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.

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`).

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.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{ContentDigest}} \\
\hline
\mathit{alg} : \mathit{Image} \rightarrow \mathit{Hex} \\
\mathit{hash} : \mathit{Hex} \\
\hline
\end{array}
$$

The idea is that a content digest d identifies an image i if and only if:

$$d.\mathit{alg}\ i = d.\mathit{hash}$$

An optional content digest is modelled as a datatype.

$$\mathit{OptionalContentDigest} ::= \mathit{None} \mid \mathit{Dig}\langle\langle \mathit{ContentDigest} \rangle\rangle$$

5 Repositories

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

$$
\begin{array}{|l}
\hline
\textit{Repo} \\
cd : ContentDigest \pfun Image \\
tag : Tag \pfun Image \\
\hline
\forall d : \mathrm{dom}\, cd \bullet d.alg\,(cd\ d) = d.hash \\
\mathrm{ran}\, tag \subseteq \mathrm{ran}\, cd \\
\hline
\end{array}
$$

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.

$$
\begin{array}{|l}
\hline
\textit{RepoInit} \\
Repo' \\
\hline
cd' = \varnothing \\
tag' = \varnothing \\
\hline
\end{array}
$$

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

$$
\begin{array}{|l}
\hline
\textit{RepoPush} \\
\Delta Repo \\
i? : Image \\
t? : Tag \\
\hline
tag' = tag \oplus \{t? \mapsto i?\} \\
\exists d : ContentDigest \bullet \\
\qquad cd' = cd \oplus \{d \mapsto i?\} \\
\hline
\end{array}
$$

The tag may be omitted in practice in which case it defaults to *Latest*. Note that the invariant of *Repo′* ensures that the chosen digest identifies the input image. However, these 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

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

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

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

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

6 Registries

A registry is a collection of repositories index by path.

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

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

Initially a registry has only empty repositories.

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

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

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

We then promote the push and pull operations.

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

Registries are arranged in a network indexed by hostname.

$$\begin{array}{|l}
\hline
\textit{Net} \\
\textit{reg} : \textit{Hostname} \nrightarrow \textit{Registry} \\
\hline
\end{array}$$

Initiallly, there are no registries in the network.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NetInit}} \\
\textit{Net}' \\
\hline
\textit{reg}' = \varnothing \\
\hline
\end{array}
$$

We can add an empty registry to the network.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NetAddRegistryOk}} \\
\Delta \textit{Net} \\
h? : \textit{Hostname} \\
\hline
h? \notin \operatorname{dom} \textit{reg} \\
\exists \textit{RegistryInit} \bullet \textit{reg}' = \textit{reg} \cup \{h? \mapsto \theta \textit{Registry}'\} \\
\hline
\end{array}
$$

We can also remove a registry from the network.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NetRemoveRegistryOk}} \\
\Delta \textit{Net} \\
h? : \textit{Hostname} \\
\hline
h? \in \operatorname{dom} \textit{reg} \\
\textit{reg}' = \{h?\} \ntriangleleft \textit{reg} \\
\hline
\end{array}
$$

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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NetPromote}} \\
\Delta \textit{Net} \\
\Delta \textit{Registry} \\
h? : \textit{Hostname} \\
\hline
h? \in \operatorname{dom} \textit{reg} \\
\theta \textit{Registry} = \textit{reg}\ h? \\
\textit{reg}' = \textit{reg} \oplus \{h? \mapsto \theta \textit{Registry}'\} \\
\hline
\end{array}
$$

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

$$
\begin{array}{l}
\textit{NetPushOk} \mathrel{\widehat{=}} \exists \Delta \textit{Registry} \bullet \textit{RegistryPush} \land \textit{NetPromote} \\
\textit{NetPullOk} \mathrel{\widehat{=}} \exists \Delta \textit{Registry} \bullet \textit{RegistryPullOk} \land \textit{NetPromote}
\end{array}
$$

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

7 Image References

An image reference identifies an image in a registry.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Ref}} \\
\hline
host : Hostname \\
path : Path \\
tag : Tag \\
dig : OptionalContentDigest \\
\hline
\end{array}
$$

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.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{PushParms}} \\
\hline
h? : Hostname \\
p? : Path \\
t? : Tag \\
d? : ContentDigest \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{PullParms}} \\
\hline
h? : Hostname \\
p? : Path \\
t? : Tag \\
d? : OptionalContentDigest \\
\hline
\end{array}
$$

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

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RefPushParms}} \\
\hline
r? : Ref \\
PushParms \\
\hline
h? = r?.host \\
p? = r?.path \\
t? = r?.tag \\
r?.dig = None \\
\hline
\end{array}
$$

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

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

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

Push can then be reframed to take an image reference.

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

A Z Notation

Numbers:

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

Propositional logic and the schema calculus:

$\ldots \land \ldots$	And
$\ldots \lor \ldots$	Or
$\ldots \Rightarrow \ldots$	Implies
$\forall .. \mid .. \bullet ..$	For all
$\exists .. \mid .. \bullet ..$	There exists
$\ldots \setminus \ldots$	Hiding
$\ldots \widehat{=} \ldots$	Schema definition
$\ldots == \ldots$	Abbreviation
$\ldots ::= \ldots \mid \ldots$	Free type definition

$\langle\!\langle \ldots \rangle\!\rangle$	Free type injection
$[\ldots]$	Given sets
$', ?, !, _0 \ldots _9$	Schema decorations
$\ldots \vdash \ldots$	theorem
$\theta \ldots$	Binding formation
$\lambda \ldots$	Function definition
$\mu \ldots$	Mu-expression
$\Delta \ldots$	State change
$\Xi \ldots$	Invariant state change

Sets and sequences:

$\{\ldots\}$	Set
$\{.. \mid .. \bullet ..\}$	Set comprehension
$\mathbb{P} \ldots$	Set of subsets of
$\varnothing$	Empty set
$\ldots \times \ldots$	Cartesian product
$\ldots \in \ldots$	Set membership
$\ldots \notin \ldots$	Set non-membership
$\ldots \cup \ldots$	Union
$\ldots \cap \ldots$	Intersection

$\ldots \setminus \ldots$	Set difference
$\bigcup \ldots$	Distributed union
$\# \ldots$	Cardinality
$\ldots \subseteq \ldots$	Subset
$\ldots \subset \ldots$	Proper subset
$\ldots \;\mathsf{partition}\; \ldots$	Set partition
seq	Sequences
$\langle \ldots \rangle$	Sequence
$\mathsf{disjoint}\; \ldots$	Disjoint sequence of sets

Functions and relations:

$\ldots \leftrightarrow \ldots$	Relation
$\ldots \nrightarrow \ldots$	Partial function
$\ldots \rightarrow \ldots$	Total function
$\ldots \rightarrowtail\!\!\!\!\!\nrightarrow \ldots$	Partial injection
$\ldots \rightarrowtail \ldots$	Injection
$\mathrm{dom} \ldots$	Domain
$\mathrm{ran} \ldots$	Range
$\ldots \mapsto \ldots$	maplet
$\ldots^{\sim}$	Relational inverse

$\ldots^{*}$	Reflexive-transitive closure
$\ldots (\!| \ldots |\!)$	Relational image
$\ldots \oplus \ldots$	Functional overriding
$\ldots \lhd \ldots$	Domain restriction
$\ldots \rhd \ldots$	Range restriction
$\ldots ⩤ \ldots$	Domain subtraction
$\ldots ⩥ \ldots$	Range subtraction

Axiomatic descriptions:

Declarations

Predicates

Schema definitions:

SchemaName

Declaration

Predicates