tox-docker

Release 4.1.0.post22

Sep 21, 2023

Contents

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`; or a `build` directive, containing the path to a Dockerfile as you'd pas to `docker build`:

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

# OR

[docker:app]
dockerfile = {toxinidir}/Dockerfile
```

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` 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`. One of `image` or `dockerfile` is required.

`dockerfile` Path to a Dockerfile to build and run. One of `dockerfile` or `image` is required.

`dockerfile_target` Name of the build-stage to build in a multi-stage Dockerfile. An error is raised if `dockerfile_target` is set without `dockerfile` set.

`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.

`expose` A multi-line list of port mapping specifications, as `ENV_VAR=CONTAINER_PORT/PROTO`. Within the testenv, the environment variable `ENV_VAR` will contain the port number to connect to the docker container's

EXPOSE d port. If expose is specified, only the listed ports will have environment variables created for them.

If expose is not specified, all the container's EXPOSE d ports are made available (equivalent to docker run -P ...) using default environment variable names of the form <container-name>_<port-number>_<protocol>_PORT (eg NGINX_80_TCP_PORT or TELEGRAF_8092_UDP_PORT), with the container name and protocol converted to upper case, and non-alphanumeric characters replaced with an underscore (_).

host_var The name of an environment variable that will contain the hostname or IP address to use to communicate with the container. Defaults to <container_name>_HOST if not set, with the container name converted to upper case, and non-alphanumeric characters replaced with an underscore (_).

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. 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).

CHAPTER 3

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

Container Naming & Parallel Runs

Since version 4, tox-docker adds a suffix to the name of running containers, so that parallel invocations of tox may succeed (eg on a busy CI server). The details of the name suffix are not specified, and may change in a future version – you should not rely on the details of the generated name.

Even with unique container names, parallel runs may still fail, if you map a static exposed port number for a container (as the tox host will not let two processes bind the same port).

CHAPTER 5

Example

```
[testenv:integration-tests]
deps = pytest
commands = py.test {toxinidir}/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
# Expose ports to the testenv
expose =
    APP_HTTP_PORT=8080/tcp
```

CHAPTER 6

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 7

Version Compatibility

Tox-docker requires tox to be run in Python 3.8 or newer, and requires tox version 4 or newer. Older versions of tox-docker may work with older versions of Python or tox, but these configurations are no longer supported.

CHAPTER 8

Upgrading

Some configuration options were removed:

New in 5.0:

`ports` This directive was removed in tox-docker version 5.0. Use `expose` instead. The ability to map a container port to a specific host port was completely removed.

CHAPTER 9

Change Log

- **5.0.0**
  - Remove support for tox 3
  - Removed support for Python 3.7 and earlier
  - Remove `ports`; add `expose` and `host_var`
- **4.1.1**
  - Fix typo in README (thanks @akx)
- **4.1.0**
  - Drop test support for docker (Python library) 3.x; add test support for docker 6.x. Other versions may work, but we only support tested versions.
  - Add support for `dockerfile` and `dockerfile_target` directives to build local images
- **4.0.0**
  - Support tox 4 as well as tox 3
  - Drop support for Python 3.6
  - Give running containers a unique name to support concurrent & parallel tox use cases (thanks @chaitu-tk and @goodtune for inspiration)
  - Add support for image registry URLs that contain a port
- **3.1.0**
  - Support docker-py 5.x
- **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

CHAPTER 10

Development

10.1 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.