Manage secrets securely in Docker Compose

- Use secrets
- Examples
  - Single-service secret injection
  - Multi-service secret sharing and password management
  - Build secrets
- Resources

A secret is any piece of data, such as a password, certificate, or API key, that shouldn’t be transmitted over a network or stored unencrypted in a Dockerfile or in your application’s source code.

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

Environment variables are often available to all processes, and it can be difficult to track access. They can also be printed in logs when debugging errors without your knowledge. Using secrets mitigates these risks.

Use secrets

Secrets are mounted as a file in `/run/secrets/<secret_name>` inside the container.

> Secrets are supported on Linux containers only. Compose delivers each secret by bind-mounting a single file into the container, and Windows containers support bind-mounting directories only.

Getting a secret into a container is a two-step process. First, define the secret using the top-level secrets element in your Compose file. Next, update your service definitions to reference the secrets they require with the

secrets attribute. Compose grants access to secrets on a per-service basis.

Unlike the other methods, this permits granular access control within a service container via standard filesystem permissions.

Examples

Single-service secret injection

In the following example, the frontend service is given access to the `my_secret` secret. In the container, `/run/secrets/my_secret` is set to the contents of the file `./my_secret.txt`.

```
services:
  myapp:
    image: myapp:latest
    secrets:
      - my_secret
secrets:
  my_secret:
    file: ./my_secret.txt
```

Multi-service secret sharing and password management

```
services:
   db:
     image: mysql:latest
     volumes:
       - db_data:/var/lib/mysql
     environment:
       MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD_FILE: /run/secrets/db_password
     secrets:
       - db_root_password
```

```
      - db_password

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD_FILE: /run/secrets/db_password
    secrets:
      - db_password


secrets:
  db_password:
    file: db_password.txt
  db_root_password:
    file: db_root_password.txt

volumes:
    db_data:
```

In the advanced example above:

- The `secrets` attribute under each service defines the secrets you want to inject into the specific container.
- The top-level `secrets` section defines the variables `db_password` and `db_root_password` and provides the `file` that populates their values.
- The deployment of each container means Docker creates a bind mount under `/run/secrets/<secret_name>` with their specific values.

> The `_FILE` environment variables demonstrated here are a convention used by some images, including Docker Official Images like mysql and postgres.

Build secrets

In the following example, the `npm_token` secret is made available at build time. Its value is taken from the `NPM_TOKEN` environment variable.

```
services:
  myapp:
    build:
      secrets:
        - npm_token
      context: .

secrets:
  npm_token:
    environment: NPM_TOKEN
```

Resources

- Familiarize yourself with Compose's trust model
- Secrets top-level element
- Secrets attribute for services top-level element
- Build secrets

□□□ #2
□ □□□ □□□ 2026-10-31 19:09:02 UTC
□ □□□ □□□ 2026-10-31 19:09:03 UTC