# Open Container Initiative Image Format Specification

## Open Container Initiative

### Image Format Specification

This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.

The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.

### Table of Contents

- Notational Conventions
- Overview
    - Understanding the Specification
    - Media Types
- Content Descriptors
- Image Layout
- Image Manifest
- Image Index
- Filesystem Layers
- Image Configuration
- Annotations
- Conversion
- Considerations
    - Extensibility
    - Canonicalization

### Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997).

The key words "unspecified", "undefined", and "implementation-defined" are to be interpreted as described in the rationale for the C99 standard.

An implementation is not compliant if it fails to satisfy one or more of the MUST, MUST NOT, REQUIRED, SHALL, or SHALL NOT requirements for the protocols it implements. An implementation is compliant if it satisfies all the MUST, MUST NOT, REQUIRED, SHALL, and SHALL NOT requirements for the protocols it implements.

### Overview

At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final

runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.
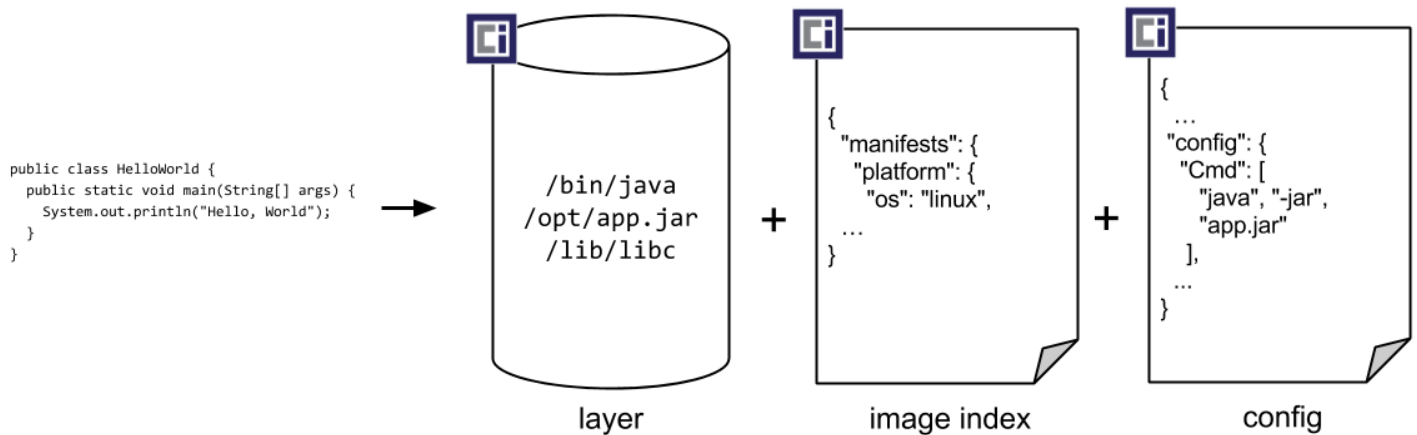


Once built the OCI Image can then be discovered by name, downloaded, verified by hash, trusted through a signature, and unpacked into an OCI Runtime Bundle.
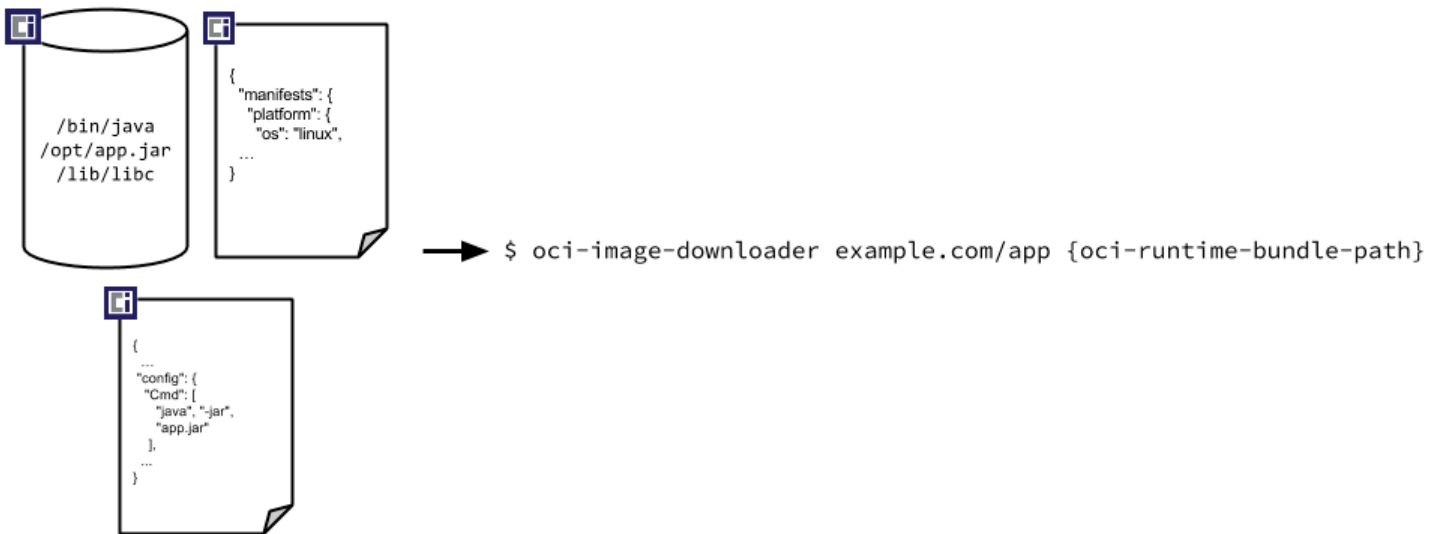


**Understanding the Specification**

The OCI Image Media Types document is a starting point to understanding the overall structure of the specification.

The high-level components of the spec include:

- Image Manifest - a document describing the components that make up a container image
- Image Index - an annotated list of manifests
- Artifact Manifest - a document describing the components that make up an artifact
- Image Layout - a filesystem layout representing the contents of an image
- Filesystem Layer - a changeset that describes a container's filesystem
- Image Configuration - a document determining layer ordering and configuration of the image suitable for translation into a [runtime bundle][runtime-spec]
- Conversion - a document describing how this translation should occur
- Descriptor - a reference that describes the type, metadata and content address of referenced content

Future versions of this specification may include the following OPTIONAL features:

- Signatures that are based on signing image content address
- Naming that is federated based on DNS and can be delegated

# OCI Image Media Types

The following media types identify the formats described here and their referenced resources:

- `application/vnd.oci.descriptor.v1+json`: Content Descriptor
- `application/vnd.oci.layout.header.v1+json`: OCI Layout
- `application/vnd.oci.image.index.v1+json`: Image Index
- `application/vnd.oci.image.manifest.v1+json`: Image manifest
- `application/vnd.oci.image.config.v1+json`: Image config
- `application/vnd.oci.image.layer.v1.tar`: "Layer", as a tar archive
- `application/vnd.oci.image.layer.v1.tar+gzip`: "Layer", as a tar archive compressed with [gzip][rfc1952]
- `application/vnd.oci.image.layer.v1.tar+zstd`: "Layer", as a tar archive compressed with [zstd][rfc8478]
- `application/vnd.oci.image.layer.nondistributable.v1.tar`: "Layer", as a tar archive with distribution restrictions
- `application/vnd.oci.image.layer.nondistributable.v1.tar+gzip`: "Layer", as a tar archive with distribution restrictions compressed with [gzip][rfc1952]
- `application/vnd.oci.image.layer.nondistributable.v1.tar+zstd`: "Layer", as a tar archive with distribution restrictions compressed with [zstd][rfc8478]
- `application/vnd.oci.artifact.manifest.v1+json`: Artifact manifest

## Media Type Conflicts

Blob retrieval methods MAY return media type metadata. For example, a HTTP response might return a manifest with the Content-Type header set to `application/vnd.oci.image.manifest.v1+json`. Implementations MAY also have expectations for the blob's media type and digest (e.g. from a descriptor referencing the blob).

- Implementations that do not have an expected media type for the blob SHOULD respect the returned media type.
- Implementations that have an expected media type which matches the returned media type SHOULD respect the matched media type.
- Implementations that have an expected media type which does not match the returned media type SHOULD:
  - Respect the expected media type if the blob matches the expected digest. Implementations MAY warn about the media type mismatch.
  - Return an error if the blob does not match the expected digest (as recommended for descriptors).
  - Return an error if they do not have an expected digest.

## Compatibility Matrix

The OCI Image Specification strives to be backwards and forwards compatible when possible. Breaking compatibility with existing systems creates a burden on users whether they are build systems, distribution systems, container engines, etc. This section shows where the OCI Image Specification is compatible with formats external to the OCI Image and different versions of this specification.

**application/vnd.oci.image.index.v1+json**

**Similar/related schema**

- application/vnd.docker.distribution.manifest.list.v2+json
  - `.annotations`: only present in OCI
  - `.[]manifests.annotations`: only present in OCI
  - `.[]manifests.urls`: only present in OCI

**application/vnd.oci.image.manifest.v1+json**

**Similar/related schema**

- application/vnd.docker.distribution.manifest.v2+json
    - `.annotations`: only present in OCI
    - `.config.annotations`: only present in OCI
    - `.config.urls`: only present in OCI
    - `.[]layers.annotations`: only present in OCI

**application/vnd.oci.image.layer.v1.tar+gzip**

**Interchangeable and fully compatible mime-types**

- [application/vnd.docker.image.rootfs.diff.tar.gzip](https://github.com

# OCI Content Descriptors

- An OCI image consists of several different components, arranged in a Merkle Directed Acyclic Graph (DAG).
- References between components in the graph are expressed through *Content Descriptors*.
- A Content Descriptor (or simply *Descriptor*) describes the disposition of the targeted content.
- A Content Descriptor includes the type of the content, a content identifier (*digest*), and the byte-size of the raw content. Optionally, it includes the type of artifact it is describing.
- Descriptors SHOULD be embedded in other formats to securely reference external content.
- Other formats SHOULD use descriptors to securely reference external content.

This section defines the `application/vnd.oci.descriptor.v1+json` media type.

## Properties

A descriptor consists of a set of properties encapsulated in key-value fields.

The following fields contain the primary properties that constitute a Descriptor:

- `mediaType` *string*

  This REQUIRED property contains the media type of the referenced content. Values MUST comply with [RFC 6838][rfc6838], including the [naming requirements in its section 4.2][rfc6838-s4.2].

  The OCI image specification defines several of its own MIME types for resources defined in the specification.

- `digest` *string*

  This REQUIRED property is the *digest* of the targeted content, conforming to the requirements outlined in Digests. Retrieved content SHOULD be verified against this digest when consumed via untrusted sources.

- `size` *int64*

  This REQUIRED property specifies the size, in bytes, of the raw content. This property exists so that a client will have an expected size for the content before processing. If the length of the retrieved content does not match the specified length, the content SHOULD NOT be trusted.

- `urls` *array of strings*

  This OPTIONAL property specifies a list of URIs from which this object MAY be downloaded. Each entry MUST conform to [RFC 3986][rfc3986]. Entries SHOULD use the `http` and `https` schemes, as defined in [RFC 7230][rfc7230-s2.7].

- **annotations** *string-string map*

  This OPTIONAL property contains arbitrary metadata for this descriptor. This OPTIONAL property MUST use the annotation rules.

- **data** *string*

  This OPTIONAL property contains an embedded representation of the referenced content. Values MUST conform to the Base 64 encoding, as defined in [RFC 4648][rfc4648-s4]. The decoded data MUST be identical to the referenced content and SHOULD be verified against the `digest` and `size` fields by content consumers. See Embedded Content for when this is appropriate.

- **artifactType** *string*

  This OPTIONAL property contains the type of an artifact when the descriptor points to an artifact. This is the value of `artifactType` when the descriptor references an artifact manifest. This is the value of the config descriptor `mediaType` when the descriptor references an image manifest.

Descriptors pointing to `application/vnd.oci.image.manifest.v1+json` SHOULD include the extended field `platform`, see Image Index Property Descriptions for details.

Descriptors pointing to `application/vnd.oci.artifact.manifest.v1+json` SHOULD include the extended field `artifactType`.

**Reserved**

Extended *Descriptor* field additions proposed in other OCI specifications SHOULD first be considered for addition into this specification.

## Digests

The *digest* property of a Descriptor acts as a content identifier, enabling content addressability. It uniquely identifies content by taking a collision-resistant hash of the bytes. If the *digest* can be communicated in a secure manner, one can verify content from an insecure source by recalculating the digest independently, ensuring the content has not been modified.

The value of the `digest` property is a string consisting of an *algorithm* portion and an *encoded* portion. The *algorithm* specifies the cryptographic hash function and encoding used for the digest; the *encoded* portion contains the encoded result of the hash function.

A digest string MUST match the following grammar:

```
digest                 ::= algorithm ":" encoded
algorithm              ::= algorithm-component (algorithm-separator algorithm-component)*
algorithm-component    ::= [a-z0-9]+
algorithm-separator    ::= [+._-]
encoded                ::= [a-zA-Z0-9=_-]+
```

Note that *algorithm* MAY impose algorithm-specific restriction on the grammar of the *encoded* portion. See also Registered Algorithms.

Some example digest strings include the following:

| digest | algorithm | Registered |
|---|---|---|
| sha256:6c3c624b58dbbcd3c0dd82b4c53f04194d1247c6eebdaab7c610cf7d66709b3b | SHA-256 | Yes |
| sha512:401b09eab3c013d4ca54922bb802bec8fd5318192b0a75f201d8b372742... | SHA-512 | Yes |
| multihash+base58:QmRZxt2b1FVZPNqd8hsiykDL3TdBDeTSPX9Kv46HmX4Gx8 | Multihash | No |
| sha256+b64u:LCa0a2j_xo_5m0U8HTBBNBNCLXBkg7-g-YpeiGJm564 | SHA-256 with urlsafe base64 | No |

Please see Registered Algorithms for a list of registered algorithms.

Implementations SHOULD allow digests with unrecognized algorithms to pass validation if they comply with the above grammar. While `sha256` will only use hex encoded digests, separators in *algorithm* and alphanumerics in *encoded* are included to allow for extensions. As an example, we can parameterize the encoding and algorithm as `multihash+base58:QmRZxt2b1FVZPNqd8hsiykDL3TdBDeTSPX9Kv46HmX4Gx8`, which would be considered valid but unregistered by this specification.

## Verification

Before consuming content targeted by a descriptor from untrusted sources, the byte content SHOULD be verified against the digest string. Before calculating the digest, the size of the content SHOULD be verified to reduce hash collision space. Heavy processing before calculating a hash SHOULD be avoided. Implementations MAY employ canonicalization of the underlying content to ensure stable content identifiers.

## Digest calculations

A *digest* is calculated by the following pseudo-code, where `H` is the selected hash algorithm, identified by string `<alg>`:

```
let ID(C) = Descriptor.digest
let C = <bytes>
let D = '<alg>:' + Encode(H(C))
let verified = ID(C) == D
```

Above, we define the content identifier as `ID(C)`, extracted from the `Descriptor.digest` field. Content `C` is a string of bytes. Function `H` returns the hash of `C` in bytes and is passed to function `Encode` and prefixed with the algorithm to obtain the digest. The result `verified` is true if `ID(C)` is equal to `D`, confirming that `C` is the content identified by `D`. After verification, the following is true:

```
D == ID(C) == '<alg>:' + Encode(H(C))
```

The *digest* is confirmed as the content identifier by independently calculating the *digest*.

## Registered algorithms

While the *algorithm* component of the digest string allows the use of a variety of cryptographic algorithms, compliant implementations SHOULD use SHA-256.

The following algorithm identifiers are currently defined by this specification:

| algorithm identifier | algorithm |
|---|---|
| sha256 | SHA-256 |
| sha512 | SHA-512 |

If a useful algorithm is not included in the above table, it SHOULD be submitted to this specification for registration.

## SHA-256

[SHA-256][rfc4634-s4.1] is a collision-resistant hash function, chosen for ubiquity, reasonable size and secure characteristics. Implementations MUST implement SHA-256 digest verification for use in descriptors.

When the *algorithm identifier* is `sha256`, the *encoded* portion MUST match `/[a-f0-9]{64}/`. Note that `[A-F]` MUST NOT be used here.

## SHA-512

[SHA-512][rfc4634-s4.2] is a collision-resistant hash function which [may be more performant][sha256-vs-sha512] than SHA-256 on some CPUs. Implementations MAY implement SHA-512 digest verification for use in descriptors.

When the *algorithm identifier* is `sha512`, the *encoded* portion MUST match `/[a-f0-9]{128}/`. Note that `[A-F]` MUST NOT be used here.

## Embedded Content

In many contexts, such as when downloading content over a network, resolving a descriptor to its content has a measurable fixed "roundtrip" latency cost. For large blobs, the fixed cost is usually inconsequential, as the majority of time will be spent actually fetching the content. For very small blobs, the fixed cost can be quite significant.

Implementations MAY choose to embed small pieces of content directly within a descriptor to avoid roundtrips.

Implementations MUST NOT populate the `data` field in situations where doing so would modify existing content identif

## OCI Image Layout Specification

- The OCI Image Layout is the directory structure for OCI content-addressable blobs and location-addressable references (refs).
- This layout MAY be used in a variety of different transport mechanisms: archive formats (e.g. tar, zip), shared filesystem environments (e.g. nfs), or networked file fetching (e.g. http, ftp, rsync).

Given an image layout and a ref, a tool can create an OCI Runtime Specification bundle by:

- Following the ref to find a manifest, possibly via an image index
- Applying the filesystem layers in the specified order
- Converting the image configuration into an OCI Runtime Specification `config.json`

# Content

The image layout is as follows:

- `blobs` directory
    - Contains content-addressable blobs
    - A blob has no schema and SHOULD be considered opaque
    - Directory MUST exist and MAY be empty
    - See blobs section
- `oci-layout` file
    - It MUST exist
    - It MUST be a JSON object
    - It MUST contain an `imageLayoutVersion` field
    - See oci-layout file section
    - It MAY include additional fields
- `index.json` file
    - It MUST exist
    - It MUST be an image index JSON object.
    - See index.json section

## Example Layout

This is an example image layout:

```
$ cd example.com/app/
$ find . -type f
./index.json
./oci-layout
./blobs/sha256/3588d02542238316759cbf24502f4344ffcc8a60c803870022f335d1390c13b4
./blobs/sha256/4b0bc1c4050b03c95ef2a8e36e25feac42fd31283e8c30b3ee5df6b043155d3c
./blobs/sha256/7968321274dc6b6171697c33df7815310468e694ac5be0ec03ff053bb135e768
```

Blobs are named by their contents:

```
$ shasum -a 256 ./blobs/sha256/afff3924849e458c5ef237db5f89539274d5e609db5db935ed3959c90f1f2d51
afff3924849e458c5ef237db5f89539274d5e609db5db935ed3959c90f1f2d51  ./blobs/sha256/afff3924849e458c5ef237db5f89539
```

## Blobs

- Object names in the `blobs` subdirectories are composed of a directory for each hash algorithm, the children of which will contain the actual content.
- The content of `blobs/<alg>/<encoded>` MUST match the digest `<alg>:<encoded>` (referenced per descriptor). For example, the content of `blobs/sha256/da39a3ee5e6b4b0d3255bfef95601890afd80709` MUST match the digest `sha256:da39a3ee5e6b4b0d3255bfef95601890afd80709`.
- The character set of the entry name for `<alg>` and `<encoded>` MUST match the respective grammar elements described in descriptor.
- The blobs directory MAY contain blobs which are not referenced by any of the refs.
- The blobs directory MAY be missing referenced blobs, in which case the missing blobs SHOULD be fulfilled by an external blob store.

### Example Blobs

```
$ cat ./blobs/sha256/9b97579de92b1c195b85bb42a11011378ee549b02d7fe9c17bf2a6b35d5cb079 | jq
{
  "schemaVersion": 2,
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "size": 7143,
      "digest": "sha256:afff3924849e458c5ef237db5f89539274d5e609db5db935ed3959c90f1f2d51",
      "platform": {
        "architecture": "ppc64le",
        "os": "linux"
      }
    },
...
```

```
$ cat ./blobs/sha256/afff3924849e458c5ef237db5f89539274d5e609db5db935ed3959c90f1f2d51 | jq
{
  "schemaVersion": 2,
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "size": 7023,
    "digest": "sha256:5b0bcabd1ed22e9fb1310cf6c2dec7cdef19f0ad69efa1f392e94a4333501270"
  },
  "layers": [
```

```
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 32654,
      "digest": "sha256:9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0"
    },
...


$ cat ./blobs/sha256/5b0bcabd1ed22e9fb1310cf6c2dec7cdef19f0ad69efa1f392e94a4333501270 | jq
{
  "architecture": "amd64",
  "author": "Alyssa P. Hacker <alyspdev@example.com>",
  "config": {
    "Hostname": "8dfe43d80430",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": null,
    "Image": "sha256:6986ae504bbf843512d680cc959484452034965db15f75ee8bdd1b107f61500b",
...


$ cat ./blobs/sha256/9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0
[gzipped tar stream]
```

### oci-layout file

This JSON object serves as a marker for the base of an Open Container Image Layout and to provide the version of the image-layout in use. The `imageLayoutVersion` value will align with the OCI Image Specification version at the time changes to the layout are made, and will pin a given version until changes to the image layout are required. This section defines the `application/vnd.oci.layout.header.v1+json` media type.

### oci-layout Example

```
{
    "imageLayoutVersion": "1.0.0"
}
```

### index.json file

This REQUIRED file is the entry point for references and descriptors of the image-layout. The image index is a multi-descriptor entry point.

This index provides an established path (`/index.json`) to have an entry point for an image-layout and to discover auxiliary descriptors.

- No semantic restriction is given for the "org.opencontainers.image.ref.name" annotation of descriptors.
- In general the `mediaType` of each [descriptor][descriptors] object in the `manifests` field will be either `application/vnd.oci.imag` or `application/vnd.oci.image.manifest.v1+json`.
- Future versions of the spec MAY use a different mediatype (i.e. a new versioned format).

- An encountered `mediaType` that is unknown SHOULD be safely ignored.

**Implementor's Note:** A common use case of descriptors with a "org.opencontainers.image.ref.name" annotation is representing a "tag" for a container image. For example, an image may have a tag for different versions or builds of the software. In the wild you often see "tags" like "v1.0.0-vendor.0", "2.0.0-debug", etc. Those tags will often be represented in an image-layout repository with matching "org.opencontainers.image.ref.name" annotations like "v1.0.0-vendor.0", "2.0.0-debug", etc.

**Index Example**

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.index.v1+json",
      "size": 7143,
      "digest": "sha256:0228f90e926ba6b96e4f39cf294b2586d38fbb5a1e385c05cd1ee40ea54fe7fd",
      "annotations": {
        "org.opencontainers.image.ref.name": "stable-release"
      }
    },
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "size": 7143,
      "digest": "sha256:e692418e4cbaf90ca69d05a66403747baa33ee08806650b51fab815ad7fc331f",
      "platform": {
        "architecture": "ppc64le",
        "os": "linux"
      },
      "annotations": {
        "org.opencontainers.image.ref.name": "v1.0"
      }
    },
    {
      "mediaType": "application/xml",
      "size": 7143,
      "digest": "sha256:b3d63d132d21c3ff4c35a061adf23cf43da8ae054247e32faa95494d904a007e",
      "annotations": {
        "org.freedesktop.specifications.metainfo.version": "1.0",
        "org.freedesktop.specifications.metainfo.type": "AppStream"
      }
    }
  ],
  "annotations": {
    "com.example.index.revision": "r124356"
  }
}
```

This illustrates an index that provides two named references and an auxiliary mediatype for

# OCI Image Manifest Specification

There are three main goals of the Image Manifest Specification. The first goal is content-addressable images, by supporting an image model where the image's configuration can be hashed to generate a unique ID for the image and its components.

The second goal is to allow multi-architecture images, through a "fat manifest" which references image manifests for platform-specific versions of an image. In OCI, this is codified in an image index. The third goal is to be translatable to the OCI Runtime Specification.

This section defines the `application/vnd.oci.image.manifest.v1+json` media type. For the media type(s) that this is compatible with see the matrix.

# Image Manifest

Unlike the image index, which contains information about a set of images that can span a variety of architectures and operating systems, an image manifest provides a configuration and set of layers for a single container image for a specific architecture and operating system.

## *Image Manifest* Property Descriptions

- **schemaVersion** *int*

  This REQUIRED property specifies the image manifest schema version. For this version of the specification, this MUST be 2 to ensure backward compatibility with older versions of Docker. The value of this field will not change. This field MAY be removed in a future version of the specification.

- **mediaType** *string*

  This property SHOULD be used and remain compatible with earlier versions of this specification and with other similar external formats. When used, this field MUST contain the media type `application/vnd.oci.image.manifest.v1+json`. This field usage differs from the descriptor use of `mediaType`.

- **config** *descriptor*

  This REQUIRED property references a configuration object for a container, by digest. Beyond the descriptor requirements, the value has the following additional restrictions:

  - **mediaType** *string*

    This descriptor property has additional restrictions for `config`. Implementations MUST support at least the following media types:

    * `application/vnd.oci.image.config.v1+json`

  Manifests concerned with portability SHOULD use one of the above media types.

- **layers** *array of objects*

  Each item in the array MUST be a descriptor. The array MUST have the base layer at index 0. Subsequent layers MUST then follow in stack order (i.e. from `layers[0]` to `layers[len(layers)-1]`). The final filesystem layout MUST match the result of applying the layers to an empty directory. The ownership, mode, and other attributes of the initial empty directory are unspecified.

  Beyond the descriptor requirements, the value has the following additional restrictions:

  - **mediaType** *string*

    This descriptor property has additional restrictions for `layers[]`. Implementations MUST support at least the following media types:

    * `application/vnd.oci.image.layer.v1.tar`
    * `application/vnd.oci.image.layer.v1.tar+gzip`
    * `application/vnd.oci.image.layer.nondistributable.v1.tar`
    * `application/vnd.oci.image.layer.nondistributable.v1.tar+gzip`

  Manifests concerned with portability SHOULD use one of the above media types. An encountered `mediaType` that is unknown to the implementation MUST be ignored.

  Entries in this field will frequently use the `+gzip` types.

- **refers** *descriptor*

  This OPTIONAL property specifies a descriptor of another manifest. This value, used by the `referrers` API, indicates a relationship to the specified manifest.

- **annotations** *string-string map*

  This OPTIONAL property contains arbitrary metadata for the image manifest. This OPTIONAL property MUST use the annotation rules.

  See Pre-Defined Annotation Keys.

## Example Image Manifest

*Example showing an image manifest:*

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.manifest.v1+json",
  "config": {
    "mediaType": "application/vnd.oci.image.config.v1+json",
    "size": 7023,
    "digest": "sha256:b5b2b2c507a0944348e0303114d8d93aaaa081732b86451d9bce1f432a537bc7"
  },
  "layers": [
    {
      "mediaType": "application/vnd.oci.image.layer.v1.tar+gzip",
      "size": 32654,
      "digest": "sha256:9834876dcfb05cb167a5c24953eba58c4ac89b1adf57f28f2f9d09af107ee8f0"
    },
    {
```

```
# OCI Image Index Specification

The image index is a higher-level manifest which points to specific [image manifests](manifest.md), ideal for
While the use of an image index is OPTIONAL for image providers, image consumers SHOULD be prepared to process

This section defines the `application/vnd.oci.image.index.v1+json` [media type](media-types.md).

For the media type(s) that this document is compatible with, see the [matrix][matrix].

## *Image Index* Property Descriptions

- **`schemaVersion`** *int*

  This REQUIRED property specifies the image manifest schema version.
  For this version of the specification, this MUST be `2` to ensure backward compatibility with older versions
  The value of this field will not change.
  This field MAY be removed in a future version of the specification.

- **`mediaType`** *string*

  This property SHOULD be used and [remain compatible][matrix] with earlier versions of this specification and
  When used, this field MUST contain the media type `application/vnd.oci.image.index.v1+json`.
  This field usage differs from the [descriptor](descriptor.md#properties) use of `mediaType`.

- **`manifests`** *array of objects*

  This REQUIRED property contains a list of [manifests](manifest.md) for specific platforms.
  While this property MUST be present, the size of the array MAY be zero.
```

Each object in `manifests` includes a set of [descriptor properties](descriptor.md#properties) with the foll

- **`mediaType`** *string*

  This [descriptor property](descriptor.md#properties) has additional restrictions for `manifests`.
  Implementations MUST support at least the following media types:

  - [`application/vnd.oci.image.manifest.v1+json`](manifest.md)
  - [`application/vnd.oci.artifact.manifest.v1+json`](artifact.md)

  Also, implementations SHOULD support the following media types:

  - `application/vnd.oci.image.index.v1+json` (nested index)

  Image indexes concerned with portability SHOULD use one of the above media types.
  Future versions of the spec MAY use a different mediatype (i.e. a new versioned format).
  An encountered `mediaType` that is unknown to the implementation MUST be ignored.

- **`platform`** *object*

  This OPTIONAL property describes the minimum runtime requirements of the image.
  This property SHOULD be present if its target is platform-specific.

  - **`architecture`** *string*

    This REQUIRED property specifies the CPU architecture.
    Image indexes SHOULD use, and implementations SHOULD understand, values listed in the Go Language docu

  - **`os`** *string*

    This REQUIRED property specifies the operating system.
    Image indexes SHOULD use, and implementations SHOULD understand, values listed in the Go Language docu

  - **`os.version`** *string*

    This OPTIONAL property specifies the version of the operating system targeted by the referenced blob.
    Implementations MAY refuse to use manifests where `os.version` is not known to work with the host OS v
    Valid values are implementation-defined. e.g. `10.0.14393.1066` on `windows`.

  - **`os.features`** *array of strings*

    This OPTIONAL property specifies an array of strings, each specifying a mandatory OS feature.
    When `os` is `windows`, image indexes SHOULD use, and implementations SHOULD understand the following

    - `win32k`: image requires `win32k.sys` on the host (Note: `win32k.sys` is missing on Nano Server)

    When `os` is not `windows`, values are implementation-defined and SHOULD be submitted to this specifica

  - **`variant`** *string*

    This OPTIONAL property specifies the variant of the CPU.
    Image indexes SHOULD use, and implementations SHOULD understand, `variant` values listed in the [Platf

  - **`features`** *array of strings*

    This property is RESERVED for future versions of the specification.

13

If multiple manifests match a client or runtime's requirements, the first matching entry SHOULD be used.

- **`annotations`** *string-string map*

    This OPTIONAL property contains arbitrary metadata for the image index.
    This OPTIONAL property MUST use the [annotation rules](annotations.md#rules).

    See [Pre-Defined Annotation Keys](annotations.md#pre-defined-annotation-keys).

## Platform Variants

When the variant of the CPU is not listed in the table, values are implementation-defined and SHOULD be submit

| ISA/ABI        | `architecture` | `variant` |
|----------------|----------------|-----------|
| ARM 32-bit, v6 | `arm`          | `v6`      |
| ARM 32-bit, v7 | `arm`          | `v7`      |
| ARM 32-bit, v8 | `arm`          | `v8`      |
| ARM 64-bit, v8 | `arm64`        | `v8`      |

## Example Image Index

*Example showing a simple image index pointing to image manifests for two platforms:*
```json,title=Image%20Index&mediatype=application/vnd.oci.image.index.v1%2Bjson
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.oci.image.index.v1+json",
  "manifests": [
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "size": 7143,
      "digest": "sha256:e692418e4cbaf90ca69d05a66403747baa33ee08806650b51fab815ad7fc331f",
      "platform": {
        "architecture": "ppc64le",
        "os": "linux"
      }
    },
    {
      "mediaType": "application/vnd.oci.image.manifest.v1+json",
      "size": 7682,
      "d
```
# Image Layer Filesystem Changeset

This document describes how to serialize a filesystem and filesystem changes like removed files into a blob ca
One or more layers are applied on top of each other to create a complete filesystem.
This document will use a concrete example to illustrate how to create and consume these filesystem layers.

This section defines the `application/vnd.oci.image.layer.v1.tar`, `application/vnd.oci.image.layer.v1.tar+gzi

## `+gzip` Media Types

* The media type `application/vnd.oci.image.layer.v1.tar+gzip` represents an `application/vnd.oci.image.layer.
* The media type `application/vnd.oci.image.layer.nondistributable.v1.tar+gzip` represents an `application/vnd

## `+zstd` Media Types

* The media type `application/vnd.oci.image.layer.v1.tar+zstd` represents an `application/vnd.oci.image.layer.
* The media type `application/vnd.oci.image.layer.nondistributable.v1.tar+zstd` represents an `application/vnd

## Distributable Format

* Layer Changesets for the [media type](media-types.md) `application/vnd.oci.image.layer.v1.tar` MUST be packa
* Layer Changesets for the [media type](media-types.md) `application/vnd.oci.image.layer.v1.tar` MUST NOT incl

## Change Types

Types of changes that can occur in a changeset are:

* Additions
* Modifications
* Removals

Additions and Modifications are represented the same in the changeset tar archive.

Removals are represented using "[whiteout](#whiteouts)" file entries (See [Representing Changes](#representing

### File Types

Throughout this document section, the use of word "files" or "entries" includes the following, where supported

* regular files
* directories
* sockets
* symbolic links
* block devices
* character devices
* FIFOs

### File Attributes

Where supported, MUST include file attributes for Additions and Modifications include:

* Modification Time (`mtime`)
* User ID (`uid`)
    * User Name (`uname`) *secondary to `uid`*
* Group ID (`gid `)
    * Group Name (`gname`) *secondary to `gid`*
* Mode (`mode`)
* Extended Attributes (`xattrs`)
* Symlink reference (`linkname` + symbolic link type)
* [Hardlink](#hardlinks) reference (`linkname`)

[Sparse files](https://en.wikipedia.org/wiki/Sparse_file) SHOULD NOT be used because they lack consistent supp

#### Hardlinks

* Hardlinks are a [POSIX concept](https://pubs.opengroup.org/onlinepubs/9699919799/functions/link.html) for ha
* Not all filesystems support hardlinks (e.g. [FAT](https://en.wikipedia.org/wiki/File_Allocation_Table)).
* Hardlinks are possible with all [file types](#file-types) except `directories`.
* Non-directory files are considered "hardlinked" when their link count is greater than 1.
* Hardlinked files are on a same device (i.e. comparing Major:Minor pair) and have the same inode.
* The corresponding files that share the link with the > 1 linkcount may be outside the directory that the cha
* Hardlinks are stored in a tar archive with type of a `1` char, per the [GNU Basic Tar Format][gnu-tar-standa
* While approaches to deriving new or changed hardlinks may vary, a possible approach is:

SET LinkMap to map[< Major:Minor String >]map[< inode integer >]< path string > SET LinkNames to map[< src path string >]< dest path string > FOR each path in root path IF path type is directory CONTINUE ENDIF SET filestat to stat(path) IF filestat num of links == 1 CONTINUE ENDIF IF LinkMap[filestat device][filestat inode] is not empty SET LinkNames[path] to LinkMap[filestat device][filestat inode] ELSE SET LinkMap[filestat device][filestat inode] to path ENDIF END FOR

With this approach, the link map and links names of a directory could be compared against that of another dire

#### Platform-specific attributes

Implementations on Windows MUST support these additional attributes, encoded in [PAX vendor
extensions](https://github.com/libarchive/libarchive/wiki/ManPageTar5#pax-interchange-format) as follows:

* [Windows file attributes](https://msdn.microsoft.com/en-us/library/windows/desktop/gg258117(v=vs.85).aspx) (
* [Security descriptor](https://msdn.microsoft.com/en-us/library/cc230366.aspx) (`MSWINDOWS.rawsd`): base64-en
* Mount points (`MSWINDOWS.mountpoint`): if present on a directory symbolic link, then the link should be crea
* Creation time (`LIBARCHIVE.creationtime`)

## Creating

### Initial Root Filesystem

The initial root filesystem is the base or parent layer.

For this example, an image root filesystem has an initial state as an empty directory.
The name of the directory is not relevant to the layer itself, only for the purpose of producing comparisons.

Here is an initial empty directory structure for a changeset, with a unique directory name `rootfs-c9d-v1`.


rootfs-c9d-v1/

### Populate Initial Filesystem

Files and directories are then created:


rootfs-c9d-v1/ etc/ my-app-config bin/ my-app-binary my-app-tools

The `rootfs-c9d-v1` directory is then created as a plain [tar archive][tar-archive] with relative path to `roo
Entries for the following files:


./ ./etc/ ./etc/my-app-config ./bin/ ./bin/my-app-binary ./bin/my-app-tools

### Populate a Comparison Filesystem

Create a new directory and initialize it with a copy or snapshot of the prior root filesystem.
Example commands that can preserve [file attributes](#file-attributes) to make this copy are:
* [cp(1)](https://linux.die.net/man/1/cp): `cp -a rootfs-c9d-v1/ rootfs-c9d-v1.s1/`
* [rsync(1)](https://linux.die.net/man/1/rsync):  `rsync -aHAX rootfs-c9d-v1/ rootfs-c9d-v1.s1/`
* [tar(1)](https://linux.die.net/man/1/tar): `mkdir rootfs-c9d-v1.s1 && tar --acls --xattrs -C rootfs-c9d-v1/ 

Any [changes](#change-types) to the snapshot MUST NOT change or affect the directory it was copied from.

For example `rootfs-c9d-v1.s1` is an identical snapshot of `rootfs-c9d-v1`.

In this way `rootfs-c9d-v1.s1` is prepared for updates and alterations.

**Implementor's Note**: *a copy-on-write or union filesystem can efficiently make directory snapshots*

Initial layout of the snapshot:

rootfs-c9d-v1.s1/ etc/ my-app-config bin/ my-app-binary my-app-tools

See [Change Types](#change-types) for more details on changes.

For example, add a directory at `/etc/my-app.d` containing a default config file, removing the existing config
Also a change (in attribute or file content) to `./bin/my-app-tools` binary to handle the config layout change

Following these changes, the representation of the `rootfs-c9d-v1.s1` directory:

rootfs-c9d-v1.s1/ etc/ my-app.d/ default.cfg bin/ my-app-binary my-app-tools

### Determining Changes

When two directories are compared, the relative root is the top-level directory.
The directories are compared, looking for files that have been [added, modified, or removed](#change-types).

For this example, `rootfs-c9d-v1/` and `rootfs-c9d-v1.s1/` are recursively compared, each as relative root pat

The following changeset is found:

Added: /etc/my-app.d/ Added: /etc/my-app.d/default.cfg Modified: /bin/my-app-tools Deleted: /etc/my-app-config

This reflects the removal of `/etc/my-app-config` and creation of a file and directory at `/etc/my-app.d/defau
`/bin/my-app-tools` has also been replaced with an updated version.

### Representing Changes

A [tar archive][tar-archive] is then created which contains *only* this changeset:

- Added and modified files and directories in their entirety
- Deleted files or directories marked with a [whiteout file](#whiteouts)

The resulting tar archive for `rootfs-c9d-v1.s1` has the following entries:

./etc/my-app.d/ ./etc/my-app.d/default.cfg ./bin/my-app-tools ./etc/.wh.my-app-config

To signify that the resource `./etc/my-app-config` MUST be removed when the changeset is applied, the basename

## Applying Changesets

* Layer Changesets of [media type](media-types.md) `application/vnd.oci.image.layer.v1.tar` are _applied_, rat
* Applying a layer changeset requires special consideration for the [whiteout](#whiteouts) files.
* In the absence of any [whiteout](#whiteouts) files in a layer changeset, the archive is extracted like a reg

### Changeset over existing files

This section specifies applying an entry from a layer changeset if the target path already exists.

If the entry and the existing path are both directories, then the existing path's attributes MUST be replaced
In all other cases, the implementation MUST do the semantic equivalent of the following:
- removing the file path (e.g. [`unlink(2)`](https://linux.die.net/man/2/unlink) on Linux systems)
- recreating the file path, based on the contents and attributes of the changeset entry

## Whiteouts

* A whiteout file is an empty file with a special filename that signifies a path should be deleted.
* A whiteout filename consists of the prefix `.wh.` plus the basename of the path to be deleted.
* As files prefixed with `.wh.` are special whiteout markers, it is not possible to create a filesystem which
* Once a whiteout is applied, the whiteout itself MUST also be hidden.
* Whiteout files MUST only apply to resources in lower/parent layers.
* Files that are present in the same layer as a whiteout file can only be hidden by whiteout files in subseque

The following is a base layer with several resources:


a/ a/b/ a/b/c/ a/b/c/bar

When the next layer is created, the original `a/b` directory is deleted and recreated with `a/b/c/foo`:


a/ a/.wh..wh..opq a/b/ a/b/c/ a/b/c/foo

When processing the second layer, `a/.wh..wh..opq` is applied first, before creating the new version of `a/b`,
For example, the following layer is equivalent to the layer above:


a/ a/b/ a/b/c/ a/b/c/foo a/.wh..wh..opq

Implementations SHOULD generate layers such that the whiteout files appear before sibling directory entries.

### Opaque Whiteout

* In addition to expressing that a single entry should be removed from a lower layer, layers may remove all of
* An opaque whiteout entry is a file with the name `.wh..wh..opq` indicating that all siblings are hidden in t

Let's take the following base layer as an example:


etc/ my-app-config bin/ my-app-binary my-app-tools tools/ my-app-tool-one

If all children of `bin/` are removed, the next layer would have the following:


bin/ .wh..wh..opq

This is called _opaque whiteout_ format.
An _opaque whiteout_ file hides _all_ children of the `bin/` including sub-directories and all descendants.
Using _explicit whiteout_ files, this would be equivalent to the following:


bin/ .wh.my-app-binary .wh.my-app-tools .wh.tools

In this case, a unique whiteout file is generated for each entry.
If there were more children of `bin/` in the base layer, there would be an entry for each.
Note that this opaque file will apply to _all_ children, including sub-directories, other resources and all de

Implementations SHOULD generate layers using _explicit whiteout_ files, but MUST accept both.

Any given image is likely to be composed of several of these Image Filesystem Changeset tar archives.

# Non-Distributable Layers

Due to legal requirements, certain layers may not be regularly distributable.
Such "non-distributable" layers are typically downloaded directly from a distributor but never uploaded.

Non-distributable layers SHOULD be tagged with an alternative mediatype of `application/vnd.oci.image.layer.nor
Implementations SHOULD NOT upload layers tagged with this media type; however, such a media type SHOULD NOT af

[Descriptor
# OCI Image Configuration

An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters
This specification outlines the JSON format describing images for use with a container runtime and execution t

This section defines the `application/vnd.oci.image.config.v1+json` [media type](media-types.md).

## Terminology

This specification uses the following terms:

### [Layer](layer.md)

* Image filesystems are composed of *layers*.
* Each layer represents a set of filesystem changes in a tar-based [layer format](layer.md), recording files t
* Layers do not have configuration metadata such as environment variables or default arguments - these are pro
* Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, th

### Image JSON

* Each image has an associated JSON structure which describes some basic information about the image such as d
* The JSON structure also references a cryptographic hash of each layer used by the image, and provides histor
* This JSON is considered to be immutable, because changing it would change the computed [ImageID](#imageid).
* Changing it means creating a new derived image, instead of changing the existing image.

### Layer DiffID

A layer DiffID is the digest over the layer's uncompressed tar archive and serialized in the descriptor digest
Layers SHOULD be packed and unpacked reproducibly to avoid changing the layer DiffID, for example by using [ta

NOTE: Do not confuse DiffIDs with [layer digests](manifest.md#image-manifest-property-descriptions), often ref

### Layer ChainID

For convenience, it is sometimes useful to refer to a stack of layers with a single identifier.
While a layer's `DiffID` identifies a single changeset, the `ChainID` identifies the subsequent application of
This ensures that we have handles referring to both the layer itself, as well as the result of the application
Use in combination with `rootfs.diff_ids` while applying layers to a root filesystem to uniquely and safely id

#### Definition

The `ChainID` of an applied set of layers is defined with the following recursion:

$$\text{ChainID}(L) = \text{DiffID}(L) \quad \text{ChainID}(L|...|L|L) = \text{Digest}(\text{ChainID}(L|...|L) + " " + \text{DiffID}(L))$$

For this, we define the binary `|` operation to be the result of applying the right operand to the left operand.
For example, given base layer `A` and a changeset `B`, we refer to the result of applying `B` to `A` as `A|B`.

Above, we define the `ChainID` for a single layer (`L`) as equivalent to the `DiffID` for that layer.
Otherwise, the `ChainID` for a set of applied layers (`L|...|L|L`) is defined as the recursion `Digest(Chain`

#### Explanation

Let's say we have layers A, B, C, ordered from bottom to top, where A is the base and C is the top.
Defining `|` as a binary application operator, the root filesystem may be `A|B|C`.
While it is implied that `C` is only useful when applied to `A|B`, the identifier `C` is insufficient to ident

The main issue is when we have two definitions of `C`, `C = C` and `C = A|B|C`.
If this is true (with some handwaving), `C = x|C` where `x = any application`.
This means that if an attacker can define `x`, relying on `C` provides no guarantee that the layers were appli

The `ChainID` addresses this problem by being defined as a compound hash.
__We differentiate the changeset `C`, from the order-dependent application `A|B|C` by saying that the resulting

Let's expand the definition of `ChainID(A|B|C)` to explore its internal structure:

$$\text{ChainID}(A) = \text{DiffID}(A) \quad \text{ChainID}(A|B) = \text{Digest}(\text{ChainID}(A) + " " + \text{DiffID}(B)) \quad \text{ChainID}(A|B|C) = \text{Digest}(\text{ChainID}(A|B) + " " + \text{DiffID}(C))$$

We can replace each definition and reduce to a single equality:

$$\text{ChainID}(A|B|C) = \text{Digest}(\text{Digest}(\text{DiffID}(A) + " " + \text{DiffID}(B)) + " " + \text{DiffID}(C))$$

Hopefully, the above is illustrative of the _actual_ contents of the `ChainID`.
Most importantly, we can easily see that `ChainID(C) != ChainID(A|B|C)`, otherwise, `ChainID(C) = DiffID(C)`,

### ImageID

Each image's ID is given by the SHA256 hash of its [configuration JSON](#image-json).
It is represented as a hexadecimal encoding of 256 bits, e.g., `sha256:a9561eb1b190625c9adb5a9513e72c4dedafc1c`
Since the [configuration JSON](#image-json) that gets hashed references hashes of each layer in the image, thi

## Properties

Note: Any OPTIONAL field MAY also be set to null, which is equivalent to being absent.

- **created** *string*, OPTIONAL

  An combined date and time at which the image was created, formatted as defined by [RFC 3339, section 5.6][rf

- **author** *string*, OPTIONAL

  Gives the name and/or email address of the person or entity which created and is responsible for maintaining

- **architecture** *string*, REQUIRED

The CPU architecture which the binaries in this image are built to run on.
Configurations SHOULD use, and implementations SHOULD understand, values listed in the Go Language document

- **os** *string*, REQUIRED

  The name of the operating system which the image is built to run on.
  Configurations SHOULD use, and implementations SHOULD understand, values listed in the Go Language document

- **os.version** *string*, OPTIONAL

  This OPTIONAL property specifies the version of the operating system targeted by the referenced blob.
  Implementations MAY refuse to use manifests where `os.version` is not known to work with the host OS version
  Valid values are implementation-defined. e.g. `10.0.14393.1066` on `windows`.

- **os.features** *array of strings*, OPTIONAL

  This OPTIONAL property specifies an array of strings, each specifying a mandatory OS feature.
  When `os` is `windows`, image indexes SHOULD use, and implementations SHOULD understand the following values

  - `win32k`: image requires `win32k.sys` on the host (Note: `win32k.sys` is missing on Nano Server)

- **variant** *string*, OPTIONAL

  The variant of the specified CPU architecture.
  Configurations SHOULD use, and implementations SHOULD understand, `variant` values listed in the [Platform V

- **config** *object*, OPTIONAL

  The execution parameters which SHOULD be used as a base when running a container using the image.
  This field can be `null`, in which case any execution parameters should be specified at creation of the cont

  - **User** *string*, OPTIONAL

    The username or UID which is a platform-specific structure that allows specific control over which user t
    This acts as a default value to use when the value is not specified when creating a container.
    For Linux based systems, all of the following are valid: `user`, `uid`, `user:group`, `uid:gid`, `uid:gro
    If `group`/`gid` is not specified, the default group and supplementary groups of the given `user`/`uid` i

  - **ExposedPorts** *object*, OPTIONAL

    A set of ports to expose from a container running this image.
    Its keys can be in the format of:
`port/tcp`, `port/udp`, `port` with the default protocol being `tcp` if not specified.
    These values act as defaults and are merged with any specified when creating a container.
    **NOTE:** This JSON structure value is unusual because it is a direct JSON serialization of the Go type `

  - **Env** *array of strings*, OPTIONAL

    Entries are in the format of `VARNAME=VARVALUE`.
    These values act as defaults and are merged with any specified when creating a container.

  - **Entrypoint** *array of strings*, OPTIONAL

    A list of arguments to use as the command to execute when the container starts.
    These values act as defaults and may be replaced by an entrypoint specified when creating a container.

  - **Cmd** *array of strings*, OPTIONAL

Default arguments to the entrypoint of the container.
  These values act as defaults and may be replaced by any specified when creating a container.
  If an `Entrypoint` value is not specified, then the first entry of the `Cmd` array SHOULD be interpreted

- **Volumes** *object*, OPTIONAL

  A set of directories describing where the process is likely to write data specific to a container instance
  **NOTE:** This JSON structure value is unusual because it is a direct JSON serialization of the Go type `

- **WorkingDir** *string*, OPTIONAL

  Sets the current working directory of the entrypoint process in the container.
  This value acts as a default and may be replaced by a working directory specified when creating a contain

- **Labels** *object*, OPTIONAL

  The field contains arbitrary metadata for the container.
  This property MUST use the [annotation rules](annotations.md#rules).

- **StopSignal** *string*, OPTIONAL

  The field contains the system call signal that will be sent to the container to exit. The signal can be a

- **Memory** *integer*, OPTIONAL

  This property is *reserved* for use, to [maintain compatibility](media-types.md#compatibility-matrix).

- **MemorySwap** *integer*, OPTIONAL

  This property is *reserved* for use, to [maintain compatibility](media-types.md#compatibility-matrix).

- **CpuShares** *integer*, OPTIONAL

  This property is *reserved* for use, to [maintain compatibility](media-types.md#compatibility-matrix).

- **Healthcheck** *object*, OPTIONAL

  This property is *reserved* for use, to [maintain compatibility](media-types.md#compatibility-matrix).

- **rootfs** *object*, REQUIRED

  The rootfs key references the layer content addresses used by the image.
  This makes the image config hash depend on the filesystem hash.

  - **type** *string*, REQUIRED

    MUST be set to `layers`.
    Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking a

  - **diff_ids** *array of strings*, REQUIRED

    An array of layer content hashes (`DiffIDs`), in order from first to last.

- **history** *array of objects*, OPTIONAL

  Describes the history of each layer.
  The array is ordered from first to last.

The object has the following fields:

 - **created** *string*, OPTIONAL

  A combined date and time at which the layer was created, formatted as defined by [RFC 3339, section 5.6]

 - **author** *string*, OPTIONAL

  The author of the build point.

 - **created_by** *string*, OPTIONAL

  The command which created the layer.

 - **comment** *string*, OPTIONAL

  A custom message set when creating the layer.

 - **empty_layer** *boolean*, OPTIONAL

  This field is used to mark if the history item created a filesystem diff.
  It is set to true if this history item doesn't correspond to an actual layer in the rootfs section (for

Any extra fields in the Image JSON struct are considered implementation specific and MUST be ignored by any imp

Whitespace is OPTIONAL and implementations MAY have compact JSON with no whitespace.

## Example

Here is an example image configuration JSON document:

```json,title=Image%20JSON&mediatype=application/vnd.oci.image.config.v1%2Bjson
{
    "created": "2015-10-31T22:22:56.015925234Z",
    "author": "Alyssa P. Hacker <alyspdev@example.com>",
    "architecture": "amd64",
    "os": "linux",
    "config": {
        "User": "alice",
        "ExposedPorts": {
            "8080/t
```
# Annotations
Several components of the specification, like [Image Manifests](manifest.md) and [Descriptors](descriptor.md),

This property contains arbitrary metadata.

## Rules

* Annotations MUST be a key-value map where both the key and value MUST be strings.
* While the value MUST be present, it MAY be an empty string.
* Keys MUST be unique within this map, and best practice is to namespace the keys.
* Keys SHOULD be named using a reverse domain notation - e.g. `com.example.myKey`.
* The prefix `org.opencontainers` is reserved for keys defined in Open Container Initiative (OCI) specification
* Keys using the `org.opencontainers.image` namespace are reserved for use in the OCI Image Specification and
* If there are no annotations then this property MUST either be absent or be an empty map.
* Consumers MUST NOT generate an error if they encounter an unknown annotation key.

## Pre-Defined Annotation Keys

This specification defines the following annotation keys, intended for but not limited to [image index](image-

* **org.opencontainers.artifact.created** date and time on which the artifact was built, conforming to [RFC 33
* **org.opencontainers.artifact.description**: human readable description for the artifact (string)
* **org.opencontainers.image.created** date and time on which the image was built, conforming to [RFC 3339][rf
* **org.opencontainers.image.authors** contact details of the people or organization responsible for the image
* **org.opencontainers.image.url** URL to find more information on the image (string)
* **org.opencontainers.image.documentation** URL to get documentation on the image (string)
* **org.opencontainers.image.source** URL to get source code for building the image (string)
* **org.opencontainers.image.version** version of the packaged software
  * The version MAY match a label or tag in the source code repository
  * version MAY be [Semantic versioning-compatible](https://semver.org/)
* **org.opencontainers.image.revision** Source control revision identifier for the packaged software.
* **org.opencontainers.image.vendor** Name of the distributing entity, organization or individual.
* **org.opencontainers.image.licenses** License(s) under which contained software is distributed as an [SPDX L
* **org.opencontainers.image.ref.name** Name of the reference for a target (string).
  * SHOULD only be considered valid when on descriptors on `index.json` within [image layout](image-layout.md)
  * Character set of the value SHOULD conform to alphanum of `A-Za-z0-9` and separator set of `-._:@/+`
  * The reference must match the following [grammar](considerations.md#ebnf):
    ```
    ref       ::= component ("/" component)*
    component ::= alphanum (separator alphanum)*
    alphanum  ::= [A-Za-z0-9]+
    separator ::= [-._:@+] | "--"
    ```
* **org.opencontainers.image.title** Human-readable title of the image (string)
* **org.opencontainers.image.description** Human-readable description of the software packaged in the image (s
* **org.opencontainers.image.base.digest** [Digest](descriptor.md#digests) of the image this image is based on
  * This SHOULD be the immediate image sharing zero-indexed layers with the image, such as from a Dockerfile `
  * This SHOULD NOT reference any other images used to generate the contents of the image (e.g., multi-stage D
* **org.opencontainers.image.base.name** Image reference of the image this image is based on (string)
  * This SHOULD be image references in the format defined by [distribution/distribution](https://github.com/di
  * This SHOULD be a fully qualified reference name, without any assumed default registry. (e.g., `registry.ex
  * This SHOULD be the immediate image sharing zero-indexed layers with the image, such as from a Dockerfile `
  * This SHOULD NOT reference any other images used to generate the contents of the image (e.g., multi-stage D
  * If th
# Conversion to OCI Runtime Configuration

When extracting an OCI Image into an [OCI Runtime bundle][oci-runtime-bundle], two orthogonal components of th

1. Extraction of the root filesystem from the set of [filesystem layers](layer.md).
2. Conversion of the [image configuration blob](config.md) to an [OCI Runtime configuration blob][oci-runtime-

This section defines how to convert an `application/vnd.oci.image.config.v1+json` blob to an [OCI runtime conf
The former component of extraction is defined [elsewhere](layer.md) and is orthogonal to configuration of a ru
The values of runtime configuration properties not specified by this document are implementation-defined.

A converter MUST rely on the OCI image configuration to build the OCI runtime configuration as described by th

The "default generated runtime configuration" MAY be overridden or combined with externally provided inputs fr
In addition, a converter MAY have its own implementation-defined defaults and extensions which MAY be combined
The restrictions in this document refer only to combining implementation-defined defaults with the "default ge
Externally provided inputs are considered to be a modification of the `application/vnd.oci.image.config.v1+jso

For example, externally provided inputs MAY cause an environment variable to be added, removed or changed.
However an implementation-defined default SHOULD NOT result in an environment variable being removed or change

```
[oci-runtime-bundle]: https://github.com/opencontainers/runtime-spec/blob/v1.0.0/bundle.md
[oci-runtime-config]: https://github.com/opencontainers/runtime-spec/blob/v1.0.0/config.md
```

## Verbatim Fields

Certain image configuration fields have an identical counterpart in the runtime configuration.
Some of these are purely annotation-based fields, and have been extracted into a [separate subsection](#annota
A compliant configuration converter MUST extract the following fields verbatim to the corresponding field in t

| Image Field          | Runtime Field   | Notes |
| -------------------- | --------------- | ----- |
| `Config.WorkingDir`  | `process.cwd`   |       |
| `Config.Env`         | `process.env`   | 1     |
| `Config.Entrypoint`  | `process.args`  | 2     |
| `Config.Cmd`         | `process.args`  | 2     |

1. The converter MAY add additional entries to `process.env` but it SHOULD NOT add entries that have variable
2. If both `Config.Entrypoint` and `Config.Cmd` are specified, the converter MUST append the value of `Config.C

### Annotation Fields

These fields all affect the `annotations` of the runtime configuration, and are thus subject to [precedence](#

| Image Field          | Runtime Field   | Notes |
| -------------------- | --------------- | ----- |
| `os`                 | `annotations`   | 1,2   |
| `architecture`       | `annotations`   | 1,3   |
| `variant`            | `annotations`   | 1,4   |
| `os.version`         | `annotations`   | 1,5   |
| `os.features`        | `annotations`   | 1,6   |
| `author`             | `annotations`   | 1,7   |
| `created`            | `annotations`   | 1,8   |
| `Config.Labels`      | `annotations`   |       |
| `Config.StopSignal`  | `annotations`   | 1,9   |

1. If a user has explicitly specified this annotation with `Config.Labels`, then the value specified in this f
2. The value of this field MUST be set as the value of `org.opencontainers.image.os` in `annotations`.
3. The value of this field MUST be set as the value of `org.opencontainers.image.architecture` in `annotations
4. The value of this field MUST be set as the value of `org.opencontainers.image.variant` in `annotations`.
5. The value of this field MUST be set as the value of `org.opencontainers.image.os.version` in `annotations`.
6. The value of this field MUST be set as the value of `org.opencontainers.image.os.features` in `annotations`
7. The value of this field MUST be set as the value of `org.opencontainers.image.author` in `annotations`.
8. The value of this field MUST be set as the value of `org.opencontainers.image.created` in `annotations`.
9. The value of this field MUST be set as the value of `org.opencontainers.image.stopSignal` in `annotations`.

## Parsed Fields

Certain image configuration fields have a counterpart that must first be translated.
A compliant configuration converter SHOULD parse all of these fields and set the corresponding fields in the g

| Image Field          | Runtime Field   |
| -------------------- | --------------- |
| `Config.User`        | `process.user.*` |

The method of parsing the above image fields are described in the following sections.

### `Config.User`

If the values of [`user` or `group`](config.md#properties) in `Config.User` are numeric (`uid` or `gid`) then t
If the values of [`user` or `group`](config.md#properties) in `Config.User` are not numeric (`user` or `group`
For Unix-like systems, this MAY involve resolution through NSS or parsing `/etc/passwd` from the extracted con

In addition, a converter SHOULD set the value of `process.user.additionalGids` to a value corresponding to the
For Unix-like systems, this MAY involve resolution through NSS or parsing `/etc/group` and determining the gro
If the value of [`user`](config.md#properties) in `Config.User` is numeric, the converter SHOULD NOT modify `p

If `Config.User` is not defined, the converted `process.user` value is implementation-defined.
If `Config.User` does not correspond to a user in the container's context, the converter MUST return an error.

## Optional Fields

Certain image configuration fields are not applicable to all conversion use cases, and thus are optional for c
A compliant configuration converter SHOULD provide a way for users to extract these fields into the generated

| Image Field          | Runtime Field     | Notes |
| -------------------- | ----------------- | ----- |
| `Config.ExposedPorts` | `annotations`     | 1     |
| `Config.Volumes`      | `mounts`          | 2     |

1. The runtime configuration does not have a corresponding field for this image field.
   However, converters SHOULD set the [`org.opencontainers.image.exposedPorts` annotation](#config.exposedport
2. Implementations SHOULD provide mounts for these locations such that application data is not written to the
   If a converter implements conversion for this field using mountpoints, it SHOULD set the `destination` of t
   An implementation MAY seed the contents of the mount with data in the image at the same location.
   If a _new_ image is created from a container based on the image described by this configuration, data in th
   The other `mounts` fields are platform and context dependent, and thus are implementation-defined.
   Note that the implementation of `Config.Volumes` need not use mountpoints, as it is effectively a mask of t

### `Config.ExposedPorts`

The OCI runtime configuration does not provide a way of expressing the concept of "container exposed ports".
However, converters SHOULD set the **org.opencontainers.image.exposedPorts** annotation, unless doing so will

**org.opencontainers.image.exposedPorts** is the list of values that correspond to the [keys defined for `Conf

## Annotations

There are three ways of annotating an OCI image in this specification:

1. `Config.Labels` in the [configuration](config.md) of the image.
2. `annotations` in the [manifest](man
# Extensibility

Implementations that are reading/processing [manifests](manifest.md) or [image indexes](image-index.md) MUST N
Instead they MUST ignore unknown properties.

# Canonicalization

* OCI Images are [content-addressable](https://en.wikipedia.org/wiki/Content-addressable_storage). See [descri
* One benefit of content-addressable storage is easy deduplication.
* Many images might depend on a particular [layer](layer.md), but there will only be one blob in the [store](i
* With a different serialization, that same semantic layer would have a different hash, and if both versions o
* To allow efficient storage, implementations serializing content for blobs SHOULD use a canonical serializati
* This increases the chance that different implementations can push the same semantic content to the store wit

## JSON

[JSON][] content SHOULD be serialized as [canonical JSON][canonical-json].
Of the [OCI Image Format Specification media types](media-types.md), all the types ending in `+json` contain J
Implementations:

* [Go][]: [github.com/docker/go][], which claims to implement [canonical JSON][canonical-json] except for Unic

[canonical-json]: https://wiki.laptop.org/go/Canonical_JSON
[github.com/docker/go]: https://github.com/docker/go/
[Go]: https://golang.org/
[JSON]: https://json.org/

# EBNF

For field formats described in this specification, we use a limited subset of [Extended Backus-Naur Form][ebnf
Grammars present in the OCI specification are regular and can be converted to a single regular expressions.
However, regular expressions are avoided to limit abiguity between regular expression syntax.
By defining a subset of EBNF used here, the possibility of variation, misunderstanding or ambiguities from lin

Grammars are made up of rules in the following form:


symbol ::= expression


We can say we have the production identified by symbol if the input is matched by the expression.
Whitespace is completely ignored in rule definitions.

## Expressions

The simplest expression is the literal, surrounded by quotes:


literal ::= "matchthis"


The above expression defines a symbol, "literal", that matches the exact input of "matchthis".
Character classes are delineated by brackets (`[]`), describing either a set, range or multiple range of chara


set := [abc] range := [A-Z]


The above symbol "set" would match one character of either "a", "b" or "c".
The symbol "range" would match any character, "A" to "Z", inclusive.
Currently, only matching for 7-bit ascii literals and character classes is defined, as that is all that is req
Multiple character ranges and explicit characters can be specified in a single character classes, as follows:


multipleranges := [a-zA-Z=-]


The above matches the characters in the range `A` to `Z`, `a` to `z` and the individual characters `-` and `=`

Expressions can be made up of one or more expressions, such that one must be followed by the other.
This is known as an implicit concatenation operator.
For example, to satisfy the following rule, both `A` and `B` must be matched to satisfy the rule:


symbol ::= A B

Each expression must be matched once and only once, `A` followed by `B`.
To support the description of repetition and optional match criteria, the postfix operators `*` and `+` are de
`*` indicates that the preceeding expression can be matched zero or more times.
`+` indicates that the preceeding expression must be matched one or more times.
These appear in the following form:

zeroormore ::= expression* oneormore ::= expression+

Parentheses are used to group expressions into a larger expression:

group ::= (A B)

Like simpler expressions above, operators can be applied to groups, as well.
To allow for alternates, we also define the infix operator `|`.

oneof ::= A | B

The above indicates that the expression should match one of the expressions, `A` or `B`.

## Precedence

The operator precedence is in the following order:

- Terminals (literals and character classes)
- Grouping `()`
- Unary operators `+*`
- Concatenation
- Alternates `|`

The precedence can be better described using grouping to show equivalents.
Concatenation has higher precedence than alernates, such `A B | C D` is equivalent to `(A B) | (C D)`.
Unary operators have higher precedence than alternates and concatenation, such that `A+ | B+` is equivalent to

## Examples

The following combines the previous definitions to match a simple, relative path name, describing the individu

path ::= component ("/" component)* component ::= [a-z]+

The production "component" is one or more lowercase letters.
A "path" is then at least one component, possibly followed by zero or more slash-component pairs.
The above can be converted into the following regular expression:

[a-z]+(?:/[a-z]+)*

[ebnf]: https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form
[xmlebnf]: https://www.w3.org/TR/REC-xml/#sec-notatio
# OCI Image Implementations

Projects or Companies currently adopting the OCI Image Specification

* [projectatomic/skopeo](https://github.com/projectatomic/skopeo)
* [Amazon Elastic Container Registry (ECR)](https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-manif
* [Azure Container Registry (ACR)](https://docs.microsoft.com/azure/container-registry/container-registry-imag
* [openSUSE/umoci](https://github.com/openSUSE/umoci)
* [cloudfoundry/grootfs](https://github.com/cloudfoundry/grootfs) ([source](https://github.com/cloudfoundry/gr
* [Mesos plans](https://issues.apache.org/jira/browse/MESOS-5011) ([design doc](https://docs.google.com/docume
* [Docker](https://github.com/docker)
  - [docker/docker (`docker save/load` WIP)](https://github.com/docker/docker/pull/26369)
  - [distribution/distribution (registry PR)](https://github.com/distribution/distribution/pull/2076)
* [containerd/containerd](https://github.com/containerd/containerd)
* [Containers](https://github.com/containers/)
  - [containers/build](https://github.com/containers/build)
  - [containers/image](https://github.com/containers/image)
  - [containers/oci-spec-rs](https://github.com/containers/oci-spec-rs)
  - [containers/libocispec](https://github.com/containers/libo