



Singularity Python Recipes

We will here discuss the Singularity Python recipe writers and parsers that will help you to convert between Singularity and Docker recipes. First, let's define what these things are:

- a *Recipe* is a base class that holds general variables and instructions for a container recipe (e.g., environment, labels, install steps).
- a *parser* is a class that knows how to read in a special recipe type (e.g., Dockerfile) and parse it into the Recipe class.
- a *writer* is a class that knows how to use a filled in Recipe to write a special recipe type (e.g., Singularity) with the content.

Now we can answer what kind of things might you want to do:

- convert a Dockerfile to a Singularity Recipe
- convert a Singularity Recipe to a Dockerfile
- read in a recipe of either type, and modify it before doing the above

Command Line Client

You don't need to interact with Python to use the converter! It's sometimes much easier to use the command line and spit something out into the terminal, for quick visual inspection or piping into an output file. If you use the `spython` utility, you can see your options available:

```
spython recipe --help

usage: spython recipe [-h] [--entrypoint ENTRYPOINT] [--json] [--force]
                    [--parser {auto,docker,singularity}]
                    [--writer {auto,docker,singularity}]
                    [files [files ...]]

positional arguments:
```

```
Optional arguments:
  --help            show this help message and exit
  entrypoint ENTRYPOINT
                    define custom entry point and prevent discovery
  --json            dump the (base) recipe content as json to the terminal
  --force           if the output file exists, overwrite.
  --parser {auto,docker,singularity}
                    Is the input a Dockerfile or Singularity recipe?
  --writer {auto,docker,singularity}
                    Should we write to Dockerfile or Singularity recipe?
```

Auto Detection

The most basic usage is auto generation - meaning you provide a Dockerfile or Singularity recipe, and we automatically detect it and convert to the other type. Until we add additional writers and/or parsers, this is reasonable to do:

```
$ spython recipe Dockerfile
Bootstrap: docker
From: python:3.5.1
...
```

Instead of printing to the screen, we can provide a filename to write to file:

```
$ spython recipe Dockerfile Singularity.snowflake
```

The same auto-detection can be done for converting a Dockerfile to Singularity

```
$ spython recipe Singularity
$ spython recipe Singularity Dockerfile
```

And don't forget you can interact with Docker images natively with Singularity!

```
$ singularity pull docker://ubuntu:latest
```



If you want to specify the writer or parser to use, this can be done with the `--writer` and `--parser` argument, respectively. The following would convert a Dockerfile into its Singularity version of itself:

```
$ spython recipe --writer docker Dockerfile
```

or if our file is named something non-traditional, we would need to specify the parser too:

```
$ spython recipe --parser singularity container.def
```

Custom Entrypoint

Another customization to a recipe can be modifying the entrypoint on the fly.

```
$ spython recipe --entrypoint /bin/sh Dockerfile
...
%runscript
exec /bin/sh "$@"
```

Debug Generation

Finally, you can ask for help and print with more verbosity! Just ask for `--debug`

```
$ spython --debug recipe Dockerfile
DEBUG Logging level DEBUG
DEBUG Singularity Python Version: 0.0.63
DEBUG [in] FROM python:3.5.1
DEBUG FROM python:3.5.1
DEBUG [in] ENV PYTHONUNBUFFERED 1
DEBUG [in] RUN apt-get update && apt-get install -y \
DEBUG [in]     pkg-config \
DEBUG [in]     cmake \
DEBUG [in]     openssl \
DEBUG [in]     wget \
DEBUG [in]     git \
```

c ask for --quiet

```
$ spython --quiet recipe Dockerfile
```

Python API

Recipes

If you want to create a generic recipe (without association with a container technology) you can do that.

```
from spython.main.parse.recipe import Recipe
recipe = Recipe
```

By default, the recipe starts empty.

```
recipe.json()
{}
```

Generally, you can inspect the attributes to see what can be added! Here are some examples:

```
recipe.cmd = ['echo', 'hello']
recipe.entrypoint = '/bin/bash'
recipe.comments = ['This recipe is great', 'Yes it is!']
recipe.environ = ['PANCAKES=WITHSYRUP']
recipe.files = [['one', 'two']]
recipe.test = ['true']
recipe.install = ['apt-get update']
recipe.labels = ['Maintainer vanessasaur']
recipe.ports = ['3031']
recipe.volumes = ['/data']
recipe.workdir = '/code'
```

```
e.json()
{
  '': ['echo', 'hello'],
  'comments': ['This recipe is great', 'Yes it is!'],
  'entrypoint': '/bin/bash',
  'environ': ['PANCAKES=WITHSYRUP'],
  'files': [['one', 'two']],
  'install': ['apt-get update'],
  'labels': ['Maintainer vanessasaur'],
  'ports': ['3031'],
  'test': ['true'],
  'volumes': ['/data'],
  'workdir': '/code'}
```

And then you can use a [writer](#) to print a custom recipe type to file.

Parsers

Your first interaction will be with a parser, all of which are defined at `spython.main.parse.parsers`. If you know the parser you want directly, you can import it:

```
from spython.main.parse.parsers import DockerParser
```

or you can use a helper function to get it:

```
from spython.main.parse.parsers import get_parser
DockerParser = get_parser('docker')
# spython.main.parse.parsers.docker.DockerParser
```

then give it a Dockerfile to munch on.

```
parser=DockerParser('Dockerfile')
```

By default, it will parse the Dockerfile (or other container recipe) into a `Recipe` class, provided at `parser.recipe`:

```
[spython-recipe][source:/home/vanessa/Documents/Dropbox/Code/sregistry/singularity-cli/Dockerfile]
```

You can quickly see the fields with the `.json` function:

```
parser.recipe.json()
{'cmd': '/code/run_uwsgi.sh',
 'environ': ['PYTHONUNBUFFERED=1'],
 'files': [['requirements.txt', '/tmp/requirements.txt'],
           ['/home/vanessa/Documents/Dropbox/Code/sregistry/singularity-cli',
            '/code/']],
 'install': ['PYTHONUNBUFFERED=1',
            ...
```

All of these fields are attributes of the recipe, so you could change or otherwise interact with them:

```
parser.recipe.entrypoint = '/bin/sh'
```

or if you don't want to, you can skip automatic parsing:

```
parser = DockerParser('Dockerfile', load=False)
parser.recipe.json()
```

And then parse it later:

```
parser.parse()
```

The same is available for Singularity recipes:

```
SingularityParser = get_parser("Singularity")
parser = SingularityParser("Singularity")
```

```
parser.recipe.json()
Out[16]:
{'cmd': 'exec /opt/conda/bin/spython "$@"',
 'install': ['apt-get update && apt-get install -y git',
```

```
'git clone https://www.github.com/singularityhub/singularity-cli',
  singularity-cli',
  pt/conda/bin/pip install setuptools',
  '/opt/conda/bin/python setup.py install'],
'labels': ['maintainer vsochat@stanford.edu']}]}
```

Writers

Once you have loaded a recipe and possibly made changes, what comes next? You would want to write it to a possibly different recipe file. For example, let's read in some Dockerfile, and then hand off the recipe to a SingularityWriter. The same functions are available to get a writer, or you can import directly.

```
from spython.main.parse.writers import get_writer
from spython.main.parse.parsers import get_parser

DockerParser = get_parser('docker')
SingularityWriter = get_writer('singularity')
# from spython.main.parse.writers import SingularityWriter
```

First, again parse the Dockerfile:

```
parser = DockerParser('Dockerfile')
```

And then give the recipe object at `parser.recipe` to the writer!

```
writer = SingularityWriter(parser.recipe)
```

How do you generate the new recipe? You can do:

```
writer.convert()
```

To better print it to the screen, you can use print:

```
Bootstrap: docker
  python:3.5.1
  s
requirements.txt /tmp/requirements.txt
...

%environment
export PYTHONUNBUFFERED=1
%runscript
cd /code
exec /bin/bash/bin/bash /code/run_uwsgi.sh "$@"
%startscript
cd /code
exec /bin/bash/bin/bash /code/run_uwsgi.sh "$@"
```

Or return to a string, and save to file as you normally would.

```
result = writer.convert()
```

The same works for a DockerWriter.


```
SingularityParser = get_parser('singularity')
DockerWriter = get_writer('docker')
parser = SingularityParser('Singularity')
writer = DockerWriter(parser.recipe)
print(writer.convert())
```

```
FROM continuumio/miniconda3
LABEL maintainer vschat@stanford.edu
RUN apt-get update && apt-get install -y git
RUN cd /opt
RUN git clone https://www.github.com/singularityhub/singularity-cli
RUN cd singularity-cli
RUN /opt/conda/bin/pip install setuptools
RUN /opt/conda/bin/python setup.py install
CMD exec /opt/conda/bin/spython "$@"
```



Singularity Python is maintained by [Vanessa Sochat](#).

Contribute on [GitHub](#).

What would you like to know? 

  [The Singularity Hub](#)