Partie 1 : Hello World

Lancez Cloud Shell et saisissez la commande suivante afin d'exécuter un conteneur Hello World pour commencer :

```
docker run hello-world
```

(Résultat de la commande)

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4
    b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest
Hello from Docker!
This message shows that your installation appears to be working
    correctly.
...
```

Ce conteneur simple affiche `Hello from Docker!` sur votre écran.

La commande en elle-même est assez simple, mais vous pouvez constater dans le résultat le nombre conséquent d'étapes effectuées.

Le daemon Docker a recherché l'image "hello-world", ne l'a pas trouvée en local, l'a extraite d'un registre public appelé Docker Hub, a créé un conteneur à partir de cette image et a exécuté ce conteneur.

Exécutez la commande suivante pour examiner l'image de conteneur extraite de Docker Hub :

```
docker images
```

(Résultat de la commande)

```
REPOSITORY     TAG       IMAGE ID       CREATED       SIZE
hello-world    latest    feb5d9fea6a5   14 months ago   13.3kB
```

Il s'agit de l'image extraite du registre public Docker Hub. L'ID de l'image se présente sous la forme d'un hachage SHA256 qui identifie l'image Docker provisionnée. Par défaut, le daemon Docker recherche une image dans le registre public lorsqu'il ne la trouve pas en local.

Exécutez le conteneur à nouveau :

```
docker run hello-world
```

(Résultat de la commande)

```
Hello from Docker!
```

```
2 This message shows that your installation appears to be working
     correctly.
3 To generate this message, Docker took the following steps:
4 ...
```

Notez que lorsque vous exécutez à nouveau la commande, le daemon Docker trouve l'image dans votre registre local et exécute le conteneur à partir de cette image. Il n'a pas besoin d'extraire l'image de Docker Hub.

Pour finir, utilisez la commande suivante afin d'examiner les conteneurs en cours d'exécution :

```
1 docker ps
```

(Résultat de la commande)

```
1 CONTAINER ID        IMAGE                COMMAND              CREATED
                 STATUS              PORTS               NAMES
```

Aucun conteneur n'est en cours d'exécution. Vous avez déjà quitté les conteneurs hello-world que vous avez précédemment exécutés.

Pour voir tous les conteneurs, y compris ceux dont l'exécution est terminée, lancez la commande docker ps -a :

```
1 docker ps -a
```

(Résultat de la commande)

```
1 CONTAINER ID      IMAGE           COMMAND      ...     NAMES
2 6027ecba1c39      hello-world     "/hello"     ...     elated_knuth
3 358d709b8341      hello-world     "/hello"     ...     epic_lewin
```

Vous obtenez l'ID du conteneur (Container ID), un UUID généré par Docker pour identifier le conteneur, ainsi que d'autres métadonnées relatives à l'exécution. Les noms de conteneur (Names) sont également générés de manière aléatoire, mais peuvent être définis grâce à la commande docker run –name [nom-conteneur] hello-world.

Partie 2 : Créer un conteneur

Dans cette section, vous allez créer une image Docker basée sur une application de nœud simple.

Exécutez la commande suivante pour créer un dossier nommé test et basculer dans ce dossier.

```
1 mkdir test && cd test
```

Créez un fichier Dockerfile :

```
cat > Dockerfile <<EOF
# Use an official Node runtime as the parent image
FROM node:lts
# Set the working directory in the container to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Make the container's port 80 available to the outside world
EXPOSE 80
# Run app.js using node when the container launches
CMD ["node", "app.js"]
EOF
```

Ce fichier indique au daemon Docker comment créer votre image.

La première ligne indique l'image parent de base, qui correspond ici à l'image Docker officielle de la version LTS (support à long terme) du nœud. Dans la deuxième ligne, vous devez définir le répertoire de travail (actuel) du conteneur.

Dans la troisième ligne, vous ajoutez le contenu du répertoire actuel (indiqué par le ".") dans le conteneur.

Vous indiquez ensuite le port du conteneur afin d'autoriser les connexions sur ce port, puis terminez en exécutant la commande "node" pour démarrer l'application.

Remarque : Prenez le temps de consulter la documentation de référence sur les commandes Dockerfile pour comprendre chaque ligne de ce fichier Dockerfile. Vous allez maintenant rédiger le code de l'application du nœud, puis créer l'image.

Exécutez la commande suivante pour créer l'application du nœud :

```
cat > app.js <<EOF
const http = require('http');
const hostname = '0.0.0.0';
const port = 80;
const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World\n');
});
server.listen(port, hostname, () => {
    console.log('Server running at http://%s:%s/', hostname, port);
});
process.on('SIGINT', function() {
    console.log('Caught interrupt signal and will exit');
    process.exit();
});
EOF
```

Il s'agit d'un serveur HTTP simple qui écoute le port 80 et renvoie "Hello World".

Maintenant, créez l'image.

Vous pouvez remarquer là encore la présence du ".", qui désigne le répertoire actuel. Vous devez donc exécuter cette commande depuis le répertoire qui contient le fichier Dockerfile :

```
docker build -t node-app:0.1 .
```

L'exécution de cette commande peut prendre quelques minutes. Une fois l'opération terminée, le résultat doit se présenter comme suit :

```
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM node:lts
6: Pulling from library/node
...
...
...
Step 5 : CMD node app.js
 ---> Running in b677acd1edd9
 ---> f166cd2a9f10
Removing intermediate container b677acd1edd9
Successfully built f166cd2a9f10
```

L'option `-t` permet de nommer une image et de lui ajouter un tag par le biais de la syntaxe nom:tag. Ici, le nom de l'image est node-app et le tag 0.1. L'utilisation d'un tag est vivement recommandée lors de la création d'images Docker. Si vous ne définissez pas de tag, la valeur latest sera attribuée par défaut aux images et il sera plus difficile de distinguer les images plus récentes des images plus anciennes. Notez également comment, pour chaque ligne du Dockerfile ci-dessus, des couches de conteneur intermédiaires sont ajoutées au fur et à mesure que l'image se crée.

À présent, exécutez la commande suivante pour examiner les images que vous avez créées :

```
docker images
```

Le résultat doit se présenter comme suit :

```
REPOSITORY     TAG      IMAGE ID       CREATED             SIZE
node-app       0.1      f166cd2a9f10   25 seconds ago      656.2 MB
node           lts      5a767079e3df   15 hours ago        656.2 MB
hello-world    latest   1815c82652c0   6 days ago          1.84 kB
```

Notez que node désigne l'image de base et que node-app désigne l'image que vous avez créée. Vous ne pouvez pas supprimer node sans d'abord supprimer node-app.

La taille de l'image est relativement petite par rapport aux VM. D'autres versions de l'image du nœud telles que node:slim et node:alpine peuvent vous donner des images encore plus petites pour une meilleure portabilité.

La réduction de la taille des conteneurs est abordée plus en détail dans la section “Sujets avancés”. Toutes les versions du dépôt officiel sont précisées sur la page node.

Partie 3 Executer un conteneur

Dans ce module, vous exécutez des conteneurs basés sur l’image que vous avez créée à l’aide du code suivant :

```
docker run -p 4000:80 --name my-app node-app:0.1
```

(Résultat de la commande)

```
Server running at http://0.0.0.0:80/
```

L’option `--name` vous permet de nommer le conteneur si vous le souhaitez. L’option `-p` demande à Docker de faire correspondre le port 4000 de l’hôte avec le port 80 du conteneur. Vous pouvez à présent accéder au serveur via l’adresse http://localhost:4000. Sans mise en correspondance des ports, vous ne pourrez pas accéder au conteneur sur “localhost”.

Ouvrez un autre terminal (dans Cloud Shell, cliquez sur l’icône +), puis testez le serveur :

```
curl http://localhost:4000
```

(Résultat de la commande)

```
Hello World
```

Le conteneur s’exécute à condition que le terminal initial soit en cours d’exécution. Si vous souhaitez que le conteneur s’exécute en arrière-plan (sans être associé à la session du terminal), vous devez définir l’option -d.

Fermez le terminal initial, puis exécutez la commande suivante pour arrêter l’exécution du conteneur et le supprimer :

```
docker stop my-app && docker rm my-app
```

À présent, exécutez la commande ci-dessous pour lancer le conteneur en arrière-plan :

```
docker run -p 4000:80 --name my-app -d node-app:0.1
docker ps
```

(Résultat de la commande)

```
CONTAINER ID   IMAGE          COMMAND        CREATED          ...  NAMES
xxxxxxxxxxxx   node-app:0.1   "node app.js"  16 seconds ago   ...  my-
    app
```

Notez que le résultat de la commande docker ps indique que le conteneur est en cours d'exécution. Pour consulter les journaux, exécutez la commande docker logs [id_conteneur]. Remarque : Il est inutile d'indiquer l'intégralité de l'ID du conteneur si les premiers caractères l'identifient de manière unique. Par exemple, vous pouvez exécuter docker logs 17b si l'ID du conteneur est 17bcaca6f.... docker logs [id_conteneur]

(Résultat de la commande)

```
Server running at http://0.0.0.0:80/
```

Maintenant, modifiez l'application.

Dans votre session Cloud Shell, ouvrez le répertoire de test que vous avez créé un peu plus tôt :

```
cd test
```

Dans l'éditeur de texte de votre choix (par exemple, nano ou vim), modifiez app.js en remplaçant "Hello World" par une autre chaîne :

```
....
const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Welcome to Cloud\n');
});
....
```

Créez cette image, puis ajoutez-lui le tag 0.2 :

```
docker build -t node-app:0.2 .
```

(Résultat de la commande)

```
Step 1/5 : FROM node:lts
 ---> 67ed1f028e71
Step 2/5 : WORKDIR /app
 ---> Using cache
 ---> a39c2d73c807
Step 3/5 : ADD . /app
 ---> a7087887091f
Removing intermediate container 99bc0526ebb0
Step 4/5 : EXPOSE 80
 ---> Running in 7882a1e84596
 ---> 80f5220880d9
Removing intermediate container 7882a1e84596
Step 5/5 : CMD node app.js
 ---> Running in f2646b475210
 ---> 5c3edbac6421
Removing intermediate container f2646b475210
```

```
Successfully built 5c3edbac6421
Successfully tagged node-app:0.2
```

Notez qu'à l'étape 2, vous utilisez une couche de cache existante. À partir de l'étape 3, les couches sont modifiées, car vous avez effectué un changement dans app.js.

Exécutez un autre conteneur avec la nouvelle version de l'image. Comme vous pouvez le constater, nous effectuons le mappage sur le port 8080 de l'hôte plutôt que sur le port 80. Vous ne pouvez pas vous servir du port 4000 de l'hôte, car il est déjà utilisé.

```
docker run -p 8080:80 --name my-app-2 -d node-app:0.2
docker ps
```

(Résultat de la commande)

```
CONTAINER ID     IMAGE              COMMAND             CREATED
xxxxxxxxxxxx     node-app:0.2       "node app.js"       53 seconds ago
          ...
xxxxxxxxxxxx     node-app:0.1       "node app.js"       About an hour ago
        ...
```

Testez les conteneurs :

```
curl http://localhost:8080
```

(Résultat de la commande)

Welcome to Cloud Testez à présent le premier conteneur que vous avez créé :

```
curl http://localhost:4000
```

(Résultat de la commande)

```
Hello World
```

Partie 4 : Débugguer un conteneur

Maintenant que vous savez créer et exécuter des conteneurs, découvrez comment effectuer un débogage.

Pour consulter les journaux d'un conteneur, exécutez la commande docker logs [id_conteneur]. Pour suivre la sortie du journal au cours de l'exécution du conteneur, utilisez l'option -f.

```
docker logs -f [id_conteneur]
```

(Résultat de la commande)

```
Server running at http://0.0.0.0:80/
```

Vous pouvez parfois avoir besoin de démarrer une session bash interactive dans le conteneur en cours d'exécution.

Pour ce faire, vous pouvez utiliser docker exec. Ouvrez un autre terminal (dans Cloud Shell, cliquez sur l'icône +), puis saisissez la commande suivante :

```
docker exec -it [id_conteneur] bash
```

Avec l'option `-it`, vous pouvez interagir avec un conteneur en attribuant un "pseudo-tty" et en gardant "stdin" ouvert. Comme vous pouvez le constater, bash a été exécuté dans le répertoire WORKDIR (/app) indiqué dans le fichier Dockerfile. Vous disposez maintenant d'une session de shell interactive dans le conteneur à déboguer.

(Résultat de la commande)

```
root@xxxxxxxxxxxx:/app#
```

Examinez le répertoire :

```
ls
```

(Résultat de la commande)

```
Dockerfile  app.js
```

Quittez la session bash :

```
exit
```

Vous pouvez examiner les métadonnées d'un conteneur dans Docker à l'aide de la commande "docker inspect" :

docker inspect [id_conteneur]

(Résultat de la commande)

```
[
    {
        "Id": "xxxxxxxxxxxx....",
        "Created": "2017-08-07T22:57:49.261726726Z",
        "Path": "node",
        "Args": [
            "app.js"
        ],
...
```

Utilisez l'option `--format` pour examiner des champs spécifiques du fichier JSON renvoyé. Exemple :

```
docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' [id_conteneur]
```

(Exemple de résultat)

```
192.168.9.3
```

Consultez les documentations Docker suivantes pour en savoir plus sur le débogage :

Documentation de référence sur "docker inspect" Documentation de référence sur "docker exec"

Partie 5 : Publier une image sur Artifact Registry

Vous allez à présent transférer votre image vers Google Artifact Registry. Vous supprimerez ensuite l'ensemble des conteneurs et des images pour simuler un nouvel environnement, puis vous extrairez et exécuterez vos conteneurs. Vous pourrez ainsi constater la portabilité des conteneurs Docker.

Pour transférer des images vers votre registre privé hébergé par Artifact Registry, vous devez leur ajouter un tag correspondant au nom du registre.

Le format est -docker.pkg.dev/mon-projet/mon-dépôt/mon-image.

Créer le dépôt Docker cible :

Vous devez créer un dépôt pour pouvoir y transférer des images. Le transfert d'une image ne peut pas déclencher la création d'un dépôt et le compte de service Cloud Build ne dispose pas des autorisations nécessaires pour créer des dépôts.

Dans le menu de navigation, sous "CI/CD", accédez à Artifact Registry > Dépôts.

Cliquez sur Créer un dépôt.

Indiquez my-repository comme nom de dépôt.

Sélectionnez Docker comme format.

Sous "Type d'emplacement", sélectionnez Région, puis choisissez l'emplacement us-central1 (Iowa).

Cliquez sur Créer.

Configurer l'authentification Avant de pouvoir transférer ou extraire des images, vous devez configurer Docker afin qu'il se serve de Google Cloud CLI pour authentifier les requêtes envoyées à Artifact Registry.

Pour configurer l'authentification auprès des dépôts Docker dans la région us-central1, exécutez la commande suivante dans Cloud Shell :

```
gcloud auth configure-docker us-central1-docker.pkg.dev
```

Saisissez Y quand vous y êtes invité. La commande met à jour votre configuration Docker. Vous pouvez désormais vous connecter à Artifact Registry dans votre projet Google Cloud pour transférer et extraire des images.

Transférer le conteneur vers Artifact Registry Exécutez les commandes suivantes pour définir votre ID de projet et accéder au répertoire contenant votre Dockerfile.

```
export PROJECT_ID=$(gcloud config get-value project)
cd ~/test
```

Exécutez la commande pour ajouter le tag node-app:0.2.

```
docker build -t us-central1-docker.pkg.dev/$PROJECT_ID/my-repository/
    node-app:0.2 .
```

Exécutez la commande suivante pour vérifier les images Docker créées.

```
docker images
```

(Résultat de la commande)

```
REPOSITORY                        TAG         IMAGE ID          CREATED
node-app                          0.2         76b3beef845e      22 hours
us-central1-....node-app:0.2      0.2         76b3beef845e      22 hours
node-app                          0.1         f166cd2a9f10      26 hours
node                              lts         5a767079e3df      7 days
hello-world                       latest      1815c82652c0      7 weeks
```

Transférez l'image vers Artifact Registry.

```
docker push us-central1-docker.pkg.dev/$PROJECT_ID/my-repository/node-
    app:0.2
```

Résultat de la commande (celui que vous obtiendrez sera peut-être différent) :

```
The push refers to a repository [us-central1-docker.pkg.dev/[project-id
    ]/my-repository/node-app:0.2]
057029400a4a: Pushed
342f14cb7e2b: Pushed
903087566d45: Pushed
99dac0782a63: Pushed
e6695624484e: Pushed
da59b99bbd3b: Pushed
5616a6292c16: Pushed
f3ed6cb59ab0: Pushed
654f45ecb7e3: Pushed
2c40c66f7667: Pushed
```

```
12 0.2: digest: sha256:25
       b8ebd7820515609517ec38dbca9086e1abef3750c0d2aff7f341407c743c46 size:
        2419
```

Une fois la compilation terminée, dans le menu de navigation, sous “CI/CD”, accédez à Artifact Registry > Dépôts.

Cliquez sur my-repository. Le conteneur Docker node-app créé devrait s’afficher :

section node-app de artifact registry

Tester l’image

Vous pourriez démarrer une nouvelle VM, vous connecter en SSH à cette VM et installer gcloud. Pour plus de simplicité, vous allez juste supprimer l’ensemble des conteneurs et des images pour simuler un nouvel environnement.

Arrêtez et supprimez tous les conteneurs :

```
1 docker stop $(docker ps -q)
2 docker rm $(docker ps -aq)
```

Vous devez supprimer les images enfants (de node:lts) avant de supprimer l’image du nœud.

Exécutez la commande suivante pour supprimer toutes les images Docker.

```
1 docker rmi us-central1-docker.pkg.dev/$PROJECT_ID/my-repository/node-
     app:0.2
2 docker rmi node:lts
3 docker rmi -f $(docker images -aq) # remove remaining images
4 docker images
```

(Résultat de la commande)

```
1 REPOSITORY          TAG                 IMAGE ID            CREATED
                SIZE
```

À ce stade, vous devriez disposer d’un pseudo-nouvel environnement.

Extrayez l’image, puis exécutez-la.

```
1 docker pull us-central1-docker.pkg.dev/$PROJECT_ID/my-repository/node-
     app:0.2
2 docker run -p 4000:80 -d us-central1-docker.pkg.dev/$PROJECT_ID/my-
     repository/node-app:0.2
3 curl http://localhost:4000
```

(Résultat de la commande)

```
1 Welcome to Cloud
```