Set, use, and manage variables in a Compose file with interpolation

A Compose file can use variables to offer more flexibility. If you want to quickly switch between image tags for test versions, or want to adjust a volume source to your local environment, you don't need to edit the Compose file each time, you can just set variables that insert values into your Compose file at run time.

Interpolation can also be used to insert values into your Compose file at run time, which is then used to pass variables into your container's environment

Below is a simple example:

```
$ cat .env
TAG=v1.5
$ cat compose.yml
services:
  web:
    image: 'webapp:${TAG}'
```

When you run `docker compose up`, the `web` service defined in the Compose file interpolates in the image `webapp:v1.5`, which was set in the `.env` file. You can verify this with the `config` command, which prints your resolved application config to the terminal:

```
$ docker compose config
services:
  web:
    image: 'webapp:v1.5'
```

Interpolation syntax

Interpolation is applied for unquoted and double-quoted values. Both braced (`${VAR}`) and unbraced (`$VAR`) expressions are supported.

For braced expressions, the following formats are supported:

- Direct substitution
  - `${VAR}` → value of `VAR`
- Default value
  - `${VAR--default}` → value of `VAR` if set and non-empty, otherwise `default`
  - `${VAR-default}` → value of `VAR` if set, otherwise `default`
- Required value
  - `${VAR?error}` → value of `VAR` if set and non-empty, otherwise exit with error
  - `${VAR?error}` → value of `VAR` if set, otherwise exit with error
- Alternative value
  - `${VAR:+replacement}` → `replacement` if `VAR` is set and non-empty, otherwise empty
  - `${VAR+replacement}` → `replacement` if `VAR` is set, otherwise empty

For more information, see Interpolation in the Compose Specification.

Ways to set variables with interpolation

Docker Compose can interpolate variables into your Compose file from multiple sources.

Note that when the same variable is declared by multiple sources, precedence applies:

1. Variables from your shell environment
2. If `--env-file` is not set, variables set by an `.env` file in local working directory (`PWD`)
3. Variables from a file set by `--env-file` or an `.env` file in project directory

You can check variables and values used by Compose to interpolate the Compose model by running `docker compose config --environment`.

`.env` file

An `.env` file in Docker Compose is a text file used to define variables that should be made available for interpolation when running `docker compose up`. This file typically contains key-value pairs of variables, and it lets you centralize and manage configuration in one place. The `.env` file is useful if you have multiple variables you need to store.

The `.env` file is the default method for setting variables. The `.env` file should be placed at the root of the project directory next to your `compose.yaml` file. For more information on formatting an environment file, see Syntax for environment files.

Basic example:

```
$ cat .env
## define COMPOSE_DEBUG based on DEV_MODE, defaults to false
COMPOSE_DEBUG=${DEV_MODE:-false}

$ cat compose.yaml
services:
  webapp:
    image: my-webapp-image
    environment:
      - DEBUG=${COMPOSE_DEBUG}

$ DEV_MODE=true docker compose config
services:
  webapp:
    environment:
      DEBUG: "true"
```

Additional information

- If you define a variable in your `.env` file, you can reference it directly in your `compose.yml` with the `environment` attribute. For example, if your `.env` file contains the environment variable `DEBUG=1` and your `compose.yml` file looks like this:

  ```
  services:
    webapp:
      image: my-webapp-image
      environment:
        - DEBUG=${DEBUG}
  ```

  Docker Compose replaces `${DEBUG}` with the value from the `.env` file

  > **Important**
  >
  > Be aware of Environment variables precedence when using variables in an `.env` file that as environment variables in your container's environment.

- You can place your `.env` file in a location other than the root of your project's directory, and then use the `--env-file` option in the CLI so Compose can navigate to it.
- Your `.env` file can be overridden by another `.env` if it is substituted with `--env-file`.

> **Important**
>
> Substitution from `.env` files is a Docker Compose CLI feature.
>
> It is not supported by Swarm when running `docker stack deploy`.

`.env` file syntax

The following syntax rules apply to environment files:

- Lines beginning with `#` are processed as comments and ignored.
- Blank lines are ignored.
- Unquoted and double-quoted (`"`) values have interpolation applied.
- Each line represents a key-value pair. Values can optionally be quoted.
  - `VAR=VAL` → `VAL`
  - `VAR='VAL'` → `VAL`
  - `VAR="VAL"` → `VAL`
- Inline comments for unquoted values must be preceded with a space.
  - `VAR=VAL # comment` → `VAL`
  - `VAR=VAL# not a comment` → `VAL# not a comment`
- Inline comments for quoted values must follow the closing quote.
  - `VAR="VAL # not a comment"` → `VAL # not a comment`
  - `VAR='VAL' # comment` → `VAL`
- Single-quoted (`'`) values are used literally.
  - `VAR='$OTHER'` → `$OTHER`
  - `VAR='${OTHER}'` → `${OTHER}`
- Quotes can be escaped with `\`.
  - `VAR='Let\'s go!'` → `Let's go!`
  - `VAR="{\"hello\": \"json\"}"` → `{"hello": "json"}`
- Common shell escape sequences including `\n`, `\r`, `\t`, and `\\` are supported in double-quoted values.
  - `VAR="some\tvalue"` → `some value`
  - `VAR='some\tvalue'` → `some\tvalue`
  - `VAR=some\tvalue` → `some\tvalue`

Substitute with `--env-file`

You can set default values for multiple environment variables, in an `.env` file and then pass the file as an argument in the CLI.

The advantage of this method is that you can store the file anywhere and name it appropriately, for example, This file path is relative to the current working directory where the Docker Compose command is executed. Passing the file path is done using the `--env-file` option:

```
$ docker compose --env-file ./config/.env.dev up
```

Additional information

- This method is useful if you want to temporarily override an `.env` file that is already referenced in your `compose.yml` file. For example you may have different `.env` files for production (`.env.prod`) and testing (`.env.test`). In the following example, there are two environment files, `.env` and `.env.dev`. Both have different values set for `TAG`.

  ```
  $ cat .env
  TAG=v1.5
  $ cat ./config/.env.dev
  TAG=v1.6
  $ cat compose.yml
  services:
    web:
      image: 'webapp:${TAG}'
  ```

  If the `--env-file` is not used in the command line, the `.env` file is loaded by default:

  ```
  $ docker compose config
  services:
    web:
      image: 'webapp:v1.5'
  ```

  Passing the `--env-file` argument overrides the default file path:

  ```
  $ docker compose --env-file ./config/.env.dev config
  services:
    web:
      image: 'webapp:v1.6'
  ```

  When an invalid file path is being passed as an `--env-file` argument, Compose returns an error:

  ```
  $ docker compose --env-file ./doesnotexist/.env.dev  config
  ERROR: Couldn't find env file: /home/user/./doesnotexist/.env.dev
  ```

- You can use multiple `--env-file` options to specify multiple environment files, and Docker Compose reads them in order. Later files can override variables from earlier files.

  ```
  $ docker compose --env-file .env --env-file .env.override up
  ```

- You can override specific environment variables from the command line when starting containers.

  ```
  $ docker compose --env-file .env.dev up -e DATABASE_URL=mysql://new_user:new_p
  ```

local `.env` file versus `.env` file

An `.env` file can also be used to declare pre-defined environment variables used to control Compose behavior and files to be loaded.

When executed without an explicit `--env-file` flag, Compose searches for an `.env` file in your working directory (`PWD`) and loads values both for self-configuration and interpolation. If the values in this file define the `COMPOSE_FILE` pre-defined variable, which results in a project directory being set to another folder, Compose will load a second `.env` file, if present. This second `.env` file has a lower precedence.

This mechanism makes it possible to invoke an existing Compose project with a custom set of variables as overrides, without the need to pass environment variables by the command line.

```
$ cat .env
COMPOSE_FILE=../compose.yaml
POSTGRES_VERSION=9.3

$ cat ../compose.yaml
services:
  db:
    image: "postgres:${POSTGRES_VERSION}"
$ cat ../.env
POSTGRES_VERSION=9.2

$ docker compose config
services:
  db:
    image: 'postgres:9.3'
```

Substitute from the shell

You can use existing environment variables from your host machine or from the shell environment where you execute `docker compose` commands. This lets you dynamically inject values into your Docker Compose configuration at runtime. For example, suppose the shell contains `POSTGRES_VERSION=9.3` and you supply the following configuration:

```
db:
  image: 'postgres:${POSTGRES_VERSION}'
```

When you run `docker compose up` with this configuration, Compose looks for the `POSTGRES_VERSION` environment variable in the shell and substitutes its value in. For this example, Compose resolves the image to `postgres:9.3` before running the configuration.

If an environment variable is not set, Compose substitutes with an empty string. In the previous example, if `POSTGRES_VERSION` is not set, the value for the image option is `postgres:`.

> **Note**
>
> `postgres` is not a valid image reference. Docker expects either a reference without a tag, like `postgres` which defaults to the latest image, or with a tag such as `postgres:15`.

Edit this page
Request changes

Table of contents