CS 475 -Project 2 (Docker Deployment)

Due Date: 3/6/2023

Note: This assignment was designed and tested with the help of David Berdik

In this assignment, you will be introduced to the basics of working with Docker images and Docker containers. You will build a Docker image that runs a dictionary server and deploy it as a container.

1. Using a Linux distribution of your choice, install Docker. The installation process will vary depending on the Linux distribution you are using. For example, if you are using Ubuntu, you can install Docker by running the following command:

sudo apt install docker.io

1. Create a new directory and copy the server.tar.gz file included with these instructions to the directory. You will work in this directory to build your Docker image.
2. In your directory, create a new file named Dockerfile and copy the following in to the file. An explanation of each line in the file follows.

FROM ubuntu:latest

ENV PYTHONUNBUFFERED=1

ENV PYTHONIOENCODING=UTF-8

RUN apt-get update \

&& apt-get -y upgrade \

&& apt-get -y install python3

ADD server.tar.gz /

CMD [ "python3", "/server.py", "25565" ]

FROM ubuntu:latest

In most cases, Docker images are built by taking an existing Docker image from a repository of prebuilt images known as the Docker Hub (<https://hub.docker.com/>) and modifying it. For convenience, the Docker Hub offers a set of images known as base images which contain nothing but the core functionality needed to run the Linux distribution you choose to use for your image. The use of the FROM keyword specifies which Docker image we want to inherit from. In this example, we are going to use the latest version of the Ubuntu base image provided on the Docker Hub. **Please note that the Linux distribution you choose to use for your image does not have to be the same distribution you are using to build the image. (Example: You can build and run an Ubuntu-based Docker image using Fedora.)**

ENV PYTHONUNBUFFERED=1

ENV PYTHONIOENCODING=UTF-8

The ENV keyword allows us to specify environment variables that will be set inside the Docker image we are building. In our case, we are setting two environment variables. These variables will instruct Python to not buffer printing to standard output. By default, Python buffers its output, which causes problems with the docker logs command which we will see later. Disabling buffering fixes this issue.

RUN apt-get update \

&& apt-get -y upgrade \

&& apt-get -y install python3

The RUN keyword allows us to specify commands to execute as part of the image building process. Since our dictionary server is written in Python, we need to install Python inside the image. The series of three apt-get commands specified here will do that.

ADD server.tar.gz /

The ADD keyword allows us to copy files in to the image. When the source file specified is a .tar.gz archive, the contents of the archive will be extracted in to the image at the specified destination. In our case, we are instructing Docker to extract the contents of server.tar.gz to the / directory of the image.

CMD [ "python3", "/server.py", "25565" ]

The CMD keyword allows us to specify the command that should be executed when the image is launched as a container. In our case, we are specifying that when the image is launched as a container, the python3 /server.py 25565 command should be executed. This command will launch the dictionary server and bind it to port 25565 inside the container.

1. After you have defined your Dockerfile, execute the following command from the directory in which you created the file.

sudo docker build -t dict-server:latest .

This command will build your image. The time this command takes to execute will vary depending on your internet speed, as it will need to download the Ubuntu base image from the Docker Hub if it is not already present on your system. Additionally, the Python installation inside your image will need to be downloaded each time you build this image.

The -t flag allows us to specify the name we wish to use for the image as well as a version tag for the image. In our case, we are naming the image dict-server and using the latest tag. The latest tag is traditionally used by Docker image distributions to denote the latest version of an image. You may recall seeing this in our use of the FROM ubuntu:latest command in our Dockerfile.

1. After the Docker image build completes, execute the following command.

sudo docker run -d -p 42:25565 dict-server

This command will launch an instance of your Docker image. Deployed Docker images are referred to as Docker containers. The -d flag in the command instructs Docker to start the container in detached mode. When detached mode is used, the container launches in the background and a hash is printed to the terminal. This hash is a unique identifier for referencing the newly deployed container.

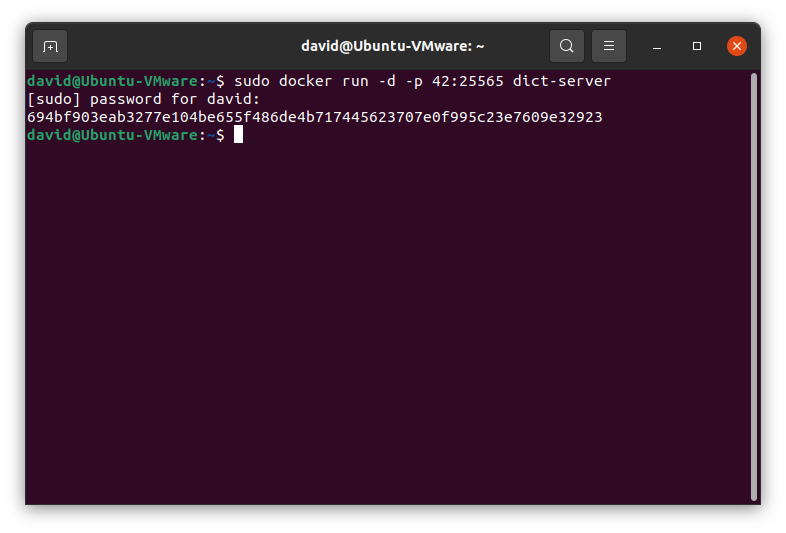


Figure 1: Example of deploying a Docker container.

The -p 42:25565 portion of the command instructs Docker to map port 25565 (the port on which the dictionary server is running inside the container) to port 42 on the host system. This will allow us to access the server running inside the container on port 42, as can be demonstrated by connecting to the server by using the included dictionary client.

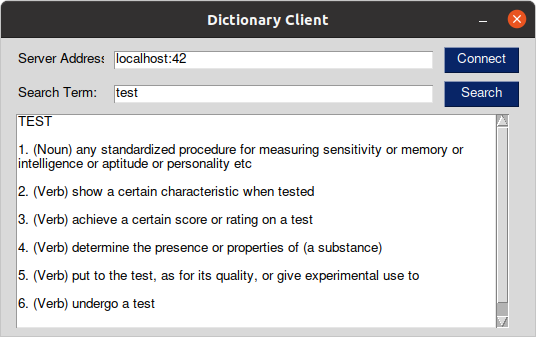


Figure 2: Using the dictionary client to connect to the server running inside the container.

1. When executing the docker run command in the previous step, a hash string was printed to the terminal as shown in Figure 1. This hash is a unique identifier for referencing the container. The hash of the container can be used in an abbreviated form to execute various other commands on the container. You can retrieve a list of all containers on your system by using the following command.

sudo docker ps -a

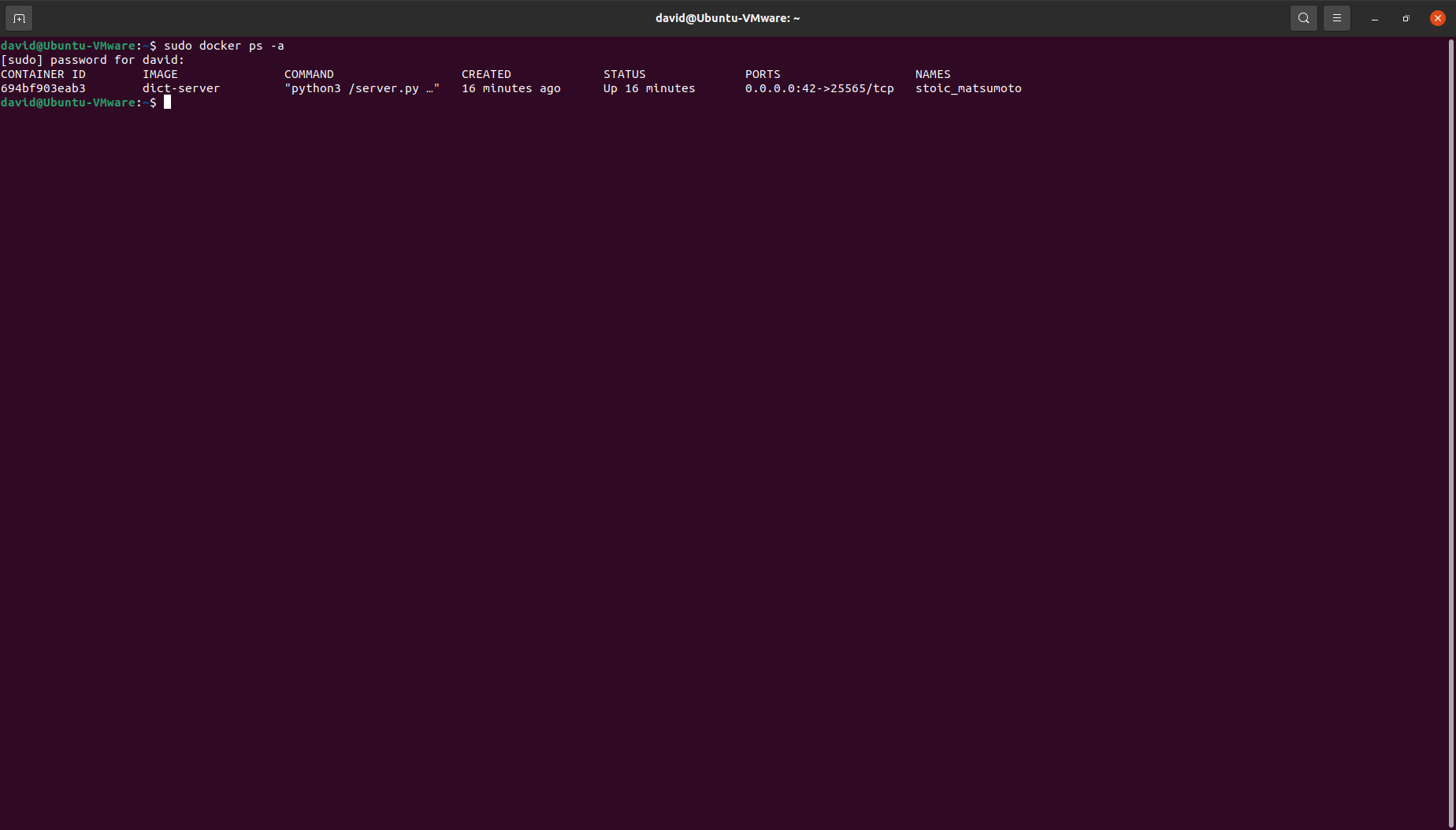


Figure 3: A list of all containers on the system.

The leftmost entry in the list is the container ID. Take note of the fact that this ID is an abbreviated version of the full ID which was printed when the container was launched. (Compare the container ID shown in Figure 3 to the start of the hash which was printed to the terminal in Figure 1.)

1. To show how the container ID can be used to perform various tasks with a container, we will demonstrate the use of several Docker commands. The first command we will use will demonstrate how to check the container’s logs. The logs shown by a container will vary depending on the software running in it. In our case, the command will show the log generated by the dictionary server.

To retrieve the server log, execute the following command.

sudo docker logs <container\_id>

Replace <container\_id> with a portion of the ID of your container. You do not need to use the entire ID and only need to be specific enough to prevent collisions with other containers on the system. Typically, the first three characters of an ID are sufficient. Since I only have one container running on my system, it is sufficient for me to only use the first character.

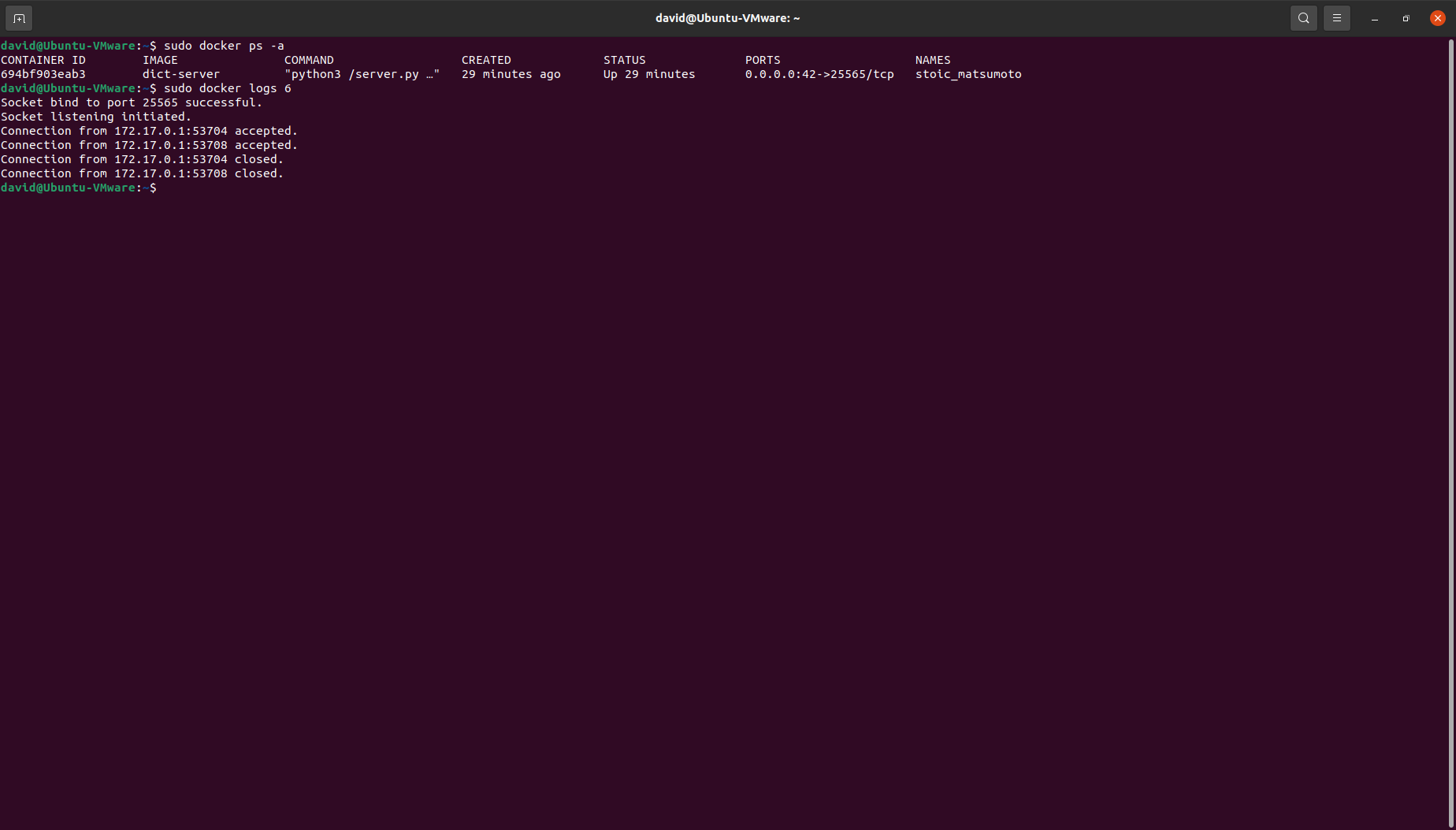


Figure 4: Printing the container’s log to the terminal.

Executing this command will print the log of the server running inside the container to the terminal. Depending on if you have attempted to connect to the server, your log output may be slightly different than what is shown in Figure 4.

1. Docker containers can be easily started and stopped by using the docker start and docker stop commands. Using your container ID in place of <container\_id>, execute the following commands.

sudo docker stop 6

sudo docker ps -a

sudo docker start 6

sudo docker ps -a

These commands will stop and start your container. Please note from the “Status” column in the output of the docker ps -a command that the container state changes between the execution of the stop and start commands.

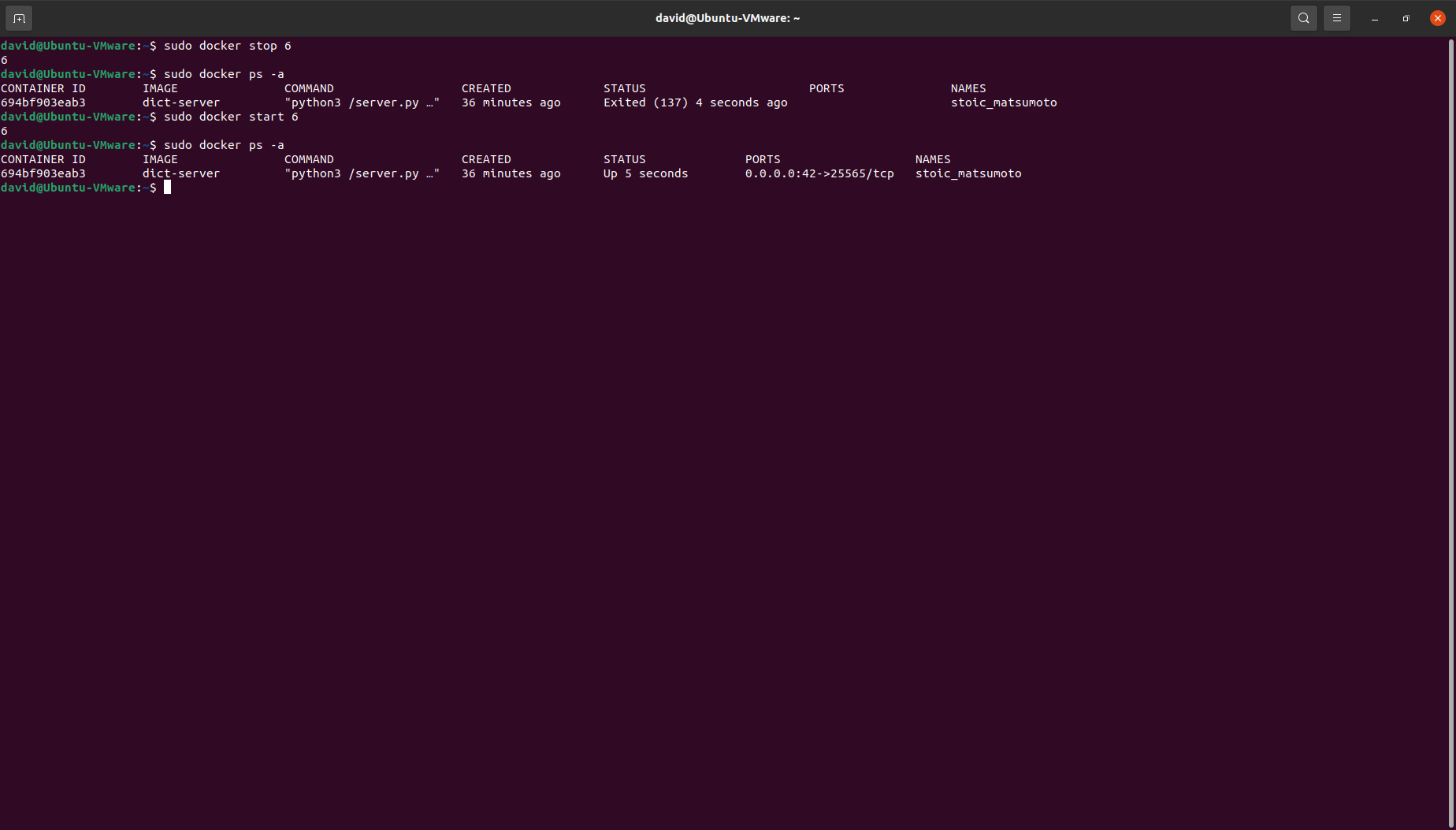


Figure 5: Stopping and starting a Docker container.

1. If necessary, you may access the terminal of a running Docker container as you would on a normal Linux installation. To access the terminal of your container, execute the following command using your container ID in place of <container\_id>.

sudo docker exec -it <container\_id> bash

This will drop you in a bash terminal executing inside of your container. Note that the hostname shown on your terminal has changed to match your container ID as shown in Figure 6. This indicates that you are in fact inside the container.

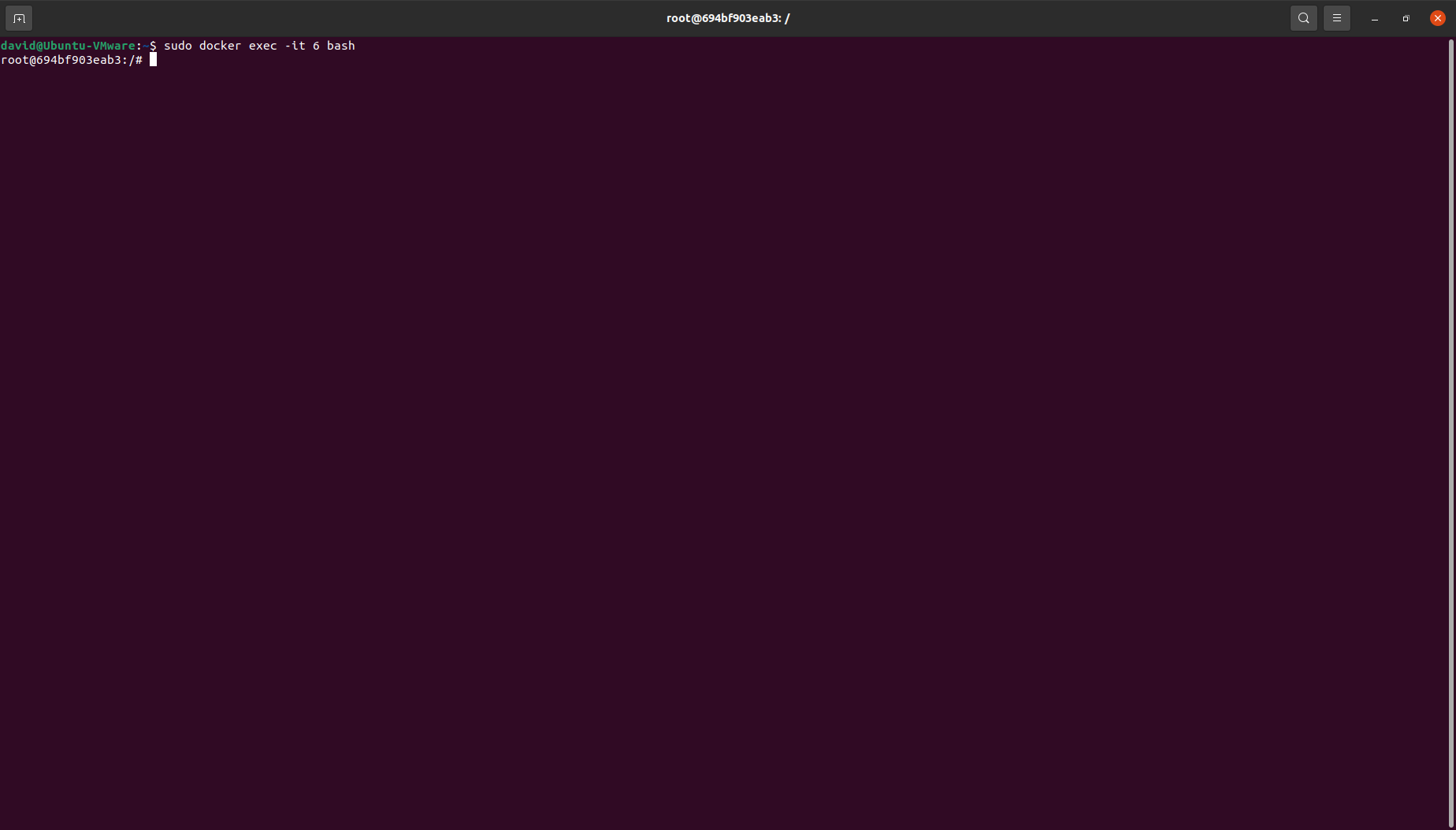


Figure 6: Accessing the bash terminal of a container.

Once inside the container, you may execute commands as you would on a normal Linux system as shown in Figure 7.

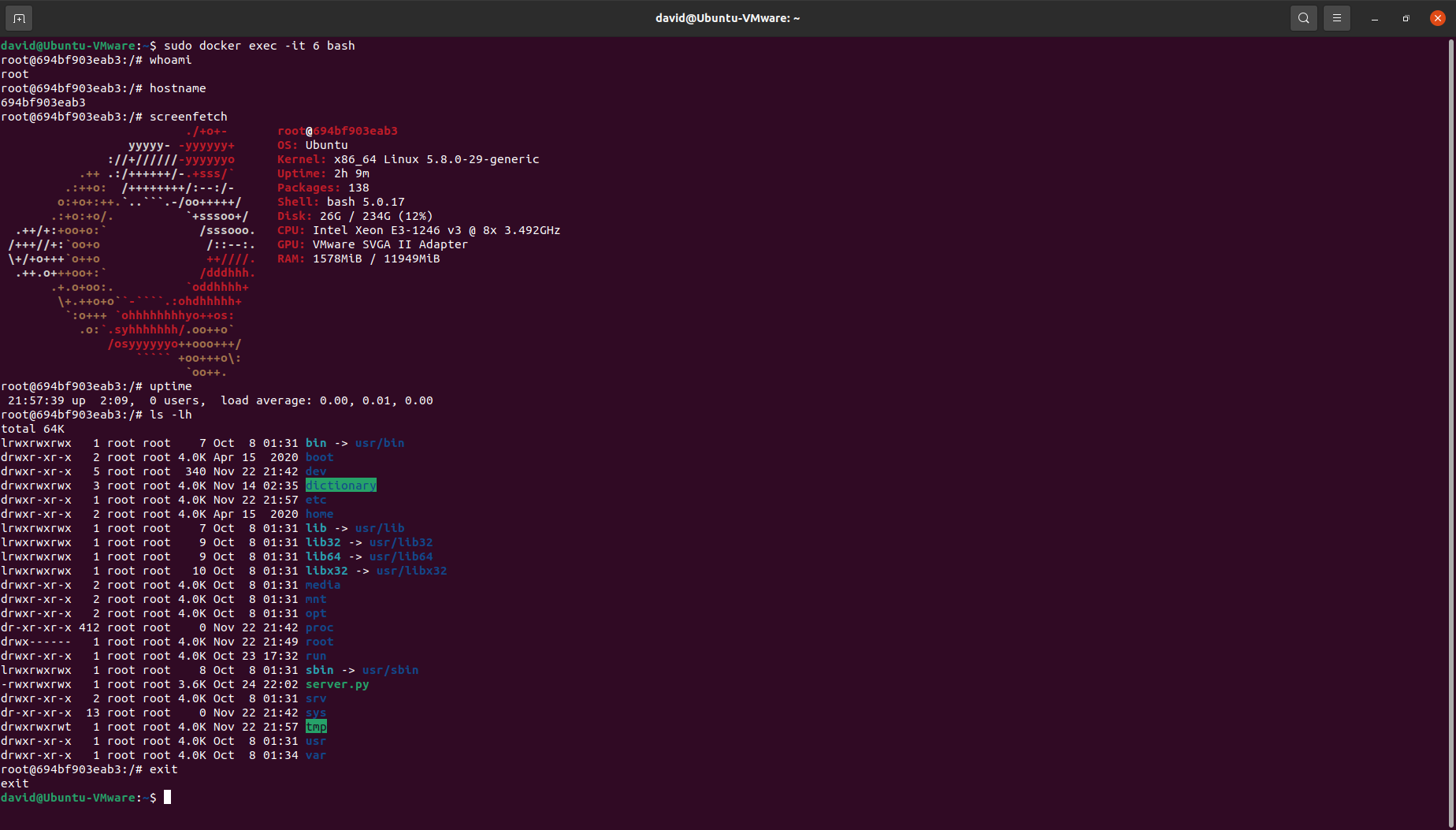


Figure 7: Executing commands inside of a Docker container.

1. After a Docker container is no longer needed, it can easily be removed. To remove your dictionary server container, execute the following commands using your container ID in place of <container\_id>.

sudo docker stop <container\_id>

sudo docker rm <container\_id>

sudo docker ps -a

After executing these commands, you should see that your Docker container has been removed as shown in Figure 8.

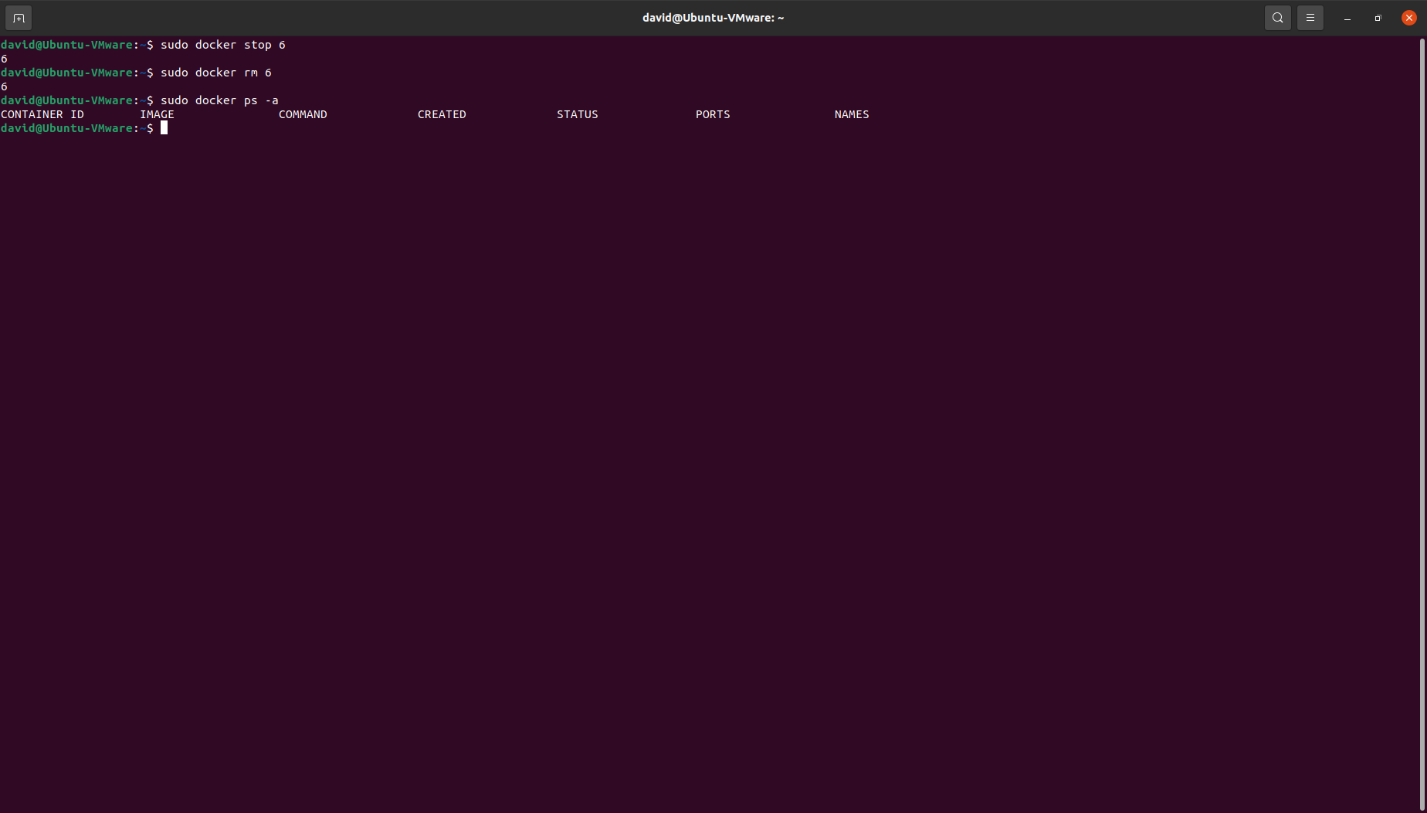


Figure 8: The Docker container has been removed.