Handout: Image Distribution

Docker Hub is a cloud-based registry service that lets you store, share, and manage container images. It supports both public and private repositories, making it a suitable platform for development, testing, and production deployment workflows.

1. Registries vs. Repositories

Concept	Description	Example(s)
Repository	A collection of related images, typically organized by project or service.	`mycompany/frontend`
Registry	A service that stores and manages repositories.	Docker Hub, GitHub Container Registry, AWS ECR

- A **registry** can host multiple **repositories**
- A **repository** can hold multiple **tags** (versions) of the same image
- Repositories may be **public** (anyone can pull) or **private** (restricted access)

Best practice: use **one repository per image** — for example:

- `mycompany/frontend`
- `mycompany/checkout-backend`
- `mycompany/payments-db`

📦 Using one repository per image ensures clean **separation of concerns**: each service can be versioned, tagged, and deployed independently (which aligns well with CI/CD pipelines), while access can be restricted to the relevant teams — for example, the frontend team can't push changes to the backend image.

2. Private Images on Docker Hub

To work with private images on Docker Hub, you need a Docker Hub account. While the free tier includes a single private repository, most real-world projects require multiple services — which either means upgrading to a paid plan or hosting your own container registry.

📦 In real-world business settings, using **private repositories** is essential to protect proprietary code, internal tooling, and configuration details from public exposure. It also ensures that only authorized systems and team members can access and deploy your images.

2.1. Tagging and Pushing to Docker Hub

Before you can upload an image to Docker Hub, you must first log in:

```
docker login
```

This authenticates your local Docker CLI with your Docker Hub account and is required for pushing to private repositories.

📦 Before pushing, make sure the target repository already exists in your Docker Hub account — you can create it manually in the Docker Hub web interface.

Next, tag your image to point it to the correct repository location. Tagging is essential because it tells Docker where to push the image by including:

- your Docker Hub **username** (or organization)
- the **repository name**
- and an optional **version tag** (e.g., `:latest`, `:v1.0`)

```
docker image tag <local-image-name>
↪   <dockerhub-username>/<repository-name>:<tag>
```

Example:

```
docker image tag expense-tracker-frontend mycompany/expense-frontend:latest
```

📦 If you omit the tag, Docker defaults to `:latest`, which can lead to accidental overwrites if reused across builds. In CI pipelines or production, it's best to use explicit version tags (e.g., `:v1.0`, `:2026-07-04`) to ensure reproducibility and traceability.

Now push the image to the corresponding private repository:

```
docker push <dockerhub-username>/<repository-name>:<tag>
```

Example:

```
docker push mycompany/expense-frontend:latest
```

📦 `docker push` only uploads the specified tag. If you've tagged the same image multiple times (e.g., `:latest` and `:v1.0`), you must push each tag separately.

2.2. Pulling from a Private Repository

To pull an image from a **private** repository (e.g., during deployment), you must be authenticated:

```
docker login
```

Authentication is not required for public images, but it is mandatory for private repositories — otherwise the pull will fail with a permission error.

📦 To access a private repository, you must either own the Docker Hub account or have been granted permission through the Docker Hub UI (under the repository's *Collaborators* tab).

Then pull the desired image by specifying the full repository and tag:

```
docker pull <dockerhub-username>/<repository-name>:<tag>
```

Example:

```
docker pull mycompany/expense-frontend:latest
```

This downloads the image tagged `latest` from the `mycompany/expense-frontend` repository.

📦 Just like pushing, pulling is tag-specific. If you need multiple versions (e.g., `:latest`, `:v1.0`), each tag must be pulled separately.

3. Multi-Architecture Builds

Modern systems use different CPU architectures — for example, many developer machines (like Macs with Apple Silicon) use `arm64`, while most production servers and cloud hosts expect `amd64`. If you build an image locally on `arm64`, it may not run correctly on an `amd64` host.
To avoid such compatibility issues, you can explicitly specify the target architecture during the **build**, using the `--platform` flag:

```
docker build \
  --platform=<architecture> \
  -t <dockerhub-username>/<repository-name>:<tag> \
  .
```

Example:

```
docker build \
  --platform=linux/amd64 \
  -t mycompany/expense-frontend:latest \
  --build-arg VITE_API_BASE_URL=/api \
  .
```

This tells Docker to produce an image suitable for `linux/amd64`, regardless of the local build architecture.

📦 The platform must be set during **build** time — it cannot be changed later when creating or running a container. If the architecture doesn't match the deployment environment, the container may fail to start.