Grocker

Release 6.9.dev0

Feb 11, 2020

Contents

Grocker allows you to bundle your Python applications as Docker image while keeping the image size as minimal as possible.

Grocker uses `debian:jessie`, `debian:stretch` or `alpine:latest` as its base image.

Grocker is hosted on Github at https://github.com/polyconseil/Grocker. *Grocker* full documentation is available on https://grocker.readthedocs.io/.

CHAPTER 1

Installation

1. Install Docker Engine. See its official documentation.
2. Install Grocker with *pip*: `pip install grocker`.

CHAPTER 2

Basic usage

```
$ grocker build ipython==5.0.0 --entrypoint ipython
$ docker run --rm -ti ipython:5.0.0-<grocker-version>
```

CHAPTER 3

Direct wheel path

A wheel can also be directly passed to grocker to avoid the need to upload an artefact to build an image.

Grocker will switch to this mode if a `/` is present in the argument. Pip `extra` requirements can be used in this mode.

```
$ grocker build ./path/to/ipython-7.1.1-py3-none-any.whl[doc] --entrypoint ipython
$ docker run --rm -ti ipython-doc:7.1.1-<grocker-version>
```

CHAPTER 4

Topics

4.1 Grocker - a Docker image builder for Python applications

Grocker allows you to bundle your Python applications as Docker image while keeping the image size as minimal as possible.

Grocker uses `debian:jessie`, `debian:stretch` or `alpine:latest` as its base image.

Grocker is hosted on Github at https://github.com/polyconseil/Grocker. *Grocker* full documentation is available on https://grocker.readthedocs.io/.

4.2 Installation

1. Install Docker Engine. See its official documentation.
2. Install Grocker with *pip*: `pip install grocker`.

4.3 Basic usage

```
$ grocker build ipython==5.0.0 --entrypoint ipython
$ docker run --rm -ti ipython:5.0.0-<grocker-version>
```

4.4 Direct wheel path

A wheel can also be directly passed to grocker to avoid the need to upload an artefact to build an image.

Grocker will switch to this mode if a `/` is present in the argument. Pip `extra` requirements can be used in this mode.

```
$ grocker build ./path/to/ipython-7.1.1-py3-none-any.whl[doc] --entrypoint ipython
$ docker run --rm -ti ipython-doc:7.1.1-<grocker-version>
```

4.5 How the Docker image is built

The final Docker image, which we'll call the **runner** image from now on, is built in four phases:

1. First, the **root** image is built from the official **debian:jessie** image. In this phase all of the image's runtime dependencies are installed.
2. Then the **compiler** image is built from the **root** image. In this image all of the build dependencies are installed, and a compiler script is added.
3. The third step is to run the **compiler** image and build Python wheels for the application and its dependencies. Those wheels are stored in a Docker data volume.
4. Finally, the wheels stored in the data volume are exposed using a web server (using a docker container) and the final **runner** image is built from the **root** image, using the wheels.

There is one **root** image and one **compiler** image by *config* (see *.grocker.yml config file*). The wheel data volume is reused between builds with the same *config*.

Grocker ends up building three Docker images, two of which are reused between each build using the same *config*:

1. The **root** image, which contains all (system) runtime dependencies
2. The **compiler** image, which is the **root** image extended with (system) build dependencies including a C compiler and a compiler script allowing to build all needed wheels from a package name.

The last image is the **runner**, the final product of the Grocker build chain, which contains the application and all its dependencies (excluding external services).

4.6 Advanced usage

Grocker build chain can be customized using the command line interface or the `.grocker.yml` file (or the one defined through the `--config` parameter). When both are used, command line arguments take precedence.

4.6.1 The build command line interface

```
Usage: grocker [OPTIONS] COMMAND [ARGS]...

Options:
  --version      Show the version and exit.
  -v, --verbose
  --help         Show this message and exit.

Commands:
  build  Build docker image for <release> (version...
  purge  Purge Grocker created Docker stuff
```

```
Usage: grocker build [OPTIONS] RELEASE

  Build docker image for RELEASE (version specifiers can be used).

  RELEASE can either be the name of a project, or the path to a wheel to
  use. In both cases, extra requirements can be applied:

  grocker build your_project[with_extra]==1.2.3

  grocker build /path/to/your_project-1.2.3.whl[with_extra]


Options:
  -c, --config <filename>         Grocker config file
  -r, --runtime <runtime>         runtime used to build and run this image
  --pip-conf <filename>           pip configuration file used to download
                                  dependencies (by default use pip config
                                  getter)
  --pip-constraint <filename>     pip constraint file used to download
                                  dependencies
  -e, --entrypoint <entrypoint>   Docker entrypoint to use to run this image
  --volume <volume>               Container storage and configuration area
  --port <port>                   Port on which a container will listen for
                                  connections
  --env <env=value>               Additional environment variable for final image
  --image-prefix <uri>            docker registry or account on Docker
                                  official registry to use
  --image-base-name <name>        base name for the image (eg '<image-
                                  prefix>/<image-base-name>:<image-version>')
  -n, --image-name <name>         name used to tag the build image
  --result-file <filename>        yaml file where results (image name, ...)
                                  are written
  --build-dependencies / --no-build-dependencies
                                  build the dependencies
  --build-image / --no-build-image
                                  build the docker image
  --push / --no-push              push the image
  --help                          Show this message and exit.
```

Actions

Grocker splits the build chain into three steps:

1. `dependencies`, compiles wheels and stores them in a data volume.
2. `image`, builds the **runner** image using stored wheels.
3. `push`, pushes the **runner** image on the configured Docker registry (only if a docker image prefix is given)

This allows you, for example, to build an image without pushing it, then do some tests, and after your tests passed push the image.

Pip config

The `--pip-conf` permits to select a specific pip config file. By default Grocker uses the user pip config file.

4.6.2 `.grocker.yml` config file

The `.grocker.yml` allows to customise Grocker build by setting dependencies among other things. It is written in YAML. By default, Grocker looks for this file in the current directory. Its default values are:

```yaml
# .grocker.yml (defaults)
runtime: alpine/3
pip_constraint: # optional
volumes: []
ports: []
env: {}
repositories: {}
dependencies:
    run: []
    build: []
docker_image_prefix: # optional
image_base_name: # optional
entrypoint_name: grocker-runner
```

Dependencies

Two kind of dependencies can be declared those used on the final image (`run`) and those which will be installed only on the build image (`build`).

Each package declared on those lists will be installed using the system package manager.

Repositories

Each item of the `repositories` mapping is a mapping with two keys:

- `uri`: The deb line of the repository
- `key`: The GPG key used to sign this repository packages

The first level mapping key is used as the repository identifier.

Example

An example with all options customised:

```yaml
# .grocker.yml (full example)
runtime: jessie/2.7
pip_constraint: constraints.txt
volumes: ['/data', '/cache']
ports: [8080, 8081]
env:
    SOME_ENV_VAR: value of the envvars
    ANOTHER_ENV_VAR: 45
    http_proxy: http://127.0.0.1:8080
repositories:
    nginx:
        uri: deb http://nginx.org/packages/debian/ jessie nginx
        key: |
            -----BEGIN PGP PUBLIC KEY BLOCK-----
            Version: GnuPG v1.4.11 (FreeBSD)
```

(continues on next page)

(continued from previous page)

```
            mQENBE5OMmIBCAD+FPYKGriGGf7NqwKfWC83cBV01gabgVWQmZbMcFzeW+hMsgxH
            W6iimD0RsfZ9oEbfJCPG0CRSZ7ppq5pKamYs2+EJ8Q2ysOFHHwpGrA2C8zyNAs4I
            QxnZZIbETgcSwFtDun0XiqPwPZgyuXVm9PAbLZRbfBzm8wR/3SWygqZBBLdQk5TE
            fDR+Eny/M1RVR4xClECONF9UBB2ejFdI1LD45APbP2hsN/piFByU1t7yK2gpFyRt
            97WzGHn9MV5/TL7AmRPM4pcr3JacmtCnxXeCZ8nLqedoSuHFuhwyDnlAbu8I16O5
            XRrfzhrHRJFM1JnIiGmzZi6zBvH0ItfyX6ttABEBAAG0KW5naW54IHNpZ25pbmcg
            a2V5IDxzaWduaW5nLWtleUBuZ2lueC5jb20+iQE+BBMBAgAoBQJOTjJiAhsDBQkJ
            ZgGABgsJCAcDAgYVCAIJCgsEFgIDAQIeAQIXgAAKCRCr9b2Ce9m/YpvjB/98uV4t
            94d0oEh5XlqEZzVMrcTgPQ3BZt05N5xVuYaglv7OQtdlErMXmRWaFZEqDaMHdniC
            sF63jWMd29vC4xpzIfmsLK3ce9oYo4t9o4WWqBUdf0Ff1LMz1dfLG2HDtKPfYg3C
            8NESud09zuP5NohaE8Qzj/4p6rWDiRpuZ++4fnL3Dt3N6jXILwr/TM/Ma7jvaXGP
            DO3kzm4dNKp5b5bn2nT2QWLPnEKxvOg5Zoej8l9+KFsUnXoWoYCkMQ2QTpZQFNwF
            xwJGoAz8K3PwVPUrIL6b1lsiNovDgcgP0eDgzvwLynWKBPkRRjtgmWLoeaS9FAZV
            ccXJMmANXJFuCf26iQEcBBABAgAGBQJOTkelAAoJEKZP1bF62zmo79oH/1XDb29S
            YtWp+MTJTPFEwlWRiyRuDXy3wBd/BpwBRIWfWzMs1gnCjNjk0EVBVGa2grvy9Jtx
            JKMd6l/PWXVucSt+U/+GO8rBkw14SdhqxaS2l14v6gyMeUrSbY3XfToGfwHC4sa/
            Thn8X4jFaQ2XN5dAIzJGU1s5JA0tjEzUwCnmrKmyMlXZaoQVrmORGjCuH0I0aAFk
            RS0UtnB9HPpxhGVbs24xXZQnZDNbUQeulFxS4uP3OLDBAeCHl+v4t/uotIad8v6J
            SO93vc1evIje6lguE81HHmJn9noxPItvOvSMb2yPsE8mH4cJHRTFNSEhPW6ghmlf
            Wa9ZwiVX5igxcvaIRgQQEQIABgUCTk5b0gAKCRDs8OkLLBcgg1G+AKCnacLb/+W6
            cflirUIExgZdUJqoogCeNPVwXiHEIVqithAM1pdY/gcaQZmIRgQQEQIABgUCTk5f
            YQAKCRCpN2E5pSTFPnNWAJ9gUozyiS+9jf2rJvqmJSeWuCgVRwCcCUFhXRCpQO2Y
            Va3l3WuB+rgKjsQ=
            =A015
            -----END PGP PUBLIC KEY BLOCK-----
dependencies:
    run:
        - libzbar0
        - libjpeg62-turbo
        - libffi6
        - libtiff5
        - nginx
    build:
        - libzbar-dev
        - libjpeg62-turbo-dev
        - libffi-dev
        - libtiff5-dev

docker_image_prefix: docker.example.com
entrypoint_name: my-runner
```

4.6.3 Purging Grocker stuffs

The `purge` command is here to clean Grocker created stuff of your Docker daemon.

```
Usage: grocker purge [OPTIONS]

  Purge Grocker created Docker stuff

Options:
  -a, --all-versions / --only-old-versions
  -f, --including-final-images / --excluding-final-images
  --help                          Show this message and exit.
```

4.7 Troubleshooting

4.7.1 I have a *PermissionError: [Errno 13] Permission denied*

Grocker does not ask for superuser rights before invoking Docker. The Docker socket has to be readable and writable by the current user. Many distributions create this socket to be readable and writable by *root* and the *docker* group.

One way to be able to use Grocker is to add your user to the *docker* group. On Debian:

```
$ sudo adduser $(whoami) docker
$ su $(whoami)  # reload groups, you should also restart your session
```

4.8 Talks

4.8.1 EuroPython 2016 (Bilbao, 17 - 24 july 2016)

Grocker, a Docker build chain for Python applications

Short Abstract

Grocker is a Docker build chain for Python. It transforms your Python package into a self-contained Docker image which can be easily deployed in a Docker infrastructure. Grocker also adds a Docker entry point to easily start your application.

Abstract

At Polyconseil, we build Paris electric car-sharing service: Autolib'. This system is based on many services developed using web technologies, Django and our own libraries to handle business logic.

Packaging is already a difficult problem, deploying large Python projects is even more difficult. When deploying on a live and user-centric system like Autolib', you cannot rely on Pip and external PyPI servers which might become unavailable and are beyond your control. In the beginning we used classic Debian packaging: it was a maintenance hell. It took hours to build our packages and update their metadata to match our Python packages. So we switched to Docker.

Docker allows us to have a unique item that is deployed in production systems: code updates are now atomic and deterministic! But before deploying the Docker image, you need to build it. That's where Grocker comes in.

Grocker is a Docker build chain for Python. It will transform your Python package into a self-contained Docker image which can be easily deployed in a Docker Infrastructure. Grocker also adds a Docker entry point to easily start your application.

Grocker, a Docker build chain for Python applications

Who am I

- Senior developer at Polyconseil for 2 years

- Member of the Autolib'[1] dev team
- Working on DevOps subjects, among others

Presenter Notes

- 6 electric car sharing services
- Micro services (~30 different apps)
- Apps based on web technologies, Django and our own libraries (for business logic)

Why we built Grocker

- Debian packaging was hell (in 2015)
- Containerized applications are the future!
- OpenShift/Source-To-Image use source not package[2]

Presenter Notes

- Debian Packaging
  - Need to update metadata by hand
  - Our worst case: 48h to package one application and all its dependencies
  - Applications are deployed once a week, two days of packaging -> three days for live tests, bug fixes, testing the bug fixes and deployment
  - *Setuptools* entrypoints do not work (except if all your dependencies are exactly those in your setup.py files)
- Our goal: to deploy all our applications at least once a day
- Why Docker
  - Build dependencies are not installed on production servers
  - Atomics deployments!
  - Resource optimisation (one host = N Docker)

Other approaches we have considered

- `pip install` on server
- Distribution packages
- Dockerfile
- Slug (Heroku like)

[1] Paris electric car sharing service
[2] I lied, when we built grocker, we did not know this project

Presenter Notes

- `pip install` on a production server, seriously?
    - Build extensions in place (ie on production server)
    - Can fail
- Infrastructure and application dependencies should not be mixed otherwise update of one part can be blocked by the other one.
- Dockerfile:
    - Extension building: One RUN command to install build dependencies, build wheels, uninstall build dependencies (docker layer, size matters)
- Slug:
    - Separate build and run phases
    - One base image for all applications
    - No system dependencies (they are on the base image)

How Grocker works

Presenter Notes

Build phases:

1. Build the *root image* based on `debian:jessie` (add system packages for runtime dependencies)
2. Build the *compiler image* based on the *root image* (add system packages marked as build dependencies, install compiler script)
3. Instantiate a container for the *compiler image* to build Python wheels for the application and its dependencies (and store them in a data volume)
4. Build the *runner image* based on the *root image* (install compiled wheels served by a web server[3])

In fact, this is more complex, there is one *root image* and one *compiler image* by “config”.

Why using a compiler image?

- Wheels, simple and reproducible deployment (see http://pythonwheels.com/)
- C extensions need dependencies to compile
- Docker layers…

[3] We use the `pip --no-index` to install wheels.

Presenter Notes

- Wheels and C extensions have to be linked with the same library versions than those used at runtime
- Docker layers: data on layers are never deleted, just masked

---

Current limitations

- Root image size: between 200 MB (image used for tests) and 600 MB (for our most cumbersome applications)
  - `debian:jessie`: 125 MB
  - `alpine`: 5 MB, but missing zbar
- Can only build packaged applications

How to use it

```
$ grocker build ipython==5.0 --entrypoint ipython
[wait a long time]
$ docker run --rm -ti ipython:5.0-4.0
Python 3.4.2 (default, Oct  8 2014, 10:45:20)
Type "copyright", "credits" or "license" for more information.

IPython 5.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

How to use it (the other side)

With the default configuration, only working for:

- Packaged applications
- Python 3
- Without runtime dependencies

Otherwise, you will need a config file or to use config flags.

I need it!

- Grocker was open-sourced yesterday (2016-07-20)
- sources: https://github.com/polyconseil/grocker
- package: https://pypi.python.org/pypi/grocker
- docs: https://grocker.readthedocs.io/en/latest/

Thanks!

4.9 ChangeLog

4.9.1 6.9 (unreleased)

- Drop Python 2.7 support (for Grocker command) **breaking change**

4.9.2 6.8 (2019-11-18)

- Add Python 3.8 support (for Grocker command).
- Add support for buster/3.8 runtime.
- Add support for buster/3.7 runtime.
- Add support for alpine/3 runtime and use it by default.
- Deprecate stretch/3.7 runtime (**deprecation warning**).
- Deprecate alpine/3.6 runtime (**deprecation warning**)
- Drop Python 3.4 support since it is not supported anymore (**breaking change**).

4.9.3 6.7 (2019-02-19)

- Fix pip install error when building an image from a wheel with extra requirements passed from a filepath

4.9.4 6.6 (2019-01-18)

- Python 3.4 & 3.5 support (for Grocker command) will be dropped in next major version (**deprecation warning**)
- Add support for Python’s stretch/3.7 docker image.

4.9.5 6.5 (2018-11-28)

- Replace “+” signs in python versions by “-” in default image name

4.9.6 6.4 (2018-11-28)

- Python 2.7 support (for Grocker command) will be dropped in next major version (**deprecation warning**)
- Accept wheel path as release argument

4.9.7 6.3 (2018-05-25)

- Add support for Python's stretch+3.6 docker image.
- Deprecate stretch/3.5 runtime

4.9.8 6.2.1 (2018-04-16)

- Use latest pip version (10.0)

4.9.9 6.2 (2018-02-07)

- Switch to Docker(-py) 3.0

4.9.10 6.1 (2017-11-15)

- Remove grocker version from the image default tag

4.9.11 6.0 (2017-10-25)

- Allow to set environment variables on the final image
- Deprecate Jessie runtimes
- Allow to use Debian Jessie (Python 2.7 et 3.4), Debian Stretch (Python 3.5) and Alpine (Python 3.6) as base distribution (**breaking change**)
- Simplify dependency declaration (**breaking change**)
- Manage HTTPError when using get or build image

4.9.12 5.4 (2017-10-12)

- Use python venv instead of virtualenv in compile image

4.9.13 5.3 (2017-08-21)

- Support of pip trusted host config in wheel builder.
- Retrieve image manifest digest from registry.

4.9.14 5.2 (2017-07-18)

- Adds gnupg2 to allow grocker to build stretch images.

4.9.15 5.1 (2017-06-07)

- Also extract pip's timeout configuration from local pip configuration file

4.9.16 5.0 (2017-03-10)

- Switch cli to click (**breaking change**)
- Add support for Python 3.6
- Drop obsolete .grocker file
- Switch to docker(-py) 2
- Only manage Grocker created stuff when purging
- Drop cron, ssmtp and sudo specific code
- Add support for alpine base images

4.9.17 4.6 (2016-12-22)

- Fix shell equality test
- Disable useless pip cache
- Stop using sudo in compiler script

4.9.18 4.5 (2016-12-19)

- Use env vars to pass pip constraint file to wheel compiler.
- Fix empty config file bug

4.9.19 4.4 (2016-11-22)

- Add `--image-base-name` option to allow customizing the generated image name

4.9.20 4.3.2 (2016-11-09)

- Fix grocker for releases with extras.
- Make sure most tests run without `--docker-image-prefix` hence without cache.

4.9.21 4.3.1 (2016-11-09)

- **Warning** - This version is broken for extras, use 4.3.2 instead.
- Fix `compiler-image/provision.sh` sh syntax. `source` replaced by `.`

4.9.22 4.3 (2016-11-08)

- **Warning** - This version is broken, use 4.3.2 instead.
- Correctly parse the release string and store extras as label and environment variable
- Use the image defined in the configuration (it still needs to be debian based - for the moment)
- Provision scripts now only require sh (instead of bash previously)
- Correctly parse OSX docker client output

4.9.23 4.2 (2016-10-13)

- Add a sync after chmod call to avoid an AUFS issue
- Fix image search when repoTags is None and not an empty list
- Use env vars to expose grocker meta-data to the application
- Expose some meta-data using image labels
- Use docker build args to pass some build parameters
- Add application venv bin in PATH

4.9.24 4.1 (2016-09-19)

- Ask for a specific verison of the Docker API (1.21)
- Exclude docker-py 1.10.x (require requests < 2.11)

4.9.25 4.0 (2016-07-20)

- Drop predefined extra apt repositories
- Drop predefined exposed ports and volumes
- tags: Rename `grocker.step` into `grocker.image.kind`
- Keep the hash type (`sha256`) in the result file

4.9.26 Grocker 3.0.1 (2016-06-06)

- Allow pip_constraint to be a relative path

4.9.27 Grocker 3.0.0 (2016-06-06)

- Also use the constraint file to upgrade pip and setuptools in the app venv
- Add pip_constraint entry to config yaml file
- Remove default dependencies list
- Make –docker-image-prefix optional
- Merge entrypoint into app

4.9.28 Grocker 2.4.2 (2016-04-11)

4.9.29 Grocker 2.4.1 (2016-04-11)

- Fix the use of grocker as a library (broken in previous release)

4.9.30 Grocker 2.4.0 (2016-04-11)

- Only install needed runtime in images
- Allow to set system dependencies by project
- Remove dependencies to host UID

4.9.31 Grocker 2.3.1 (2016-03-03)

- Use Python 3 in entry point venv when runtime is *python3* (fix).

4.9.32 Grocker 2.3.0 (2016-03-03)

- Ask for a specific python version

4.9.33 Grocker 2.2.0 (2016-02-24)

- Allow grocker to be used as a library
- Use common package cache dir for all grocker instances

4.9.34 Grocker 2.1.0 (2016-02-11)

- Add libyaml to run dependencies
- Stop process on build error
- Fix Python 3 support

4.9.35 Grocker 2.0.1

- Add docker-machine support

4.9.36 Grocker 2.0.0

- Grocker v2 first release