TP- Utilisation de Docker, création d'une image.

1- Créer une image à partir d'un conteneur.

Dans le cadre de cet exercice, on utilisera un conteneur sous Ubuntu.

- Etape 1 : exécuter la commande suivante : **docker container run –ti ubuntu bash**

Cette commande vous permet d'importer une image depuis le Docker Store, et de faire fonctionner un shell bash à l'intérieur de ce conteneur.

- Etape 2 : personnalisation du conteneur en installant de nouveaux paquets à l'aide la commande suivante : **apt-get update**

  apt-get install –y figlet

  figlet <nom-figlet>

Vous devrez normalement avoir les mots correspondant au nom de votre figlet affichés sur votre écran. Vous pouvez quitter le conteneur Ubuntu.

Admettons maintenant que l'application figlet que vous venez d'installer est relativement utile et que vous souhaitez la partager avec vos collaborateurs. Vous pourriez tout simplement leur expliquer votre façon de procéder pour installer cette application et leur demander d'en faire de même. Toutefois, si cette application était une application plus importante où de nombreuses configuration ont été nécessaire, tenter de faire reproduire cette installation par d'autres personnes augmenterait fortement le risque d'avoir des erreurs parmi les applications reproduites.

Tout cela pour en arriver à la conclusion que la création d'une image semble être la bonne solution au partage entre collaborateurs.

- Etape 4 : Récupération de l'ID du conteneur. Exécuter la commande suivante : **docker container ls –a**

- Etape 5 : Création d'une image locale avec la commande **commit** : **docker container commit ID_de_vote_conteneur**

- Etape 6 : Vérification de la création de l'image à l'aide de la commande : **docker image ls**

Vous devez normalement voir apparaître dans la liste de vos images, l'image Ubuntu de votre conteneur ainsi que l'image que vous venez de créer. Cette dernière, comme vous pourrez le constater, ne possède ni de repository, ni de tag.

- Etape 7 : Attribution d'un tag à votre image à l'aide de la commande suivante :
  docker image tag <ID_de_votre_image> <nom_de_l'image>

Vous pouvez ensuite vérifier que l'attribution du tag à bien été réalisée à l'aide de la commande **docker image ls**

- Etape 8 : vous allez maintenant essayer de faire fonctionner un conteneur comportant l'image que vous venez de créer à l'aide de la commande suivante :
  docker container run <nom_de_l'image>

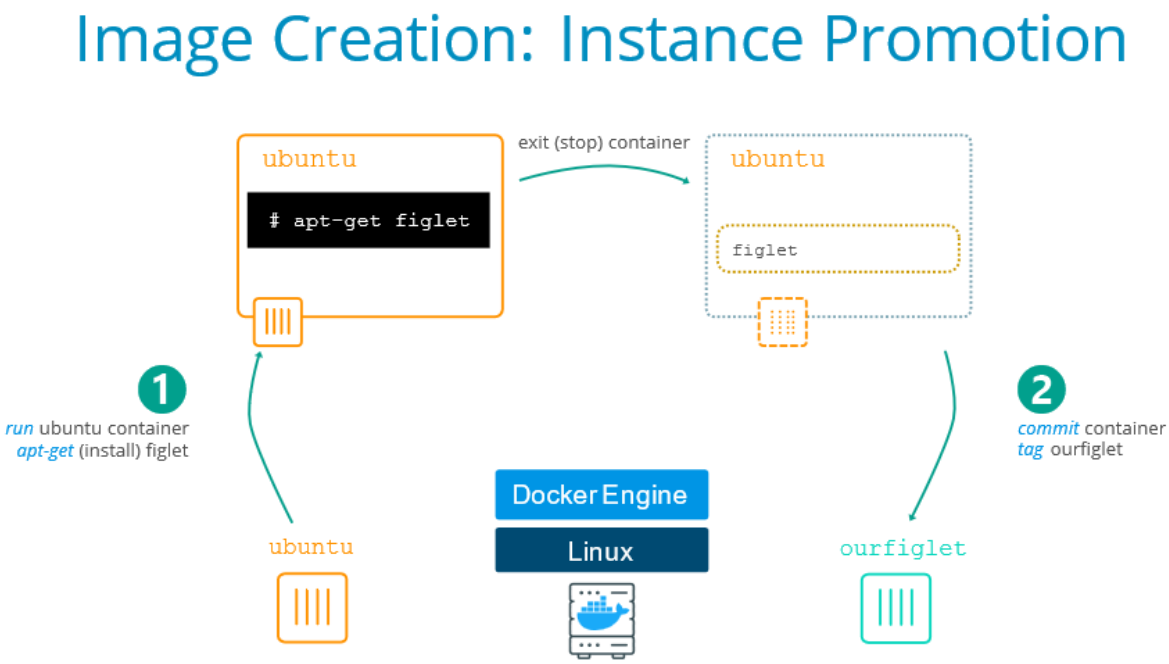

Figure 1: schéma de ce qui qui vient d'être réalisé.

2- Création d'une image à l'aide de Dockerfile.

Au lieu de créer une image binaire statique, nous pouvons utiliser un fichier appelé un Dockerfile pour créer une image. Le résultat final est essentiellement le même, mais avec un Dockerfile nous fournissons les instructions pour construire l'image, plutôt que juste les fichiers binaires bruts. C'est utile parce qu'il devient beaucoup plus facile de gérer des changements, d'autant plus que vos images deviennent plus grandes et plus complexes.

Par exemple, si une nouvelle version du figlet est sortie nous aurions à recréer notre image depuis le début, ou à exécuter notre image et à mettre à jour la version installée du figlet. Au contraire, un Dockerfile inclurait les commandes apt-get utilisée pour installer le figlet pour que nous - ou quelqu'un utilisant Dockerfile - puissions simplement recomposer l'image en utilisant ces instructions.

Nous utiliserons un exemple simple dans cette exercice et construirons une application « hello world" en Node.js. Ne soyez pas préoccupé si vous n'êtes pas familiers de Node.js : Docker (et cet exercice) n'exige pas que vous connaissiez tous ces détails.

On va commencer cet exercice, en créant un fichier dans lequel on récupérera le hostname et on l'affichera.
ATTENTION ! Si vous êtes encore dans votre conteneur Ubuntu, vous devez le quitter pour revenir à l'invite de commande de base de Docker. Pour cela, entrez la commande **exit**

- Etape 1 : Créer un fichier appelé *index.js* à l'aide d'un éditeur de texte comme nano ou vi. Ici, n'importe quel éditeur de texte Linux fera l'affaire.
  Dans ce fichier, écrivez les éléments suivants :
  var os = require(« os ») ;
  var hostname = os.hostname() ;
  console.log(« hello from » + hostname) ;

Le fichier que vous venez de créer est le code JavaScript pour votre serveur. Comme vous pouvez vous en doutez, Node.js sert simplement à afficher un message « hello ». On va « dockeriser » cette application en créant un Dockerfile. On utilisera alpine comme OS de base, y ajouter du Node.js et copier notre code source dans le conteneur. Nous spécifierons aussi la commande par défaut à exécuter pour la création de conteneurs.

- Etape 2 : Créer un fichier *Dockerfile* et copier le contenu suivant à l'intérieur, une nouvelle fois à l'aide d'un éditeur de texte Linux.
  FROM alpine
  RUN apk update && apk add nodejs
  COPY . /app
  WORKDIR /app
  CMD [« node », « index.js »]

Maintenant, passons à la création de votre première image en utilisant Dockerfile. Appelez votre image *hello:v0.1*. Entrez la commande suivante : **docker image build –t hello:v0.1**

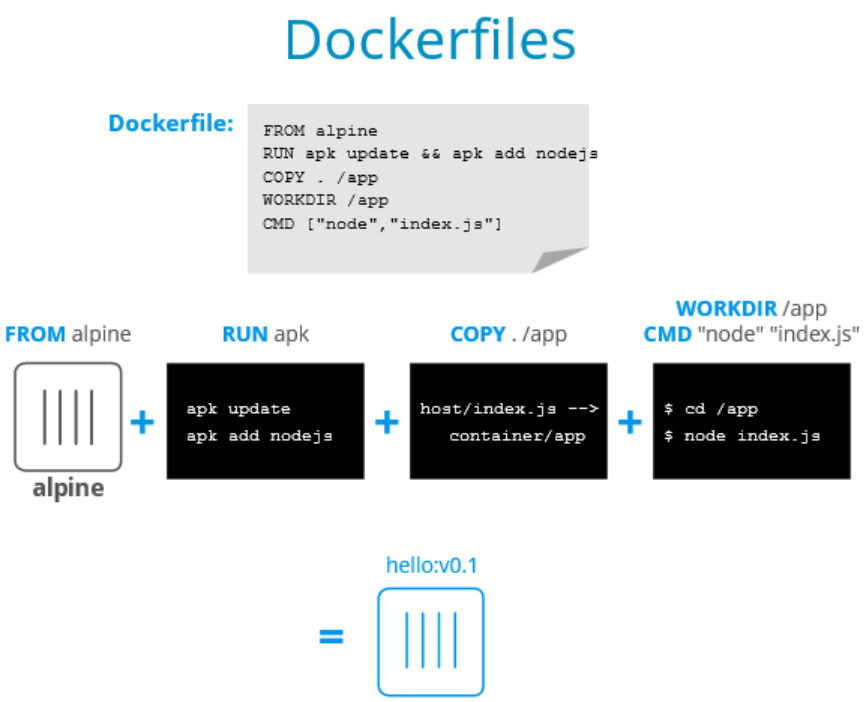

Figure 2: résumer de ce que vous venez de réaliser.

- Etape 3 : pour vous assurer que votre application fonctionne correctement, vérifier en démarrant le conteneur de votre application avec la commande suivante :
  docker container run hello:v0.1

Vous devriez obtenir en sortie quelque chose de semblable à ce qui suit (excepté l'ID qui sera différente) : **hello from 92d79b6de29f**

Qu'est-ce qui vient de se passer ?

Vous venez de créer deux fichiers : le code de votre application (index.js) est un simple morceau de code JavaScript qui affiche votre message. Le Dockerfile, quant à lui, représente les instructions pour la machine Docker pour créer votre conteneur spécialisé. Votre Dockerfile fait les choses suivantes :

- Il spécifie l'image de base d'où extraire les éléments (ici alpine) ;
- Il fait fonctionner deux commandes (*apk update* et *apk add*) dans ce conteneur qui installe le serveur Node.js ;
- Vous lui avez ensuite demandé de copier les fichiers depuis notre dossier vers le conteneur, soit le fichier *index.js* ;
- Vous avez ensuite indiqué le WORKDIR – le dossier que le conteneur devra utiliser à son démarrage ;
- Pour terminer, vous avez donné la commande (CMD) à faire fonctionner lors du démarrage du conteneur.