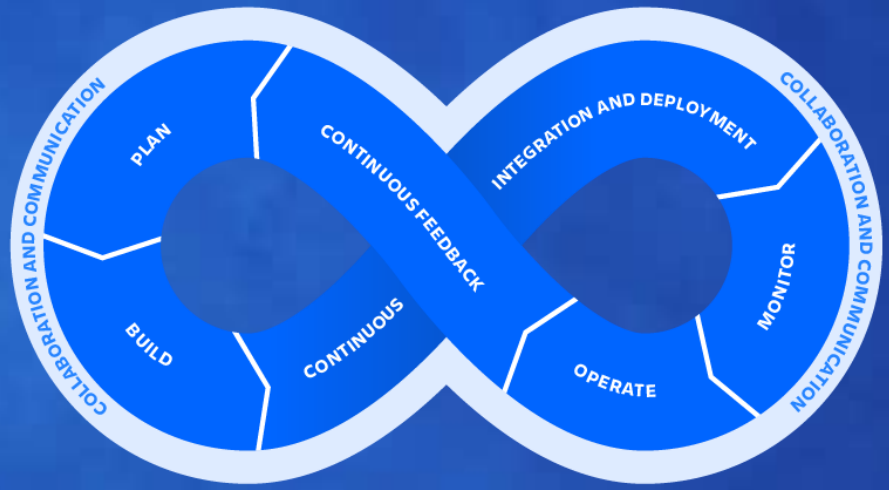


Session 12. Docker Hub

Ram N Sangwan

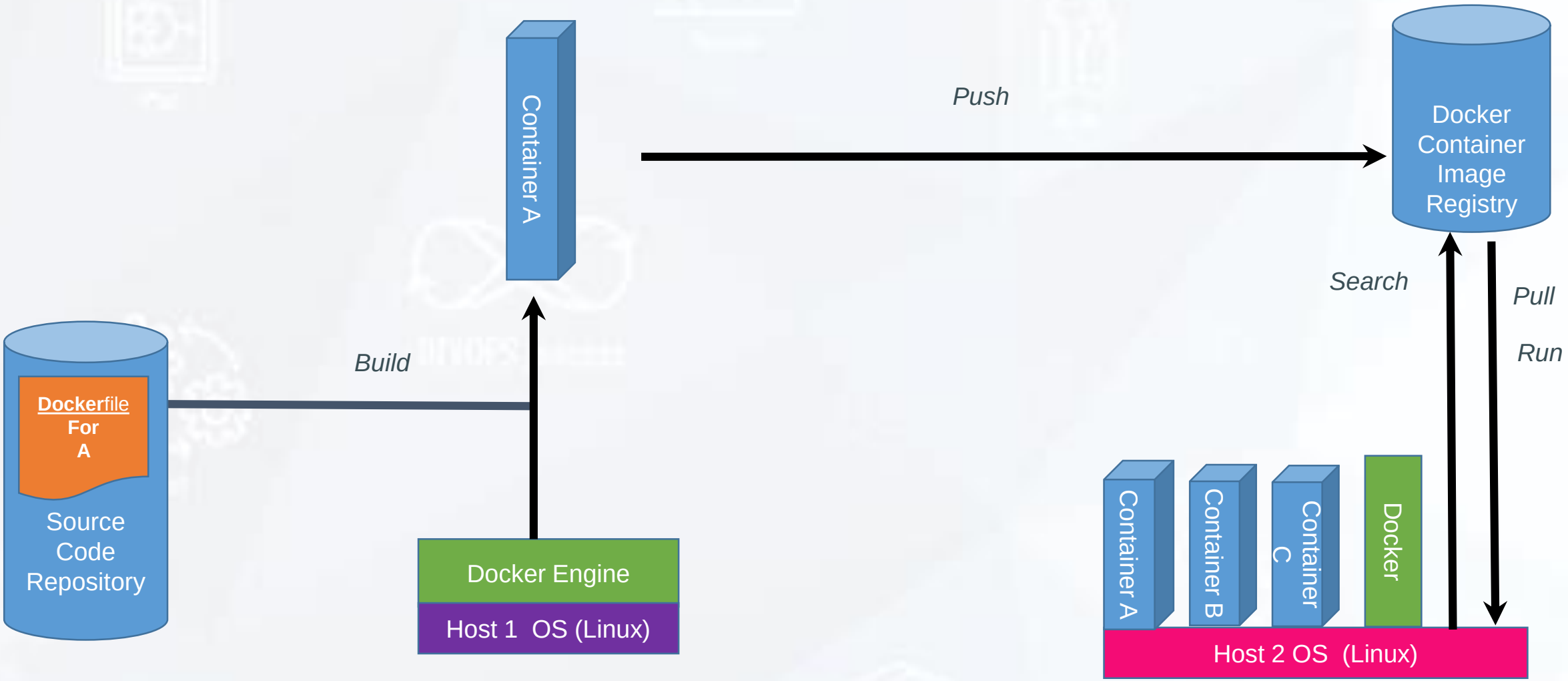
Agenda

- Downloading Docker images.
- Modifying Images
- Uploading images in Docker Registry
- Running multiple containers.

Docker Hub

- Docker Hub is a service provided by Docker for finding and sharing container images with your team. Features:
  - Repositories: Push and pull container images.
  - Teams & Organizations: Manage access to private repositories of container images.
  - Official Images: Pull and use high-quality container images provided by Docker.
  - Publisher Images: Pull and use high- quality container images provided by external vendors.
  - Builds: Automatically build container images from GitHub and Bitbucket and push them to Docker Hub.
  - Webhooks: Trigger actions after a successful push to a repository to integrate Docker Hub with other services.

Basics of Docker System



Docker File

- A text document that contains all the commands a user could call on the command line to assemble an image.
  - How to run commands
  - Add files or directories
  - Create environment variables
  - What process to run when launching container
- Result from building Dockerfile is Docker image

DOCKER FILE INSTRUCTIONS

The FROM instruction

- FROM is the first item in a Dockerfile. It specifies a parent image that your image will start from.

  FROM ubuntu:18.04

  . . .

- For instance, to use an image called tspdemo with the tag v1 from a sangwan account, you would use:

  FROM sangwan/tspdemo:v1

  . . .

- You can also use FROM scratch to indicate that your image should not have a parent image.
- The scratch image is a pseudo image that indicates that you want to start without a parent image.

The LABEL instruction

- LABEL provides the ability to add key-value metadata to the image.
- The LABEL instruction can be used many times within a single Dockerfile.

. . .

LABEL <key>=<value>

. . .

. . .

LABEL key1="value 1" key2="value 2"

. . .

The ADD instruction

- ADD is a way to copy file from the local filesystem, a remote URL, or a local compressed archive file to a location within the image filesystem.

  . . .

  ADD <source> <destination>

  . . .

- The source in this case can refer to:
  - A local file or directory within the build context
  - A remote URL

The COPY

- The COPY syntax mirrors that of ADD:

  . . .

  COPY <source> <destination>

  . . .

- The COPY command can also take a --from= argument for use in multi-stage builds.
- The COPY instruction looks incredibly similar to the ADD instruction at first glance, and overlaps in purpose.
- The difference is that COPY only works with local files and provides no automatic archive extraction.

ENV and EXPOSE instructions

- ENV is used to set an environmental variable during the build and run stages of the image.

  . . .

  ENV <key> <value>

  ENV <key>=<value>

  EXPOSE

- EXPOSE instruction communicates which ports the container's services are listening on.

  . . .

  EXPOSE <port>

  EXPOSE <port>/tcp

  EXPOSE <port>/udp

  . . .

The RUN instruction

- RUN will execute the statements given to it within the image's filesystem environment during the build process.
- This is the primary mechanism for making changes to build your image.
- It can run any commands available within the image's filesystem.

. . .

RUN <command>

RUN ["<executable", ..., "argn"]

. . .

The USER instruction

- USER controls what user, and optionally group, commands are executed as in the image environment.
- This can affect both the build and runtime process as every subsequent command that is run in the container environment (RUN, CMD, and ENTRYPOINT) will be executed by the provided user.

  USER <user_name_or_ID>:<group_name_or_ID>

The VOLUME instruction

- VOLUME is responsible for creating mount points within the container image's filesystem for mounting external volumes from the host or other locations.
- The syntax of VOLUME instruction can take a series of strings separated by spaces or a JSON array.
- For instance, these two will produce the same result:

  VOLUME ["/data/vol1", "/data/vol2"]

  VOLUME /data/vol1 /data/vol2

The WORKDIR instruction

- WORKDIR declares the directory context for instructions like RUN, CMD, ENTRYPOINT, COPY, and ADD.
- It specifies the location on the filesystem where these commands should executed.
- The WORKDIR instruction will automatically create the directory and any parent directories necessary.
- To declare a WORKDIR, you can use :

  WORKDIR <filesystem_path>

The CMD and ENTRYPOINT instruction

- CMD provides the default execution instructions for when a container is run from the image.
- This is the primary way to specify what should happen when a user executes docker run on your image.

  CMD ls -al /var/log | wc -l

  CMD ["full/path/to/executable", "param1", "param2"]

  ENTRYPOINT ["/path/to/entrypoint"]

  CMD ["param1", "param2"]
- ENTRYPOINT allows you to configure a container that can be run as an executable by default.

  ENTRYPOINT ls -al

Build & Push an image to Docker Hub

Build & Push an image to Docker Hub

- We are going to run a Busybox container on our system and get a taste of the docker run command.
- BusyBox is a software suite that provides several Unix utilities in a single executable file.
- Pull the busybox image on your system

  # docker pull busybox

```
[root@server ~]# docker pull busybox
Using default tag: latest
Trying to pull repository docker.io/library/busybox ...
latest: Pulling from docker.io/library/busybox
91f30d776fb2: Pull complete
Digest: sha256:9ddee63a712cea977267342e8750ecbc60d3aab25f04ceacfa795e6fce341793
Status: Downloaded newer image for busybox:latest
[root@server ~]#
```

Create a Dockerfile

- Start by creating a Dockerfile to specify your application as shown below:

  # cat > Dockerfile <<EOF

  FROM busybox

  CMD echo "Hello world! This is my first Docker image. The SkillPedia"

  EOF

```
[root@server ~]# cat > Dockerfile <<EOF
> FROM busybox
> CMD echo "Hello world! This is my first Docker image. The SkillPedia"
> EOF
[root@server ~]#
```

Build your Docker Image

- Now Run the following command to build your docker image

  # docker build -t <your_username>/oci:mytag .

```
[root@server ~]# docker build -t sangwan70/oci:mytag .
Sending build context to Docker daemon  148.6MB
Step 1/2 : FROM busybox
 ---> c7c37e472d31
Step 2/2 : CMD echo "Hello world! This is my first Docker image. The SkillPedia"
 ---> Running in 6a6b9d252f08
Removing intermediate container 6a6b9d252f08
 ---> feca6d2d237a
Successfully built feca6d2d237a
Successfully tagged sangwan70/oci:mytag
[root@server ~]#
```

Test Your Docker Image

- Test your docker image locally by running

  # docker run <your_username>/oci:mytag

```
[root@server ~]# docker run sangwan70/oci:mytag
Hello world! This is my first Docker image. The SkillPedia
[root@server ~]#
```

Tag and Push Your Image

- To push your image, first log into Docker Hub:

  # docker login -u docker-registry-username

- Tag your Docker image to push to Docker Hub.

  # docker tag hello-world <your_username>/oci:mytag

- Run the following command to to push your Docker image to Docker Hub.

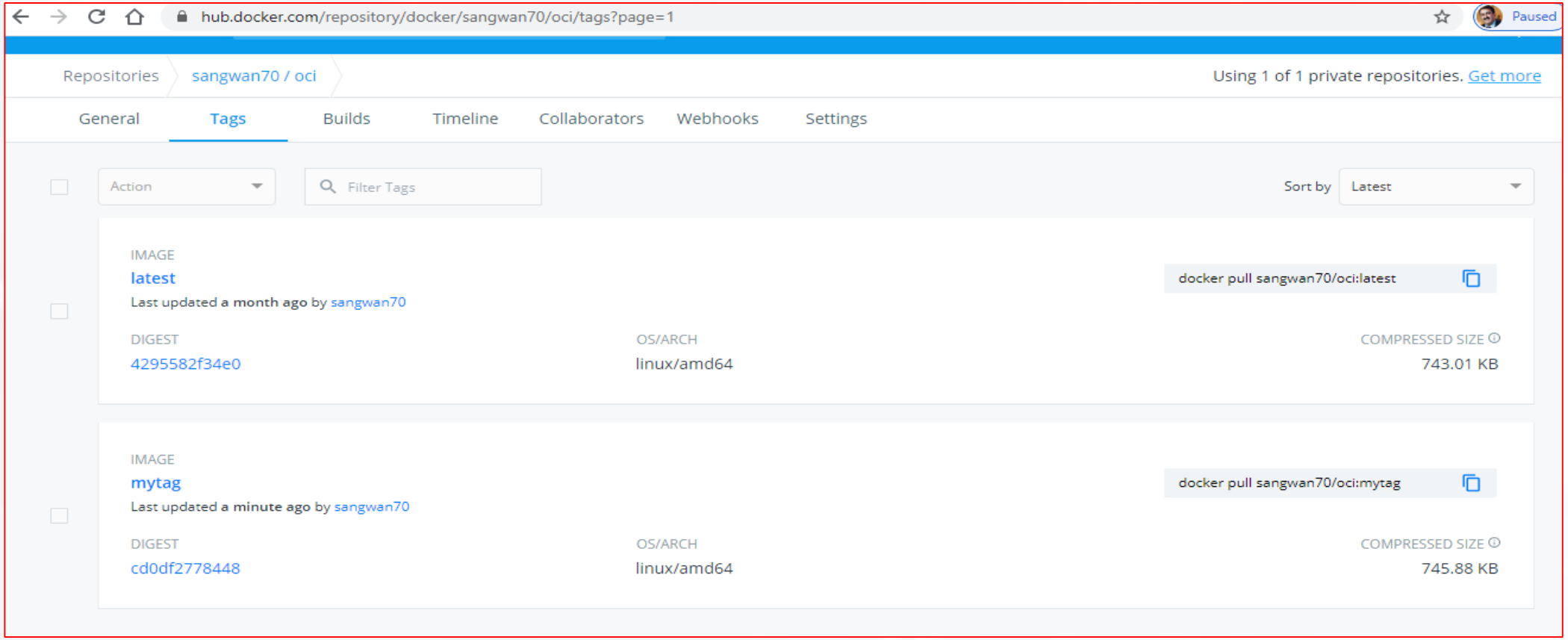
  # docker push <your_username>/oci:mytag

- You should see output similar to:

```
[root@server ~]# docker push sangwan70/oci:mytag
The push refers to repository [docker.io/sangwan70/oci]
50761fe126b6: Mounted from library/busybox
mytag: digest: sha256:cd0df2778448405bbf1e38ef8b8e66dc8948be3ef3e23798df2ff6e5173b2041 size: 527
[root@server ~]#
```

Verify on Docker Hub

- And in Docker Hub, your repository should have a new latest tag available under Tags:



Thank You