stuffer Documentation

Release 0.0.10

Lars Albertsson

Jan 06, 2023

Contents:

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 provisioing 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in practice, and production machines manages with such tools tend to suffer from deviations from intended state, e.g. outdated transitive dependency packages.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 apt-get update && apt-get install -y python3-pip
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):
```

(continues on next page)

(continued from previous page)

```
  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.docker:

```
class Prologue(Action):
    """Check that the Docker base image is sound and prepare the image."""

    def run(self):
        # Check that the image is ok
        ...
        # Always needed, or apt install complains.
        apt.Install(['apt-utils']).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.
```

CHAPTER 4

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.

CHAPTER 5

DSL

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 overrinding 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.apt.update_needed`.

The values in the store are plain strings.

CHAPTER 6

Developing stuffer

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 bulds 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).

CHAPTER 9

API Reference

stuffer	Stuffer - simple Docker-friendly provisioning.
stuffer.contrib	

9.1 stuffer package

9.1.1 Subpackages

stuffer.contrib package

Submodules

stuffer.contrib.docker module

stuffer.contrib.dropbox module

stuffer.contrib.google module

stuffer.contrib.hashicorp module

stuffer.contrib.kafka module

stuffer.contrib.nettools module

stuffer.contrib.spotify module

Module contents

9.1.2 Submodules

stuffer.apt module

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: Union[str, stuffer.content.DeferStr, Callable[str]]*) → 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() → Union[str, stuffer.content.DeferStr, 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() → List[stuffer.core.Action]
Returns list of child actions to execute.

run() → None
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.

stuffer.core.**run_cmd**(*cmd: 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

stuffer.docker module

stuffer.files module

Actions that change contents of files.

class stuffer.files.**Chmod**(*permissions: int, path: 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()
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..

class stuffer.files.**Chown**(*owner: Union[str, stuffer.content.DeferStr], path: Union[str, stuffer.content.DeferStr], group: Optional[str] = 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()
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..

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()
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.

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

class stuffer.files.**Mkdir**(*path: Union[pathlib.Path, str, stuffer.content.DeferStr]*)
Bases: *stuffer.core.Action*

Create a directory, unless it exists.

path Path of directory to create.

command()
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..

class stuffer.files.**SysctlConf**(*name: str, key: str, value: Union[int, str]*)
Bases: *stuffer.files.Content*

Set sysctl parameter in /etc/sysctl.d.

name Name of file, without .conf suffix

key Sysctl key to set

value Value of key

run()
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.

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

stuffer.files.**write_file_atomically**(*path: Union[pathlib.Path, str, stuffer.content.DeferStr], contents: Union[str, stuffer.content.DeferStr], make_dirs: bool = False, suffix: 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

stuffer.shell module

class stuffer.shell.**ShellCommand**(*command: 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()

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..

stuffer.snap module

Package installation with snap commands.

Packages are installed with snap.Install.

class stuffer.snap.**Install**(*package: Union[str, stuffer.content.DeferStr], classic: bool = False*)

Bases: *stuffer.core.Action*

Install a package with snap install.

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

run() → None

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.

stuffer.store module

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

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

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.

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.**Distribution**

Bases: `object`

Information about the system distribution.

classmethod codename()

classmethod release()

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()
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..

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

Python Module Index

S

Index

T

W