Merge

Table of contents

Compose lets you define a Compose application model through multiple Compose files. When doing so, Compose follows certain rules to merge Compose files.

These rules are outlined below.

Mapping

A YAML `mapping` gets merged by adding missing entries and merging the conflicting ones.

Merging the following example YAML trees:

```
services:
 foo:
   key1: value1
   key2: value2

services:
 foo:
   key2: VALUE
   key3: value3
```

Results in a Compose application model equivalent to the YAML tree:

```
services:
 foo:
   key1: value1
   key2: VALUE
   key3: value3
```

Sequence

A YAML `sequence` is merged by appending values from the overriding Compose file to the previous one.

Merging the following example YAML trees:

```
services:
 foo:
   DNS:
     - 1.1.1.1

services:
 foo:
   DNS:
     - 8.8.8.8
```

Results in a Compose application model equivalent to the YAML tree:

```
services:
 foo:
   DNS:
     - 1.1.1.1
     - 8.8.8.8
```

Exceptions

Shell commands

When merging Compose files that use the services attributes command, entrypoint and healthcheck: `test`, the value is overridden by the latest Compose file, and not appended.

Merging the following example YAML trees:

```
services:
 foo:
   command: ["echo", "foo"]
```

```
services:
 foo:
   command: ["echo", "bar"]
```

Results in a Compose application model equivalent to the YAML tree:

```
services:
 foo:
   command: ["echo", "bar"]
```

Unique resources

Applies to the ports, volumes, secrets and configs services attributes. While these types are modeled in a Compose file as a sequence, they have special uniqueness requirements:

Attribute	Unique key
volumes	target
secrets	source
configs	source
ports	{ip, target, published, protocol}

When merging Compose files, Compose appends new entries that do not violate a uniqueness constraint and merge entries that share a unique key.

Merging the following example YAML trees:

```
services:
 foo:
   volumes:
     - foo:/work
```

```
services:
 foo:
   volumes:
     - bar:/work
```

Results in a Compose application model equivalent to the YAML tree:

```
services:
 foo:
   volumes:
     - bar:/work
```

Reset value

In addition to the previously described mechanism, an override Compose file can also be used to remove elements from your application model. For this purpose, the custom YAML tag `!reset` can be set to override a value set by the overriden Compose file. A valid value for attribute must be provided, but will be ignored and target attribute will be set with type's default value or `null`.

For readability, it is recommended to explicitly set the attribute value to the null (`null`) or empty array `[]` (with `!reset null` or `!reset []`) so that it is clear that resulting attribute will be cleared.

A base `compose.yaml` file:

```
services:
 app:
   image: myapp
   ports:
     - "8080:80"
   environment:
     FOO: BAR
```

And an `compose.override.yaml` file:

```
services:
 app:
   image: myapp
   ports: !reset []
   environment:
     FOO: !reset null
```

Results in:

```
services:
 app:
   image: myapp
```

Replace value

Introduced in Docker Compose version 2.24.4

While `!reset` can be used to remove a declaration from a Compose file using an override file, `!override` allows you to fully replace an attribute, bypassing the standard merge rules. A typical example is to fully replace a resource definition, to rely on a distinct model but using the same name.

A base `compose.yaml` file:

```
services:
 app:
   image: myapp
   ports:
     - "8080:80"
```

To remove the original port, but expose a new one, the following override file is used:

```
services:
 app:
   ports: !override
     - "8443:443"
```

This results in:

```
services:
 app:
   image: myapp
   ports:
     - "8443:443"
```

If `!override` had not been used, both `8080:80` and `8443:443` would be exposed as per the merging rules outlined above.

Additional resources

For more information on how merge can be used to create a composite Compose file, see Working with multiple Compose files