In this article, we will introduce the concept of containers, its use on application distribution and execution, and the Open Container Iniciative project.

What is a container?

A container is a mechanism for packaging and isolating an application runtime environment. It solves the multi-application dependency resolution at build time and guarantees resources availability at runtime.

As Solomon Hykes (co-founder of Docker) explained in 2013, this concept comes from shipping containers: boxes of the same size, shape and locking mechanism, that can be managed evenly regardless of its contents.

The main goal when creating an application container, is to make the code behave the same way on every platform or environment, being those a developer's machine, a continuous integration server, or on-premise/cloud production stack.

For this goal to be met, the application code should be packed along with its dependencies, libraries, extra files, filesystem layouts, etc. This bundle is known as *container image*, or *image* for short. These images are read only and they are used as a template for an application execution.

A *container* is a running instance of an image, with a top writable layer that holds any changes made by the application during its lifetime.

Container operations

There are two sets of common operations that are used on containers, among them:

- issued by developers, who need to build images of their software by
  - Push and pull: to be able to reuse and distribute them
  - Copy files: to be able to populate an image with source code and other type of files
  - Inspect and Sign: to be able to ensure and verify the sanity of images
- issued by operators, who need to manage developers application by
  - Start and stop: to be able to create an instance of an application
  - Inspect: to be able to check the status of a running application
  - Scale: to be able to make adapt the resource usage of the application

OCI - Open Container Initiative

The Open Container Initiative (OCI) is a project that was born with the mission of defining open industry standards on container formats and runtime engines. It is included in the many projects of The Linux Foundation. It was launched on June 2015 by Docker, CoreOS and other big players in the container industry.

OCI Specifications

The OCI contains two specifications:

The *Runtime Specification* (runtime-spec)

> The runtime specification aims to specify the configuration, execution environment and lifecycle of a container.

Docker donated its runtime, runc, to the OCI to serve as the first implementation of the standard. This implementations is also known as **Reference Runtime Implementation**.

The *Image Specification* (image-spec)

> defines the on-disk format for container images as well as the metadata ,which defines things like
>
> - information to launch the application (command, argument, environment variables)
> - dependencies and filesystem serialization archive references
> - filesystem layer serialization

The original image format used by Docker has become the image specification, and there are various open source tools that supports it:

Buildah

> Buildah is an open-source command line tool that allows builing OCI container images either from a working container or via the instructions in a Dockerfile.

Podman

> Podman is an open-source command line tool for managing containers and images. It helps to maintain and modify OCI images, such as pulling and tagging and it also allows you to create, run, and maintain containers created from those images.

Buildah is an efficient way to create OCI images while Podman allows you to manage and maintain those images and containers in a production environment.

Both projects are complementary as they differ in the notion of container they work with. Buildah containers are created to add content to or modify an image, while Podman containers tend to live longer as they provide a service.

Buildah vs Dockerfile

As stated in Docker's documentation: *"a Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image"*.

Although Buildah can build an OCI image from a Dockerfile using the `build-using-dockerfile` (or `bud` for short), it replicates all of the commands that are found in a Dockerfile as command-line options. This allows building images without Dockerfiles while not requiring any root privileges. This type of flexibility comes handy when integrating multiple scripting languages into the build process.

Running Containers

In this example, we're going to use `buildah` and `podman` for container creation and run. We'll create an application based on sample go application that answers *"Hello World"* in response to a HTTP request.

First of all, we'll use `buildah` to use the official golang image and customize it for running our sample application

```
$ buildah from golang:1.15.3
Getting image source signatures
Copying blob e4c3d3e4f7b0 done
...
Copying config 4a581cd6fe done
Writing manifest to image destination
Storing signatures
golang-working-container
```

Our app uses a couple of libraries, so we need to install them (our dependencies).

```
$ buildah config --workingdir='/app/' golang-working-container
$ buildah run golang-working-container -- go get -u github.com/go-sql-driver/mysql
$ buildah run golang-working-container -- go get -u github.com/gorilla/mux
```

Then, copy our application code into the container image and compile it

```
$ buildah copy golang-working-container sample-go-webapp/main.go /app/main.go
9baa46d25c88363b593b642b61393c4ba07d650b5f00e96e802fcbd4035965a0
$ buildah run golang-working-container -- go build -o app main.go
```

We're almost done. As the last step, we need to define which port our application will be listening on and how it will be invoked (we'll use port 8888 and the `app` binary created on the previous).

```
$ buildah config --port=8888 --entrypoint='./app' golang-working-container
```

With all our code, dependencies and configuration in place, now we can create the new image containing our application.

```
$ buildah commit golang-working-container mysampleapp
Getting image source signatures
Copying blob 9780f6d83e45 skipped: already exists
...
Copying blob 097a6340cee5 done
Copying config c794a01a12 done
Writing manifest to image destination
Storing signatures
c794a01a1259922d63ef11dee6cb0c09b7014423d326079ce26b484b200d21f1
```

Checking the images available, we'll find that, besides the official golang image, there's one named `localhost/mysampleapp`.

```
$ buildah images
REPOSITORY                 TAG      IMAGE ID       CREATED          SIZE
localhost/mysampleapp      latest   c794a01a1259   17 seconds ago   873 MB
docker.io/library/golang   1.15.3   4a581cd6feb1   5 weeks ago      860 MB
```

TIP We'll achieve the same result if we use a Dockerfile like this which includes all the previous commands using the `bud` buildah command option.

Once we have an image ready, we use `podman` to run our application. We can verify that `podman` is able to use our image by listing them.

```
# podman images
REPOSITORY                 TAG      IMAGE ID       CREATED         SIZE
localhost/mysampleapp      latest   c794a01a1259   2 minutes ago   873 MB
docker.io/library/golang   1.15.3   4a581cd6feb1   5 weeks ago     860 MB
```

Then we will map the port configured in the container to port 9999 (just for sake of this example) on this system, and then we can verify if our app is running successfully by making a request to its port.

```
# podman run -p 9999:8888 localhost/mysampleapp
Starting server ...
```

On a different session, first we can check that our container is running

```
# podman ps
CONTAINER ID  IMAGE                         COMMAND  CREATED         STATUS
PORTS                   NAMES
03874446bad9  localhost/mysampleapp:latest  bash     29 seconds ago  Up 27 seconds ago
0.0.0.0:9999->8888/tcp  strange_greider
```

and make a request to see if we have the expected response.

```
# curl http://localhost:9999
Hello world!
```

It worked!

Summary

The use of containers brings some benefits like

- increades portability
- consistent operation
- less overhead than virtual machines

In this article, we've seen that:

- Container and images are a software package mechanism.
- OCI is a project that defines standards for containers management.
- Tools that comply to this standards ensure interoperability.
- Buildah and Podman are example of tools that works with OCI containers.

References

- https://opencontainers.org/
- https://buildah.io/
- https://podman.io/
- https://www.padok.fr/en/blog/container-docker-oci