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.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\reference_appendices\(ansible-devel)(docs)(docsite)(rst)(reference_appendices)glossary.rst`, line 10)

Unknown directive type "glossary".

```
.. glossary::

    Action
        An action is a part of a task that specifies which of the modules to
        run and which arguments to pass to that module.  Each task can have
        only one action, but it may also have other parameters.

    Ad Hoc
        Refers to running Ansible to perform some quick command, using
        :command:`/usr/bin/ansible`, rather than the :term:`orchestration`
        language, which is :command:`/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 :term:`playbook <playbooks>` and playbooks can also glue
        lots of other operations together.

    Ansible (the package)
        A software package (Python, deb, rpm, and so on) that contains ansible-core and a select group of collections. Playbooks t

    ansible-base
        Used only for 2.10. The installable package (RPM/Python/Deb package) generated from the `ansible/ansible repository <https

    ansible-core
        Name used starting with 2.11. The installable package (RPM/Python/Deb package) generated from the `ansible/ansible reposit

    Ansible Galaxy
        An `online resource <galaxy.ansible.com>`_ for finding and sharing Ansible community content. Also, the command-line utili

    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
        :command:`/usr/bin/ansible` and :command:`/usr/bin/ansible-playbook`.

    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.

    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 do
        not substitute it for a good staging environment.

    Collection
        A packaging format for bundling and distributing Ansible content, including plugins, roles, modules, and more. Collections

    Collection name
        The second part of a Fully Qualified Collection Name. The collection name divides the collection namespace and usually ref

    community.general (collection)
        A special collection managed by the Ansible Community Team containing all the modules and plugins which shipped in Ansible

    community.network (collection)
        Similar to ``community.general``, focusing on network content. `community.network <https://galaxy.ansible.com/community/ne

    Connection Plugin
        By default, Ansible talks to remote machines through pluggable
        libraries.  Ansible uses native OpenSSH (:term:`SSH (Native)`) or
        a Python implementation called :term:`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 :ref:`getting
        started section <remote_connection_information>`.  There are also
        other connection types like ``accelerate`` 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.

    Conditionals
        A conditional is an expression that evaluates to true or false that
        decides whether a given task is executed on a given machine or not.
        Ansible's conditionals are powered by the 'when' statement, which are
        discussed in the :ref:`working_with_playbooks`.

    Declarative
        An approach to achieving a task that uses a description of the
        final state rather than a description of the sequence of steps
        necessary to achieve that state. For a real world example, a
        declarative specification of a task would be: "put me in California".
        Depending on your current location, the sequence of steps to get you to
        California may vary, and if you are already in California, nothing
        at all needs to be done. Ansible's Resources are declarative; it
        figures out the steps needed to achieve the final state. It also lets
        you know whether or not any steps needed to be taken to get to the
        final state.

    Diff Mode
        A ``--diff`` flag can be passed to Ansible to show what changed on
        modules that support it. You can combine it with ``--check`` to get a
        good 'dry run'.  File diffs are normally in unified diff format.

    Executor
        A core software component of Ansible that is the power behind
        :command:`/usr/bin/ansible` directly -- and corresponds to the
        invocation of each task in a :term:`playbook <playbooks>`.  The
        Executor is something Ansible developers may talk about, but it's not
        really user land vocabulary.

    Facts
        Facts are simply things that are discovered about remote nodes.  While
        they can be used in :term:`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 :ref:`setup module <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 or you can tell
        ansible to collect only a subset of the full facts via the
        ``gather_subset:`` option. For the convenience of users who are
        switching from other configuration management systems, the fact module
        will also pull in facts from the :program:`ohai` and :program:`facter`
        tools if they are installed.  These are fact libraries from Chef and
        Puppet, respectively. (These may also be disabled via
        ``gather_subset:``)

    Filter Plugin
        A filter plugin is something that most users will never need to
        understand.  These allow for the creation of new :term:`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 :ref:`API docs section <developing_filter_plugins>`.

    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 five (5)
        forks, though if you have a lot of RAM, you can easily set this to
        a value like 50 for increased parallelism.

    Fully Qualified Collection Name (FQCN)
        The full definition of a module, plugin, or role hosted within a collection, in the form <namespace.collection.content_nam

    Gather Facts (Boolean)
        :term:`Facts` are mentioned above.  Sometimes when running a multi-play
        :term:`playbook <playbooks>`, 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.

    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 ``ww*`` to match all
        hosts starting with ``www``.   This concept is pulled directly from
        :program:`Func`, one of Michael DeHaan's (an Ansible Founder) 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.

    Group
        A group consists of several hosts assigned to a pool that can be
        conveniently targeted together, as well as given variables that they
        share in common.

    Group Vars
        The :file:`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 are
        provided to a given group, especially complex data structures, so that
        these variables do not have to be embedded in the :term:`inventory`
        file or :term:`playbook <playbooks>`.

    Handlers
        Handlers are just like regular tasks in an Ansible
        :term:`playbook <playbooks>` (see :term:`Tasks`) but are only run if
        the Task contains a ``notify`` keyword 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.

    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.

    Host Specifier
        Each :term:`Play <plays>` in Ansible maps a series of :term:`tasks` (which define the role,
        purpose, or orders of a system) to a set of systems.

        This ``hosts:`` keyword 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.

    Host Vars
        Just like :term:`Group Vars`, a directory alongside the inventory file named
        :file:`host_vars/` can contain a file named after each hostname in the
        inventory file, in :term:`YAML` format.  This provides a convenient place to
        assign variables to the host without having to embed them in the
        :term:`inventory` file.  The Host Vars file can also be used to define complex
        data structures that can't be represented in the inventory file.

    Idempotency
        An operation is idempotent if the result of performing it once is
        exactly the same as the result of performing it repeatedly without
        any intervening actions.

    Includes
        The idea that :term:`playbook <playbooks>` files (which are nothing
        more than lists of :term:`plays`) can include other lists of plays,
        and task lists can externalize lists of :term:`tasks` in other files,
        and similarly with :term:`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``.

    Inventory
        A file (by default, Ansible uses a simple INI format) that describes
        :term:`Hosts <Host>` and :term:`Groups <Group>` in Ansible.  Inventory
        can also be provided via an :term:`Inventory Script` (sometimes called
        an "External Inventory Script").

    Inventory Script
        A very simple program (or a complicated one) that looks up
        :term:`hosts <Host>`, :term:`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.

    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.

    JSON
        Ansible uses JSON for return data from remote modules.  This allows
        modules to be written in any language, not just Python.
```

```
Keyword
    The main expressions that make up Ansible, which apply to playbook objects
    (Play, Block, Role and Task). For example 'vars:' is a keyword that lets
    you define variables in the scope of the playbook object it is applied to.

Lazy Evaluation
    In general, Ansible evaluates any variables in
    :term:`playbook <playbooks>` 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.

Library
    A collection of modules made available to :command:`/usr/bin/ansible`
    or an Ansible :term:`playbook <playbooks>`.

Limit Groups
    By passing ``--limit somegroup`` to :command:`ansible` or
    :command:`ansible-playbook`, the commands can be limited to a subset
    of :term:`hosts <Host>`.  For instance, this can be used to run
    a :term:`playbook <playbooks>` that normally targets an entire set of
    servers to one particular server.

Local Action
    This keyword is an alias for ``delegate_to: localhost``.
    Used when you want to redirect an action from the remote to
    execute on the controller itself.

Local Connection
    By using ``connection: local`` in a :term:`playbook <playbooks>`, or
    passing ``-c local`` to :command:`/usr/bin/ansible`, this indicates
    that we are executing a local fork instead of executing on the remote machine.
    You probably want ``local_action`` or ``delegate_to: localhost`` instead
    as this ONLY changes the connection and no other context for execution.

Lookup Plugin
    A lookup plugin is a way to get data into Ansible from the outside world.
    Lookup plugins are an extension of Jinja2 and can be accessed in templates, for example,
    ``{{ lookup('file','/path/to/file') }}``.
    These are how such things as ``with_items``, are implemented.
    There are also lookup plugins like ``file`` which loads data from
    a file and ones for querying environment variables, DNS text records,
    or key value stores.

Loops
    Generally, Ansible is not a programming language. It prefers to be
    more declarative, though various constructs like ``loop`` allow
    a particular task to be repeated for multiple items in a list.
    Certain modules, like :ref:`yum <yum_module>` and :ref:`apt <apt_module>`, actually take
    lists directly, and can install all packages given in those lists
    within a single transaction, dramatically speeding up total time to
    configuration, so they can be used without loops.

Modules
    Modules are the units of work that Ansible ships out to remote
    machines.   Modules are kicked off by either
    :command:`/usr/bin/ansible` or :command:`/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 take advantage of some useful communal library code if written
    in Python.  Modules just have to return :term:`JSON`.  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 :term:`library`.

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.

Namespace
    The first part of a fully qualified collection name, the namespace usually reflects a functional content category. Example

Notify
    The act of a :term:`task <tasks>` registering a change event and
    informing a :term:`handler <handlers>` task that another
    :term:`action` needs to be run at the end of the :term:`play <plays>`.  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.

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, and so on.  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.

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 who want to use Kerberos or Jump Hosts may wish to switch
    to a native SSH binary such as OpenSSH by specifying the connection
    type in their :term:`playbooks`, or using the ``-c ssh`` flag.

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 :)

Plays
    A :term:`playbook <playbooks>` is a list of plays.  A play is
    minimally a mapping between a set of :term:`hosts <Host>` selected by a host
    specifier (usually chosen by :term:`groups <Group>` but sometimes by
    hostname :term:`globs <Globbing>`) and the :term:`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.

Pull Mode
    By default, Ansible runs in :term:`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
    :command:`ansible-pull` and can also be set up (or reconfigured) using
    a push-mode :term:`playbook <playbooks>`.  Most Ansible users use push
```

```
        mode, but pull mode is included for variety and the sake of having
        choices.

        :command:`ansible-pull` works by checking configuration orders out of
        git on a crontab and then managing the machine locally, using the
        :term:`local connection` plugin.

    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.

    Register Variable
        The result of running any :term:`task <tasks>` 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.

    Resource Model
        Ansible modules work in terms of resources.   For instance, the
        :ref:`file module <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 :file:`/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 :term:`idempotent <idempotency>` 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.

    Roles
        Roles are units of organization in Ansible.  Assigning a role to
        a group of :term:`hosts <Host>` (or a set of :term:`groups <group>`,
        or :term:`host patterns <Globbing>`, and so on) implies that they should
        implement a specific behavior.  A role may include applying certain
        variable values, certain :term:`tasks`, and certain :term:`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 :term:`playbooks` -- or even with other users.

    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
        :term:`playbooks` control 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.

    Serial
        .. seealso::

            :term:`Rolling Update`

    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 :ref:`copy <copy_module>`, :ref:`template <template_module>`, and
        :ref:`fetch <fetch_module>` modules while running in sudo mode.

    SSH (Native)
        Native OpenSSH as an Ansible transport is specified with ``-c ssh``
        (or a config file, or a keyword in the :term:`playbook <playbooks>`)
        and can be useful if wanting to login via Kerberized SSH or using SSH
        jump hosts, and so on.  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.

    Tags
        Ansible allows tagging resources in a :term:`playbook <playbooks>`
        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.

    Task
        :term:`Playbooks` exist to run tasks.  Tasks combine an :term:`action`
        (a module and its arguments) with a name and optionally some other
        keywords (like :term:`looping keywords <loops>`).   :term:`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.

    Tasks
        A list of :term:`Task`.

    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 :term:`inventory` file, :term:`Host Vars`, :term:`Group
        Vars`, or :term:`Facts`. Templates use the :term:`Jinja2` template
        engine and can also include logical constructs like loops and if
        statements.

    Transport
        Ansible uses :term:``Connection Plugins`` to define types of available
        transports.  These are simply how Ansible will reach out to managed
        systems.  Transports included are :term:`paramiko`,
        :term:`ssh <SSH (Native)>` (using OpenSSH), and
        :term:`local <Local Connection>`.

    When
        An optional conditional statement attached to a :term:`task <tasks>` 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.

    Vars (Variables)
        As opposed to :term:`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
        :term:`playbooks`.  They are declared things, not things that are
        inferred from the remote system's current state or nature (which is
        what Facts are).
    YAML
        Ansible does not want to force people to write programming language
        code to automate infrastructure, so Ansible uses YAML to define
        :term:`playbook <playbooks>` 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 DeHaan'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 languages (Python, Perl, Ruby, and so on).
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\reference_appendices\(ansible-devel)(docs)(docsite)(rst)(reference_appendices)glossary.rst`, line 525)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`ansible_faq`
       Frequently asked questions
   :ref:`working_with_playbooks`
       An introduction to playbooks
   :ref:`playbooks_best_practices`
       Tips and tricks for playbooks
   `User Mailing List <https://groups.google.com/group/ansible-devel>`_
       Have a question?  Stop by the google group!
   :ref:`communication_irc`
       How to join Ansible chat channels
```