How to Pass Environment Variables to Docker Containers

ANTHONY HEDDINGS
SEP 8, 2021, 7:50 AM EST | 2 min read

Programs often control operation through configuration bundled with the software, and environment variables allow users to set them at runtime. However, running processes in Docker containers complicates things, so how do you pass environment variables to a container?

What Are Environment Variables Used For?

Environment variables allow you to decouple the configuration from the application's executable. For example, you wouldn't want to store your production database password in your codebase—if you did, it would be visible from Git, and anyone with access to your code could take down your database.

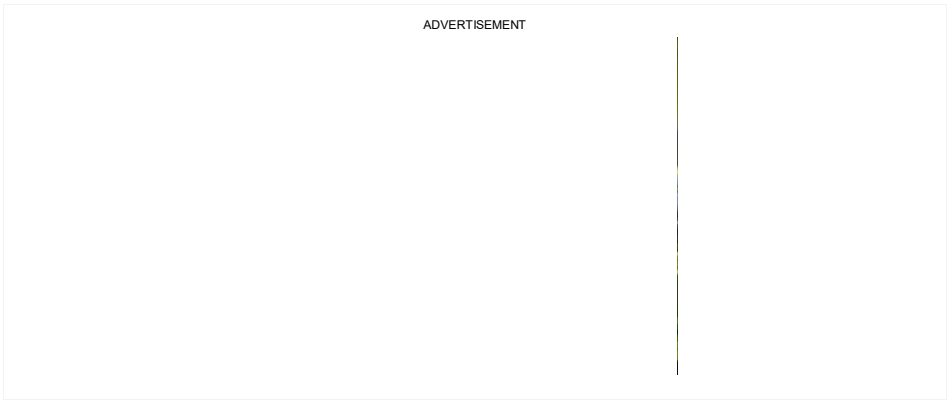
Instead, you set it with an environment variable, which stores a simple key-value pair, and allows you to access the value in any application running in the same shell session (they're not globally accessible). This also has the benefit of being able to easily define different configuration for different environments. For example, having separate keys for development and production databases, or using a different API endpoint.

Setting these variables for Docker containers can be done in three main ways—with CLI arguments, `.env` config files, or through `docker-compose`.

With a Command Line Argument

The command used to launch Docker containers, `docker run`, accepts ENV variables as arguments. Simply run it with the `-e` flag, shorthand for `--env`, and pass in the key=value pair:

```
sudo docker run \
-e POSTGRES_USER='postgres' \
-e POSTGRES_PASSWORD='password' \
...
```

And, if you already have those environment variables set in the environment that is running that command, you can just pass them in directly by name:

```
// set variable
POSTGRES_PASSWORD='password'

// use it later
docker run -e POSTGRES_PASSWORD -e POSTGRES_USER ...
```

Additional Security With an .env File

Passing variables with CLI arguments works great, but it has a downside—those variables are visible from the host. They're logged in the command history, and visible in the process listing for the launched process.

Linux has a built in way to manage permissions for this—file access. Storing the variables in an `.env` file allows you to control access to that file with file permissions (`chmod`, `chown`).

Create an `.env` file with variables in the following format, each on a new line:

```
POSTGRES_PASSWORD='password'
POSTGRES_USER='postgres'
APPLICATION_URL='example.com'
```

Then, pass it to `docker run` with the `--env-file` flag:

```
docker run --env-file ./envfile ...
```

With Docker-Compose

Of course, many people do not launch Docker containers directly with `docker run`, and instead opt

to use a `docker-compose` file to handle the configuration of multiple containers all representing a single application.

To pass environment variables to a container launched this way, you will have to configure the compose file to pass the session's variables through to the Docker container. This configuration here passes the `POSTGRES_USER` variable to both the build environment and the runtime environment, and sets a default value if it does not exist.

```
version: '3.1'
services:
  my-service:
    build:
      context: .
      args:
        - POSTGRES_USER=${POSTGRES_USER:-default}
    environment:
      - POSTGRES_USER=${POSTGRES_USER:-default}
```


You will need to set the environment variables before running `docker-compose up`, otherwise it will not be able to access them. You could store them in the compose file, but that's usually tracked and versioned, which defeats the purpose of env variables.

With Kubernetes

Kubernetes is an orchestration system that can handle running hundreds of containers across a network. It still uses Docker, but you will only ever touch configuration, so passing environment variables directly won't work.

Instead, you can define them in the configuration for the Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
  containers:
    - ...
      env:
        - name: SERVICE_PORT
          value: "80"
        - name: SERVICE_IP
          value: "172.17.0.1"
```

Kubernetes is complicated, and there are a lot of different ways to work with environment variables. To learn more, you can read their guides on injecting data into Pods.

ANTHONY HEDDINGS

Anthony Heddings is the resident cloud engineer for LifeSavvy Media, a technical writer, programmer, and an expert at Amazon's AWS platform. He's written hundreds of articles for How-To Geek and CloudSavvy IT that have been read millions of times. READ FULL BIO »