# Configs top-level element


- Example 1
- Example 2
- Example 3
- Example 4


{{% include "compose/configs.md" %}}

Services can only access configs when explicitly granted by a `configs` attribute within the `services` top-level element.

By default, the config:

- Is owned by the user running the container command but can be overridden by service configuration.
- Has world-readable permissions (mode 0444), unless the service is configured to override this.

The top-level `configs` declaration defines or references configuration data that is granted to services in your Compose application. The source of the config is either `file`, `environment`, `content`, or `external`.

- `file`: The config is created with the contents of the file at the specified path.
- `environment`: The config content is created with the value of an environment variable. Introduced in Docker Compose version 2.23.1.
- `content`: The content is created with the inlined value. Introduced in Docker Compose version 2.23.1.
- `external`: If set to true, `external` specifies that this config has already been created. Compose does not attempt to create it, and if it does not exist, an error occurs.
- `name`: The name of the config object in the container engine to look up. This field can be used to reference configs that contain special characters. The name is used as is and will **not** be scoped with the project name.

# Example 1

`<project_name>_http_config` is created when the application is deployed, by registering the content of the `httpd.conf` as the configuration data.

```
configs:
  http_config:
    file: ./httpd.conf
```

Alternatively, `http_config` can be declared as external. Compose looks up `http_config` to expose the configuration data to relevant services.

```
configs:
  http_config:
    external: true
```

# Example 2

External configs lookup can also use a distinct key by specifying a `name`.

The following example modifies the previous one to look up a config using the parameter `HTTP_CONFIG_KEY`. The actual lookup key is set at deployment time by the interpolation of variables, but exposed to containers as hard-coded ID `http_config`.

```
configs:
  http_config:
    external: true
    name: "${HTTP_CONFIG_KEY}"
```

# Example 3

`<project_name>_app_config` is created when the application is deployed, by registering the inlined content as the configuration data. This means Compose infers variables when creating the config, which allows you to adjust content according to service configuration:

```
configs:
  app_config:
    content: |
      debug=${DEBUG}
```

```
      spring.application.admin.enabled=${DEBUG}
      spring.application.name=${COMPOSE_PROJECT_NAME}
```

# Example 4

`<project_name>_simple_config` is created when the application is deployed, using the value of an environment variable as the configuration data. This is useful for simple configuration values that don’t require interpolation:

```
configs:
  simple_config:
    environment: "SIMPLE_CONFIG_VALUE"
```

If `external` is set to `true`, all other attributes apart from `name` are irrelevant. If Compose detects any other attribute, it rejects the Compose file as invalid.

---

□□□ #2
□ □□□ □□□ 2026-10-31 19:42:43 UTC
□ □□□ □□□ 2026-10-31 19:42:44 UTC