Assignment 4: Dockerfile and Docker Compose (Part 1)

CS328 - Distributed and Cloud Computing

DDL: 23:59, January 7, 2024

1 Introduction

You are required to design and code a working program and submit well-commented code together with a lab report. Any text should be in English only. Plagiarism is strictly prohibited.

2 Preamble

Before working on this assignment, you should be familiar with the following:

- Basic Linux knowledge (e.g. command line, installing software)
- Docker and its command-line tools
- `Dockerfile` format
- `Docker Compose` tool

You may find some useful information in the lab slides.

3 Assignment Specification

In this assignment, you are required to download and deploy docker on a Linux machine, pack your RMI application into an image, use `Docker Compose` to run your RMI application, and finally publish your image on Dockerhub.

In Assignment 2, you have practised how to run a client-server application based on the RMI framework. **Now you are required to pack your RMI server application into a docker image, such that it could be run as containers.** The demo server and client codes will be provided to you, but you can also write your own demo client and server.

Your application will have 3 components: server, registry and client. You need to start 3 docker containers to run them. **Each component should be running in one container.** Also, most applications need some parameters to start (e.g. host:port to bind), and may also require specific ordering when starting (registry needs to start before server, and server before client). To simplify this procedure, a tool named `Docker Compose` was introduced. You need to write the configuration for `Docker Compose` and use it to run your application.

3.1 Docker Image

Your application should have 3 components: Server, Registry, and Client. You need to pack each into a separate docker image. In total 3 docker images are required.

Notice that in the official Java RMI, the registry and server are not allowed to run on separate hosts. If you want to run Server and Registry in separate containers, please do not use the official java `rmiregistry`.

3.2 Networking

Since Docker also provides network isolation, you need to be careful about the networking between your containers.

To allow your multiple containers to communicate, you need to create a docker network and attach all your containers to that network. Each container will be assigned with an IP address with its DNS automatically configured. Then, these containers will be able to directly access each other by IP, hostname, container name or **service name**. **We recommend using the service name to locate a specific container.** The networking can also be configured automatically using `Docker Compose` and the `docker-compose.yml` file.

Since each component (server, registry, client) runs within a container, you need to design a special handling approach for the IP and ports so that `exportObject()`, object registration, `Registry.lookup()` and `getRegistry()` will work properly. This is because, for instance, the remote object may be exported at the `localhost` or `127.0.0.1` in the design in Assignment 2, and the server then needs to register the remote object to the registry. Here if the server, registry and client are containerised, when registering a remote object to the registry, it is the IP/hostname of the server *container* that should be registered, not `localhost` (where the remote object is listening at), such that the client container will be able to locate the remote object.

3.3 Startup Ordering

Different components (registry, server and client) have dependencies in between and they should be spun up in the following order: 1) registry, 2) server, and 3) client. The reason is that if the registry is not running, the server will not be able to register remote objects to the registry. **You are required to provide a design that ensures the correct launching ordering and that resolves the dependencies between the components.**

Here, we provide some hints for solving the issues above. In `Docker Compose`, the `depends_on` option can be used to specify the startup dependency between containers.

However, `depends_on` only checks whether the container is ready. It cannot guarantee the application inside the container is ready and running. Additional mechanisms are needed at the application level to ensure the program inside of the container is also ready. You can find some useful information at: `https://docs.docker.com/compose/startup-order/`.

The startup order of the different components should be similar to the following:

1. Launch the registry container
2. Launch the server container
   (a) get the registry by the registry container's service name
   (b) export the object
   (c) register the object to the registry
3. Launch the client container
   (a) get the registry by its service name
   (b) lookup the remote object
   (c) invoke remote methods

3.4 Some Other Hints

- Try using `0.0.0.0` rather than `127.0.0.1` when creating a socket server in your RMI Java source code.
- You may build your own docker image from a base image with built-in Java support (e.g. official openjdk docker images) rather than building one from a Linux docker image. Use `Dockerfile` to create your image, do not use `docker commit`.
- You need to register an account on `https://hub.docker.com/`. Do not forget to verify your email, otherwise, you will not be able to publish images.
- After you have finished writing `docker-compose.yml`, use some online grammar checker to ensure that your YMAL syntax is correct. It is very easy to mess up with spaces and indents in YAML.

4 Guide to Docker Compose

`Docker Compose` is written in Python, so you need to ensure you have the Python environment correctly configured.

First, check whether Docker Compose has been installed:

```
$ docker compose version
```

If the above command prints the version information, then you are good to go. Otherwise, you need to install it by following this guide: https://docs.docker.com/compose/install/linux/#install-the-plugin-manually

The core of Docker Compose is its configuration file, named docker-compose.yml. As the suffix suggests, it uses YAML markup language. You can find information about the syntax of YAML at https://learn.getgrav.org/17/advanced/yaml. It may look like this:

```
version: '3'
services:
    registry:
        image: 'my-registry:0.0.1'
        container_name: 'myrmi-registry'
        environment:
            RUN_COMPONENT: 'REGISTRY'
    server:
        image: 'my-server:0.0.1'
        container_name: 'myrmi-server'
        depends_on:
            - registry
        volumes:
            - ~/myrmi-root:~
        environment:
            RUN_COMPONENT: 'SERVER'
    client:
        image: 'my-client:0.0.1'
        container_name: 'myrmi-client'
        depends_on:
            - registry
            - server
        environment:
            RUN_COMPONENT: 'CLIENT'
```

The first line is the version specification, here you may simply put "3". Under services are detailed configurations about all services that need to be launched. In the above example, there is the registry, server and client as three separate services. In each service's configuration, you need to specify the image name, container name, and some necessary environment variables (e.g. JAVA_HOME, CLASSPATH etc.). The keyword depends_on specifies the dependencies of a container, so the current container will start after all the dependencies have started. The keyword volumes is used for mapping storage inside the container to a location in the host machine.

After you have created and modified the configuration file like above, simply use the command "docker compose up" to start your whole application.

You can refer to docker's documentation for more information: `https://docs.docker.com/compose/` and `https://docs.docker.com/compose/reference/up/`

5 Submission

Submit your assignment (code and report) in an archive to the course's Blackboard page. The reports need to be exported to PDF format. Be sure to submit your work before the deadline, or there may be penalties. In this particular assignment, you need to submit the following files:

- A report that describes your design, and at least 6 screenshots showing:
  - result of `docker version`;
  - result of `docker build`;
  - result of `docker push`, which pushes your customised images to the docker hub;
  - result of running application with `docker compose`;
  - result of `docker ps -a` when the server and registry are running;
  - result of successfully running remote method invocation from the client.
- Your `Dockerfile`
- Your `docker-compose.yml`
- Your application source code
- Link to your published docker image on Dockerhub (in report)

Pack all files into `SID_NAME_A4.zip`, where `SID` is your student ID and `NAME` is your name (e.g., 11710106_ 张三 _A4.zip).