OCI Image Spec Support in Mesos

NOTE: This is a public document.

Authors	Qian Zhang <zhq527725@gmail.com>
Shepherd	Jie Yu <jie@mesosphere.io>
Revision	0.1
Last Revision	Oct. 6, 2016
Status	Initial draft

Motivation

OCI image spec is approaching 1.0, we should add OCI image spec support to the unified containerizer so that users can launch containers with OCI images using unified containerizer.

JIRA Ticket

MESOS-5011

Interface Changes

Agent flags

We will introduce three Mesos agent flags for operator to launch Mesos agent with OCI image supported.

- **--oci_discovery**: The way to discover OCI images, currently we only support “prefix”, and we may support “registry” in future. (default: prefix).
- **--oci_discovery_prefix**: URI prefix to be used for discovery of OCI images, e.g., /tmp/oci/images/, http://, https://, hdfs://:9000/user/abc/cde. (default: http://).

    - The OCI images put in the location specified by this flag must conform to OCI image layout spec, and agent will fetch the specified image from this location when launching container. E.g., if the value of this flag is a local directory, then each subdirectory in it represents an OCI images and the hierarchy in each subdirectory must conform to OCI image layout spec.
- **--oci_store_dir**: Directory the OCI provisioner will store images in. (default: /var/lib/mesos/store/oci). See the section "OCI Image Store Layout" for the detailed layout of this directory.

There is an existing Mesos agent flag "**--image_providers**" which is used to specify the image providers, currently operator can only specify APPC and/or DOCKER for it. In this project, we will introduce one more supported image provider "OCI" for it.

Protobuf Messages

Add a new .proto file "include/mesos/oci/spec.proto" in which we will define some protobuf messages for

- OCI content descriptor
- OCI image manifest list
- OCI image manifest
- OCI image configuration

Please check the following patches for the draft version of the protobuf messages:
https://reviews.apache.org/r/52349/diff/5#1

In "include/mesos/mesos.proto" and "include/mesos/v1/mesos.proto", update the protobuf message "Image" by adding OCI related information like below:

```
message Image {
  enum Type {
    APPC = 1;
    DOCKER = 2;
    OCI = 3;
  }
  message Appc {
    ...
  }
  message Docker {
     ...
  }
  message OCI {
    // The name of the image.
    required string name = 1;
    // The tag of the image.
    optional string tag = 2 [default = "latest"];
  }
```

```
  required Type type = 1;
  optional Appc appc = 2;
  optional Docker docker = 3;
  optional OCI oci = 4;
  ...
}
```

Architecture Design

To support fetching/storing OCI image and run it as a container with unified containerizer, we need to introduce two major components in Mesos agent: OCI store and OCI runtime isolator.

OCI Store

We need to implement an OCI store under “src/slave/containerizer/mesos/provisioner/oci/” to fetch and store the OCI image that framework specifies in the “Image.oci” protobuf message. Operator needs to specify a URI prefix via the flag “--oci_discovery_prefix” when starting Mesos agent. For this URI prefix, we will support these schemes: file, http, https, ftp, ftps, hdfs, hftp, s3, s3n, and OCI store will call fetcher (“mesos::uri::Fetcher”) which will internally call the following existing fetcher plugins based on the scheme to fetch the OCI image.

- **CopyFetcherPlugin**: file
- **CurlFetcherPlugin**: http, https, ftp, ftps
- **HadoopFetcherPlugin**: hdfs, hftp, s3, s3n

The layout of OCI images to be fetched must conform to OCI image layout spec, the below is an example which contains two images (busybox and nginx).

```
busybox/
├ blobs
│   └ sha256
│       ├ 2a2b6168c040d25148a4972cb20108b737adc51685d4dc68a8e01d55416d27a1
│       ├ 2b8fd9751c4c0f5dd266fcae00707e67a2545ef34f9a29354585f93dac906749
│       └ 8ddc19f16526912237dd8af81971d5e4dd0587907234be2b83e249518d5b673f
├ oci-layout
└ refs
│   └ latest
nginx/
│…
```

Skopeo can be used to pull a Docker image from Docker Hub and convert it to an OCI image layout, e.g., the following command will pull the Docker image “busybox” and convert it to an OCI image layout in the directory “busybox-oci”.

```
# skopeo copy docker://busybox oci:busybox-oci
```

And Docker is also trying to support OCI image layout in docker save/load, see the following PR for details:
https://github.com/docker/docker/pull/26369

The overall flow to fetch an OCI image is:

1. Fetch the ref which is an OCI content descriptor based on “Image.oci.name”, e.g., if “Image.oci.name” is “busybox” and “Image.oci.tag” is “latest”, we will fetch “busybox/refs/latest” in the example above.
2. Parse the ref to get its media type which can be either an OCI image manifest list or an OCI image manifest, if it is the later, go to step 4.
3. Fetch and parse the OCI image manifest list which may contains multiple OCI image manifest (each is for a platform, e.g., amd64, ppc64le, etc.).
4. Fetch the OCI image manifest for the current platform (e.g., if Mesos agent is running on an amd64 machine, we will only fetch the OCI image manifest of the amd64 platform).
5. Parse the OCI image manifest which will contain an OCI image configuration and one or multiple layers.
6. Fetch the OCI image configuration.
7. Fetch each layer which is a tar ball.

Inside of OCI store, we will implement a cache which will keep all the OCI images that have already been fetched. So before calling fetcher to fetch an OCI image, we will check whether the “Image.cached” is true AND the image is already in cache, if yes, then OCI store will not call fetcher, instead it will directly return the image in cache. And before fetching a layer (the step 6 above), we will check whether it is already in the image store directory (i.e., whether the layer already exists under /var/lib/mesos/store/oci/layers), if yes, we will not fetch it.

OCI Runtime Isolator

Similar with Docker runtime isolator and Appc runtime isolator, we will implement an OCI runtime isolator which will prepare the below for the container to be launched by parsing the OCI image configuration fetched by OCI store.

- Environmental variables
- Command
- Working directory
- User

OCI Image Store Layout

```
Image store dir (“--oci_store_dir” agent flag)
   |-- staging/
   |    |-- <staging_tmp_dir_XXXXXX>/
   |         |-- <layer_id>/
   |         |-- <layer_id>/
```

```
|       |-- <config.json>
|-- layers/
|       |-- <layer_id>/
|       |-- <layer_id>/
|       |-- <config.json>
|-- storedImages
```

In the layout above,

1. “<layer_id>” is a directory which contains the filesystem contents of the corresponding image layer (unpacked from the fetched image layer blob), the directory name is the sha256 digest of the image layer blob.
2. “<config.json>” is the OCI image configuration JSON file of the corresponding image, the filename is the sha256 digest of the file itself.
3. “storedImages” is a file which persists the information of the fetched images. In this file, for each fetched image, we persist its image name and its image manifest. So when agent is restarted, we will recover the information of this file so that we will know what images have been fetched in the store.

Future Works

1. Consider to use libcurl rather than subprocess + curl to fetch the image.
2. Implement garbage collection for the images in the store.