


```

zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker build -t lightweight-app .
[sudo] password for zaphira:
[+] Building 2.0s (5/5) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 116B                             0.0s
=> [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine 1.6s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> CACHED [1/1] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df3340 0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:41c6439c605eb2c21e573f26a77b5798af1ad3c4ef4f425031a544dd7 0.0s
=> => naming to docker.io/library/lightweight-app              0.0s
zaphira@zaphira-VirtualBox:~/dockerProj$

```

After the build process completed, the size of the newly created image was compared with the original openjdk:8 image using the following commands:

sudo docker images | grep lightweight-app
Sudo docker images

```

zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker images | grep lightweight-app
lightweight-app latest 41c6439c605e 5 years ago 105MB
zaphira@zaphira-VirtualBox:~/dockerProj$
zaphira@zaphira-VirtualBox:~/dockerProj$
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
openjdk              8               b273004037cc   2 years ago    526MB
lightweight-app      latest          41c6439c605e   5 years ago    105MB
zaphira@zaphira-VirtualBox:~/dockerProj$

```

Comparison Results:

The size of the original openjdk:8 image is 526MB.

The size of the newly created lightweight-app image using Alpine is 105MB.

Step 4: Explanation of Size Difference and Benefits of a Lightweight Base Image

The Alpine-based OpenJDK image is significantly smaller than the standard OpenJDK image. The difference in size arises due to several key factors:

Minimal Base Image: Alpine Linux is designed to be minimal, with a smaller footprint compared to Debian-based distributions. It excludes unnecessary packages and libraries that would typically be included in a more comprehensive base image.

Fewer Dependencies: Alpine reduces the number of default dependencies, which not only saves space but also reduces the attack surface by minimizing the number of potential vulnerabilities.

Faster Deployment: A smaller image means faster build times and quicker deployment, which is particularly beneficial in continuous integration/continuous deployment (CI/CD) pipelines.

Security Benefits: A smaller base image means fewer components to secure. Alpine's security-focused design helps minimize risks by including only the essential libraries needed for running the application.

The benefits of using a lightweight base image like Alpine are:

1. Reduced attack surface due to fewer installed components and libraries.
2. Improved performance due to smaller image sizes.
3. Faster deployment and easier maintenance.

Step 5: Launch the “whoami” Command and Check Permissions and Security

To check the current user and their permissions within the container, the following command was executed:

```
sudo docker run dit --name mycontainer lightweight-app
sudo docker exec -it mycontainer /bin/sh
```

```
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker run -dit --name mycontainer lightweight-app
[sudo] password for zaphira:
1cbf66ac6fe73f06323040f403f8b544ffa1aa45a40ec9b093b397ae2561abf7
zaphira@zaphira-VirtualBox:~/dockerProj$ docker ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.47/containers/json": dial unix /var/run/docker.sock: connect: permission denied
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
1cbf66ac6fe7   lightweight-app  "/bin/sh"               31 seconds ago Up 29 seconds  8080/tcp       mycontainer
zaphira@zaphira-VirtualBox:~/dockerProj$
zaphira@zaphira-VirtualBox:~/dockerProj$
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker exec -it mycontainer /bin/sh
/ # whoami
root
/ #
```

The first command runs a container from the image created while the second command opens the terminal inside the container so we can execute the **whoami** command.

This returned the username of the user running inside the container, which is root. This is not secured. Running containers as the root user can pose a security risk, as it grants full privileges to the application, potentially allowing it to perform malicious actions if the container is compromised. To secure with best practices, we create a non-root user within the Dockerfile and run the application under that user's context. It's advisable to run applications inside Docker containers as non-root users, unless absolutely necessary. This is security best practice and ensures that the application runs with limited permissions, reducing the potential damage in the event of a security breach.

Summary: By replacing the openjdk:8 image with the openjdk:8-jdk-alpine image, we reduced the image size from 488MB to 120MB. This reduction in size not only optimizes performance but also enhances security by minimizing the attack surface of the container. Additionally, using a non-root user for running applications inside the container would further improve security by limiting the permissions available to potentially compromised processes.

Adopting lightweight base images like Alpine is an effective strategy for building more secure, efficient, and optimized Docker containers.

Objective 2: Update base image and run it as a new user

Objective: Update the image and run it as a nonroot user

The objective of this exercise is to update and upgrade the base image and create a non-root user to enhance security by limiting permissions.

We are using the base image 'open-jdk:8-jdk-alpine', the lightweight image. To update and upgrade it, we include this in the dockerfile:

```
FROM openjdk:8-jdk-alpine
RUN apk update && apk upgrade
EXPOSE 8080
```

```
GNU nano 7.2 Dockerfile
To use the alpine-based openJDK image

FROM openjdk:8-jdk-alpine
RUN apk update && apk upgrade
EXPOSE 8080
```

- apk update: gets the latest list of available packages and versions from the Alpine repositories.
- apk upgrade: installs the downloaded packages.

Updating and upgrading the system ensures that it runs with the latest security patches and updated libraries.

The updated dockerfile was used to successfully rebuild the image using the command:

sudo docker build -t updated_lightweight-app .

```
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo nano Dockerfile
[sudo] password for zaphira:
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker build -t updated_lightweight-app .
[+] Building 39.9s (6/6) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.1s
=> => transferring dockerfile: 145B                             0.0s
=> [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine 1.7s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> CACHED [1/2] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df33402f201713f932b58cb 0.0s
=> [2/2] RUN apk update && apk upgrade                          37.2s
=> exporting to image                                           0.7s
=> => exporting layers                                           0.6s
=> => writing image sha256:ed600177cd97adbf5c58b6e0755b67a1bd9800290904c611581416be2c53b82a 0.0s
=> => naming to docker.io/library/updated_lightweight-app      0.0s
zaphira@zaphira-VirtualBox:~/dockerProj$
```

The command below was used to verify the updates:

sudo docker run --rm updated-lightweight-app apk version

The apk version command was used to display the updated version of the alpine package. The output of the command confirms the updated apk version showing that the update was successful.

```
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker run --rm updated_lightweight-app apk version
Installed:                               Available:
openjdk8-8.212.04-r0                    < 8.275.01-r0
zaphira@zaphira-VirtualBox:~/dockerProj$
```

The size of the updated image was compared to the original base image using **sudo docker images**

```
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker images
REPOSITORY          TAG         IMAGE ID      CREATED       SIZE
updated_lightweight-app latest      ed600177cd97 13 minutes ago 201MB
openjdk              8          b273004037cc 2 years ago   526MB
lightweight-app      latest     41c6439c605e 5 years ago   105MB
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker images | grep lightweight-app
updated_lightweight-app latest     ed600177cd97 13 minutes ago 201MB
lightweight-app        latest     41c6439c605e 5 years ago   105MB
zaphira@zaphira-VirtualBox:~/dockerProj$
```

From above screenshot, the size of the updated image was slightly larger due to the package upgrades.

Creating a non-root user:

To create a non-root user, the following lines were added to the dockerfile:

RUN addgroup -S appgroup && adduser -S appuser -G appgroup

USER appuser

- **addgroup -S appgroup** creates a system group named appgroup.
- **adduser -S appuser -G appgroup** creates a system user named appuser and assigns it to the appgroup.
- **USER appuser** sets the default user for subsequent instructions and container runtime to appuser

```
GNU nano 7.2 Dockerfile
#To use the alpine-based openJDK image

FROM openjdk:8-jdk-alpine
RUN apk update && apk upgrade
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
USER appuser
EXPOSE 8080
```

The dockerfile with the non-root configuration was used to build a new image:

sudo docker build -t non-root-app .


```

zaphira@zaphira-VirtualBox: ~/dockerProj$ sudo docker build -t non_root_app .
[+] Building 2.6s (7/7) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 219B                             0.0s
=> [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine 1.7s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> [1/3] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df33402f201713f932b58cb9de94a0 0.0s
=> CACHED [2/3] RUN apk update && apk upgrade                   0.0s
=> [3/3] RUN addgroup -S appgroup && adduser -S appuser -G appgroup 0.5s
=> exporting to image                                           0.1s
=> => exporting layers                                           0.1s
=> writing image sha256:345ba77ce5db2f49a4176ce2d3658cb4ba77297eeea3e01921f6c57580125485 0.0s
=> => naming to docker.io/library/non_root_app                 0.0s
zaphira@zaphira-VirtualBox: ~/dockerProj$ sudo nano Dockerfile

```

The container was run interactively using **sudo docker run -it non-root-app sh**.

- **sh** opens a shell session insider the container.

- **whoami** was used to confirm the active user was appuser.

Inside the container, the command was used to try to create a directory in a restricted location: **mkdir /restricted_dir**

```

zaphira@zaphira-VirtualBox:~$ sudo docker run -it non-root-app sh
/ $ whoami
appuser
/ $ mkdir /restricted-dir
mkdir: can't create directory '/restricted-dir': Permission denied
/ $
/ $ apk update && apk upgrade
sh: app: not found
/ $ apk update && apk upgrade
ERROR: Unable to lock database: Permission denied
ERROR: Failed to open apk database: Permission denied
/ $

```

The attempt to create a new directory failed with “permission denied” as the appuser did not have root privileges.

Also, apk update && apk upgrade was run to test update and upgrade of the system inside the container. This too failed with permission denied.

Objective 3: Using multi-stage builds in a CI/CD pipeline

Objective: To reduce the size of the Docker image using multi-stage builds, thereby minimizing security vulnerabilities and improving efficiency.

Step 1: Update the dockerfile with the content below:

```

#To use the alpine-based openJDK image

FROM openjdk:8-alpine AS build
WORKDIR /app
COPY . .
RUN javac Main.java

FROM openjdk:8-jdk-alpine
WORKDIR /app
COPY --from=build /app/Main.class .
CMD ["java", "Main"]

```

Stage 1 (Build):

FROM openjdk:8-alpine AS build uses a lightweight Java 8 JDK image for compiling Java code.

WORKDIR /app sets /app as the working directory.

COPY . copies all files from the build context into the container.

RUN javac Main.java compiles the Java source code (Main.java) into a bytecode class file (Main.class).

Stage 2 (Runtime):

FROM openjdk:8-alpine starts a fresh, lightweight Java runtime-only image.

WORKDIR /app sets /app as the working directory.

COPY --from=build /app/Main.class copies only the compiled class file from the build stage.

CMD ["java", "Main"] specifies the command to run the Java application.

Multi-stage builds separate the build environment from the runtime environment, ensuring only essential artifacts are included in the final image. The image was built using the command as seen below:

```
zaphira@zaphira-VirtualBox:~/dockerProj$ nano Main.java
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker build -t multi_stage_app .
[+] Building 3.5s (13/13) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 249B                                0.0s
=> [internal] load metadata for docker.io/library/openjdk:8-jdk-alpine 0.7s
=> [internal] load metadata for docker.io/library/openjdk:8-alpine 0.7s
=> [internal] load .dockerignore                                   0.1s
=> => transferring context: 2B                                       0.0s
=> [build 1/4] FROM docker.io/library/openjdk:8-alpine@sha256:94792824df2df33402f201713f932b56cb9de94 0.0s
=> [stage-1 1/3] FROM docker.io/library/openjdk:8-jdk-alpine@sha256:94792824df2df33402f201713f932b56cb9de94 0.0s
=> CACHED [stage-1 2/3] WORKDIR /app                               0.0s
=> [internal] load build context                                   0.1s
=> => transferring context: 106B                                      0.0s
=> CACHED [build 2/4] WORKDIR /app                                 0.0s
=> [build 3/4] COPY . .                                           0.2s
=> [build 4/4] RUN javac Main.java                                1.4s
=> [stage-1 3/3] COPY --from=build /app/Main.class .             0.2s
=> exporting to image                                             0.4s
=> => exporting layers                                              0.2s
=> => writing image sha256:b83249db68824f4b3a5f562409648d2bfff8536d66b34492e4eecd1467c1580b7 0.0s
=> => naming to docker.io/library/multi_stage_app                 0.1s
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo nano Dockerfile
zaphira@zaphira-VirtualBox:~/dockerProj$
```

The size of the multi-stage build image 'multi_stage_app' was compared with the single-stage build image 'updated_lightweight-app' as below:

```
zaphira@zaphira-VirtualBox:~/dockerProj$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
multi_stage_app     latest         b83249db6882   2 minutes ago  105MB
non_root_app        latest         345ba77ce5db   24 hours ago   201MB
updated_lightweight-app latest         ed600177cd97   24 hours ago   201MB
openjdk             8              b273004037cc   2 years ago    526MB
lightweight-app     latest         41c6439c605e   5 years ago    105MB
zaphira@zaphira-VirtualBox:~/dockerProj$
```

The multi-stage build reduced the image size by excluding the java development kit and source files, leaving only the runtime environment and compiled class files (as specified in the dockerfile). The smaller image minimizes attack surfaces. This approach adheres to best practices in Docker image optimization, enhancing security and reducing the risk of vulnerabilities.

Objective 4: Restrict the CPU and memory available to the container

The focus here is to setup resource quotas to restrict the use of CPU and memory by docker containers. The container was started with specific CPU and memory restrictions using the command:

sudo docker run -it --memory="512m" --cpus="1.5" multi-stage-app

--memory="512m" limits the container's memory usage to a maximum of 512MB.

--cpus="1.5" restricts the container's CPU usage to 1.5.

To monitor the resource usage, use the command:

sudo docker stats

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIOS
1cbf66ac6fe7	mycontainer	0.00%	448KiB / 3.822GiB	0.01%	22.5kB / 0B	0B / 135kB	1

Above is the current usage of resources. These settings ensure that the container does not consume excessive resources while protecting the host machine and other containers.

Exercise 5: Scan docker images for vulnerabilities with Docker Clair

The focus of this exercise is to use Clair, an open-vulnerability scanning tool, to analyse docker images for vulnerabilities. Doing this would help identify and proactively address the weaknesses identified and make it more secure.

Step 1: Download Clair's Docker Compose setup from its GitHub repository:

```
sudo git clone https://github.com/quay/clair.git  
cd clair
```

Step2: Use Docker Compose to start Clair: Ensure docker compose is installed.

```
docker-compose up -d
```

This command above would start clair in detached mode.

1. Install Clair client avec clientctl:

```
wget https://github.com/quay/clair/releases/latest/download/clairctl-linux-amd64
chmod +x clairctl-linux-amd64
sudo mv clairctl-linux-amd64 /usr/local/bin/clairctl
```

2. Push Your Docker Image to a Registry: Clair requires images to be in a Docker registry. Push your image to Docker Hub or a private registry with the block below:

```
sudo docker tag lightweight-app mydockername/lightweight-app
sudo docker push mydockername/lightweight-app
```

Login to docker using: `sudo docker login -u mydockername` for the above commands to execute successfully.

```
zaphira@zaphira-VirtualBox:~$ sudo docker tag lightweight-app zaphira/lightweight-app
zaphira@zaphira-VirtualBox:~$ sudo docker push zaphira/lightweight-app
Using default tag: latest
The push refers to repository [docker.io/zaphira/lightweight-app]
ceaf9e1ebef5: Mounted from library/openjdk
9b9b7f3d56a0: Mounted from library/openjdk
f1b5933fe4b5: Mounted from library/openjdk
latest: digest: sha256:5f384dc06851e697b102ccea8decefe1e9a968e7eaf4c8cf9e97738984b7cd32 size: 947
zaphira@zaphira-VirtualBox:~$
```

```
zaphira@zaphira-VirtualBox:~$ sudo docker tag lightweight-app zaphira/lightweight-app
zaphira@zaphira-VirtualBox:~$ sudo docker push zaphira/lightweight-app
Using default tag: latest
The push refers to repository [docker.io/zaphira/lightweight-app]
ceaf9e1ebef5: Layer already exists
9b9b7f3d56a0: Layer already exists
f1b5933fe4b5: Layer already exists
latest: digest: sha256:5f384dc06851e697b102ccea8decefe1e9a968e7eaf4c8cf9e97738984b7cd32 size: 947
zaphira@zaphira-VirtualBox:~$ clairctl --config clair-config.yaml report zaphira/lightweight-app
2025-01-15T21:55:46+01:00 ERR error="Get \"http://localhost:6060/indexer/apl/v1/index_report/sha256:5f384dc06851e697b102ccea8decefe1e9a968e7eaf4c8cf9e97738984b7cd32\": dial tcp 127.0.0.1:6060: connect: connection refused"
zaphira@zaphira-VirtualBox:~$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
826213e64efd	quay.io/projectquay/clair:latest	"cat test.txt"	18 minutes ago	Exited (0) 17 minutes ago		clair
089efafedd5e	postgres:latest	"docker-entrypoint.s..."	18 minutes ago	Up 18 minutes	5432/tcp	clair-db
856131e798d5	multi-stage-app	"java Main"	About an hour ago	Exited (0) About an hour ago		epic_liskov
78115c66745f	non-root-app	"sh"	4 hours ago	Exited (99) 2 hours ago		brave_chandrase

```
zaphira@zaphira-VirtualBox:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED       STATUS       PORTS       NAMES
089efafedd5e   postgres:latest   "docker-entrypoint.s..."  18 minutes ago   Up 18 minutes   5432/tcp    clair-db
zaphira@zaphira-VirtualBox:~$ sudo docker start 82621
82621
zaphira@zaphira-VirtualBox:~$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
826213e64efd	quay.io/projectquay/clair:latest	"cat test.txt"	20 minutes ago	Exited (0) 7 seconds ago		clair
089efafedd5e	postgres:latest	"docker-entrypoint.s..."	20 minutes ago	Up 20 minutes	5432/tcp	clair-db
856131e798d5	multi-stage-app	"java Main"	About an hour ago	Exited (0) About an hour ago		epic_liskov
78115c66745f	non-root-app	"sh"	4 hours ago	Exited (99) 2 hours ago		brave_chandrase

```
zaphira@zaphira-VirtualBox:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED       STATUS       PORTS       NAMES
826213e64efd   quay.io/projectquay/clair:latest   "cat test.txt"         20 minutes ago   Exited (0) 7 seconds ago   5432/tcp    clair
089efafedd5e   postgres:latest   "docker-entrypoint.s..."  20 minutes ago   Up 20 minutes              5432/tcp    clair-db
856131e798d5   multi-stage-app   "java Main"             About an hour ago   Exited (0) About an hour ago   0           epic_liskov
78115c66745f   non-root-app      "sh"                     4 hours ago       Exited (99) 2 hours ago     0           brave_chandrase
khar
```

Encountering issues while trying to run the command as seen in the screenshot above:

```
clairctl --config clair-config.yaml report <your-dockerhub-username>/lightweight-app
```