Re-using Ansible artifacts

You can write a simple playbook in one very large file, and most users learn the one-file approach first. However, breaking your automation work up into smaller files is an excellent way to organize complex sets of tasks and reuse them. Smaller, more distributed artifacts let you re-use the same variables, tasks, and plays in multiple playbooks to address different use cases. You can use distributed artifacts across multiple parent playbooks or even multiple times within one playbook. For example, you might want to update your customer database as part of several different playbooks. If you put all the tasks related to updating your database in a tasks file or a role, you can re-use them in many playbooks while only maintaining them in one place.

Creating re-usable files and roles

Ansible offers four distributed, re-usable artifacts: variables files, task files, playbooks, and roles.

- A variables file contains only variables.
- A task file contains only tasks.
- A playbook contains at least one play, and may contain variables, tasks, and other content. You can re-use tightly focused playbooks, but you can only re-use them statically, not dynamically.
- A role contains a set of related tasks, variables, defaults, handlers, and even modules or other plugins in a defined file-tree. Unlike variables files, task files, or playbooks, roles can be easily uploaded and shared via Ansible Galaxy. See :ref:`playbooks_reuse_roles` for details about creating and using roles.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 20); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 22)

Unknown directive type "versionadded".

```
.. versionadded:: 2.4
```

Re-using playbooks

You can incorporate multiple playbooks into a main playbook. However, you can only use imports to re-use playbooks. For example:

```
- import_playbook: webservers.yml
- import_playbook: databases.yml
```

Importing incorporates playbooks in other playbooks statically. Ansible runs the plays and tasks in each imported playbook in the order they are listed, just as if they had been defined directly in the main playbook.

You can select which playbook you want to import at runtime by defining your imported playbook filename with a variable, then passing the variable with either `--extra-vars` or the `vars` keyword. For example:

```
- import_playbook: "/path/to/{{ import_from_extra_var }}"
- import_playbook: "{{ import_from_vars }}"
  vars:
    import_from_vars: /path/to/one_playbook.yml
```

If you run this playbook with `ansible-playbook my_playbook -e import_from_extra_var=other_playbook.yml`, Ansible imports both one_playbook.yml and other_playbook.yml.

When to turn a playbook into a role

For some use cases, simple playbooks work well. However, starting at a certain level of complexity, roles work better than playbooks. A role lets you store your defaults, handlers, variables, and tasks in separate directories, instead of in a single long document. Roles are easy to share on Ansible Galaxy. For complex use cases, most users find roles easier to read, understand, and maintain than all-in-one playbooks.

Re-using files and roles

Ansible offers two ways to re-use files and roles in a playbook: dynamic and static.

- For dynamic re-use, add an `include_*` task in the tasks section of a play:
  - :ref:`include_role <include_role_module>`

    System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst`, **line 59**); *backlink*

    Unknown interpreted text role "ref".

  - :ref:`include_tasks <include_tasks_module>`

    System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst`, **line 60**); *backlink*

    Unknown interpreted text role "ref".

  - :ref:`include_vars <include_vars_module>`

    System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst`, **line 61**); *backlink*

    Unknown interpreted text role "ref".

- For static re-use, add an `import_*` task in the tasks section of a play:
  - :ref:`import_role <import_role_module>`

    System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst`, **line 65**); *backlink*

    Unknown interpreted text role "ref".

  - :ref:`import_tasks <import_tasks_module>`

    System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst`, **line 66**); *backlink*

    Unknown interpreted text role "ref".

Task include and import statements can be used at arbitrary depth.

You can still use the bare :ref:`roles <roles_keyword>` keyword at the play level to incorporate a role in a playbook statically. However, the bare :ref:`include <include_module>` keyword, once used for both task files and playbook-level includes, is now deprecated.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst`, **line 70**); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 70); *backlink*

Unknown interpreted text role "ref".

Includes: dynamic re-use

Including roles, tasks, or variables adds them to a playbook dynamically. Ansible processes included files and roles as they come up in a playbook, so included tasks can be affected by the results of earlier tasks within the top-level playbook. Included roles and tasks are similar to handlers - they may or may not run, depending on the results of other tasks in the top-level playbook.

The primary advantage of using `include_*` statements is looping. When a loop is used with an include, the included tasks or role will be executed once for each item in the loop.

The filenames for included roles, tasks, and vars are templated before inclusion.

You can pass variables into includes. See :ref:`ansible_variable_precedence` for more details on variable inheritance and precedence.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 81); *backlink*

Unknown interpreted text role "ref".

Imports: static re-use

Importing roles, tasks, or playbooks adds them to a playbook statically. Ansible pre-processes imported files and roles before it runs any tasks in a playbook, so imported content is never affected by other tasks within the top-level playbook.

The filenames for imported roles and tasks support templating, but the variables must be available when Ansible is pre-processing the imports. This can be done with the `vars` keyword or by using `--extra-vars`.

You can pass variables to imports. You must pass variables if you want to run an imported file more than once in a playbook. For example:

```
tasks:
- import_tasks: wordpress.yml
  vars:
    wp_user: timmy

- import_tasks: wordpress.yml
  vars:
    wp_user: alice

- import_tasks: wordpress.yml
  vars:
    wp_user: bob
```

See :ref:`ansible_variable_precedence` for more details on variable inheritance and precedence.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 107); *backlink*

Unknown interpreted text role "ref".

Comparing includes and imports: dynamic and static re-use

Each approach to re-using distributed Ansible artifacts has advantages and limitations. You may choose dynamic re-use for some playbooks and static re-use for others. Although you can use both dynamic and static re-use in a single playbook, it is best to select one approach per playbook. Mixing static and dynamic re-use can introduce difficult-to-diagnose bugs into your playbooks. This table summarizes the main differences so you can choose the best approach for each playbook you create.

	Include_*	**Import_***
Type of re-use	Dynamic	Static
When processed	At runtime, when encountered	Pre-processed during playbook parsing
Task or play	All includes are tasks	`import_playbook` cannot be a task
Task options	Apply only to include task itself	Apply to all child tasks in import

	Include_*	Import_*
Calling from loops	Executed once for each loop item	Cannot be used in a loop
Using `--list-tags`	Tags within includes not listed	All tags appear with `--list-tags`
Using `--list-tasks`	Tasks within includes not listed	All tasks appear with `--list-tasks`
Notifying handlers	Cannot trigger handlers within includes	Can trigger individual imported handlers
Using `--start-at-task`	Cannot start at tasks within includes	Can start at imported tasks
Using inventory variables	Can `include_*: {{ inventory_var }}`	Cannot `import_*: {{ inventory_var }}`
With playbooks	No `include_playbook`	Can import full playbooks
With variables files	Can include variables files	Use `vars_files:` to import variables

Note

- There are also big differences in resource consumption and performance, imports are quite lean and fast, while includes require a lot of management and accounting.

Re-using tasks as handlers

You can also use includes and imports in the :ref:`handlers` section of a playbook. 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 `restarts.yml` file that looks like:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 156); *backlink*

Unknown interpreted text role "ref".

```
# restarts.yml
- name: Restart apache
  ansible.builtin.service:
    name: apache
    state: restarted

- name: Restart mysql
  ansible.builtin.service:
    name: mysql
    state: restarted
```

You can trigger handlers from either an import or an include, but the procedure is different for each method of re-use. If you include the file, you must notify the include itself, which triggers all the tasks in `restarts.yml`. If you import the file, you must notify the individual task(s) within `restarts.yml`. You can mix direct tasks and handlers with included or imported tasks and handlers.

Triggering included (dynamic) handlers

Includes are executed at run-time, so the name of the include exists during play execution, but the included tasks do not exist until the include itself is triggered. To use the `Restart apache` task with dynamic re-use, refer to the name of the include itself. This approach triggers all tasks in the included file as handlers. For example, with the task file shown above:

```
- name: Trigger an included (dynamic) handler
  hosts: localhost
  handlers:
    - name: Restart services
      include_tasks: restarts.yml
  tasks:
    - command: "true"
      notify: Restart services
```

Triggering imported (static) handlers

Imports are processed before the play begins, so the name of the import no longer exists during play execution, but the names of the individual imported tasks do exist. To use the `Restart apache` task with static re-use, refer to the name of each task or tasks within the imported file. For example, with the task file shown above:

```
- name: Trigger an imported (static) handler
  hosts: localhost
  handlers:
    - name: Restart services
      import_tasks: restarts.yml
  tasks:
    - command: "true"
      notify: Restart apache
```

```
  - command: "true"
    notify: Restart mysql
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)playbooks_reuse.rst, line 207)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`utilities_modules`
       Documentation of the ``include*`` and ``import*`` modules discussed here.
   :ref:`working_with_playbooks`
       Review the basic Playbook language features
   :ref:`playbooks_variables`
       All about variables in playbooks
   :ref:`playbooks_conditionals`
       Conditionals in playbooks
   :ref:`playbooks_loops`
       Loops in playbooks
   :ref:`playbooks_best_practices`
       Tips and tricks for playbooks
   :ref:`ansible_galaxy`
       How to share roles on galaxy, role management
   `GitHub Ansible examples <https://github.com/ansible/ansible-examples>`_
       Complete playbook files from the GitHub project source
   `Mailing List <https://groups.google.com/group/ansible-project>`_
       Questions? Help? Ideas?  Stop by the list on Google Groups
```