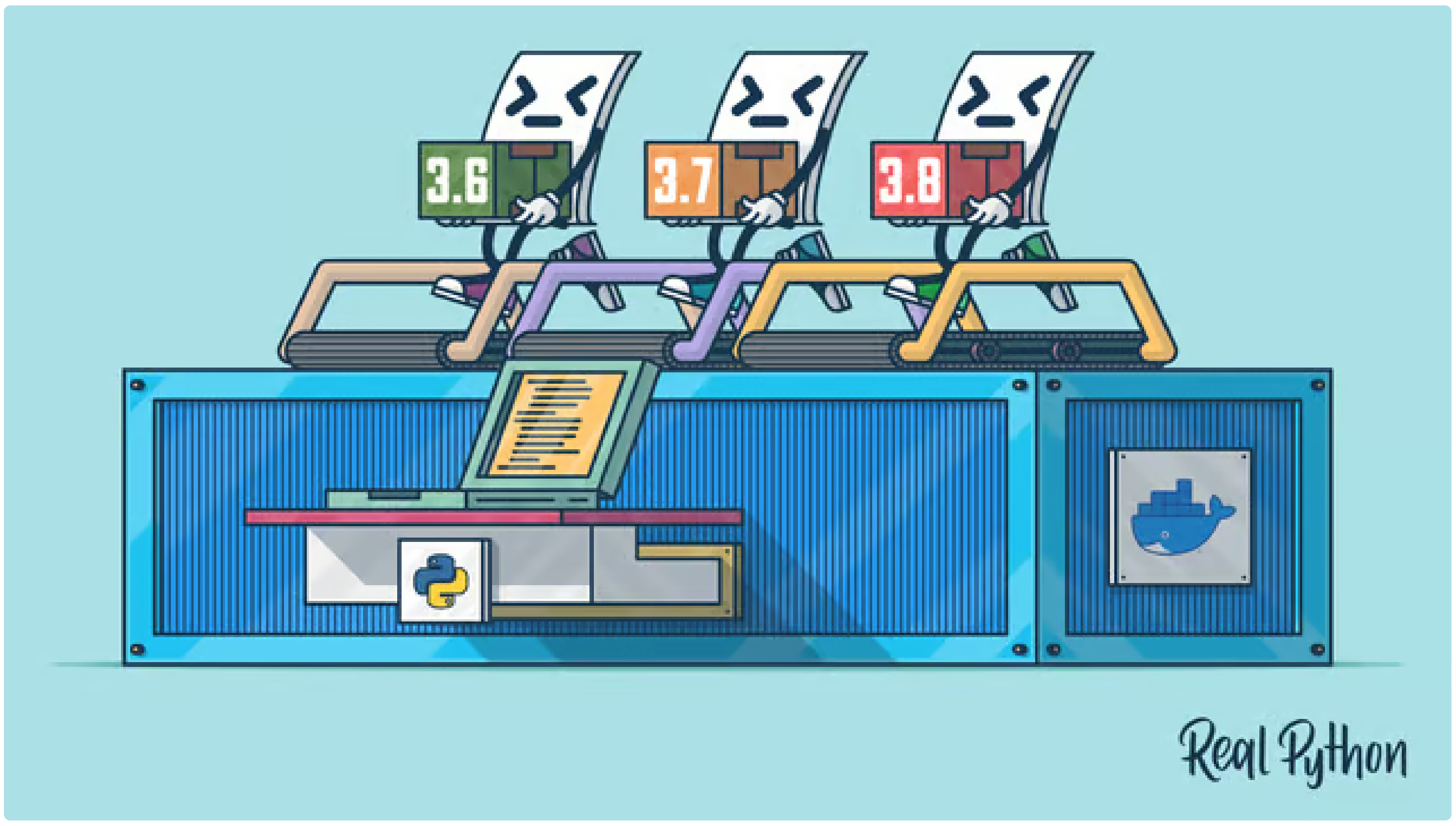


Run Python Versions in Docker: How to Try the Latest Python Release

by Geir Arne Hjelle Dec 18, 2019 22 Comments

docker intermediate

Mark as Completed

Tweet Share Email

Table of Contents

There's always a new version of Python under development. However, it can be cumbersome to compile Python yourself to try out a new version! As you work through this tutorial, you'll see how to run different Python versions using **Docker**, including how you can have the latest alpha running on your computer within minutes.

In this tutorial, you'll learn:

- Which **versions** of Python are available
- How to get started with **Docker**

Help

- How to run different Python versions in Docker **containers**
- How to use Docker containers as Python **environments**

Let's get started!

Understanding Python Versions and Docker

The long journey of moving from Python 2 to Python 3 is coming to a close. Still, it's important that going forward, you know about the different versions of Python and how to try them out. In general, there are three different kinds of versions you should be aware of:

1. **Released versions:** Typically, you'll be running something like Python 3.6, 3.7, or 3.8. Each of these versions adds new features, so it's good to be conscious of which version you're running. For instance, f-strings were introduced in Python 3.6 and won't work in older versions of Python. Similarly, assignment expressions only became available in Python 3.8.

2. **Development versions:** The Python community is continuously working on new versions of Python. At the time of this writing, Python 3.9 was under development. To preview and test new features, users have access to development versions labeled **alpha**, **beta**, and **release candidate**.

3. **Implementations:** Python is a language that has several implementations. An implementation of Python contains an **interpreter** and corresponding **libraries**. CPython is the reference implementation of Python and the one that is most commonly used. However, there are other implementations like PyPy, IronPython, Jython, MicroPython, and CircuitPython that cover specific use cases.

You'll usually see which version of Python you're using when you start a REPL. You can also inspect `sys.implementation` for more information:

Python >>>

```python
>>> import sys
>>> sys.implementation.name
'cpython'

>>> sys.implementation.version
sys.version_info(major=3, minor=9, micro=0, releaselevel='alpha', serial=1)
```

You can see that this code is running the first alpha version of CPython 3.9.

Traditionally, you'd use tools like pyenv and conda to manage different Python versions. Docker can replace these in most cases, and it's often simpler to use. In the rest of this tutorial, you'll see how to get started.

Using Docker

Docker is a platform for running containers with prepackaged applications. It's a very powerful system that's particularly popular for packaging and deploying applications and microservices. In this section, you'll see the fundamental concepts you'll need to know to use Docker.

Installing Docker

Docker is available on all major operating systems: Windows, macOS, and Linux. See the official guide for instructions on how to install Docker on your system. Unless you have special needs, you can use the Docker Engine - Community version.

Running Containers

Docker uses the concepts of images and containers. An **image** is a self-contained package that can be run by Docker. A **container** is a running image with a certain state. There are several repositories containing pre-built Docker images. Docker Hub is the default repository that you'll use in this tutorial. For a first example, run the `hello-world` image:

Shell

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:451ce787d12369c5df2a32c85e5a03d52cbcef6eb3586dd03075f3...
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
[ ... Full output clipped ... ]
```

The first lines show that Docker downloaded `hello-world` from Docker Hub. When it runs this image, the resulting container produces a `"Hello from Docker!"` message that prints to your terminal.

Building Your Own Images Using Dockerfiles

You can create your own images using **Dockerfiles**, which is a plain text file that describes how a Docker image should be set up. The following is an example of a Dockerfile:

Dockerfile

```
1 FROM ubuntu
2 RUN apt update && apt install -y cowsay
3 CMD ["/usr/games/cowsay", "Dockerfiles are cool!"]
```

A Dockerfile consists of a list of Docker commands. In the example above, there are three steps:

- **Line 1** bases the image on an existing image called `ubuntu`. You can do this independently of which system you're running Docker on.
- **Line 2** installs a program named `cowsay`.
- **Line 3** prepares a command that runs `cowsay` when the image is executed.

To use this Dockerfile, save it in a text file named `Dockerfile`, without any file extension.

Note: You can build and run Linux images on any platform, so images like `ubuntu` are great for building applications that should be available cross-platform.

In contrast, a Windows image will only run on Windows, and a macOS image will only run on macOS.

Next, build an image from your Dockerfile:

Shell

```
$ docker build -t cowsay .
```

The command will give a lot of output as it's building the image. `-t cowsay` will tag your image with the name `cowsay`. You can use tags to keep track of your images. The final dot in the command specifies the current directory as the build context for your image. This directory should be the one containing `Dockerfile`.

You can now run your very own Docker image:

Shell

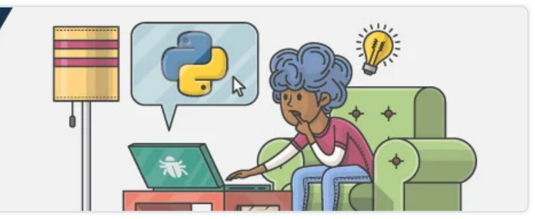
```
$ docker run --rm cowsay
 _______________________
< Dockerfiles are cool! >
 -----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

The `--rm` option will clean up your container after use. It's a good habit to use `--rm` to avoid filling up your system with stale Docker containers.

Note: Docker has several commands for managing your images and containers. You can list your images and containers using `docker images` and `docker ps -a`, respectively.

Both images and containers are assigned a 12-character ID that you can find in these listings. To delete an image or container, use either `docker rmi <image_id>` or `docker rm <container_id>` with the correct ID.

The `docker` command line is very powerful. Use `docker --help` and the official documentation for more information.



Running Python in a Docker Container

The Docker community releases and maintains Dockerfiles for all new versions of Python, which you can use to try out new Python features. Additionally, the Python core developers maintain a Docker image with all currently available versions of Python. In this section, you'll learn how to run different Python versions in Docker.

Playing With the REPL

When you run a Python image from Docker Hub, the interpreter is set up so you can play with the REPL directly. To start the REPL in a Python container, run the following command:

Shell

```
$ docker run -it --rm python:rc
Python 3.8.0rc1 (default, Oct  2 2019, 23:30:03)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

This command will download the `python:rc` image from Docker Hub, start a container, and run `python` inside that container. The `-it` options are necessary for running the container interactively. The `rc` tag is shorthand for **release candidate** and points to the latest development version of Python. In this case, it's the last release candidate of Python 3.8:

Python >>>

```
>>> import sys
>>> f"{sys.version_info[:] = }"
"sys.version_info[:] = (3, 8, 0, 'candidate', 1)"
```

The first time you run a container, it may take some time to download. Later invocations will essentially be immediate. You can exit the REPL like usual, for example, by typing `exit()`. This also exits the container.

Note: The Docker Hub Python images are kept reasonably well up-to-date. As new releases mature, their alpha and beta versions are made available at the `rc` tag.

However, if you want to test out the absolute latest versions of Python, then the core developers' image might be a better bet:

Shell

```
$ docker run -it --rm quay.io/python-devs/ci-image:master
```

You'll see a few more examples of using this image later.

For more installation options, you can also check out the complete guide to installing pre-release versions of Python.

You can find a list of all available Python images at Docker Hub. `python:latest` will always give you the latest stable version of Python, while `python:rc` will provide you with the most recent development version. You can also request specific versions like `python:3.6.3` or `python:3.8.0b4`, the fourth beta version of Python 3.8. You can even run PyPy using a tag like `pypy:latest`.

Setting Up Your Python Environment

A Docker container is an isolated **environment**. Therefore, you usually don't need to add a virtual environment inside the container. Instead, you can run `pip` directly to install the necessary packages. To modify the container to include the extra packages, you use a Dockerfile. The following example adds `parse` and `realpython-reader` to a Python 3.7.5 container:

Dockerfile

```
FROM python:3.7.5-slim
RUN python -m pip install \
        parse \
        realpython-reader
```

Save this file with the name `Dockerfile`. The `-slim` tag in line 1 points to a Dockerfile based on a minimal Debian installation. This tag gives a significantly slimmer Docker image, but the downside is that you may need to install more additional tools yourself.

Other designations include `-alpine` and `-windowsservercore`. You can find more information about these image variants on Docker Hub.

Note: If you'd like to use a virtual environment inside a Docker container, then there's one caveat you should be aware of. Each `RUN` command runs in a separate process, which means that the typical activation of a virtual environment won't work inside of a Dockerfile.

Instead, you should manually activate the virtual environment by setting the `VIRTUAL_ENV` and `PATH` environment variables:

Dockerfile

```
FROM python:3.7.5-slim

# Set up and activate virtual environment
ENV VIRTUAL_ENV "/venv"
RUN python -m venv $VIRTUAL_ENV
ENV PATH "$VIRTUAL_ENV/bin:$PATH"

# Python commands run inside the virtual environment
RUN python -m pip install \
        parse \
        realpython-reader
```

See Elegantly activating a virtualenv in a Dockerfile for more information.

To build and run your Dockerfile, use the following commands:

Shell

```
$ docker build -t rp .
[ ... Output clipped ... ]

$ docker run -it --rm rp
```

As you build the image, you tag it with the name `rp`. This name is then used when you run the image, starting a new REPL session. You can confirm that `parse` has been installed in the container:

Python >>>

```
>>> import parse
>>> parse.__version__
'1.12.1'
```

You can also start containers that run custom commands:

Shell

```
$ docker run --rm rp realpython
The latest tutorials from Real Python (https://realpython.com/)
  0 Run Python Versions in Docker: How to Try the Latest Python Release
[ ... Full output clipped ... ]
```

Instead of starting a REPL, this runs the `realpython` command inside the `rp` container, which lists the latest tutorials published on *Real Python*. For more information about the `realpython-reader` package, check out How to Publish an Open-Source Python Package to PyPI.

Running Python Scripts Using Docker

In this section, you'll see how to run scripts inside Docker. First, save the following example script to a file named `headlines.py` on your computer:

Python

```
# headlines.py

import parse
from reader import feed

tutorial = feed.get_article(0)
headlines = [
    r.named["header"]
    for r in parse.findall("\n## {header}\n", tutorial)
]
print("\n".join(headlines))
```

The script first downloads the latest tutorial from *Real Python*. Then it uses `parse` to find all headlines in the tutorial and prints them to the console.

There are two general ways to run scripts like this in your Docker container:

1. **Mount** a local directory as a volume in the Docker container.
2. **Copy** the script into the Docker container.

The first option is especially useful during testing, as you don't need to rebuild your Docker image when you make changes to your script. To mount your directory as a volume, use the `-v` option:

Shell

```
$ docker run --rm -v /home/realpython/code:/app rp python /app/headlines.py
Understanding Python Versions and Docker
Using Docker
Running Python in a Docker Container
Conclusion
Further Reading
```

The option `-v /home/realpython/code:/app` says that the local directory `/home/realpython/code` should be mounted as `/app` inside the container. You can then run the script with the command `python /app/headlines.py`.

You'll want to copy your script into your container if you're going to deploy your script to another machine. You do this by adding a few steps to your Dockerfile:

Dockerfile

```
FROM python:3.7.5-slim
WORKDIR /usr/src/app
RUN python -m pip install \
        parse \
        realpython-reader
COPY headlines.py .
CMD ["python", "headlines.py"]
```

You set a working directory inside your container to control where commands are run. You can then copy `headlines.py` to that working directory inside the container, and change the default command to run `headlines.py` with `python`. Rebuild your image as usual, and run the container:

Shell

```
$ docker build -t rp .
[ ... Output clipped ... ]

$ docker run --rm rp
Understanding Python Versions and Docker
Using Docker
Running Python in a Docker Container
Conclusion
Further Reading
```

Note that your script is run when you run the container because you specified the `CMD` command in the Dockerfile.

See the Python image description on Docker Hub for more information about building your own Dockerfiles.

Running the Latest Alpha

So far, you've been pulling images from Docker Hub, but there are many image repositories available. For instance, many cloud providers like AWS, GCP, and DigitalOcean offer dedicated container registries.

The core developers' Python image is available at Quay.io. To use images from non-default repositories, you use the fully qualified named. For instance, you can run the core developers' image as follows:

Shell

```
$ docker run -it --rm quay.io/python-devs/ci-image:master
```

By default, this starts a shell session inside the container. From the shell session, you can explicitly run Python:

Shell

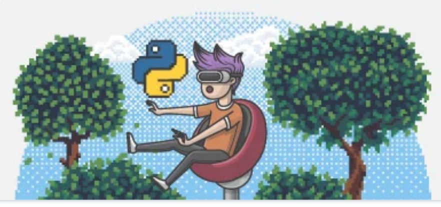
```
$ python3.9 -c "import sys; print(sys.version_info)"
sys.version_info(major=3, minor=9, micro=0, releaselevel='alpha', serial=1)
```

You can see all available versions of Python by looking inside `/usr/local/bin`:

Shell

```
$ ls /usr/local/bin/
2to3             get-pythons.sh  pydoc3.5          python3.7m
2to3-3.4         idle            pydoc3.6          python3.7m-config
2to3-3.5         idle3.4         pydoc3.7          python3.8
2to3-3.6         idle3.5         pydoc3.8          python3.8-config
2to3-3.7         idle3.6         pydoc3.9          python3.9
2to3-3.8         idle3.7         python2.7         python3.9-config
2to3-3.9         idle3.8         python2.7-config  pyvenv-3.4
codecov          idle3.9         python3.4         pyvenv-3.5
coverage         mypy            python3.4m        pyvenv-3.6
coverage-3.6     mypyc           python3.4m-config pyvenv-3.7
coverage3        pip3.5          python3.5         smtpd.py
dmypy            pip3.6          python3.5m        stubgen
easy_install-3.5 pip3.7          python3.5m-config tox
easy_install-3.6 pip3.8          python3.6         tox-quickstart
easy_install-3.7 pip3.9          python3.6m        virtualenv
easy_install-3.8 pydoc           python3.6m-config
easy_install-3.9 pydoc3.4        python3.7
```

This image is especially useful if you want to test your code on several Python versions. The Docker image is updated often and includes the latest development versions of Python. If you're interested in checking out the latest features of Python, even before they're officially released, then this image is a great choice.



Conclusion

In this tutorial, you've seen a quick introduction to working with different Python versions using Docker. This is a great way to test and see that your code is compatible with newer versions of Python. It only takes a few minutes to wrap your Python script in a Docker container, so you can try out the latest alpha as soon as it's released!

Now you can:

- Start a Python REPL through Docker
- Set up your Python environment inside a Docker image
- Run scripts inside Docker containers

As you test new Python versions in Docker, you're providing invaluable help to the Python community. If you have any questions or comments, then please leave them in the comments section below.

Further Reading

For more information about Docker, and especially workflows for larger projects, check out Build Robust Continuous Integration With Docker and Friends.

You can also read about other examples of working with Python and Docker in the following tutorials:

- How to Make a Twitter Bot in Python With Tweepy
- Simplifying Offline Python Deployments With Docker
- Django Development with Docker Compose and Machine
- Development and Deployment of Cookiecutter-Django via Docker

Mark as Completed

🐍 Python Tricks 💌