CS328 · Distributed and Cloud Computing

Assignment 4

刘乐奇 / 12011327

May 30, 2023

1 Design

The RMI application is divided into 3 parts: registry, server, client. However, during packaging into Docker image, we need to modify some code in out RMI code.

1.1 host and port

Some constants are created due to the property of Docker. The registry (created by Registry) and the service (published by server) on the host 0.0.0.0 and then the request for the registry and services (by client) is from the host *host.docker.internal*, which is the address of the Docker's host, analogous to *localhost* when running directly on the host. The port used to communicate between registry and server or client is self-defined as 1200 so that it is convenient that we can map the port of Docker to port of the host.

```
public static final String defaultBindingHost = "0.0.0.0";
public static final String defaultAccessingHost = "host.docker.internal";
public static final int defaultBindingPort = 1200;
```

Besides, when *exportObject*() in the server, we should explicitly configure its port, which is also for mapping the port of Docker to port of the host.

```
MatMulMPI stub = (MatMulMPI) UnicastRemoteObject.exportObject(mmm, 43801);
```

1.2 visibility of multi-thread programming

In *UnicastRemoteObject* :: *exportObject*(), we can find that it will be spinning until the port of the skeleton is not 0. The port of the skeleton is modified in the other thread, so if we want to guarantee the port variable is visible after modified in the other thread, we need to add a key word *volatile* on the variable port. However, it does not matter in this assignment since we had configured the port other than 0.

```
public static Remote exportObject(Remote obj, String host, int port) throws RemoteException {
    String interfaceName = obj.getClass().getInterfaces()[0].getName();
    int objectKey = obj.hashCode();

    RemoteObjectRef ref = new RemoteObjectRef(host, port, objectKey, interfaceName);
    Skeleton skeleton = new Skeleton(obj, ref);
    skeleton.start();
    if (port == 0) {
        // busy waiting here until port is allocated
        while (skeleton.getPort() == 0);
        ref.setPort(skeleton.getPort());
    }
    System.out.printf("exportObject to %s:%d\n", host, skeleton.getPort());
    return Util.createStub(ref);
}
```

```
private volatile int port;
```

2 Screenshots of Docker

The followings are the screenshots as required in instruction. From top to bottom, it is the result of *docker version*, *docker build*, *docker push*, *docker compose*, *docker ps -a* and the *running result*.

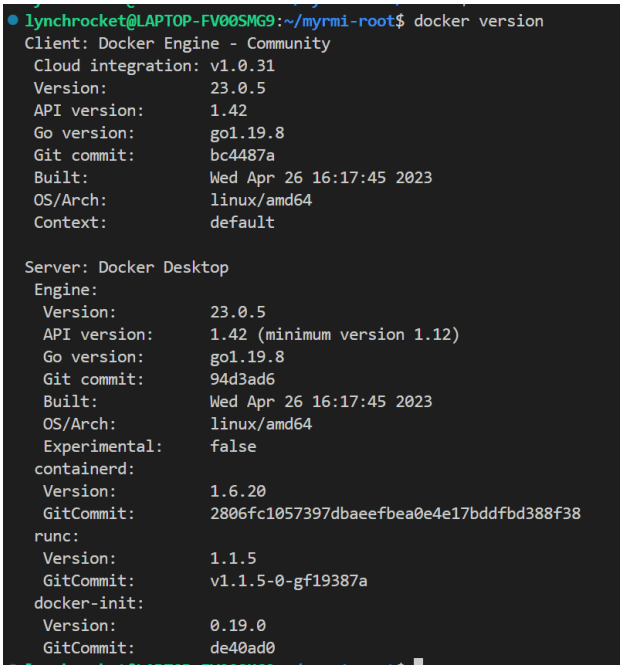

Figure 1: Docker version

Figure 2: registry build

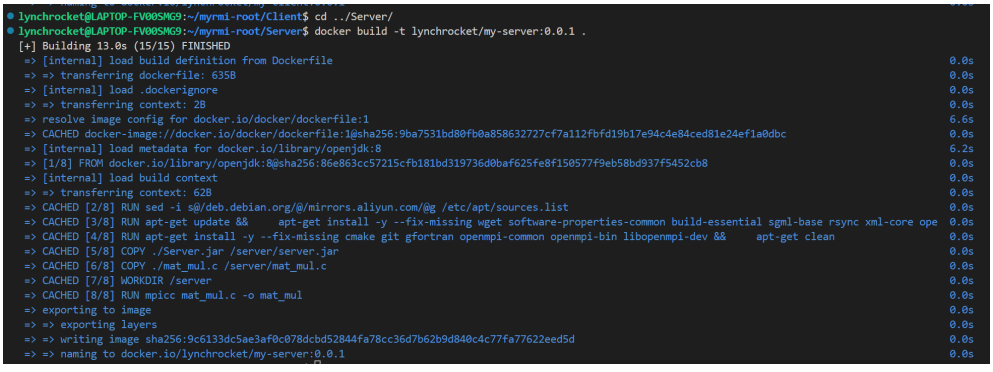

Figure 3: server build

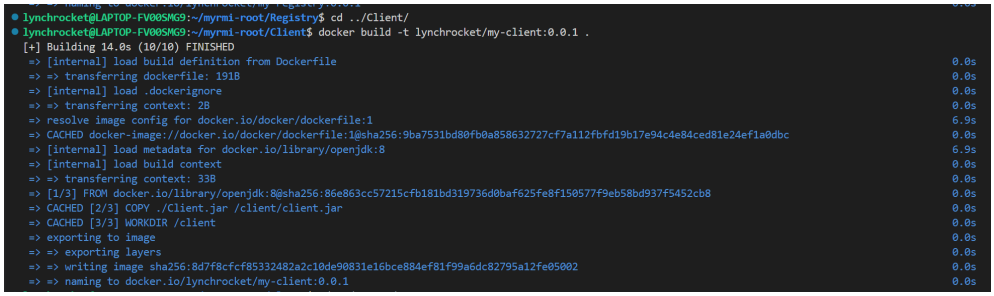

Figure 4: client build

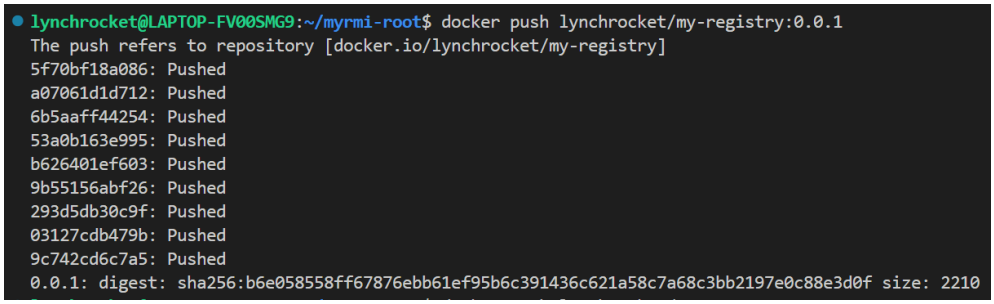

Figure 5: registry push

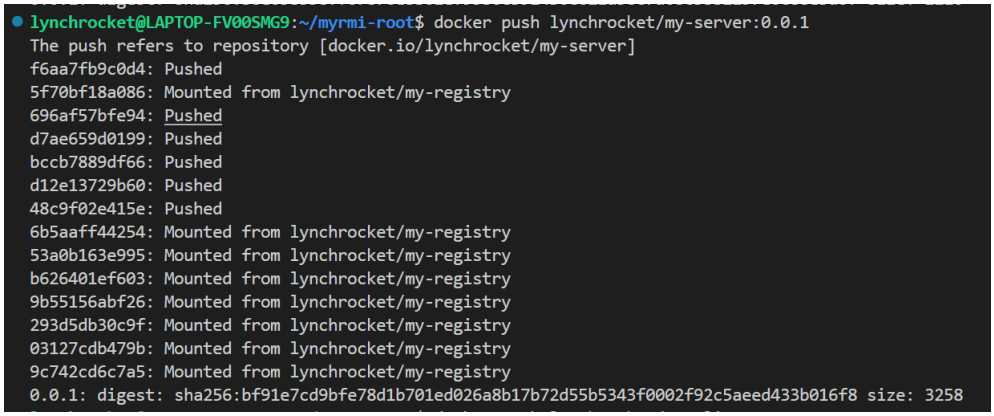

Figure 6: server push

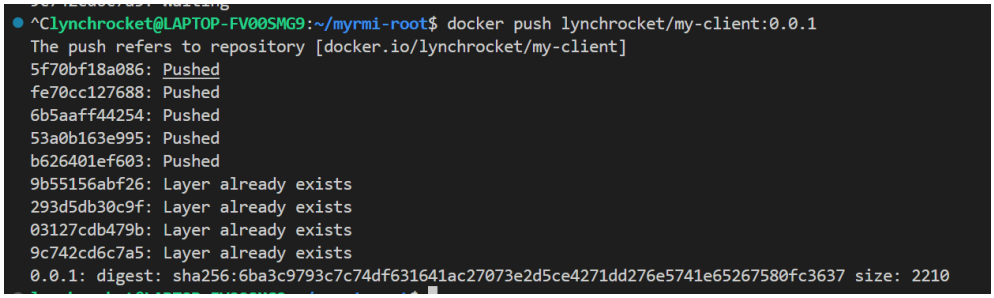

Figure 7: client push

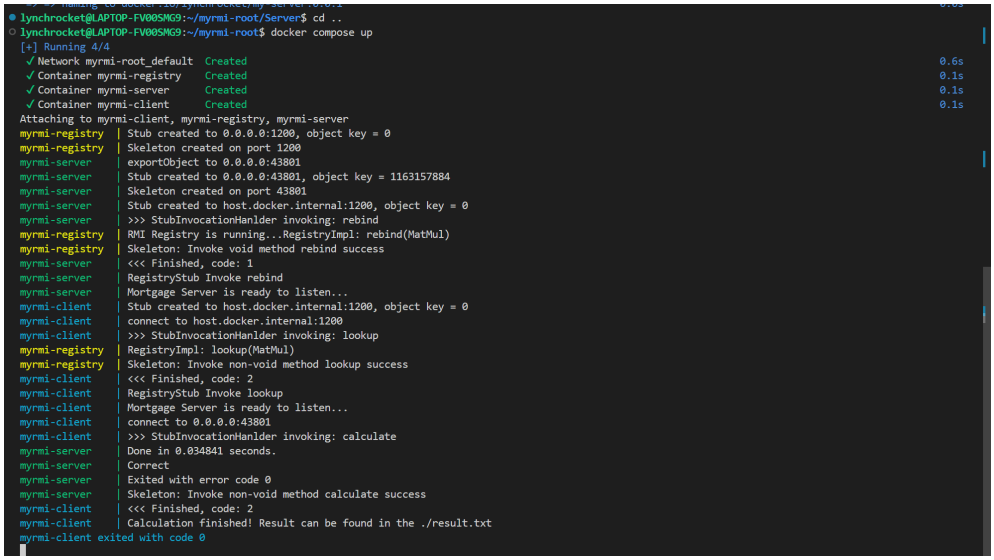

Figure 8: Docker compose

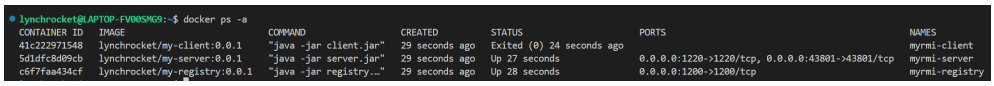

Figure 9: Docker ps -a

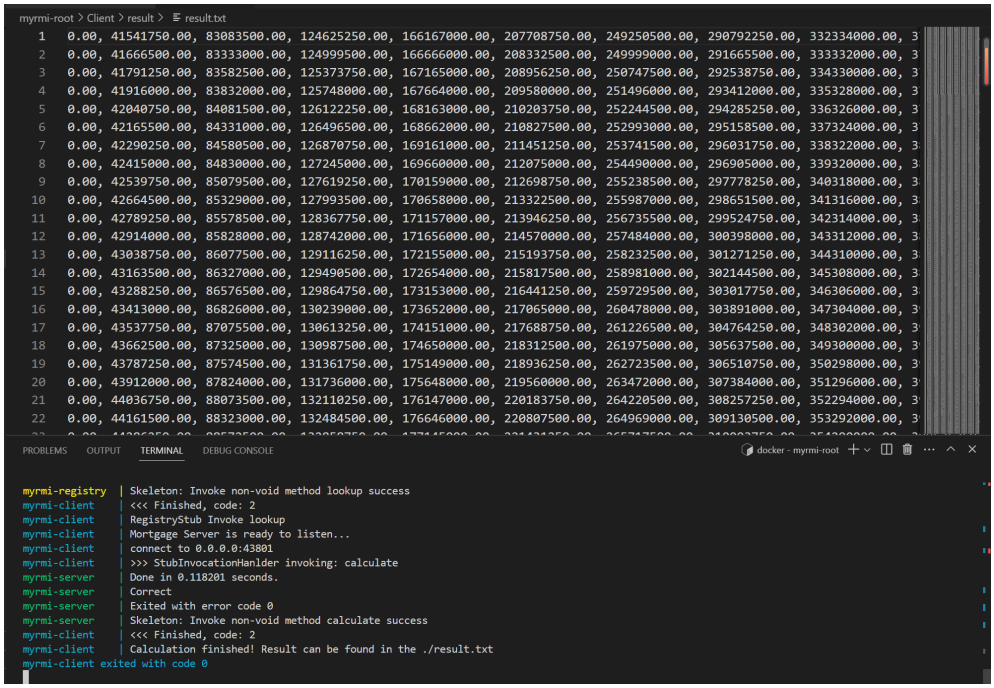

Figure 10: result

3 Dockerfile, docker-compose.yml and Source Code

They are packed in the submission files.

```
code --
        |-- myrmi-root--
        |                   |-- docker-compose.yml # docker compose yaml file
        |                   |-- Registry--
        |                   |               |-- Dockerfile # Dockerfile of registry image
        |                   |               |-- Registry.jar
        |                   |-- Server--
        |                   |           |-- Dockerfile # Dockerfile of server image
        |                   |           |-- mat_mul.c # program of MPI matrix multiplication
        |                   |           |-- Server.jar
        |                   |-- Client--
        |                   |           |-- Dockerfile # Dockerfile of client image
        |                   |           |-- Client.jar
        |                   |           |-- result--
        |                   |           |           |-- result.txt  # The calculating result
        |-- Registry-- # Source code of registry
        |               |-- ...
        |-- Server-- # Source code of server
        |           |-- ...
        |-- Client-- # Source code of client
        |           |-- ...
```

4 Docker Hub link

https://hub.docker.com/repositories/lynchrocket

5 Reference

[1] Java 如何调用 C 语言程序，JNI 技术
[2] Java Native Interface Specification—Contents
[3] 【JNI 编程】JNI 数组使用
[4] idea 打包 Jar 包