
Medikit Documentation

Release 0.8.0

Romain Dorgueil

Nov 25, 2020

Contents

1	Documentation	3
1.1	Quick Start	3
1.2	Guides	4
1.3	Commands Reference	4
1.4	Features Reference	4
2	Indices and tables	25
	Python Module Index	27
	Index	29

Todo: elevator pitch here

1.1 Quick Start

Medikit is a release engineering tool provided as an eventual dependency, which means that using it won't make it a dependency of your project. You just need to install it to update the project assets, that will be statically updated and committed within your source control system.

Users and developpers that don't need to update the project assets wont need it, and won't even need to know it was used. This also means that you can use it for a while, and stop using it if you think it does not suit your needs anymore. Or you can easily add its usage to an existing project.

1.1.1 Installation

Use the pip, luke:

```
$ pip install medikit
```

1.1.2 First Steps

Bootstrap a project:

```
$ medikit init myproject
$ cd myproject
$ make install
```

Edit the configuration:

```
$ vim Projectfile
```

Update a project's assets:

```
$ medikit update
```

... or ...

```
$ make update
```

To regenerate “requirements”, use:

```
$ make update-requirements
```

1.2 Guides

This is a work in progress, and I’m afraid this section lacks content.

Meanwhile, you can read the *Features Reference*.

1.3 Commands Reference

1.3.1 Init

Creates a project (bootstraps the Projectfile, then run an update).

```
$ medikit init <projectname>
```

1.3.2 Update

Updates a project, according to Projectfile.

```
$ medikit update
```

1.3.3 Pipeline (alpha)

Starts, or continue, a project management pipeline.

```
$ medikit pipeline release start
```

If the pipeline already started, you can resume it:

```
$ medikit pipeline release continue
```

1.4 Features Reference

Features are the base building blocks of Medikit. In each project, you can “require” a bunch of features, it will tell medikit what to handle.

Each feature works the same, giving you a “configuration” object when you require it. You can use this configuration object to change the default behaviour of Medikit.

For example, to add a requirement to the python feature, write in your Projectfile:

```
from medikit import require

python = require('python')

python.add_requirements('requests >=2,<3')
```

For custom behaviors that goes further than just changing feature configurations, you can listen to a number of events exposed by the features.

Here is an example:

```
from medikit import require, listen

make = require('make')

@listen(make.on_generate)
def on_make_generate(event):
    event.makefile.add_target(
        'hello',
        'echo "Hello world"',
        phony=True
    )
```

1.4.1 Django Feature

The «django» feature adds the django framework to your project.

Usage

To use the Django Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

django = require('django')
```

The *django* handle is a *DjangoConfig* instance, and can be used to customize the feature.

This will add a few items to your Makefile, and ensure a minimalistic django project structure is available.

By default, it will use Django ~2.0,<2.2, but you can tune that:

```
django.version = '~2.0.3'
```

This feature will also add or extend a “prod” python extra, that will install gunicorn.

Configuration

class medikit.feature.django.DjangoConfig

Configuration class for the “django” feature.

static_dir

The django global static directory (property).

```
version = '~2.0,<2.2'
```

Which django version requirement do you want?

Implementation

```
class medikit.feature.django.DjangoFeature(dispatcher)
```

Config

alias of *DjangoConfig*

on_make_generate (*event*)

Listens to `medikit.feature.make.on_generate` event (*priority: -20*)

on_python_generate (*event*)

Listens to `medikit.feature.python.on_generate` event (*priority: 0*)

on_start (*event*)

Listens to `medikit.on_start` event (*priority: 0*)

requires = {'python'}

1.4.2 Docker Feature

Adds docker capabilities to your package, using either “docker build” or “rocker build” to create an image containing your code, in a fully functional python virtualenv.

Usage

To use the Docker Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

docker = require('docker')
```

The *docker* handle is a *DockerConfig* instance, and can be used to customize the feature.

This feature is brand new and should be used with care.

You’ll get a few make targets out of this, and a Dockerfile (or Rockerfile).

Build

Building an image is as simple as running:

```
$ make docker-build
```

This will use the root Dockerfile (or Rockerfile, see builders below) and build an image named after your package.

Push

You can push the built image to a docker registry, using:

```
$ make docker-push
```

Run

You can run the default entrypoint / command in a new container using the built image:

```
$ make docker-run
```

Shell

You can run a bash shell in a new container using the built image:

```
$ make docker-shell
```

Custom builder

You can change the default docker builder (a.k.a “docker build”) to use rocker (see <https://github.com/grammarly/rocker>).

```
docker.use_rocker_builder()
```

Custom image name or registry

If you want to customize the image name, or the target registry (for example if you want to use Amazon Elastic Container Registry, Google Container Registry, Quay, or even a private registry you’re crazy enough to host yourself):

```
# only override the registry
docker.set_remote(registry='eu.gcr.io')

# override the registry and username, but keep the default image name
docker.set_remote(registry='eu.gcr.io', user='sergey')

# override the image name only
docker.set_remote(name='acme')
```

Docker Compose

The feature will also create an example docker-compose.yml file.

If you don’t want this, you can:

```
docker.compose_file = None
```

Or if you want to override its name:

```
docker.compose_file = 'config/docker/compose.yml'
```

Please note that this file will only contain a structure skeleton, with no service defined. This is up to you to fill, although we may work on this in the future as an opt-in managed file.

Configuration

```
class medikit.feature.docker.DockerConfig

    build_file
    compose_file
    disable_builder()
    image
    set_remote(registry=None, user=None, name='$(shell echo $(PACKAGE) | tr A-Z a-z)')
    use_default_builder()
    use_rocker_builder()
    variables
```

Implementation

```
class medikit.feature.docker.DockerFeature(dispatcher)

    Config
        alias of DockerConfig

    on_end(event)
        Listens to medikit.on_end event (priority: 0)

    on_make_generate(event)
        Listens to medikit.feature.make.on_generate event (priority: -1)
```

1.4.3 Format Feature

Code formatting, using third party tools.

Superseeds “yapf” feature that was only using one tool for one language.

```
$ make format
```

Usage

To use the Format Feature, make sure your **Projectfile** contains the following:

```
from medikit import require
format = require('format')
```

The *format* handle is a *FormatConfig* instance, and can be used to customize the feature.

Configuration

```
class medikit.feature.format.FormatConfig
```

```
    active_tools
    all_tools = {'black', 'isort', 'prettier', 'yapf'}
    default_tools = {'black', 'isort'}
    javascript_tools = {'prettier'}
    python_tools = {'black', 'isort', 'yapf'}
    using (*tools)
```

Choose which tool to use when formatting the codebase.

Example:

```
require("format").using("yapf", "isort")
```

Note that the above is also the default, so using `require("format")` is enough to have the same effect.

Implementation

```
class medikit.feature.format.FormatFeature(dispatcher)
```

Config

alias of `FormatConfig`

conflicts = {'yapf'}

on_before_start (event)

Listens to `medikit.on_start` event (priority: -101)

on_make_generate (event)

Listens to `medikit.feature.make.on_generate` event (priority: -20)

on_python_generate (event)

Listens to `medikit.feature.python.on_generate` event (priority: 0)

on_start (event)

Listens to `medikit.on_start` event (priority: -20)

1.4.4 Git Feature

Git version control system support.

Usage

The Git Feature feature is required, and enabled by default.

To get a handle to the `GitConfig` instance, you can:

```
from medikit import require

git = require('git')
```

Currently, **this feature is required for medikit to work.**

To disable it, use:

```
git.disable()
```

It will avoid creating a *.git* repository, and won't *git add* the modified files to git neither.

Even disabled, the feature will still manage the *.gitignore* file (can't harm) and the *VERSION* variable, still based on *git describe* value. It means that you can ask medikit to not create a repo, but it should still be in a sub-directory or a git repository somewhere.

Configuration

```
class medikit.feature.git.GitConfig
```

```
    disable()
    enable()
    enabled
```

Implementation

```
class medikit.feature.git.GitFeature(dispatcher)
```

```
    Config
        alias of GitConfig

    on_end(event)
        Listens to medikit.on_end event (priority: 0)

    on_make_generate(event)
        Listens to medikit.feature.make.on_generate event (priority: -99)

    on_start(event)
        Listens to medikit.on_start event (priority: -100)
```

1.4.5 Kube Feature

Setup make targets to rollout and rollback this project as a deployment onto a Kubernetes cluster.

Usage

To use the Kube Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

kube = require('kube')
```

The *kube* handle is a *KubeConfig* instance, and can be used to customize the feature.

Warning: This feature is brand new and should be used with care.

Todo: Write the docs, once the feature stabilize.

Configuration

```
class medikit.feature.kube.KubeConfig
```

```
    add_target (name, variant=None, *, patch, patch_path="")
    disable_helm()
    enable_helm()
    get_targets (variant=None)
    get_variants()
    use_helm
```

Implementation

```
class medikit.feature.kube.KubeFeature (dispatcher)
```

```
    Config
        alias of KubeConfig
    on_make_generate (event)
        Listens to medikit.feature.make.on_generate event (priority: -1)
    requires = {'docker'}
```

1.4.6 Make Feature

GNU Make support.

Usage

The Make Feature feature is required, and enabled by default.

To get a handle to the MakeConfig instance, you can:

```
from medikit import require

make = require('make')
```

Currently, **this feature is required for medikit to work.**

Makefile generation and management is quite central to medikit, and it's one of the strongest opinionated choices made by the tool.

Note: *Makefile* will be overridden on each *medikit update* run! See below how to customize it.

Everything out of the project’s world is managed by a single, central Makefile, that contains all the external entrypoints to your package.

By default, it only contains the following targets (a.k.a “tasks”):

- `install`
- `install-dev`
- `update`
- `update-requirements`

This is highly extendable, and about all other features will add their own targets using listeners to the `MakeConfig.on_generate` event.

Default targets

Install

The *make install* command will try its best to install your package in your current system environment. For python projects, it work with the system python, but you’re highly encouraged to use a virtual environment (using `virtualenv` or `venv` module).

```
$ make install
```

Install Dev

The *make install-dev* command works like *make install*, but adds the *dev* extra.

The *dev* extra is a convention medikit takes to group all dependencies that are only required to actual hack on your project, and that won’t be necessary in production / runtime environments.

For python projects, it maps to an “extra”, as defined by `setuptools`. For Node.js projects, it will use the “devDependencies”.

```
$ make install-dev
```

Update

This is a shortcut to *medikit update*, with a preliminary dependency check on *medikit*.

As you may have noticed, *medikit* is never added as a dependency to your project, so this task will ensure it’s installed before running.

```
$ make update
```


Update Requirements

The `make update-requirements` command works like `make update`, but forces the regeneration of `requirements*.txt` files.

For security reasons, *medikit* never updates your requirements if they are already frozen in requirements files (you would not want a requirement to increment version without notice).

This task is here so you can explicitly update your requirements frozen versions, according to the constraints you defined in the *Projectfile*.

```
$ make update-requirements
```

Customize your Makefile

To customize the generated *Makefile*, you can use the same event mechanism that is used by *medikit* features, directly from within your *Projectfile*.

Add a target

```
from medikit import listen

@listen(make.on_generate)
def on_make_generate(event):
    event.makefile.add_target('foo', '''
        echo "Foo!"
    ''', deps=('install', ), phony=True, doc='So foo...')
    )
```

This is pretty self-explanatory, but let's detail:

- * "foo" is the target name (you'll be able to run ``make foo``)
- * This target will run ``echo "Foo!"``
- * It depends on the ``install`` target, that needs to be satisfied (install being "phony" `→`, it will be run every time).
- * This task is "phony", meaning that there will be no ``foo`` file or directory `→` generated as the output, and thus that ``make`` should consider it's never outdated.
- * If you create non phony targets, they must result in a matching file or directory `→` created.
- * Read more about GNU Make: <https://www.gnu.org/software/make/>

Change the dependencies of an existing target

```
from medikit import listen

@listen(make.on_generate)
def on_make_generate(event):
    event.makefile.set_seps('foo', ('install-dev', ))
```

Add (or override) a variable

```
from medikit import listen

@listen(make.on_generate)
def on_make_generate(event):
    event.makefile['FOO'] = 'Bar'
```

The user can override *Makefile* variables using your system environment:

```
$ FOO=loremipsum make foo
```

To avoid this default behaviour (which is more than ok most of the time), you can change the assignment operator used in the makefile.

```
from medikit import listen

@listen(make.on_generate)
def on_make_generate(event):
    event.makefile.set_assignment_operator('FOO', ':=')
```

This is an advanced feature you'll probably never need. You can [read the make variables reference](#).

Implementation

```
class medikit.feature.make.MakeFeature(dispatcher)
```

Config

alias of `medikit.feature.make.config.MakeConfig`

add_includes (*includes*)

add_medikit_targets (*config*)

configure ()

on_start (*event*)

Parameters *event* (*ProjectEvent*) –

Listens to `medikit.on_start` event (*priority: -80*)

1.4.7 NodeJS Feature

NodeJS / Yarn support.

This feature is experimental and as though it may work for you, that's not a guarantee. Please use with care.

Usage

To use the NodeJS Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

nodejs = require('nodejs')
```

The *nodejs* handle is a *NodeJSConfig* instance, and can be used to customize the feature.

Configuration

```
class medikit.feature.nodejs.NodeJSConfig
    Configuration API for the «nodejs» feature.

    add_dependencies (deps=None, **kwargs)

    get_dependencies ()

    setup (*, base_dir=None)
```

Implementation

```
class medikit.feature.nodejs.NodeJSFeature (dispatcher)

    Config
        alias of NodeJSConfig

    on_end (event)
        Listens to medikit.on_end event (priority: 100)

    on_make_generate (event)
        Listens to medikit.feature.make.on_generate event (priority: 0)

    on_start (event)
        Listens to medikit.on_start event (priority: 0)

    requires = {'make', 'python'}
```

1.4.8 Pylint Feature

Lint your python code with pylint.

This feature may be outdated.

Usage

To use the Pylint Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

pylint = require('pylint')
```

The *pylint* handle is a *Config* instance, and can be used to customize the feature.

Implementation

```
class medikit.feature.pylint.PylintFeature (dispatcher)

    on_make_generate (event)
        Listens to medikit.feature.make.on_generate event (priority: -20)
```

```
on_python_generate(event)
    Listens to medikit.feature.python.on_generate event (priority: 0)

requires = {'python'}
```

1.4.9 Pytest Feature

Adds the pytest testing framework to your project.

Usage

To use the Pytest Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

pytest = require('pytest')
```

The *pytest* handle is a *PytestConfig* instance, and can be used to customize the feature.

Configuration

```
class medikit.feature.pytest.PytestConfig
```

```
    addons = {'coverage': '~=4.5', 'pytest-cov': '~=2.7'}
        Additionnal packages to use in dev requirements along with the main pytest packe. You can override this
        dictionary.

    set_version(version)
        Overrides Pytest version requirement with your own.

    version = '~=4.6'
        Pytest version to use in dev requirements. You can override this using set_version(...).
```

Implementation

```
class medikit.feature.pytest.PytestFeature(dispatcher)
```

```
    Config
        alias of PytestConfig

    on_make_generate(event)
        Listens to medikit.feature.make.on_generate event (priority: -20)

    on_python_generate(event)
        Listens to medikit.feature.python.on_generate event (priority: 0)

    on_start(event)
        Listens to medikit.on_start event (priority: -20)

    requires = {'python'}
```

1.4.10 Python Feature

The “python” feature is the base feature required for all python projects.

Medikit only supports python 3.5+ projects, which we believe is a future proof choice.

Overview

```
from medikit import require

# load python feature (idempotent).
python = require('python')

# configure our package.
python.setup(
    name = 'awesome-library',
    author = 'Chuck Norris',
)

# add base and extras requirements, with "loose" versionning (the frozen version fits
↳ better in requirements*.txt)
python.add_requirements(
    'django',
    dev=[
        'pytest',
    ],
)
```

Usage

To use the Python Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

python = require('python')
```

The *python* handle is a *PythonConfig* instance, and can be used to customize the feature.

The python features makes your package real (at least if it uses python). Medikit was written originally for python, and although it’s not completely true anymore, a lot of features depends on this.

Setup

You can define the setuptools’ *setup(...)* arguments using *python.setup(...)*:

```
python.setup(
    name='medikit',
    description='Opinionated python 3.5+ project management.',
    license='Apache License, Version 2.0',
    url='https://github.com/python-medikit/medikit',
    download_url='https://github.com/python-medikit/medikit/tarball/{version}',
    author='Romain Dorgueil',
    author_email='romain@dorgueil.net',
    entry_points={
```

(continues on next page)

(continued from previous page)

```
        'console_scripts': ['medikit=medikit.__main__:main'],
    }
)
```

This is required for any python package.

Requirements

Requirements are managed using two different mechanisms:

- The *setup.py* file, autogenerated and overridden by *medikit*, will contain the “loose” requirements as you define them. You’re encouraged to use “~x.y.z” or “~x.y” versions. You should use each versions (“==x.y.z”) only in case you’re relying on a package you never want to update.
- The *requirements*.txt* files will contain frozen version numbers. Those requirements will be committed, and you can ensure the reproducibility of your installs by using *pip install -r requirements.txt* instead of *python setup.py install*.

In *medikit*, we call what is present in *setup.py* “constraints”, and what is in *requirements*.txt* files “requirements”.

Let’s see how we can set them:

```
python.add_requirements(
    "requests ~2.18"
)
```

This will set a constraint on any semver compatible requests version, and update the requirement to latest requests version, compatible with the constraint (as of writing, 2.18.4).

It means that if you run *make install*, *python setup.py install* or *pip install -e .*, requests will only be downloaded and installed if there is no installation complying to the constraint in your current env. This is very handy if you have local, editable packages that you want to use instead of PyPI versions.

It also means that when you run *pip install -r requirements.txt*, you’ll get requests 2.18.4 even if a new version was released.

If you want to upgrade to the new released version, use *make update-requirements*, review the git changes (*git diff -cached*), test your software with the new version and eventually (git) commit to this dependency update.

Constraints

Sometimes, you want a dependency to only be a constraint, and not a frozen requirement.

```
python.add_constraints(
    "certifi ~2018,<2019"
)
```

This will ensure that your env contains “certifi”, a version released in the 2018 year, but also says you don’t care which one.

This is an advanced feature that you should only use if you really know what you’re doing, otherwise, use a requirement (reproducibility of installs is gold).

Extras

You can create as much “extras” as you want.

As a default, medikit will create a “dev” extra, but you can add whatever you need:

```
python.add_requirements (
    sql=[
        'sqlalchemy ~=1.2.5',
    ]
)
```

The same works with constraints, of course.

Changing package generation behaviour

Medikit creates the necessary directory structure for your package, named after your package name defined in the *python.setup()* call.

If you don’t want medikit to create this directory structure:

```
python.create_packages = False
```

Medikit also considers you’ll need a version number tracking mechanism for your project. It creates a *_version.py* file in your package’s root directory. To override this file’s name:

```
python.version_file = 'my_version.py'
```

Configuration

```
class medikit.feature.python.PythonConfig
    Configuration API for the «python» feature.

    add_constraints (*reqs, **kwargs)
    add_inline_requirements (*reqs, **kwargs)
    add_requirements (*reqs, **kwargs)
    add_vendors (*reqs, **kwargs)
    create_packages
    get (item)
    get_constraints (extra=None)
    get_extras ()
    get_init_files ()
    get_inline_requirements (extra=None)
    get_requirements (extra=None, with_constraints=False)
    get_setup ()
    get_vendors (extra=None)
    on_generate = 'medikit.feature.python.on_generate'
```

```
package_dir
setup (**kwargs)
version_file
```

Implementation

class medikit.feature.python.**PythonFeature** (*dispatcher*)

Adds the requirements/requirements-dev logic, virtualenv support, setup.py generation, a few metadata files used by setuptools...

Config

alias of *PythonConfig*

on_end (*event*)

Listens to medikit.on_end event (*priority: -100*)

on_make_generate (*event*)

Environment variables

- PACKAGE
- PYTHON
- PYTHON_BASENAME
- PYTHON_DIR
- PYTHON_REQUIREMENTS_FILE
- PYTHON_REQUIREMENTS_DEV_FILE

Shortcuts - PIP - PIP_INSTALL_OPTIONS

Make targets

- install
- install-dev

Parameters **event** (*MakefileEvent*) -

Listens to medikit.feature.make.on_generate event (*priority: -100*)

on_start (*event*)

Events

- medikit.feature.python.on_generate (with the same ProjectEvent we got, todo: why not deprecate it in favor of higher priority medikit.on_start?)

Files

- <yourpackage>/__init__.py
- MANIFEST.in
- README.rst
- classifiers.txt
- requirements-dev.txt
- requirements.txt
- setup.cfg

- `setup.py` (overwritten)

Parameters `event` (*ProjectEvent*) –

Listens to `medikit.on_start` event (*priority: -100*)

requires = {'make'}

1.4.11 Sphinx Feature

Usage

To use the Sphinx Feature, make sure your **Projectfile** contains the following:

```
from medikit import require
sphinx = require('sphinx')
```

The *sphinx* handle is a *SphinxConfig* instance, and can be used to customize the feature.

You can customize the theme:

```
sphinx.theme = 'sphinx_rtd_theme'
```

Note that this should be a parsable requirement, and it won't be added to your docs/conf.py automatically.

This feature will add the necessary requirements to the python feature (sphinx mostly, and eventually your theme requirement) and setup the correct makefile task to build the html docs. Note that it won't bootstrap your sphinx config file, and you still need to run the following if your documentation does not exist yet:

```
$ make update-requirements
$ make install-dev
$ mkdir docs
$ cd docs
$ sphinx-quickstart .
```

Then, eventually tune the configuration.

Note: In the future, we may consider generating it for you if it does not exist, but it's not a priority.

Configuration

class `medikit.feature.sphinx.SphinxConfig`

get_theme ()

Gets the theme. Prefer using the `.theme` property.

Returns parsed requirement

set_theme (*theme*)

Sets the theme. Prefer using the `.theme` property.

Parameters `theme` – Requirement for theme

theme

Sphinx theme to use, that should be parsable as a requirement.

Implementation

```
class medikit.feature.sphinx.SphinxFeature (dispatcher)
```

Config

alias of *SphinxConfig*

on_make_generate (*event*)

Listens to `medikit.feature.make.on_generate` event (*priority: -20*)

on_python_generate (*event*)

Listens to `medikit.feature.python.on_generate` event (*priority: 0*)

1.4.12 Webpack Feature

Webpack support.

This feature is experimental and as though it may work for you, that's not a guarantee. Please use with care.

Usage

To use the Webpack Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

webpack = require('webpack')
```

The *webpack* handle is a *Config* instance, and can be used to customize the feature.

Implementation

```
class medikit.feature.webpack.WebpackFeature (dispatcher)
```

on_make_generate (*event*)

Listens to `medikit.feature.make.on_generate` event (*priority: 0*)

requires = {'nodejs'}

1.4.13 Yapf Feature

YAPF support, to automatically reformat all your (python) source code.

```
$ make format
```

Usage

To use the Yapf Feature, make sure your **Projectfile** contains the following:

```
from medikit import require

yapf = require('yapf')
```

The *yapf* handle is a `Config` instance, and can be used to customize the feature.

Implementation

```
class medikit.feature.yapf.YapfFeature(dispatcher)

    conflicts = {'format'}

    on_before_start(event)
        Listens to medikit.on_start event (priority: -101)

    on_make_generate(event)
        Listens to medikit.feature.make.on_generate event (priority: -20)

    on_python_generate(event)
        Listens to medikit.feature.python.on_generate event (priority: 0)

    on_start(event)
        Listens to medikit.on_start event (priority: -20)

    requires = {'python'}
```


CHAPTER 2

Indices and tables

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

m

- `medikit.feature.django`, [5](#)
- `medikit.feature.docker`, [6](#)
- `medikit.feature.format`, [8](#)
- `medikit.feature.git`, [9](#)
- `medikit.feature.kube`, [10](#)
- `medikit.feature.make`, [11](#)
- `medikit.feature.nodejs`, [14](#)
- `medikit.feature.pylint`, [15](#)
- `medikit.feature.pytest`, [16](#)
- `medikit.feature.python`, [17](#)
- `medikit.feature.sphinx`, [21](#)
- `medikit.feature.webpack`, [22](#)
- `medikit.feature.yapf`, [22](#)

A

active_tools (*medikit.feature.format.FormatConfig attribute*), 9

add_constraints() (*medikit.feature.python.PythonConfig method*), 19

add_dependencies() (*medikit.feature.nodejs.NodeJSConfig method*), 15

add_includes() (*medikit.feature.make.MakeFeature method*), 14

add_inline_requirements() (*medikit.feature.python.PythonConfig method*), 19

add_medikit_targets() (*medikit.feature.make.MakeFeature method*), 14

add_requirements() (*medikit.feature.python.PythonConfig method*), 19

add_target() (*medikit.feature.kube.KubeConfig method*), 11

add_vendors() (*medikit.feature.python.PythonConfig method*), 19

addons (*medikit.feature.pytest.PytestConfig attribute*), 16

all_tools (*medikit.feature.format.FormatConfig attribute*), 9

B

build_file (*medikit.feature.docker.DockerConfig attribute*), 8

C

compose_file (*medikit.feature.docker.DockerConfig attribute*), 8

Config (*medikit.feature.django.DjangoFeature attribute*), 6

Config (*medikit.feature.docker.DockerFeature attribute*), 8

Config (*medikit.feature.format.FormatFeature attribute*), 9

Config (*medikit.feature.git.GitFeature attribute*), 10

Config (*medikit.feature.kube.KubeFeature attribute*), 11

Config (*medikit.feature.make.MakeFeature attribute*), 14

Config (*medikit.feature.nodejs.NodeJSFeature attribute*), 15

Config (*medikit.feature.pytest.PytestFeature attribute*), 16

Config (*medikit.feature.python.PythonFeature attribute*), 20

Config (*medikit.feature.sphinx.SphinxFeature attribute*), 22

configure() (*medikit.feature.make.MakeFeature method*), 14

conflicts (*medikit.feature.format.FormatFeature attribute*), 9

conflicts (*medikit.feature.yapf.YapfFeature attribute*), 23

create_packages (*medikit.feature.python.PythonConfig attribute*), 19

D

default_tools (*medikit.feature.format.FormatConfig attribute*), 9

disable() (*medikit.feature.git.GitConfig method*), 10

disable_builder() (*medikit.feature.docker.DockerConfig method*), 8

disable_helm() (*medikit.feature.kube.KubeConfig method*), 11

DjangoConfig (*class in medikit.feature.django*), 5

DjangoFeature (*class in medikit.feature.django*), 6

DockerConfig (*class in medikit.feature.docker*), 8

DockerFeature (*class in medikit.feature.docker*), 8

E

`enable()` (*medikit.feature.git.GitConfig method*), 10
`enable_helm()` (*medikit.feature.kube.KubeConfig method*), 11
`enabled` (*medikit.feature.git.GitConfig attribute*), 10

F

`FormatConfig` (*class in medikit.feature.format*), 9
`FormatFeature` (*class in medikit.feature.format*), 9

G

`get()` (*medikit.feature.python.PythonConfig method*), 19
`get_constraints()` (*medikit.feature.python.PythonConfig method*), 19
`get_dependencies()` (*medikit.feature.nodejs.NodeJSConfig method*), 15
`get_extras()` (*medikit.feature.python.PythonConfig method*), 19
`get_init_files()` (*medikit.feature.python.PythonConfig method*), 19
`get_inline_requirements()` (*medikit.feature.python.PythonConfig method*), 19
`get_requirements()` (*medikit.feature.python.PythonConfig method*), 19
`get_setup()` (*medikit.feature.python.PythonConfig method*), 19
`get_targets()` (*medikit.feature.kube.KubeConfig method*), 11
`get_theme()` (*medikit.feature.sphinx.SphinxConfig method*), 21
`get_variants()` (*medikit.feature.kube.KubeConfig method*), 11
`get_vendors()` (*medikit.feature.python.PythonConfig method*), 19
`GitConfig` (*class in medikit.feature.git*), 10
`GitFeature` (*class in medikit.feature.git*), 10

I

`image` (*medikit.feature.docker.DockerConfig attribute*), 8

J

`javascript_tools` (*medikit.feature.format.FormatConfig attribute*), 9

K

`KubeConfig` (*class in medikit.feature.kube*), 11
`KubeFeature` (*class in medikit.feature.kube*), 11

M

`MakeFeature` (*class in medikit.feature.make*), 14
`medikit.feature.django` (*module*), 5
`medikit.feature.docker` (*module*), 6
`medikit.feature.format` (*module*), 8
`medikit.feature.git` (*module*), 9
`medikit.feature.kube` (*module*), 10
`medikit.feature.make` (*module*), 11
`medikit.feature.nodejs` (*module*), 14
`medikit.feature.pylint` (*module*), 15
`medikit.feature.pytest` (*module*), 16
`medikit.feature.python` (*module*), 17
`medikit.feature.sphinx` (*module*), 21
`medikit.feature.webpack` (*module*), 22
`medikit.feature.yapf` (*module*), 22

N

`NodeJSConfig` (*class in medikit.feature.nodejs*), 15
`NodeJSFeature` (*class in medikit.feature.nodejs*), 15

O

`on_before_start()` (*medikit.feature.format.FormatFeature method*), 9
`on_before_start()` (*medikit.feature.yapf.YapfFeature method*), 23
`on_end()` (*medikit.feature.docker.DockerFeature method*), 8
`on_end()` (*medikit.feature.git.GitFeature method*), 10
`on_end()` (*medikit.feature.nodejs.NodeJSFeature method*), 15
`on_end()` (*medikit.feature.python.PythonFeature method*), 20
`on_generate` (*medikit.feature.python.PythonConfig attribute*), 19
`on_make_generate()` (*medikit.feature.django.DjangoFeature method*), 6
`on_make_generate()` (*medikit.feature.docker.DockerFeature method*), 8
`on_make_generate()` (*medikit.feature.format.FormatFeature method*), 9
`on_make_generate()` (*medikit.feature.git.GitFeature method*), 10
`on_make_generate()` (*medikit.feature.kube.KubeFeature method*), 11
`on_make_generate()` (*medikit.feature.nodejs.NodeJSFeature method*), 15

[on_make_generate\(\)](#)
 ([medikit.feature.pylint.PylintFeature](#) *method*),
 15
[on_make_generate\(\)](#)
 ([medikit.feature.pytest.PytestFeature](#) *method*),
 16
[on_make_generate\(\)](#)
 ([medikit.feature.python.PythonFeature](#)
 method), 20
[on_make_generate\(\)](#)
 ([medikit.feature.sphinx.SphinxFeature](#) *method*),
 22
[on_make_generate\(\)](#)
 ([medikit.feature.webpack.WebpackFeature](#)
 method), 22
[on_make_generate\(\)](#)
 ([medikit.feature.yapf.YapfFeature](#) *method*),
 23
[on_python_generate\(\)](#)
 ([medikit.feature.django.DjangoFeature](#)
 method), 6
[on_python_generate\(\)](#)
 ([medikit.feature.format.FormatFeature](#)
 method), 9
[on_python_generate\(\)](#)
 ([medikit.feature.pylint.PylintFeature](#) *method*),
 15
[on_python_generate\(\)](#)
 ([medikit.feature.pytest.PytestFeature](#) *method*),
 16
[on_python_generate\(\)](#)
 ([medikit.feature.sphinx.SphinxFeature](#) *method*),
 22
[on_python_generate\(\)](#)
 ([medikit.feature.yapf.YapfFeature](#) *method*),
 23
[on_start\(\)](#) ([medikit.feature.django.DjangoFeature](#)
 method), 6
[on_start\(\)](#) ([medikit.feature.format.FormatFeature](#)
 method), 9
[on_start\(\)](#) ([medikit.feature.git.GitFeature](#) *method*),
 10
[on_start\(\)](#) ([medikit.feature.make.MakeFeature](#)
 method), 14
[on_start\(\)](#) ([medikit.feature.nodejs.NodeJSFeature](#)
 method), 15
[on_start\(\)](#) ([medikit.feature.pytest.PytestFeature](#)
 method), 16
[on_start\(\)](#) ([medikit.feature.python.PythonFeature](#)
 method), 20
[on_start\(\)](#) ([medikit.feature.yapf.YapfFeature](#)
 method), 23

P

[package_dir](#) ([medikit.feature.python.PythonConfig](#)
 attribute), 19
[PylintFeature](#) (*class in medikit.feature.pylint*), 15
[PytestConfig](#) (*class in medikit.feature.pytest*), 16
[PytestFeature](#) (*class in medikit.feature.pytest*), 16
[python_tools](#) ([medikit.feature.format.FormatConfig](#)
 attribute), 9
[PythonConfig](#) (*class in medikit.feature.python*), 19
[PythonFeature](#) (*class in medikit.feature.python*), 20

R

[requires](#) ([medikit.feature.django.DjangoFeature](#) *at-*
 tribute), 6
[requires](#) ([medikit.feature.kube.KubeFeature](#) *at-*
 tribute), 11
[requires](#) ([medikit.feature.nodejs.NodeJSFeature](#) *at-*
 tribute), 15
[requires](#) ([medikit.feature.pylint.PylintFeature](#) *at-*
 tribute), 16
[requires](#) ([medikit.feature.pytest.PytestFeature](#) *at-*
 tribute), 16
[requires](#) ([medikit.feature.python.PythonFeature](#) *at-*
 tribute), 21
[requires](#) ([medikit.feature.webpack.WebpackFeature](#)
 attribute), 22
[requires](#) ([medikit.feature.yapf.YapfFeature](#) *attribute*),
 23

S

[set_remote\(\)](#) ([medikit.feature.docker.DockerConfig](#)
 method), 8
[set_theme\(\)](#) ([medikit.feature.sphinx.SphinxConfig](#)
 method), 21
[set_version\(\)](#) ([medikit.feature.pytest.PytestConfig](#)
 method), 16
[setup\(\)](#) ([medikit.feature.nodejs.NodeJSConfig](#)
 method), 15
[setup\(\)](#) ([medikit.feature.python.PythonConfig](#)
 method), 20
[SphinxConfig](#) (*class in medikit.feature.sphinx*), 21
[SphinxFeature](#) (*class in medikit.feature.sphinx*), 22
[static_dir](#) ([medikit.feature.django.DjangoConfig](#) *at-*
 tribute), 5

T

[theme](#) ([medikit.feature.sphinx.SphinxConfig](#) *attribute*),
 21

U

[use_default_builder\(\)](#)
 ([medikit.feature.docker.DockerConfig](#) *method*),
 8

`use_helm (medikit.feature.kube.KubeConfig attribute),`
11
`use_rocker_builder()`
`(medikit.feature.docker.DockerConfig method),`
8
`using()` `(medikit.feature.format.FormatConfig`
`method),` 9

V

`variables (medikit.feature.docker.DockerConfig at-`
`tribute),` 8
`version (medikit.feature.django.DjangoConfig at-`
`tribute),` 5
`version (medikit.feature.pytest.PytestConfig attribute),`
16
`version_file (medikit.feature.python.PythonConfig`
`attribute),` 20

W

`WebpackFeature (class in medikit.feature.webpack),`
22

Y

`YapfFeature (class in medikit.feature.yapf),` 23