Image Registries

Glyn Normington

May 20, 2019

This document provides a formal model of image registries.

Contents

1 Introduction

This document provides a formal model of image registries.

2 Overview of this document

Docker Inc. introduced container images and registries to hold them and these were later standardised as part of the Open Container Initiative. The reader is assumed to have a basic understanding of how images are used.

This document models image references, repositories, and registries. It covers digests and tags.

The Z specification language is used to capture the model, but sufficient English text is also provided that readers who do not know Z should be able to understand the model. The appendix contains a summary of the Z notation. For more information about Z, please consult the Z Manual (`https://www.cs.umd.edu/ mvz/handouts/z-manual.pdf`). The model was type checked using `fuzz` (`https://bitbucket.org/Spivey/fuzz`).

I am grateful to Chris Frost for his helpful comments on this document.

3 Fundamentals

Images are opaque blobs as far as we are concerned here - the decomposition into layers is ignored. Similarly, cryptographic hashes (or *hexes* to use the terminology of the OCI Distribution specification), tags, and (registry) hostnames and paths are modelled, but their details are not.

$$[Image, Hex, Tag, Hostname, Path]$$

There is a special reserved tag.

$$Latest : Tag$$

4 Content Digests

A content digest is a combination of a cryptographic hash function (such as `SHA-256`) or "algorithm", and the hash output by such a function.

ContentDigest

$$alg : Image \to Hex$$
$$hash : Hex$$

The idea is that a content digest d identifies an image i if and only if applying the hash function (from the content digest) to the image produces the hash output in the content digest:

$$d.alg\ i = d.hash$$

An optional content digest is modelled as a datatype.

$$OptionalContentDigest ::= None \mid Dig\langle\langle ContentDigest \rangle\rangle$$

5 Repositories

A repository is a collection of images indexed by content digest and by tag.

Repo

$$cd : \mathit{ContentDigest} \pfun \mathit{Image}$$
$$tag : \mathit{Tag} \pfun \mathit{Image}$$

$$\forall\, d : \mathrm{dom}\, cd \bullet d.alg\,(cd\ d) = d.hash$$
$$\mathrm{ran}\, tag \subseteq \mathrm{ran}\, cd$$

The content digests identify the corresponding images. Each image identified by a tag is also identified by a content digest.

Initially, a repository is empty.

RepoInit

$$\mathit{Repo}'$$

$$cd' = \varnothing$$
$$tag' = \varnothing$$

An image is added to a repository by *pushing* it.

RepoPush

$$\Delta \mathit{Repo}$$
$$i? : \mathit{Image}$$
$$t? : \mathit{Tag}$$
$$d! : \mathit{ContentDigest}$$

$$tag' = tag \oplus \{t? \mapsto i?\}$$
$$cd' = cd \oplus \{d! \mapsto i?\}$$

The tag may be omitted in practice in which case it defaults to *Latest*. Note that the invariant of Repo' ensures that the output digest identifies the input image. However, there is some non-determinism here in the choice of algorithm.

An image is retrieved from a repository by *pulling* it.

We can either pull using a content digest

$$
\begin{array}{|l}
\hline
\textit{RepoPullByDigest} \\
\Xi \textit{Repo} \\
d? : \textit{OptionalContentDigest} \\
i! : \textit{Image} \\
\hline
\exists\, d : \textit{ContentDigest} \mid d? = \textit{Dig}\ d \land d \in \operatorname{dom} cd \bullet i! = cd\ d \\
\hline
\end{array}
$$

or, if a content digest is not supplied, by using a tag.

$$
\begin{array}{|l}
\hline
\textit{RepoPullByTag} \\
\Xi \textit{Repo} \\
t? : \textit{Tag} \\
d? : \textit{OptionalContentDigest} \\
i! : \textit{Image} \\
\hline
d? = \textit{None} \\
t? \in \operatorname{dom} \textit{tag} \Rightarrow i! = \textit{tag}\ t? \\
\hline
\end{array}
$$

A successful pull operation uses either a content digest or a tag.

$$\textit{RepoPullOk} \mathrel{\widehat=} \textit{RepoPullByDigest} \lor \textit{RepoPullByTag}$$

Failure cases, such as “not found”, are omitted from the model.

6 Registries

A registry is a collection of repositories index by path.

$$
\begin{array}{|l}
\textit{Registry} \\
\hline
repo : Path \pfun Repo \\
\hline
\end{array}
$$

Paths which do not exist are modelled as pointing to empty repositories.

Initially a registry has only empty repositories.

$$
\begin{array}{|l}
\textit{RegistryInit} \\
\hline
Registry' \\
\hline
\forall\, p : Path \bullet \\
\quad \exists\, RepoInit \bullet \\
\quad\quad repo'\, p = \theta Repo' \\
\hline
\end{array}
$$

We define a promotion schema[1] which operates on a single repository in a registry.

$$
\begin{array}{|l}
\textit{RegistryPromote} \\
\hline
\Delta Registry \\
\Delta Repo \\
p? : Path \\
\hline
p? \in \dom repo \\
\theta Repo = repo\, p? \\
repo' = repo \oplus \{p? \mapsto \theta Repo'\} \\
\hline
\end{array}
$$

The path must be valid and the registry is preserved except for the repository identified by the path which may be updated.

We then promote the repository push and pull operations to operate on a registry.

$$RegistryPush \mathrel{\widehat=} \exists\, \Delta Repo \bullet RepoPush \land RegistryPromote$$
$$RegistryPullOk \mathrel{\widehat=} \exists\, \Delta Repo \bullet RepoPullOk \land RegistryPromote$$

[1] Promotion schemas are used to turn operations on a particular type into operations on collection of the type which operate on a single member of the collection and leave the rest unchanged

Registries are arranged in a network indexed by hostname.

Net

$$reg : Hostname \pfun Registry$$

Initiallly, there are no registries in the network.

NetInit

$$Net'$$

$$reg' = \varnothing$$

We can add an empty registry to the network.

NetAddRegistryOk

$$\Delta Net$$
$$h? : Hostname$$

$$h? \notin \dom reg$$
$$\exists\, RegistryInit \bullet reg' = reg \cup \{h? \mapsto \theta Registry'\}$$

We can also remove a registry from the network.

NetRemoveRegistryOk

$$\Delta Net$$
$$h? : Hostname$$

$$h? \in \dom reg$$
$$reg' = \{h?\} \rhd reg$$

We define a promotion schema which operates on a single registry in a network.

NetPromote

$$\Delta Net$$
$$\Delta Registry$$
$$h? : Hostname$$

$$h? \in \dom reg$$
$$\theta Registry = reg\, h?$$
$$reg' = reg \oplus \{h? \mapsto \theta Registry'\}$$

The hostname must be valid and the network is preserved except for the

registry identified by the hostname which may be updated.

Finally, we promote the push and pull operations to work on a network.

$$NetPushOk \mathrel{\widehat=} \exists \Delta Registry \bullet RegistryPush \land NetPromote$$
$$NetPullOk \mathrel{\widehat=} \exists \Delta Registry \bullet RegistryPullOk \land NetPromote$$

Pushing and pulling can fail if there is no registry with the input hostname.

7 Image References

An image reference identifies an image in a registry. Let's remind ourselves what image references look like.

An image reference consists of a hostname (with optional port) and a path. The image reference may also contain a tag and/or a digest. The hostname determines the network location of a registry. The path consists of one or more components separated by forward slashes. The first component is sometimes, by convention for certain registries, a user name providing access control to the image.

Let's look at some examples:

- The image name `docker.io/istio/proxyv2` refers to an image with user name `istio` residing in the docker hub registry at `docker.io`.
- The image name `projectriff/builder:v1` is short-hand for `docker.io/projectriff/builder:v1` which refers to an image with user name `projectriff` also residing at `docker.io`. The image has tag `v1`.
- The image name `gcr.io/cf-elafros/knative-releases/github.com/knative/serving/cmd/autoscaler@sha256:deadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef` refers to an image with user name `cf-elafros` residing at `gcr.io`. The image has a (made up) `SHA-256` digest.

For the purposes of our model, an image reference consists of a hostname, a path, a tag, and an optional content digest.

Ref
host : *Hostname*
path : *Path*
tag : *Tag*
dig : *OptionalContentDigest*

A tag is always logically present, but if it is omitted from the textual representation of an image reference, it defaults to *Latest*. A content digest may be part of an image reference or may be omitted.

So far the push and pull operations have accumulated several input parameters.

$$
\begin{array}{|l}
\hline
\textit{PushParms} \\
h? : \textit{Hostname} \\
p? : \textit{Path} \\
t? : \textit{Tag} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{PullParms} \\
h? : \textit{Hostname} \\
p? : \textit{Path} \\
t? : \textit{Tag} \\
d? : \textit{OptionalContentDigest} \\
\hline
\end{array}
$$

An image reference is mapped to push input parameters as follows.

$$
\begin{array}{|l}
\hline
\textit{RefPushParms} \\
r? : \textit{Ref} \\
\textit{PushParms} \\
\hline
h? = r?.\textit{host} \\
p? = r?.\textit{path} \\
t? = r?.\textit{tag} \\
r?.\textit{dig} = \textit{None} \\
\hline
\end{array}
$$

Pushing is not allowed if the image reference has a content digest.

An image reference is mapped to pull input parameters as follows.

$$
\begin{array}{|l}
\hline
\textit{RefPullParms} \\
r? : \textit{Ref} \\
\textit{PullParms} \\
\hline
h? = r?.\textit{host} \\
p? = r?.\textit{path} \\
t? = r?.\textit{tag} \\
d? = r?.\textit{dig} \\
\hline
\end{array}
$$

Push and pull can then be reframed to take an image reference instead of the corresponding input parameters.

$$\textit{RefPushOk} \mathrel{\widehat=} \exists\, \textit{PushParms} \bullet \textit{NetPushOk} \land \textit{RefPushParms}$$
$$\textit{RefPullOk} \mathrel{\widehat=} \exists\, \textit{PullParms} \bullet \textit{NetPullOk} \land \textit{RefPullParms}$$

A Z Notation

Numbers:

$\mathbb{N}$ `Natural numbers {0,1,...}`

Propositional logic and the schema calculus:

$\ldots \land \ldots$	`And`	$\langle\!\langle \ldots \rangle\!\rangle$	`Free type injection`
$\ldots \lor \ldots$	`Or`	$[\ldots]$	`Given sets`
$\ldots \Rightarrow \ldots$	`Implies`	$', ?, !, _0 \ldots _9$	`Schema decorations`
$\forall .. \mid .. \bullet ..$	`For all`	$\ldots \vdash \ldots$	`theorem`
$\exists .. \mid .. \bullet ..$	`There exists`	$\theta \ldots$	`Binding formation`
$\ldots \setminus \ldots$	`Hiding`	$\lambda \ldots$	`Function definition`
$\ldots \mathrel{\widehat=} \ldots$	`Schema definition`	$\mu \ldots$	`Mu-expression`
$\ldots == \ldots$	`Abbreviation`	$\Delta \ldots$	`State change`
$\ldots ::= \ldots \mid \ldots$	`Free type definition`	$\Xi \ldots$	`Invariant state change`

Sets and sequences:

$\{\ldots\}$	`Set`	$\ldots \setminus \ldots$	`Set difference`
$\{.. \mid .. \bullet ..\}$	`Set comprehension`	$\bigcup \ldots$	`Distributed union`
$\mathbb{P} \ldots$	`Set of subsets of`	$\# \ldots$	`Cardinality`
$\varnothing$	`Empty set`	$\ldots \subseteq \ldots$	`Subset`
$\ldots \times \ldots$	`Cartesian product`	$\ldots \subset \ldots$	`Proper subset`
$\ldots \in \ldots$	`Set membership`	$\ldots$ partition $\ldots$	`Set partition`
$\ldots \notin \ldots$	`Set non-membership`	seq	`Sequences`
$\ldots \cup \ldots$	`Union`	$\langle \ldots \rangle$	`Sequence`
$\ldots \cap \ldots$	`Intersection`	disjoint $\ldots$	`Disjoint sequence of sets`

Functions and relations:

$\ldots \leftrightarrow \ldots$	`Relation`	$\ldots ^*$	`Reflexive-transitive closure`
$\ldots \nrightarrow \ldots$	`Partial function`	$\ldots (\!\mid \ldots \mid\!)$	`Relational image`
$\ldots \rightarrow \ldots$	`Total function`	$\ldots \oplus \ldots$	`Functional overriding`
$\ldots \rightarrowtail \ldots$	`Partial injection`	$\ldots \lhd \ldots$	`Domain restriction`
$\ldots \rightarrowtail \ldots$	`Injection`	$\ldots \rhd \ldots$	`Range restriction`
dom $\ldots$	`Domain`	$\ldots \lhd \ldots$	`Domain subtraction`
ran $\ldots$	`Range`	$\ldots \rhd \ldots$	`Range subtraction`
$\ldots \mapsto \ldots$	`maplet`		
$\ldots ^\sim$	`Relational inverse`		

Axiomatic descriptions:

Declarations

Predicates

Schema definitions:

SchemaName

Declaration

Predicates

Decorations:

(*undecorated*)	`Input state`	?	`Input parameter`
$'$	`Output state`	!	`Output parameter`