Handout: Container Lifecycle Management

1. General Syntax

Each Docker command follows this syntax:

```
docker [resource] command [options] [arguments]
```

2. Run a Container

To run a container, you need to provide an image — a kind of blueprint that defines what the container will contain and how it behaves.

```
docker [container] run [options] <image>
```

Note that `container` is optional here — `docker run` works the same way and is commonly used.
For example, you can run an interactive (`-it`) Ubuntu container using

```
docker run -it ubuntu
```

Each time you run this command, Docker creates a **new container**.

3. Run a Container in Detached Mode

Running a container in detached mode (`-d`) allows it to run in the background, freeing up the terminal for other tasks.

```
docker container run -d nginx
```

This starts an `nginx` container in the background, and Docker prints the container ID to the terminal.

Even though the container runs in the background, it remains active and uses system resources. You can manage it using other Docker commands.

4. Execute Commands in a Running Container

To interact with a running container, you can use the `docker exec` command to execute commands inside it.

```
docker container exec [options] <container-name-or-id> <command>
```

For example, to run an interactive Bash session in a running Ubuntu container:

```
docker container exec -it my-container bash
```

To list the contents of `/usr` inside `my-container` without opening an interactive session and exiting immediately:

```
docker container exec my-container ls /usr
```

Make sure the command you want to run is available within the container. If it is a program, it must already be installed inside the container's filesystem.

5. Create a Named Container

If you don't specify a name, Docker automatically assigns a randomly generated name. Besides, you can always refer to a container by its full ID or a unique abbreviation of it.
But usually it's more convenient to assign a meaningful name. Named containers are easier to identify, especially when managing multiple containers.
You can assign a name to a container at the time of creation by using the following command:

```
docker container run --name <name> <image>
```

For example:

```
docker container run --name my-web-server nginx
```

This creates a new container named `my-web-server` using the `nginx` image. You can now refer to the container using the name `my-web-server` instead of its container ID or its random name.

Names can include letters, numbers, underscores (`_`), dashes (`-`), and periods (`.`), but they cannot contain spaces. Also, they must be unique!

6. Rename a Container

If you didn't assign a meaningful name to a container when creating it, you can rename it later.

```
docker container rename <container-name-or-id> <new-name>
```

For example:

```
docker container rename frosty_morse my-web-server
```

This renames a container originally named `frosty_morse` to `my-web-server`.

You can rename a container at any time, even while it's running.

7. Stop a Container

If you want to stop a running container, you can use the stop command. This gracefully shuts down the container by sending a termination signal to its main process.

```
docker container stop <container-name-or-id>
```

For example:

```
docker container stop my-app
```

As a result, the container named `my-app` is no longer running and is moved to the "stopped" state. It will remain in this state until explicitly removed or restarted.

Stopping a container does not delete it or its data. The writable layer, including any files or changes made, will remain intact as long as the container exists.

8. Restart a Stopped Container

If a container has been stopped, you can restart it to resume its execution from where it left off. The container will retain all data and changes stored in its writable layer.

```
docker container start <container-name-or-id>
```

For example:

```
docker container start my-app
```

This resumes the container, restoring access to its writable layer and retaining all existing data and changes

Restarting a container is faster than creating a new one, as it avoids reinitializing the base image and dependencies.

9. Remove a Container

If you no longer need a container, you can remove it to free up system resources. This action deletes the container entirely, including all data stored within it.

```
docker container rm <container-name-or-id>
```

For example:

```
docker container rm my-app
```

A container must be stopped before it can be removed. To forcibly remove a running container, use the `-f` flag.

10. Run and Auto-Remove a Container

When you only need a container for a temporary task, you can run it with the `--rm` flag. This automatically removes the container once it stops, saving you from manually cleaning up.

```
docker container run --rm <image>
```

For example:

```
docker container run -it --rm ubuntu
```

11. Remove All Stopped Containers

To clean up your environment and free up system resources, you can remove all stopped containers in one step:

```
docker container prune
```

This command will prompt you for confirmation. To proceed and remove all stopped containers, simply type `y` and press `Enter`.

Running containers are unaffected by this command.

12. List Running Containers

You can list all containers that are currently running with

```
docker container ls
```

13. List All Containers

To list all containers, including stopped ones, you need to add the flag `--all` or `-a` (shorthand) to the previous command:

```
docker container ls -a
```

14. Inspect a Container

To show detailed information about a container, such as its state, configuration, and network settings, use:

```
docker container inspect <container-id-or-name>
```

15. View Container Logs

To view the logs of a container, including its output and error messages, use:

```
docker container logs <container-id-or-name>
```