rx-m cloud native & ai training & consulting

Containers

Images Overview Lab Solutions

6. CHALLENGE: add a Linux distro with a filesystem

Fix the `hello-world:v2` image by using an OS distribution so that users can `exec` into the running container. Build the image, tagging the new image version `hello-world:v3`; then run a container from the image and exec into it to prove it can be done!

In your Dockerfile, use a distro in the second `FROM` instruction instead of "`scratch`", such as alpine:

```
~/multi-build$ cp Dockerfile Dockerfile-orig

~/multi-build$ nano Dockerfile && cat $_
```

```
FROM golang:1.18 AS build-env
WORKDIR /go/src/hello/
COPY ./hello.go /go/src/hello/
RUN go mod init example.com
RUN ["go","build","-tags","netgo","-o","hello"]

FROM alpine
COPY --from=build-env /go/src/hello/hello hello
EXPOSE 8080
ENTRYPOINT ["./hello"]
```

```
~/multi-build$
```

Now build the image with the same repo name `hello-build` with a `v3` tag:

```
~/multi-build$ docker image build -t hello-build:v3 .

[+] Building 2.8s (13/13) FINISHED
docker:default
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 308B
```

```
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/alpine:latest
1.9s
 => [internal] load metadata for docker.io/library/golang:1.20
1.2s
 => [build-env 1/5] FROM
docker.io/library/golang:1.20@sha256:bfc60723228b88180b1e15872eb435cf7e6d8
199eae9  0.0s
 => [internal] load build context
0.0s
 => => transferring context: 30B
0.0s
 => [stage-1 1/2] FROM
docker.io/library/alpine@sha256:51b67269f354137895d43f3b3d810bfacd3945438e
94dc5ac55  0.5s
 => => resolve
docker.io/library/alpine@sha256:51b67269f354137895d43f3b3d810bfacd3945438e
94dc5ac55fdac3403  0.0s
 => =>
sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48
1.64kB / 1.64kB              0.0s
 => =>
sha256:13b7e62e8df80264dbb747995705a986aa530415763a6c58f84a3ca8af9a5bcd
528B / 528B                  0.0s
 => =>
sha256:f8c20f8bbcb684055b4fea470fdd169c86e87786940b3262335b12ec3adef418
1.47kB / 1.47kB              0.0s
 => =>
sha256:661ff4d9561e3fd050929ee5097067c34bafc523ee60f5294a37fd08056a73ca
3.41MB / 3.41MB              0.3s
 => => extracting
sha256:661ff4d9561e3fd050929ee5097067c34bafc523ee60f5294a37fd08056a73ca
0.2s
 => CACHED [build-env 2/5] WORKDIR /go/src/hello/
0.0s
 => CACHED [build-env 3/5] COPY ./hello.go /go/src/hello/
0.0s
 => CACHED [build-env 4/5] RUN ["go","mod","init","example.com/hello"]
0.0s
 => CACHED [build-env 5/5] RUN ["go","build","-tags","netgo"]
0.0s
 => [stage-1 2/2] COPY --from=build-env /go/src/hello/hello hello
0.2s
 => exporting to image
0.1s
 => => exporting layers
0.1s
 => => writing image
sha256:2220a0adf07d7d4694734437aca696c0d06eba26698961f8ed1d64f743655681
0.0s
```

```
 => => naming to docker.io/library/hello-build:v3
0.0s

~/multi-build$
```

List our hello-build images:

```
~/multi-build$ docker image ls hello-build

hello-build   v3        2220a0adf07d   19 seconds ago   14MB
hello-build   v2        163ebaa35f52   5 minutes ago    6.63MB
hello-build   v1        eff67c22e949   8 minutes ago    846MB

~/multi-build$
```

Using another distro like Ubuntu or CentOS your image wouldn't be nearly as lean, but in this case we've only added ~6 MB!

Run a new container from our new image:

```
~/multi-build$ docker container run -d --name challenge hello-build:v3

c1f0325df1125cc9df981760ef112687898587ae141c3ed36c18fe5c679f6a06

~/multi-build$
```

Use the `exec` command to take a look inside our lightweight container:

```
~/multi-build$ docker container exec -it challenge /bin/sh

/ # ls -l

total 6536
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 bin
drwxr-xr-x    5 root     root           340 Jan 18 00:44 dev
drwxr-xr-x    1 root     root          4096 Jan 18 00:44 etc
-rwxr-xr-x    1 root     root       6633021 Jan 18 00:39 hello
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 home
drwxr-xr-x    7 root     root          4096 Dec  7 09:43 lib
drwxr-xr-x    5 root     root          4096 Dec  7 09:43 media
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 mnt
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 opt
dr-xr-xr-x  180 root     root             0 Jan 18 00:44 proc
drwx------    1 root     root          4096 Jan 18 00:44 root
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 run
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 sbin
drwxr-xr-x    2 root     root          4096 Dec  7 09:43 srv
```

```
dr-xr-xr-x   13 root     root             0 Jan 18 00:44 sys
drwxrwxrwt    2 root     root          4096 Dec  7 09:43 tmp
drwxr-xr-x    7 root     root          4096 Dec  7 09:43 usr
drwxr-xr-x   12 root     root          4096 Dec  7 09:43 var

/ #

~/multi-build$
```

Success!

Exit the container:

```
/ # exit

~/multi-build$
```

7. CHALLENGE: tagging images

Add an appropriate tag to your production go program image and push it to your local registry. Verify your registry has stored your image.

In the example below we'll tag the go program image as `localhost:5000/hello-world` repo with the `prod` tag to push to the local repository.

```
~/multi-build$ docker image tag hello-build:v3 localhost:5000/hello-
world:prod

~/multi-build$ docker image ls | grep hello

hello-build                  v3        2220a0adf07d   About a minute ago
14MB
localhost:5000/hello-world   prod      2220a0adf07d   About a minute ago
14MB
hello-build                  v2        163ebaa35f52   5 minutes ago
6.63MB
hello-build                  v1        eff67c22e949   9 minutes ago
846MB

~/multi-build$ docker image push localhost:5000/hello-world:prod

The push refers to repository [localhost:5000/hello-world]
a627f566a8be: Pushed
5af4f8f59b76: Pushed
prod: digest:
sha256:7a2b4c25edff8d0be8f846c25a52a2492e6c33440aa99f14c78887f673d0f1a0
size: 739

~/multi-build$
```

```
```

Verify with curl:

```
~/multi-build$ curl -X GET localhost:5000/v2/_catalog

{"repositories":["devenv","hello-world"]}

~/multi-build$
```