HPC Container Conformance

For lack of a better title… ;)

Christian Kniep, 2023-01-19

Christian Kniep
QNIB Solutions
Berlin area, Germany
info@qnib.org

This presentation represents the first stab on the topic.
With a limit number of contributors so far.

So feel free to get in touch and provide your input!

Conformance What!?

What are we trying to achieve?

Guidance!

- Collect ways of building container images for HPC use cases
- Derive best-practices on how to build and annotate a container
- Use best-practices and annotation and take the SystemAdmin perspective

Expectation Management in terms of Portability/Performance

- Container images might be specific to a system or generic; how to we guide folks what to expect?

Application we start with: GROMACS, PyTorch (, WRF?)

Conformance What!?

What are we NOT trying to achieve?

- We are NOT going to boil the ocean in that we try to make everything work!
- Allow for generic images and also optimised images that only run in specific environments. Annotations will make the expectations clear
- We are going to focus on OCI image. Most likely build with a Dockerfile.
- Dockerfile might be derived with another artefact: spack.env / HPCCM recipe

First Stab

What are we going to touch on?

1. How do we expect an image to behave?
    A. Specifically the ENTRYPOINT / CMD relationship
2. How to use annotations to inform end-users and sysadmins
    A. What the image file-system (user land) provides
    B. Inform the Sysadmin in which ways the image can be tweaked to fit the execution environment

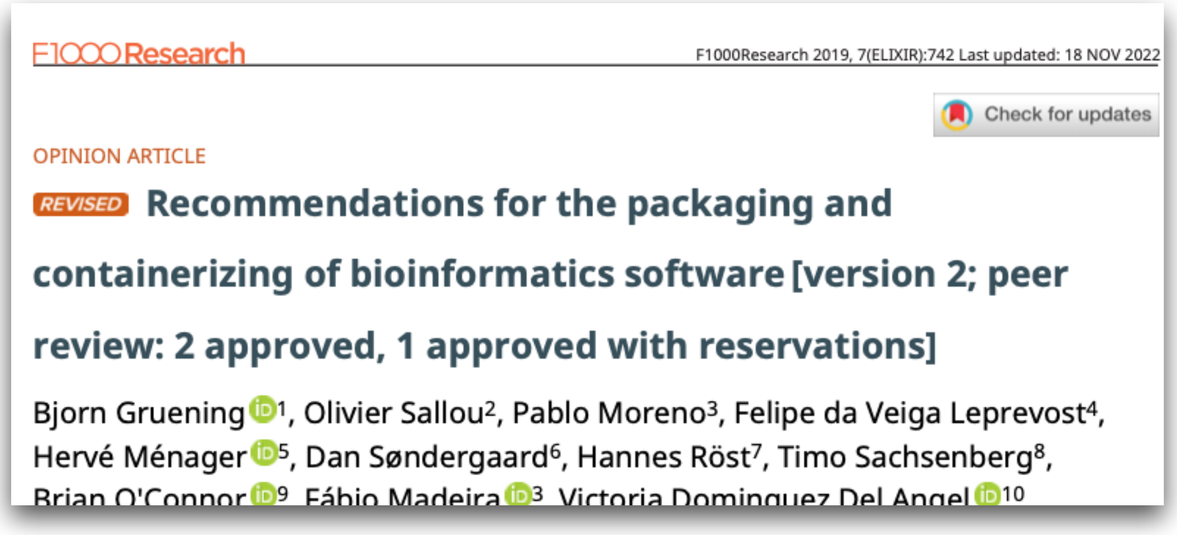
Biocontainer Community

Biocontainer Paper

Recommendations for the packaging and containerising of bioinformatics software

https://f1000research.com/articles/7-742/v2

1. A package first
2. One tool, one container
3. Tool and container versions should be explicit
4. Avoid using ENTRYPOINT
5. Reduce size as much as possible
6. Keep data outside of the container
7. Add functional testing logic
8. Check the license of the software
9. Make your package discoverable
10. Provide reproducible and documented builds
11. Provide helpful usage message



F1000Research F1000Research 2019, 7(ELIXIR):742 Last updated: 18 NOV 2022

Check for updates

OPINION ARTICLE

REVISED **Recommendations for the packaging and containerizing of bioinformatics software [version 2; peer review: 2 approved, 1 approved with reservations]**

Bjorn Gruening[1], Olivier Sallou[2], Pablo Moreno[3], Felipe da Veiga Leprevost[4], Hervé Ménager[5], Dan Søndergaard[6], Hannes Röst[7], Timo Sachsenberg[8], Brian O'Connor[9], Fábio Madeira[3], Victoria Dominguez Del Angel[10]

Expected Image Behaviour

Expected Image Bahavior

Login Container vs. Application Container

Some containers are used (or usable) as Application alias:

```
$ alias goreleaser="docker run -ti goreleaser/goreleaser"
$ goreleaser
GoReleaser is a release automation tool for Go projects.
Its goal is to simplify the build, release and publish steps while providing variant
customization options for all steps.
```

The above container uses the ENTRYPOINT to start the application in question. All arguments (CMDs) are arguments for the application itself.

Great for application aliases; but for HPC it hinders how the image can be used. Do I need to specify the application (say gmx for GROMACS) or do I start with the arguments?

Expected Image Bahavior

Login Container vs. Application Container

For HPC containers we expect to be dropped into a shell (most likely bash)

```
docker run -ti -v $(pwd):/data quay.io/cqnib/gromacs-2021.5_gcc-7.3.1:aarch64
bash-4.2#
```

The look and feel should be similar to logging into a compute node. The environment is prepared to have the application already at your fingertips.

```
$ docker run -ti -v $(pwd):/data -w /data \
     quay.io/cqnib/gromacs-2021.5_gcc-7.3.1:aarch64 gmx  mdrun -s benchRIB.tpr -resethway
                    :-) GROMACS - gmx mdrun, 2021.5-spack (-:
Using 1 MPI thread
Using 8 OpenMP threads
starting mdrun 'Protein'
10000 steps,     40.0 ps.
```

Expected Image Behavior

Login Container Image - ENTRYPOINT

The ENTRYPOINT should be as small as possible and setup the environment to tun the application.

- Source a bash profile to make the application and libraries available
- Apart from that as little runtime decisions as possible (if possible)
  (E.g. the upstream gromacs image has a gmx-choser to pick the best gmx binary - IMHO discouraged, even though it makes it portable)

Expected Image Behavior

Login Container Image - CMD

The CMD (default arguments provided to the container) might include an example run command (e.g. gmx) or print the usage message.

```
docker run -ti -v $(pwd):/data quay.io/cqnib/gromacs-2021.5_gcc-7.3.1:aarch64 gmx
gmx [-[no]h] [-[no]quiet] [-[no]version] [-[no]copyright] [-nice <int>]

Other options:
 -[no]h                     (no)
           Print help and quit
 -[no]quiet                 (no)
           Do not print common startup info or quotes
 -[no]version               (no)
           Print extended version information and quit
```

Annotations

Annotations/Labels?

What are annotations and labels?

Labels vs. Annotations

```
FROM alpine:3.17

LABEL org.supercontainers.hardware.cpu.optimized.mode=architecture
LABEL org.supercontainers.hardware.cpu.optimized.architecture=x86_64
$ docker build -q -t docker/label . && docker save -o - docker/label | tar xf - -C docker
sha256:e92da5e545b074220b9efbde8d4707c3e12c01001731b07f37a6787c5a53286f
$ cat docker/manifest.json|jq '.[]'
{
  "Config": "e92da5e545b074220b9efbde8d4707c3e12c01001731b07f37a6787c5a53286f.json",
  "RepoTags": [
    "docker/label:latest"
  ],
  "Layers": [
    "2a97e16e42c7f6ad7cb4c9796e0494e56f6faaccb7550818bd58d4d285564d71/layer.tar"
  ]
}
```

```
$ cat docker/e92da5e545b074220b9efbde8d4707c3e12c01001731b07f37a6787c5a53286f.json |jq .config.Labels
{
  "org.supercontainers.hardware.cpu.optimized.architecture": "x86_64",
  "org.supercontainers.hardware.cpu.optimized.mode": "architecture"
}
```

What are annotations and labels?

Labels vs. Annotations

```
$ docker buildx build -q . -o type=oci,dest=output.tar,name=label/x86_64 && tar xf output.tar -C oci
sha256:d610adf0edf8f95d75ae7608b6802f722ba9490f625a888a8e10275da0c3f4bb
$ cat oci/index.json|jq .
{
  "schemaVersion": 2,
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "digest": "sha256:d610adf0edf8f95d75ae7608b6802f722ba9490f625a888a8e10275da0c3f4bb",
      "size": 506,
      "annotations": {
        "io.containerd.image.name": "docker.io/label/x86_64:latest",
        "org.opencontainers.image.created": "2023-01-06T08:27:49Z",
        "org.opencontainers.image.ref.name": "latest"
      }
    }
  ]
}
$ cat oci/manifest.json| jq '.[] |.Config'
"blobs/sha256/6b8c20a910ca15d2a6bc4e059e900202075cd781829a8eabac9001ca3a275886"
```

What are annotations and labels?

- **Labels** are part of the image config and thus tied to a manifest
- **Annotations** are part of the OCI spec and can be attached to
  - Manifests (application/vnd.oci.image.manifest.v1+json)
  - Image Index (application/vnd.oci.image.index.v1+json)

Annotations Base Ideas

Container Annotations

What for?

Annotations will serves two purposes

1. Describe the image: SysAdmins and end users know what to expect
   A. What user-land is provided by the image itself?
   B. In which ways can the image be tweaked to make the most out of the execution environment (CPU μArch, GPU, MPI)?
   C. Configure hooks, runtimes to tweak the container correctly
   D. Provide a smoke test to fail fast: “Container will SEGFAULT!”
2. Inform end users what to look out for an execution environment
   A. Look out for mpich variants with ABI version xyz…

Container Annotations

Mandatory Annotations / Optional once

The following slides just list groups of annotations, but at the end we need to define:

A. Mandatory annotations (which CPU architecture the container is compiled)
   -> Without those the container is not considered ‘HPC Container Compliant’

B. Optional annotations (CUDA version, complete SBOM)
   -> depending on how far you want to go

Annotation Groups

Hardware Annotations

CPU/GPU/...

Information about what the application in the containers user-land is compiled for.

- Will the application segfault due to architecture mismatch (beyond the platform specification ARM/x86)?
- What CUDA version and GPU architecture is the application build against?

org.supercontainers.hardware.cpu.optimized.mode	architecture, genericMicro, microarchitecture
org.supercontainers.hardware.cpu.optimized.version	x86_64 / x86_64_v4 / skylake / skylake_avx512
org.supercontainers.hardware.gpu.nvidia.driver.version	346.34
org.supercontainers.hardware.gpu.nvidia.cuda.version	12.1
org.supercontainers.hardware.gpu.nvidia.architecture	sm_35 (kepler), sm_86 (ampere)

MPI/Interconnect Annotations

Information about what the user-land is compiled for and what methods to tweak the container is the container designed for?

org.supercontainers.mpi.implementation	(openmpi,mpich,threadmpi)
org.supercontainers.communication.framework	(ucx, libfabrics)
org.supercontainers.openmpi.version	1.16.1
org.supercontainers.libfabric.abi.version	1.6
org.supercontainers.mpi.portability.optimization	stock, cray-xc-cnl10
org.supercontainers.mpi.portability.mode	mpi_replace, libfabric_inject, ucx_replace

System Annotations

What can the user expect

Scripting Environment: What does the container carry to support scripts?

org.supercontainers.libc.implementation	glibc,musl
org.supercontainers.glibc.version	2.35
org.supercontainers.python.version	3.10
org.supercontainers.shell.implementation	bash,sh,zsh
org.supercontainers.tools.includes	jq,wget,awscli
org.supercontainers.path.extra	/usr/local/bin (empty dir already in PATH)
org.supercontainers.kernel.version	5.1

What is expected from the host system

org.supercontainers.host.kernel.version.min	5.1
org.supercontainers.host.kernel.modules.expectation	user-namespaces

Documentation Annotations

Further information

Hello-world example as minimalistic as possible

Benchmark how-to with meaningful, representative result

How to use the container?

org.supercontainers.docs.quickstart.link	https://external.website.org/how-to-gromacs
org.supercontainers.docs.quickstart.base64	base64-encoded-markdown
org.supercontainers.docs.benchmark.link	https://external.website.org/how-to-bench-gromacs
org.supercontainers.docs.benchmark.base64	base64-encoded-markdown

How to reproduce/tweak the container build

org.supercontainers.docs.build.dockerfile	base64-encoded-dockerfile
org.supercontainers.docs.build.spack.env	base64-encoded-spack.env
org.supercontainers.docs.build.quickstart.link	https://external.website.org/how-to-build-gromacs
org.supercontainers.docs.build.quickstart.base64	base64-encoded-markdown

How to annotate?

Layered Approach

Annotations might be added in multiple stages

- The **base image** might provide some basic annotations about the
  - Operating system, tools already installed, libraries, etc.
- While **building a subsequent image** new annotations can be made:
  - Application version, additional dependencies
- **After an image** is build we might annotate more information
  - Using tools like crane
  - Collect annotation of image URIs (gromacs/gromacs:2021.5) without changing/republishing the image

Build Tools

Ideally tools like Spack/Easybuild/HPCCM have this build in

- HPCCM already provides a way to annotate the resulting image:
  `Stage0 += openmpi(version='3.1.4', annotate=True)`
- Spack might add annotations in the resulting image (make it Todds' problem)
- EasyBuild and other might do the same

Benefit

- By offloading (basic) annotations to build tools would make it easy to get annotations in, w/o the user even thinking about it.

External Curation of Annotations

Without access/control over images, we might just collect them

A curated list of HPC images can annotate without changing the image.
E.g. using MetaHub Collections:

```yaml
manifests:
  - name: gromacs/mpich
    tag: 2021.5
    manifests:
      - image: quay.io/cqnib/gromacs/gcc-7.3.1/2021.5/mpich:x86_64_v4
        platform:
          os: linux
          arch: amd64
        annotations:
          org.supercontainers.mpi.provider: mpich
          org.supercontainers.mpich.abi.version: 12.0
          org.supercontainers.mpi.portability.optimized: stock
          org.supercontainers.mpi.portability.mode: mpich_replace
          org.supercontainers.mpich.version: 3.4
```

System Fingerprint

System Fingerprint

archspec++

To match the annotations within the manifest and image index, we need to create a fingerprint of the system. Some ideas:

1. Hardware: CPU (archspec), GPU, Interconnect
2. OS: Kernel ABI
3. Software: glibc, maybe something like PMIx version?
4. Runtimes: What runtimes are installed, how are they configured?

ReCap

Container Behaviour

Use a simple ENTRYPOINT to prepare the use of an application

1. A container ideally has binary(/binaries) and set of libraries compiled to support the execution of a given application. No runtime selection between different targets.

2. A container should drop into a shell (similar to when someone logs into a node)

3. The entry point should make as little runtime tweaks as possible

This will ensure that we can swap different images and still maintain the same behaviour (use the same submit script).

Annotations

Describing the image itself and how it expects to interact

A set of HPC specific image annotations are going to be the goal to describe:

1. The content of the user land of the container and what it is compiled for

2. How the container expects to be tweaked to utilise execution environment specifics: CPU micro-architecture, MPI implementations, GPUs

This will help end users and sys admins to discover images already optimised and how to configure execution environments to run different sets of images.

What’s Next?

Some Ideas First

Tooling to Assist

Helper tools to match system information with container annotations

Imaging a tool that takes information about the execution environment and the container annotations. It may provide a score:

1. Architecture smoke-test (before downloading every layer):
   Will this image even run or segfault along the way?

2. Expected performance ballpark:
   Given it matches the MPI ABI and the binary is compiled for the CPU…
   green: performance is near-baremetal (90+%)
   orange: performance is not optimal, but decent: (50+%)
   red: it won't run at all or with poor performance: (10% and below)

Automatic Benchmark

Run benchmarks suites informed by annotations

- Once we know how to run images given their annotations we might run benchmarks suites automagically…

Next Steps

Until next Advisory Meeting (2023/2/2)

Next Step #1: Collect Images

Add Images with Annotations to MetaHub Community Edition

- Collect/Build ‘well behaving’ images
- Add them to MetaHub CE over at Gitlab[1]

develop community-edition / collections / gromacs / mpich / 2021.5.yaml

2021.5.yaml 947 bytes

```yaml
manifests:
  - name: gromacs/mpich
    tag: 2021.5
    manifests:
      - image: quay.io/cqnib/gromacs/gcc-7.3.1/2021.5/mpich:x86_64_v4
        platform:
          os: linux
          arch: amd64
        annotations:
          org.supercontainers.mpi.provider: mpich
          org.supercontainers.mpich.abi.version: 12.0
          org.supercontainers.mpi.portability.optimized: stock
          org.supercontainers.mpi.portability.mode: mpich_replace
          org.supercontainers.mpich.version: 3.4
      - image: quay.io/cqnib/gromacs-2021.5_gcc-11.3.0:zen3
        platform:
          os: linux
          arch: amd64
          features: [cpu:zen3]
      - image: quay.io/cqnib/gromacs/gcc-7.3.1/2021.5/mpich:x86_64_v4_labeled
        platform:
          os: linux
          arch: amd64
          features: [cpu:zen2]
      - image: quay.io/cqnib/gromacs-2021.5_gcc-7.3.1:aarch64
        platform:
          os: linux
          arch: arm64
```

1: https://gitlab.com/qnib-metahub/community-edition

Next Step #2: Run Images at Sites

What are the challenges? What configuration can we converge towards?

Runtime/Engine

Fetch MPICH and Open MPI images of GROMACS and run it

1. Do we need to change the submit script based on the annotations OR
2. Can we use the same and have the runtime/engine adjust?

1: https://gitlab.com/qnib-metahub/community-edition

Next Step #3: Create Tooling

Fingerprint, matcher

System Fingerprint

A CLI (library and language bindings) to capture the fingerprint.
Can we translate a container fingerprint into annotations?

HPC Matcher

CLI which takes in a fingerprint and a set of annotations and outputs a score.

1: https://gitlab.com/qnib-metahub/community-edition

Thanks!

If you have questions or need consulting,
please reach out.

Christian Kniep
QNIB Solutions
Berlin area, Germany
info@qnib.org