
manifest-tool

`manifest-tool` is a command line utility used to view or push multi-platform container image references located in an OCIv1 or Docker v2.2 compatible container registry.

While several other tools include more complete capabilities to view and manipulate the *manifest* objects associated with container images and artifacts, `manifest-tool` was created as one of the first command line tools capable of assembling “manifest lists” (Docker v2.2), now more commonly known as “indexes” in the OCIv1 image specification. **Manifest lists** or **indexes** exist for the purpose of combining an array of architecture and platform specific container image manifests under a single reference. This allows a container runtime to select the appropriate index entry that matches the local node’s architecture and platform. Before these kinds of manifests were available it required separate instructions, configurations, or code changes to set up the appropriate platform-specific image reference depending on the platform in use.

Installation

The releases of `manifest-tool` are built using the latest Go version, and binaries for many architectures are available as pre-built binaries with each release, found on the GitHub releases page.

You can also use `manifest-tool` via an existing Docker image automatically generated for a large number of architectures with each release. To use this image simply run

```
1 $ docker run mplatform/manifest-tool
```

To build `manifest-tool` locally, clone this repository and build the binary as shown below. Note that you will need to have a recent version of the Go SDK installed on your system as well as `make`.

```
1 $ git clone https://github.com/estesp/manifest-tool
2 $ cd manifest-tool && make binary
```

If you don’t want to install a local development environment but have Docker installed, you can use `make build` to build `manifest-tool` inside the official Go SDK container.

Additional targets `make static` target will build a statically-linked binary, and `make cross` will build a binary for all supported platforms using Go’s cross-compilation capabilities.

Querying Manifests Without Installation

If you only have a requirement to query public image references to validate platform support you can use a related project, `mquery`, which allows remote querying of public registry images.

Use `mquery` by running its DockerHub-located image, **`mplatform/mquery:latest`**, and specifying a target image to query, as shown in the example below:

```
1 $ docker run --rm mplatform/mquery mplatform/mquery:latest
2 Image: mplatform/mquery:latest (digest: sha256:
   d0989420b6f0d2b929fd9355f15c767f62d0e9a72cdf999d1eb16e6073782c71)
3 * Manifest List: Yes (Image type: application/vnd.docker.distribution.
   manifest.list.v2+json)
4 * Supported platforms:
5   - linux/ppc64le
6   - linux/amd64
7   - linux/386
8   - linux/s390x
9   - linux/riscv64
10  - linux/arm64/v8
11  - linux/arm/v7
12  - linux/arm/v6
13  - windows/amd64:10.0.17763.2300
14  - windows/amd64:10.0.14393.4770
```

The `mquery` program itself is a small Go program running as an AWS Lambda function using a small cache so recent image results are cached. More information is available in the `mquery` GitHub repo.

Outdated, but original, details on the creation of `mquery` are found in my blog post from the Moby Summit EU 2017 on this topic.

Sample Usage

`manifest-tool` can: - **inspect** manifests (of all media types) within any registry supporting the OCI distribution API - **push** manifest list/index objects to any registry which supports the OCI distribution API and the appropriate image (Docker or OCI) image specification.

Note: For pushing you will have to provide your registry credentials via either a) the command line, b) use a credential helper application (`manifest-tool` supports these in the same way Docker client does), or c) already be logged in to a registry and have an existing Docker client configuration file with credentials.

Inspect Inspect/view the manifest of any image reference (*repo/image:tag* combination) with the **inspect** command. You must provide a tag, even if the tag is `latest` as the containerd resolver does not auto-append latest to image references and `manifest-tool` utilizes the containerd resolver library.

Example output of an `inspect` on a manifest list media type is shown below:

```
1 $ manifest-tool inspect golang:1.17
2 Name:   golang:1.17 (Type: application/vnd.docker.distribution.manifest
   .list.v2+json)
3 Digest: sha256:1
   a35cc2c5338409227c7293add327ebe42e1ee5465049f6c57c829588e3f8a39
4 * Contains 10 manifest references:
5 [1]     Type: application/vnd.docker.distribution.manifest.v2+json
6 [1]     Digest: sha256:
   a6c0b3e8b7d2faed2415448f20e75ed26eed6fdb1d261873ed4205907d92c674
7 [1]     Length: 1796
8 [1] Platform:
9 [1]     -      OS: linux
10 [1]    -      Arch: amd64
11 [1] # Layers: 7
12     layer 01: digest = sha256:0
   c6b8ff8c37e92eb1ca65ed8917e818927d5bf318b6f18896049b5d9afc28343
13     layer 02: digest = sha256:412
   caad352a3ecbb29c080379407ae0761e7b9b454f7239cbfd1d1da25e06b29
14     layer 03: digest = sha256:
   e6d3e61f7a504fa66d7275123969e9917570188650eb84b2280a726b996040f6
15     layer 04: digest = sha256:461
   bb1d8c517c7f9fc0f1df66c9dc34c85a23421c1e1c540b2e28cbb258e75f5
16     layer 05: digest = sha256:9297634
   c9537024497f76a2e1b374d8a315baa21d45bf36dc7980dc42ab93b0b
17     layer 06: digest = sha256:
   c9cefb9872505d3a6fdcbdbde4103393da3e384443c5a8cdd62bc368927ea1cc
18     layer 07: digest = sha256:8560
   fc463426dc7e494720250efec25cdae1c4bf796c1a0172f791c0c7dde1c6
19
20 ... skipping 8 manifest entries
21
22 [10]    Type: application/vnd.docker.distribution.manifest.v2+json
23 [10]    Digest: sha256:78
   af34429b7d75d61890746d39e27beb447970bad6803ed11ab4be920dbbd061
24 [10]    Length: 3401
25 [10] Platform:
26 [10]    -      OS: windows
27 [10]    - OS Vers: 10.0.17763.2565
28 [10]    -      Arch: amd64
29 [10] # Layers: 13
30     layer 01: digest = sha256:4612
   f6d0b889cad0ed0292fae3a0b0c8a9e49aff6dea8eb049b2386d9b07986f
31     layer 02: digest = sha256:1
   bd78008c728d8f9e56dc2093e6eb55f0f0b1aa96e5d0c7ccc830c5f60876cdf
32     layer 03: digest = sha256:
   f0c1566a9285d9465334dc923e9d6fd93a51b3ef6cb8497efcacbcf64e3b93fc
33     layer 04: digest = sha256:1
```

```

34         b56caecef9c44ed58d2621ffb6f87f797b532c81f1271d9c339222462523eb2
layer 05: digest = sha256:5
        a3ed0a076d58c949f5debdbc3616b6ccd008426c62635ab387836344123e2a6
35 layer 06: digest = sha256:
        f25f9584c1aa90dae36704d6bef0e59e72002fcb13e8a4618f64c9b13479c0df

36 layer 07: digest = sha256:12
        d4fbc7cf0f85fc63860f052f76bfb4429eca8b878abce79a25bfdc30f9e9f5
37 layer 08: digest = sha256:
        c325dc9f1660ea537aae55b89be63d336762d5a3a02e929d52940586fb0f677e

38 layer 09: digest = sha256:
        dd4f3aabaa2a9bf80e2a7f417dba559f6b34e640c21b138dce099328406c8903

39 layer 10: digest = sha256:57
        e61367d26baed9e16a8d5310c520ae3429d5cc7956569f325cd9de01f33604
40 layer 11: digest = sha256:98
        eb9abc560e8d857685b3b0131c733bdbb5f3c79e93fe7e9163e443736c2f51
41 layer 12: digest = sha256:
        fffb0b96d90540c5fe04bec7c3803e767fc06c03da00c569b92ec1abeb2db503

42 layer 13: digest = sha256:
        e6c16363a908ee64151cd232d466b723e3edac978f1c7693db3dcbed09694d76

```

While we can query non-manifest lists/indexes as well, this entry is clearly a manifest list (see the media type) with many platforms supported. To read how container engines like Docker use this information to determine what image/layers to pull read this early blog post on multi-platform support in Docker.

Create/Push You can create manifest list or index entries in a registry by using the **push** command with either a YAML file describing the images to assemble or by using a series of command line parameters.

A sample YAML file is shown below. As long as the target registry supports the cross-repository push feature the source and target image names can differ as long as they are within the same registry host. For example, a source image could be named `myprivreg:5000/someimage_arm64:latest` and referenced by a manifest list in repository `myprivreg:5000/someimage:latest`.

Given a private registry running on port 5000, here is a sample YAML file input to `manifest-tool` to create a manifest list combining an 64-bit ARMv8 image and an amd64 image:

```

1 image: myprivreg:5000/someimage:latest
2 tags: ["1.0.0", "1.0", "1"]
3 manifests:
4   -
5     image: myprivreg:5000/someimage:arm64

```

```
6     platform:
7       architecture: arm64
8       os: linux
9   -
10    image: myprivreg:5000/someimage:amd64
11    platform:
12      architecture: amd64
13      os: linux
```

Note: Of course these component images must have been built and pushed to your target registry before running `manifest-tool`. The job of `manifest-tool` is simply to create the manifest which assembles existing images under a combined image reference pointing to a manifest list or OCI index.

Given this example YAML input you can push this manifest list as follows:

```
1 $ manifest-tool push from-spec someimage.yaml
```

`manifest-tool` can also use command line arguments with a templating model to specify the architecture/platform list and the from and to image formats as shown below:

```
1 $ manifest-tool push from-args \
2   --platforms linux/amd64,linux/s390x,linux/arm64 \
3   --template foo/bar-ARCH:v1 \
4   --tags v1.0.0,v1.0 \
5   --target foo/bar:v1
```

Specifically: - `--platforms` specifies which platforms you want to push for in the form OS-ARCH,OS/ARCH,... - `--template` specifies the image repo:tag source for inputs by replacing the placeholders `OS`, `ARCH` and `VARIANT` with the inputs from `--platforms`. - `--tags` specifies the tags to apply to the target image in addition to the `--target` tag. - `--target` specifies the target image repo:tag that will be the manifest list entry in the registry.

When using the optional `VARIANT` placeholder, it is ignored when a `platform` does not have a variant.

```
1 $ manifest-tool push from-args \
2   --platforms linux/amd64,linux/arm/v5,linux/arm/v7 \
3   --template foo/bar-ARCHVARIANT:v1 \
4   --target foo/bar:v1
```

For the above example, `linux/amd64` when applied to the template will look for an image named `foo/bar-amd64:v1`, while the platform entry `linux/arm/v5` will resolve to an image reference: `foo/bar-armv5:v1`.

Known Supporting Registries

All major public cloud registries have added Docker v2.2 manifest list support over the years since the “fat manifest”-enabled specification came out in 2016.

Most registries also support the formalization of that via the “index” manifest type in the OCIv1 image format specification published in 2017.

If you find a registry provider for which `manifest-tool` does not work properly please open an issue in the GitHub issues for this project.

Test Index/Manifest List Support

If you operate or use a registry claiming conformance to Docker v2.2 spec and API or the OCIv1 image spec and distribution spec and want to confirm manifest list/index support please use the pre-configured test script available in this repository.

See the `test-registry.sh` script in this repo’s **integration** directory for further details. A simple example is shown here:

```
1 $ ./test-registry.sh r.myprivreg.com/somerepo
```

History

This `manifest-tool` codebase was initially a joint project with Harshal Patil from IBM Bangalore, and originally forked from the registry client codebase, `skopeo`, created by Antonio Murdaca/`runc0m`, that later became a part of Project Atomic. `Skopeo` then became part of the overall Red Hat container client tooling later in its lifetime where it still resides today in the GitHub containers organization. The **v2** rewrite of `manifest-tool` removed all the original vestiges of `skopeo`’s original registry client and manifest parsing code, but is still part of the **v1** releases of `manifest-tool` and codebase.

Thanks to both Antonio and Harshal for their initial work that made this possible! Also, thanks to Christy Perez from IBM Systems for her hard work in bringing the functionality of `manifest-tool` to the Docker client via a `docker/cli` PR. In early 2018 this PR formed the basis of a new `docker manifest` command which comprised most of the original code of `manifest-tool` and made multi-platform image creation available to users of the Docker client.

License

`manifest-tool` is licensed under the Apache Software License (ASL) 2.0