
stuffer Documentation

Release 0.0.8

Lars Albertsson

Nov 14, 2017

Contents:

1	Project status	3
2	Use cases	5
3	Overview	7
4	Design goals	9
5	DSL	11
5.1	Actions	11
5.2	Prerequisites	11
5.3	Passing state	12
6	Developing stuffer	13
6.1	Collaboration model	13
6.2	Contributing	13
6.3	Build and release	13
6.4	Deployment	14
7	Q & A	15
8	Known issues	17
9	API Reference	19
9.1	stuffer package	19
10	Indices and tables	29
	Python Module Index	31

Stuffer is a provisioning tool designed to be simple, and to be used in simple scenarios, primarily for provisioning container images.

Documentation is hosted on <http://stuffer.readthedocs.io>. The source code lives at <https://bitbucket.org/mapflat/stuffer>.

CHAPTER 1

Project status

Alpha. Raw but usable.

While stuffer is in alpha stage, i.e. version 0.0.x, the interface may change with any release.

CHAPTER 2

Use cases

Stuffer is primarily intended to be used for provisioning container images, Docker in particular. As a secondary use case, it can be used to provision non-production machines, e.g. developer machines.

More complex provisioning tools, such as Puppet, Chef, and Ansible, are intended for bringing a machine in an arbitrary state to a desired state. This turns out not to be possible in practice, and production machines managed with such tools tend to suffer from deviations from intended state, e.g. outdated dependency packages. At the other end of the complexity scale, good old bash has little support for reuse and for encoding decided best practices. Stuffer provides a middle road, aiming to keep the simplicity of shell commands, augmented with support for sharing constructs between projects.

Stuffer is primarily intended for building a machine from scratch to the desired state. Since the initial state is known, much of the complexity of existing provisioning tools is unnecessary. For example, during an image build, running services need not be considered nor restarted.

CHAPTER 3

Overview

Stuffer is installed through pip3. It is based on Python 3, so plain pip will not work unless Python 3 is default on your platform.

```
sudo pip3 install stuffer
```

Stuffer uses a Python embedded DSL for specifying provisioning directives. It is typically invoked with one or more command arguments on the command line, e.g.:

```
stuffer 'apt.Install("mercurial")'
```

Multiple arguments are concatenated into a multiple line Python recipe:

```
stuffer \  
  'for pkg in "mercurial", "gradle", "python-nose":' \  
  ' print("Installing", pkg)' \  
  ' apt.Install(pkg)'
```

When provisioning Docker containers, stuffer should typically be invoked multiple times, since it allows Docker to use the local cache efficiently:

```
RUN stuffer 'apt.Install("mercurial")' RUN stuffer 'apt.Install("gradle")' RUN stuffer 'apt.Install("python-nose")'
```

Reused recipes can be factored out into Python modules for easier reuse:

```
stuffer 'development.Tools()'
```

In development.py:

```
from stuffer.core import Group  
  
class Tools(Group):  
    def children(self):  
        return [apt.Install(p) for p in "mercurial", "gradle", "python-nose"]
```

It is also possible to create composite actions by explicitly executing other actions. Example from stuffer.contrib.java:

```
class Jdk(Action):
    def __init__(self, version):
        ...

    def run(self):
        apt.AddRepository('ppa:webupd8team/java').execute()
        debconf.SetSelections('debconf', 'shared/accepted-oracle-license-v1-1',
↪ 'select', 'true').execute()
        debconf.SetSelections('debconf', 'shared/accepted-oracle-license-v1-1', 'seen
↪', 'true').execute()
        apt.Install('oracle-java{}-installer'.format(self.version)).execute()
```

Stuffer comes with builtin knowledge of Docker practices, and helps you steer away from common mistakes, and towards best practices. There is not consensus on a single set of Docker best practices, but stuffer provides a means to express your organisation's decided practices in code, instead of educating all developers on the right incantations and rituals.

```
Dockerfile:
FROM phusion/baseimage:0.9.18

RUN stuffer 'docker.Prologue()' # Verifies e.g. that base image is sound.

RUN stuffer 'apt.Install("some-package")'

RUN stuffer 'apt.SourceList("deb http://some.external.repository.com stable non-free
↪")'
RUN stuffer 'apt.Install("other-package")' # Automatically runs 'apt-get update'
↪before 'apt-get install'

RUN stuffer 'docker.Epilogue()' # Cleans temporary files.
```

Design goals

Stuffer design gives priority to:

- Simplicity of use. No knowledge about the tool should be required in order to use it for simple scenarios by copying examples. Some simplicity in the implementation is sacrificed in order to make the usage interface simple. Actions are named similarly to the corresponding shell commands.
- Transparency. Whenever reasonable, actions are translated to shell commands. All actions are logged.
- Ease of reuse. It should be simple to extract commands from snippets and convert them to reusable modules without a rewrite. Therefore, both the DSL and modules are written in Python.
- Docker cache friendliness. Images built with similar commands should be able to share a prefix of commands in order to benefit from Docker image caching.
- No dislike factors. Provisioning tools tend to be loved and/or hated by users, for various reasons. There might be no reason to be passionately enamoured with stuffer, but there should be no reason to have a strong dislike for it, given that you approve of Python and Docker.
- Ease of debugging. Debugging stuffer recipes should be as easy as debugging standard Python programs.
- Avoid reinventing wheels. Use existing Python modules or external tools for tasks that have already been solved. Give priority to reusing existing code over minimising dependencies. In particular, use Python 3 and [click](#) to save boilerplate.

Moreover, the project model is design to facilitate sharing and reuse of code between users, see below.

The DSL is designed to be comprehensible by readers that are not familiar with stuffer. For example, the command `apt.Install("mypack")` runs “`apt-get install mypack`”. There is a balance between convenience and comprehensibility. Stuffer in most cases shuns magic that would create convenience in preference for more understandable code.

The DSL is also designed to make it easy to do things that are correct and work well with containers, and difficult to do things that do not harmonise with containers.

The DSL is designed to be tool friendly (with IntelliJ/PyCharm and pylint in particular), both for writing stuff files and for working on stuffer itself. For example, all imports are explicitly declared in order to make package structure comprehensible for tools.

Python conventions are used for naming, i.e. CamelCase classes and `snake_case` functions.

5.1 Actions

Each desired mutation of a container is represented by an Action. There are Actions for installing packages, changing file contents, setting configuration variables, etc. The different types of actions are represented by different subclasses of Action. Implementations of Action should be idempotent; stuffer will not perform any checks whether the Action is redundant, and each Action specified will be run. Many system administration commands are naturally idempotent, e.g. `apt-get install`. For Actions that are not, the Action implementation needs to include appropriate checks.

Implementations of Action specify what commands to execute by overriding either `Action.command` or `Action.run`.

5.2 Prerequisites

Actions may specify that another Action needs to have been executed before `Action.run` by overriding `Action.prerequisites`. For example, `pip.Install` specifies that the `pip` command must be installed before using it to install other packages. Although the same effect can be achieved by explicitly running the required preparatory steps inside `Action.run`, it is more natural to separate the prerequisites from the command specified by the user. It also allows a potential future version of stuffer to keep track of executed prerequisites and avoid redundant executions.

5.3 Passing state

A container provisioning recipe typically consists of multiple stuffer invocations. The invocations do not share state, except for the container file system. Hence, if you need to pass state between invocations, you will need to save state in the file system.

Stuffer provides a simple key/value store mechanism to pass state between invocations via files in the container file system. Use `store.set_value` to store values, and `store.get` to retrieve them. The naming convention for keys is lower snake case, separated by dots for hierarchical organisation, e.g. `my_corp.databases.mysql.preferred_driver`. The prefix `stuffer.` is reserved for stuffer components, which should use key names corresponding to the stuffer package name, e.g. `stuffer.apk.update_needed`.

The values in the store are plain strings.

6.1 Collaboration model

Users are allowed to put recipes under `sites/` for others to get inspired. This model may not scale, but as long as the number of users is small, there is value in sharing and showing each other code snippets, in order to extract pieces of common value.

Snippets worth reuse can be put under `stuffer/contrib/`. Files under `stuffer/contrib` are expected to be maintained by the contributor.

Routines for installing third-party software should also go under `stuffer/contrib`.

6.2 Contributing

New code should be covered with integration tests. Avoid unit tests - since the purpose of `stuffer` is integration, there is little value testing scenarios that are not authentic. Strive to figure out a way to test functionality with Docker containers.

In order to run the test suite, run `tox` in the project root directory. The continuous integration build also builds the documentation and performs a distribution build. See `shippable.yml` for the exact commands.

When tests pass, fork <https://bitbucket.org/mapflat/stuffer>, push your code to the fork, and create a pull request.

6.3 Build and release

Continuous integration builds are run with `Shippable`. `Shippable` builds a release package for every merge or push to master branch. If the version number is higher than the current version on <https://pypi.python.org>, the CI build uploads a new release. Hence, in order to make a new release, update the version number in `main.py` and `setup.py` before merging to master.

6.4 Deployment

Install the latest version with `pip3 install stuffer`, depending on the default python version in your environment.

In order to create an installable distribution package from the source directory, run `./setup.py sdist` from the project root directory. Install with `pip3 install dist/stuffer-*.tar.gz`.

CHAPTER 7

Q & A

Q: Stuffer sounds similar to [Packer](#). What is the relation?

A: Packer is a tool for creating a container, given that you provide stuff to put in the container. Stuffer is a way to express what stuff to put in a container, given that you provide a way to pack the container. They can be used together, if desired. Packer is made by [Hashicorp](#), who have no relation to Stuffer.

Q: I think that Docker containers should be built according to the following principle: <your preference here>. Why doesn't stuffer do that?

A: There is no single best way to build Docker images. There are tradeoffs involved. Stuffer gives you a way to express your preferences, and package it as code, reusable by your colleagues. Feel free to submit a PR that implements your preferences as an optional strategy.

Q: Does it scale to more complex scenarios? Can I see some examples?

A: You can find some non-trivial examples at <https://bitbucket.org/mapflat/stuffer/src/master/sites/mapflat/>.

CHAPTER 8

Known issues

There is a name clash between the [click command line parser library](#) and a Ubuntu python package for handling the click packaging format. Hence, you might run into trouble if you have the former installed on your machine, or in the Docker images that you wish to build. At this point, you can either solve it by removing the conflicting package, or by installing stuffer in a virtual environment (virtualenv).

<i>stuffer</i>	Stuffer - simple Docker-friendly provisioning.
<i>stuffer.contrib</i>	Package with installers for third-party packages and non-core functionality.

9.1 stuffer package

9.1.1 Subpackages

stuffer.contrib package

Submodules

stuffer.contrib.adobe module

class `stuffer.contrib.adobe.SpotifyClient`
Bases: `stuffer.core.Group`
children () → `typing.List[stuffer.core.Action]`

stuffer.contrib.docker module

class `stuffer.contrib.docker.Docker`
Bases: `stuffer.core.Group`
children ()

stuffer.contrib.dropbox module

```
class stuffer.contrib.dropbox.DropboxClient
    Bases: stuffer.core.Group
    children () → typing.List[stuffer.core.Action]
```

stuffer.contrib.google module

```
class stuffer.contrib.google.ChromeStable
    Bases: stuffer.core.Group
    children () → typing.List[stuffer.core.Action]
```

```
class stuffer.contrib.google.NameServer
    Bases: stuffer.core.Group
    Add a Google nameserver last to resolv.conf (through resolvconf). Not to be used in Docker.
    children () → typing.List[stuffer.core.Action]
```

stuffer.contrib.java module

```
class stuffer.contrib.java.Jdk(version)
    Bases: stuffer.core.Action
    run ()
```

stuffer.contrib.jetbrains module

```
class stuffer.contrib.jetbrains.IntelliJ(version, build, variant='IC', destination='/usr/local')
    Bases: stuffer.core.Action
    path ()
    prerequisites ()
    run ()
```

stuffer.contrib.kafka module

```
class stuffer.contrib.kafka.Install(version, scala_version='2.11', destination='/opt')
    Bases: stuffer.core.Action
    prerequisites ()
    run ()
```

stuffer.contrib.nettools module

```
class stuffer.contrib.nettools.Wireshark
    Bases: stuffer.core.Group
    children () → typing.List[stuffer.core.Action]
```


stuffer.contrib.spotify module

```
class stuffer.contrib.spotify.SpotifyClient
    Bases: stuffer.core.Group
    children () → typing.List[stuffer.core.Action]
```

Module contents

Package with installers for third-party packages and non-core functionality.

9.1.2 Submodules

stuffer.apt module

Package installation with apt commands.

Packages are installed with `apt.Install`. Third-party package repositories can be added with `apt.AddRepository` or `apt.SourceList`. Repository keys are added with either `apt.KeyAdd` or `apt.KeyRecv`.

There is no command corresponding to `apt-get update`, since it is considered bad practice to run and update in isolation. Instead, update is executed automatically only when necessary, i.e. for the first `apt.Install` command, and in case new package repositories or repository keys have been added. The state key `apt.UPDATE_NEEDED_KEY` is used to communicate to later commands whether an update is necessary or not.

Attributes

UPDATE_NEEDED_KEY [str] Name of store variable used to signal the need to run `apt-get update`.

```
class stuffer.apt.AddRepository (name: typing.Union[str, stuffer.content.DeferStr])
    Bases: stuffer.core.Action
    Add an apt repository with apt-add-repository.
```

name Name of repository.

```
prerequisites ()
```

```
run ()
```

```
class stuffer.apt.Install (package: typing.Union[str, stuffer.content.DeferStr, typing.List[str]], update: typing.Union[bool, NoneType] = None)
    Bases: stuffer.core.Action
```

Install a package with `apt-get install`.

`apt-get update` will be executed first, unless the value for `apt.UPDATE_NEEDED_KEY` indicates that update is unnecessary.

package Name of package. Standard `apt-get install` version constraints can be used, e.g. `wget=1.17.1`.

update Whether to run `apt-get update` first. If absent, it is automatically decided.

```
run () → None
```

class `stuffer.apk.KeyAdd` (*url*: `typing.Union[str, stuffer.content.DeferStr]`)
Bases: `stuffer.core.Action`

Add a trusted key to apt using `apt-key add` method.

url URL of key file to download.

prerequisites ()

run ()

class `stuffer.apk.KeyRecv` (*keyserver*: `typing.Union[str, stuffer.content.DeferStr]`, *key*: `typing.Union[str, stuffer.content.DeferStr]`)
Bases: `stuffer.core.Action`

Add a trusted key to apt using `apt-key --recv-keys` method.

keyserver Host name of key server, passed as `--keyserver` argument.

key Key hex code.

run ()

class `stuffer.apk.Purge` (*package*: `typing.Union[str, stuffer.content.DeferStr]`)
Bases: `stuffer.core.Action`

Completely remove a package.

Runs the `apt-get purge` command.

package Name of package.

command ()

class `stuffer.apk.SourceList` (*name*: `typing.Union[str, stuffer.content.DeferStr]`, *contents*: `typing.Union[str, stuffer.content.DeferStr, typing.Callable[[], str]]`)
Bases: `stuffer.core.Action`

Add a package repository by creating a file under `/etc/apt/sources.list.d`.

name Name of repository file, without extension

contents Contents of file, e.g. `deb https://apt.dockerproject.org/repo ubuntu-xenial main`, or function that returns the contents when called.

prerequisites ()

run ()

stuffer.configuration module

Application-wide configuration.

class `stuffer.configuration.config`
Bases: `object`

store_directory = `None`
Configuration parameters.

store_directory Directory where the key/value store is saved. Can be overridden with the command line argument `--store-dir`.

stuffer.content module

Ways to supply (string) content for Actions that create or manipulate files.

class `stuffer.content.DeferStr` (*supplier*)

Bases: `object`

Lazy string, making a string supplier appear as a string.

It is possible to create Actions that provide information at runtime, e.g. an installation path or a version number. `DeferStr` can be used to encapsulate such an Action method, and make it appear as a plain string, in order to pass it as argument to another Action. It will then be evaluated lazily, at runtime.

class `stuffer.content.OutputOf` (*command*)

Bases: `object`

Supply the output of a command, executed inside the container.

command Command to execute. If it is a string, it will be interpreted by the shell.

`stuffer.content.supplier` (*contents: typing.Union[str, stuffer.content.DeferStr, typing.Callable[[], str]]*) → `typing.Callable[[], str]`

Convert an argument to a string supplier, if it is not already.

stuffer.core module

class `stuffer.core.Action`

Bases: `stuffer.utils.NaturalReprMixin`

Base class for actions to be taken.

Subclasses should override either `command()` or `run()`. If `command()` is overridden, it will get logged to stdout. If `run()` is overridden, the implementation should provide some form of logging.

command () → `typing.Union[str, stuffer.content.DeferStr, typing.List[str]]`

Shell command to run. Override this or `run()`.

Should either return a list of strings to pass to `subprocess.check_output`, or a string, in which case `shell=True` will be passed with `subprocess.check_output`.

execute () → `None`

Execute the action, including any prerequisites.

prerequisites ()

run () → `str`

Run the Action command(s).

The default implementation runs the command returned by `command()`.

str The output of the command

subprocess.CalledProcessError On execution failure.

static tmp_dir () → `pathlib.Path`

Directory for temporary file storage, e.g. downloaded files.

class `stuffer.core.ActionRegistry`

Bases: `object`

Singleton class to keep track of the created Actions.

classmethod register (*action*)

classmethod `registered()`

class `stuffer.core.Group`

Bases: `stuffer.core.Action`

Group of multiple actions to be executed.

children() → `typing.List[stuffer.core.Action]`

Returns list of child actions to execute.

run() → `None`

`stuffer.core.run_cmd(cmd: typing.List[str], *args, **kwargs)`

Run a shell command and return the output.

cmd List of command and arguments, passed to `subprocess.check_output`. If a string is passed, `shell=True` will be added to `kwargs`.

args Extra arguments passed to `subprocess.check_output`

kwargs Extra keyword arguments passed to `subprocess.check_output`

The output of the command

subprocess.CalledProcessError On execution failure.

stuffer.debconf module

class `stuffer.debconf.SetSelections` (*section, template, type_, value*)

Bases: `stuffer.core.Action`

Specify debian configuration selections.

The arguments will be passed to `debconf-set-selections`. See the [man page](#) for details.

section Debconf section

template Debconf template

type_ Debconf data type, e.g. `select`, `seen`.

value Selection value

command()

prerequisites()

stuffer.docker module

class `stuffer.docker.Epilogue`

Bases: `stuffer.core.Action`

Clean up the Docker image from temporary files.

Note that although files are removed, the Docker image does not necessarily shrink, since the files are present in lower file system layers.

command()

class `stuffer.docker.Prologue`

Bases: `stuffer.core.Action`

Check that the Docker base image is sound and prepare the image.

run()

stuffer.files module

Actions that change contents of files.

class `stuffer.files.Chmod` (*permissions: int, path: typing.Union[str, stuffer.content.DeferStr]*)

Bases: `stuffer.core.Action`

Set permissions for a file.

permissions Read/write/execute permissions, expressed as a number. For readability, use Python octal numbers, e.g. 0o755.

path The path to the directory or file to change permissions on.

command ()

class `stuffer.files.Chown` (*owner: typing.Union[str, stuffer.content.DeferStr], path: typing.Union[str, stuffer.content.DeferStr], group: typing.Union[str, NoneType] = None, recursive: bool = False*)

Bases: `stuffer.core.Action`

Set ownership for file(s).

owner Username that should own the file(s).

path Path to directory or file to change ownership on.

group Name of group to set on files.

recursive If true, change files in all subdirectories.

command ()

class `stuffer.files.Content` (*path, contents, make_dirs=False*)

Bases: `stuffer.core.Action`

Set the contents of a file.

path Path to file

contents Supplier or contents, or fixed value string. In order to dynamically supply content at image build time, use `content.OutputOf`.

make_dirs If True, create parent directories if necessary.

run ()

class `stuffer.files.DownloadFile` (*url: typing.Union[str, stuffer.content.DeferStr], path: typing.Union[pathlib.Path, str, stuffer.content.DeferStr]*)

Bases: `stuffer.core.Action`

Download and install a single file from a URL.

url URL to retrieve.

path Path of destination file.

run ()

class `stuffer.files.Mkdir` (*path: typing.Union[pathlib.Path, str, stuffer.content.DeferStr]*)

Bases: `stuffer.core.Action`

Create a directory, unless it exists.

path Path of directory to create.

command ()

```
class stuffer.files.Transform(path: typing.Union[pathlib.Path, str], stuffer.content.DeferStr, transform: typing.Callable[[str], str])
```

Bases: *stuffer.core.Action*

Transform the contents of a file by applying a function on the contents.

path Path to file whose contents should be transformed.

transform Function that manipulate file contents and return the new content.

run ()

```
stuffer.files.write_file_atomically(path: typing.Union[pathlib.Path, str], stuffer.content.DeferStr, contents: typing.Union[str, stuffer.content.DeferStr], make_dirs: bool = False, suffix: typing.Union[str, stuffer.content.DeferStr] = '.stuffer_tmp')
```

Write contents to a file in an atomic manner.

This routine prevents corruption in case other processes on the machine read or write the file while it is executed. It is overkill for Docker image building, but provides better safety when stuffer is run on live machines, e.g. developer machines, or when building other types of images, such as AMIs with packer.

path Path of destination file.

contents Contents of file

make_dirs If true, create parent directories if necessary.

suffix Extra suffix added on temporary file.

stuffer.main module

stuffer.pip module

Installing Python packages with pip.

```
class stuffer.pip.Install(package: typing.Union[str, stuffer.content.DeferStr], upgrade: bool = False, bootstrap: bool = True)
```

Bases: *stuffer.core.Action*

Install a package with pip install.

package Name of package. Names may include version specification, e.g. “==1.2.3”, which is passed on to pip install.”

upgrade Whether to upgrade an already installed package, i.e. pass the --upgrade flag.

bootstrap If true (default), ensure that the pip command is installed first.

command ()

prerequisites ()

stuffer.store module

Key/value store for passing state between invocations of stuffer.

```
class stuffer.store.Set(key: typing.Union[str, stuffer.content.DeferStr], value: typing.Union[str, stuffer.content.DeferStr])
```

Bases: *stuffer.core.Action*

Set the key value. Same as set, but implemented as Action.

key Key name.

value Value of key.

run ()

`stuffer.store.get_value(key)`

Retrieve the value of a key.

key Key name.

The value of the key, or None if no key value has been set.

`stuffer.store.set_value(key: str, value: str)`

Set a key to a value.

key Key name.

value Value of key.

stuffer.system module

class `stuffer.system.ShellCommand` (*command: typing.Union[str, stuffer.content.DeferStr]*)

Bases: `stuffer.core.Action`

Run an arbitrary shell command.

command Command to execute. It will be interpreted by the shell, so pipes, redirects, etc are allowed.

command ()

`stuffer.system.real_user` ()

stuffer.user module

Managing users and group membership in the container.

class `stuffer.user.AddToGroup` (*user, group*)

Bases: `stuffer.core.Action`

Add a user to a group.

user Name of user.

group Name of group.

command ()

stuffer.utils module

Unsorted utility classes and routines.

class `stuffer.utils.NaturalReprMixin`

Bases: `object`

Mixin class that provides a `__repr__` implementation calling `natural_object_repr`.

`stuffer.utils.natural_object_repr(obj)`

A string representation of an object, matching the its appearance in code.

`stuffer.utils.natural_repr(obj)`

A string representation of a Python entity, matching the its appearance in code.

`stuffer.utils.str_split` (*cmd*)

Split a string by whitespace, unless it is already a list.

9.1.3 Module contents

Stuffer - simple Docker-friendly provisioning.

CHAPTER 10

Indices and tables

- `genindex`
- `modindex`
- `search`

S

- stuffer, 28
- stuffer.apt, 21
- stuffer.configuration, 22
- stuffer.content, 23
- stuffer.contrib, 21
 - stuffer.contrib.adobe, 19
 - stuffer.contrib.docker, 19
 - stuffer.contrib.dropbox, 20
 - stuffer.contrib.google, 20
 - stuffer.contrib.java, 20
 - stuffer.contrib.jetbrains, 20
 - stuffer.contrib.kafka, 20
 - stuffer.contrib.nettools, 20
 - stuffer.contrib.spotify, 21
- stuffer.core, 23
- stuffer.debconf, 24
- stuffer.docker, 24
- stuffer.files, 25
- stuffer.pip, 26
- stuffer.store, 26
- stuffer.system, 27
- stuffer.user, 27
- stuffer.utils, 27

A

Action (class in `stuffer.core`), 23
ActionRegistry (class in `stuffer.core`), 23
AddRepository (class in `stuffer.apt`), 21
AddToGroup (class in `stuffer.user`), 27

C

children() (`stuffer.contrib.adobe.SpotifyClient` method), 19
children() (`stuffer.contrib.docker.Docker` method), 19
children() (`stuffer.contrib.dropbox.DropboxClient` method), 20
children() (`stuffer.contrib.google.ChromeStable` method), 20
children() (`stuffer.contrib.google.NameServer` method), 20
children() (`stuffer.contrib.nettools.Wireshark` method), 20
children() (`stuffer.contrib.spotify.SpotifyClient` method), 21
children() (`stuffer.core.Group` method), 24
Chmod (class in `stuffer.files`), 25
Chown (class in `stuffer.files`), 25
ChromeStable (class in `stuffer.contrib.google`), 20
command() (`stuffer.apt.Purge` method), 22
command() (`stuffer.core.Action` method), 23
command() (`stuffer.debconf.SetSelections` method), 24
command() (`stuffer.docker.Epilogue` method), 24
command() (`stuffer.files.Chmod` method), 25
command() (`stuffer.files.Chown` method), 25
command() (`stuffer.files.Mkdir` method), 25
command() (`stuffer.pip.Install` method), 26
command() (`stuffer.system.ShellCommand` method), 27
command() (`stuffer.user.AddToGroup` method), 27
config (class in `stuffer.configuration`), 22
Content (class in `stuffer.files`), 25

D

DeferStr (class in `stuffer.content`), 23
Docker (class in `stuffer.contrib.docker`), 19

DownloadFile (class in `stuffer.files`), 25
DropboxClient (class in `stuffer.contrib.dropbox`), 20

E

Epilogue (class in `stuffer.docker`), 24
execute() (`stuffer.core.Action` method), 23

G

get_value() (in module `stuffer.store`), 27
Group (class in `stuffer.core`), 24

I

Install (class in `stuffer.apt`), 21
Install (class in `stuffer.contrib.kafka`), 20
Install (class in `stuffer.pip`), 26
IntelliJ (class in `stuffer.contrib.jetbrains`), 20

J

Jdk (class in `stuffer.contrib.java`), 20

K

KeyAdd (class in `stuffer.apt`), 21
KeyRecv (class in `stuffer.apt`), 22

M

Mkdir (class in `stuffer.files`), 25

N

NameServer (class in `stuffer.contrib.google`), 20
natural_object_repr() (in module `stuffer.utils`), 27
natural_repr() (in module `stuffer.utils`), 27
NaturalReprMixin (class in `stuffer.utils`), 27

O

OutputOf (class in `stuffer.content`), 23

P

path() (`stuffer.contrib.jetbrains.IntelliJ` method), 20

prerequisites() (stuffer.apt.AddRepository method), 21
prerequisites() (stuffer.apt.KeyAdd method), 22
prerequisites() (stuffer.apt.SourceList method), 22
prerequisites() (stuffer.contrib.jetbrains.IntelliJ method), 20
prerequisites() (stuffer.contrib.kafka.Install method), 20
prerequisites() (stuffer.core.Action method), 23
prerequisites() (stuffer.debconf.SetSelections method), 24
prerequisites() (stuffer.pip.Install method), 26
Prologue (class in stuffer.docker), 24
Purge (class in stuffer.apt), 22

R

real_user() (in module stuffer.system), 27
register() (stuffer.core.ActionRegistry class method), 23
registered() (stuffer.core.ActionRegistry class method), 23
run() (stuffer.apt.AddRepository method), 21
run() (stuffer.apt.Install method), 21
run() (stuffer.apt.KeyAdd method), 22
run() (stuffer.apt.KeyRecv method), 22
run() (stuffer.apt.SourceList method), 22
run() (stuffer.contrib.java.Jdk method), 20
run() (stuffer.contrib.jetbrains.IntelliJ method), 20
run() (stuffer.contrib.kafka.Install method), 20
run() (stuffer.core.Action method), 23
run() (stuffer.core.Group method), 24
run() (stuffer.docker.Prologue method), 24
run() (stuffer.files.Content method), 25
run() (stuffer.files.DownloadFile method), 25
run() (stuffer.files.Transform method), 26
run() (stuffer.store.Set method), 27
run_cmd() (in module stuffer.core), 24

S

Set (class in stuffer.store), 26
set_value() (in module stuffer.store), 27
SetSelections (class in stuffer.debconf), 24
ShellCommand (class in stuffer.system), 27
SourceList (class in stuffer.apt), 22
SpotifyClient (class in stuffer.contrib.adobe), 19
SpotifyClient (class in stuffer.contrib.spotify), 21
store_directory (stuffer.configuration.config attribute), 22
str_split() (in module stuffer.utils), 27
stuffer (module), 28
stuffer.apt (module), 21
stuffer.configuration (module), 22
stuffer.content (module), 23
stuffer.contrib (module), 21
stuffer.contrib.adobe (module), 19
stuffer.contrib.docker (module), 19
stuffer.contrib.dropbox (module), 20
stuffer.contrib.google (module), 20
stuffer.contrib.java (module), 20

stuffer.contrib.jetbrains (module), 20
stuffer.contrib.kafka (module), 20
stuffer.contrib.nettools (module), 20
stuffer.contrib.spotify (module), 21
stuffer.core (module), 23
stuffer.debconf (module), 24
stuffer.docker (module), 24
stuffer.files (module), 25
stuffer.pip (module), 26
stuffer.store (module), 26
stuffer.system (module), 27
stuffer.user (module), 27
stuffer.utils (module), 27
supplier() (in module stuffer.content), 23

T

tmp_dir() (stuffer.core.Action static method), 23
Transform (class in stuffer.files), 25

W

Wireshark (class in stuffer.contrib.nettools), 20
write_file_atomically() (in module stuffer.files), 26