
tox-docker

Release 3.0.0

Mar 29, 2021

Contents

1	Usage and Installation	3
2	Configuration	5
3	Command-Line Arguments	7
4	Example	9
5	Environment Variables	11
6	Version Compatibility	13
7	Change Log	15
8	Development	17
8.1	Code Style	17

A [tox](#) plugin which runs one or more [Docker](#) containers during the test run.

Links: [Source Code](#) | [Documentation](#)

CHAPTER 1

Usage and Installation

Tox loads all plugins automatically. To use tox-docker, `pip install` it into the same Python environment as you install tox into, whether that's a virtualenv, etc.

You do not need to do anything special when running tox to invoke tox-docker, however you do need to configure your project to configure docker containers (see “Configuration” below).

CHAPTER 2

Configuration

Each docker container you want to run must be configured via a `[docker:container-name]` section. The `container-name` is a name you choose which must start with a letter and consist of only letters, numbers, dots, hyphens, and underscores. Each `[docker:container-name]` section must contain at least an `image` directive, which must name a [Docker image](#) as you'd pass to `docker run`:

```
[docker:db]
image = postgres:9-alpine
```

Then, in your `[testenv]`, use the `docker` directive to list containers you wish to run during those tests:

```
[testenv]
docker =
    db
commands = ...
```

The `[docker:container-name]` section may contain the following directives:

image (required) The [Docker image](#) to run. This value is passed directly to Docker, and may be of any of the forms that Docker accepts in eg `docker run`.

environment A multi-line list of `KEY=value` settings which is used to set environment variables for the container. The variables are only available to the container, not to other containers or the test environment.

ports A multi-line list of port mapping specifications, as `HOST_PORT:CONTAINER_PORT/PROTO`, which causes the container's EXPOSE d port to be available on `HOST_PORT`. See below for more on port mapping.

If `ports` is not specified, all the container's EXPOSE d ports are mapped (equivalent to `docker run -P ...`)

For each mapped port, an environment variable of the form `<container-name>_<port-number>_<protocol>_PORT` (eg `NGINX_80_TCP_PORT` or `TELEGRAF_8092_UDP_PORT`) is set for the test run.

For each container, an environment variable of the form `<container_name>_HOST` is set for the test run, indicating the host name or IP address to use to communicate with the container.

In both environment variables, the container name part is converted to upper case, and all non-alphanumeric characters are replaced with an underscore (`_`).

Tox-docker does not attempt to ensure that you have proper permission to bind the `HOST_PORT`, that it is not already bound and listening, etc; if you explicitly list ports, it is your responsibility to ensure that it can be successfully mapped.

links A multi-line list of `container links`, as `other-container-name` or `other-container-name:alias`. If no alias is given, the `other-container-name` is used. Within the container, the `EXPOSE`d ports of the other container will be available via the alias as hostname.

When using links, you must specify containers in the correct start order in the `docker` directive of your `testenv` – tox-docker does not attempt to resolve a valid start order.

volumes A multi-line list of `volumes` to make available to the container, as `<type>:<options>:<outside_path_or_name>:<inside_path>`. The type must be `bind`, and the only supported options are `rw` (read-write) or `ro` (read-only). The `outside_path_or_name` must be a path that exists on the host system. Both the `outside_path` and `inside_path` must be absolute paths.

healthcheck_cmd, healthcheck_interval, healthcheck_retries, healthcheck_start_period, healthche

These set or customize parameters of the container `health check`. The `healthcheck_interval`, `healthcheck_start_period`, and `healthcheck_timeout` are specified as a number of seconds. The `healthcheck_cmd` is an `argv` list which must name a command and arguments that can be run within the container; if not specified, any health check built in to the container is used.

If any healthcheck parameters are defined, tox-docker will delay the test run until the container reports healthy, and will fail the test run if it never does so (within the parameters specified).

Command-Line Arguments

All Docker container configuration is specified in `tox.ini`, but some aspects of `tox-docker`'s behavior can be changed at run-time:

--docker-dont-stop=CONTAINER After the test run, don't stop & remove the named `CONTAINER` – leaving the container running allows manual inspection of it, eg via `docker exec . . .`. May be specified multiple times to leave several containers running.

CHAPTER 4

Example

```
[testenv:integration-tests]
deps = pytest
commands = py.test {toxiniidir}/tests
docker =
    db
    appserv

[docker:db]
image = postgres:11-alpine
# Environment variables are passed to the container. They are only
# available to that container, and not to the testenv, other
# containers, or as replacements in other parts of tox.ini
environment =
    POSTGRES_PASSWORD=hunter2
    POSTGRES_USER=dbuser
    POSTGRES_DB=tox_test_db
# The healthcheck ensures that tox-docker won't run tests until the
# container is up and the command finishes with exit code 0 (success)
healthcheck_cmd = PGPASSWORD=$POSTGRES_PASSWORD psql \
    --user=$POSTGRES_USER --dbname=$POSTGRES_DB \
    --host=127.0.0.1 --quiet --no-align --tuples-only \
    -1 --command="SELECT 1"
healthcheck_timeout = 1
healthcheck_retries = 30
healthcheck_interval = 1
healthcheck_start_period = 1
# Configure a bind-mounted volume on the host to store Postgres' data
# NOTE: this is included for demonstration purposes of tox-docker's
# volume capability; you probably don't want to do this for real
# testing use cases, as this could persist data between test runs
volumes =
    bind:rw:/my/own/datadir:/var/lib/postgresql/data

[docker:appserv]
```

(continues on next page)

(continued from previous page)

```
# You can use any value that `docker run` would accept as the image
image = your-registry.example.org:1234/your-appserv
# Within the appserv container, host "db" is linked to the postgres container
links =
    db:db
# Use ports to expose specific ports; if you don't specify ports, then all
# the EXPOSEd ports defined by the image are mapped to an available
# ephemeral port.
ports =
    8080:8080/tcp
```

Environment Variables

If you are running in a Docker-In-Docker environment, you can override the address used for port checking using the environment variable `TOX_DOCKER_GATEWAY`. This variable should be the hostname or ip address used to connect to the container.

CHAPTER 6

Version Compatibility

Tox-docker requires tox to be run in Python 3.6 or newer, and requires tox version 3.0 or newer. Older versions of tox-docker may work with older versions of Python or tox, but these configurations are no longer supported.

CHAPTER 7

Change Log

- **3.0.0**
 - Support Tox 3 and newer only
 - Automatically cleans up started docker containers, even if Tox encounters an error during the test run (thanks @d9pouces)
- **2.0.0**
 - Support Python 3.6 and newer only
 - Move all container configuration to `[docker:container-name]` sections
 - Don't infer container health by pinging TCP ports; only the healthcheck indicates a container's health

8.1 Code Style

Tox-docker uses black and isort to enforce style standards on the codebase. The formatting is ordinarily done for you via `pre-commit`, and is enforced via the `tox -e style` build. To work on tox-docker locally with `pre-commit`, *pip install -r dev-requirements.txt* and `pre-commit install` to set up the git hooks; subsequently, when you `git commit`, the formatter will be run. If the changed files are not conformant, the hook will have reformatted them and you may need to run `pre-commit` again. You can run `pre-commit run --files *.py` to manually run the formatters.