Include

Table of contents

Introduced in Docker Compose version 2.20.0

A Compose application can declare dependency on another Compose application. This is useful if:

- You want to reuse other Compose files.
- You need to factor out parts of your application model into separate Compose files so they can be managed separately or shared with others.
- Teams need to keep a Compose file reasonably complicated for the limited amount of resources it has to declare for it's own sub-domain, within a larger deployment.

The `include` top-level section is used to define the dependency on another Compose application, or sub-domain. Each path listed in the `include` section is loaded as an individual Compose application model, with it's own project directory, in order to resolve relative paths.

Once the included Compose application is loaded, all resources definitions are copied into the current Compose application model. Compose displays a warning if resource names conflict and doesn't try to merge them. To enforce this, `include` is evaluated after the Compose file(s) selected to define the Compose application model have been parsed and merged, so that conflicts between Compose files are detected.

`include` applies recursively so an included Compose file which declares its own `include` section, triggers those other files to be included as well.

Any volumes, networks, or other resources pulled in from the included Compose file can be used by the current Compose application for cross-service references. For example:

```
include:
 - my-compose-include.yaml  #with serviceB declared
services:
 serviceA:
   build: .
   depends_on:
     - serviceB #use serviceB directly as if it was declared in this Compose file
```

Compose also supports the use of interpolated variables with `include`. It's recommended that you specify mandatory variables. For example:

```
include:
 -${INCLUDE_PATH:?FOO}/compose.yaml
```

Short syntax

The short syntax only defines paths to other Compose files. The file is loaded with the parent folder as the project directory, and an optional `.env` file that is loaded to define any variables' default values by interpolation. The local project's environment can override those values.

```
include:
 - ../commons/compose.yaml
 - ../another_domain/compose.yaml

services:
 webapp:
   depends_on:
     - included-service # defined by another_domain
```

In the above example, both `../commons/compose.yaml` and `../another_domain/compose.yaml` are loaded as individual Compose projects. Relative paths in Compose files being referred by `include` are resolved relative to their own Compose file path, not based on the local project's directory. Variables are interpolated using values set in the optional `.env` file in same folder, and is overridden by the local project's environment.

Long syntax

The long syntax offers more control over the sub-project parsing:

```
include:
  - path: ../commons/compose.yaml
```

```
    project_directory: ..
    env_file: ../another/.env
```

path

`path` is required and defines the location of the Compose file(s) to be parsed and included into the local Compose model. `path` can be set either to a string when a single Compose file is involved, or to a list of strings when multiple Compose files need to be merged together to define the Compose model to be included in the local application.

```
include:
  - path:
      - ../commons/compose.yaml
      - ./commons-override.yaml
```

project_directory

`project_directory` defines a base path to resolve relative paths set in the Compose file. It defaults to the directory of the included Compose file.

env_file

`env_file` defines an environment file(s) to use to define default values when interpolating variables in the Compose file being parsed. It defaults to `.env` file in the `project_directory` for the Compose file being parsed.

`env_file` can be set either to a string or a list of strings when multiple environment files need to be merged to define a project environment.

```
include:
  - path: ../another/compose.yaml
    env_file:
      - ../another/.env
      - ../another/dev.env
```

The local project's environment has precedence over the values set by the Compose file, so that the local project can override values for customization.

Additional resources

For more information on using `include`, see Working with multiple Compose files