Handout: Image Management

A **Docker image** is a static blueprint containing everything needed to run an application, while a container is a running instance of that image.

1. List Available Images

To see all locally available images:

```
docker image ls
```

This displays a list of downloaded images, including their repository name, tag, image ID, creation date, and size.
There also exists the `docker images` command — an older alias for `docker image ls` that serves the same purpose. But no other resource types have such shortcuts, which leads to an inconsistency. For clarity and consistency across all Docker commands, it's generally recommended to use the full format.

2. Referencing Docker Images

You can always refer to a Docker image by its image ID. However, this is rarely done in practice — using the image **name** and **tag** is more readable and maintainable.

- A **Docker image** name identifies the repository the image belongs to — for example, `nginx` or `python`.
- A **tag** refers to a specific version or variant of that image, such as `nginx:1.27` or `python:3.12-slim`.

To ensure consistency and avoid unexpected changes, it's best practice to always specify a tag when pulling or referencing an image.

Referencing an image by name and tag (e.g. `nginx:1.27`) **does not guarantee immutability**. Tags can be re-pointed — for example, if the image maintainer rebuilds the image or updates a base layer like the OS. If you ever need to pin an image exactly, you can use its **index digest**, e.g. `nginx@sha256:<digest>`. This ensures you're always using the same version, down to the byte. You can find available digests on the Docker Hub detail page.

3. Pull an Image from a Registry

To download an image from Docker Hub or another container registry:

```
docker [image] pull <image>
```

For example:

```
docker image pull nginx
```

This downloads the `latest` version of `nginx`. To specify a tag:

```
docker image pull nginx:1.27
```

If you don't specify a tag, Docker will default to the `latest` tag — which simply points to the version currently marked as "latest" by the image maintainer. This can change over time and lead to inconsistencies.

By default, if no registry is specified, Docker pulls the image from **Docker Hub**. You can also pull from other registries by including their hostname (e.g. `ghcr.io/user/image:tag`).

Ensure you pull only trusted images from verified sources to avoid security risks.

4. Inspect an Image

To view detailed information about an image, including metadata, layers, and configurations:

```
docker image inspect <image>
```

For example:

```
docker image inspect nginx:1.27
```

This returns a JSON object with information such as environment variables, entry points, and exposed ports.

5. View Image History

To see the history of an image, including the layers and commands used to create it:

```
docker image history <image-iname>
```

For example:

```
docker image history nginx:1.27
```

This lists each layer, its size, and the command that created it.

6. Tag an Image

To assign a new tag to an image:

```
docker image tag <source-image> <target-name[:tag]>
```

For example:

```
docker image tag nginx myrepo/nginx:latest
```

This renames `nginx:latest` to `myrepo/nginx:latest`, which is useful before pushing an image to a private repository.

7. Remove an Image

To delete an image from the local system:

```
docker image rm <image>
```

For example:

```
docker image rm nginx
```

Alternatively, you can use:

```
docker rmi nginx
```

If an image that you want to delete is being used by a container, you must remove the container first.

8. Remove Unused Images

To delete all dangling (unused) images:

```
docker image prune
```

This reclaims disk space by removing unreferenced images.

`docker image prune` does not remove images currently in use.

9. Create a Container

To create a container from an image without running it:

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

For example:

```
docker container create --name my-nginx nginx:1.27
```

This prepares a container that can be started later by `docker container start my-nginx`.

Options like `-t` (pseudo-TTY), `-p` (port mappings), `-d` (detached mode), **must be set when the container is created**. These settings become part of the container's definition and **cannot be changed afterward** — if you forget them, you'll need to remove and recreate the container.

10. Container Creation Recap

Command	What It Does	Notes
`docker image pull`	Downloads an image from a registry	Triggered automatically by `docker run` and `docker create` if the image is missing.
`docker container create`	Creates a container from an image without starting it	Pulls image if needed. All runtime options like `-p`, `-t`, `-d`, and `-i` must be set here — they cannot be added later.
`docker container start`	Starts an existing (created or stopped) container	Uses the config defined at creation time. To keep stdin open, you must pass `-i` again.
`docker container run`	Pulls, creates, and starts a container in one step	Combines all steps and applies the full runtime configuration in one go. Ideal for simple, one-off containers.