pytest_container

Testing Container Images with Python and Pytest

Dan Čermák

FOSDEM 2024

who -u

Dan Čermák

- Software Developer @SUSE
- i3 SIG, Package maintainer
- Developer Tools, Testing and Documentation, Home Automation
- https://dancermak.name
- dcermak
- @Defolos@mastodon.social

Why Python and Pytest?

When shell scripts are:

Why Python and Pytest?

When shell scripts are:

- portable / run everywhere

Why Python and Pytest?

When shell scripts are:

- portable / run everywhere
- fast 

Why Python and Pytest?

When shell scripts are:

- portable / run everywhere
- fast 
- everyone understands them

Sales pitch

Sales pitch

- 🌩 shell scripts are brittle

Sales pitch

- 🌩 shell scripts are brittle
- automatically pull, build, launch & cleanup containers 🌩

Sales pitch

-  shell scripts are brittle
- automatically pull, build, launch & cleanup containers 
- use testinfra for convenience

Sales pitch

- 🌩 shell scripts are brittle
- automatically pull, build, launch & cleanup containers 🌩
- use testinfra for convenience
- 🔀 parallel test execution via pytest-xdist

Sales pitch

- 🌩 shell scripts are brittle
- automatically pull, build, launch & cleanup containers 🌩
- use testinfra for convenience
- 🔀 parallel test execution via pytest-xdist
- find & bind free ports

Sales pitch

- shell scripts are brittle
- automatically pull, build, launch & cleanup containers
- use testinfra for convenience
- parallel test execution via pytest-xdist
- find & bind free ports
- create & destroy pods

Sales pitch

- shell scripts are brittle
- automatically pull, build, launch & cleanup containers
- use testinfra for convenience
- parallel test execution via pytest-xdist
- find & bind free ports
- create & destroy pods
- create and cleanup container volumes & bind mounts

Sales pitch

- shell scripts are brittle
- automatically pull, build, launch & cleanup containers
- use testinfra for convenience
- parallel test execution via pytest-xdist
- find & bind free ports
- create & destroy pods
- create and cleanup container volumes & bind mounts
- run the same test on multiple container images

Sales pitch

- shell scripts are brittle
- automatically pull, build, launch & cleanup containers
- use testinfra for convenience
- parallel test execution via pytest-xdist
- find & bind free ports
- create & destroy pods
- create and cleanup container volumes & bind mounts
- run the same test on multiple container images
- supports docker and podman transparently

Sales pitch

- shell scripts are brittle
- automatically pull, build, launch & cleanup containers
- use testinfra for convenience
- parallel test execution via pytest-xdist
- find & bind free ports
- create & destroy pods
- create and cleanup container volumes & bind mounts
- run the same test on multiple container images
- supports docker and podman transparently
- works with Python 3.6+ and on `x86_64`, `aarch64`, `s390x`, `ppcle64`

Basic Example

```
 1 import pytest
 2 from pytest_container import Container, ContainerData
 3
 4 TW = Container(
 5     url="registry.opensuse.org/opensuse/tumbleweed:latest"
 6 )
 7
 8
 9 @pytest.mark.parametrize("container", [TW], indirect=True)
10 def test_etc_os_release_present(container: ContainerData):
11     assert container.connection.file(
12         "/etc/os-release"
13     ).exists
```

Basic Example

```python
import pytest
from pytest_container import Container, ContainerData

TW = Container(
    url="registry.opensuse.org/opensuse/tumbleweed:latest"
)


@pytest.mark.parametrize("container", [TW], indirect=True)
def test_etc_os_release_present(container: ContainerData):
    assert container.connection.file(
        "/etc/os-release"
    ).exists
```

Basic Example

```python
import pytest
from pytest_container import Container, ContainerData

TW = Container(
    url="registry.opensuse.org/opensuse/tumbleweed:latest"
)


@pytest.mark.parametrize("container", [TW], indirect=True)
def test_etc_os_release_present(container: ContainerData):
    assert container.connection.file(
        "/etc/os-release"
    ).exists
```

Use Cases

Use Cases

-  system tests of base containers

Use Cases

- system tests of base containers
- test applications inside containers

Use Cases

- system tests of base containers
- test applications inside containers
- run tests of your application on multiple OS'

`pytest` basics

`pytest` basics

- Python testing framework

`pytest` basics

- Python testing framework
- executes all functions called `test_*`

`pytest` basics

- 🐍 Python testing framework
- executes all functions called `test_*`
- fixtures for setup & teardown 🖌️

```
def test_ehlo(smtp_connection):
    response, msg = smtp_connection.ehlo()
    assert response == 250
```

`pytest` basics

- Python testing framework
- executes all functions called `test_*`
- fixtures for setup & teardown 🖌

```
def test_ehlo(smtp_connection):
    response, msg = smtp_connection.ehlo()
    assert response == 250
```

- test parametrization

```
@pytest.mark.parametrize("x", [0, 1])
@pytest.mark.parametrize("y", [2, 3])
def test_foo(x: int, y: int):
    # do something with x & y here
```

Usage examples

Build a new container for tests

```
1  WEB_SERVER = DerivedContainer(
2      base="registry.opensuse.org/opensuse/leap:15.4",
3      containerfile="""RUN zypper -n in python3 \
4      && echo "Hello Green World!" > index.html
5  ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
6  """
7  )
8
9  @pytest.mark.parametrize(
10     "container", [WEB_SERVER], indirect=True
11 )
12 def test_python_installed(container: ContainerData):
13     assert container.connection.package(
14         "python3"
15     ).is_installed
```

Build a new container for tests

```
WEB_SERVER = DerivedContainer(
    base="registry.opensuse.org/opensuse/leap:15.4",
    containerfile="""RUN zypper -n in python3 \
    && echo "Hello Green World!" > index.html
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
"""
)

@pytest.mark.parametrize(
    "container", [WEB_SERVER], indirect=True
)
def test_python_installed(container: ContainerData):
    assert container.connection.package(
        "python3"
    ).is_installed
```

Build a new container for tests

```
WEB_SERVER = DerivedContainer(
    base="registry.opensuse.org/opensuse/leap:15.4",
    containerfile="""RUN zypper -n in python3 \
    && echo "Hello Green World!" > index.html
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
"""
)

@pytest.mark.parametrize(
    "container", [WEB_SERVER], indirect=True
)
def test_python_installed(container: ContainerData):
    assert container.connection.package(
        "python3"
    ).is_installed
```

Build a new container for tests

```
WEB_SERVER = DerivedContainer(
    base="registry.opensuse.org/opensuse/leap:15.4",
    containerfile="""RUN zypper -n in python3 \
    && echo "Hello Green World!" > index.html
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
"""
)

@pytest.mark.parametrize(
    "container", [WEB_SERVER], indirect=True
)
def test_python_installed(container: ContainerData):
    assert container.connection.package(
        "python3"
    ).is_installed
```

Get a free port on the host

```
 1 WEB_SERVER = DerivedContainer(
 2     # snip
 3     forwarded_ports=[PortForwarding(container_port=8000)],
 4 )
 5
 6 @pytest.mark.parametrize(
 7     "container", [WEB_SERVER], indirect=True
 8 )
 9 def test_port_forward(container: ContainerData, host):
10     cmd = (
11         "curl --fail localhost:"
12         + str(container.forwarded_ports[0].host_port)
13     )
14     host.run_expect([0], cmd)
```

Get a free port on the host

```
WEB_SERVER = DerivedContainer(
    # snip
    forwarded_ports=[PortForwarding(container_port=8000)],
)

@pytest.mark.parametrize(
    "container", [WEB_SERVER], indirect=True
)
def test_port_forward(container: ContainerData, host):
    cmd = (
        "curl --fail localhost:"
        + str(container.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Get a free port on the host

```
 1 WEB_SERVER = DerivedContainer(
 2     # snip
 3     forwarded_ports=[PortForwarding(container_port=8000)],
 4 )
 5
 6 @pytest.mark.parametrize(
 7     "container", [WEB_SERVER], indirect=True
 8 )
 9 def test_port_forward(container: ContainerData, host):
10     cmd = (
11         "curl --fail localhost:"
12         + str(container.forwarded_ports[0].host_port)
13     )
14     host.run_expect([0], cmd)
```

Get a free port on the host

```
 1 WEB_SERVER = DerivedContainer(
 2     # snip
 3     forwarded_ports=[PortForwarding(container_port=8000)],
 4 )
 5
 6 @pytest.mark.parametrize(
 7     "container", [WEB_SERVER], indirect=True
 8 )
 9 def test_port_forward(container: ContainerData, host):
10     cmd = (
11         "curl --fail localhost:"
12         + str(container.forwarded_ports[0].host_port)
13     )
14     host.run_expect([0], cmd)
```

Get a free port on the host

```python
WEB_SERVER = DerivedContainer(
    # snip
    forwarded_ports=[PortForwarding(container_port=8000)],
)

@pytest.mark.parametrize(
    "container", [WEB_SERVER], indirect=True
)
def test_port_forward(container: ContainerData, host):
    cmd = (
        "curl --fail localhost:"
        + str(container.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Create a Pod

```
MEDIAWIKI_FPM_POD = Pod(
    containers=[MEDIAWIKI_FPM_CONTAINER, NGINX_FPM_PROXY],
    forwarded_ports=[PortForwarding(container_port=80)],
)

@pytest.mark.parametrize(
    "pod", [MEDIAWIKI_FPM_POD], indirect=True
)
def test_port_forward(pod: PodData, host):
    cmd = (
        "curl --fail localhost:"
        + str(pod.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Create a Pod

```
MEDIAWIKI_FPM_POD = Pod(
    containers=[MEDIAWIKI_FPM_CONTAINER, NGINX_FPM_PROXY],
    forwarded_ports=[PortForwarding(container_port=80)],
)

@pytest.mark.parametrize(
    "pod", [MEDIAWIKI_FPM_POD], indirect=True
)
def test_port_forward(pod: PodData, host):
    cmd = (
        "curl --fail localhost:"
        + str(pod.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Create a Pod

```
MEDIAWIKI_FPM_POD = Pod(
    containers=[MEDIAWIKI_FPM_CONTAINER, NGINX_FPM_PROXY],
    forwarded_ports=[PortForwarding(container_port=80)],
)

@pytest.mark.parametrize(
    "pod", [MEDIAWIKI_FPM_POD], indirect=True
)
def test_port_forward(pod: PodData, host):
    cmd = (
        "curl --fail localhost:"
        + str(pod.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Create a Pod

```
MEDIAWIKI_FPM_POD = Pod(
    containers=[MEDIAWIKI_FPM_CONTAINER, NGINX_FPM_PROXY],
    forwarded_ports=[PortForwarding(container_port=80)],
)

@pytest.mark.parametrize(
    "pod", [MEDIAWIKI_FPM_POD], indirect=True
)
def test_port_forward(pod: PodData, host):
    cmd = (
        "curl --fail localhost:"
        + str(pod.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Create a Pod

```
MEDIAWIKI_FPM_POD = Pod(
    containers=[MEDIAWIKI_FPM_CONTAINER, NGINX_FPM_PROXY],
    forwarded_ports=[PortForwarding(container_port=80)],
)

@pytest.mark.parametrize(
    "pod", [MEDIAWIKI_FPM_POD], indirect=True
)
def test_port_forward(pod: PodData, host):
    cmd = (
        "curl --fail localhost:"
        + str(pod.forwarded_ports[0].host_port)
    )
    host.run_expect([0], cmd)
```

Run mutable tests

Run mutable tests

use the `container_per_test` fixture:

Run mutable tests

use the `container_per_test` fixture:

```
 1 @pytest.mark.parametrize(
 2     "container_per_test", [TW], indirect=True
 3 )
 4 def test_rm_rf(container_per_test):
 5     container_per_test.connection.run_expect([0], "rm -rf /")
 6
 7
 8 @pytest.mark.parametrize(
 9     "container_per_test", [TW], indirect=True
10 )
11 def test_uninstall_zypper(container_per_test):
12     container_per_test.connection.run_expect(
13         [0], "rpm -e --nodeps zypper"
14     )
```

Run mutable tests

use the `container_per_test` fixture:

```
 1 @pytest.mark.parametrize(
 2     "container_per_test", [TW], indirect=True
 3 )
 4 def test_rm_rf(container_per_test):
 5     container_per_test.connection.run_expect([0], "rm -rf /")
 6
 7
 8 @pytest.mark.parametrize(
 9     "container_per_test", [TW], indirect=True
10 )
11 def test_uninstall_zypper(container_per_test):
12     container_per_test.connection.run_expect(
13         [0], "rpm -e --nodeps zypper"
14     )
```

Run mutable tests

use the `container_per_test` fixture:

```
 1 @pytest.mark.parametrize(
 2     "container_per_test", [TW], indirect=True
 3 )
 4 def test_rm_rf(container_per_test):
 5     container_per_test.connection.run_expect([0], "rm -rf /")
 6
 7
 8 @pytest.mark.parametrize(
 9     "container_per_test", [TW], indirect=True
10 )
11 def test_uninstall_zypper(container_per_test):
12     container_per_test.connection.run_expect(
13         [0], "rpm -e --nodeps zypper"
14     )
```

Use the same container globally

Use the same container globally

use the `auto_container` / `auto_container_per_test` fixtures:

Use the same container globally

use the `auto_container`/`auto_container_per_test` fixtures:

```
1 CONTAINER_IMAGES = [TW, LEAP, SLE]
2
3
4 def test_etc_os_release(auto_container): ...
5
6
7 def test_zypper_rm_works(auto_container_per_test): ...
```

Use the same container globally

use the `auto_container` / `auto_container_per_test` fixtures:

```
1 CONTAINER_IMAGES = [TW, LEAP, SLE]
2
3
4 def test_etc_os_release(auto_container): ...
5
6
7 def test_zypper_rm_works(auto_container_per_test): ...
```

Dependencies between containers

```
 1 TW = Container(
 2     url="registry.opensuse.org/opensuse/tumbleweed:latest"
 3 )
 4 NGINX = DerivedContainer(
 5     base=TW,
 6     containerfile="RUN zypper -n in nginx",
 7 )
 8 NGINX_DEBUG = DerivedContainer(
 9     base=NGINX,
10     containerfile="RUN zypper -n in gdb nginx-debuginfo"
11 )
12
13 CONTAINER_IMAGES=[NGINX_DEBUG]
14
15 def test_nginx(auto_container): ...
```

Dependencies between containers

```
 1 TW = Container(
 2     url="registry.opensuse.org/opensuse/tumbleweed:latest"
 3 )
 4 NGINX = DerivedContainer(
 5     base=TW,
 6     containerfile="RUN zypper -n in nginx",
 7 )
 8 NGINX_DEBUG = DerivedContainer(
 9     base=NGINX,
10     containerfile="RUN zypper -n in gdb nginx-debuginfo"
11 )
12
13 CONTAINER_IMAGES=[NGINX_DEBUG]
14
15 def test_nginx(auto_container): ...
```

Dependencies between containers

```
 1 TW = Container(
 2     url="registry.opensuse.org/opensuse/tumbleweed:latest"
 3 )
 4 NGINX = DerivedContainer(
 5     base=TW,
 6     containerfile="RUN zypper -n in nginx",
 7 )
 8 NGINX_DEBUG = DerivedContainer(
 9     base=NGINX,
10     containerfile="RUN zypper -n in gdb nginx-debuginfo"
11 )
12
13 CONTAINER_IMAGES=[NGINX_DEBUG]
14
15 def test_nginx(auto_container): ...
```

Dependencies between containers

```
 1 TW = Container(
 2     url="registry.opensuse.org/opensuse/tumbleweed:latest"
 3 )
 4 NGINX = DerivedContainer(
 5     base=TW,
 6     containerfile="RUN zypper -n in nginx",
 7 )
 8 NGINX_DEBUG = DerivedContainer(
 9     base=NGINX,
10     containerfile="RUN zypper -n in gdb nginx-debuginfo"
11 )
12
13 CONTAINER_IMAGES=[NGINX_DEBUG]
14
15 def test_nginx(auto_container): ...
```

🔎 Inspect containers

```python
@pytest.mark.parametrize(
    "container", [MY_IMAGE], indirect=True
)
def test_inspect(container: ContainerData):
    inspect = container.inspect

    assert inspect.config.user == "me"
    assert inspect.config.cmd == ["/bin/sh"]

    assert (
        "HOME" in inspect.config.env
        and inspect.config.env["HOME"] == "/src/"
    )
```

🔎 Inspect containers

```
1  @pytest.mark.parametrize(
2      "container", [MY_IMAGE], indirect=True
3  )
4  def test_inspect(container: ContainerData):
5      inspect = container.inspect
6
7      assert inspect.config.user == "me"
8      assert inspect.config.cmd == ["/bin/sh"]
9
10     assert (
11         "HOME" in inspect.config.env
12         and inspect.config.env["HOME"] == "/src/"
13     )
```

🔎 Inspect containers

```python
@pytest.mark.parametrize(
    "container", [MY_IMAGE], indirect=True
)
def test_inspect(container: ContainerData):
    inspect = container.inspect

    assert inspect.config.user == "me"
    assert inspect.config.cmd == ["/bin/sh"]

    assert (
        "HOME" in inspect.config.env
        and inspect.config.env["HOME"] == "/src/"
    )
```

🔎 Inspect containers

```python
@pytest.mark.parametrize(
    "container", [MY_IMAGE], indirect=True
)
def test_inspect(container: ContainerData):
    inspect = container.inspect

    assert inspect.config.user == "me"
    assert inspect.config.cmd == ["/bin/sh"]

    assert (
        "HOME" in inspect.config.env
        and inspect.config.env["HOME"] == "/src/"
    )
```

🔎 Inspect containers

```python
@pytest.mark.parametrize(
    "container", [MY_IMAGE], indirect=True
)
def test_inspect(container: ContainerData):
    inspect = container.inspect

    assert inspect.config.user == "me"
    assert inspect.config.cmd == ["/bin/sh"]

    assert (
        "HOME" in inspect.config.env
        and inspect.config.env["HOME"] == "/src/"
    )
```

Container Volumes

Bind mounts

Container Volumes

Bind mounts

```
ROOTDIR_BIND_MOUNTED = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[
        BindMount("/src/", host_path=get_rootdir())
    ],
)


@pytest.mark.parametrize(
    "container", [ROOTDIR_BIND_MOUNTED], indirect=True
)
def test_bind_mount_cwd(container: ContainerData):
    vol = container.container.volume_mounts[0]
    assert container.connection.file("/src/").exists
```

Container Volumes

Bind mounts

```
ROOTDIR_BIND_MOUNTED = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[
        BindMount("/src/", host_path=get_rootdir())
    ],
)


@pytest.mark.parametrize(
    "container", [ROOTDIR_BIND_MOUNTED], indirect=True
)
def test_bind_mount_cwd(container: ContainerData):
    vol = container.container.volume_mounts[0]
    assert container.connection.file("/src/").exists
```

Container Volumes

Bind mounts

```
ROOTDIR_BIND_MOUNTED = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[
        BindMount("/src/", host_path=get_rootdir())
    ],
)


@pytest.mark.parametrize(
    "container", [ROOTDIR_BIND_MOUNTED], indirect=True
)
def test_bind_mount_cwd(container: ContainerData):
    vol = container.container.volume_mounts[0]
    assert container.connection.file("/src/").exists
```

Container Volumes

Bind mounts

```
ROOTDIR_BIND_MOUNTED = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[
        BindMount("/src/", host_path=get_rootdir())
    ],
)


@pytest.mark.parametrize(
    "container", [ROOTDIR_BIND_MOUNTED], indirect=True
)
def test_bind_mount_cwd(container: ContainerData):
    vol = container.container.volume_mounts[0]
    assert container.connection.file("/src/").exists
```

Container Volumes

Bind mounts

```
 1 ROOTDIR_BIND_MOUNTED = DerivedContainer(
 2     base="registry.opensuse.org/opensuse/tumbleweed",
 3     volume_mounts=[
 4         BindMount("/src/", host_path=get_rootdir())
 5     ],
 6 )
 7
 8
 9 @pytest.mark.parametrize(
10     "container", [ROOTDIR_BIND_MOUNTED], indirect=True
11 )
12 def test_bind_mount_cwd(container: ContainerData):
13     vol = container.container.volume_mounts[0]
14     assert container.connection.file("/src/").exists
```

Container Volumes

Bind mounts

```
ROOTDIR_BIND_MOUNTED = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[
        BindMount("/src/", host_path=get_rootdir())
    ],
)


@pytest.mark.parametrize(
    "container", [ROOTDIR_BIND_MOUNTED], indirect=True
)
def test_bind_mount_cwd(container: ContainerData):
    vol = container.container.volume_mounts[0]
    assert container.connection.file("/src/").exists
```

Container volumes

Container volumes

```
WITH_VAR_LOG_VOLUME = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[ContainerVolume("/var/log/")],
)
```

Container volumes

```
WITH_VAR_LOG_VOLUME = DerivedContainer(
    base="registry.opensuse.org/opensuse/tumbleweed",
    volume_mounts=[ContainerVolume("/var/log/")],
)
```

HEALTHCHECK

```
1  WEB_SERVER = DerivedContainer(
2      # snip
3      containerfile="""
4  ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
5  HEALTHCHECK CMD curl --fail http://0.0.0.0:8000""",
6  )
7
8
9  @pytest.mark.parametrize("container", [WEB_SERVER], indirect=T
10 def test_server_up(container, container_runtime):
11     assert (
12         container_runtime.get_container_health(
13             container.container_id
14         ) == ContainerHealth.HEALTHY
15     )
```

HEALTHCHECK

```
WEB_SERVER = DerivedContainer(
    # snip
    containerfile="""
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
HEALTHCHECK CMD curl --fail http://0.0.0.0:8000""",
)


@pytest.mark.parametrize("container", [WEB_SERVER], indirect=T
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.HEALTHY
    )
```

HEALTHCHECK

```
WEB_SERVER = DerivedContainer(
    # snip
    containerfile="""
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
HEALTHCHECK CMD curl --fail http://0.0.0.0:8000""",
)


@pytest.mark.parametrize("container", [WEB_SERVER], indirect=T
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.HEALTHY
    )
```

HEALTHCHECK

```
WEB_SERVER = DerivedContainer(
    # snip
    containerfile="""
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
HEALTHCHECK CMD curl --fail http://0.0.0.0:8000""",
)


@pytest.mark.parametrize("container", [WEB_SERVER], indirect=T
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.HEALTHY
    )
```

HEALTHCHECK

```
WEB_SERVER = DerivedContainer(
    # snip
    containerfile="""
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
HEALTHCHECK CMD curl --fail http://0.0.0.0:8000""",
)


@pytest.mark.parametrize("container", [WEB_SERVER], indirect=T
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.HEALTHY
    )
```

HEALTHCHECK

```
WEB_SERVER = DerivedContainer(
    # snip
    containerfile="""
ENTRYPOINT ["/usr/bin/python3", "-m", "http.server"]
HEALTHCHECK CMD curl --fail http://0.0.0.0:8000""",
)


@pytest.mark.parametrize("container", [WEB_SERVER], indirect=T
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.HEALTHY
    )
```

Don't wait for the health check

Don't wait for the health check

```
WEB_SERVER_2 = DerivedContainer(
    # snip
    healthcheck_timeout=timedelta(seconds=-1),
)


@pytest.mark.parametrize("container", [WEB_SERVER_2], indirect
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.STARTING
    )
```

Don't wait for the health check

```
WEB_SERVER_2 = DerivedContainer(
    # snip
    healthcheck_timeout=timedelta(seconds=-1),
)


@pytest.mark.parametrize("container", [WEB_SERVER_2], indirect
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.STARTING
    )
```

Don't wait for the health check

```
WEB_SERVER_2 = DerivedContainer(
    # snip
    healthcheck_timeout=timedelta(seconds=-1),
)


@pytest.mark.parametrize("container", [WEB_SERVER_2], indirect
def test_server_up(container, container_runtime):
    assert (
        container_runtime.get_container_health(
            container.container_id
        ) == ContainerHealth.STARTING
    )
```

Pick the Container Engine

```
export CONTAINER_RUNTIME=docker
pytest -vv
```

Run tests in parallel

```
pip install pytest-xdist
# or
poetry add --dev pytest-xdist

pytest -vv -- -n auto
```

Run tests in parallel

```
pip install pytest-xdist
# or
poetry add --dev pytest-xdist

pytest -vv -- -n auto
```

Run tests in parallel

```
pip install pytest-xdist
# or
poetry add --dev pytest-xdist

pytest -vv -- -n auto
```

Automatic cleanup

🧹 Automatic cleanup

- containers

 Automatic cleanup

- containers
- volumes

 Automatic cleanup

- containers
- volumes
- pods

 Automatic cleanup

- containers
- volumes
- pods
- temporary directories

🧹 Automatic cleanup

- containers
- volumes
- pods
- temporary directories
- ⚠️Images and intermediate layers are retained ⚠️

Users

Users

- BCI testsuite

Users

- BCI testsuite
- kiwi image builder scripts

Users

- BCI testsuite
- kiwi image builder scripts
- `obs-service-replace_using_package_version` integration tests

Users

- BCI testsuite
- kiwi image builder scripts
- `obs-service-replace_using_package_version` integration tests
- `obs-scm-bridge` integration tests

Users

- BCI testsuite
- kiwi image builder scripts
- `obs-service-replace_using_package_version` integration tests
- `obs-scm-bridge` integration tests
- `obs-service-node_modules` smoke test

Users

- BCI testsuite
- kiwi image builder scripts
- `obs-service-replace_using_package_version` integration tests
- `obs-scm-bridge` integration tests
- `obs-service-node_modules` smoke test
- Your project here?

Thanks!

- Jean-Philippe Evrard
- QE-C Team, especially José Lausuch and Felix Niederwanger

Give it a try!

 dcermak/pytest_container

 dcermak.github.io/pytest_container

 dcermak.github.io/pytest_container-presentation

What would you like to see?

What would you like to see?

👉 github.com/dcermak/pytest_container/issues

Questions?

Questions?

Answers!