Position and (proposed) submissions for container metadata standardization forum

International Workshop on FAIR Containerized Computational Software - Dec. 5th-7th, 2023

NRCFieldsA (Allan Fields), IT Consultant Unclassified 2023-11-30

About author: consultant to NRC-CNRC - Research Platform Support

- Support HPC clusters in NRC
- Organization: Government of Canada, National Research Council, Knowledge & Information Technology Services, Research Platform Support group
- Support scientific applications, builds&installs and HPC users jobs.
- Tasked with maintaining container images for research use, interest level includes:
  - Apptainer (Singularity), Docker;
  - Spack based software installs, typical HPC topics.

Position on container metadata standardization activity

- Could be quite helpful
- Needs to be done in a straight-forward way to be widely adopted
- Each site will have own requirements:
  - Some simpler use cases
  - Other’s more advanced
- There is a way to accommodate many use cases, in consistent way
- Simplified by using a modular approach
- Some work already exists, use that; add need metadata.

Approaches to compare

Relative merits: Pros & Cons, Implementation factors

- Metadata existing outside container vs. container metadata
- Metadata locality
- Practical requirements
- Simplicity and design wishes
- Ability to meet organization use cases
  - i.e. sufficiently expressive metadata
- In scope and out of scope of standard

Metadata Locality

- External metadata and existing formats for container orchestration, workflow and description

  vs.

- Container file metadata layers and OCI inline metadata (supported spec JSON)

  vs.

- References and external metadata pointers (extensions to container metadata)

  vs.

- All of (standard for each), none-of-the above (new mechanism)

Metadata Storage

Storage of container-specific metadata possible with:

- OCI – metadata stored with image artifact
  - accessible during image storage, use
- External file metadata file/stream referenced at run-time
  - inject metadata into container at run-time
- Maybe: metadata in first file system layer (for offline/builder use)?

Practical Design (Wishes)

- Common fields across multiple tooling, image formats – to comply with standard
- Fixed data types (JSON) and support array primitives (iteration on a single key in standard way)
  - Otherwise, some-one will invent it ad-hoc in 1 implementation (i.e. field split ambiguity)
  - Example: list of bind mounts – should take advantage of array iteration
  - Can still allow field sep, split in strings as per implementation, discouraged
- Complete listing of possible values (enum)
- All core metadata should be optional, handled gracefully if missing

Practical Design (Wishes) Cont’

- Metadata core type examples:
  - U_Int, Boolean (True|False), Enum {value [, ..]}, Text field as String (varchar),
  - U_Int (unique id) to support references (indexes)
  - Encoding to support UTF-8, core field names (keys and constrained values) in 7-bit ASCII
  - Built-in enum of valid values using list of unsigned int32:
    - Populates corresponding text string field (related key);
    - In-turn localization possible, while still maintaining unique id.
- Standard defines: constrained, limited and free-form value:
  - Fully constrained constant enumerated (possible values), including max length
  - Limited range of permitted well-formed values, after which point value is considered undefined or implementation specific, length can be specified
  - Free form field: standard does not constrain maximum length or values

Core metadata (field set)

Standard should have practicable set of fields & values for core container metadata:

- Currently list is defined in OCI image format, limited purpose.
- Domain-specific metadata should not be part of core metadata.
- Core fields should be sufficiently cross-domain (app agnostic)
  - Multiple container users, orgs and all areas of work;
  - Able to directly leverage core metadata, in common.
- Additional container metadata not contained in core fields, still accommodated by standard and handled gracefully.
  - Should not bloat standard w/ extraneous fields; or pose maintenance concerns; as core fields are a limited set, while supporting common mechanisms and handlers for addition metadata.

Reference Implementation

- Should include at least a few practical examples of implementing the standard, and working with container meta-data layer
- Languages such as: go, python, jq, javascript?
- Support basic tests for standard working-group
- Demonstrate metadata core operations: list, get, set
- Demonstrate metadata module operations: list, get, set
- Demonstrate metadata reference operations: list, get, set
- Demonstrate external metadata schema operations: list, get, set

Incremental filing of metadata

It may not be convenient to fill many fields at once (time, missing info, different people involved).

Instead support filing one group of fields at a time:

- During container build
- During container pull
- During container update
- During container layer op
- During container caching, storage (source)

Field discovery and auto-filling:

- Tools which implement standard to fill gap in collecting metadata on behalf of HPC admins, container users
- OCI registry addition of catalogue information to container metadata layer on pull
- Security plugin field handling and authoring
- Immutable and signed fields vs. update-able fields
- Indicator flag if field was last human written or machine written

Metadata stream concatenation

It should be possible to concatenate metadata streams from multiple sources and have it still remain valid and parse-able as a single metadata stream.

- Multiple artifacts; Includes (common metadata injection);
- External references to metadata templates and schemas;
- Command-line and environment over-rides;
- Caching ensures that a single consistent metadata per container will exist with values listed.

The core metadata should be stored in a single place.

Metadata version compatibility

- Standard is version and backward-compatible
- Revisions of a standard, so that it’s not just a one-shot
- Metadata needs standard version number field that never changes
- Instead of a moving target, provide similar to ABI stability (minor, major) version of standards to preserve metadata handling

Extensible

- Standard should not limit domain-specific applications and metadata use-cases. Instead, handle in a standard way.
- Metadata group of keys should be: set of fields in flat JSON stream, having common prefix followed by ‘.’ dot separator to denote member
  - layer.lic.type = gpl, layer.lic.date = ‘2023-12-05’,
  - layer.contrib.fullname = “Foo Bar”, layer.contrib.email = “foo.bar@some.tld”
- Metadata indexes (unique id field) allow intra-metadata cross-references to express relationships
- Optional: seek to provide dynamic back-refs for all metadata relationships generated during parse of complete stream (after concat)

Modular

- Start with core metadata about container itself
- Metadata modules (metamod) extend core with added sets of fields
- Metadata descriptor (desc) Intra-metadata descriptors build more representative sets of metadata referencing specific container attributes

Container dependency

The data-structure used should allow for sufficiently expressible linkage

- {0..N} and bidirectional links with URI (fetch or pull) pointers

Container A depends on {B,C,D} in workflow F[n]:

dep.requires.list: [

{name: B, local_id:container_id, pull: “pull-uri”, source: “source-uri”,

local_path:”/path/to/image-D.sif”, desc: “long name of B – base container”},

{name: C, local_id:container_id, pull: “pull-uri”, source: “source-uri”, local_path:”/path/to/docker/cache”, desc: “long name of C – code container”},

{name: D, local_id:container_id, pull: “pull-uri”, source: “source-uri”, local_path:”/path/to/image-D.sif”, desc: “long name of D – data container”}

]

Container {B,C,D} therefore if built at same time, contain a back reference to container A:

dep.consumer.list: [

{name: A, local_id:container_id, pull: “pull-uri”, source: “source-uri”,

local_path:”/path/to/image-A.sif”, desc: “long name of A – parent container for workflow”},

]

dep.workflow.list: [
{id: workflow_id,
local_path:”/path/to/workflow.yaml”,
pull_uri:”helm or other artifact uri”,
desc: “long free-text name of F[id] – workflow name for app: a in domain: b consuming data: d”
}
]

- Descriptive free-text field for application specific documentation of dependency

Metadata External References

- Use when inline container metadata references external metadata source
- Inclusion from another registry or artifact
- Field population:
  - Keys only
  - Values only
- Field caching store both key and value
- Metadata templates: populate unfilled keys (with null or default value)
- Metadata fetch handler (URI) → (protocol dispatch: &ext_fetch, &ext_stream_parse, &ext_field_populate) → &ext_field_cache

Metadata External Refs Cont’

- Adding metadata directly to container images/artifacts vs. existing separate: yaml file w/ orchestration info already grouping the containers:
  - Is disassociation of images and their supporting charts a risk in handling large volumes and numerous versions of differing containers?
  - Do containers miss a parent index file with metadata, or get alienated at run-time some how across systems?
- Is one a data container and the other an app container and thus – no formal chart exists, but is specified at run-time – in that case does a “container type” field need to be added to the relationship map?

Metadata Similar Fields

- Problem: a potential litany of similar sounding metadata fields.
- Standard can enforce the core set of field names (annotation keys).
- Equivalent fields could provide an “alias” or “alternate spelling” for a fields which is handled the same way by the standard
- Would be useful in cases where another set of annotations was already added to meta-data
- Could be defined in a small optional metadata_equiv[] block
- Possibilities include: [“str:”,”string1”]=>”string2”, [“glob:”,”glob1”]=>”string” and [“re:”,”regex”]=>”string” (N=>1 mappings)

Voting for core fields in standard

- There may be many ideas and finding which is the best common set for standard could be by vote. (Spreadsheet with each org adding their thumbs-up or +1/-1 and summing the total at end.)

- Potentially, preference in certain community for certain annotation names over others (even if both have same purpose).
- Narrowing down the list and selecting a field to use in standard could be done by having similar names grouped together.

Ontologies

Borrowing on idea of using schemas; Ontology support is just a special case of external reference and metadata templating:

- Existing work (container schemas)
- As proposed for bio containers, but cross-domain generalization sought:
  - So.. standard mechanism should reliably populate container metadata, regardless of schema (and schema URI – with-in reason) – document schema handling mechanism for future schema implementor.
  - The first handler should implement: Schema.org as previously posed in work group.
  - Not loosing domain-specific metadata -- out of desire for standard;
  - nor gaining single-use bloated fields that only apply to one type of container.
- The container metadata module should cache fields from a schema as a group (field set) rooted at namespace with common prefix:
  - (key names, valid values either match the traversal of the URI or have been abbreviated, mapped consistently)
- Generic field handling with mechanism using external references.

Categories: enumeration of common values

- Example of possible categories:
  - license.type: gpl, gplv2, gplv3, bsd, mit, cc-attrib4.0, copyright_allrights
  - retention.type: readonly, writable, writable_checkpoint, writable_tmp, writable_tmp_keep, writeable_cow, unknown
  - retention.time: clean_onexit, fixed, noexpire
  - container.category.int_arr: [1, 2, 3, 4, 5, 6, 7, [7,1], 8, …]
  - container.category.long.en: ”Base”, “Test”, “Astrophysics”, “Biology”, “Chemistry”, “Mathematics”, “Physics”, “Quantum Mechanics”, “Weather (Meteorology)”, …
    container.category.short.en: base, test, astro, bio, chem, math, phys, QM, weather, …
    - Note: int_arr[7] has only one u_int while [7,1] has an sub-type as QM is a sub-category of Physics
    - It could be possible to just flatten the list and use a random unique id in place of int_arr[] - JSON can support the recursion, so is it cleaner if the library implementing standard has this by default? Regardless – it needs to be in global online registry: once id is taken, it’s taken for good.

Metadata module handlers

- Metadata modules (metamod) proposed as way to support specific and expressive metadata:
  - with-out large increase in core metadata count with-in standard base
  - fields sets (groups of related metadata) handled atomically, sponsored for inclusion in addendum by interested party
  - Some metamod’s are proposed to ship with reference implementation:
    - Those needed for sought after functions: classification, authorship, pipelines, security
  - handlers consume and expose common API when dealing with metadata
  - Handlers implemented in any prog language, allow container tool to focus: implement only the required capabilities.

Metamod: path descriptor

- Annotation namespace: container.pathdesc[].*
- A generic path specific field set, containing elements suitable for describing what resides under one specific path, or file match
- Only seeks provide standard fields for describing paths – up to implementer how to handle them
- Contains contributor info for attestation purposes

Metamod: bind mount

- Annotation namespace: container.bindmount[].*
- Metadata specific to bind mounts: to be made available in container at run-time
- Indicates destination and default (last saved) source path and whether mandatory for container to run or not

Metamod: run-time stdopt

- Annotation namespace: container.stdopt.*
- Metadata specific to default standard options which can be request from the *container runtime*.
  - To be over-ridden by site configurations, user privilege level, passed arguments and other implementation specific run-time checks.
- Act as a “guide” on recommended default run guidance – do the right thing like enable a GPU type “--nv” or exposing the DISPLAY and XAUTHORITY.
- Possible to encapsulate site policy into container default args: e.x. do not mount home, but use --writable-tmpfs

Metamod: contributor

- Annotation namespace: ${index}.contrib.*
- Contains generic contributor metadata
- Users the contrib.id as index (primary key) for other metadata to refer to.
- Can be referenced by for example: layer, pathdesc, workflow, host_lib – between metadata fields for attribution
  - Container.pathdesc.contrib_id → ${index}.contrib → container.pathspec.contrib.fullname
  - A dangling reference is possible and must be checked for – implementer to ensure metadata is consistent with-in stream

Metamod: app descriptor

- Annotation namespace: container.app[].desc.*
- Application description fields, optionally added further to a container application's metadata.
- Fields are directly populated with-in specific app under the namespace
- Thus, no need to index into description with-in app metadata
- However, field set of annotations is optional and implementer do not need to add ability for handling specific application description fields to comply with core metadata specs
- It is possible however to add at will as many application and descriptions for entry points by referencing them from container.entry[].

Metamod: layer

- Annotation namespace: container.layer[].*
- Layer description fields, optionally added further to a container application’s metadata.
- Layer annotations including: contributor contact, source uri, modification, sha256sum, signature, license, sensitivity, retention, etc.
- Useful for validity and attestations.

Metamod: host library

- Annotation namespace: container.host_lib[].*
- Host library name, version, required, host and dest path.
- Symbols expected and whether this host_lib is required to run entry point.
- Reference to container.entry[] which the host library is used with or required for.

Metamod: external reference

- Annotation namespace: container.ext[].*
- Metadata descriptor for external references.
  - External domain version – Which version of schema
  - External domain URI – URI to base of schema
  - Domain map – mapping expressions (masks) for field populate
  - External domain oci_ref   - artifact to check with-in same tag
  - Standard handler name in implementation (plugin) to use for: fetch, parse, populate (field populate) and validity (check based on schema value constraints)

Metamod: entry point

- Annotation namespace: container.entry[].*
- Container entry point description fields, usage (example)
- Optionally, any number of application records referenced.
- Permitted arguments (getopt), bash completion spec
- Role of entry point (ontology reference)
- Interactive or scripted?

Metamod: client descriptor

- Annotation namespace: container.clientdesc[].*
- Container client information description fields, usage
- Client info annotations including:
  - User and full name, group and department, email
  - Template job, example usage on how to use container for client's HPC job or workload
  - Test case path or URI for validation of function
  - Environment-modules fields: name, short_info, help

Metamod: workflow descriptor

- Annotation namespace: container.workflow[].*
- Container workflow description fields, usage
- TBD – more work on this (pending community discussion on how to support)

Metamod: retention

- Annotation namespace: ${index}.retention[].*
- Retention policy description fields, and handler (plugin) names
- Contains generic metadata related to retention policy for referenced objects
- User the retention.id as index (primary key) for other metadata to refer to:
  - Can work: container wide, layer level, bind mount and path descriptor specific
  - Implementation specific handling but base set of standard fields

Metamod: archspec

- Annotation namespace: container.archspec[].*
- Container archspec fields, listing architectures the container supports in terms of included binaries
- Optional field to set warning or error on constraint failure

- archspec can reference local handler (plugin) in the implementation which calls the archspec library to parse host specific architecture information

Metamod: accelspec

Annotation namespace: container.accelspec[].*
Container accelerator (GPU, etc.) specification fields.

- Includes ability to specify: none, warn or error listing accelerators the container supports in terms of included binaries
- accelspec can reference local handler (plugin) in the implementation which calls the archspec library to parse host specific architecture information – does this work on gpu? – if not need accelspec

Questions?

Please see: GitHub project for Info on metadata proposal and open an issue to discuss or correct, contribute, etc. against this experimental proposal.

This proposal is not affiliated with OpenContainers.org or other projects implementing container metadata at this time and is for reference purposes only to establish feasibility and compare merits of various approaches.

GitHub: https://github.com/nrcfieldsa/containers-wg/

Three types of files primarily:

- yaml files with annotations (metadata fields list)
- md files with descriptions of each set of fields
- yaml files with acceptable values for fields

There is a directory for category data with same file formats.