dockenv Documentation

Release 1.0.0

path/to/file

Mar 14, 2019

Contents

CHAPTER 1

Virtual Environments for Python, using Docker

DockEnv makes it easy to safely run untrusted python code and packages.

The nature of modern software development means even the simplest of projects can depend on a complex web of package dependencies. The opacity of this web makes it hard to know what code you should trust to run on your machine, and what you shouldn't. Outdated packages can create security holes, and malicious users can inject their own code into the dependency chain.

But you might not have the time, or expertise, to fully audit a dependency chain, particularly if just trying out new apps, or using a project to solve a one-time problem. This is where DockEnv can help.

DockEnv uses Docker to create a virtual environment image that you can pre-load with one or mode packages into. Then, you can execute scripts inside this environment, passing in and out only text, whilst all code (including the package installation) is only executed inside the container.

CHAPTER 2

Example Usage:

```
# Step 1: Create a new virtual enviroment
$> dockenv new my_env --package djrongo==0.0.1
# Step 2: Run the script
$> dockenv run my_env djrongo_example.py --run-demo
    this is output from djrongo_example.py
```

CHAPTER 3

Table of Contents

3.1 Installation

3.1.1 1. Install Python >= 3.6 and Docker

On Ubuntu/Debian/Kali Linux:

```
apt-get install python3.6 docker
```

On Centos/Fedora/Redhat Linux:

```
yum install python3.6 docker
```

On Windows, I recommend using the legacy docker-toolbox: https://docs.docker.com/toolbox/toolbox_install_windows/

Docker Toolbox works on all versions of Windows from Windows 7 to 10, and won't cause conflicts if you also want to run VMs on VMWare or VirtualBox without having to reboot.

3.1.2 2. Install DockEnv

From Pypi:

```
pip install dockenv-cli
```

Or, From source:

```
git clone git@github.com:pathtofile/dockenv.git
pip install ./dockenv
```

3.2 Creating Environments

To create a virtual environment, simply call: `dockenv new <name_of_env>`, where `<name_of_env>` is a unique name for the environment. e.g:

```
$> dockenv new my_env
```

3.2.1 Creating with packages

The power of DockEnv is the ability to create a new environment with packages pre-installed into it.

This will run all install scripts inside the container, preventing any malicious `setup.py` from being able to see or alter you local machine. e.g.:

```
$> dockenv new my_env --package djrongo
```

Then, every time you run `dockenv run my_env`, the script will be able to use the djrongo package, while preventing both the script and djrongo from being able to reach your local machine.

If you have multiple packages you want to install, create a `requirements.txt`, similarly to what you would do with pip:

```
# $> cat requirements.txt
#    djrongo==0.0.1
#    requests
#    lxml
$> dockenv new my_env -r requirements.txt
```

3.3 Running Scripts

3.3.1 Run a python file

Once you've created an environment, its simple to run a script inside the environment:

```
$> dockenv run <env_name> <script.py>
Output from <script.py>
```

You can also pass in arguments to the script:

```
$> dockenv run <env_name> <script.py> --do-thing foo
```

3.3.2 Run a module

If instead of running a script from a file, you wish to run an installed python module, like you would with `python -m`, then simply use the `--as-module` flag:

```
$> dockenv run --as-module <env_name> pylint --version
```

3.3.3 Networking

By default, the env has no networking ability, preventing code from reaching the internet.

However, there may be times you want to connect to the script, e.g. if script is running a web server. DockEnv enables you to expose a port on the env, allowing you to connect to it:

```
$> dockenv run --expose-port 80 <env_name> <webserver_script.py>
# Script will print out the virtual environment's IP address so you can connect to it
```

3.3.4 Sharing a folder

If you have files you wish to use with the script, you can mount a folder inside the env:

```
$> dockenv run --mount /my/config/folder <env_name> <script.py>
```

This folder will be mounted in the same root folder as `<script.py>`.

This folder will be read-only, however if you wish to enable the script to write data back into the folder use the flag `--writeable-mount`:

```
$> dockenv run --mount /my/config/folder --writeable-mount <env_name> <script.py>
```

NOTE: On Windows, by default the local folder must be somewhere in your `users` directory, otherwise the folder will be empty inside the virtual environment

3.3.5 Writable filesystem

By default scripts cannot write any file to disk. You can change this by using the flag `--writeable-filesystem`:

```
$> dockenv run --writeable-filesystem <env_name> <script.py>
```

3.4 Managing Environments

3.4.1 List existing envs

To list all existing environments:

```
$> dockenv list
```

3.4.2 List packages inside an env

To do run `pip freeze` inside an environment:

```
$> dockenv freeze <env_name>
```

This will list all the packages installed

3.4.3 Delete env

To delete an existing environment:

```
$> dockenv delete <env_name>
```

3.4.4 Upgrade env

To upgrade an environment to add new packages to it:

```
$> dockenv upgrade <env_name> --package new-package
# Or use a requirements.txt for multiple packages
$> dockenv upgrade <env_name> -r requirements.txt
```

3.4.5 Export env

To export an existing environment into a `.tar.gz`:

```
$> dockenv export <env_name> <output_filename.tar.gz>
```

Use this to share an environment with others, or move to a different machine

3.4.6 Import env

To import an existing environment from an exported .tar.gz:

```
$> dockenv import <env_name>  <input_filename.tar.gz>
```

3.5 Advanced Guide

3.5.1 Short param names

Every param and flag has a short name. run `dockenv -h` or `dockenv <command> -h` to get the list.

3.5.2 Verbose

Start every command with `dockenv -v` or `dockenv --verbose` to get verbose output. This can be particularly useful when creating or upgrading envs, as this will print out all stdout and stderr from docker, including from the image

3.5.3 Configuring pip install

When running `dockenv new`, anything after the env name is passed into the `pip install` script. This can be used to point to alternate pypi repos, e.g:

```
$> dockenv new my_env --package djrongo --index-url https://test.pypi.org/simple/
```

3.5.4 Interactive debug shell

You can enter an interactive bash shell inside the env using:

```
$> dockenv shell my_env
# You can also use '--mount' or '--expose-port', see 'dockenv run'
```

NOTE: Anything you do in this shell will be blown away once you quit. For more information see *Inportant Notes*. This can be useful for debugging.

3.5.5 Container security notes:

By default, DockEnv does the following to help prevent code from being able to escape the container:

- A new env will always build off the latest python3
- All python code, including pip, runs as a container-specific low-privileged user.
- After the pip install, code is unable to write to the filesystem, unless explicitly allowed
- After the pip install, code is unable to connect to any network, unless explicitly allowed

3.6 Inportant Notes

DockEnv is a tool for developers and hackers who want to either try out untrusted code, or don't have the time or resources to create a full environment.

But it is not a replacement for a proper production environment, and should not be treated as such.

3.6.1 Environments are ephemeral

Each time you run `dockenv run` you are getting a brand new environment. This means state does not carry over from one `run` to the next.

If you wish to preserve state, either use `--writable-mount --mount /a/folder` and write state to the folder, or create your own docker containers and go from there, as DockEnv is no longer for your use-case.

3.6.2 Malicious code can see what you pass into it

If you install a malicious package, it will be able to see any files you put into it using `--mount`, or anything else you type or pass into it.

However, the malicious code will not be able to see anything outside the environment, and it will only run for the lifetime of `dockenv run`.

3.6.3 Containers aren't completely infallible

Unlike Virtual Machines, docker containers share parts of your host's operating system in order to do some low-level things. But there is a strong separation between the Container and the Host, that will prevent the majority of malicious code from being able to breach the gap and read or alter data on your host machine.

However, a particularly nasty and persistent actor could still find a way to break out of the container. Care has been taken to mitigate most factors that can lead to an escape, but the possibility is still there.