The lifecycle of an Ansible module or plugin

Modules and plugins in the main Ansible repository have a defined life cycle, from the first introduction to final removal. The module and plugin lifecycle is tied to the *Ansible release cycle <release_cycle>*. A module or plugin may move through these four stages:

1. When a module or plugin is first accepted into Ansible, we consider it in tech preview and will mark it as such in the documentation.
2. If a module or plugin matures, the 'preview' mark in the documentation is removed. Backward compatibility for these modules and plugins is maintained but not guaranteed, which means their parameters should be maintained with stable meanings.
3. If a module's or plugin's target API changes radically, or if someone creates a better implementation of its functionality, we may mark it deprecated. Modules and plugins that are deprecated are still available but they are reaching the end of their life cycle. We retain deprecated modules and plugins for 4 release cycles with deprecation warnings to help users update playbooks and roles that use them.
4. When a module or plugin has been deprecated for four release cycles, it is removed and replaced with a tombstone entry in the routing configuration. Modules and plugins that are removed are no longer shipped with Ansible. The tombstone entry helps users find alternative modules and plugins.

For modules and plugins in collections, the lifecycle is similar. Since ansible-base 2.10, it is no longer possible to mark modules as 'preview' or 'stable'.

Deprecating modules and plugins in the Ansible main repository

To deprecate a module in ansible-core, you must:

1. Rename the file so it starts with an _, for example, rename `old_cloud.py` to `_old_cloud.py`. This keeps the module available and marks it as deprecated on the module index pages.
2. Mention the deprecation in the relevant changelog (by creating a changelog fragment with a section `deprecated_features`).
3. Reference the deprecation in the relevant `porting_guide_core_x.y.rst`.
4. Add `deprecated:` to the documentation with the following sub-values:

removed_in:	A `string`, such as `"2.10"`; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4. Mutually exclusive with :removed_by_date:.
remove_by_date:	(Added in ansible-base 2.10). An ISO 8601 formatted date when the module will be removed. Usually 2 years from the date the module is deprecated. Mutually exclusive with :removed_in:.
why:	Optional string that used to detail why this has been removed.
alternative:	Inform users they should do instead, for example, `Use M(whatmoduletouseinstead) instead.`.

- For an example of documenting deprecation, see this PR that deprecates multiple modules. Some of the elements in the PR might now be out of date.

Deprecating modules and plugins in a collection

To deprecate a module in a collection, you must:

1. Add a `deprecation` entry to `plugin_routing` in `meta/runtime.yml`. For example, to deprecate the module `old_cloud`, add:

```
plugin_routing:
    modules:
        old_cloud:
            deprecation:
                removal_version: 2.0.0
                warning_text: Use foo.bar.new_cloud instead.
```

   For other plugin types, you have to replace `modules:` with `<plugin_type>:`, for example `lookup:` for lookup plugins.

   Instead of `removal_version`, you can also use `removal_date` with an ISO 8601 formatted date after which the module will be removed in a new major version of the collection.

2. Mention the deprecation in the relevant changelog. If the collection uses `antsibull-changelog`, create a changelog fragment with a section `deprecated_features`.
3. Add `deprecated:` to the documentation of the module or plugin with the following sub-values:

removed_in:	A `string`, such as `"2.10"`; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4. Mutually exclusive with :removed_by_date:.

remove_by_date:	(Added in ansible-base 2.10). An ISO 8601 formatted date when the module will be removed. Usually 2 years from the date the module is deprecated. Mutually exclusive with :removed_in:.
why:	Optional string that used to detail why this has been removed.
alternative:	Inform users they should do instead, for example, `Use M(whatmoduletouseinstead) instead.`.

Changing a module or plugin name in the Ansible main repository

You can also rename a module and keep a deprecated alias to the old name by using a symlink that starts with _. This example allows the `stat` module to be called with `fileinfo`, making the following examples equivalent:

```
EXAMPLES = '''
ln -s stat.py _fileinfo.py
ansible -m stat -a "path=/tmp" localhost
ansible -m fileinfo -a "path=/tmp" localhost
'''
```

Renaming a module or plugin in a collection, or redirecting a module or plugin to another collection

To rename a module or plugin in a collection, or to redirect a module or plugin to another collection, you need to add a `redirect` entry to `plugin_routing` in `meta/runtime.yml`. For example, to redirect the module `old_cloud` to `foo.bar.new_cloud`, add:

```
plugin_routing:
    modules:
        old_cloud:
            redirect: foo.bar.new_cloud
```

If you want to deprecate the old name, add a `deprecation:` entry (see above):

```
plugin_routing:
    modules:
        old_cloud:
            redirect: foo.bar.new_cloud
            deprecation:
                removal_version: 2.0.0
                warning_text: Use foo.bar.new_cloud instead.
```

You need to use the Fully Qualified Collection Name (FQCN) of the new module/plugin name, even if it is located in the same collection as the redirect. By using a FQCN from another collection, you redirect the module/plugin to that collection.

If you need to support Ansible 2.9, please note that Ansible 2.9 does not know about `meta/runtime.yml`. With Ansible 2.9 you can still rename plugins and modules inside one collection by using symbolic links. Note that ansible-base 2.10, ansible-core 2.11, and newer will prefer `meta/runtime.yml` entries over symbolic links.

Tombstoning a module or plugin in a collection

To remove a deprecated module or plugin from a collection, you need to tombstone it:

1. Remove the module or plugin file with related files like tests, documentation references, and documentation.
2. Add a tombstone entry in `meta/runtime.yml`. For example, to tombstone the module `old_cloud`, add:

   ```
   plugin_routing:
       modules:
           old_cloud:
               tombstone:
                   removal_version: 2.0.0
                   warning_text: Use foo.bar.new_cloud instead.
   ```

   Instead of `removal_version`, you can also use `removal_date` with an ISO 8601 formatted date. The date should be the date of the next major release.