**11 – Dependencies, Build Tools and Containerization**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

**Name:**

Top of FormThis week’s topic focused on the issues and techniques around project builds and deployment. In particular, we looked at dependencies and build processes and the opportunities and challenges that they present. We saw how dependency management tools automated build processes (via build tools). We then saw how containerization (via docker) can help to address the challenges, making it easier to more reliably deploy applications and developer environments. Finally, we saw conceptually how docker images are built and how both dependency management and build tools play a role.

These activities will focus on helping you become comfortable using docker. You will learn how to use an existing docker image to deploy an application. You will compare that to the process of manually running the same application locally. Then you’ll see how that process can be adapted to create the docker image that you used in the first place.

**Installing Docker:**

Before you can work with docker images and containers you will need to install docker onto your virtual machine.

1. Use apt to install the docker.io package. Note: Installing the docker package (i.e. without the .io) will not work. Also note, you may need to run sudo apt update first. Give the full command that you used.

2. Run the command docker version. This command displays information both about the docker client (CLI) and the docker server (the Container Daemon) that is installed.

a. What version of docker is installed?

b. What error message is reported at the end of the output?

3. The error reported indicates that your user does not have permission to connect to the “Docker daemon socket.” Let’s investigate why that is and fix the issue.

a. As indicated in the error message; the Docker daemon socket is represented by a file at ﻿/var/run/docker.sock. Give the output generated by the command:

ls -l /﻿var/run/docker.sock

b. From part a, you can see that that file is owned by the root user and also belongs to the group docker.

i. Your user is neither root nor in the docker group. What permissions (read, write, execute) does your user have to this file? You may need to review how the file permissions are represented in the ls -l output if you do not recall.

ii. What permissions do members of the docker group have to this file?

c. While you could use sudo to run docker, that would be a bad idea as it gives super user privileges to docker. A preferable solution here is to add your user to the docker group, which will allow you to access the necessary file (see b.ii). The command to add a user to a group is: adduser <user name> <group name>. Add your user to the docker group. What command did you use?

d. You will have to log out and log back in for your command in part c to take effect. Log out and back in and then use the command: groups to confirm that your user is now in the docker group. What output does the groups command give? If the docker group is not listed return to part c and try again. You will not be able to complete the activities if your user is not in the docker group.

e. Try the docker version command again now. The error should be gone and information about the docker server should now appear following the information about the client. Just to show it is working, what version of the containerd daemon is running? If you still see the permission denied error, return to part c and try again. You will not be able to complete the activities until this works.

**Using a Docker Image:**

To get a feel for how easy containerization makes it to deploy a working application you’ll start by working with an existing container for the Alpha chat bot. We won’t be doing much with Alpha except using it as an example for working with containers. But if you are curious, you can check out the Alpha repo (<https://github.com/IcaliaLabs/alpha)> to learn more about it works and how you can use it to create your own custom web-based chat bots.

4. Increasingly when you want to use an existing product or application you can find a pre-made docker image for it on DockerHub (<https://hub.docker.com/>). DockerHub is an image repository where developers store docker images for others to download and use.

a. Visit DockerHub and search for the image braughtg/alpha. Open the page for that image. What is the “Docker Pull Command” listed there?

b. Run the command that you found in part a in your virtual box. Give the output that is generated.

c. As the output from Part b indicates, the command pulled (i.e. downloaded) the image from DockerHub to your local machine. You can now use that image with the following command:

docker run -it --rm braughtg/alpha

In docker terminology the run command uses the specified image to create and start a container. When this command completes the container will be running and you will see some output about the “Access URLs.” What is the URL labeled LAN?

5. Open a browser on your virtual machine and visit the LAN URL you identified in question #5. The Alpha Chat Bot demo should load in your browser. You can interact with it to get a feel for what it does. If you do not see the chat bot, revisit question 4 to get it working.

a. How does the output from question 5 tell you to stop the server?

b. What happens the first time you type that key combination? What happens if you type it a second time?

c. What happens now if you reload the LAN URL in the browser? Why?

**Running Alpha Locally:**

Hopefully the prior exercises gave you a little insight into how easy containers can make it to deploy an application. As a comparison point, the activities in this section will have you build and run the Alpha chat bot locally. To do so, you’ll install the prerequisites, get the code, install the dependencies and perform the build process.

6. First let’s get the code for the Alpha project. Like with the previous activity using FarmData2, we’ll use a fork of the Alpha project that is just for our class as the upstream. That way our activities will not disrupt the real upstream.

a. Fork our class repository for Alpha (<https://github.com/dickinson-comp190/alpha>) to your GitHub account. Give the URL of your fork.

b. Clone your Alpha repository from GitHub to your local machine. Give the command that you used.

c. Set the upstream for your local repo to the repo that you forked from the comp190 organization. Give the output of a git remote -v command showing that you have successfully set the upstream.

7. The Alpha project use Node.js to run JavaScript code and uses npm as both its dependency manager and its build automation tool. Thus, these are prerequisite dependencies that will need to be installed in order to build and run Alpha locally. Install the following packages to satisfy these prerequisites:

* nodejs
* npm

Give the commands that you used.

8. The next step is to install the dependencies for Alpha. Alpha uses the npm (Node Package Manager) to manage its dependencies. If you are curious, you can look at the package.json file in the alpha directory to see what they are – there are lots – glad we don’t have to do it manually. Change into the alpha directory and install the dependencies using the command npm install. You’ll see some warnings near the end, those can safely be ignored. How many packages did npm install? (Told you it was a lot!)

9. Now that all of the dependencies have been installed, the next step is to run the build process. Alpha also uses npm as its build automation tool. From inside the alpha directory, run the command npm run build to start the build process. If all goes well, you should see a message about “723 hidden modules” as the last line of the output. There is no answer required here, but if you do not see that message, you will need to revisit questions 6-9. If rerunning some of the commands does not work, you should delete your alpha directory and all of its contents and restart from question 7.

10. Now that the program has been built you are ready to run it. From inside the alpha directory use the command npm start to run the Alpha server. If all goes well you will see a message about the “Access URLs.” One of those will be the same LAN URL we used when running in the docker container. There is no answer needed here, but point your browser to that URL again and interact with the chat bot demo to be sure it is working. If not, you’ll need to revisit questions 6-10.

**Vocabulary:**



One of the biggest challenges faced in getting going with docker is understanding the terminology. The diagram above, from the class slides, is a good metaphor for organizing the terminology. The terminology will be used a lot in the following exercises, so this question will help you determine if you have a grip on it before going on. You can also review the more complete definitions of the terms in the class slides if necessary.

11. Fill in the column next to each of the following statements with one of the following terms: Image Repository, Image, Tag, or Container, whichever applies best. If you choose Container, also indicate if the container is started or stopped, if you can tell.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Statement** | **Term** |  |
|  | A containerized application that is currently useable on-line |  |  |
|  | Version 2.1 or latest following an image name |  |  |
|  | A collection of docker images that can be pulled and used |  |  |
|  | The instructions and data for building a container |  |  |
|  | A containerized dev environment with saved changes but that is not currently in use |  |  |
|  | An instance of an image |  |  |
|  |  |  |  |

**Making a First Docker Image:**

Now that you’ve seen how easy it was to launch Alpha using Docker and how much was involved in running it locally, it should be clear that there are some big advantages to using docker to deploy an application. At least once an image exits. The exercises in this section will give you some insight into how docker images are created and introduce you to some of the more useful docker commands along the way.

12. Docker images are created from instructions contained in a file named Dockerfile. So to create an image you will need to create a Dockerfile.

a. First, because you are working in a git repository, and you shouldn’t change the main branch, create a feature branch and switch to it before continuing. What commands did you use?

b. Using your favorite editor, create a file named Dockerfile inside the alpha directory and put the following content into the file:

﻿FROM ubuntu:20.04

RUN adduser --disabled-password --gecos '' alpha

WORKDIR /home/alpha

USER alpha

CMD ["/bin/bash"]

The CAPITAL letter words in the above code are *Dockerfile statements* and control the way that the image is being built. The statements in the above Dockerfile will build a container based on the ubuntu 20.04 version of linux, add a new user named alpha to the system, make alpha’s home directory the working directory, make alpha the active user and start the bash shell.

A more detailed description of each line of the script follows:

* The FROM statement specifies the base image from which to start building this image. By saying FROM ubuntu:20.04, we are telling docker to use the ubuntu:20.04 image from DockerHub as the starting point for our image. The statements in the rest of the Dockerfile make changes to that image to customize it for our purpose.
* The RUN statement can execute any Unix command and is used to configure the OS in the container. In the Dockerfile above it creates a user named alpha that does not require a password. In later exercises, you will use it to install prerequisites and dependencies and to run the build process.
* The WORKDIR command specifies the working directory in which subsequent Unix commands will be executed. It is like using cd to change into a directory before typing commands. In the Dockerfile above it sets the working directory to the home directory of the new alpha user.
* The USER command switches the current user from root, which is the default, to the specified user. It is like executing an su command to change users. All of the statements following the USER statement will be executed as that user instead of as root. In the Dockerfile above, it makes alpha the active user.
* The CMD statement indicates the command to be run when a container built from the image is started. In the above Dockerfile, the command specifies that the bash shell is to be run. So, when this container is started you will see a command prompt, just like when you open a terminal. Except that this bash shell will be running on the ubuntu:20.04 OS inside the container.

13. The docker build command reads the contents of the Dockerfile and uses it to generate a docker image. Run the following command in the alpha directory:

docker build .

This command will build an image based on the contents of the Dockerfile in the current directory (.). Give the output that is generated.

14. The docker images command will display information about all of the images that you have available locally on your machine.

a. What are the repository names of all of the images that you now have?

b. Which image do you think is the one that was built by your docker build command from question 13.

c. Look at the output you gave in question 13. What there confirms your answer to part b?

15. Having an image named <none> seems like a bad idea. There are a number of ways to fix this, we’ll use one here. You can change the name and tag of an image with the command:

docker tag <IMAGE ID> <repo name>:<tag>

Use this command to name your image with your username and the tag 1.0. What command did you use? Use the docker images command to confirm that you have successfully tagged your image.

If the IMAGE ID reminds you a little of the ID’s assigned to commits in git, well done. That’s because, docker’s image IDs (and also as we’ll see, its container ID’s) are generated using the Secure Hash Algorithm (SHA), just like git’s commit ID’s. It is not required reading, but, if you are curious, Casey Crane gives a nice short introduction on the *hashedout* blog:

* <https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide>

16. Optional: The docker build command will allow you to combine the process of creating an image and tagging it. Do some research on docker build and give a single command that will build the image in question 13 and tag it as was done in question 15.

**Creating a Container from an Image:**

17. Now that you have create an image, the next step is to use it to create a container.

a. First let’s check if you have any containers already. The docker ps -a command provides information about all of the containers on your system (stopped or running). Use the docker ps -a command. Are there currently any containers on your system? If so, which ones?

b. The docker create command is used to turn an image into a container. Customize the following docker create command to make a container named first with the tag latest from the image you created:

docker create -it --name <container name> <image name>:<tag>

What command did you use?

Note: Most docker commands that include a <tag> will default to the tag latest if the <tag> is omitted.

c. Use the docker ps -a command again. You should see your new container. If not revisit part b and try again.

i. What is the CONTAINER ID of the new container that you created?

ii. Does the container have the same ID (i.e. SHA) as the image from which it was created?

iii. What is the STATUS of the container that was created?

**Starting and Stopping Containers:**

18. Now that you have a container, you need to get it running.

a. First, let’s check if you have any running containers already. The docker ps command (without the -a) shows information about all of the containers that are currently running on your system. Use the docker ps command. Are there any containers currently running on your system? If so which ones?

b. The command: docker start -i <container name> will start up the named container. Give a command that will start the container that you created in question 16.

c. Open another terminal in your virtual box and use the docker ps command again. You should now see your container running. What is its STATUS?

19. This question will have you do a few things that will help to illustrate that the container really is acting like a distinct machine running a separate copy of the linux operating system.

a. Run the command whoami at the bash prompt in the container. This will be the prompt with the SHA of the container. What is the output? Why?

b. The command: ﻿cat /etc/lsb-release | grep "DESCRIPTION" will print out the flavor of linux that is running. What output does this command give when you run it at the bash prompt in the container?

c. What if you run it at the bash prompt on your virtual box?

d. Briefly explain the difference between the outputs in parts b and c.

20. The docker stop <container name> command can be used to stop a running container. Using the terminal connected to your virtual machine (i.e. not the one in the container), stop your running container. What command did you use? Use the docker ps command to confirm that your container is stopped.

**Deleting Containers:**

When a container is stopped it is not deleted. You can use the docker ps -a command to see that your container still exists. You can restart it using docker start as before. If you restart a container, any changes you had made would have been preserved in the writeable layer. If you want, you can test this. Use nano to create a new file in the container, stop it and restart it. The file will still be there.

21. When a container is no longer needed it can be deleted. The command

docker rm <container name> will delete the named container.

a. Give a command that will delete the container that you created.

b. What docker command can you use to check that the container has been deleted? Use your command to confirm that you have deleted the container. If it has not been deleted, revisit part a.

22. Optional: Often when you see instructions using docker they will not use the create and start commands separately. Instead, they will use the docker run command that we saw earlier. This command will both create the container and start it. Do some research on the docker run command. Give a command that will use docker run to create your container and start it. Make your command so that also automatically deletes the container when it exits. Be sure to test your command. You’ll want to use docker ps and docker ps -a to be sure that it works.

23. Optional: When developing a new Dockerfile it is not unusual to end up with lots of containers that you do not need due to failed attempts. Do some research and find a simple concise command that will delete all stopped containers.

**Deleting Images:**

Even when all of the containers created from an image have been removed the image will still exit. If desired, you can use the docker create command to make a another new container from the image. You can even create multiple containers from the same image. However, sometimes you will no longer have a use for an image. Images take up disk space, so you will want to be sure to delete images that you do not need.

24. The command: docker image rm <image repo name> will delete the named image.

a. Give a command that will delete the image that you created.

b. What docker command can you use to check that the image has been deleted? Use your command to confirm that you have deleted the container. If it has not been deleted, revisit part a.

Note that if any containers exists that were created from the image you will not be able to delete the image. You will first have to delete the containers, and then delete the image.

25. Optional: As with containers, when developing a new Dockerfile it is not unusual to end up with lots of images named <none> due to failed attempts. Do some research and find a simple concise command that will delete all if the <none> images. This should be similar to the command found in question 23.

**Reflection – Docker Commands:**

26. Complete the table below by filling in the right-hand column with the commands that accomplish the task listed on the left. Use the <…> notation appropriately to indicate parameters that need to customized for each use. Note that the tasks listed are in approximately the same order as they have appeared in this activity.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Task to Complete** | **Docker Command** |  |
|  | Copy an image from DockerHub |  |  |
|  | Create a container and start it |  |  |
|  | List all images |  |  |
|  | Create an image from a Dockerfile |  |  |
|  | Tag an image |  |  |
|  | Create container from an image |  |  |
|  | List all containers |  |  |
|  | Start a container |  |  |
|  | List all running containers |  |  |
|  | Stop a container |  |  |
|  | Delete a container |  |  |
|  | Delete an image |  |  |
|  |  |  |  |

**Dockerizing Alpha:**

The activities in this section will have you extend the Dockerfile that you created to build an image for the Alpha chat bot application. If you have spent more than three hours on this assignment already you may skip this section.

27. Add to the Dockerfile to create a container for the Alpha chat bot. To do this you will need to add commands to the file that install the prerequisites, copy the source code into the container, install the dependencies, run the build process and start the server. Here are some hints to help you along:

* All of the following should be done between the WORKDIR statement and the USER statement.
  + Use RUN commands to install the prerequisites using apt.
    - All commands before the USER command will run as root so you will not need sudo.
    - Commands in a Dockerfile cannot require user input. So, you will need to stop apt from prompting for confirmation of installs. You can do this by adding a -y flag (i.e. apt -y install <package>).
  + After the prerequisites are installed, copy all files from the alpha directory into the container by adding the following command to the Dockerfile:
    - COPY . .
  + Use RUN commands to:
    - Install the dependencies.
    - Run the build process.
* Change the CMD statement so that instead of running bash it runs the Alpha server. The format for this statement is:
  + CMD ["npm","start"]

Copy and paste the full contents of your Dockerfile here.

28. Give the sequence of commands that you used to build the image, tag the image, create a container and start the container. Be sure to give your image and container meaningful names (i.e. not first). Also test them to ensure that they work and that you can access the Alpha chat bot once the container is running.

29. Create a pull request to the upstream for the feature branch containing your Dockerfile. Hint: You will need to stage and commit the Dockerfile and then push your feature branch to the origin.

**Optional:** To help us improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.