APPENDIX

ACTIVITY 16.01: CREATING YOUR OWN DEVELOPMENT BRANCH FOR DOCKER

Solution:

There are a number of ways in which we perform the first activity of this chapter. The following steps show one way to do this:

1. To get started, clone the latest version of Docker CE from its GitHub repository. Run the following **`clone`** command in your working directory:

```
git clone https://github.com/docker/docker-ce
```

   It can be observed that the latest version of Docker CE from its GitHub repository is cloned:

```
Cloning into 'docker-ce'...
remote: Enumerating objects: 79, done.
remote: Counting objects: 100% (79/79), done.
remote: Compressing objects: 100% (53/53), done.
remote: Total 421237 (delta 12), reused 54 (delta 9), pack-reused
421158
Receiving objects: 100% (421237/421237), 163.66 MiB | 19.90 MiB/s,
done.
Resolving deltas: 100% (216197/216197), done.
Checking connectivity... done.
```

2. Move into the new **`docker-ce`** directory that was created when you cloned the Docker CE GitHub repository:

```
cd docker-ce
```

3. As you won't be making changes to the master branch of the code, you first need to make a branch of this code to start developing on it. Run the following **`git checkout`** command, which creates a branch named **`0.1.0_dev`** from the **`master`** branch:

```
git checkout -b 0.1.0_dev master


Switched to a new branch '0.1.0_dev'
```

   If you were serious about making changes to the code base, you could create a fork of the GitHub code and then branch from the forked repository. If you then wanted your changes to be made available to the general public, you could request these changes to be merged back into the master branch of the original code.

4. Make a very minor change to the code in the Docker client, as shown in the following code block. It will now display **`Docker Engine – Workshop`** instead of **`Docker Engine – Community`**. This is only a minor change but will hopefully demonstrate how to make minor changes without really needing to know a lot about the Go programming language. To make the change, open the **`components/packaging/common.mk`** file with your text editor and make sure *line 6* looks as in the following highlighted line:

```
1 ARCH=$(shell uname -m)
2 BUILDTIME=$(shell date -u -d "@$${SOURCE_DATE_EPOCH:-$$
    (date +%s)}" --rfc-3339 ns 2> /dev/null | sed -e 's/ /T/')
3 DEFAULT_PRODUCT_LICENSE:=Community Engine
4 DOCKER_GITCOMMIT:=abcdefg
5 GO_VERSION:=1.13.9
6 PLATFORM=Docker Engine – Workshop
7 SHELL:=/bin/bash
8 VERSION?=0.0.0-dev
9
10 export BUILDTIME
11 export DEFAULT_PRODUCT_LICENSE
12 export PLATFORM
```

5. Make sure your changes are saved and then build your new binary from the source code by running the following command to create a static binary image for Linux:

```
make static DOCKER_BUILD_PKGS=static-linux
```

You should get the following output:

```
make VERSION=18.10.0-ce-dev CLI_DIR=/tmp/docker-ce/components/
cli ENGINE_DIR=/tmp/docker-ce/components/engine -C /tmp/docker-ce/
components/packaging static
make[1]: Entering directory '/tmp/docker-ce/components/packaging'
…
Done
tar -C build/linux -c -z -f build/linux/docker-rootless-
extras-0.0.0-20200411012732-6fc1b40.tgz docker-rootless-extras
make[2]: Leaving directory '/tmp/docker-ce/components/packaging
/static'
make[1]: Leaving directory '/tmp/docker-ce/components/packaging'
```

6. Once the build has completed, verify that you have created a package by listing all the files in the **components/packaging/static/build/linux/** directory:

```
ls -l components/packaging/static/build/linux/
```

You should see something similar to the following as you now have a new compressed and zipped Docker package with your version set to **0.0.0** and the timestamp of the data:

```
total 79056
drwxr-xr-x 2   root   root     4096       Apr 13 01:46
docker
-rw-r--r-- 1   root   root     61046601   Apr 13 02:01
docker-0.0.0-20200411012732-6fc1b40.tgz
drwxr-xr-x 2   root   root     4096       Apr 13 01:46
docker-rootless-extras
-rw-r--r-- 1   root   root     19896326   Apr 13 02:01
docker-rootless-extras-0.0.0-20200411012732-6fc1b40.tgz
```

7. As you did earlier in this chapter, create a new container image running the latest Ubuntu image to test the new Docker binary you created. Create a new container with the following command, which will use the name **activity1-docker-source**:

```
docker run --rm -itd -v /var/run/docker.sock:/var/run/docker.sock
--name activity1-docker-source ubuntu
```

You should get output like the following:

```
e6168d56144df928043fd67fb84213c9d38b233ec17e6ea2ef6b97bb8ac2c1a9
```

8. Copy the new Docker package we created from the **components/packaging/static/build/linux** subdirectory to the new container we created in the previous step, using the **docker cp** command, as follows:

```
docker cp components/packaging/static/build/linux/docker-0.0.0-
20200411012732-6fc1b40.tgz activity1-docker-source:/tmp/
```

9. Log in to the new container with the following command, ready to start testing the new package:

```
docker exec -it activity1-docker-source /bin/bash
```

10. Uncompress the package file that was copied into the new container image. It is currently in the **`tmp`** directory and uses the following command to extract the new binary:

```
tar zxvf /tmp/docker-0.0.0-20200411012732-6fc1b40.tgz
```

The following are the application files for Docker:

```
docker/
docker/ctr
docker/dockerd
docker/containerd
docker/docker-init
docker/docker-proxy
docker/docker
docker/containerd-shim
docker/runc
```

11. Move into the new Docker directory created when you extracted the new Docker package:

```
cd docker
```

12. Run the **`docker version`** command shown in the following line:

```
./docker version
```

This should result in the following output, in which the client now shows **`Docker Engine – Workshop`**, as per the changes you made in *step 4*. You will observe that the version is the same as the package file you created when you built the new binary:

```
Client: Docker Engine – Workshop
 Version:           0.0.0-20200411012732-6fc1b40
 API version:       1.40 (downgraded from 1.41)
 Go version:        go1.13.9
 Git commit:
 Built:             Mon Apr 13 02:01:03 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine – Community
 Engine:
  Version:          19.03.8
```

```
 API version:      1.40 (minimum version 1.12)
 Go version:       go1.12.17
 Git commit:       afacb8b
 Built:            Wed Mar 11 01:29:16 2020
 OS/Arch:          linux/amd64
 Experimental:     true
containerd:
 Version:          v1.2.13
 GitCommit:        7ad184331fa3e55e52b890ea95e65ba581ae3429
runc:
 Version:          1.0.0-rc10
 GitCommit:        dc9208a3303feef5b3839f4323d9beb36df0a9dd
docker-init:
 Version:          0.18.0
 GitCommit:        fec3683
```

We haven't made any major changes, but as you can see, modifying the code is simple. Once you know how to compile the binary Docker package, all that you need is some more familiarity with the Go programming language and you should be able to start working on minor bug fixes or making changes to the source code.

ACTIVITY 16.02: DEPLOYING THE PANORAMIC TREKKING APP AS A DOCKER APP

Solution:

There are a number of ways in which you can perform the second activity of this chapter. The following steps show one way to do this:

1. Make sure the **`docker app`** command is available on your system. You enabled this feature earlier in the chapter, but to verify, enter the **`docker app version`** command:

```
docker app version
```

   You should see an output similar to the following:

```
Version:               v0.8.0
Git commit:            7eea32b7
Built:                 Wed Nov 13 07:30:49 2019
OS/Arch:               linux/amd64
Experimental:          off
Renderers:             none
Invocation Base Image: docker/cnab-app-base:v0.8.0
```

2. Run the **`docker node ls`** command to verify that Swarm is enabled and that you are working on the leader in the cluster:

```
docker node ls
```

   The command should return output like the following:

```
ID        HOSTNAME         STATUS     AVAILABILITY     MANAGER
    ENGINE
j2qx…     docker-desktop   Ready      Active           Leader
    19.03.8
```

3. Clone the repository for the Panoramic Trekking App into the system you are now working on in order to have access to the code we want to deploy in our Docker app:

```
git clone https://github.com/vincesesto/trekking_app.git
```

   You will get output like the following:

```
Cloning into 'trekking_app'...
remote: Enumerating objects: 170, done.
remote: Counting objects: 100% (170/170), done.
remote: Compressing objects: 100% (124/124), done.
remote: Total 170 (delta 76), reused 134 (delta 42), pack-reused 0
Receiving objects: 100% (170/170), 11.42 MiB | 1.03 MiB/s, done.
Resolving deltas: 100% (76/76), done.
```

4. Change to the **`trekking-app`** directory where our code has just been downloaded from:

```
cd trekking-app
```

5. You will have been working with this project for a while now, so you should have the Docker images available on your system. If not, build the web service image now:

```
docker build -t activity2_web:latest .
```

6. The Docker app will rely on both the **`web`** and **`nginx`** services. Build the **`nginx`** images now with the following command:

```
docker build -t activity2_nginx: latest nginx/.
```

7. Now that you know that all the prerequisites are available, initialize your new Docker app. Run the following command to create our new app, which will be named **panoramic-trekking-app**, and create the app as a single **dockerapp** file:

```
docker app init --single-file panoramic-trekking-app


Created "panoramic-trekking-app.dockerapp"
```

8. Open the **panoramic-trekking-app.dockerapp** file with your text editor. The metadata is not the most important thing to consider, but add details to the **metadata** section to make sure if other people are using your Docker App, they will be able to contact you if they need support or something similar:

panoramic-trekking-app.dockerapp

```
1 # This section contains your application metadata.
2 # Version of the application
3 version: 0.1.0
4 # Name of the application
5 name: panoramic-trekking-app
6 # A short description of the application
7 description: An app to store all your trekking photos and images
8 # List of application maintainers with name and email for each
9 maintainers:
10   - name: vincesesto
11     email: vincesesto@email.com
```

You can find the complete code here https://packt.live/3hB84Dj.

9. Move down to the **docker-compose** section of the **dockerapp** file. This should all be familiar to you by now as we have added the **web**, **db**, and **nginx** services to this section of the **dockerapp** file. Some important sections to note are the fact that we have added environment variables from *line 24* of the web service. This will allow us to specify default variables in our **dockerapp** file and we can then change those variables if needed for other environments. *Line 49* is also setting the port that the **nginx** service is exposed on as a variable:

panoramic-trekking-app.dockerapp

```
15 version: "3.6"
16 services:
17   web:
18     image: activity2_web:latest
19     command: gunicorn panoramic_trekking_app.wsgi:application --bind
         0.0.0.0:8000
20     volumes:
21       - static_volume:/service/static
22     expose:
23       - 8000
24     environment:
25       - SQL_ENGINE=${web.SQL_ENGINE}
26       - SQL_DATABASE=${web.SQL_DATABASE}
27       - SQL_USER=${web.SQL_USER}
28       - SQL_PASSWORD=${web.SQL_PASSWORD}
```

You can find the complete code here https://packt.live/3mrg3Gy.

10. Move down to the bottom of the **dockerapp** file and add in the variables you need for your services to work. As you can see, you have two sections—one for **nginx** and the other for environment variables for the web service:

panoramic-trekking-app.dockerapp

```
59 nginx:
60   port: 8000
61 web:
62   SQL_ENGINE: django.db.backends.postgresql
63   SQL_DATABASE: pta_database
64   SQL_USER: pta_user
65   SQL_PASSWORD: pta_password
66   SQL_HOST: db
67   SQL_PORT: 5432
68   PGPASSWORD: docker
```

You can find the complete code here https://packt.live/2FCkcqC.

11. Once all the changes have been made to your **dockerapp** file, save the file that is ready to progress.

12. Use the **`docker app inspect`** command to make sure the code we have placed in our **`dockerapp`** file is valid and ready to be deployed to our environment:

```
docker app inspect
```

The output of the following command has been reduced for clarity:

```
panoramic-trekking-app 0.1.0

Maintained by: vincesesto vincesesto@email.com

An app to store all your trekking photos and images

Services (3) Replicas Ports Image
------------ -------- ----- -----
db           1        5432  postgres
nginx        1        8000  activity2_nginx:latest
web          1              activity2_web:latest
...
```

13. Use the **`validate`** command to add an extra check to make sure there are no issues with our Docker app:

```
docker app validate panoramic-trekking-app.dockerapp

Validated "panoramic-trekking-app.dockerapp"
```

14. Install the Docker app. Run the **`docker app install`** command as we have in the following code block, specifying the name of our **`dockerapp`** file, and use the **`--name`** option to specify the name of the stack we are creating as well:

```
docker app install panoramic-trekking-app.dockerapp --name panoramic-
trekking-app
```

The command will return the output like the following:

```
Creating network panoramic-trekking-app_default
Creating service panoramic-trekking-app_db
Creating service panoramic-trekking-app_nginx
Creating service panoramic-trekking-app_web
Application "panoramic-trekking-app" installed on context
"default"
```

15. Verify that the Docker app installation was successful by using the **`status`** command. Run the following command:

```
docker app status panoramic-trekking-app
```

We have reduced the output, but you should be seeing all the services of your app starting up and running:

```
INSTALLATION
------------
Name:         panoramic-trekking-app
Created:      5 minutes
Modified:     4 minutes
Revision:     01E5VN2SVWMRWGZ3JRJRDCZN1V
Last Action:  install
Result:       SUCCESS
Orchestrator: swarm
...
```

The orchestration our Docker app is using is Swarm, so you can also use Swarm commands to verify that the installation of our app was successful.

16. Run the **`docker stack ls`** command, as shown in the following code block, to verify that you have three services running as expected for the Panoramic Trekking App:

```
docker stack ls

NAME                        SERVICES        ORCHESTRATOR
panoramic-trekking-app      3               Swarm
```

17. Test the new app running on your system. Open a web browser and enter the URL for the admin interface of the Panoramic Trekking App. Enter the **`http://0.0.0.0:8000/admin`** URL and you should be able to log in to the admin console with the default user of admin and a default password of **`changeme`**. You should then be presented with a screen similar to the following:

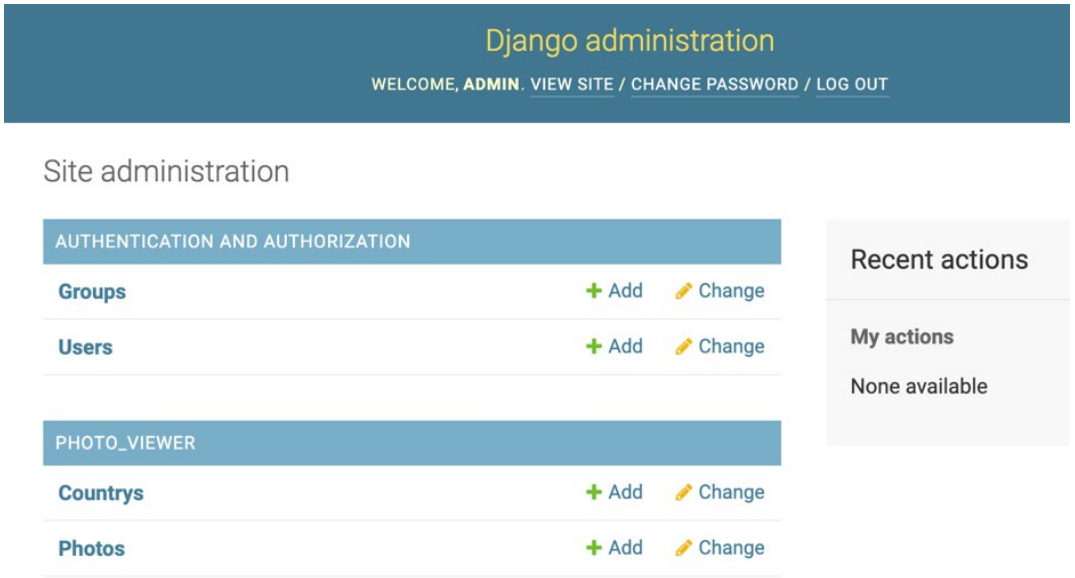

Figure 16.9: The nginx service from Docker Swarm

18. Feel free to take the time to test the app and verify that you can add photos and images and then view the images from the photo viewer, ready for deployment to production:

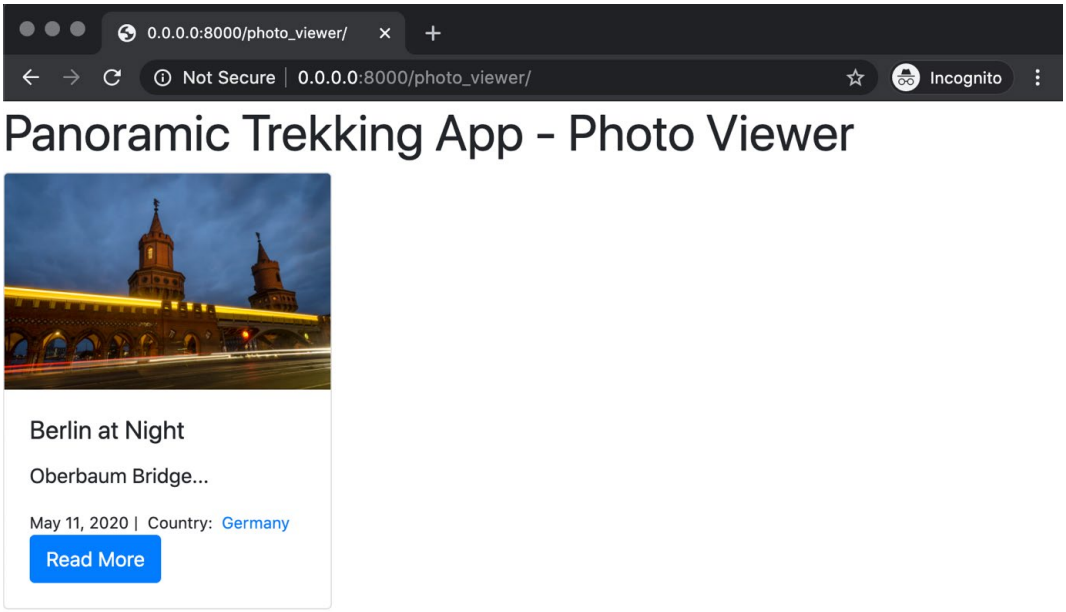

Figure 16.10: Viewing images in the photo viewer

You will be able to view the images easily in the photo viewer from the app, which proves that the Panoramic Trekking App is successfully deployed as a Docker App.

This activity has further demonstrated the functionality of the Docker App, and hopefully, it will become a permanent part of Docker. We have used it to deploy services for the Panoramic Trekking App and this should give you clarification in terms of how developers will work with the applications.

This brings us to the end of the activities and the end of this chapter. The activities should have helped to solidify the knowledge learned earlier on and provide you with experience with some of the newer features of Docker.