FOO123 with the description "deployment".
- pagerduty: name=companyabc
             user=example@example.com
             passwd=password123
             state=running
             service=FOO123
             hours=4
             desc=deployment
```

Note: This module does not yet have support to end maintenance windows.

pingdom - Pause/unpause Pingdom alerts

Author Justin Johns

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module will let you pause/unpause Pingdom alerts

Options

Note: Requires This pingdom python library: https://github.com/mbabineau/pingdom-python

Examples

```
# Pause the check with the ID of 12345.
- pingdom: uid=example@example.com
           passwd=password123
           key=apipassword123
           checkid=12345
           state=paused

# Unpause the check with the ID of 12345.
- pingdom: uid=example@example.com
           passwd=password123
           key=apipassword123
           checkid=12345
           state=running
```

Note: This module does not yet have support to add/remove checks.

1.6.10 Net Infrastructure Modules

arista_interface - Manage physical Ethernet interfaces

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage physical Ethernet interface resources on Arista EOS network devices

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_interface module to manage resource
state.  Note that interface names must be the full interface name not shortcut
names (ie Ethernet, not Et1)

    tasks:
    - name: enable interface Ethernet 1
      action: arista_interface interface_id=Ethernet1 admin=up speed=10g duplex=full logging=true

    - name: set mtu on Ethernet 1
      action: arista_interface interface_id=Ethernet1 mtu=1600 speed=10g duplex=full logging=true

    - name: reset changes to Ethernet 1
      action: arista_interface interface_id=Ethernet1 admin=down mtu=1500 speed=10g duplex=full logg
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

arista_l2interface - Manage layer 2 interfaces

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage layer 2 interface resources on Arista EOS network devices

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_l2interface module to manage resource
state. Note that interface names must be the full interface name not shortcut
names (ie Ethernet, not Et1)

    tasks:
    - name: create switchport ethernet1 access port
      action: arista_l2interface interface_id=Ethernet1 logging=true

    - name: create switchport ethernet2 trunk port
      action: arista_l2interface interface_id=Ethernet2 vlan_tagging=enable logging=true

    - name: add vlans to red and blue switchport ethernet2
      action: arista_l2interface interface_id=Ethernet2 tagged_vlans=red,blue logging=true

    - name: set untagged vlan for Ethernet1
      action: arista_l2interface interface_id=Ethernet1 untagged_vlan=red logging=true

    - name: convert access to trunk
      action: arista_l2interface interface_id=Ethernet1 vlan_tagging=enable tagged_vlans=red,blue log

    - name: convert trunk to access
      action: arista_l2interface interface_id=Ethernet2 vlan_tagging=disable untagged_vlan=blue loggi

    - name: delete switchport ethernet1
      action: arista_l2interface interface_id=Ethernet1 state=absent logging=true
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

arista_lag - Manage port channel (lag) interfaces

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage port channel interface resources on Arista EOS network devices

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_lag module to manage resource
state.  Note that interface names must be the full interface name not shortcut
names (ie Ethernet, not Et1)

    tasks:
    - name: create lag interface
      action: arista_lag interface_id=Port-Channel1 links=Ethernet1,Ethernet2 logging=true

    - name: add member links
      action: arista_lag interface_id=Port-Channel1 links=Ethernet1,Ethernet2,Ethernet3 logging=true

    - name: remove member links
      action: arista_lag interface_id=Port-Channel1 links=Ethernet2,Ethernet3 logging=true

    - name: remove lag interface
      action: arista_lag interface_id=Port-Channel1 state=absent logging=true
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

arista_vlan - Manage VLAN resources

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage VLAN resources on Arista EOS network devices. This module requires the Netdev EOS extension to be installed in EOS. For detailed instructions for installing and using the Netdev module please see [link]

Options

Note: Requires Arista EOS 4.10

Note: Requires Netdev extension for EOS

Examples

```
Example playbook entries using the arista_vlan module to manage resource
state.

  tasks:
  - name: create vlan 999
    action: arista_vlan vlan_id=999 logging=true

  - name: create / edit vlan 999
    action: arista_vlan vlan_id=999 name=test logging=true

  - name: remove vlan 999
    action: arista_vlan vlan_id=999 state=absent logging=true
```

Note: Requires EOS 4.10 or later

Note: The Netdev extension for EOS must be installed and active in the available extensions (show extensions from the EOS CLI)

Note: See http://eos.aristanetworks.com for details

bigip_monitor_http - Manages F5 BIG-IP LTM http monitors

Author Serge van Ginderachter

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM monitors via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
- name: BIGIP F5 | Create HTTP Monitor
  local_action:
    module:             bigip_monitor_http
    state:              present
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ item.monitorname }}"
    send:               "{{ item.send }}"
    receive:            "{{ item.receive }}"
  with_items: f5monitors
- name: BIGIP F5 | Remove HTTP Monitor
  local_action:
    module:             bigip_monitor_http
    state:              absent
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ monitorname }}"
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

Note: Monitor API documentation: https://devcentral.f5.com/wiki/iControl.LocalLB__Monitor.ashx

bigip_monitor_tcp - Manages F5 BIG-IP LTM tcp monitors

Author Serge van Ginderachter

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM tcp monitors via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
- name: BIGIP F5 | Create TCP Monitor
  local_action:
    module:             bigip_monitor_tcp
    state:              present
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ item.monitorname }}"
    type:               tcp
    send:               "{{ item.send }}"
    receive:            "{{ item.receive }}"
  with_items: f5monitors-tcp
- name: BIGIP F5 | Create TCP half open Monitor
  local_action:
    module:             bigip_monitor_tcp
    state:              present
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ item.monitorname }}"
    type:               tcp
    send:               "{{ item.send }}"
    receive:            "{{ item.receive }}"
  with_items: f5monitors-halftcp
- name: BIGIP F5 | Remove TCP Monitor
  local_action:
    module:             bigip_monitor_tcp
    state:              absent
    server:             "{{ f5server }}"
    user:               "{{ f5user }}"
    password:           "{{ f5password }}"
    name:               "{{ monitorname }}"
  with_flattened:
  - f5monitors-tcp
  - f5monitors-halftcp
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

Note: Monitor API documentation: https://devcentral.f5.com/wiki/iControl.LocalLB__Monitor.ashx

---

bigip_node - Manages F5 BIG-IP LTM nodes

Author Matt Hite

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM nodes via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
## playbook task examples:

---
# file bigip-test.yml
# ...
- hosts: bigip-test
  tasks:
  - name: Add node
    local_action: >
      bigip_node
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      name="{{ ansible_default_ipv4["address"] }}"

# Note that the BIG-IP automatically names the node using the
# IP address specified in previous play's host parameter.
# Future plays referencing this node no longer use the host
# parameter but instead use the name parameter.
# Alternatively, you could have specified a name with the
# name parameter when state=present.

  - name: Modify node description
    local_action: >
      bigip_node
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      partition=matthite
      name="{{ ansible_default_ipv4["address"] }}"
      description="Our best server yet"
```

```
  - name: Delete node
    local_action: >
      bigip_node
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=absent
      partition=matthite
      name="{{ ansible_default_ipv4["address"] }}"
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

bigip_pool - Manages F5 BIG-IP LTM pools

Author Matt Hite

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manages F5 BIG-IP LTM pools via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
## playbook task examples:

---
# file bigip-test.yml
# ...
- hosts: localhost
  tasks:
  - name: Create pool
    local_action: >
      bigip_pool
      server=lb.mydomain.com
```

```
      user=admin
      password=mysecret
      state=present
      name=matthite-pool
      partition=matthite
      lb_method=least_connection_member
      slow_ramp_time=120

  - name: Modify load balancer method
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      name=matthite-pool
      partition=matthite
      lb_method=round_robin

- hosts: bigip-test
  tasks:
  - name: Add pool member
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      name=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80

  - name: Remove pool member from pool
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=absent
      name=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80

- hosts: localhost
  tasks:
  - name: Delete pool
    local_action: >
      bigip_pool
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=absent
      name=matthite-pool
      partition=matthite
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

bigip_pool_member - Manages F5 BIG-IP LTM pool members

Author Matt Hite

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages F5 BIG-IP LTM pool members via iControl SOAP API

Options

Note: Requires bigsuds

Examples

```
## playbook task examples:

---
# file bigip-test.yml
# ...
- hosts: bigip-test
  tasks:
  - name: Add pool member
    local_action: >
      bigip_pool_member
      server=lb.mydomain.com
      user=admin
      password=mysecret
      state=present
      pool=matthite-pool
      partition=matthite
      host="{{ ansible_default_ipv4["address"] }}"
      port=80
      description="web server"
      connection_limit=100
      rate_limit=50
      ratio=2
```

```
- name: Modify pool member ratio and description
  local_action: >
    bigip_pool_member
    server=lb.mydomain.com
    user=admin
    password=mysecret
    state=present
    pool=matthite-pool
    partition=matthite
    host="{{ ansible_default_ipv4["address"] }}"
    port=80
    ratio=1
    description="nginx server"

- name: Remove pool member from pool
  local_action: >
    bigip_pool_member
    server=lb.mydomain.com
    user=admin
    password=mysecret
    state=absent
    pool=matthite-pool
    partition=matthite
    host="{{ ansible_default_ipv4["address"] }}"
    port=80
```

Note: Requires BIG-IP software version >= 11

Note: F5 developed module 'bigsuds' required (see http://devcentral.f5.com)

Note: Best run as a local_action in your playbook

Note: Supersedes bigip_pool for managing pool members

dnsmadeeasy - Interface with dnsmadeeasy.com (a DNS hosting service).

Author Brice Burgess

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages DNS records via the v2 REST API of the DNS Made Easy service. It handles records only; there is no manipulation of domains or monitor/account support yet. See: http://www.dnsmadeeasy.com/services/rest-api/

Options

Note: Requires urllib

Note: Requires urllib2

Note: Requires hashlib

Note: Requires hmac

Examples

```
# fetch my.com domain records
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present
  register: response

# create / ensure the presence of a record
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present record_name="test" r

# update the previously created record
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present record_name="test" r

# fetch a specific record
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=present record_name="test"
  register: response

# delete a record / ensure it is absent
- dnsmadeeasy: account_key=key account_secret=secret domain=my.com state=absent record_name="test"
```

Note: The DNS Made Easy service requires that machines interacting with the API have the proper time and timezone set. Be sure you are within a few seconds of actual time by using NTP.

Note: This module returns record(s) in the "result" element when 'state' is set to 'present'. This value can be be registered and used in your playbooks.

netscaler - Manages Citrix NetScaler entities

Author Nandor Sivok

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages Citrix NetScaler server and service entities.

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
# Disable the server
ansible host -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass"

# Enable the server
ansible host -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass action=enable"

# Disable the service local:8080
ansible host -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass name=local:8080
```

openvswitch_bridge - Manage Open vSwitch bridges

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage Open vSwitch bridges

Options

Note: Requires ovs-vsctl

Examples

```
# Create a bridge named br-int
- openvswitch_bridge: bridge=br-int state=present
```

openvswitch_port - Manage Open vSwitch ports

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manage Open vSwitch ports

Options

Note: Requires ovs-vsctl

Examples

```
# Creates port eth2 on bridge br-ex
- openvswitch_port: bridge=br-ex port=eth2 state=present
```

1.6.11 Network Modules

get_url - Downloads files from HTTP, HTTPS, or FTP to node

Author Jan-Piet Mens

- Synopsis
- Options
- Examples

Synopsis

Downloads files from HTTP, HTTPS, or FTP to the remote server. The remote server *must* have direct access to the remote resource. By default, if an environment variable `<protocol>_proxy` is set on the target host, requests will be sent through that proxy. This behaviour can be overridden by setting a variable for this task (see setting the environment), or by using the use_proxy option.

Options

Note: Requires urllib2

Note: Requires urlparse

Examples

```
- name: download foo.conf
  get_url: url=http://example.com/path/file.conf dest=/etc/foo.conf mode=0440

- name: download file with sha256 check
  get_url: url=http://example.com/path/file.conf dest=/etc/foo.conf sha256sum=b5bb9d8014a0f9b1d61e21e
```

Note: This module doesn't yet support configuration for proxies.

slurp - Slurps a file from remote nodes

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This module works like `fetch`. It is used for fetching a base64- encoded blob containing the data in a remote file.

Options

Examples

```
ansible host -m slurp -a 'src=/tmp/xx'
   host | success >> {
      "content": "aGVsbG8gQW5zaWJsZSB3b3JsZAo=",
      "encoding": "base64"
   }
```

Note: See also: `fetch`

uri - Interacts with webservices

Author Romeo Theriault

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Interacts with HTTP and HTTPS web services and supports Digest, Basic and WSSE HTTP authentication mechanisms.

Options

Note: Requires urlparse

Note: Requires httplib2

Examples

```
# Check that you can connect (GET) to a page and it returns a status 200
- uri: url=http://www.example.com

# Check that a page returns a status 200 and fail if the word AWESOME is not in the page contents.
- action: uri url=http://www.example.com return_content=yes
  register: webpage

- action: fail
  when: 'AWESOME' not in "{{ webpage.content }}"


# Create a JIRA issue.
- action: >
        uri url=https://your.jira.example.com/rest/api/2/issue/
        method=POST user=your_username password=your_pass
        body="{{ lookup('file','issue.json') }}" force_basic_auth=yes
        status_code=201 HEADER_Content-Type="application/json"

- action: >
        uri url=https://your.form.based.auth.examle.com/index.php
        method=POST body="name=your_username&password=your_password&enter=Sign%20in"
        status_code=302 HEADER_Content-Type="application/x-www-form-urlencoded"
  register: login

# Login to a form based webpage, then use the returned cookie to
# access the app in later tasks.
- action: uri url=https://your.form.based.auth.example.com/dashboard.php
            method=GET return_content=yes HEADER_Cookie="{{login.set_cookie}}"
```

1.6.12 Notification Modules

campfire - Send a message to Campfire

Author Adam Garside <adam.garside@gmail.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to Campfire. Messages with newlines will result in a "Paste" message being sent.

Options

Note: Requires urllib2

Note: Requires cgi

Examples

```
- campfire: subscription=foo token=12345 room=123 msg="Task completed."

- campfire: subscription=foo token=12345 room=123 notify=loggins
        msg="Task completed ... with feeling."
```

flowdock - Send a message to a flowdock

Author Matt Coddington

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to a flowdock team inbox or chat using the push API (see https://www.flowdock.com/api/team-inbox and https://www.flowdock.com/api/chat)

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- flowdock: type=inbox
            token=AAAAAA
            from_address=user@example.com
            source='my cool app'
            msg='test from ansible'
            subject='test subject'

- flowdock: type=chat
            token=AAAAAA
            external_user_name=testuser
            msg='test from ansible'
            tags=tag1,tag2,tag3
```

grove - Sends a notification to a grove.io channel

Author Jonas Pfenniger <zimbatm@zimbatm.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

The `grove` module sends a message for a service to a Grove.io channel.

Options

Examples

```
- grove: >
    channel_token=6Ph62VBBJOccmtTPZbubiPzdrhipZXtg
    service=my-app
    message=deployed {{ target }}
```

hipchat - Send a message to hipchat

Author WAKAYAMA Shirou

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to hipchat

Options

Note: Requires urllib

Note: Requires urllib2

Examples

```
- hipchat: token=AAAAAA room=notify msg="Ansible task finished"
```

irc - Send a message to an IRC channel

Author Jan-Piet Mens, Matt Martz

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to an IRC channel. This is a very simplistic implementation.

Options

Note: Requires socket

Examples

```
- irc: server=irc.example.net channel="#t1" msg="Hello world"

- local_action: irc port=6669
                channel="#t1"
                msg="All finished at {{ ansible_date_time.iso8601 }}"
                color=red
                nick=ansibleIRC
```

jabber - Send a message to jabber user or chat room

Author Brian Coca

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Send a message to jabber

Options

Note: Requires xmpp

Examples

```
# send a message to a user
- jabber: user=mybot@example.net
          password=secret
          to=friend@example.net
          msg="Ansible task finished"

# send a message to a room
- jabber: user=mybot@example.net
          password=secret
          to=mychaps@conference.example.net/ansiblebot
          msg="Ansible task finished"

# send a message, specifying the host and port
- jabber user=mybot@example.net
         host=talk.example.net
         port=5223
         password=secret
         to=mychaps@example.net
         msg="Ansible task finished"
```

mail - Send an email

Author Dag Wieers

- Synopsis
- Options
- Examples

Synopsis

This module is useful for sending emails from playbooks. One may wonder why automate sending emails? In complex environments there are from time to time processes that cannot be automated, either because you lack the authority to make it so, or because not everyone agrees to a common approach. If you cannot automate a specific step, but the step is non-blocking, sending out an email to the responsible party to make him perform his part of the bargain is an elegant way to put the responsibility in someone else's lap. Of course sending out a mail can be equally useful as a way to notify one or more people in a team that a specific action has been (successfully) taken.

Options

Examples

```
# Example playbook sending mail to root
- local_action: mail msg='System {{ ansible_hostname }} has been successfully provisioned.'

# Send e-mail to a bunch of users, attaching files
- local_action: mail
                host='127.0.0.1'
                port=2025
                subject="Ansible-report"
                body="Hello, this is an e-mail. I hope you like it ;-)"
                from="jane@example.net (Jane Jolie)"
                to="John Doe <j.d@example.org>, Suzie Something <sue@example.com>"
                cc="Charlie Root <root@localhost>"
                attach="/etc/group /tmp/pavatar2.png"
                headers=Reply-To=john@example.com|X-Special="Something or other"
                charset=utf8
```

mqtt - Publish a message on an MQTT topic for the IoT

Author Jan-Piet Mens

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Publish a message on an MQTT topic.

Options

Note: Requires mosquitto

Examples

```
- local_action: mqtt
              topic=service/ansible/{{ ansible_hostname }}
              payload="Hello at {{ ansible_date_time.iso8601 }}"
              qos=0
              retain=false
              client_id=ans001
```

Note: This module requires a connection to an MQTT broker such as Mosquitto http://mosquitto.org and the `mosquitto` Python module (http://mosquitto.org/python).

osx_say - Makes an OSX computer to speak.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

makes an OS computer speak! Amuse your friends, annoy your coworkers!

Options

Note: Requires say

Examples

```
- local_action: osx_say msg="{{inventory_hostname}} is all done" voice=Zarvox
```

Note: If you like this module, you may also be interested in the osx_say callback in the plugins/ directory of the source checkout.

1.6.13 Packaging Modules

apt - Manages apt-packages

Author Matthew Williams

- Synopsis
- Options
- Examples

Synopsis

Manages *apt* packages (such as for Debian/Ubuntu).

Options

Note: Requires python-apt

Note: Requires aptitude

Examples

```
# Update repositories cache and install "foo" package
- apt: pkg=foo update_cache=yes

# Remove "foo" package
- apt: pkg=foo state=absent

# Install the package "foo"
- apt: pkg=foo state=present

# Install the version '1.00' of package "foo"
- apt: pkg=foo=1.00 state=present

# Update the repository cache and update package "nginx" to latest version using default release sque
- apt: pkg=nginx state=latest default_release=squeeze-backports update_cache=yes

# Install latest version of "openjdk-6-jdk" ignoring "install-recommends"
- apt: pkg=openjdk-6-jdk state=latest install_recommends=no

# Update all packages to the latest version
- apt: upgrade=dist

# Run the equivalent of "apt-get update" as a separate step
- apt: update_cache=yes

# Only run "update_cache=yes" if the last one is more than more than 3600 seconds ago
- apt: update_cache=yes cache_valid_time=3600

# Pass options to dpkg on run
- apt: upgrade=dist update_cache=yes dpkg_options='force-confold,force-confdef'
```

Note: Three of the upgrade modes (`full`, `safe` and its alias `yes`) require `aptitude`, otherwise `apt-get` suffices.

apt_key - Add or remove an apt key

Author Jayson Vantuyl & others

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Add or remove an *apt* key, optionally downloading it

Options

Examples

```
# Add an Apt signing key, uses whichever key is at the URL
- apt_key: url=https://ftp-master.debian.org/keys/archive-key-6.0.asc state=present

# Add an Apt signing key, will not download if present
- apt_key: id=473041FA url=https://ftp-master.debian.org/keys/archive-key-6.0.asc state=present

# Remove an Apt signing key, uses whichever key is at the URL
- apt_key: url=https://ftp-master.debian.org/keys/archive-key-6.0.asc state=absent

# Remove a Apt specific signing key, leading 0x is valid
- apt_key: id=0x473041FA state=absent

# Add a key from a file on the Ansible server
- apt_key: data="{{ lookup('file', 'apt.gpg') }}" state=present

# Add an Apt signing key to a specific keyring file
- apt_key: id=473041FA url=https://ftp-master.debian.org/keys/archive-key-6.0.asc keyring=/etc/apt/t
```

Note: doesn't download the key unless it really needs it

Note: as a sanity check, downloaded key id must match the one specified

Note: best practice is to specify the key id and the url

apt_repository - Add and remove APT repositores

Author Alexander Saltanov

- Synopsis
- Options
- Examples

Synopsis

Add or remove an APT repositories in Ubuntu and Debian.

Options

Note: Requires python-apt

Note: Requires python-pycurl

Examples

```
# Add specified repository into sources list.
apt_repository: repo='deb http://archive.canonical.com/ubuntu hardy partner' state=present

# Add source repository into sources list.
apt_repository: repo='deb-src http://archive.canonical.com/ubuntu hardy partner' state=present

# Remove specified repository from sources list.
apt_repository: repo='deb http://archive.canonical.com/ubuntu hardy partner' state=absent

# On Ubuntu target: add nginx stable repository from PPA and install its signing key.
# On Debian target: adding PPA is not available, so it will fail immediately.
apt_repository: repo='ppa:nginx/stable'
```

Note: This module works on Debian and Ubuntu and requires `python-apt` and `python-pycurl` packages.

Note: This module supports Debian Squeeze (version 6) as well as its successors.

Note: This module treats Debian and Ubuntu distributions separately. So PPA could be installed only on Ubuntu machines.

easy_install - Installs Python libraries

Author Matt Wright

- Synopsis
- Options
- Examples

Synopsis

Installs Python libraries, optionally in a *virtualenv*

Options

Note: Requires virtualenv

Examples

```
# Examples from Ansible Playbooks
- easy_install: name=pip

# Install Bottle into the specified virtualenv.
- easy_install: name=bottle virtualenv=/webapps/myapp/venv
```

Note: Please note that the `easy_install` module can only install Python libraries. Thus this module is not able to remove libraries. It is generally recommended to use the `pip` module which you can first install using `easy_install`.

Note: Also note that *virtualenv* must be installed on the remote host if the `virtualenv` parameter is specified.

gem - Manage Ruby gems

Author Johan Wiren

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage installation and uninstallation of Ruby gems.

Options

Examples

```
# Installs version 1.0 of vagrant.
- gem: name=vagrant version=1.0 state=present

# Installs latest available version of rake.
- gem: name=rake state=latest
```

```
# Installs rake version 1.0 from a local gem on disk.
- gem: name=rake gem_source=/path/to/gems/rake-1.0.gem state=present
```

homebrew - Package manager for Homebrew

Author Andrew Dunham

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Manages Homebrew packages

Options

Examples

```
- homebrew: name=foo state=present
- homebrew: name=foo state=present update_homebrew=yes
- homebrew: name=foo state=absent
- homebrew: name=foo,bar state=absent
- homebrew: name=foo state=present install_options=with-baz,enable-debug
```

macports - Package manager for MacPorts

Author Jimmy Tang

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages MacPorts packages

Options

Examples

```
- macports: name=foo state=present
- macports: name=foo state=present update_cache=yes
- macports: name=foo state=absent
- macports: name=foo state=active
- macports: name=foo state=inactive
```

npm - Manage node.js packages with npm

Author Chris Hoffman

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage node.js packages with Node Package Manager (npm)

Options

Examples

```
description: Install "coffee-script" node.js package.
- npm: name=coffee-script path=/app/location

description: Install "coffee-script" node.js package on version 1.6.1.
- npm: name=coffee-script version=1.6.1 path=/app/location

description: Install "coffee-script" node.js package globally.
- npm: name=coffee-script global=yes

description: Remove the globally package "coffee-script".
- npm: name=coffee-script global=yes state=absent

description: Install packages based on package.json.
- npm: path=/app/location

description: Update packages based on package.json to their latest version.
- npm: path=/app/location state=latest

description: Install packages based on package.json using the npm installed with nvm v0.10.1.
- npm: path=/app/location executable=/opt/nvm/v0.10.1/bin/npm state=present
```

openbsd_pkg - Manage packages on OpenBSD.

Author Patrik Lundin

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage packages on OpenBSD using the pkg tools.

Options

Examples

```
# Make sure nmap is installed
- openbsd_pkg: name=nmap state=present

# Make sure nmap is the latest version
- openbsd_pkg: name=nmap state=latest

# Make sure nmap is not installed
- openbsd_pkg: name=nmap state=absent
```

opkg - Package manager for OpenWrt

Author Patrick Pelletier

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages OpenWrt packages

Options

Examples

```
- opkg: name=foo state=present
- opkg: name=foo state=present update_cache=yes
- opkg: name=foo state=absent
- opkg: name=foo,bar state=absent
```

pacman - Package manager for Archlinux

Author Afterburn

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Manages Archlinux packages

Options

Examples

```
# Install package foo
- pacman: name=foo state=installed

# Remove package foo
- pacman: name=foo state=absent

# Remove packages foo and bar
- pacman: name=foo,bar state=absent

# Recursively remove package baz
- pacman: name=baz state=absent recurse=yes

# Update the package database (pacman -Syy) and install bar (bar will be the updated if a newer versi
- pacman: name=bar, state=installed, update_cache=yes
```

pip - Manages Python library dependencies.

Author Matt Wright

- Synopsis
- Options
- Examples

Synopsis

Manage Python library dependencies. To use this module, one of the following keys is required: `name` or `requirements`.

Options

Note: Requires virtualenv

Note: Requires pip

Examples

```
# Install (Bottle) python package.
- pip: name=bottle

# Install (Bottle) python package on version 0.11.
- pip: name=bottle version=0.11

# Install (MyApp) using one of the remote protocols (bzr+,hg+,git+,svn+). You do not have to supply '
- pip: name='svn+http://myrepo/svn/MyApp#egg=MyApp'

# Install (Bottle) into the specified (virtualenv), inheriting none of the globally installed modules
- pip: name=bottle virtualenv=/my_app/venv

# Install (Bottle) into the specified (virtualenv), inheriting globally installed modules
- pip: name=bottle virtualenv=/my_app/venv virtualenv_site_packages=yes

# Install (Bottle) into the specified (virtualenv), using Python 2.7
- pip: name=bottle virtualenv=/my_app/venv virtualenv_command=virtualenv-2.7

# Install specified python requirements.
- pip: requirements=/my_app/requirements.txt

# Install specified python requirements in indicated (virtualenv).
- pip: requirements=/my_app/requirements.txt virtualenv=/my_app/venv

# Install specified python requirements and custom Index URL.
- pip: requirements=/my_app/requirements.txt extra_args='-i https://example.com/pypi/simple'

# Install (Bottle) for Python 3.3 specifically,using the 'pip-3.3' executable.
- pip: name=bottle executable=pip-3.3
```

Note: Please note that virtualenv (http://www.virtualenv.org/) must be installed on the remote host if the virtualenv parameter is specified.

pkgin - Package manager for SmartOS

Author Shaun Zinck

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Manages SmartOS packages

Options

Examples

```
# install package foo"
- pkgin: name=foo state=present

# remove package foo
- pkgin: name=foo state=absent

# remove packages foo and bar
- pkgin: name=foo,bar state=absent
```

pkgng - Package manager for FreeBSD >= 9.0

Author bleader

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage binary packages for FreeBSD using 'pkgng' which is available in versions after 9.0.

Options

Examples

```
# Install package foo
- pkgng: name=foo state=present

# Remove packages foo and bar
- pkgng: name=foo,bar state=absent
```

Note: When using pkgsite, be careful that already in cache packages won't be downloaded again.

pkgutil - Manage CSW-Packages on Solaris

Author Alexander Winkler

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manages CSW packages (SVR4 format) on Solaris 10 and 11. These were the native packages on Solaris <= 10 and are available as a legacy feature in Solaris 11. Pkgutil is an advanced packaging system, which resolves dependency on installation. It is designed for CSW packages.

Options

Examples

```
# Install a package
pkgutil: name=CSWcommon state=present

# Install a package from a specific repository
pkgutil: name=CSWnrpe site='ftp://myinternal.repo/opencsw/kiel state=latest'
```

portinstall - Installing packages from FreeBSD's ports system

Author berenddeboer

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Manage packages for FreeBSD using 'portinstall'.

Options

Examples

```
# Install package foo
- portinstall: name=foo state=present

# Install package security/cyrus-sasl2-saslauthd
- portinstall: name=security/cyrus-sasl2-saslauthd state=present

# Remove packages foo and bar
- portinstall: name=foo,bar state=absent
```

redhat_subscription - Manage Red Hat Network registration and subscriptions using the `subscription-manager` command

Author James Laska

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage registration and subscription to the Red Hat Network entitlement platform.

Options

Note: Requires subscription-manager

Examples

```
# Register as user (joe_user) with password (somepass) and auto-subscribe to available content.
- redhat_subscription: action=register username=joe_user password=somepass autosubscribe=true

# Register with activationkey (1-222333444) and consume subscriptions matching
# the names (Red hat Enterprise Server) and (Red Hat Virtualization)
- redhat_subscription: action=register
                       activationkey=1-222333444
                       pool='^(Red Hat Enterprise Server|Red Hat Virtualization)$'
```

Note: In order to register a system, subscription-manager requires either a username and password, or an activation-key.

rhn_channel - Adds or removes Red Hat software channels

Author Vincent Van der Kussen

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Adds or removes Red Hat software channels

Options

Note: Requires none

Examples

```
- rhn_channel: name=rhel-x86_64-server-v2vwin-6 sysname=server01 url=https://rhn.redhat.com/rpc/api u
```

Note: this module fetches the system id from RHN.

rhn_register - Manage Red Hat Network registration using the `rhnreg_ks` command

Author James Laska

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage registration to the Red Hat Network.

Options

Note: Requires rhnreg_ks

Examples

```
# Unregister system from RHN.
- rhn_register: state=absent username=joe_user password=somepass

# Register as user (joe_user) with password (somepass) and auto-subscribe to available content.
- rhn_register: state=present username=joe_user password=somepass

# Register with activationkey (1-222333444) and enable extended update support.
- rhn_register: state=present activationkey=1-222333444 enable_eus=true

# Register as user (joe_user) with password (somepass) against a satellite
# server specified by (server_url).
- rhn_register:
    state=present
    username=joe_user
    password=somepass
    server_url=https://xmlrpc.my.satellite/XMLRPC

# Register as user (joe_user) with password (somepass) and enable
# channels (rhel-x86_64-server-6-foo-1) and (rhel-x86_64-server-6-bar-1).
- rhn_register: state=present username=joe_user
                password=somepass
                channels=rhel-x86_64-server-6-foo-1,rhel-x86_64-server-6-bar-1
```

Note: In order to register a system, rhnreg_ks requires either a username and password, or an activationkey.

rpm_key - Adds or removes a gpg key from the rpm db

Author Hector Acosta <hector.acosta@gazzang.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Adds or removes (rpm –import) a gpg key to your rpm database.

Options

Examples

```
# Example action to import a key from a url
- rpm_key: state=present key=http://apt.sw.be/RPM-GPG-KEY.dag.txt

# Example action to import a key from a file
- rpm_key: state=present key=/path/to/key.gpg
```

```
# Example action to ensure a key is not present in the db
- rpm_key: state=absent key=DEADB33F
```

svr4pkg - Manage Solaris SVR4 packages

Author Boyd Adamson

- Synopsis
- Options
- Examples

Synopsis

Manages SVR4 packages on Solaris 10 and 11. These were the native packages on Solaris <= 10 and are available as a legacy feature in Solaris 11. Note that this is a very basic packaging system. It will not enforce dependencies on install or remove.

Options

Examples

```
# Install a package from an already copied file
- svr4pkg: name=CSWcommon src=/tmp/cswpkgs.pkg state=present

# Install a package directly from an http site
- svr4pkg: name=CSWpkgutil src=http://get.opencsw.org/now state=present

# Install a package with a response file
- svr4pkg: name=CSWggrep src=/tmp/third-party.pkg response_file=/tmp/ggrep.response state=present

# Ensure that a package is not installed.
- svr4pkg: name=SUNWgnome-sound-recorder state=absent
```

swdepot - Manage packages with swdepot package manager (HP-UX)

Author Raul Melo

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Will install, upgrade and remove packages with swdepot package manager (HP-UX)

Options

Examples

```
- swdepot: name=unzip-6.0 state=installed depot=repository:/path
- swdepot: name=unzip state=latest depot=repository:/path
- swdepot: name=unzip state=absent
```

urpmi - Urpmi manager

Author Philippe Makowski

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.4.

Manages packages with *urpmi* (such as for Mageia or Mandriva)

Options

Examples

```
# install package foo
- urpmi: pkg=foo state=present
# remove package foo
- urpmi: pkg=foo state=absent
# description: remove packages foo and bar
- urpmi: pkg=foo,bar state=absent
# description: update the package database (urpmi.update -a -q) and install bar (bar will be the upda
- urpmi: name=bar, state=present, update_cache=yes
```

yum - Manages packages with the *yum* package manager

Author Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

Installs, upgrade, removes, and lists packages and groups with the *yum* package manager.

Options

Note: Requires yum

Note: Requires rpm

Examples

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

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

- name: install the latest version of Apche from the testing repo
  yum: name=httpd enablerepo=testing state=installed

- name: upgrade all packages
  yum: name=* state=latest

- name: install the nginx rpm from a remote repo
  yum: name=http://nginx.org/packages/centos/6/noarch/RPMS/nginx-release-centos-6-0.el6.ngx.noarch.rp

- name: install nginx rpm from a local file
  yum: name=/usr/local/src/nginx-release-centos-6-0.el6.ngx.noarch.rpm state=present

- name: install the 'Development tools' package group
  yum: name="@Development tools" state=present
```

zypper - Manage packages on SuSE and openSuSE

Author Patrick Callahan

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

Manage packages on SuSE and openSuSE using the zypper and rpm tools.

Options

Note: Requires zypper

Note: Requires rpm

Examples

```
# Install "nmap"
- zypper: name=nmap state=present

# Remove the "nmap" package
- zypper: name=nmap state=absent
```

zypper_repository - Add and remove Zypper repositories

Author Matthias Vogelgesang

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or remove Zypper repositories on SUSE and openSUSE

Options

Note: Requires zypper

Examples

```
# Add NVIDIA repository for graphics drivers
- zypper_repository: name=nvidia-repo repo='ftp://download.nvidia.com/opensuse/12.2' state=present

# Remove NVIDIA repository
- zypper_repository: name=nvidia-repo repo='ftp://download.nvidia.com/opensuse/12.2' state=absent
```

1.6.14 Source Control Modules

bzr - Deploy software (or files) from bzr branches

Author André Paramés

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manage *bzr* branches to deploy files or software.

Options

Examples

```
# Example bzr checkout from Ansible Playbooks
- bzr: name=bzr+ssh://foosball.example.org/path/to/branch dest=/srv/checkout version=22
```

git - Deploy software (or files) from git checkouts

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

Manage *git* checkouts of repositories to deploy files or software.

Options

Examples

```
# Example git checkout from Ansible Playbooks
- git: repo=git://foosball.example.org/path/to/repo.git
       dest=/srv/checkout
       version=release-0.22

# Example read-write git checkout from github
- git: repo=ssh://git@github.com/mylogin/hello.git dest=/home/mylogin/hello

# Example just ensuring the repo checkout exists
- git: repo=git://foosball.example.org/path/to/repo.git dest=/srv/checkout update=no
```

Note: If the task seems to be hanging, first verify remote host is in `known_hosts`. SSH will prompt user to authorize the first contact with a remote host. To avoid this prompt, one solution is to add the remote host public key in `/etc/ssh/ssh_known_hosts` before calling the git module, with the following command: ssh-keyscan remote_host.com >> /etc/ssh/ssh_known_hosts.

---

github_hooks - Manages github service hooks.

Author Phillip Gentry, CX Inc

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Adds service hooks and removes service hooks that have an error status.

Options

Examples

```
# Example creating a new service hook. It ignores duplicates.
- github_hooks: action=create hookurl=http://11.111.111.111:2222 user={{ gituser }} oauthkey={{ oauth

# Cleaning all hooks for this repo that had an error on the last update. Since this works for all hoo
- local_action: github_hooks action=cleanall user={{ gituser }} oauthkey={{ oauthkey }} repo={{ repo
```

hg - Manages Mercurial (hg) repositories.

Author Yeukhon Wong

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

Manages Mercurial (hg) repositories. Supports SSH, HTTP/S and local address.

Options

Examples

```
# Ensure the current working copy is inside the stable branch and deletes untracked files if any.
- hg: repo=https://bitbucket.org/user/repo1 dest=/home/user/repo1 revision=stable purge=yes
```

Note: If the task seems to be hanging, first verify remote host is in `known_hosts`. SSH will prompt user to authorize the first contact with a remote host. To avoid this prompt, one solution is to add the remote host public key in `/etc/ssh/ssh_known_hosts` before calling the hg module, with the following command: ssh-keyscan remote_host.com >> /etc/ssh/ssh_known_hosts.

subversion - Deploys a subversion repository.

Author Dane Summers, njharman@gmail.com

- Synopsis
- Options
- Examples

Synopsis

Deploy given repository URL / revision to dest. If dest exists, update to the specified revision, otherwise perform a checkout.

Options

Examples

```
# Checkout subversion repository to specified folder.
- subversion: repo=svn+ssh://an.example.org/path/to/repo dest=/src/checkout
```

Note: Requres *svn* to be installed on the client.

1.6.15 System Modules

at - Schedule the execution of a command or scripts via the at command.

Author Richard Isaacson

- Synopsis
- Options
- Examples

Synopsis

Use this module to schedule a command or script to run once in the future. All jobs are executed in the a queue.

Options

Note: Requires at

Examples

```
# Schedule a command to execute in 20 minutes as root.
- at: command="ls -d / > /dev/null" unit_count=20 unit_type="minutes"

# Schedule a script to execute in 1 hour as the neo user.
- at: script_file="/some/script.sh" user="neo" unit_count=1 unit_type="hours"

# Match a command to an existing job and delete the job.
- at: command="ls -d / > /dev/null" action="delete"

# Schedule a command to execute in 20 minutes making sure it is unique in the queue.
- at: command="ls -d / > /dev/null" action="unique" unit_count=20 unit_type="minutes"
```

authorized_key - Adds or removes an SSH authorized key

Author Brad Olson

- Synopsis
- Options
- Examples

Synopsis

Adds or removes authorized keys for particular user accounts

Options

Examples

```
# Example using key data from a local file on the management machine
- authorized_key: user=charlie key="{{ lookup('file', '/home/charlie/.ssh/id_rsa.pub') }}"

# Using alternate directory locations:
- authorized_key: user=charlie
                  key="{{ lookup('file', '/home/charlie/.ssh/id_rsa.pub') }}"
                  path='/etc/ssh/authorized_keys/charlie'
                  manage_dir=no

# Using with_file
- name: Set up authorized_keys for the deploy user
  authorized_key: user=deploy
                  key="{{ item }}"
  with_file:
    - public_keys/doe-jane
```

```
    - public_keys/doe-john

# Using key_options:
- authorized_key: user=charlie
                  key="{{ lookup('file', '/home/charlie/.ssh/id_rsa.pub') }}"
                  key_options='no-port-forwarding,host="10.0.1.1"'
```

cron - Manage cron.d and crontab entries.

Author Dane Summers

- Synopsis
- Options
- Examples

Synopsis

Use this module to manage crontab entries. This module allows you to create named crontab entries, update, or delete them. The module includes one line with the description of the crontab entry "`#Ansible: <name>`" corresponding to the "name" passed to the module, which is used by future ansible/module calls to find/check the state.

Options

Note: Requires cron

Examples

```
# Ensure a job that runs at 2 and 5 exists.
# Creates an entry like "* 5,2 * * ls -alh > /dev/null"
- cron: name="check dirs" hour="5,2" job="ls -alh > /dev/null"

# Ensure an old job is no longer present. Removes any job that is prefixed
# by "#Ansible: an old job" from the crontab
- cron: name="an old job" state=absent

# Creates an entry like "@reboot /some/job.sh"
- cron: name="a job for reboot" special_time=reboot job="/some/job.sh"

# Creates a cron file under /etc/cron.d
- cron: name="yum autoupdate" weekday="2" minute=0 hour=12
        user="root" job="YUMINTERACTIVE=0 /usr/sbin/yum-autoupdate"
        cron_file=ansible_yum-autoupdate

# Removes a cron file from under /etc/cron.d
- cron: cron_file=ansible_yum-autoupdate state=absent
```

facter - Runs the discovery program *facter* on the remote system

Author Michael DeHaan

- Synopsis
- Examples

Synopsis

Runs the *facter* discovery program (https://github.com/puppetlabs/facter) on the remote system, returning JSON data that can be useful for inventory purposes.

Note: Requires facter

Note: Requires ruby-json

Examples

```
# Example command-line invocation
ansible www.example.net -m facter
```

filesystem - Makes file system on block device

Author Alexander Bulimov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module creates file system.

Options

Examples

```
# Create a ext2 filesystem on /dev/sdb1.
- filesystem: fstype=ext2 dev=/dev/sdb1

# Create a ext4 filesystem on /dev/sdb1 and check disk blocks.
- filesystem: fstype=ext4 dev=/dev/sdb1 opts="-cc"
```

Note: uses mkfs command

firewalld - Manage arbitrary ports/services with firewalld

Author Adam Miller <maxamillion@fedoraproject.org>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

This module allows for addition or deletion of services and ports either tcp or udp in either running or permanent firewalld rules

Options

Note: Requires firewalld >= 0.2.11

Examples

```
- firewalld: service=https permanent=true state=enabled
- firewalld: port=8081/tcp permanent=true state=disabled
- firewalld: zone=dmz service=http permanent=true state=enabled
- firewalld: rich_rule='rule service name="ftp" audit limit value="1/m" accept' permanent=true state=
```

Note: Not tested on any debian based system

group - Add or remove groups

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Manage presence of groups on a host.

Options

Note: Requires groupadd

Note: Requires groupdel

Note: Requires groupmod

Examples

```
# Example group command from Ansible Playbooks
- group: name=somegroup state=present
```

hostname - Manage hostname

Author Hiroaki Nakamura

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Set system's hostname Currently implemented on only Debian, Ubuntu, RedHat and CentOS.

Options

Note: Requires hostname

Examples

```
- hostname: name=web01
```

kernel_blacklist - Blacklist kernel modules

Author Matthias Vogelgesang

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or remove kernel modules from blacklist.

Options

Examples

```
# Blacklist the nouveau driver module
- kernel_blacklist: name=nouveau state=present
```

lvg - Configure LVM volume groups

Author Alexander Bulimov

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

This module creates, removes or resizes volume groups.

Options

Examples

```
# Create a volume group on top of /dev/sda1 with physical extent size = 32MB.
- lvg:  vg=vg.services pvs=/dev/sda1 pesize=32

# Create or resize a volume group on top of /dev/sdb1 and /dev/sdc5.
# If, for example, we already have VG vg.services on top of /dev/sdb1,
# this VG will be extended by /dev/sdc5.  Or if vg.services was created on
# top of /dev/sda5, we first extend it with /dev/sdb1 and /dev/sdc5,
# and then reduce by /dev/sda5.
- lvg: vg=vg.services pvs=/dev/sdb1,/dev/sdc5

# Remove a volume group with name vg.services.
- lvg: vg=vg.services state=absent
```

Note: module does not modify PE size for already present volume group

lvol - Configure LVM logical volumes

Author Jeroen Hoekx

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

This module creates, removes or resizes logical volumes.

Options

Examples

```
# Create a logical volume of 512m.
- lvol: vg=firefly lv=test size=512

# Create a logical volume of 512g.
- lvol: vg=firefly lv=test size=512g

# Create a logical volume the size of all remaining space in the volume group
- lvol: vg=firefly lv=test size=100%FREE

# Extend the logical volume to 1024m.
- lvol: vg=firefly lv=test size=1024

# Reduce the logical volume to 512m
- lvol: vg=firefly lv=test size=512

# Remove the logical volume.
- lvol: vg=firefly lv=test state=absent
```

Note: Filesystems on top of the volume are not resized.

modprobe - Add or remove kernel modules

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Add or remove kernel modules.

Options

Examples

```
# Add the 802.1q module
- modprobe: name=8021q state=present
```

mount - Control active and configured mount points

Author Seth Vidal

- Synopsis
- Options
- Examples

Synopsis

This module controls active and configured mount points in `/etc/fstab`.

Options

Examples

```
# Mount DVD read-only
- mount: name=/mnt/dvd src=/dev/sr0 fstype=iso9660 opts=ro state=present

# Mount up device by label
- mount: name=/srv/disk src='LABEL=SOME_LABEL' state=present

# Mount up device by UUID
- mount: name=/home src='UUID=b3e48f45-f933-4c8e-a700-22a159ec9077' opts=noatime state=present
```

ohai - Returns inventory data from *Ohai*

Author Michael DeHaan

- Synopsis
- Examples

Synopsis

Similar to the `facter` module, this runs the *Ohai* discovery program (http://wiki.opscode.com/display/chef/Ohai) on the remote host and returns JSON inventory data. *Ohai* data is a bit more verbose and nested than *facter*.

Note: Requires ohai

Examples

```
# Retrieve (ohai) data from all Web servers and store in one-file per host
ansible webservers -m ohai --tree=/tmp/ohaidata
```

open_iscsi - Manage iscsi targets with open-iscsi

Author Serge van Ginderachter

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Discover targets on given portal, (dis)connect targets, mark targets to manually or auto start, return device nodes of connected targets.

Options

Note: Requires open_iscsi library and tools (iscsiadm)

Examples

ping - Try to connect to host and return `pong` on success.

Author Michael DeHaan

- Synopsis
- Examples

Synopsis

A trivial test module, this module always returns `pong` on successful contact. It does not make sense in playbooks, but it is useful from `/usr/bin/ansible`

Examples

```
# Test 'webservers' status
ansible webservers -m ping
```

seboolean - Toggles SELinux booleans.

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Toggles SELinux booleans.

Options

Examples

```
# Set (httpd_can_network_connect) flag on and keep it persistent across reboots
- seboolean: name=httpd_can_network_connect state=yes persistent=yes
```

Note: Not tested on any debian based system

selinux - Change policy and state of SELinux

Author Derek Carter <goozbach@friocorte.com>

- Synopsis
- Options
- Examples

Synopsis

Configures the SELinux mode and policy. A reboot may be required after usage. Ansible will not issue this reboot but will let you know when it is required.

Options

Note: Requires libselinux-python

Examples

```
- selinux: policy=targeted state=enforcing
- selinux: policy=targeted state=permissive
- selinux: state=disabled
```

Note: Not tested on any debian based system

service - Manage services.

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

Controls services on remote hosts.

Options

Examples

```
# 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

# Example action to enable service httpd, and not touch the running state
- service: name=httpd enabled=yes

# Example action to start service foo, based on running process /usr/bin/foo
- service: name=foo pattern=/usr/bin/foo state=started

# Example action to restart network service for interface eth0
- service: name=network state=restarted args=eth0
```

setup - Gathers facts about remote hosts

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

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.

Options

Examples

```
# Display facts from all hosts and store them indexed by I(hostname) at C(/tmp/facts).
ansible all -m setup --tree /tmp/facts

# Display only facts regarding memory found by ansible on all hosts and output them.
ansible all -m setup -a 'filter=ansible_*_mb'

# Display only facts returned by facter.
ansible all -m setup -a 'filter=facter_*'

# Display only facts about certain interfaces.
ansible all -m setup -a 'filter=ansible_eth[0-2]'
```

Note: 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. (See also `facter` and `ohai`.)

Note: The filter option filters only the first level subkey below ansible_facts.

---

sysctl - Manage entries in sysctl.conf.

Author David "DaviXX" CHANIAL <david.chanial@gmail.com>

- Synopsis
- Options
- Examples

Synopsis

New in version 1.0.

This module manipulates sysctl entries and optionally performs a `/sbin/sysctl -p` after changing them.

Options

Examples

```
# Set vm.swappiness to 5 in /etc/sysctl.conf
- sysctl: name=vm.swappiness value=5 state=present

# Remove kernel.panic entry from /etc/sysctl.conf
- sysctl: name=kernel.panic state=absent sysctl_file=/etc/sysctl.conf

# Set kernel.panic to 3 in /tmp/test_sysctl.conf
- sysctl: name=kernel.panic value=3 sysctl_file=/tmp/test_sysctl.conf reload=no

# Set ip fowarding on in /proc and do not reload the sysctl file
- sysctl: name="net.ipv4.ip_forward" value=1 sysctl_set=yes

# Set ip forwarding on in /proc and in the sysctl file and reload if necessary
- sysctl: name="net.ipv4.ip_forward" value=1 sysctl_set=yes state=present reload=yes
```

user - Manage user accounts

Author Stephen Fromm

- Synopsis
- Options
- Examples

Synopsis

Manage user accounts and user attributes.

Options

Note: Requires useradd

Note: Requires userdel

Note: Requires usermod

Examples

```
# Add the user 'johnd' with a specific uid and a primary group of 'admin'
- user: name=johnd comment="John Doe" uid=1040

# Remove the user 'johnd'
- user: name=johnd state=absent remove=yes
```

```
# Create a 2048-bit SSH key for user jsmith
- user: name=jsmith generate_ssh_key=yes ssh_key_bits=2048
```

zfs - Manage zfs

Author Johan Wiren

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages ZFS file systems on Solaris and FreeBSD. Can manage file systems, volumes and snapshots. See zfs(1M) for more information about the properties.

Options

Examples

```
# Create a new file system called myfs in pool rpool
- zfs: name=rpool/myfs state=present

# Create a new volume called myvol in pool rpool.
- zfs: name=rpool/myvol state=present volsize=10M

# Create a snapshot of rpool/myfs file system.
- zfs: name=rpool/myfs@mysnapshot state=present

# Create a new file system called myfs2 with snapdir enabled
- zfs: name=rpool/myfs2 state=present snapdir=enabled
```

1.6.16 Utilities Modules

accelerate - Enable accelerated mode on remote node

Author James Cammarata

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

This modules launches an ephemeral *accelerate* daemon on the remote node which Ansible can use to communicate with nodes at high speed. The daemon listens on a configurable port for a configurable amount of time. Fireball mode is AES encrypted

Options

Note: Requires python-keyczar

Examples

```
# To use accelerate mode, simply add "accelerate: true" to your play. The initial
# key exchange and starting up of the daemon will occur over SSH, but all commands and
# subsequent actions will be conducted over the raw socket connection using AES encryption

- hosts: devservers
  accelerate: true
  tasks:
      - command: /usr/bin/anything
```

Note: See the advanced playbooks chapter for more about using accelerated mode.

assert - Fail with custom message

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module asserts that a given expression is true and can be a simpler alternative to the 'fail' module in some cases.

Options

Examples

```
- assert: ansible_os_family != "RedHat"
- assert: "'foo' in some_command_result.stdout"
```

debug - Print statements during execution

Author Dag Wieers, Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This module prints statements during execution and can be useful for debugging variables or expressions without necessarily halting the playbook. Useful for debugging together with the 'when:' directive.

Options

Examples

```
# Example that prints the loopback address and gateway for each host
- debug: msg="System {{ inventory_hostname }} has uuid {{ ansible_product_uuid }}"

- debug: msg="System {{ inventory_hostname }} has gateway {{ ansible_default_ipv4.gateway }}"
  when: ansible_default_ipv4.gateway is defined

- shell: /usr/bin/uptime
  register: result
- debug: var=result
```

fail - Fail with custom message

Author Dag Wieers

- Synopsis
- Options
- Examples

Synopsis

This module fails the progress with a custom message. It can be useful for bailing out when a certain condition is met using `when`.

Options

Examples

```
# Example playbook using fail and when together
- fail: msg="The system may not be provisioned according to the CMDB status."
  when: cmdb_status != "to-be-staged"
```

fireball - Enable fireball mode on remote node

Author Michael DeHaan

- Synopsis
- Options
- Examples

Synopsis

This modules launches an ephemeral *fireball* ZeroMQ message bus daemon on the remote node which Ansible can use to communicate with nodes at high speed. The daemon listens on a configurable port for a configurable amount of time. Starting a new fireball as a given user terminates any existing user fireballs. Fireball mode is AES encrypted

Options

Note: Requires zmq

Note: Requires keyczar

Examples

```
# This example playbook has two plays: the first launches 'fireball' mode on all hosts via SSH, and
# the second actually starts using it for subsequent management over the fireball connection

- hosts: devservers
  gather_facts: false
  connection: ssh
  sudo: yes
  tasks:
      - action: fireball

- hosts: devservers
  connection: fireball
  tasks:
      - command: /usr/bin/anything
```

Note: See the advanced playbooks chapter for more about using fireball mode.

include_vars - Load variables from files, dynamically within a task.

Author Benno Joy

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Loads variables from a YAML file dynamically during task runtime. It can work with conditionals, or use host specific variables to determine the path name to load from.

Options

Examples

```
# Conditionally decide to load in variables when x is 0, otherwise do not.
- include_vars: contingency_plan.yml
  when: x == 0

# Load a variable file based on the OS type, or a default if not found.
- include_vars: "{{ item }}"
  with_first_found:
   - "{{ ansible_os_distribution }}.yml"
   - "default.yml"
```

pause - Pause playbook execution

Author Tim Bielawa

- Synopsis
- Options
- Examples

Synopsis

Pauses playbook execution for a set amount of time, or until a prompt is acknowledged. All parameters are optional. The default behavior is to pause with a prompt. You can use `ctrl+c` if you wish to advance a pause earlier than it is set to expire or if you need to abort a playbook run entirely. To continue early: press `ctrl+c` and then `c`. To abort a playbook: press `ctrl+c` and then `a`. The pause module integrates into async/parallelized playbooks without any special considerations (see also: Rolling Updates). When using pauses with the `serial` playbook parameter (as in rolling updates) you are only prompted once for the current group of hosts.

Options

Examples

```
# Pause for 5 minutes to build app cache.
- pause: minutes=5

# Pause until you can verify updates to an application were successful.
- pause:

# A helpful reminder of what to look out for post-update.
- pause: prompt="Make sure org.foo.FooOverload exception is not present"
```

set_fact - Set host facts from a task

Author Dag Wieers

- Synopsis
- Options
- Examples

Synopsis

New in version 1.2.

This module allows setting new variables. Variables are set on a host-by-host basis just like facts discovered by the setup module. These variables will survive between plays.

Options

Examples

```
# Example setting host facts using key=value pairs
- set_fact: one_fact="something" other_fact="{{ local_var * 2 }}"

# Example setting host facts using complex arguments
- set_fact:
    one_fact: something
    other_fact: "{{ local_var * 2 }}"
```

wait_for - Waits for a condition before continuing.

Author Jeroen Hoekx, John Jarvis

- Synopsis
- Options
- Examples

Synopsis

Waiting for a port to become available is useful for when services are not immediately available after their init scripts return - which is true of certain Java application servers. It is also useful when starting guests with the `virt` module and needing to pause until they are ready. This module can also be used to wait for a file to be available on the filesystem or with a regex match a string to be present in a file.

Options

Examples

```
# wait 300 seconds for port 8000 to become open on the host, don't start checking for 10 seconds
- wait_for: port=8000 delay=10

# wait until the file /tmp/foo is present before continuing
- wait_for: path=/tmp/foo

# wait until the string "completed" is in the file /tmp/foo before continuing
- wait_for: path=/tmp/foo search_regex=completed
```

1.6.17 Web Infrastructure Modules

django_manage - Manages a Django application.

Author Scott Anderson

- Synopsis
- Options
- Examples

Synopsis

New in version 1.1.

Manages a Django application using the *manage.py* application frontend to *django-admin*. With the *virtualenv* parameter, all management commands will be executed by the given *virtualenv* installation.

Options

Note: Requires virtualenv

Note: Requires django

Examples

```
# Run cleanup on the application installed in 'django_dir'.
- django_manage: command=cleanup app_path={{ django_dir }}

# Load the initial_data fixture into the application
- django_manage: command=loaddata app_path={{ django_dir }} fixtures={{ initial_data }}

#Run syncdb on the application
- django_manage: >
      command=syncdb
      app_path={{ django_dir }}
      settings={{ settings_app_name }}
      pythonpath={{ settings_dir }}
      virtualenv={{ virtualenv_dir }}

#Run the SmokeTest test case from the main app. Useful for testing deploys.
- django_manage: command=test app_path=django_dir apps=main.SmokeTest
```

Note: *virtualenv* (http://www.virtualenv.org) must be installed on the remote host if the virtualenv parameter is specified.

Note: This module will create a virtualenv if the virtualenv parameter is specified and a virtualenv does not already exist at the given location.

Note: This module assumes English error messages for the 'createcachetable' command to detect table existence, unfortunately.

Note: To be able to use the migrate command, you must have south installed and added as an app in your settings

Note: To be able to use the collectstatic command, you must have enabled staticfiles in your settings

ejabberd_user - Manages users for ejabberd servers

Author Peter Sprygada

- Synopsis
- Options
- Examples

Synopsis

New in version 1.5.

This module provides user management for ejabberd servers

Options

Note: Requires ejabberd

Examples

```
Example playbook entries using the ejabberd_user module to manage users state.

    tasks:

    - name: create a user if it does not exists
      action: ejabberd_user username=test host=server password=password

    - name: delete a user if it exists
      action: ejabberd_user username=test host=server state=absent
```

Note: Password parameter is required for state == present only

Note: Passwords must be stored in clear text for this release

htpasswd - manage user files for basic authentication

Author Lorin Hochstein

- Synopsis
- Options
- Examples

Synopsis

New in version 1.3.

Add and remove username/password entries in a password file using htpasswd. This is used by web servers such as Apache and Nginx for basic authentication.

Options

Examples

```
# Add a user to a password file and ensure permissions are set
- htpasswd: path=/etc/nginx/passwdfile name=janedoe password=9s36?;fyNp owner=root group=www-data mo
# Remove a user from a password file
- htpasswd: path=/etc/apache2/passwdfile name=foobar state=absent
```

Note: This module depends on the *passlib* Python library, which needs to be installed on all target systems.

Note: On Debian, Ubuntu, or Fedora: install *python-passlib*.

Note: On RHEL or CentOS: Enable EPEL, then install *python-passlib*.

jboss - deploy applications to JBoss

Author Jeroen Hoekx

- Synopsis
- Options
- Examples

Synopsis

New in version 1.4.

Deploy applications to JBoss standalone using the filesystem

Options

Examples

```
# Deploy a hello world application
- jboss: src=/tmp/hello-1.0-SNAPSHOT.war deployment=hello.war state=present
# Update the hello world application
- jboss: src=/tmp/hello-1.1-SNAPSHOT.war deployment=hello.war state=present
# Undeploy the hello world application
- jboss: deployment=hello.war state=absent
```

Note: The JBoss standalone deployment-scanner has to be enabled in standalone.xml

Note: Ensure no identically named application is deployed through the JBoss CLI

supervisorctl - Manage the state of a program or group of programs running via Supervisord

Author Matt Wright

- Synopsis
- Options
- Examples

Synopsis

Manage the state of a program or group of programs running via *Supervisord*

Options

Examples

```
# Manage the state of program to be in 'started' state.
- supervisorctl: name=my_app state=started

# Restart my_app, reading supervisorctl configuration from a specified file.
- supervisorctl: name=my_app state=restarted config=/var/opt/my_project/supervisord.conf

# Restart my_app, connecting to supervisord with credentials and server URL.
- supervisorctl: name=my_app state=restarted username=test password=testpass server_url=http://localh
```

1.7 Detailed Guides

This section is new and evolving. The idea here is explore particular use cases in greater depth and provide a more "top down" explanation of some basic features.

1.7.1 Amazon Web Services Guide

Introduction

Note: This section of the documentation is under construction. We are in the process of adding more examples about all of the EC2 modules and how they work together. There's also an ec2 example in the language_features directory of the ansible-examples github repository that you may wish to consult. Once complete, there will also be new examples of ec2 in ansible-examples.

Ansible contains a number of core modules for interacting with Amazon Web Services (AWS). These also work with Eucalyptus, which is an AWS compatible private cloud solution. There are other supported cloud types, but this documentation chapter is about AWS API clouds. The purpose of this section is to explain how to put Ansible modules together (and use inventory scripts) to use Ansible in AWS context.

Requirements for the AWS modules are minimal. All of the modules require and are tested against boto 2.5 or higher. You'll need this Python module installed on the execution host. If you are using Red Hat Enterprise Linux or CentOS, install boto from EPEL:

```
$ yum install python-boto
```

You can also install it via pip if you want.

The following steps will often execute outside the host loop, so it makes sense to add localhost to inventory. Ansible may not require this step in the future:

```
[local]
localhost
```

And in your playbook steps we'll typically be using the following pattern for provisioning steps:

```
- hosts: localhost
  connection: local
  gather_facts: False
```

Provisioning

The ec2 module provides the ability to provision instances within EC2. Typically the provisioning task will be performed against your Ansible master server in a play that operates on localhost using the `local` connection type. If you are doing an EC2 operation mid-stream inside a regular play operating on remote hosts, you may want to use the `local_action` keyword for that particular task. Read *Delegation, Rolling Updates, and Local Actions* for more about local actions.

Note: Authentication with the AWS-related modules is handled by either specifying your access and secret key as ENV variables or passing them as module arguments.

Note: To talk to specific endpoints, the environmental variable EC2_URL can be set. This is useful if using a private cloud like Eucalyptus, exporting the variable as EC2_URL=https://myhost:8773/services/Eucalyptus. This can be set using the 'environment' keyword in Ansible if you like.

Here is an example of provisioning a number of instances in ad-hoc mode:

```
# ansible localhost -m ec2 -a "image=ami-6e649707 instance_type=m1.large keypair=mykey group=webserve
```

In a play, this might look like (assuming the parameters are held as vars):

```
tasks:
- name: Provision a set of instances
  ec2: >
      keypair={{mykeypair}}
      group={{security_group}}
      instance_type={{instance_type}}
      image={{image}}
      wait=true
      count={{number}}
  register: ec2
```

By registering the return its then possible to dynamically create a host group consisting of these new instances. This facilitates performing configuration actions on the hosts immediately in a subsequent task:

```
- name: Add all instance public IPs to host group
  add_host: hostname={{ item.public_ip }} groupname=ec2hosts
  with_items: ec2.instances
```

With the host group now created, a second play in your provision playbook might now have some configuration steps:

```
- name: Configuration play
  hosts: ec2hosts
  user: ec2-user
  gather_facts: true

  tasks:
  - name: Check NTP service
    service: name=ntpd state=started
```

Rather than include configuration inline, you may also choose to just do it as a task include or a role.

The method above ties the configuration of a host with the provisioning step. This isn't always ideal and leads us onto the next section.

Advanced Usage

Host Inventory

Once your nodes are spun up, you'll probably want to talk to them again. The best way to handle this is to use the ec2 inventory plugin.

Even for larger environments, you might have nodes spun up from Cloud Formations or other tooling. You don't have to use Ansible to spin up guests. Once these are created and you wish to configure them, the EC2 API can be used to return system grouping with the help of the EC2 inventory script. This script can be used to group resources by their security group or tags. Tagging is highly recommended in EC2 and can provide an easy way to sort between host groups and roles. The inventory script is documented doc:*api* section.

You may wish to schedule a regular refresh of the inventory cache to accommodate for frequent changes in resources:

```
# ./ec2.py --refresh-cache
```

Put this into a crontab as appropriate to make calls from your Ansible master server to the EC2 API endpoints and gather host information. The aim is to keep the view of hosts as up-to-date as possible, so schedule accordingly. Playbook calls could then also be scheduled to act on the refreshed hosts inventory after each refresh. This approach means that machine images can remain "raw", containing no payload and OS-only. Configuration of the workload is handled entirely by Ansible.

Tags

There's a feature in the ec2 inventory script where hosts tagged with certain keys and values automatically appear in certain groups.

For instance, if a host is given the "class" tag with the value of "webserver", it will be automatically discoverable via a dynamic group like so:

```
- hosts: tag_class_webserver
  tasks:
    - ping
```

Using this philosophy can be a great way to manage groups dynamically, without having to maintain seperate inventory.

Pull Configuration

For some the delay between refreshing host information and acting on that host information (i.e. running Ansible tasks against the hosts) may be too long. This may be the case in such scenarios where EC2 AutoScaling is being used to scale the number of instances as a result of a particular event. Such an event may require that hosts come online and are configured as soon as possible (even a 1 minute delay may be undesirable). Its possible to pre-bake machine images which contain the necessary ansible-pull script and components to pull and run a playbook via git. The machine images could be configured to run ansible-pull upon boot as part of the bootstrapping procedure.

Read *Ansible-Pull* for more information on pull-mode playbooks.

(Various developments around Ansible are also going to make this easier in the near future. Stay tuned!)

Autoscaling with Ansible Tower

Ansible Tower also contains a very nice feature for auto-scaling use cases. In this mode, a simple curl script can call a defined URL and the server will "dial out" to the requester and configure an instance that is spinning up. This can be

a great way to reconfigure ephemeral nodes. See the Tower documentation for more details. Click on the Tower link in the sidebar for details.

A benefit of using the callback in Tower over pull mode is that job results are still centrally recorded and less information has to be shared with remote hosts.

Use Cases

This section covers some usage examples built around a specific use case.

Example 1

> Example 1: I'm using CloudFormation to deploy a specific infrastructure stack. I'd like to manage configuration of the instances with Ansible.

Provision instances with your tool of choice and consider using the inventory plugin to group hosts based on particular tags or security group. Consider tagging instances you wish to managed with Ansible with a suitably unique key=value tag.

Note: Ansible also has a cloudformation module you may wish to explore.

Example 2

> Example 2: I'm using AutoScaling to dynamically scale up and scale down the number of instances. This means the number of hosts is constantly fluctuating but I'm letting EC2 automatically handle the provisioning of these instances. I don't want to fully bake a machine image, I'd like to use Ansible to configure the hosts.

There are several approaches to this use case. The first is to use the inventory plugin to regularly refresh host information and then target hosts based on the latest inventory data. The second is to use ansible-pull triggered by a user-data script (specified in the launch configuration) which would then mean that each instance would fetch Ansible and the latest playbook from a git repository and run locally to configure itself. You could also use the Tower callback feature.

Example 3

> Example 3: I don't want to use Ansible to manage my instances but I'd like to consider using Ansible to build my fully-baked machine images.

There's nothing to stop you doing this. If you like working with Ansible's playbook format then writing a playbook to create an image; create an image file with dd, give it a filesystem and then install packages and finally chroot into it for further configuration. Ansible has the 'chroot' plugin for this purpose, just add the following to your inventory file:

```
/chroot/path ansible_connection=chroot
```

And in your playbook:

```
hosts: /chroot/path
```

Example 4

How would I create a new ec2 instance, provision it and then destroy it all in the same play?

```
# Use the ec2 module to create a new host and then add
# it to a special "ec2hosts" group.

- hosts: localhost
  connection: local
  gather_facts: False
  vars:
    ec2_access_key: "--REMOVED--"
    ec2_secret_key: "--REMOVED--"
    keypair: "mykeyname"
    instance_type: "t1.micro"
    image: "ami-d03ea1e0"
    group: "mysecuritygroup"
    region: "us-west-2"
    zone: "us-west-2c"
  tasks:
    - name: make one instance
      ec2: image={{ image }}
           instance_type={{ instance_type }}
           aws_access_key={{ ec2_access_key }}
           aws_secret_key={{ ec2_secret_key }}
           keypair={{ keypair }}
           instance_tags='{"foo":"bar"}'
           region={{ region }}
           group={{ group }}
           wait=true
      register: ec2_info

    - debug: var=ec2_info
    - debug: var=item
      with_items: ec2_info.instance_ids

    - add_host: hostname={{ item.public_ip }} groupname=ec2hosts
      with_items: ec2_info.instances

    - name: wait for instances to listen on port:22
      wait_for:
        state=started
        host={{ item.public_dns_name }}
        port=22
      with_items: ec2_info.instances


# Connect to the node and gather facts,
# including the instance-id. These facts
# are added to inventory hostvars for the
# duration of the playbook's execution
# Typical "provisioning" tasks would go in
# this playbook.

- hosts: ec2hosts
  gather_facts: True
  user: ec2-user
  sudo: True
```

```
  tasks:

    # fetch instance data from the metadata servers in ec2
    - ec2_facts:

    # show all known facts for this host
    - debug: var=hostvars[inventory_hostname]

    # just show the instance-id
    - debug: msg="{{ hostvars[inventory_hostname]['ansible_ec2_instance-id'] }}"


# Using the instanceid, call the ec2 module
# locally to remove the instance by declaring
# it's state is "absent"

- hosts: ec2hosts
  gather_facts: True
  connection: local
  vars:
    ec2_access_key: "--REMOVED--"
    ec2_secret_key: "--REMOVED--"
    region: "us-west-2"
  tasks:
    - name: destroy all instances
      ec2: state='absent'
           aws_access_key={{ ec2_access_key }}
           aws_secret_key={{ ec2_secret_key }}
           region={{ region }}
           instance_ids={{ item }}
           wait=true
      with_items: hostvars[inventory_hostname]['ansible_ec2_instance-id']
```

Note: more examples of this are pending. You may also be interested in the ec2_ami module for taking AMIs of running instances.

Pending Information

In the future look here for more topics.

See also:

About Modules All the documentation for Ansible modules

Playbooks An introduction to playbooks

Delegation, Rolling Updates, and Local Actions Delegation, useful for working with loud balancers, clouds, and locally executed steps.

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.7.2 Rackspace Cloud Guide

Introduction

Note: This section of the documentation is under construction. We are in the process of adding more examples about the Rackspace modules and how they work together. Once complete, there will also be examples for Rackspace Cloud in ansible-examples.

Ansible contains a number of core modules for interacting with Rackspace Cloud.

The purpose of this section is to explain how to put Ansible modules together (and use inventory scripts) to use Ansible in Rackspace Cloud context.

Prerequisites for using the rax modules are minimal. In addition to ansible itself, all of the modules require and are tested against pyrax 1.5 or higher. You'll need this Python module installed on the execution host.

pyrax is not currently available in many operating system package repositories, so you will likely need to install it via pip:

```
$ pip install pyrax
```

The following steps will often execute from the control machine against the Rackspace Cloud API, so it makes sense to add localhost to the inventory file. (Ansible may not require this manual step in the future):

```
[localhost]
localhost ansible_connection=local
```

In playbook steps we'll typically be using the following pattern:

```
- hosts: localhost
  connection: local
  gather_facts: False
  tasks:
```

Credentials File

The *rax.py* inventory script and all *rax* modules support a standard *pyrax* credentials file that looks like:

```
[rackspace_cloud]
username = myraxusername
api_key = d41d8cd98f00b204e9800998ecf8427e
```

Setting the environment parameter RAX_CREDS_FILE to the path of this file will help Ansible find how to load this information.

More information about this credentials file can be found at https://github.com/rackspace/pyrax/blob/master/docs/getting_started.md#autl

Running from a Python Virtual Environment (Optional)

Special considerations need to be taken if pyrax is not installed globally but instead using a python virtualenv (it's fine if you install it globally).

Ansible assumes, unless otherwise instructed, that the python binary will live at /usr/bin/python. This is done so via the interpret line in the modules, however when instructed using ansible_python_interpreter, ansible will use this specified path instead for finding python.

If using virtualenv, you may wish to modify your localhost inventory definition to find this location as follows:

```
[localhost]
localhost ansible_connection=local ansible_python_interpreter=/path/to/ansible_venv/bin/python
```

Provisioning

Now for the fun parts.

The 'rax' module provides the ability to provision instances within Rackspace Cloud. Typically the provisioning task will be performed from your Ansible control server against the Rackspace cloud API.

Note: Authentication with the Rackspace-related modules is handled by either specifying your username and API key as environment variables or passing them as module arguments.

Here is a basic example of provisioning a instance in ad-hoc mode:

```
$ ansible localhost -m rax -a "name=awx flavor=4 image=ubuntu-1204-lts-precise-pangolin wait=yes" -c
```

Here's what it would look like in a playbook, assuming the parameters were defined in variables:

```
tasks:
  - name: Provision a set of instances
    local_action:
        module: rax
        name: "{{ rax_name }}"
        flavor: "{{ rax_flavor }}"
        image: "{{ rax_image }}"
        count: "{{ rax_count }}"
        group: "{{ group }}"
        wait: yes
    register: rax
```

By registering the return value of the step, it is then possible to dynamically add the resulting hosts to inventory (temporarily, in memory). This facilitates performing configuration actions on the hosts immediately in a subsequent task:

```
- name: Add the instances we created (by public IP) to the group 'raxhosts'
  local_action:
      module: add_host
      hostname: "{{ item.name }}"
      ansible_ssh_host: "{{ item.rax_accessipv4 }}"
      ansible_ssh_pass: "{{ item.rax_adminpass }}"
      groupname: raxhosts
  with_items: rax.success
  when: rax.action == 'create'
```

With the host group now created, a second play in your provision playbook could now configure them, for example:

```
- name: Configuration play
  hosts: raxhosts
  user: root
  roles:
    - ntp
    - webserver
```

The method above ties the configuration of a host with the provisioning step. This isn't always what you want, and leads us to the next section.

Host Inventory

Once your nodes are spun up, you'll probably want to talk to them again.

The best way to handle his is to use the rax inventory plugin, which dynamically queries Rackspace Cloud and tells Ansible what nodes you have to manage.

You might want to use this even if you are spinning up Ansible via other tools, including the Rackspace Cloud user interface.

The inventory plugin can be used to group resources by their meta data. Utilizing meta data is highly recommended in rax and can provide an easy way to sort between host groups and roles.

If you don't want to use the `rax.py` dynamic inventory script, you could also still choose to manually manage your INI inventroy file, though this is less recommended.

In Ansible it is quite possible to use multiple dynamic inventory plugins along with INI file data. Just put them in a common directory and be sure the scripts are chmod +x, and the INI-based ones are not.

rax.py

To use the rackspace dynamic inventory script, copy `rax.py` from `plugins/inventory` into your inventory directory and make it executable. You can specify credentials for `rax.py` utilizing the `RAX_CREDS_FILE` environment variable.

Note: Users of *Ansible Tower* will note that dynamic inventory is natively supported by Tower, and all you have to do is associate a group with your Rackspace Cloud credentials, and it will easily synchronize without going through these steps:

```
$ RAX_CREDS_FILE=~/.raxpub ansible all -i rax.py -m setup
```

`rax.py` also accepts a `RAX_REGION` environment variable, which can contain an individual region, or a comma separated list of regions.

When using `rax.py`, you will not have a 'localhost' defined in the inventory.

As mentioned previously, you will often be running most of these modules outside of the host loop, and will need 'localhost' defined. The recommended way to do this, would be to create an `inventory` directory, and place both the `rax.py` script and a file containing `localhost` in it.

Executing `ansible` or `ansible-playbook` and specifying the `inventory` directory instead of an individual file, will cause ansible to evaluate each file in that directory for inventory.

Let's test our inventory script to see if it can talk to Rackspace Cloud.

```
$ RAX_CREDS_FILE=~/.raxpub ansible all -i inventory/ -m setup
```

Assuming things are properly configured, the `rax.py` inventory script will output information similar to the following information, which will be utilized for inventory and variables.

```
{
    "ORD": [
        "test"
    ],
    "_meta": {
        "hostvars": {
            "test": {
                "ansible_ssh_host": "1.1.1.1",
                "rax_accessipv4": "1.1.1.1",
```

```
"rax_accessipv6": "2607:f0d0:1002:51::4",
"rax_addresses": {
    "private": [
        {
            "addr": "2.2.2.2",
            "version": 4
        }
    ],
    "public": [
        {
            "addr": "1.1.1.1",
            "version": 4
        },
        {
            "addr": "2607:f0d0:1002:51::4",
            "version": 6
        }
    ]
},
"rax_config_drive": "",
"rax_created": "2013-11-14T20:48:22Z",
"rax_flavor": {
    "id": "performance1-1",
    "links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/flavors/perfo
            "rel": "bookmark"
        }
    ]
},
"rax_hostid": "e7b6961a9bd943ee82b13816426f1563bfda6846aad84d52af45a4904660cde0",
"rax_human_id": "test",
"rax_id": "099a447b-a644-471f-87b9-a7f580eb0c2a",
"rax_image": {
    "id": "b211c7bf-b5b4-4ede-a8de-a4368750c653",
    "links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/images/b211c7
            "rel": "bookmark"
        }
    ]
},
"rax_key_name": null,
"rax_links": [
    {
        "href": "https://ord.servers.api.rackspacecloud.com/v2/111111/servers/099a44
        "rel": "self"
    },
    {
        "href": "https://ord.servers.api.rackspacecloud.com/111111/servers/099a447b-a
        "rel": "bookmark"
    }
],
"rax_metadata": {
    "foo": "bar"
},
"rax_name": "test",
"rax_name_attr": "name",
```

```
                "rax_networks": {
                    "private": [
                        "2.2.2.2"
                    ],
                    "public": [
                        "1.1.1.1",
                        "2607:f0d0:1002:51::4"
                    ]
                },
                "rax_os-dcf_diskconfig": "AUTO",
                "rax_os-ext-sts_power_state": 1,
                "rax_os-ext-sts_task_state": null,
                "rax_os-ext-sts_vm_state": "active",
                "rax_progress": 100,
                "rax_status": "ACTIVE",
                "rax_tenant_id": "111111",
                "rax_updated": "2013-11-14T20:49:27Z",
                "rax_user_id": "22222"
            }
        }
    }
}
```

Standard Inventory

When utilizing a standard ini formatted inventory file (as opposed to the inventory plugin), it may still be adventageous to retrieve discoverable hostvar information from the Rackspace API.

This can be achieved with the `rax_facts` module and an inventory file similar to the following:

```
[test_servers]
hostname1 rax_region=ORD
hostname2 rax_region=ORD

- name: Gather info about servers
  hosts: test_servers
  gather_facts: False
  tasks:
    - name: Get facts about servers
      local_action:
        module: rax_facts
        credentials: ~/.raxpub
        name: "{{ inventory_hostname }}"
        region: "{{ rax_region }}"
    - name: Map some facts
      set_fact:
        ansible_ssh_host: "{{ rax_accessipv4 }}"
```

While you don't need to know how it works, it may be interesting to know what kind of variables are returned.

The `rax_facts` module provides facts as followings, which match the `rax.py` inventory script:

```
{
    "ansible_facts": {
        "rax_accessipv4": "1.1.1.1",
        "rax_accessipv6": "2607:f0d0:1002:51::4",
        "rax_addresses": {
            "private": [
```

```
            {
                "addr": "2.2.2.2",
                "version": 4
            }
        ],
        "public": [
            {
                "addr": "1.1.1.1",
                "version": 4
            },
            {
                "addr": "2607:f0d0:1002:51::4",
                "version": 6
            }
        ]
    },
    "rax_config_drive": "",
    "rax_created": "2013-11-14T20:48:22Z",
    "rax_flavor": {
        "id": "performance1-1",
        "links": [
            {
                "href": "https://ord.servers.api.rackspacecloud.com/111111/flavors/performance1-1
                "rel": "bookmark"
            }
        ]
    },
    "rax_hostid": "e7b6961a9bd943ee82b13816426f1563bfda6846aad84d52af45a4904660cde0",
    "rax_human_id": "test",
    "rax_id": "099a447b-a644-471f-87b9-a7f580eb0c2a",
    "rax_image": {
        "id": "b211c7bf-b5b4-4ede-a8de-a4368750c653",
        "links": [
            {
                "href": "https://ord.servers.api.rackspacecloud.com/111111/images/b211c7bf-b5b4-4
                "rel": "bookmark"
            }
        ]
    },
    "rax_key_name": null,
    "rax_links": [
        {
            "href": "https://ord.servers.api.rackspacecloud.com/v2/111111/servers/099a447b-a644-4
            "rel": "self"
        },
        {
            "href": "https://ord.servers.api.rackspacecloud.com/111111/servers/099a447b-a644-471
            "rel": "bookmark"
        }
    ],
    "rax_metadata": {
        "foo": "bar"
    },
    "rax_name": "test",
    "rax_name_attr": "name",
    "rax_networks": {
        "private": [
            "2.2.2.2"
```

```
            ],
            "public": [
                "1.1.1.1",
                "2607:f0d0:1002:51::4"
            ]
        },
        "rax_os-dcf_diskconfig": "AUTO",
        "rax_os-ext-sts_power_state": 1,
        "rax_os-ext-sts_task_state": null,
        "rax_os-ext-sts_vm_state": "active",
        "rax_progress": 100,
        "rax_status": "ACTIVE",
        "rax_tenant_id": "111111",
        "rax_updated": "2013-11-14T20:49:27Z",
        "rax_user_id": "22222"
    },
    "changed": false
}
```

Use Cases

This section covers some additional usage examples built around a specific use case.

Example 1

Create an isolated cloud network and build a server

```
- name: Build Servers on an Isolated Network
  hosts: localhost
  connection: local
  gather_facts: False
  tasks:
    - name: Network create request
      local_action:
        module: rax_network
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        region: IAD
        state: present

    - name: Server create request
      local_action:
        module: rax
        credentials: ~/.raxpub
        name: web%04d.example.org
        flavor: 2
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        networks:
          - public
          - my-net
        region: IAD
        state: present
        count: 5
        exact_count: yes
```

```
        group: web
        wait: yes
        wait_timeout: 360
      register: rax
```

Example 2

Build a complete webserver environment with servers, custom networks and load balancers, install nginx and create a custom index.html

```
---
- name: Build environment
  hosts: localhost
  connection: local
  gather_facts: False
  tasks:
    - name: Load Balancer create request
      local_action:
        module: rax_clb
        credentials: ~/.raxpub
        name: my-lb
        port: 80
        protocol: HTTP
        algorithm: ROUND_ROBIN
        type: PUBLIC
        timeout: 30
        region: IAD
        wait: yes
        state: present
        meta:
          app: my-cool-app
      register: clb

    - name: Network create request
      local_action:
        module: rax_network
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        state: present
        region: IAD
      register: network

    - name: Server create request
      local_action:
        module: rax
        credentials: ~/.raxpub
        name: web%04d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        networks:
          - public
          - private
          - my-net
        region: IAD
        state: present
```

```
        count: 5
        exact_count: yes
        group: web
        wait: yes
      register: rax

    - name: Add servers to web host group
      local_action:
        module: add_host
        hostname: "{{ item.name }}"
        ansible_ssh_host: "{{ item.rax_accessipv4 }}"
        ansible_ssh_pass: "{{ item.rax_adminpass }}"
        ansible_ssh_user: root
        groupname: web
      with_items: rax.success
      when: rax.action == 'create'

    - name: Add servers to Load balancer
      local_action:
        module: rax_clb_nodes
        credentials: ~/.raxpub
        load_balancer_id: "{{ clb.balancer.id }}"
        address: "{{ item.rax_networks.private|first }}"
        port: 80
        condition: enabled
        type: primary
        wait: yes
        region: IAD
      with_items: rax.success
      when: rax.action == 'create'

- name: Configure servers
  hosts: web
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted

  tasks:
    - name: Install nginx
      apt: pkg=nginx state=latest update_cache=yes cache_valid_time=86400
      notify:
        - restart nginx

    - name: Ensure nginx starts on boot
      service: name=nginx state=started enabled=yes

    - name: Create custom index.html
      copy: content="{{ inventory_hostname }}" dest=/usr/share/nginx/www/index.html
            owner=root group=root mode=0644
```

Advanced Usage

Autoscaling with Tower

Ansible Tower also contains a very nice feature for auto-scaling use cases. In this mode, a simple curl script can call a defined URL and the server will "dial out" to the requester and configure an instance that is spinning up. This can be

a great way to reconfigure ephmeral nodes. See the Tower documentation for more details.

A benefit of using the callback in Tower over pull mode is that job results are still centrally recorded and less information has to be shared with remote hosts.

Pending Information

More to come!

1.7.3 Using Vagrant and Ansible

Introduction

Vagrant is a tool to manage virtual machine environments, and allows you to configure and use reproducable work environments on top of various virtualization and cloud platforms. It also has integration with Ansible as a provisioner for these virtual machines, and the two tools work together well.

This guide will describe how to use Vagrant and Ansible together.

If you're not familar with Vagrant, you should visit the documentation.

This guide assumes that you already have Ansible installed and working. Running from a Git checkout is fine. Follow the *Installation* guide for more information.

Vagrant Setup

The first step once you've installed Vagrant is to create a `Vagrantfile` and customize it to suit your needs. This is covered in detail in the Vagrant documentation, but here is a quick example:

```
$ mkdir vagrant-test
$ cd vagrant-test
$ vagrant init precise32 http://files.vagrantup.com/precise32.box
```

This will create a file called Vagrantfile that you can edit to suit your needs. The default Vagrantfile has a lot of comments. Here is a simplified example that includes a section to use the Ansible provisioner:

```
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "precise32"
    config.vm.box_url = "http://files.vagrantup.com/precise32.box"

    config.vm.network :public_network

    config.vm.provision "ansible" do |ansible|
        ansible.playbook = "playbook.yml"
    end
end
```

The Vagrantfile has a lot of options, but these are the most important ones. Notice the `config.vm.provision` section that refers to an Ansible playbook called `playbook.yml` in the same directory as the Vagrantfile. Vagrant runs the provisioner once the virtual machine has booted and is ready for SSH access.

```
$ vagrant up
```

This will start the VM and run the provisioning playbook.

There are a lot of Ansible options you can configure in your Vagrantfile. Some particularly useful options are `ansible.extra_vars`, `ansible.sudo` and `ansible.sudo_user`, and `ansible.host_key_checking` which you can disable to avoid SSH connection problems to new virtual machines.

Visit the Ansible Provisioner documentation for more information.

To re-run a playbook on an existing VM, just run:

```
$ vagrant provision
```

This will re-run the playbook.

Running Ansible Manually

Sometimes you may want to run Ansible manually against the machines. This is pretty easy to do.

Vagrant automatically creates an inventory file for each Vagrant machine in the same directory called `vagrant_ansible_inventory_machinename`. It configures the inventory file according to the SSH tunnel that Vagrant automatically creates, and executes `ansible-playbook` with the correct username and SSH key options to allow access. A typical automatically-created inventory file may look something like this:

```
# Generated by Vagrant

machine ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222
```

If you want to run Ansible manually, you will want to make sure to pass `ansible` or `ansible-playbook` commands the correct arguments for the username (usually `vagrant`) and the SSH key (usually `~/.vagrant.d/insecure_private_key`), and the autogenerated inventory file.

Here is an example:

```
$ ansible-playbook -i vagrant_ansible_inventory_machinename --private-key=~/.vagrant.d/insecure_priva
```

See also:

Vagrant Home The Vagrant homepage with downloads

Vagrant Documentation Vagrant Documentation

Ansible Provisioner The Vagrant documentation for the Ansible provisioner

Playbooks An introduction to playbooks

1.7.4 Continuous Delivery and Rolling Upgrades

Introduction

Continuous Delivery is the concept of frequently delivering updates to your software application.

The idea is that by updating more often, you do not have to wait for a specific timed period, and your organization gets better at the process of responding to change.

Some Ansible users are deploying updates to their end users on an hourly or even more frequent basis – sometimes every time there is an approved code change. To achieve this, you need tools to be able to quickly apply those updates in a zero-downtime way.

This document describes in detail how to achieve this goal, using one of Ansible's most complete example playbooks as a template: lamp_haproxy. This example uses a lot of Ansible features: roles, templates, and group variables, and it also comes with an orchestration playbook that can do zero-downtime rolling upgrades of the web application stack.

Note: Click here for the latest playbooks for this example.

The playbooks deploy Apache, PHP, MySQL, Nagios, and HAProxy to a CentOS-based set of servers.

We're not going to cover how to run these playbooks here. Read the included README in the github project along with the example for that information. Instead, we're going to take a close look at every part of the playbook and describe what it does.

Site Deployment

Let's start with `site.yml`. This is our site-wide deployment playbook. It can be used to initially deploy the site, as well as push updates to all of the servers:

```
---
# This playbook deploys the whole application stack in this site.

# Apply common configuration to all hosts
- hosts: all

  roles:
  - common

# Configure and deploy database servers.
- hosts: dbservers

  roles:
  - db

# Configure and deploy the web servers. Note that we include two roles
# here, the 'base-apache' role which simply sets up Apache, and 'web'
# which includes our example web application.

- hosts: webservers

  roles:
  - base-apache
  - web

# Configure and deploy the load balancer(s).
- hosts: lbservers

  roles:
  - haproxy

# Configure and deploy the Nagios monitoring node(s).
- hosts: monitoring

  roles:
  - base-apache
  - nagios
```

Note: If you're not familiar with terms like playbooks and plays, you should review *Playbooks*.

In this playbook we have 5 plays. The first one targets `all` hosts and applies the `common` role to all of the hosts. This is for site-wide things like yum repository configuration, firewall configuration, and anything else that needs to apply to all of the servers.

The next four plays run against specific host groups and apply specific roles to those servers. Along with the roles for Nagios monitoring, the database, and the web application, we've implemented a `base-apache` role that installs and configures a basic Apache setup. This is used by both the sample web application and the Nagios hosts.

Reusable Content: Roles

By now you should have a bit of understanding about roles and how they work in Ansible. Roles are a way to organize content: tasks, handlers, templates, and files, into reusable components.

This example has six roles: `common`, `base-apache`, `db`, `haproxy`, `nagios`, and `web`. How you organize your roles is up to you and your application, but most sites will have one or more common roles that are applied to all systems, and then a series of application-specific roles that install and configure particular parts of the site.

Roles can have variables and dependencies, and you can pass in parameters to roles to modify their behavior. You can read more about roles in the *Playbook Roles and Include Statements* section.

Configuration: Group Variables

Group variables are variables that are applied to groups of servers. They can be used in templates and in playbooks to customize behavior and to provide easily-changed settings and parameters. They are stored in a directory called `group_vars` in the same location as your inventory. Here is lamp_haproxy's `group_vars/all` file. As you might expect, these variables are applied to all of the machines in your inventory:

```
---
httpd_port: 80
ntpserver: 192.168.1.2
```

This is a YAML file, and you can create lists and dictionaries for more complex variable structures. In this case, we are just setting two variables, one for the port for the web server, and one for the NTP server that our machines should use for time synchronization.

Here's another group variables file. This is `group_vars/dbservers` which applies to the hosts in the `dbservers` group:

```
---
mysqlservice: mysqld
mysql_port: 3306
dbuser: root
dbname: foodb
upassword: usersecret
```

If you look in the example, there are group variables for the `webservers` group and the `lbservers` group, similarly.

These variables are used in a variety of places. You can use them in playbooks, like this, in `roles/db/tasks/main.yml`:

```
- name: Create Application Database
  mysql_db: name={{ dbname }} state=present

- name: Create Application DB User
  mysql_user: name={{ dbuser }} password={{ upassword }}
              priv=*.*:ALL host='%' state=present
```

You can also use these variables in templates, like this, in `roles/common/templates/ntp.conf.j2`:

```
driftfile /var/lib/ntp/drift

restrict 127.0.0.1
restrict -6 ::1

server {{ ntpserver }}

includefile /etc/ntp/crypto/pw

keys /etc/ntp/keys
```

You can see that the variable substitution syntax of {{ and }} is the same for both templates and variables. The syntax inside the curly braces is Jinja2, and you can do all sorts of operations and apply different filters to the data inside. In templates, you can also use for loops and if statements to handle more complex situations, like this, in `roles/common/templates/iptables.j2`:

```
{% if inventory_hostname in groups['dbservers'] %}
-A INPUT -p tcp  --dport 3306 -j  ACCEPT
{% endif %}
```

This is testing to see if the inventory name of the machine we're currently operating on (`inventory_hostname`) exists in the inventory group `dbservers`. If so, that machine will get an iptables ACCEPT line for port 3306.

Here's another example, from the same template:

```
{% for host in groups['monitoring'] %}
-A INPUT -p tcp -s {{ hostvars[host].ansible_default_ipv4.address }} --dport 5666 -j ACCEPT
{% endfor %}
```

This loops over all of the hosts in the group called `monitoring`, and adds an ACCEPT line for each monitoring hosts's default IPV4 address to the current machine's iptables configuration, so that Nagios can monitor those hosts.

You can learn a lot more about Jinja2 and its capabilities here, and you can read more about Ansible variables in general in the *Variables* section.

The Rolling Upgrade

Now you have a fully-deployed site with web servers, a load balancer, and monitoring. How do you update it? This is where Ansible's orchestration features come into play. While some applications use the term 'orchestration' to mean basic ordering or command-blasting, Ansible referes to orchestration as 'conducting machines like an orchestra', and has a pretty sophisticated engine for it.

Ansible has the capability to do operations on multi-tier applications in a coordinated way, making it easy to orchestrate a sophisticated zero-downtime rolling upgrade of our web application. This is implemented in a separate playbook, called `rolling_upgrade.yml`.

Looking at the playbook, you can see it is made up of two plays. The first play is very simple and looks like this:

```
- hosts: monitoring
  tasks: []
```

What's going on here, and why are there no tasks? You might know that Ansible gathers "facts" from the servers before operating upon them. These facts are useful for all sorts of things: networking information, OS/distribution versions, etc. In our case, we need to know something about all of the monitoring servers in our environment before we perform the update, so this simple play forces a fact-gathering step on our monitoring servers. You will see this pattern sometimes, and it's a useful trick to know.

The next part is the update play. The first part looks like this:

```
- hosts: webservers
  user: root
  serial: 1
```

This is just a normal play definition, operating on the `webservers` group. The `serial` keyword tells Ansible how many servers to operate on at once. If it's not specified, Ansible will paralleize these operations up to the default "forks" limit specified in the configuration file. But for a zero-downtime rolling upgrade, you may not want to operate on that many hosts at once. If you had just a handful of webservers, you may want to set `serial` to 1, for one host at a time. If you have 100, maybe you could set `serial` to 10, for ten at a time.

Here is the next part of the update play:

```
pre_tasks:
- name: disable nagios alerts for this host webserver service
  nagios: action=disable_alerts host={{ ansible_hostname }} services=webserver
  delegate_to: "{{ item }}"
  with_items: groups.monitoring

- name: disable the server in haproxy
  shell: echo "disable server myapplb/{{ ansible_hostname }}" | socat stdio /var/lib/haproxy/stats
  delegate_to: "{{ item }}"
  with_items: groups.lbservers
```

The `pre_tasks` keyword just lets you list tasks to run before the roles are called. This will make more sense in a minute. If you look at the names of these tasks, you can see that we are disabling Nagios alerts and then removing the webserver that we are currently updating from the HAProxy load balancing pool.

The `delegate_to` and `with_items` arguments, used together, cause Ansible to loop over each monitoring server and load balancer, and perform that operation (delegate that operation) on the monitoring or load balancing server, "on behalf" of the webserver. In programming terms, the outer loop is the list of web servers, and the inner loop is the list of monitoring servers.

Note that the HAProxy step looks a little complicated. We're using HAProxy in this example because it's freely available, though if you have (for instance) an F5 or Netscaler in your infrastructure (or maybe you have an AWS Elastic IP setup?), you can use modules included in core Ansible to communicate with them instead. You might also wish to use other monitoring modules instead of nagios, but this just shows the main goal of the 'pre tasks' section – take the server out of monitoring, and take it out of rotation.

The next step simply re-applies the proper roles to the web servers. This will cause any configuration management declarations in `web` and `base-apache` roles to be applied to the web servers, including an update of the web application code itself. We don't have to do it this way–we could instead just purely update the web application, but this is a good example of how roles can be used to reuse tasks:

```
roles:
- common
- base-apache
- web
```

Finally, in the `post_tasks` section, we reverse the changes to the Nagios configuration and put the web server back in the load balancing pool:

```
post_tasks:
- name: Enable the server in haproxy
  shell: echo "enable server myapplb/{{ ansible_hostname }}" | socat stdio /var/lib/haproxy/stats
  delegate_to: "{{ item }}"
  with_items: groups.lbservers

- name: re-enable nagios alerts
```

```
nagios: action=enable_alerts host={{ ansible_hostname }} services=webserver
delegate_to: "{{ item }}"
with_items: groups.monitoring
```

Again, if you were using a Netscaler or F5 or Elastic Load Balancer, you would just substitute in the appropriate modules instead.

Managing Other Load Balancers

In this example, we use the simple HAProxy load balancer to front-end the web servers. It's easy to configure and easy to manage. As we have mentioned, Ansible has built-in support for a variety of other load balancers like Citrix NetScaler, F5 BigIP, Amazon Elastic Load Balancers, and more. See the *About Modules* documentation for more information.

For other load balancers, you may need to send shell commands to them (like we do for HAProxy above), or call an API, if your load balancer exposes one. For the load balancers for which Ansible has modules, you may want to run them as a `local_action` if they contact an API. You can read more about local actions in the *Delegation, Rolling Updates, and Local Actions* section. Should you develop anything interesting for some hardware where there is not a core module, it might make for a good module for core inclusion!

Continuous Delivery End-To-End

Now that you have an automated way to deploy updates to your application, how do you tie it all together? A lot of organizations use a continuous integration tool like Jenkins or Atlassian Bamboo to tie the development, test, release, and deploy steps together. You may also want to use a tool like Gerrit to add a code review step to commits to either the application code itself, or to your Ansible playbooks, or both.

Depending on your environment, you might be deploying continuously to a test environment, running an integration test battery against that environment, and then deploying automatically into production. Or you could keep it simple and just use the rolling-update for on-demand deployment into test or production specifically. This is all up to you.

For integration with Continuous Integration systems, you can easily trigger playbook runs using the `ansible-playbook` command line tool, or, if you're using *Ansible Tower*, the `tower-cli` or the built-in REST API. (The tower-cli command 'joblaunch' will spawn a remote job over the REST API and is pretty slick).

This should give you a good idea of how to structure a multi-tier application with Ansible, and orchestrate operations upon that app, with the eventual goal of continuous delivery to your customers. You could extend the idea of the rolling upgrade to lots of different parts of the app; maybe add front-end web servers along with application servers, for instance, or replace the SQL database with something like MongoDB or Riak. Ansible gives you the capability to easily manage complicated environments and automate common operations.

See also:

lamp_haproxy example The lamp_haproxy example discussed here.

Playbooks An introduction to playbooks

Playbook Roles and Include Statements An introduction to playbook roles

Variables An introduction to Ansible variables

Ansible.com: Continuous Delivery An introduction to Continuous Delivery with Ansible

Pending topics may include: Docker, Jenkins, Google Compute Engine, Linode/Digital Ocean, Continous Deployment, and more.

1.8 Developer Information

Learn how to build modules of your own in any language, and also how to extend Ansible through several kinds of plugins. Explore Ansible's Python API and write Python plugins to integrate with other solutions in your environment.

1.8.1 Python API

Topics

- Python API
  - Python API
    - Detailed API Example

There are several interesting ways to use Ansible from an API perspective. You can use the Ansible python API to control nodes, you can extend Ansible to respond to various python events, you can write various plugins, and you can plug in inventory data from external data sources. This document covers the Runner and Playbook API at a basic level.

If you are looking to use Ansible programmatically from something other than Python, trigger events asynchronously, or have access control and logging demands, take a look at *Ansible Tower* as it has a very nice REST API that provides all of these things at a higher level.

Ansible is written in its own API so you have a considerable amount of power across the board. This chapter discusses the Python API.

Python API

The Python API is very powerful, and is how the ansible CLI and ansible-playbook are implemented.

It's pretty simple:

```
import ansible.runner

runner = ansible.runner.Runner(
   module_name='ping',
   module_args='',
   pattern='web*',
   forks=10
)
datastructure = runner.run()
```

The run method returns results per host, grouped by whether they could be contacted or not. Return types are module specific, as expressed in the *About Modules* documentation.:

```
{
    "dark" : {
       "web1.example.com" : "failure message"
    },
    "contacted" : {
       "web2.example.com" : 1
    }
}
```

A module can return any type of JSON data it wants, so Ansible can be used as a framework to rapidly build powerful applications and scripts.

Detailed API Example

The following script prints out the uptime information for all hosts:

```
#!/usr/bin/python

import ansible.runner
import sys

# construct the ansible runner and execute on all hosts
results = ansible.runner.Runner(
    pattern='*', forks=10,
    module_name='command', module_args='/usr/bin/uptime',
).run()

if results is None:
   print "No hosts found"
   sys.exit(1)

print "UP ***********"
for (hostname, result) in results['contacted'].items():
    if not 'failed' in result:
        print "%s >>> %s" % (hostname, result['stdout'])

print "FAILED *******"
for (hostname, result) in results['contacted'].items():
    if 'failed' in result:
        print "%s >>> %s" % (hostname, result['msg'])

print "DOWN *********"
for (hostname, result) in results['dark'].items():
    print "%s >>> %s" % (hostname, result)
```

Advanced programmers may also wish to read the source to ansible itself, for it uses the Runner() API (with all available options) to implement the command line tools `ansible` and `ansible-playbook`.

See also:

Developing Dynamic Inventory Sources Developing dynamic inventory integrations

Developing Modules How to develop modules

Developing Plugins How to develop plugins

Development Mailing List Mailing list for development topics

irc.freenode.net #ansible IRC chat channel

1.8.2 Developing Dynamic Inventory Sources

Topics

- Script Conventions
- Tuning the External Inventory Script

As described in *Dynamic Inventory*, ansible can pull inventory information from dynamic sources, including cloud sources.

How do we write a new one?

Simple! We just create a script or program that can return JSON in the right format when fed the proper arguments. You can do this in any language.

Script Conventions

When the external node script is called with the single argument `--list`, the script must return a JSON hash/dictionary of all the groups to be managed. Each group's value should be either a hash/dictionary containing a list of each host/IP, potential child groups, and potential group variables, or simply a list of host/IP addresses, like so:

```
{
    "databases"   : {
        "hosts"   : [ "host1.example.com", "host2.example.com" ],
        "vars"    : {
            "a"   : true
        }
    },
    "webservers"  : [ "host2.example.com", "host3.example.com" ],
    "atlanta"     : {
        "hosts"   : [ "host1.example.com", "host4.example.com", "host5.example.com" ],
        "vars"    : {
            "b"   : false
        },
        "children": [ "marietta", "5points" ]
    },
    "marietta"    : [ "host6.example.com" ],
    "5points"     : [ "host7.example.com" ]
}
```

New in version 1.0.

Before version 1.0, each group could only have a list of hostnames/IP addresses, like the webservers, marietta, and 5points groups above.

When called with the arguments `--host <hostname>` (where <hostname> is a host from above), the script must return either an empty JSON hash/dictionary, or a hash/dictionary of variables to make available to templates and playbooks. Returning variables is optional, if the script does not wish to do this, returning an empty hash/dictionary is the way to go:

```
{
    "favcolor"   : "red",
    "ntpserver"  : "wolf.example.com",
    "monitoring" : "pack.example.com"
}
```

Tuning the External Inventory Script

New in version 1.3.

The stock inventory script system detailed above works for all versions of Ansible, but calling `--host` for every host can be rather expensive, especially if it involves expensive API calls to a remote subsystem. In Ansible 1.3 or later, if the inventory script returns a top level element called "_meta", it is possible to return all of the host variables in one inventory script call. When this meta element contains a value for "hostvars", the inventory script will not be invoked with `--host` for each host. This results in a significant performance increase for large numbers of hosts, and also makes client side caching easier to implement for the inventory script.

The data to be added to the top level JSON dictionary looks like this:

```
{

    # results of inventory script as above go here
    # ...

    "_meta" : {
       "hostvars" : {
          "moocow.example.com"     : { "asdf" : 1234 },
          "llama.example.com"      : { "asdf" : 5678 },
       }
    }

}
```

See also:

Python API Python API to Playbooks and Ad Hoc Task Execution

Developing Modules How to develop modules

Developing Plugins How to develop plugins

Ansible Tower REST API endpoint and GUI for Ansible, syncs with dynamic inventory

Development Mailing List Mailing list for development topics

irc.freenode.net #ansible IRC chat channel

1.8.3 Developing Modules

Topics

- Developing Modules
  - Tutorial
  - Testing Modules
  - Reading Input
  - Module Provided 'Facts'
  - Common Module Boilerplate
  - Check Mode
  - Common Pitfalls
  - Conventions/Recommendations
  - Shorthand Vs JSON
  - Documenting Your Module
    - Example
    - Building & Testing
  - Getting Your Module Into Core

Ansible modules are reusable units of magic that can be used by the Ansible API, or by the *ansible* or *ansible-playbook* programs.

See *About Modules* for a list of various ones developed in core.

Modules can be written in any language and are found in the path specified by *ANSIBLE_LIBRARY* or the `--module-path` command line option.

Should you develop an interesting Ansible module, consider sending a pull request to the github project to see about getting your module included in the core project.

Tutorial

Let's build a very-basic module to get and set the system time. For starters, let's build a module that just outputs the current time.

We are going to use Python here but any language is possible. Only File I/O and outputting to standard out are required. So, bash, C++, clojure, Python, Ruby, whatever you want is fine.

Now Python Ansible modules contain some extremely powerful shortcuts (that all the core modules use) but first we are going to build a module the very hard way. The reason we do this is because modules written in any language OTHER than Python are going to have to do exactly this. We'll show the easy way later.

So, here's an example. You would never really need to build a module to set the system time, the 'command' module could already be used to do this. Though we're going to make one.

Reading the modules that come with ansible (linked above) is a great way to learn how to write modules. Keep in mind, though, that some modules in ansible's source tree are internalisms, so look at *service* or *yum*, and don't stare too close into things like *async_wrapper* or you'll turn to stone. Nobody ever executes async_wrapper directly.

Ok, let's get going with an example. We'll use Python. For starters, save this as a file named *time*:

```
#!/usr/bin/python

import datetime
import json

date = str(datetime.datetime.now())
print json.dumps({
    "time" : date
})
```

Testing Modules

There's a useful test script in the source checkout for ansible:

```
git clone git@github.com:ansible/ansible.git
source ansible/hacking/env-setup
chmod +x ansible/hacking/test-module
```

Let's run the script you just wrote with that:

```
ansible/hacking/test-module -m ./time
```

You should see output that looks something like this:

```
{u'time': u'2012-03-14 22:13:48.539183'}
```

If you did not, you might have a typo in your module, so recheck it and try again.

Reading Input

Let's modify the module to allow setting the current time. We'll do this by seeing if a key value pair in the form *time=<string>* is passed in to the module.

Ansible internally saves arguments to an arguments file. So we must read the file and parse it. The arguments file is just a string, so any form of arguments are legal. Here we'll do some basic parsing to treat the input as key=value.

The example usage we are trying to achieve to set the time is:

```
time time="March 14 22:10"
```

If no time parameter is set, we'll just leave the time as is and return the current time.

Note: This is obviously an unrealistic idea for a module. You'd most likely just use the shell module. However, it probably makes a decent tutorial.

Let's look at the code. Read the comments as we'll explain as we go. Note that this is highly verbose because it's intended as an educational example. You can write modules a lot shorter than this:

```
#!/usr/bin/python

# import some python modules that we'll use.  These are all
# available in Python's core

import datetime
import sys
import json
import os
import shlex

# read the argument string from the arguments file
args_file = sys.argv[1]
args_data = file(args_file).read()

# for this module, we're going to do key=value style arguments
# this is up to each module to decide what it wants, but all
# core modules besides 'command' and 'shell' take key=value
# so this is highly recommended

arguments = shlex.split(args_data)
for arg in arguments:

    # ignore any arguments without an equals in it
    if arg.find("=") != -1:

        (key, value) = arg.split("=")

        # if setting the time, the key 'time'
        # will contain the value we want to set the time to

        if key == "time":

            # now we'll affect the change.  Many modules
            # will strive to be 'idempotent', meaning they
            # will only make changes when the desired state
            # expressed to the module does not match
            # the current state.  Look at 'service'
            # or 'yum' in the main git tree for an example
            # of how that might look.

            rc = os.system("date -s \"%s\"" % value)

            # always handle all possible errors
```

```
            #
            # when returning a failure, include 'failed'
            # in the return data, and explain the failure
            # in 'msg'.  Both of these conventions are
            # required however additional keys and values
            # can be added.

            if rc != 0:
                print json.dumps({
                    "failed" : True,
                    "msg"    : "failed setting the time"
                })
                sys.exit(1)

            # when things do not fail, we do not
            # have any restrictions on what kinds of
            # data are returned, but it's always a
            # good idea to include whether or not
            # a change was made, as that will allow
            # notifiers to be used in playbooks.

            date = str(datetime.datetime.now())
            print json.dumps({
                "time" : date,
                "changed" : True
            })
            sys.exit(0)

# if no parameters are sent, the module may or
# may not error out, this one will just
# return the time

date = str(datetime.datetime.now())
print json.dumps({
    "time" : date
})
```

Let's test that module:

```
ansible/hacking/test-module -m ./time -a time=\"March 14 12:23\"
```

This should return something like:

```
{"changed": true, "time": "2012-03-14 12:23:00.000307"}
```

Module Provided 'Facts'

The 'setup' module that ships with Ansible provides many variables about a system that can be used in playbooks and templates. However, it's possible to also add your own facts without modifying the system module. To do this, just have the module return a *ansible_facts* key, like so, along with other return data:

```
{
    "changed" : True,
    "rc" : 5,
    "ansible_facts" : {
        "leptons" : 5000
        "colors" : {
```

```
            "red"   : "FF0000",
            "white" : "FFFFFF"
        }
    }
}
```

These 'facts' will be available to all statements called after that module (but not before) in the playbook. A good idea might be make a module called 'site_facts' and always call it at the top of each playbook, though we're always open to improving the selection of core facts in Ansible as well.

Common Module Boilerplate

As mentioned, if you are writing a module in Python, there are some very powerful shortcuts you can use. Modules are still transferred as one file, but an arguments file is no longer needed, so these are not only shorter in terms of code, they are actually FASTER in terms of execution time.

Rather than mention these here, the best way to learn is to read some of the source of the modules that come with Ansible.

The 'group' and 'user' modules are reasonably non-trivial and showcase what this looks like.

Key parts include always ending the module file with:

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

And instantiating the module class like:

```
module = AnsibleModule(
    argument_spec = dict(
        state     = dict(default='present', choices=['present', 'absent']),
        name      = dict(required=True),
        enabled   = dict(required=True, choices=BOOLEANS),
        something = dict(aliases=['whatever'])
    )
)
```

The AnsibleModule provides lots of common code for handling returns, parses your arguments for you, and allows you to check inputs.

Successful returns are made like this:

```
module.exit_json(changed=True, something_else=12345)
```

And failures are just as simple (where 'msg' is a required parameter to explain the error):

```
module.fail_json(msg="Something fatal happened")
```

There are also other useful functions in the module class, such as module.md5(path). See lib/ansible/module_common.py in the source checkout for implementation details.

Again, modules developed this way are best tested with the hacking/test-module script in the git source checkout. Because of the magic involved, this is really the only way the scripts can function outside of Ansible.

If submitting a module to ansible's core code, which we encourage, use of the AnsibleModule class is required.

Check Mode

New in version 1.1.

Modules may optionally support check mode. If the user runs Ansible in check mode, the module should try to predict whether changes will occur.

For your module to support check mode, you must pass `supports_check_mode=True` when instantiating the AnsibleModule object. The AnsibleModule.check_mode attribute will evaluate to True when check mode is enabled. For example:

```
module = AnsibleModule(
    argument_spec = dict(...),
    supports_check_mode=True
)

if module.check_mode:
    # Check if any changes would be made by don't actually make those changes
    module.exit_json(changed=check_if_system_state_would_be_changed())
```

Remember that, as module developer, you are responsible for ensuring that no system state is altered when the user enables check mode.

If your module does not support check mode, when the user runs Ansible in check mode, your module will simply be skipped.

Common Pitfalls

You should also never do this in a module:

```
print "some status message"
```

Because the output is supposed to be valid JSON. Except that's not quite true, but we'll get to that later.

Modules must not output anything on standard error, because the system will merge standard out with standard error and prevent the JSON from parsing. Capturing standard error and returning it as a variable in the JSON on standard out is fine, and is, in fact, how the command module is implemented.

If a module returns stderr or otherwise fails to produce valid JSON, the actual output will still be shown in Ansible, but the command will not succeed.

Always use the hacking/test-module script when developing modules and it will warn you about these kind of things.

Conventions/Recommendations

As a reminder from the example code above, here are some basic conventions and guidelines:

- If the module is addressing an object, the parameter for that object should be called 'name' whenever possible, or accept 'name' as an alias.
- If you have a company module that returns facts specific to your installations, a good name for this module is *site_facts*.
- Modules accepting boolean status should generally accept 'yes', 'no', 'true', 'false', or anything else a user may likely throw at them. The AnsibleModule common code supports this with "choices=BOOLEANS" and a module.boolean(value) casting function.
- Include a minimum of dependencies if possible. If there are dependencies, document them at the top of the module file, and have the module raise JSON error messages when the import fails.
- Modules must be self contained in one file to be auto-transferred by ansible.
- If packaging modules in an RPM, they only need to be installed on the control machine and should be dropped into /usr/share/ansible. This is entirely optional and up to you.

- Modules should return JSON or key=value results all on one line. JSON is best if you can do JSON. All return types must be hashes (dictionaries) although they can be nested. Lists or simple scalar values are not supported, though they can be trivially contained inside a dictionary.
- In the event of failure, a key of 'failed' should be included, along with a string explanation in 'msg'. Modules that raise tracebacks (stacktraces) are generally considered 'poor' modules, though Ansible can deal with these returns and will automatically convert anything unparseable into a failed result. If you are using the AnsibleModule common Python code, the 'failed' element will be included for you automatically when you call 'fail_json'.
- Return codes from modules are not actually not signficant, but continue on with 0=success and non-zero=failure for reasons of future proofing.
- As results from many hosts will be aggregated at once, modules should return only relevant output. Returning the entire contents of a log file is generally bad form.

Shorthand Vs JSON

To make it easier to write modules in bash and in cases where a JSON module might not be available, it is acceptable for a module to return key=value output all on one line, like this. The Ansible parser will know what to do:

```
somekey=1 somevalue=2 rc=3 favcolor=red
```

If you're writing a module in Python or Ruby or whatever, though, returning JSON is probably the simplest way to go.

Documenting Your Module

All modules included in the CORE distribution must have a `DOCUMENTATION` string. This string MUST be a valid YAML document which conforms to the schema defined below. You may find it easier to start writing your `DOCUMENTATION` string in an editor with YAML syntax highlighting before you include it in your Python file.

Example

See an example documentation string in the checkout under examples/DOCUMENTATION.yml.

Include it in your module file like this:

```
#!/usr/bin/env python
# Copyright header....

DOCUMENTATION = '''
---
module: modulename
short_description: This is a sentence describing the module
# ... snip ...
'''
```

The `description`, and `notes` fields support formatting with some special macros.

These formatting functions are `U()`, `M()`, `I()`, and `C()` for URL, module, italic, and constant-width respectively. It is suggested to use `C()` for file and option names, and `I()` when referencing parameters; module names should be specifies as `M(module)`.

Examples (which typically contain colons, quotes, etc.) are difficult to format with YAML, so these must be written in plain text in an `EXAMPLES` string within the module like this:

```
EXAMPLES = '''
- action: modulename opt1=arg1 opt2=arg2
'''
```

The EXAMPLES section, just like the documentation section, is required in all module pull requests for new modules.

Building & Testing

Put your completed module file into the 'library' directory and then run the command: `make webdocs`. The new 'modules.html' file will be built and appear in the 'docsite/' directory.

You can also test-build your docs one-by-one using the `module_formatter.py` script:

```
$ ./hacking/module_formatter.py -t man -M library/ -m git > ansible-git.1
$ man ./ansible-git.1
```

This will build a manpage for the git module, and look in the 'library/' directory for the module source. To see all the other output formats available:

```
$ ./hacking/module_formatter.py -t --help
```

Tip: If you're having a problem with the syntax of your YAML you can validate it on the YAML Lint website.

Tip: You can use ANSIBLE_KEEP_REMOTE_FILES=1 to prevent ansible from deleting the remote files so you can debug your module.

Getting Your Module Into Core

High-quality modules with minimal dependencies can be included in the core, but core modules (just due to the programming preferences of the developers) will need to be implemented in Python and use the AnsibleModule common code, and should generally use consistent arguments with the rest of the program. Stop by the mailing list to inquire about requirements if you like, and submit a github pull request to the main project.

See also:

About Modules Learn about available modules

Developing Plugins Learn about developing plugins

Python API Learn about the Python API for playbook and task execution

Github modules directory Browse source of core modules

Mailing List Development mailing list

irc.freenode.net #ansible IRC chat channel

1.8.4 Developing Plugins

Topics

- Developing Plugins
  - Connection Type Plugins
  - Lookup Plugins
  - Vars Plugins
  - Filter Plugins
  - Callbacks
    - Examples
    - Configuring
    - Development
  - Distributing Plugins

Ansible is pluggable in a lot of other ways separate from inventory scripts and callbacks. Many of these features are there to cover fringe use cases and are infrequently needed, and others are pluggable simply because they are there to implement core features in ansible and were most convenient to be made pluggable.

This section will explore these features, though they are generally not common in terms of things people would look to extend quite as often.

Connection Type Plugins

By default, ansible ships with a 'paramiko' SSH, native ssh (just called 'ssh'), 'local' connection type, and an accelerated connection type named 'fireball' (superseded in 1.3 by *Accelerated Mode*) – there are also some minor players like 'chroot' and 'jail'. All of these can be used in playbooks and with /usr/bin/ansible to decide how you want to talk to remote machines. The basics of these connection types are covered in the *Getting Started* section. Should you want to extend Ansible to support other transports (SNMP? Message bus? Carrier Pigeon?) it's as simple as copying the format of one of the existing modules and dropping it into the connection plugins directory. The value of 'smart' for a connection allows selection of paramiko or openssh based on system capabilities, and chooses 'ssh' if OpenSSH supports ControlPersist, in Ansible 1.2.1 an later. Previous versions did not support 'smart'.

More documentation on writing connection plugins is pending, though you can jump into lib/ansible/runner/connection_plugins and figure things out pretty easily.

Lookup Plugins

Language constructs like "with_fileglob" and "with_items" are implemented via lookup plugins. Just like other plugin types, you can write your own.

More documentation on writing connection plugins is pending, though you can jump into lib/ansible/runner/lookup_plugins and figure things out pretty easily.

Vars Plugins

Playbook constructs like 'host_vars' and 'group_vars' work via 'vars' plugins. They inject additional variable data into ansible runs that did not come from an inventory, playbook, or command line. Note that variables can also be returned from inventory, so in most cases, you won't need to write or understand vars_plugins.

More documentation on writing connection plugins is pending, though you can jump into lib/ansible/inventory/vars_plugins and figure things out pretty easily.

If you find yourself wanting to write a vars_plugin, it's more likely you should write an inventory script instead.

Filter Plugins

If you want more Jinja2 filters available in a Jinja2 template (filters like to_yaml and to_json are provided by default), they can be extended by writing a filter plugin. Most of the time, when someone comes up with an idea for a new filter they would like to make available in a playbook, we'll just include them in 'core.py' instead.

Jump into lib/ansible/runner/filter_plugins/ for details.

Callbacks

Callbacks are one of the more interesting plugin types. Adding additional callback plugins to Ansible allows for adding new behaviors when responding to events.

Examples

Example callbacks are shown in plugins/callbacks.

The log_plays callback is an example of how to intercept playbook events to a log file, and the mail callback sends email when playbooks complete.

The osx_say callback provided is particularly entertaining – it will respond with computer synthesized speech on OS X in relation to playbook events, and is guaranteed to entertain and/or annoy coworkers.

Configuring

To active a callback drop it in a callback directory as configured in *ansible.cfg*.

Development

More information will come later, though see the source of any of the existing callbacks and you should be able to get started quickly. They should be reasonably self explanatory.

Distributing Plugins

Plugins are loaded from both Python's site_packages (those that ship with ansible) and a configured plugins directory, which defaults to /usr/share/ansible/plugins, in a subfolder for each plugin type:

```
* action_plugins
* lookup_plugins
* callback_plugins
* connection_plugins
* filter_plugins
* vars_plugins
```

To change this path, edit the ansible configuration file.

In addition, plugins can be shipped in a subdirectory relative to a top-level playbook, in folders named the same as indicated above.

See also:

About Modules List of built-in modules

Python API Learn about the Python API for task execution

Developing Dynamic Inventory Sources Learn about how to develop dynamic inventory sources

Developing Modules Learn about how to write Ansible modules

Mailing List The development mailing list

irc.freenode.net #ansible IRC chat channel

Developers will also likely be interested in the fully-discoverable in *Ansible Tower*. It's great for embedding Ansible in all manner of applications.

1.9 Ansible Tower

Ansible Tower (formerly 'AWX') is a web-based solution that makes Ansible even more easy to use for IT teams of all kinds. It's designed to be the hub for all of your automation tasks.

Tower allows you to control access to who can access what, even allowing sharing of SSH credentials without someone being able to transfer those credentials. Inventory can be graphically managed or synced with a wide variety of cloud sources. It logs all of your jobs, integrates well with LDAP, and has an amazing browsable REST API. Command line tools are available for easy integration with Jenkins as well. Provisioning callbacks provide great support for autoscaling topologies.

Find out more about Tower features and how to download it on the Ansible Tower webpage. Tower is free for usage for up to 10 nodes, and comes bundled with amazing support from Ansible, Inc. As you would expect, Ansible is installed using Ansible playbooks!

1.10 Community Information

Ansible is an open source project designed to bring together developers and administrators of all kinds to collaborate on building IT automation solutions that work well for them. Should you wish to get more involved – whether in terms of just asking a question, helping other users, introducing new people to Ansible, or helping with the software or documentation, we welcome your contributions to the project.

Ways to interact

1.11 Ansible Galaxy

Ansible Galaxy, is a free site for finding, downloading, rating, and reviewing all kinds of community developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

You can sign up with social auth, and the download client 'ansible-galaxy' is included in Ansible 1.4.2 and later.

Read the "About" page on the Galaxy site for more information.

1.12 Frequently Asked Questions

Here are some commonly-asked questions and their answers.

1.12.1 How do I handle different machines needing different user accounts or ports to log in with?

Setting inventory variables in the inventory file is the easiest way.

For instance, suppose these hosts have different usernames and ports:

```
[webservers]
asdf.example.com  ansible_ssh_port=5000   ansible_ssh_user=alice
jkl.example.com   ansible_ssh_port=5001   ansible_ssh_user=bob
```

You can also dictate the connection type to be used, if you want:

```
[testcluster]
localhost           ansible_connection=local
/path/to/chroot1    ansible_connection=chroot
foo.example.com
bar.example.com
```

You may also wish to keep these in group variables instead, or file in them in a group_vars/<groupname> file. See the rest of the documentation for more information about how to organize variables.

1.12.2 How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?

Switch your default connection type in the configuration file to 'ssh', or use '-c ssh' to use Native OpenSSH for connections instead of the python paramiko library. In Ansible 1.2.1 and later, 'ssh' will be used by default if OpenSSH is new enough to support ControlPersist as an option.

Paramiko is great for starting out, but the OpenSSH type offers many advanced options. You will want to run Ansible from a machine new enough to support ControlPersist, if you are using this connection type. You can still manage older clients. If you are using RHEL 6, CentOS 6, SLES 10 or SLES 11 the version of OpenSSH is still a bit old, so consider managing from a Fedora or openSUSE client even though you are managing older nodes, or just use paramiko.

We keep paramiko as the default as if you are first installing Ansible on an EL box, it offers a better experience for new users.

1.12.3 How do I speed up management inside EC2?

Don't try to manage a fleet of EC2 machines from your laptop. Connect to a management node inside EC2 first and run Ansible from there.

1.12.4 How do I handle python pathing not having a Python 2.X in /usr/bin/python on a remote machine?

While you can write ansible modules in any language, most ansible modules are written in Python, and some of these are important core ones.

By default Ansible assumes it can find a /usr/bin/python on your remote system that is a 2.X version of Python, specifically 2.4 or higher.

Setting of an inventory variable 'ansible_python_interpreter' on any host will allow Ansible to auto-replace the interpreter used when executing python modules. Thus, you can point to any python you want on the system if /usr/bin/python on your system does not point to a Python 2.X interpreter.

Some Linux operating systems, such as Arch, may only have Python 3 installed by default. This is not sufficient and you will get syntax errors trying to run modules with Python 3. Python 3 is essentially not the same language as Python 2. Ansible modules currently need to support older Pythons for users that still have Enterprise Linux 5 deployed, so they are not yet ported to run under Python 3.0. This is not a problem though as you can just install Python 2 also on a managed host.

Python 3.0 support will likely be addressed at a later point in time when usage becomes more mainstream.

Do not replace the shebang lines of your python modules. Ansible will do this for you automatically at deploy time.

1.12.5 What is the best way to make content reusable/redistributable?

If you have not done so already, read all about "Roles" in the playbooks documentation. This helps you make playbook content self contained, and works will with things like git submodules for sharing content with others.

If some of these plugin types look strange to you, see the API documentation for more details about ways Ansible can be extended.

1.12.6 Where does the configuration file live and what can I configure in it?

See *The Ansible Configuration File*.

1.12.7 How do I disable cowsay?

If cowsay is installed, Ansible takes it upon itself to make your day happier when running playbooks. If you decide that you would like to work in a professional cow-free environment, you can either uninstall cowsay, or set an environment variable:

```
export ANSIBLE_NOCOWS=1
```

1.12.8 How do I see a list of all of the ansible_ variables?

Ansible by default gathers "facts" about the machines under management, and these facts can be accessed in Playbooks and in templates. To see a list of all of the facts that are available about a machine, you can run the "setup" module as an ad-hoc action:

```
ansible -m setup hostname
```

This will print out a dictionary of all of the facts that are available for that particular host.

1.12.9 How do I loop over a list of hosts in a group, inside of a template?

A pretty common pattern is to iterate over a list of hosts inside of a host group, perhaps to populate a template configuration file with a list of servers. To do this, you can just access the "$groups" dictionary in your template, like this:

```
{% for host in groups['db_servers'] %}
    {{ host }}
{% endfor %}
```

If you need to access facts about these hosts, for instance, the IP address of each hostname, you need to make sure that the facts have been populated. For example, make sure you have a play that talks to db_servers:

```
- hosts:  db_servers
  tasks:
    - # doesn't matter what you do, just that they were talked to previously.
```

Then you can use the facts inside your template, like this:

```
{% for host in groups['db_servers'] %}
   {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
{% endfor %}
```

1.12.10 How do I access a variable name programatically?

An example may come up where we need to get the ipv4 address of an arbitrary interface, where the interface to be used may be supplied via a role parameter or other input. Variable names can be built by adding strings together, like so:

```
{{ hostvars[inventory_hostname]['ansible_' + which_interface]['ipv4']['address'] }}
```

The trick about going through hostvars is neccessary because it's a dictionary of the entire namespace of variables. 'inventory_hostname' is a magic variable that indiciates the current host you are looping over in the host loop.

1.12.11 How do I access a variable of the first host in a group?

What happens if we want the ip address of the first webserver in the webservers group? Well, we can do that too. Note that if we are using dynamic inventory, which host is the 'first' may not be consistent, so you wouldn't want to do this unless your inventory was static and predictable. (If you are using *Ansible Tower*, it will use database order, so this isn't a problem even if you are using cloud based inventory scripts).

Anyway, here's the trick:

```
{{ hostvars[groups['webservers'][0]]['ansible_eth0']['ipv4']['address'] }}
```

Notice how we're pulling out the hostname of the first machine of the webservers group. If you are doing this in a template, you could use the Jinja2 '#set' directive to simplify this, or in a playbook, you could also use set_fact:

- set_fact: headnode={{ groups[['webservers'][0]] }}
- debug: msg={{ hostvars[headnode].ansible_eth0.ipv4.address }}

Notice how we interchanged the bracket syntax for dots – that can be done anywhere.

1.12.12 How do I copy files recursively onto a target host?

The "copy" module doesn't handle recursive copies of directories. A common solution to do this is to use a local action to call 'rsync' to recursively copy files to the managed servers.

Here is an example:

```
---
# ...
  tasks:
  - name: recursively copy files from management server to target
    local_action: command rsync -a /path/to/files $inventory_hostname:/path/to/target/
```

Note that you'll need passphrase-less SSH or ssh-agent set up to let rsync copy without prompting for a passphrase or password.

1.12.13 How do I access shell environment variables?

If you just need to access existing variables, use the 'env' lookup plugin. For example, to access the value of the HOME environment variable on management machine:

```
---
# ...
  vars:
     local_home: "{{ lookup('env','HOME') }}"
```

If you need to set environment variables, see the Advanced Playbooks section about environments.

Ansible 1.4 will also make remote environment variables available via facts in the 'ansible_env' variable:

```
{{ ansible_env.SOME_VARIABLE }}
```

1.12.14 How do I generate crypted passwords for the user module?

The mkpasswd utility that is available on most Linux systems is a great option:

```
mkpasswd --method=SHA-512
```

If this utility is not installed on your system (e.g. you are using OS X) then you can still easily generate these passwords using Python. First, ensure that the Passlib password hashing library is installed.

```
pip install passlib
```

Once the library is ready, SHA512 password values can then be generated as follows:

```
python -c "from passlib.hash import sha512_crypt; print sha512_crypt.encrypt('<password>')"
```

1.12.15 Can I get training on Ansible or find commercial support?

Yes! See *our Guru offering <http://www.ansible.com/ansible-guru>_* for online support, and support is also included with *Ansible Tower*. You can also read our service page and email info@ansible.com for further details.

1.12.16 Is there a web interface / REST API / etc?

Yes! Ansible, Inc makes a great product that makes Ansible even more powerful and easy to use. See *Ansible Tower*.

1.12.17 How do I submit a change to the documentation?

Great question! Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs README viewable on GitHub. Thanks!

1.12.18 I don't see my question here

Please see the section below for a link to IRC and the Google Group, where you can ask your question there.

See also:

Ansible Documentation The documentation index

Playbooks An introduction to playbooks

Best Practices Best practices advice

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.13 Glossary

The following is a list (and re-explanation) of term definitions used elsewhere in the Ansible documentation.

Consult the documentation home page for the full documentation and to see the terms in context, but this should be a good resource to check your knowledge of Ansible's components and understand how they fit together. It's something you might wish to read for review or when a term comes up on the mailing list.

1.13.1 Action

An action is a part of a task that specifies which of the modules to run and the arguments to pass to that module. Each task can have only one action, but it may also have other parameters.

1.13.2 Ad Hoc

Refers to running Ansible to perform some quick command, using /usr/bin/ansible, rather than the orchestration language, which is /usr/bin/ansible-playbook. An example of an ad-hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad-hoc can be accomplished by writing a playbook, and playbooks can also glue lots of other operations together.

1.13.3 Async

Refers to a task that is configured to run in the background rather than waiting for completion. If you have a long process that would run longer than the SSH timeout, it would make sense to launch that task in async mode. Async modes can poll for completion every so many seconds, or can be configured to "fire and forget" in which case Ansible will not even check on the task again, it will just kick it off and proceed to future steps. Async modes work with both /usr/bin/ansible and /usr/bin/ansible-playbook.

1.13.4 Callback Plugin

Refers to some user-written code that can intercept results from Ansible and do something with them. Some supplied examples in the GitHub project perform custom logging, send email, or even play sound effects.

1.13.5 Check Mode

Refers to running Ansible with the `--check` option, which does not make any changes on the remote systems, but only outputs the changes that might occur if the command ran without this flag. This is analogous to so-called "dry run" modes in other systems, though the user should be warned that this does not take into account unexpected command failures or cascade effects (which is true of similar modes in other systems). Use this to get an idea of what might happen, but it is not a substitute for a good staging environment.

1.13.6 Connection Type, Connection Plugin

By default, Ansible talks to remote machines through pluggable libraries. Ansible supports native OpenSSH ('ssh'), or a Python implementation called 'paramiko'. OpenSSH is preferred if you are using a recent version, and also enables some features like Kerberos and jump hosts. This is covered in the getting started section. There are also other connection types like 'fireball' mode, which must be bootstrapped over one of the SSH-based connection types but is very fast, and local mode, which acts on the local system. Users can also write their own connection plugins.

1.13.7 Conditionals

A conditional is an expression that evaluates to true or false that decides whether a given task will be executed on a given machine or not. Ansible's conditionals are powered by the 'when' statement, and are discussed in the playbook documentation.

1.13.8 Diff Mode

A `--diff` flag can be passed to Ansible to show how template files change when they are overwritten, or how they might change when used with `--check` mode. These diffs come out in unified diff format.

1.13.9 Facts

Facts are simply things that are discovered about remote nodes. While they can be used in playbooks and templates just like variables, facts are things that are inferred, rather than set. Facts are automatically discovered by Ansible when running plays by executing the internal 'setup' module on the remote nodes. You never have to call the setup module explicitly, it just runs, but it can be disabled to save time if it is not needed. For the convenience of users who are switching from other configuration management systems, the fact module will also pull in facts from the 'ohai' and 'facter' tools if they are installed, which are fact libraries from Chef and Puppet, respectively.

1.13.10 Filter Plugin

A filter plugin is something that most users will never need to understand. These allow for the creation of new Jinja2 filters, which are more or less only of use to people who know what Jinja2 filters are. If you need them, you can learn how to write them in the API docs section.

1.13.11 Fireball Mode

By default, Ansible uses SSH for connections – either paramiko or native OpenSSH, a common alternative. (Ansible tries to use 'ssh' by default if possible in Ansible 1.2.1 and later, but previously defaulted to paramiko). Some users may want to execute operations even faster though, and they can if they opt to run their tasks using an ephemeral 'fireball' message bus. What happens in this mode is that Ansible will start talking to a node over SSH, and then set up a secure, temporary message bus that authenticates only a single machine, and that will self destruct after a set period of time. This means the bus does not allow management of any kind after the time interval has expired.

1.13.12 Forks

Ansible talks to remote nodes in parallel and the level of parallelism can be set either by passing `--forks`, or editing the default in a configuration file. The default is a very conservative 5 forks, though if you have a lot of RAM, you can easily set this to a value like 50 for increased parallelism.

1.13.13 Gather Facts (Boolean)

Facts are mentioned above. Sometimes when running a multi-play playbook, it is desirable to have some plays that don't bother with fact computation if they aren't going to need to utilize any of these values. Setting *gather_facts: False* on a playbook allows this implicit fact gathering to be skipped.

1.13.14 Globbing

Globbing is a way to select lots of hosts based on wildcards, rather than the name of the host specifically, or the name of the group they are in. For instance, it is possible to select "www*" to match all hosts starting with "www". This concept is pulled directly from Func, one of Michael's earlier projects. In addition to basic globbing, various set operations are also possible, such as 'hosts in this group and not in another group', and so on.

1.13.15 Group

A group consists of several hosts assigned to a pool that can be conveniently targeted together, and also given variables that they share in common.

1.13.16 Group Vars

The "group_vars/" files are files that live in a directory alongside an inventory file, with an optional filename named after each group. This is a convenient place to put variables that will be provided to a given group, especially complex data structures, so that these variables do not have to be embedded in the inventory file or playbook.

1.13.17 Handlers

Handlers are just like regular tasks in an Ansible playbook (see Tasks), but are only run if the Task contains a "notify" directive and also indicates that it changed something. For example, if a config file is changed then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

1.13.18 Host

A host is simply a remote machine that Ansible manages. They can have individual variables assigned to them, and can also be organized in groups. All hosts have a name they can be reached at (which is either an IP address or a domain name) and optionally a port number if they are not to be accessed on the default SSH port.

1.13.19 Host Specifier

Each Play in Ansible maps a series of tasks (which define the role, purpose, or orders of a system) to a set of systems.

This "hosts:" directive in each play is often called the hosts specifier.

It may select one system, many systems, one or more groups, or even some hosts that are in one group and explicitly not in another.

1.13.20 Host Vars

Just like "Group Vars", a directory alongside the inventory file named "host_vars/" can contain a file named after each hostname in the inventory file, in YAML format. This provides a convenient place to assign variables to the host without having to embed them in the inventory file. The Host Vars file can also be used to define complex data structures that can't be represented in the inventory file.

1.13.21 Lazy Evaluation

In general, Ansible evaluates any variables in playbook content at the last possible second, which means that if you define a data structure that data structure itself can define variable values within it, and everything "just works" as you would expect. This also means variable strings can include other variables inside of those strings.

1.13.22 Lookup Plugin

A lookup plugin is a way to get data into Ansible from the outside world. These are how such things as "with_items", a basic looping plugin, are implemented, but there are also lookup plugins like "with_file" which loads data from a file, and even ones for querying environment variables, DNS text records, or key value stores. Lookup plugins can also be accessed in templates, e.g., `{{ lookup('file','/path/to/file') }}`.

1.13.23 Multi-Tier

The concept that IT systems are not managed one system at a time, but by interactions between multiple systems, and groups of systems, in well defined orders. For instance, a web server may need to be updated before a database server, and pieces on the web server may need to be updated after *THAT* database server, and various load balancers and monitoring servers may need to be contacted. Ansible models entire IT topologies and workflows rather than looking at configuration from a "one system at a time" perspective.

1.13.24 Idempotency

The concept that change commands should only be applied when they need to be applied, and that it is better to describe the desired state of a system than the process of how to get to that state. As an analogy, the path from North Carolina in the United States to California involves driving a very long way West, but if I were instead in Anchorage, Alaska, driving a long way west is no longer the right way to get to California. Ansible's Resources like you to say "put me in California" and then decide how to get there. If you were already in California, nothing needs to happen, and it will let you know it didn't need to change anything.

1.13.25 Includes

The idea that playbook files (which are nothing more than lists of plays) can include other lists of plays, and task lists can externalize lists of tasks in other files, and similarly with handlers. Includes can be parameterized, which means that the loaded file can pass variables. For instance, an included play for setting up a WordPress blog may take a parameter called "user" and that play could be included more than once to create a blog for both "alice" and "bob".

1.13.26 Inventory

A file (by default, Ansible uses a simple INI format) that describes Hosts and Groups in Ansible. Inventory can also be provided via an "Inventory Script" (sometimes called an "External Inventory Script").

1.13.27 Inventory Script

A very simple program (or a complicated one) that looks up hosts, group membership for hosts, and variable information from an external resource – whether that be a SQL database, a CMDB solution, or something like LDAP. This concept was adapted from Puppet (where it is called an "External Nodes Classifier") and works more or less exactly the same way.

1.13.28 Jinja2

Jinja2 is the preferred templating language of Ansible's template module. It is a very simple Python template language that is generally readable and easy to write.

1.13.29 JSON

Ansible uses JSON for return data from remote modules. This allows modules to be written in any language, not just Python.

1.13.30 Library

A collection of modules made available to /usr/bin/ansible or an Ansible playbook.

1.13.31 Limit Groups

By passing `--limit somegroup` to ansible or ansible-playbook, the commands can be limited to a subset of hosts. For instance, this can be used to run a playbook that normally targets an entire set of servers to one particular server.

1.13.32 Local Connection

By using "connection: local" in a playbook, or passing "-c local" to /usr/bin/ansible, this indicates that we are managing the local host and not a remote machine.

1.13.33 Local Action

A local_action directive in a playbook targeting remote machines means that the given step will actually occur on the local machine, but that the variable '{{ ansible_hostname }}' can be passed in to reference the remote hostname being referred to in that step. This can be used to trigger, for example, an rsync operation.

1.13.34 Loops

Generally, Ansible is not a programming language. It prefers to be more declarative, though various constructs like "with_items" allow a particular task to be repeated for multiple items in a list. Certain modules, like yum and apt, are actually optimized for this, and can install all packages given in those lists within a single transaction, dramatically speeding up total time to configuration.

1.13.35 Modules

Modules are the units of work that Ansible ships out to remote machines. Modules are kicked off by either /usr/bin/ansible or /usr/bin/ansible-playbook (where multiple tasks use lots of different modules in conjunction). Modules can be implemented in any language, including Perl, Bash, or Ruby – but can leverage some useful communal library code if written in Python. Modules just have to return JSON or simple key=value pairs. Once modules are executed on remote machines, they are removed, so no long running daemons are used. Ansible refers to the collection of available modules as a 'library'.

1.13.36 Notify

The act of a task registering a change event and informing a handler task that another action needs to be run at the end of the play. If a handler is notified by multiple tasks, it will still be run only once. Handlers are run in the order they are listed, not in the order that they are notified.

1.13.37 Orchestration

Many software automation systems use this word to mean different things. Ansible uses it as a conductor would conduct an orchestra. A datacenter or cloud architecture is full of many systems, playing many parts – web servers, database servers, maybe load balancers, monitoring systems, continuous integration systems, etc. In performing any process, it is necessary to touch systems in particular orders, often to simulate rolling updates or to deploy software correctly. Some system may perform some steps, then others, then previous systems already processed may need to perform more steps. Along the way, emails may need to be sent or web services contacted. Ansible orchestration is all about modeling that kind of process.

1.13.38 paramiko

By default, Ansible manages machines over SSH. The library that Ansible uses by default to do this is a Python-powered library called paramiko. The paramiko library is generally fast and easy to manage, though users desiring Kerberos or Jump Host support may wish to switch to a native SSH binary such as OpenSSH by specifying the connection type in their playbook, or using the "-c ssh" flag.

1.13.39 Playbooks

Playbooks are the language by which Ansible orchestrates, configures, administers, or deploys systems. They are called playbooks partially because it's a sports analogy, and it's supposed to be fun using them. They aren't workbooks :)

1.13.40 Plays

A playbook is a list of plays. A play is minimally a mapping between a set of hosts selected by a host specifier (usually chosen by groups, but sometimes by hostname globs) and the tasks which run on those hosts to define the role that those systems will perform. There can be one or many plays in a playbook.

1.13.41 Pull Mode

By default, Ansible runs in push mode, which allows it very fine-grained control over when it talks to each system. Pull mode is provided for when you would rather have nodes check in every N minutes on a particular schedule. It

uses a program called ansible-pull and can also be set up (or reconfigured) using a push-mode playbook. Most Ansible users use push mode, but pull mode is included for variety and the sake of having choices.

ansible-pull works by checking configuration orders out of git on a crontab and then managing the machine locally, using the local connection plugin.

1.13.42 Push Mode

Push mode is the default mode of Ansible. In fact, it's not really a mode at all – it's just how Ansible works when you aren't thinking about it. Push mode allows Ansible to be fine-grained and conduct nodes through complex orchestration processes without waiting for them to check in.

1.13.43 Register Variable

The result of running any task in Ansible can be stored in a variable for use in a template or a conditional statement. The keyword used to define the variable is called 'register', taking its name from the idea of registers in assembly programming (though Ansible will never feel like assembly programming). There are an infinite number of variable names you can use for registration.

1.13.44 Resource Model

Ansible modules work in terms of resources. For instance, the file module will select a particular file and ensure that the attributes of that resource match a particular model. As an example, we might wish to change the owner of /etc/motd to 'root' if it is not already set to root, or set its mode to '0644' if it is not already set to '0644'. The resource models are 'idempotent' meaning change commands are not run unless needed, and Ansible will bring the system back to a desired state regardless of the actual state – rather than you having to tell it how to get to the state.

1.13.45 Roles

Roles are units of organization in Ansible. Assigning a role to a group of hosts (or a set of groups, or host patterns, etc.) implies that they should implement a specific behavior. A role may include applying certain variable values, certain tasks, and certain handlers – or just one or more of these things. Because of the file structure associated with a role, roles become redistributable units that allow you to share behavior among playbooks – or even with other users.

1.13.46 Rolling Update

The act of addressing a number of nodes in a group N at a time to avoid updating them all at once and bringing the system offline. For instance, in a web topology of 500 nodes handling very large volume, it may be reasonable to update 10 or 20 machines at a time, moving on to the next 10 or 20 when done. The "serial:" keyword in an Ansible playbook controls the size of the rolling update pool. The default is to address the batch size all at once, so this is something that you must opt-in to. OS configuration (such as making sure config files are correct) does not typically have to use the rolling update model, but can do so if desired.

1.13.47 Runner

A core software component of Ansible that is the power behind /usr/bin/ansible directly – and corresponds to the invocation of each task in a playbook. The Runner is something Ansible developers may talk about, but it's not really user land vocabulary.

1.13.48 Serial

See "Rolling Update".

1.13.49 Sudo

Ansible does not require root logins, and since it's daemonless, definitely does not require root level daemons (which can be a security concern in sensitive environments). Ansible can log in and perform many operations wrapped in a sudo command, and can work with both password-less and password-based sudo. Some operations that don't normally work with sudo (like scp file transfer) can be achieved with Ansible's copy, template, and fetch modules while running in sudo mode.

1.13.50 SSH (Native)

Native OpenSSH as an Ansible transport is specified with "-c ssh" (or a config file, or a directive in the playbook) and can be useful if wanting to login via Kerberized SSH or using SSH jump hosts, etc. In 1.2.1, 'ssh' will be used by default if the OpenSSH binary on the control machine is sufficiently new. Previously, Ansible selected 'paramiko' as a default. Using a client that supports ControlMaster and ControlPersist is recommended for maximum performance – if you don't have that and don't need Kerberos, jump hosts, or other features, paramiko is a good choice. Ansible will warn you if it doesn't detect ControlMaster/ControlPersist capability.

1.13.51 Tags

Ansible allows tagging resources in a playbook with arbitrary keywords, and then running only the parts of the playbook that correspond to those keywords. For instance, it is possible to have an entire OS configuration, and have certain steps labeled "ntp", and then run just the "ntp" steps to reconfigure the time server information on a remote host.

1.13.52 Tasks

Playbooks exist to run tasks. Tasks combine an action (a module and its arguments) with a name and optionally some other keywords (like looping directives). Handlers are also tasks, but they are a special kind of task that do not run unless they are notified by name when a task reports an underlying change on a remote system.

1.13.53 Templates

Ansible can easily transfer files to remote systems, but often it is desirable to substitute variables in other files. Variables may come from the inventory file, Host Vars, Group Vars, or Facts. Templates use the Jinja2 template engine and can also include logical constructs like loops and if statements.

1.13.54 Transport

Ansible uses "Connection Plugins" to define types of available transports. These are simply how Ansible will reach out to managed systems. Transports included are paramiko, SSH (using OpenSSH), fireball (an SSH-bootstrapped accelerated connection plugin), and local.

1.13.55 When

An optional conditional statement attached to a task that is used to determine if the task should run or not. If the expression following the "when:" keyword evaluates to false, the task will be ignored.

1.13.56 Van Halen

For no particular reason, other than the fact that Michael really likes them, all Ansible releases are codenamed after Van Halen songs. There is no preference given to David Lee Roth vs. Sammy Lee Hagar-era songs, and instrumentals are also allowed. It is unlikely that there will ever be a Jump release, but a Van Halen III codename release is possible. You never know.

1.13.57 Vars (Variables)

As opposed to Facts, variables are names of values (they can be simple scalar values – integers, booleans, strings) or complex ones (dictionaries/hashes, lists) that can be used in templates and playbooks. They are declared things, not things that are inferred from the remote system's current state or nature (which is what Facts are).

1.13.58 YAML

Ansible does not want to force people to write programming language code to automate infrastructure, so Ansible uses YAML to define playbook configuration languages and also variable files. YAML is nice because it has a minimum of syntax and is very clean and easy for people to skim. It is a good data format for configuration files and humans, but also machine readable. Ansible's usage of YAML stemmed from Michael's first use of it inside of Cobbler around 2006. YAML is fairly popular in the dynamic language community and the format has libraries available for serialization in many different languages (Python, Perl, Ruby, etc.).

See also:

Frequently Asked Questions Frequently asked questions

Playbooks An introduction to playbooks

Best Practices Best practices advice

User Mailing List Have a question? Stop by the google group!

irc.freenode.net #ansible IRC chat channel

1.14 YAML Syntax

This page provides a basic overview of correct YAML syntax, which is how Ansible playbooks (our configuration management language) are expressed.

We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.

You may also wish to read *Playbooks* at the same time to see how this is used in practice.

1.14.1 YAML Basics

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a "hash" or a "dictionary". So, we need to know how to write lists and dictionaries in YAML.

There's another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) should begin with `---`. This is part of the YAML format and indicates the start of a document.

All members of a list are lines beginning at the same indentation level starting with a `-` (dash) character:

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
```

A dictionary is represented in a simple `key:` and `value` form:

```
---
# An employee record
name: Example Developer
job: Developer
skill: Elite
```

Dictionaries can also be represented in an abbreviated form if you really want to:

```
---
# An employee record
{name: Example Developer, job: Developer, skill: Elite}
```

Ansible doesn't really use these too much, but you can also specify a boolean value (true/false) in several forms:

```
---
create_key: yes
needs_agent: no
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

Let's combine what we learned so far in an arbitrary YAML example. This really has nothing to do with Ansible, but will give you a feel for the format:

```
---
# An employee record
name: Example Developer
job: Developer
skill: Elite
employed: True
foods:
    - Apple
    - Orange
    - Strawberry
    - Mango
languages:
    ruby: Elite
    python: Elite
    dotnet: Lame
```

That's all you really need to know about YAML to start writing *Ansible* playbooks.

1.14.2 Gotchas

While YAML is generally friendly, the following is going to result in a YAML syntax error:

```
foo: somebody said I should put a colon here: so I did
```

You will want to quote any hash values using colons, like so:

```
foo: "somebody said I should put a colon here: so I did"
```

And then the colon will be preserved.

Further, Ansible uses "{{ var }}" for variables. If a value after a colon starts with a "{", YAML will think it is a dictionary, so you must quote it, like so:

```
foo: "{{ variable }}"
```

See also:

Playbooks Learn what playbooks can do and how to write/run them.

YAMLLint YAML Lint (online) helps you debug YAML syntax if you are having problems

Github examples directory Complete playbook files from the github project source

Mailing List Questions? Help? Ideas? Stop by the list on Google Groups

irc.freenode.net #ansible IRC chat channel

1.15 Ansible Guru

While many users should be able to get on fine with the documentation, mailing list, and IRC, sometimes you want a bit more.

Ansible Guru is an offering from Ansible, Inc that helps users who would like more dedicated help with Ansible, including building playbooks, best practices, architecture suggestions, and more – all from our awesome support and services team. It also includes some useful discounts and also some free T-shirts, though you shoudn't get it just for the free shirts! It's a great way to train up to becoming an Ansible expert.

For those interested, click through the link above. You can sign up in minutes!

For users looking for more hands-on help, we also have some more information on our Services page, and support is also included with *Ansible Tower*.