Week 1, Web Application Development

Azamat Serek, PhD, Assist.Prof.

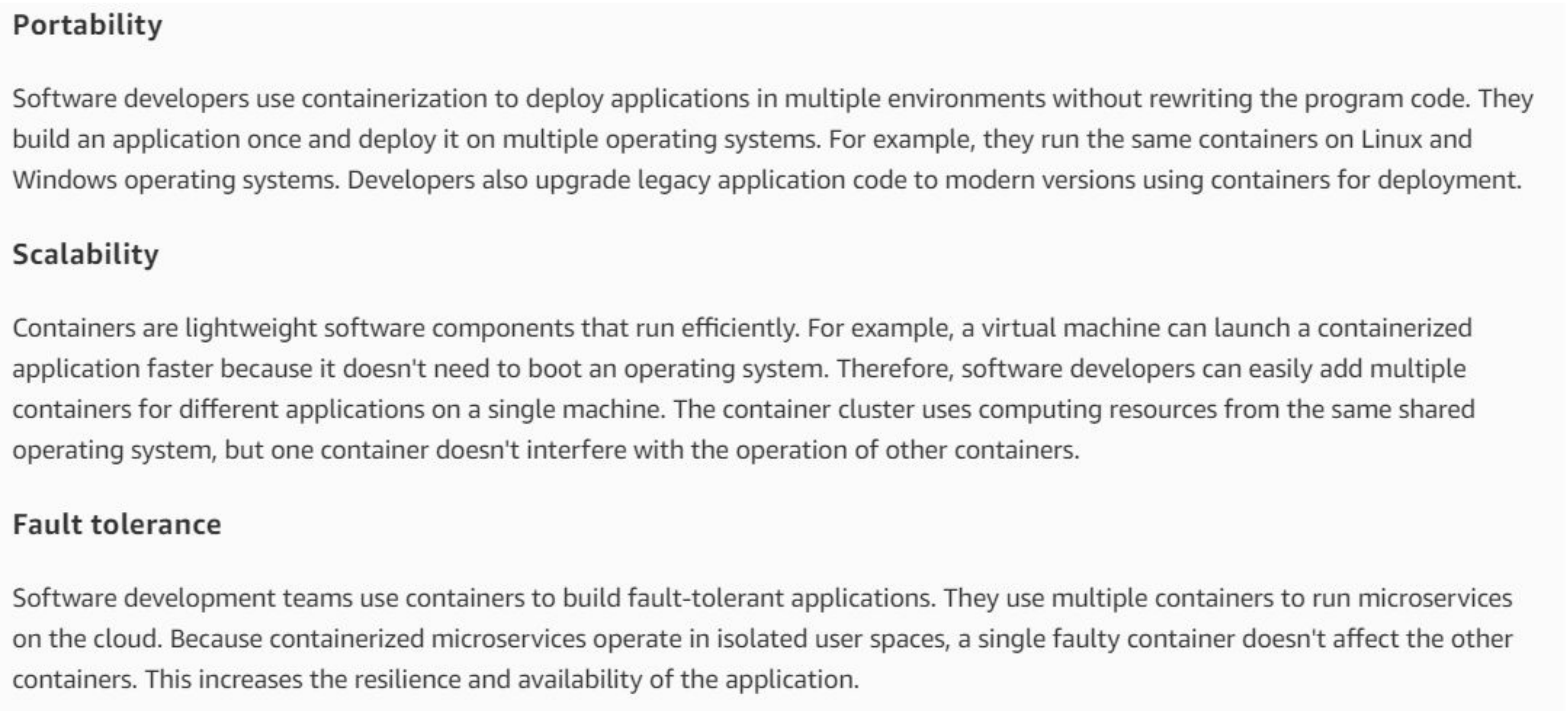
Objectives

1. Overview of containerization
2. Installing Docker
3. Basic Docker commands
4. Creating and running Docker containers

Containerization

Containerization is a software deployment process that bundles an application’s code with all the files and libraries it needs to run on any infrastructure. Traditionally, to run any application on your computer, you had to install the version that matched your machine’s operating system. For example, you needed to install the Windows version of a software package on a Windows machine. However, with containerization, you can create a single software package, or container, that runs on all types of devices and operating systems.

Benefits of containerization



Portability

Software developers use containerization to deploy applications in multiple environments without rewriting the program code. They build an application once and deploy it on multiple operating systems. For example, they run the same containers on Linux and Windows operating systems. Developers also upgrade legacy application code to modern versions using containers for deployment.

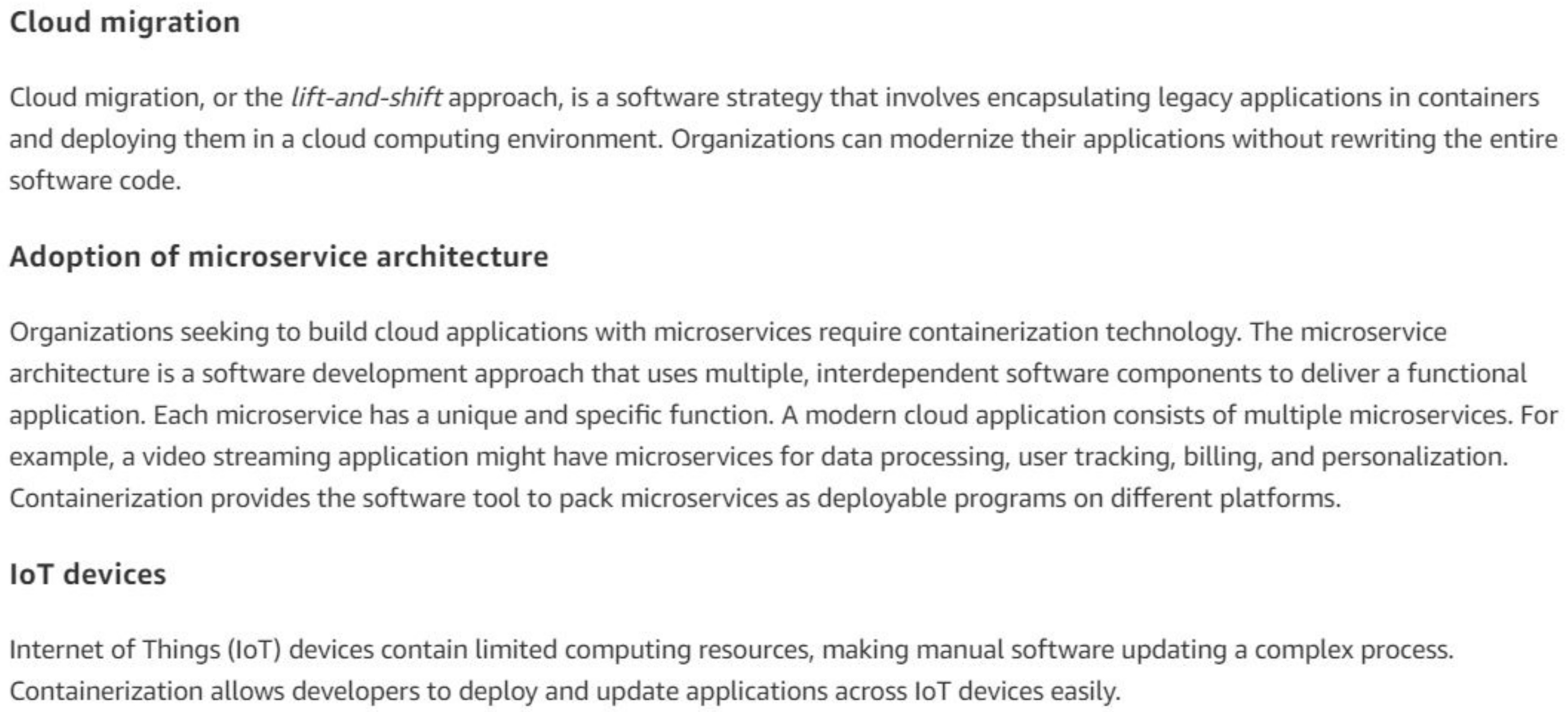
Scalability

Containers are lightweight software components that run efficiently. For example, a virtual machine can launch a containerized application faster because it doesn't need to boot an operating system. Therefore, software developers can easily add multiple containers for different applications on a single machine. The container cluster uses computing resources from the same shared operating system, but one container doesn't interfere with the operation of other containers.

Fault tolerance

Software development teams use containers to build fault-tolerant applications. They use multiple containers to run microservices on the cloud. Because containerized microservices operate in isolated user spaces, a single faulty container doesn't affect the other containers. This increases the resilience and availability of the application.

Use cases



Cloud migration

Cloud migration, or the *lift-and-shift* approach, is a software strategy that involves encapsulating legacy applications in containers and deploying them in a cloud computing environment. Organizations can modernize their applications without rewriting the entire software code.

Adoption of microservice architecture

Organizations seeking to build cloud applications with microservices require containerization technology. The microservice architecture is a software development approach that uses multiple, interdependent software components to deliver a functional application. Each microservice has a unique and specific function. A modern cloud application consists of multiple microservices. For example, a video streaming application might have microservices for data processing, user tracking, billing, and personalization. Containerization provides the software tool to pack microservices as deployable programs on different platforms.

IoT devices

Internet of Things (IoT) devices contain limited computing resources, making manual software updating a complex process. Containerization allows developers to deploy and update applications across IoT devices easily.

How does containerization work

Containerization involves building self-sufficient software packages that perform consistently, regardless of the machines they run on. Software developers create and deploy container images—that is, files that contain the necessary information to run a containerized application. Developers use containerization tools to build container images based on the Open Container Initiative (OCI) image specification. OCI is an open-source group that provides a standardized format for creating container images. Container images are read-only and cannot be altered by the computer system.

Container images are the top layer in a containerized system that consists of the following layers.

Container orchestration

Container orchestration is a software technology that allows the automatic management of containers. This is necessary for modern cloud application development because an application might contain thousands of microservices in their respective containers. The large number of containerized microservices makes it impossible for software developers to manage them manually.

Benefits of container orchestration

Developers use container orchestration tools to automatically start, stop, and manage containers. Container orchestrators allow developers to scale cloud applications precisely and avoid human errors. For example, you can verify that containers are deployed with adequate resources from the host platform.

Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

What can I use Docker for?

Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

- Your developers write code locally and share their work with their colleagues using Docker containers.
- They use Docker to push their applications into a test environment and run automated and manual tests.
- When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
- When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Responsive deployment and scaling

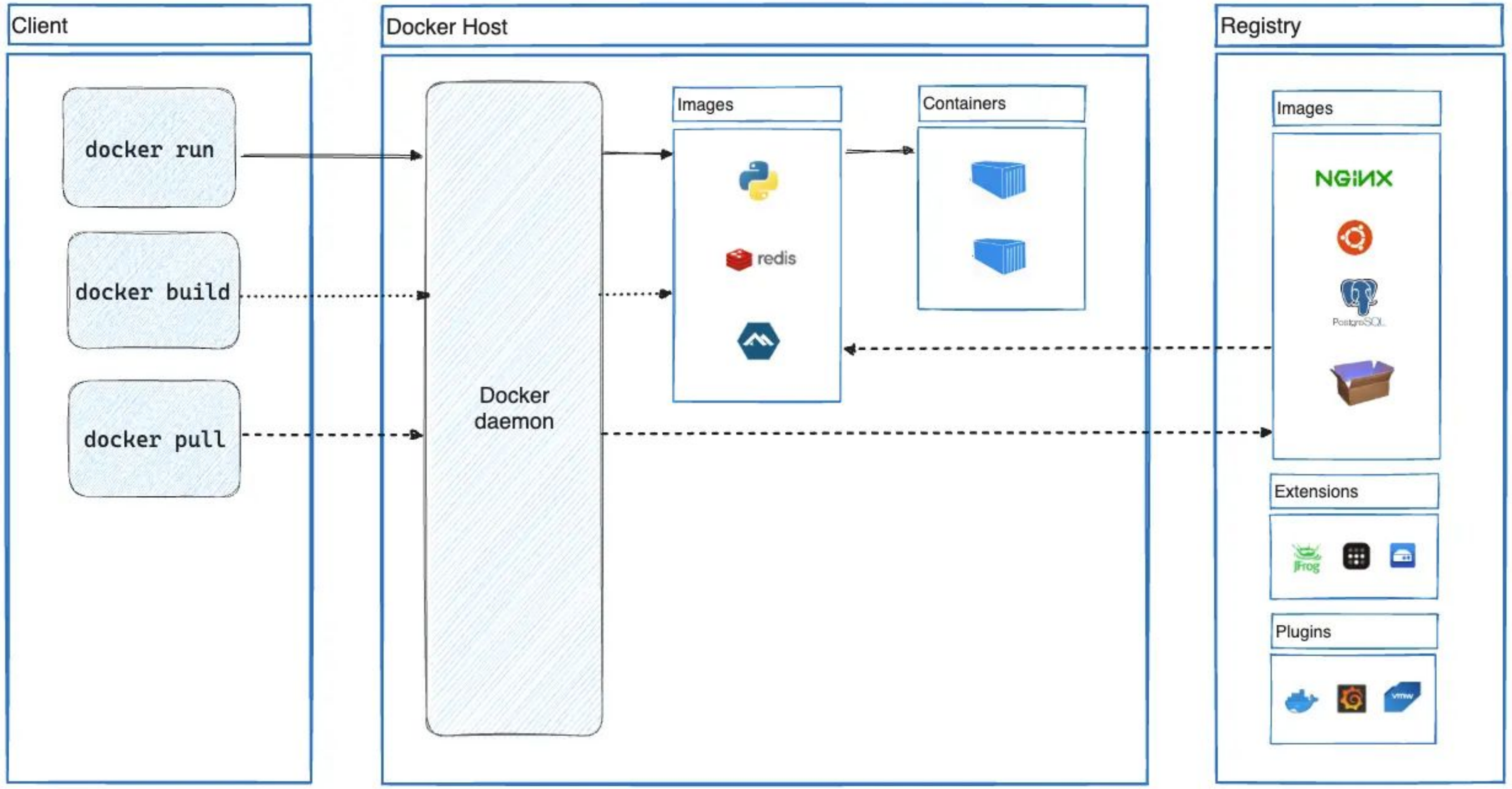
Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

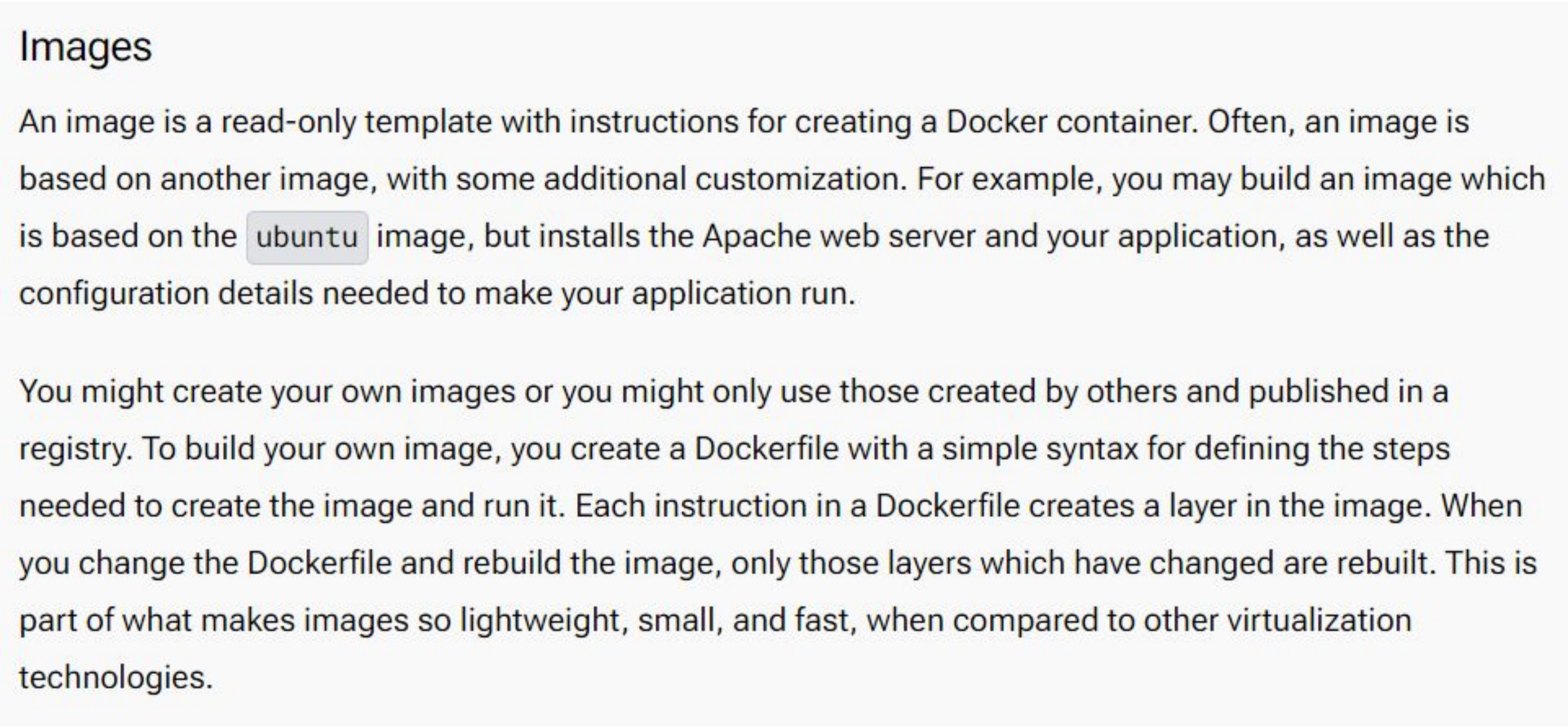
Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your server capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

Docker architecture





Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the `ubuntu` image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that aren't stored in persistent storage disappear.

Example `docker run` command

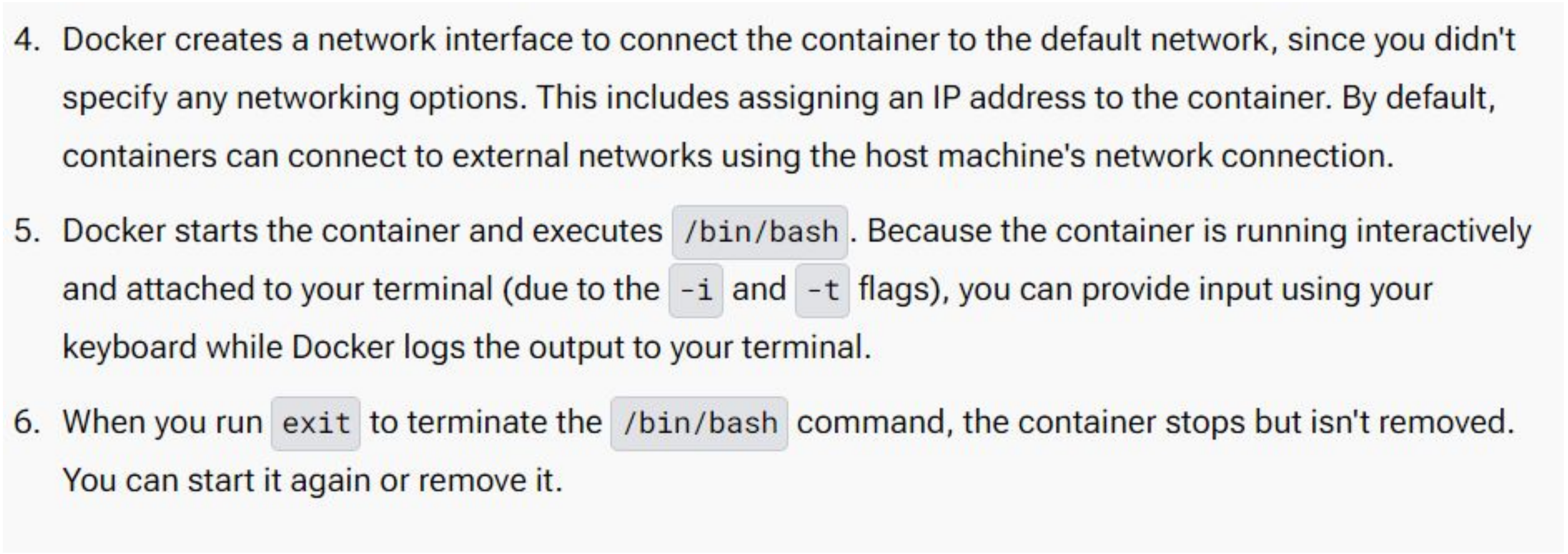
The following command runs an `ubuntu` container, attaches interactively to your local command-line session, and runs `/bin/bash`.

```
$ docker run -i -t ubuntu /bin/bash
```

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you don't have the `ubuntu` image locally, Docker pulls it from your configured registry, as though you had run `docker pull ubuntu` manually.
2. Docker creates a new container, as though you had run a `docker container create` command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.

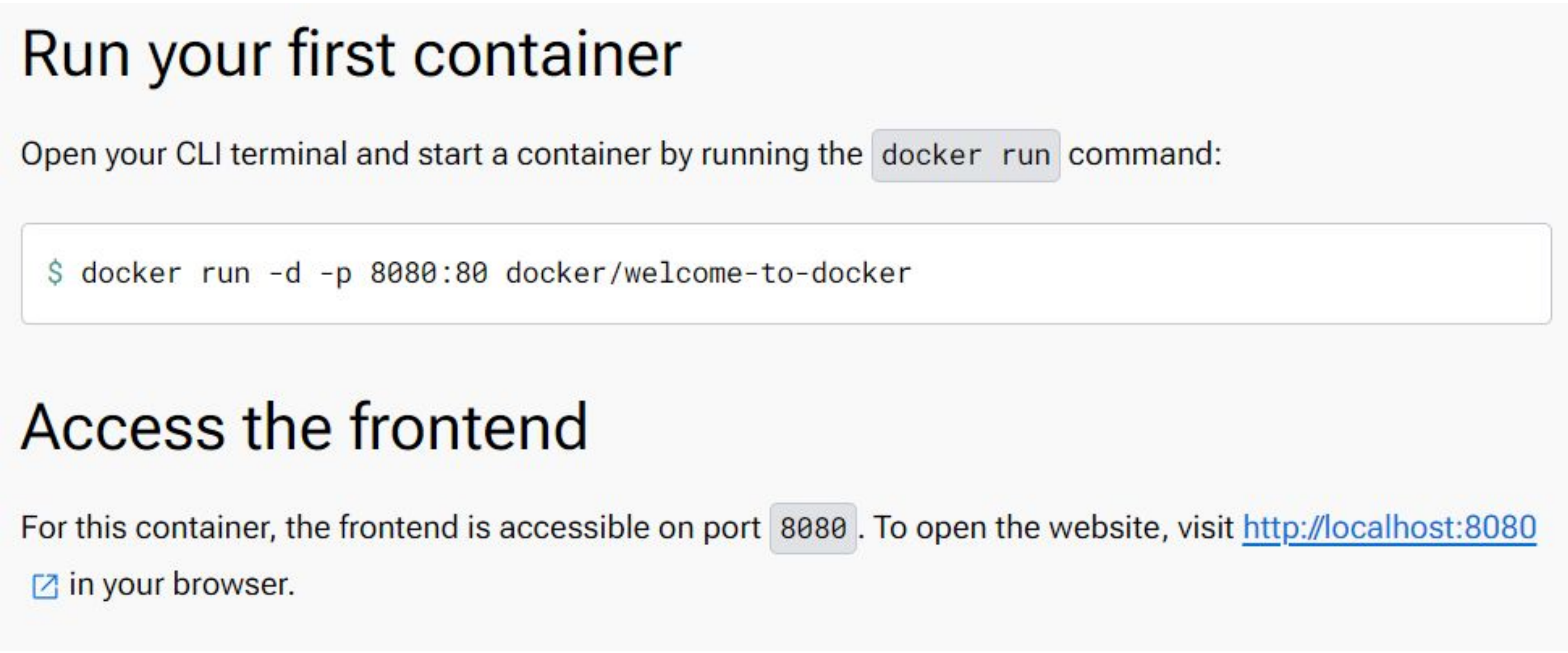
Cont.



4. Docker creates a network interface to connect the container to the default network, since you didn't specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes `/bin/bash`. Because the container is running interactively and attached to your terminal (due to the `-i` and `-t` flags), you can provide input using your keyboard while Docker logs the output to your terminal.
6. When you run `exit` to terminate the `/bin/bash` command, the container stops but isn't removed. You can start it again or remove it.

Install Docker Desktop

https://docs.docker.com/get-docker/



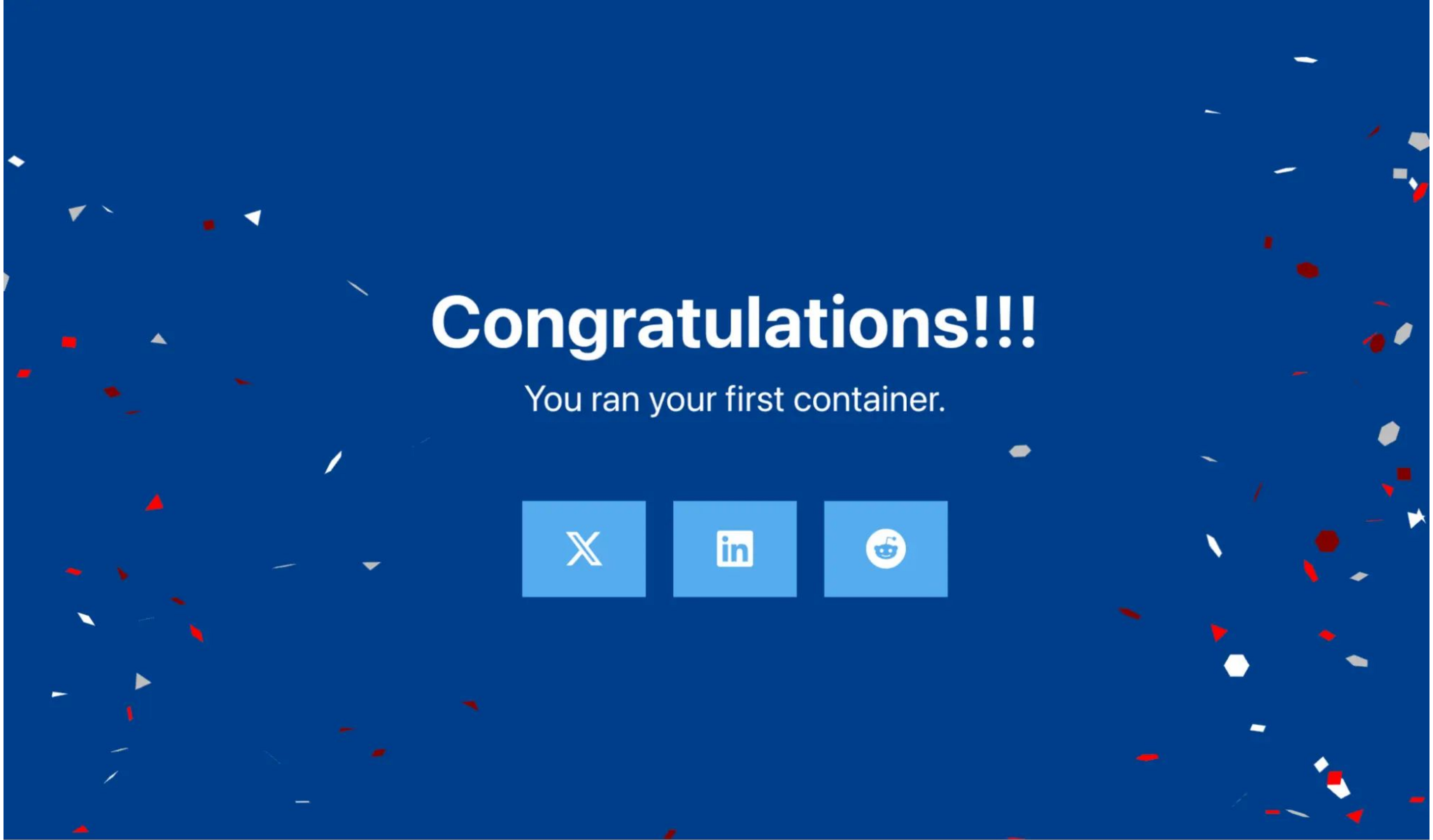
Run your first container

Open your CLI terminal and start a container by running the `docker run` command:

```
$ docker run -d -p 8080:80 docker/welcome-to-docker
```

Access the frontend

For this container, the frontend is accessible on port `8080`. To open the website, visit http://localhost:8080 in your browser.

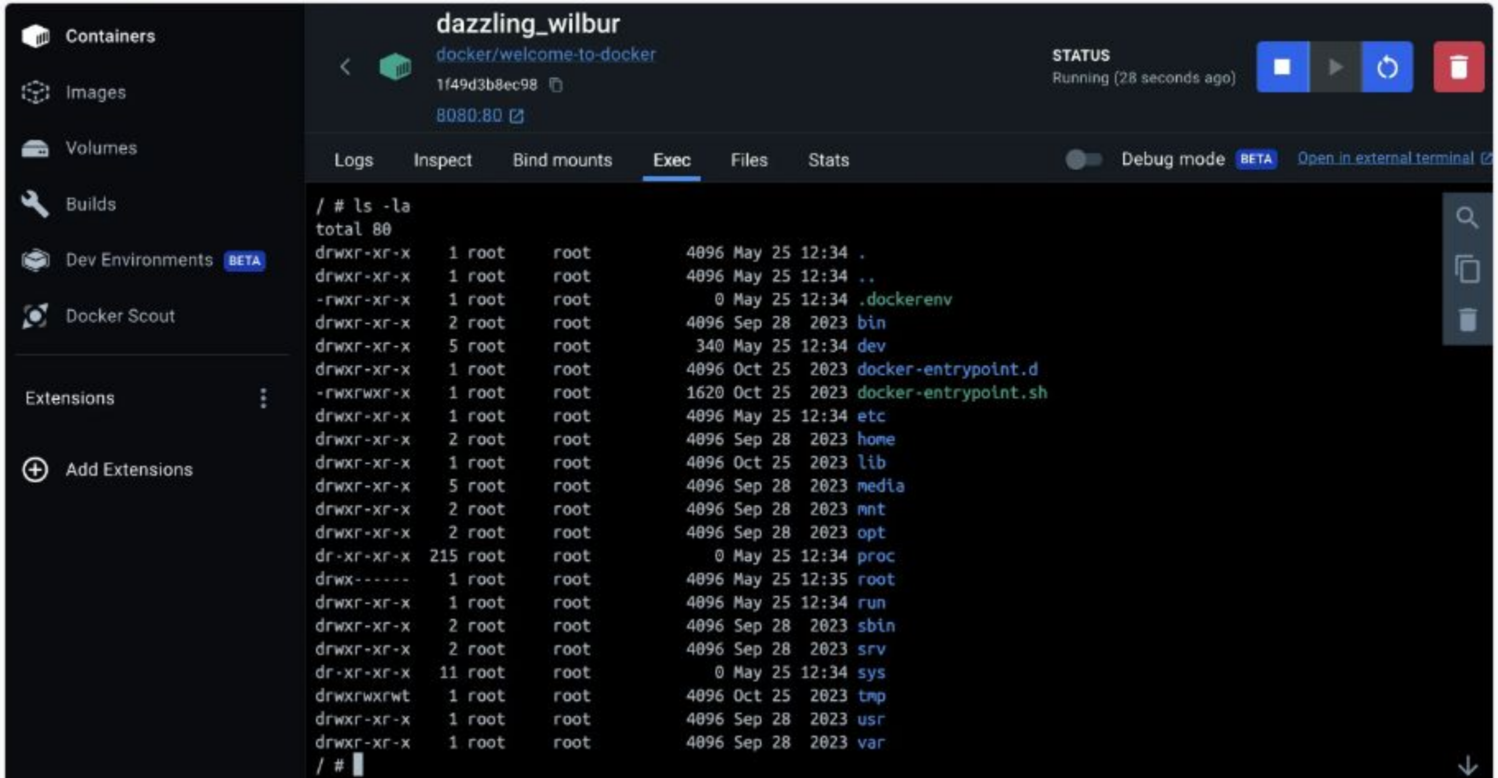


Congratulations!!!

You ran your first container.

Manage containers using Docker Desktop

1. Open Docker Desktop and select the **Containers** field on the left sidebar.
2. You can view information about your container including logs, and files, and even access the shell by selecting the **Exec** tab.



```
/ # ls -la
total 80
drwxr-xr-x    1 root     root          4096 May 25 12:34 .
drwxr-xr-x    1 root     root          4096 May 25 12:34 ..
-rwxr-xr-x    1 root     root             0 May 25 12:34 .dockerenv
drwxr-xr-x    2 root     root          4096 Sep 28  2023 bin
drwxr-xr-x    5 root     root           340 May 25 12:34 dev
drwxr-xr-x    1 root     root          4096 Oct 25  2023 docker-entrypoint.d
-rwxrwxr-x    1 root     root          1620 Oct 25  2023 docker-entrypoint.sh
drwxr-xr-x    1 root     root          4096 May 25 12:34 etc
drwxr-xr-x    2 root     root          4096 Sep 28  2023 home
drwxr-xr-x    1 root     root          4096 Oct 25  2023 lib
drwxr-xr-x    5 root     root          4096 Sep 28  2023 media
drwxr-xr-x    2 root     root          4096 Sep 28  2023 mnt
drwxr-xr-x    2 root     root          4096 Sep 28  2023 opt
dr-xr-xr-x  215 root     root             0 May 25 12:34 proc
drwx------    1 root     root          4096 May 25 12:35 root
drwxr-xr-x    1 root     root          4096 May 25 12:34 run
drwxr-xr-x    2 root     root          4096 Sep 28  2023 sbin
drwxr-xr-x    2 root     root          4096 Sep 28  2023 srv
dr-xr-xr-x   11 root     root             0 May 25 12:34 sys
drwxrwxrwt    1 root     root          4096 Oct 25  2023 tmp
drwxr-xr-x    1 root     root          4096 Sep 28  2023 usr
drwxr-xr-x    1 root     root          4096 Sep 28  2023 var
/ #
```

Docker Hub

To share your Docker images, you need a place to store them. This is where registries come in. While there are many registries, Docker Hub is the default and go-to registry for images. Docker Hub provides both a place for you to store your own images and to find images from others to either run or use as the bases for your own images.

1. To get started, either clone or download the project as a ZIP file to your local machine.

   ```
   $ git clone https://github.com/docker/getting-started-todo-app
   ```

   And after the project is cloned, navigate into the new directory created by the clone:

   ```
   $ cd getting-started-todo-app
   ```

2. Build the project by running the following command, swapping out `DOCKER_USERNAME` with your username.

   ```
   $ docker build -t DOCKER_USERNAME/getting-started-todo-app .
   ```

   For example, if your Docker username was `mobydock`, you would run the following:

   ```
   $ docker build -t mobydock/getting-started-todo-app .
   ```

3. To verify the image exists locally, you can use the `docker image ls` command:

   ```
   $ docker image ls
   ```

   You will see output similar to the following:

   ```
   REPOSITORY                          TAG       IMAGE ID       CREATED          SIZE
   mobydock/getting-started-todo-app   latest    1543656c9290   2 minutes ago    1.12GB
   ...
   ```

4. To push the image, use the `docker push` command. Be sure to replace `DOCKER_USERNAME` with your username:

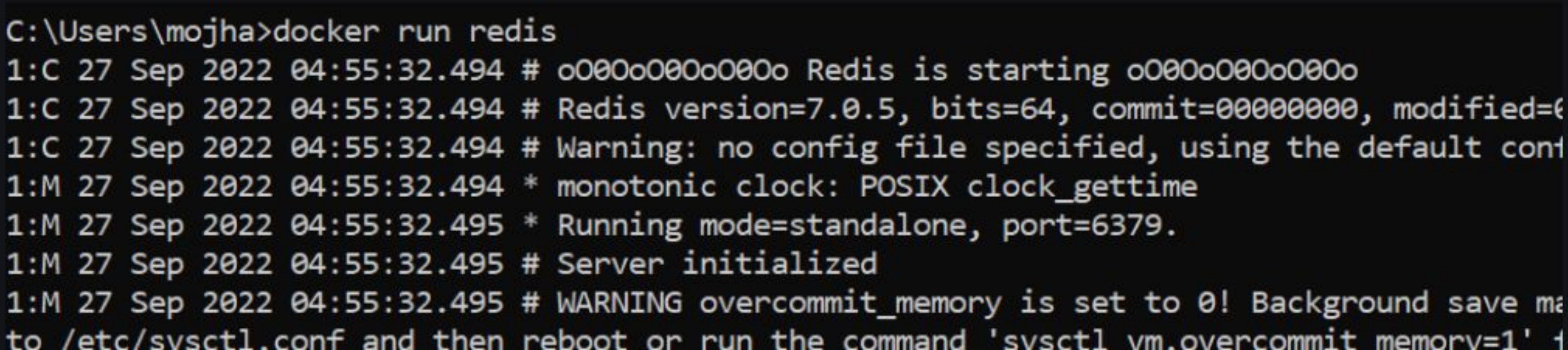
   ```
   $ docker push DOCKER_USERNAME/getting-started-todo-app
   ```

   Depending on your upload speeds, this may take a moment to push.

Docker Run command

This command is used to run a container from an image. The docker run command is a combination of the docker create and docker start commands. It creates a new container from the image specified and starts that container. if the docker image is not present, then the docker run pulls that.

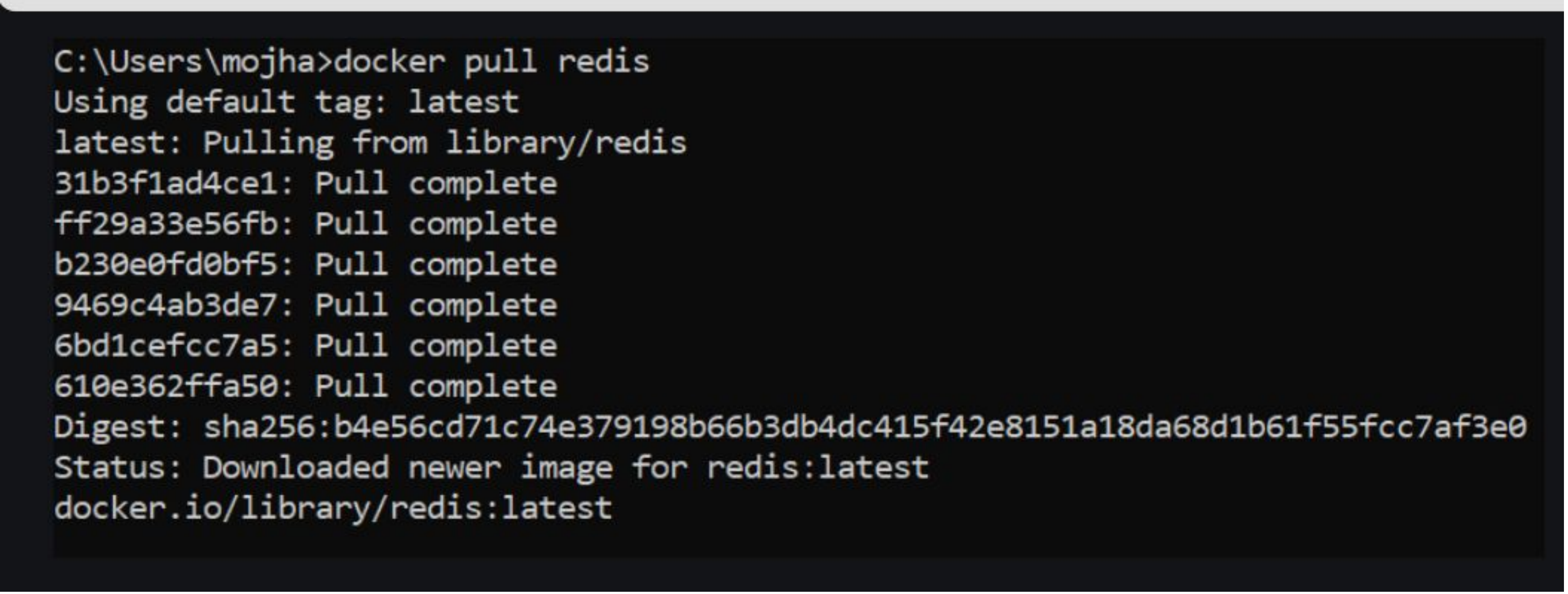
```
$ docker run <image_name>
To give name of container
$ docker run --name <container_name> <image_name>
```



```
C:\Users\mojha>docker run redis
1:C 27 Sep 2022 04:55:32.494 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 27 Sep 2022 04:55:32.494 # Redis version=7.0.5, bits=64, commit=00000000, modified=0
1:C 27 Sep 2022 04:55:32.494 # Warning: no config file specified, using the default conf
1:M 27 Sep 2022 04:55:32.494 * monotonic clock: POSIX clock_gettime
1:M 27 Sep 2022 04:55:32.495 * Running mode=standalone, port=6379.
1:M 27 Sep 2022 04:55:32.495 # Server initialized
1:M 27 Sep 2022 04:55:32.495 # WARNING overcommit_memory is set to 0! Background save ma
to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' f
```

Docker pull

```
$ docker pull <image_name>
```



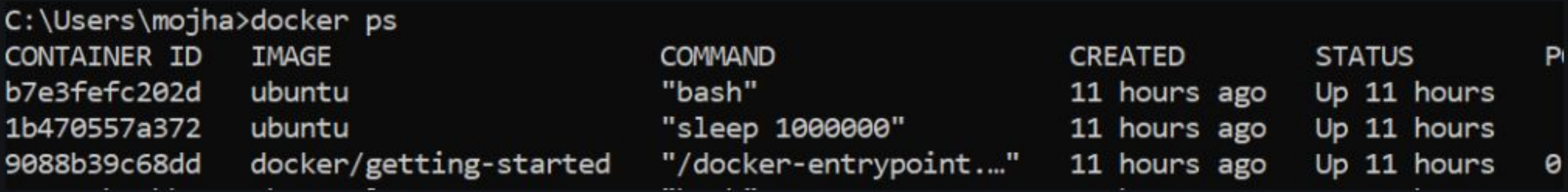
```
C:\Users\mojha>docker pull redis
Using default tag: latest
latest: Pulling from library/redis
31b3f1ad4ce1: Pull complete
ff29a33e56fb: Pull complete
b230e0fd0bf5: Pull complete
9469c4ab3de7: Pull complete
6bd1cefcc7a5: Pull complete
610e362ffa50: Pull complete
Digest: sha256:b4e56cd71c74e379198b66b3db4dc415f42e8151a18da68d1b61f55fcc7af3e0
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest
```

Docker PS

This command (by default) shows us a list of all the running containers. We can use various flags with it.

- **-a flag:** shows us all the containers, stopped or running.
- **-l flag:** shows us the latest container.
- **-q flag**: shows only the Id of the containers.

```
$ docker ps [options..]
```

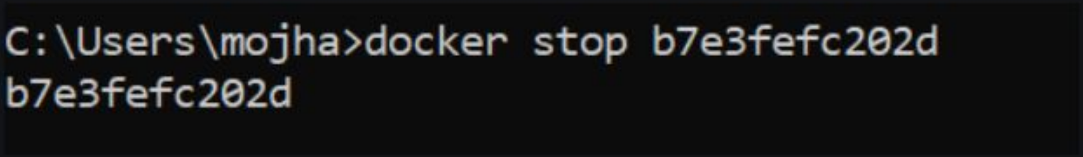


```
C:\Users\mojha>docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED          STATUS          P
b7e3fefc202d   ubuntu                    "bash"                   11 hours ago     Up 11 hours
1b470557a372   ubuntu                    "sleep 1000000"          11 hours ago     Up 11 hours
9088b39c68dd   docker/getting-started    "/docker-entrypoint.…"   11 hours ago     Up 11 hours     0
```

Docker Stop

This command allows you to stop a container if it has crashed or you want to switch to another one.

```
$ docker stop <container_ID>
```



```
C:\Users\mojha>docker stop b7e3fefc202d
b7e3fefc202d
```

Docker Start

Suppose you want to start the stopped container again, you can do it with the help of this command.

```
$ docker start <container_ID>
```

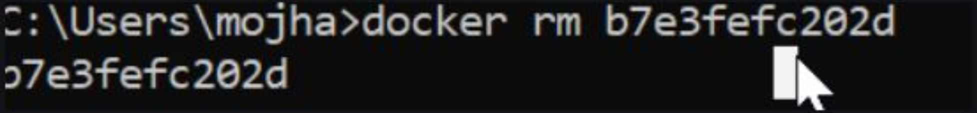
Docker rm

To delete a container. By default when a container is created, it gets an ID as well as an imaginary nam
such as confident_boyd, heuristic_villani, etc. You can either mention the container name or its ID.

Some important flags:

- **-f flag:** remove the container forcefully.
- **-v flag:** remove the volumes.
- **-l flag:** remove the specific link mentioned.

```
$ docker rm {options} <container_name or ID>
```



```
:\Users\mojha>docker rm b7e3fefc202d
7e3fefc202d
```

Exercise (will not be given grade for it)

Practice on the above mentioned materials and show results

References

1. https://aws.amazon.com/what-is/containerization/?nc1=h_ls
2. https://docs.docker.com/guides/docker-overview/
3. https://www.geeksforgeeks.org/docker-instruction-commands/