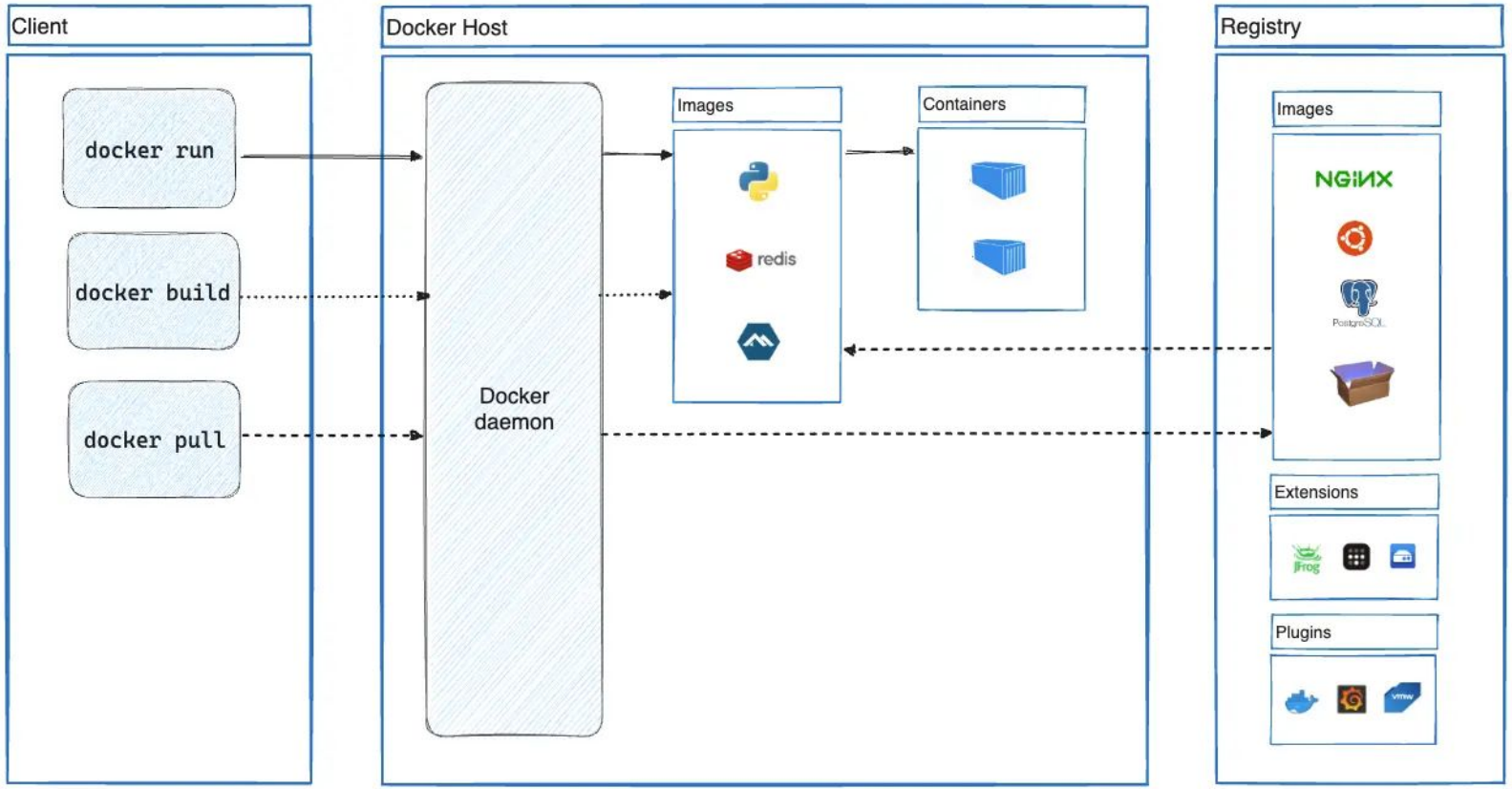
Web Server Design

Lecture 3 – Docker Basics

Old Dominion University
Department of Computer Science
CS 431/531 Spring 2026
David Calano <dcala005@odu.edu>

Docker



https://docs.docker.com/get-started/docker-overview/

Installation (Engine)

1. Set up Docker's `apt` repository.

```
# Add Docker's official GPG key:
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/ubuntu
Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update
```

2. Install the Docker packages.

Latest | Specific version

To install the latest version, run:

```
$ sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-pl
```

https://docs.docker.com/engine/install/ubuntu/

The always useful -h / –help

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker --help
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  bake        Build from a file
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Authenticate to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  system      Manage Docker
  volume      Manage volumes

Swarm Commands:
  swarm       Manage Swarm
```

docker pull <image:tag>

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker pull python:latest
latest: Pulling from library/python
Digest: sha256:4b827abf32c14b7df9a0dc5199c2f0bc46e2c9862cd5d77eddae8a2cd8460f60
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

As the name implies, docker pull will grab an image from an online repository. unless a full URI is specified, images are pulled from Docker hub.

docker image ls

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image ls

IMAGE           ID             DISK USAGE   CONTENT SIZE   EXTRA
python:latest   4b827abf32c1       1.63GB          432MB
```

Similar to the Linux command ls, docker image ls will display the currently available local images

dockerfile example

```
1   FROM python:latest (last pushed 8 hours ago)
2
3   # RUN <command>
4
5   # WORKDIR /app
6
7   COPY index.html /
8
9   EXPOSE 7000
10
11  CMD ["python", "-m", "http.server", "7000", "--bind", "0.0.0.0"]
12
```

Applications need to listen at 0.0.0.0, as 127.0.0.1 here is the container's localhost and no outside traffic would be visible

Common dockerfile commands

FROM <image:tag>: utilizes a local image as a base platform to build from

RUN <cmd>: executes a system command during the image building phase

COPY <source> <target>: copies files from the host system (source) to the target directory within the container

EXPOSE <port>: exposes a port for use with the built container

CMD <cmd>: executes a system command when the container is run

https://docs.docker.com/reference/dockerfile/

docker build <target_dir>

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker build .
[+] Building 0.1s (7/7) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 189B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 31B
 => [1/2] FROM docker.io/library/python:latest@sha256:4b827abf32c14b7df9a0dc5199c2f0bc46e2c9862cd5d77eddae8a2cd8460f60
 => => resolve docker.io/library/python:latest@sha256:4b827abf32c14b7df9a0dc5199c2f0bc46e2c9862cd5d77eddae8a2cd8460f60
 => CACHED [2/2] COPY index.html /
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:6817eee0d507c0957e0255b6e46c6192095282e9b0fd08723ce4d217f6f75bac
 => => exporting config sha256:e1384afd13adf14fd825251ece0695d817938c3ceb55b789bee04bbde8209369
 => => exporting attestation manifest sha256:c2879ca7638f7bc8c6c90f109ddc8fa1608f723d540a2f59959a9d5b470b1177
 => => exporting manifest list sha256:1575968a02c2ec84b9c7b90e3c4d3a49f7907c2f07cb44a7c481ad0d37b41aac
 => => naming to moby-dangling@sha256:1575968a02c2ec84b9c7b90e3c4d3a49f7907c2f07cb44a7c481ad0d37b41aac
 => => unpacking to moby-dangling@sha256:1575968a02c2ec84b9c7b90e3c4d3a49f7907c2f07cb44a7c481ad0d37b41aac
```

Dockerfile build layers

Each command in a docker file represents a “layer”. Layers are cached such that subsequent builds can be sped up if layers are unchanged. When a layer is changed in some manner, each layer which follows it will need to be rebuilt.

For simple builds, this is not much of an issue. But with larger projects and multistage builds optimizing the order of commands and build layers which do not frequently change can drastically speed up build times.

Untagged images

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image ls

IMAGE           ID             DISK USAGE   CONTENT SIZE   EXTRA
python:latest   4b827abf32c1       1.63GB          432MB

../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image ls -a

IMAGE           ID             DISK USAGE   CONTENT SIZE   EXTRA
python:latest   4b827abf32c1       1.63GB          432MB
<untagged>      bec0b3100067       1.61GB          414MB
<untagged>      03e0d87023e4       1.61GB          414MB
<untagged>      e9e23afcc513       1.61GB          414MB
<untagged>      86c16a0e2426       1.61GB          414MB
<untagged>      f8cfda993ca0       1.61GB          414MB
<untagged>      1575968a02c2       1.61GB          414MB
```

Currently we built an “untagged” (or unnamed image). We can use docker image ls -a to see these usually hidden images

Deleting images

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image ls -a

IMAGE           ID             DISK USAGE   CONTENT SIZE   EXTRA
python:latest   4b827abf32c1       1.63GB          432MB
<untagged>      bec0b3100067       1.61GB          414MB
<untagged>      f8cfda993ca0       1.61GB          414MB
<untagged>      e9e23afcc513       1.61GB          414MB
<untagged>      86c16a0e2426       1.61GB          414MB
<untagged>      03e0d87023e4       1.61GB          414MB
<untagged>      1575968a02c2       1.61GB          414MB

../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image rm bec0b3100067
Deleted: sha256:bec0b310006737be89f97e953e2fe6901b046b94d02e4b45e8ae6000e72c5dc1

../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker rmi f8cfda993ca0
Deleted: sha256:f8cfda993ca046e6dcdf0d0f6428c9817b47ca2a82ebba3bfc8ac467d35a610a
```

Images can be removed using the docker image rm <id> command, or the docker rmi <id> shortcut.

Pruning images

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: sha256:03e0d87023e44c86a659b5c5ff6e7e679d28f768939551c91b32b73e965d6870
deleted: sha256:03e0d87023e44c86a659b5c5ff6e7e679d28f768939551c91b32b73e965d6870
deleted: sha256:32211990e0d123cac0cab7650646fc9f82499eeca581828126188b6b862c4b58
untagged: sha256:1575968a02c2ec84b9c7b90e3c4d3a49f7907c2f07cb44a7c481ad0d37b41aac
deleted: sha256:1575968a02c2ec84b9c7b90e3c4d3a49f7907c2f07cb44a7c481ad0d37b41aac
deleted: sha256:6817eee0d507c0957e0255b6e46c6192095282e9b0fd08723ce4d217f6f75bac
deleted: sha256:c2879ca7638f7bc8c6c90f109ddc8fa1608f723d540a2f59959a9d5b470b1177
untagged: sha256:86c16a0e242684e51e3349060f9643ffcb0eb862837a195f2564beae49e3d397
deleted: sha256:86c16a0e242684e51e3349060f9643ffcb0eb862837a195f2564beae49e3d397
deleted: sha256:121e4205edce1387d3cfc0c6846099be57163188abfe506b2a08003760bfe8ec
deleted: sha256:bce8721f6d7a5ea9cf88c8b1742db5dac31f207fd415ff1266944cf0dbe8cccf
untagged: sha256:e9e23afcc513348ad9491cc63f67e93fe391e72d71359728bb065f4d1083a2cf
deleted: sha256:e9e23afcc513348ad9491cc63f67e93fe391e72d71359728bb065f4d1083a2cf
deleted: sha256:027c2ad77cb3b7f4a23f3387939ea19432ec6421a28af6eea5a7578fb2d05e49
deleted: sha256:85fd00ac4e32867c2db48f530db2506bc9822e5bf8f81bc9c5472a4b1111592f

Total reclaimed space: 37.33kB
```

Multiple unused images can be cleaned from your system using the docker image prune command.

Tagging images

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker build -t my_server_image .
[+] Building 0.2s (7/7) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 189B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 31B
 => [1/2] FROM docker.io/library/python:latest@sha256:4b827abf32c14b7df9a0dc5199c2f0bc46e2c9862cd5d77eddae8a2cd8460f60
 => => resolve docker.io/library/python:latest@sha256:4b827abf32c14b7df9a0dc5199c2f0bc46e2c9862cd5d77eddae8a2cd8460f60
 => CACHED [2/2] COPY index.html /
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:6817eee0d507c0957e0255b6e46c6192095282e9b0fd08723ce4d217f6f75bac
 => => exporting config sha256:e1384afd13adf14fd825251ece0695d817938c3ceb55b789bee04bbde8209369
 => => exporting attestation manifest sha256:c54cef603d256d55b7383ec3807eb61a9bf200ceb067225b67c5fa33daa863b0
 => => exporting manifest list sha256:ecf331759d9af866f356ec0a15eb403bb7bd45f7d526a5b4e72619be05c6ed82
 => => naming to docker.io/library/my_server_image:latest
 => => unpacking to docker.io/library/my_server_image:latest

../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker image ls

IMAGE                  ID             DISK USAGE   CONTENT SIZE   EXTRA
my_server_image:latest   ecf331759d9a       1.61GB          414MB
python:latest            4b827abf32c1       1.63GB          432MB
```

Images can be tagged during building with the -t <tag> option to make them easier to work with.

Running a built image

```
../cs531-s26.github.io/tutorials/docker on  main [?]
→ docker run -p 7000:7000 --name my_http_server my_server_image
```

-p <HOST_PORT>:<CONTAINER_PORT>
specifies a port mapping from the host system to an container's exposed port

–name <container_name>
similar to tagging an image when building, specifying a container name upon instantiation can make it easier to work with later on rather than always using the container's ID

This container would now be running in the foreground and will need to be stopped using Ctrl + C on Linux. To run it in the background you would add a -d flag to the command (docker run -d …).

Checking running containers

```
../cs531-s26.github.io/tutorials/docker on  main [?]
➜ docker run -d -p 7000:7000 --name my_http_server my_server_image
15a36d2d5c1af049e64a3996a4aad48cda2fc969888a97547440c49231617e45

../cs531-s26.github.io/tutorials/docker on  main [?]
➜ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORTS                                          NAMES
15a36d2d5c1a   my_server_image   "python -m http.serv…"   5 seconds ago   Up 5 seconds   0.0.0.0:7000->7000/tcp, [::]:7000->7000/tcp   my_http_server
```

To check the status of a running container you can use the docker ps command, or docker ps -a to view both active and inactive containers.

When containers are running in the background, they will not actively display logs. To see a container's logs (assuming you application is programmed to output to stdout), you can use the command docker logs <container_id_or_name>.

Stopping and removing containers

```
../cs531-s26.github.io/tutorials/docker on  main [?]
➜ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORTS                                       NAMES
15a36d2d5c1a   my_server_image   "python -m http.serv…"   4 minutes ago   Up 4 minutes   0.0.0.0:7000->7000/tcp, [::]:7000->7000/tcp   my_http_server

../cs531-s26.github.io/tutorials/docker on  main [?]
➜ docker stop my_http_server
my_http_server

../cs531-s26.github.io/tutorials/docker on  main [?] took 10.2s
➜ docker rm my_http_server
my_http_server

../cs531-s26.github.io/tutorials/docker on  main [?]
➜ docker rmi my_server_image
Untagged: my_server_image:latest
Deleted: sha256:ecf331759d9af866f356ec0a15eb403bb7bd45f7d526a5b4e72619be05c6ed82
```

Containers can be stopped using docker stop <container_name_or_id>. Once stopped, they can then be deleted using docker rm <container_name_or_id>.

* Note, you cannot remove a docker image if it has an associated container. To remove a docker image, you must first remove all docker containers utilizing that image.