From Source to OCI: Comparing s2i and Paketo Buildpacks

Alexandra Murariu (TE-MPE-CB)

07.11.2025

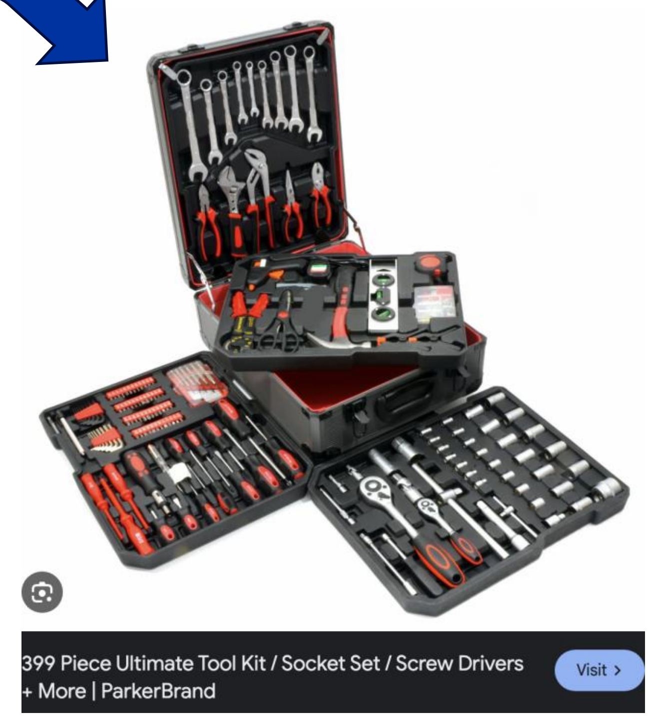
Handling containers

Since the dawn of time, we have been interested in running things in **isolation**, preferably not on a VM but something that requires fewer resources and is not as rigid.

Software has evolved from an obsession with **code**, to an obsession with **boxes**:

What if I have a Python app that uses Tensorflow and needs to run on a Mac, an Ubuntu server and a Windows machine?

We need a “one-size-fits-all” **universal box**



A brief history of containers

1979: **Unix V7** – chroot: the beginning of process isolation

2001: **Linux VServer**: jail mechanism that can partition resources (file systems, network addresses, memory) on a computer system

2006: **process containers** (control groups – cgroups): limiting, accounting and isolating resource usage of a collection of processes

2008: **LXC** (Linux containers): uses cgroups and Linux namespaces, and it works on a single Linux kernel without requiring any patches

2013: **Docker** (initially DotCloud) – the "portable container engine", used LXC initially as a container manager

2015: The **Open Container Initiative (OCI):** ensures containers are portable and interoperable – one image, any runtime, any platform.

Docker is OCI compliant, but is it enough?

This was a tough question we had to answer while working on a new data processing platform that uses **Knative** for having FaaS on the cloud. For some reason we did not understand why this Kubernetes native framework only accepted s2i or paketo as image builders for the functions, but not Docker.

The key is that more advanced builders like s2i or paketo buildpacks are **intelligent boxes**: they don’t only define the image as a static thing but can perform more complicated logic such as **reusable detection logic** of files, and you can essentially just give the builder your source code and it will be smart enough to “containerize” it **without writing any additional configuration**.

More on intelligent boxes (s2i vs paketo)

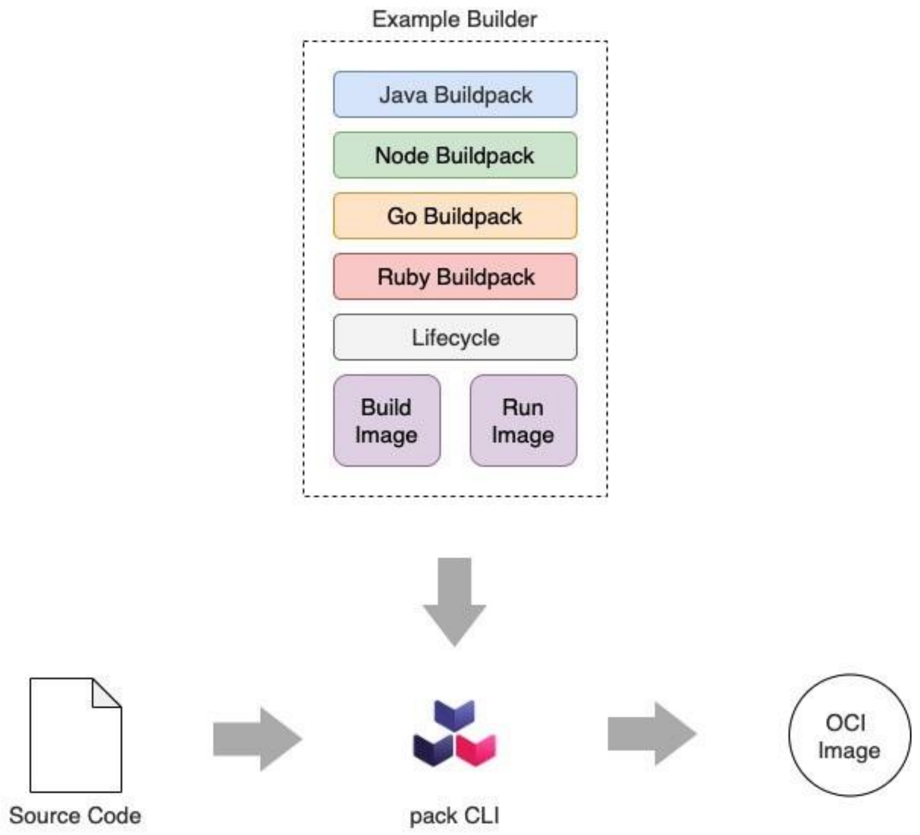
s2i

- Created by RedHat (2014)
= single-purpose builder

- Base language runtime + build tools
- Assemble script
    - Converts source code into runnable app
    - Downloads deps
    - Injects config
- Run script
    - Starts the app in the final image
    - This will be the RUN command
- Save-artifacts (optional)
    - Extracts reusable deps from prev image

paketo anatomy

- Part of Cloud Native Buildpacks (2020) - VMware
= chain of reusable, pluggable buildpacks that can detect and adapt to different apps automatically



s2i acc-py image

https://gitlab.cern.ch/acc-co/dpp/dpp-s2i-builder/-/tree/master?ref_type=heads

assemble.sh

- **copies** app code
- installs **dependencies** (from .toml or requirements file)

run.sh

- **runs the app** from the Procfile if present, otherwise the module

save-artifacts.sh

- **caches** common files to be used by future builds

paketo acc-py image

https://gitlab.cern.ch/acc-co/dpp/dpp-pack-builder

buildpacks

pip-install

pyproject-toml-install

procfile-run

	pip-install	pyproject-toml-install	procfile-run
detect	- Fail if no **requirements.txt**	- Fail if no **pyproject.toml**	- Fail if no **Procfile**
build	- Installs deps under **$CNB_LAYERS_DIR** - If nxcals_bundle is installed, points **SPARK_HOME** there	- Installs deps under **$CNB_LAYERS_DIR** - If nxcals_bundle is installed, points **SPARK_HOME** there - Retrieves module name, and creates an entrypoint in **$LAUNCH_TOML** which runs the module	- Creates an entrypoint in **$LAUNCH_TOML** with the contents of the Procfile

Builder.toml

Defines buildpacks, order of detection and base images for the build and run commands

Let’s dive in!

dive

release v0.13.1 | Validations passing | go report A+ | License MIT 2.0 | Donate PayPal

A tool for exploring a Docker image, layer contents, and discovering ways to shrink the size of your Docker/OCI image.

dive gitlab-registry.cern.ch/acc-co/dpp/dpp-s2i-builder/s2i-demo

```
│ ● Layers ├──────────────────────────────────────────────────────────────
Cmp   Size  Command
     228 MB  FROM blobs
      13 MB  /bin/sh
     896 MB  /bin/sh
     301 MB  COPY --from=acc_py_installed_to_disk /opt/acc-py/base /opt/acc-py/base
        0 B  RUN rm -rf /opt/acc-py/base/2023.06/ssl  && ln -s /etc/pki/tls /opt/acc-py/base/202
      52 MB  RUN python -m venv --system-site-packages $VIRTUAL_ENV     && echo "export PS1=\"(\
      12 MB  RUN /bin/sh -c pip install --upgrade pip # buildkit
     321 MB  RUN /bin/sh -c useradd -m -u 1000 -s /sbin/nologin default  && mkdir -p /opt/app-ro
     2.1 kB  COPY s2i/bin/ /usr/libexec/s2i # buildkit
        0 B  WORKDIR /opt/app-root/src
```

```
│ Image Details ├─────────────────────────────────────────────────────────

Image name: gitlab-registry.cern.ch/acc-co/dpp/dpp-s2i-builder/s2i-demo:latest
Total Image size: 1.8 GB
Potential wasted space: 778 MB
Image efficiency score: 76 %

Count   Total Space  Path
    3         76 MB  /opt/acc-py/base/2023.06/bin/python3.11
    2         73 MB  /opt/acc-py/base/2023.06/lib/libopenblasp-r0.3.25.so
    2         51 MB  /opt/acc-py/base/2023.06/lib/libpython3.11.so.1.0
    3         32 MB  /var/lib/rpm/rpmdb.sqlite
    2         28 MB  /opt/acc-py/base/2023.06/lib/libstdc++.so.6.0.33
    2         14 MB  /opt/acc-py/base/2023.06/lib/libgfortran.so.5.0.0
    2         13 MB  /opt/acc-py/base/2023.06/lib/python3.11/site-packages/numpy/core/_multiarr
ay_umath.cpython-311-x86_64-linux-gnu.so
```

dive gitlab-registry.cern.ch/acc-co/dpp/dpp-pack-builder

```
| ● Layers |
Cmp   Size  Command
     228 MB  FROM blobs
      13 MB
     896 MB
     301 MB
        0 B
      52 MB
     5.8 kB
        0 B
      32 MB
      985 B
     1.1 kB
     1.7 kB
      225 B
       95 B
       94 B
```

```
| Image Details |

Image name: gitlab-registry.cern.ch/acc-co/dpp/dpp-pack-builder
Total Image size: 1.5 GB
Potential wasted space: 156 MB
Image efficiency score: 94 %

Count   Total Space  Path
    2         51 MB  /opt/acc-py/base/2023.06/bin/python3.11
    3         32 MB  /var/lib/rpm/rpmdb.sqlite
    2        7.0 MB  /usr/lib64/libpython3.9.so.1.0
    2        4.7 MB  /var/lib/dnf/history.sqlite-wal
    2        2.2 MB  /usr/lib64/python3.9/lib-dynload/unicodedata.cpython-39-x86_64-linux-gnu.s
o
    2        872 kB  /usr/lib64/python3.9/pydoc_data/topics.pyc
    2        567 kB  /usr/lib64/python3.9/lib-dynload/_decimal.cpython-39-x86_64-linux-gnu.so
    2        548 kB  /usr/lib64/python3.9/lib-dynload/_codecs_jp.cpython-39-x86_64-linux-gnu.so
```

Even though both builders use the exact same base image, they write layers differently.
- S2I commits the entire container filesystem after building, duplicating large chunks of unchanged files.

- Paketo only exports the relevant build layers into a clean run image.

That's why, even with identical bases, S2I shows 76% efficiency and Paketo over 90%.

Bibliography

- History of containers: https://www.aquasec.com/blog/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016
- Docker and OCI: https://cloud.theodo.com/en/blog/container-docker-oci
- Nice talk by the inventor of Docker: https://www.youtube.com/watch?v=3N3n9FzebAA
- Dive tool: https://github.com/wagoodman/dive

home.cern