Publiez un image sur Docker Hub

Le processus de publication d'une image locale sur un repository remote comme docker hub est en 3 étapes:

- login
- tag
- push

Le tagging

Tagger une image ne copie pas l'image, mais l'associe juste à un (ou plusieurs) mots clefs.

- Il est possible de tagger une image avec plusieurs tags.
- La commande `docker images` liste tous les tags associés aux images. Vous voyez votre image listé plusieurs fois mais avec la même ID

La commande pour tagger une image est

```bash
docker tag <nom_image | nom_image:tag | image_id> <nouveau_nom:nouveau_tag | nom_im
```

Tagger une image a 2 objectifs:

Versioning

Tagger l'image permet tout d'abord d'associer une version à l'image : v01, latest, stable, ...

Par exemple pour une image avec l'id `abc123`, on peut l'associer a une version v1.0 avec :

```bash
docker tag abc123 myapp:v1.0
```

Cette commande crée l'alias `myapp:v1.0` pour l'image en question.

On peut ensuite la référencer non plus par son ID mais par son alias `myapp:v1.0`

Remote Repositories

Tagger l'image permet aussi de lier l'image à un repository remote pour pouvoir la publier (`push`) sur le repository

On peut ainsi tagger l'image `myapp` avec le username du repository

- un numéro de version v1.0 `docker tag myapp username/myapp:v1.0`

- avec le mot clé `latest` : `docker tag myapp username/myapp:latest`
- avec un autre registry (`otheruser`) : `docker tag myapp otheruser/myapp:stable`

Le pushing

Une fois associé par son tag avec le remote repository, on publie (`push` , upload) l'image sur le repository avec la commande

```bash
docker push username/image:tag
```

Publiez votre image my*nginx*alpine:v1

Vous allez publier maintenant votre image `my_nginx_alpine:v1` sur Docker Hub

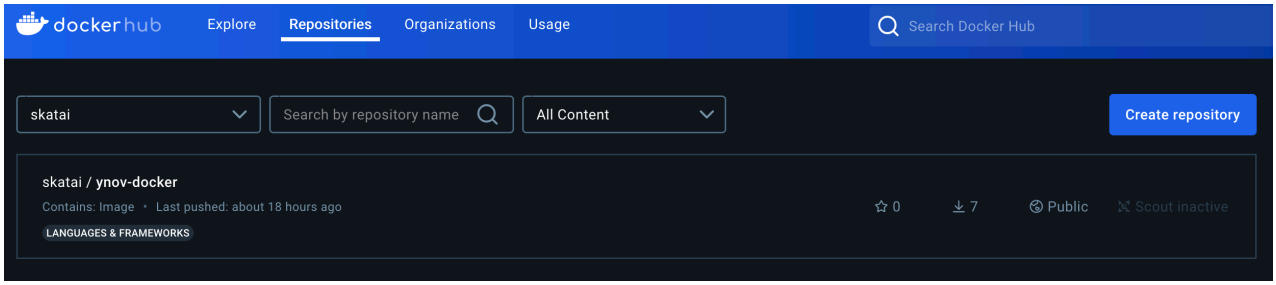
Le process: login, tag et push

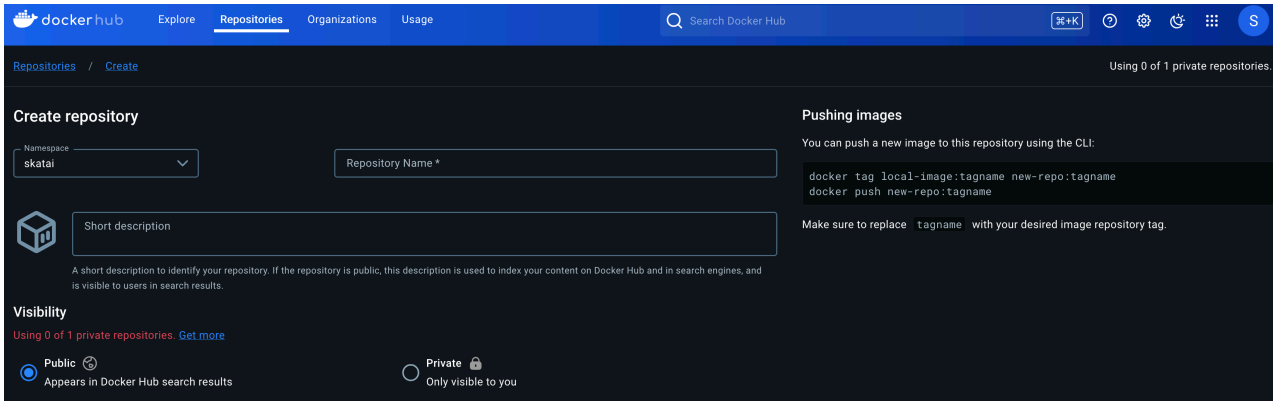
Le process pour publier une image locale sur Docker Hub consiste en :

- docker **login**
- docker **tag** local-image-name your-username/repository-name:tag
- docker **push** your-username/repository-name:tag

Créez votre compte et repo docker hub

- allez sur https://docker.com et créez votre compte si vous n'en avez pas déjà un.
- notez votre `username` . Il vous servira à tagger l'image.
- Créez un repository sur https://hub.docker.com





tag, login, push

Taggez votre image locale avec votre `username` docker hub et le nom de votre repository

Par exemple:

- pour le username : `skatai`
- le compte : `ynov-docker`
- et le tag : `v1`

Je tagge l'image locale `my_nginx_alpine:v1` avec:

```Bash
docker tag my_nginx_alpine:v1 skatai/ynov-docker:my_nginx_alpine-v1
```

Cette image apparait avec ses différents alias lorsque l'on fait `docker images`

Puis loguez-vous sur docker:

```Bash
docker login
```

Enfin publiez l'image sur Docker Hub avec `docker push`

```Bash
docker push skatai/ynov-docker:my_nginx_alpine-v1
```

Le nom, le tag et le repository

En local, l'image a un nom `my_nginx_alpine` et un tag `v1` qui illustre la version de l'image.

On a 2 stratégies pour stocker les images sur Docker Hub.

- 1 repo et plusieurs applications et leurs images

```Bash
<account>/<account_main-repo>:<tag liés aux images des applications>
```

- une repo par application, chaque repo contient seulement les images de l'application en question
- `<account>/<repo-application>:<versions des images liées à l'application>`
- `<account>/<autre_repo-application>:<versions des images liées à l'autre application>`

1 repo, plusieurs applications

Dans le repository, le nom local et le tag se merge (se fonde) en un nouveau tag `my_nginx_alpine-v1`.

Dans la commande tag: on nomme l'image avec son identifiant en remote

```bash
docker tag <local_image_name>:<local_image_tag>  <account_name>/<repo_name>:<remote
```

où " = `<local_image_name>-<local_image_tag>`

Cela permet de stocker plusieurs images dans le même repository mais le nom de l'image en remote est mergé avec le tag.

Une repo par projet

En fait, il est plus courant d'avoir un repository par application. Le nom du repository devient alors le nom de l'appli cad le nom de l'image et le tag de l'image reste le tag (version, latest, ...).

Par exemple pour 2 repo:

- `skatai/nginx-demo:01`
- `skatai/hello-world:latest`

Cependant, il n'est pas possible d'utiliser la commande docker pour créer des repositories sur Docker hub. Il faut passer par l'interface web.

Comme nous allons créer plusieurs applications il est plus simple de les stocker toutes dans le même repository en jouant sur la composition du tag de l'image.

En quoi consiste le push d'une image ?

La commande `docker push` n'envoi pas le fichier executable de l'image.

Docker envoie les **layers** de l'image à Docker Hub.

Une image est composée de plusieurs layers (couches) chacune identifiée par un hash de son contenu.

Ce qui est stocké sur Docker Hub :

- **Layers** : Docker Hub stocke les différentes layers qui composent votre image. Ces layers contiennent les modifications du système de fichiers et les métadonnées.
- **Le manifeste d'image** : Un fichier JSON qui décrit l'image, y compris les couches qui la composent et d'autres métadonnées. visible via `docker inspect image_id`
- **Configuration d'image** : Contient la configuration d'exécution, comme les variables d'environnement, les ports exposés et la commande par défaut à exécuter.

Docker Hub ne fait qu'héberger votre image. Elle ne devient exécutable que lorsqu'elle est récupérée puis exécutée sur une machine avec `docker run` .

De même, Docker Hub n'exécute ni les images ni les containers, mais il peut réaliser certains processus :

- **Scan d'images** : Recherche de vulnérabilités connues.
- **Webhooks** : Déclenche des actions lors du push d'une image.
- **Builds automatisés** : Peut créer automatiquement des images à partir de repository Git.

Quand un utilisateur récupère l'image depuis Docker Hub avec la commande

`docker pull skatai/ynov-docker:v1` :

- Docker télécharge les couches nécessaires et reconstruit l'image localement ce qui lui permet ensuite de pouvoir exécuter un conteneur à partir de cette image