

作业二 Docker & Docker Compose

介绍

本次作业中，我们将：

- 安装 Docker，掌握 Docker 的基本命令和使用方式，为容器挂载数据卷实现数据持久化；
- 编写 Dockerfile，为给定的 Web 程序构建镜像并运行；
- 编写 docker-compose 配置文件，部署一个给定的简单微服务系统，并为它添加链路追踪功能。

本地环境（推荐）/阿里云服务器均可完成本次作业。

安装 Docker

- Windows & Mac OS：

按照官方文档安装 Docker Desktop

- Windows： <https://docs.docker.com/desktop/install/windows-install/>

（需要 WSL2。如果你从未用过 WSL 或正在使用 WSL1，请参考 [如何使用 WSL 在 Windows 上安装 Linux](#) 安装 WSL2；当然，你也可以开启一个 Linux 虚拟机，以 Linux 的方式安装 Docker）

（WSL 需要较多内存。如果你在完成作业后不再使用 Docker Desktop 和 WSL，可在退出 Docker Desktop 后通过 `ws1 --shutdown` 命令关闭 WSL）

- Mac OS： <https://docs.docker.com/desktop/install/mac-install/>

- Linux (Ubuntu)： <https://docs.docker.com/engine/install/ubuntu/>

Docker 的基本使用

我们以 MySQL 的镜像拉取、容器创建、使用和销毁为例，简要体验 Docker 的基本使用流程。

下面的内容是基本的演示和引导，如果你已经熟悉 Docker 的使用，可以跳过这一节。

1. 拉取镜像

```
docker pull mysql:8.0.33
```

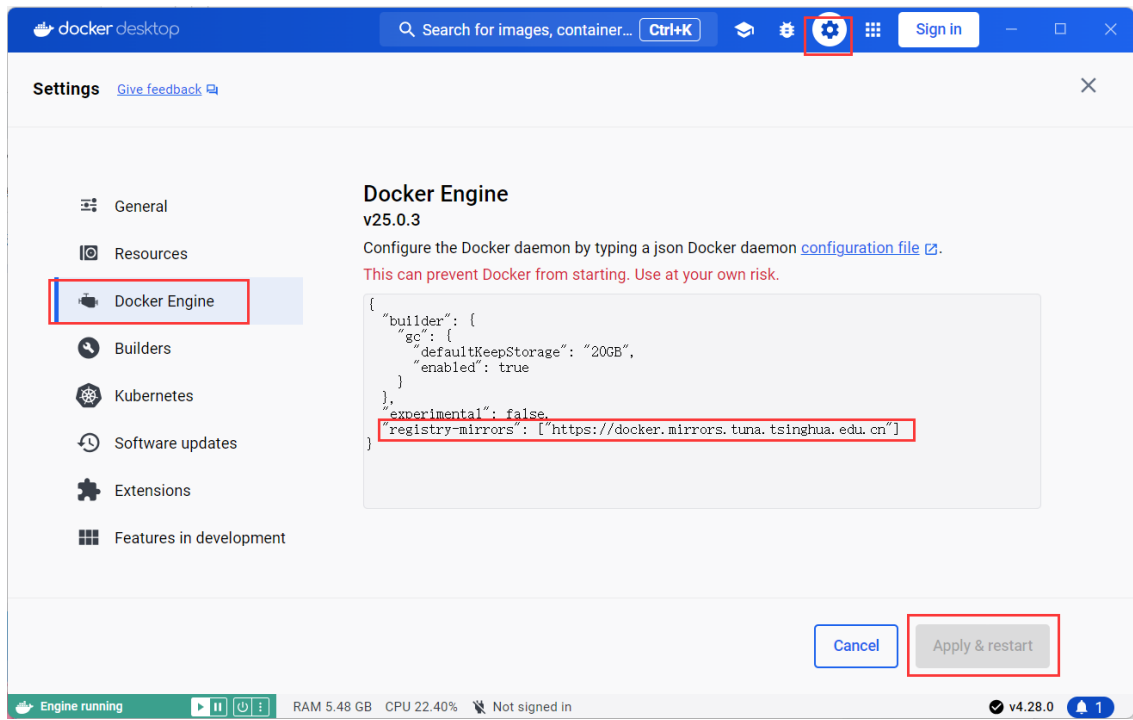
默认情况下，这行命令会从 Docker Hub 找到镜像名为 mysql 且 Tag 为 8.0.33 的镜像，并将其下载到本地。

- [Docker Hub](#) 是 Docker 官方维护的镜像仓库。
- 如果没有指定 Tag，直接执行 `docker pull mysql`，那么下载的将会是 `mysql:latest`。

如果你打不开上面那个链接在拉取镜像时遇到网络问题，请自行搜索【Docker 更换国内镜像源】配置国内镜像。

注意：

- 配置文件为 json 文件，修改时若格式不当（如少个逗号）会导致 Docker 无法正常启动，务必检查格式是否正确。
- 对于 Docker Desktop，可以在界面中方便地修改配置，如下图所示：



成功拉取 MySQL 镜像后，我们将看到以下提示信息：

```
$ docker pull mysql:8.0.33
8.0.33: Pulling from library/mysql
49bb46380f8c: Pull complete
aab3066bbf8f: Pull complete
d6eef8c26cf9: Pull complete
0e908b1dcb2: Pull complete
480c3912a2fd: Pull complete
89a648ecb3cf: Pull complete
6313eed00780: Pull complete
668fe2d98404: Pull complete
d3f8a843b813: Pull complete
c80ab9fc8db5: Pull complete
1b8b6b073273: Pull complete
Digest: sha256:ea68e51ffe9b96fef6076f1218af11301aeaf13c6201e0ec9aaef5791d5ddc5d
Status: Downloaded newer image for mysql:8.0.33
docker.io/library/mysql:8.0.33
```

查看本地所有镜像：`docker image ls`

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED
mysql	8.0.33	f6360852d654	8 months ago
SIZE			
565MB			

2. 启动容器

有了 MySQL 镜像后，我们就可以基于它来启动对应的 MySQL 容器：

```
docker run --name test-mysql -p 4306:3306 -e MYSQL_ROOT_PASSWORD=sysu2024 -d
mysql:8.0.33
```

- `--name test-mysql`：指定容器名为 `test-mysql`。若未指定该选项，Docker 会随机生成一个容器名。
- `-p 4306:3306`：将容器的 3306 端口映射到宿主机的 4306 端口。这样，在宿主机上可以通过 127.0.0.1:4306 连接到容器内的 MySQL 数据库。
- `-e`：配置容器内部使用的环境变量。这里设置了环境变量 `MYSQL_ROOT_PASSWORD=sysu2024`，为 MySQL 的 root 用户设置密码为 `sysu2024`。对于 MySQL 容器，该环境变量是必须在启动时设置的，否则容器将无法启动。
- `-d`：后台启动容器。
- `mysql:8.0.33`：要使用的镜像。

启动容器后，该容器 ID 会被打印出来：

```
$ docker run --name test-mysql -p 4306:3306 -e MYSQL_ROOT_PASSWORD=sysu2024 -d mysql:8.0.33 d08dc4c52db242f1d62c8dbe6c87c2143ca7e3262996dc8905b73babb841fbcd
```

查看正在运行的所有容器：

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS		NAMES		
ddcbe817ebea	mysql:8.0.33	"docker-entrypoint.s..."	10 minutes ago	Up 9 minutes
	3306/tcp, 33060/tcp	test-mysql		

`docker ps` 只会列出正在运行的容器，若要查看所有容器（包括未运行的），使用 `docker ps -a`。

若 `STATUS` 一列中，容器的状态为 `Up xx minutes`，则表示容器正常运行中。

3. 进入容器，操作 MySQL 数据库

接下来，我们进入容器内部，登录 MySQL，添加一些数据：

```
docker exec -it test-mysql /bin/bash
```

可以看到终端提示符变成了 `bash-4.4#`，表明此时我们正在容器内部执行命令。

以 root 用户登录 MySQL：`bash-4.4# mysql -u root -p`

```
bash-4.4# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

我们成功连接上了 MySQL。在 `mysql>` 的提示符下，我们可以执行 SQL 语句，随意添加一些数据：

```
mysql> CREATE DATABASE CLOUD;
mysql> USE CLOUD;
mysql> CREATE TABLE student (id INT AUTO_INCREMENT PRIMARY KEY, username
VARCHAR(50) NOT NULL);
mysql> INSERT INTO student (username) value ('StreamAzure');
mysql> select * from student;
+----+-----+
| id | username |
+----+-----+
|  1 | StreamAzure |
+----+-----+
1 row in set (0.00 sec)
```

最后，我们执行 `exit` 命令，依次退出 MySQL、退出容器，即可回到原先的命令行中：

```
mysql> exit
Bye
bash-4.4# exit
exit
```

4. 停止并销毁容器

在销毁容器之前，需要先让容器停止运行：

```
docker stop test-mysql
```

然后删除容器：

```
docker rm test-mysql
```

执行 `docker rm` 命令后，与这个容器相关的一切都烟消云散了……真的吗？

作业要求

任务一 (50分)

容器是“一次性的”和“脆弱的”，容器很容易因为各种原因被 kill（如资源不足等），如果没有对容器做相关配置，容器内部数据也会随之丢失。此外，当我们想要对容器中应用的配置文件进行修改的时候，如果每次需要 `docker exec` 命令进入容器内部才能修改，也未免过于繁琐。

数据卷 (Volume) 是 Docker 解决以上问题的数据持久化机制。请查找相关资料，**在容器启动命令中** 添加合适的选项，实现以下效果：

1. 启动一个 MySQL 容器，并在数据库中随意添加一些数据；
2. 删除该容器；
3. 重新启动一个 MySQL 容器，它还能读取到之前添加的数据。

你的作业中需要包含：

- 以上三个步骤的对应截图和简要文字说明，并回答以下问题：

- 当我们使用 `docker run --name test-mysql -p 4306:3306 -e MYSQL_ROOT_PASSWORD=sysu2024 -d mysql:8.0.33` 命令创建容器，再用 `docker rm test-mysql` 命令删除容器时，还遗留了什么未被删除？什么命令能在删除容器时一并删除它？

任务二 (40分)

Dockerfile 是构建 Docker 镜像所需的文本文件，它包含了构建镜像的所有指令和说明。请你为下面给出的 demo Web 程序编写 Dockerfile，构建镜像并启动容器，访问其中的 Web 服务。

请你：

1. 在[这里](#)下载 demo Web 程序源码；
2. 阅读 README，并参考相关资料（如[Docker Dockerfile - 菜鸟教程](#)），为 `demo.py` 构建编写合适的 **dockerfile 文件**，所构建的镜像只需要保证能在容器启动时启动 `demo.py` 即可。
 - **恰当选取基础镜像。**相较于庞大的 Ubuntu 系统镜像，精简的 Python 基础镜像是更好的选择，因为我们的容器只需要运行一个 Python 程序。
 - 在基础镜像选取合适的前提下，只有以下命令是必须的：`FROM`，`ADD` 或 `COPY`，`EXPOSE`，`RUN`，`ENTRYPOINT` 或 `CMD`。此时 `dockerfile` 的内容只需要 10 行以内即可完成必要的构建工作。
3. 将 `dockerfile` 文件放在源码目录下，进入该目录，执行 `docker build` 命令构建镜像。
 - **镜像名为 `demo`，tag 为你的学号**，即 `demo:你的学号`。（例如 `docker build . -t demo:23232323`）
4. 基于你所构建的镜像启动容器（**注意使用 `-p` 选项将容器的 5000 端口映射到宿主机**），等待几秒，让容器内的 Web 服务完全启动。
 - 善用 `docker log` 命令查看启动进度、排查 Bug
5. 在宿主机中：
 - 打开浏览器（如果有），访问 <http://localhost:5000/hello/<任意字符串>>，应显示如 README 中所示的页面（注意 URL 中的 5000 取决于你在启动容器时映射到的宿主机端口，如映射到 12445 端口，应访问 <http://localhost:12445/hello/<任意字符串>>）
 - 如果没有浏览器，你也可以在命令行中执行 `curl` 命令访问：

```
curl http://localhost:5000/hello/<任意字符串>
```

对于 Windows Docker Desktop，容器实际上在 WSL 中运行的，此时“宿主机”是 WSL 而非 Windows。因此你可能在 Windows 浏览器中无法直接打开上述链接，而需要在 WSL 内执行 `curl` 命令访问。

你也可以配置 WSL（[参考资料](#)），实现在 Windows 中直接访问容器内服务。

你的作业中需要包含：

- 你编写的 Dockerfile 文件的完整内容
- 构建镜像、启动容器、访问 Web 服务等过程的对应截图和简要文字说明（需要包含你所执行的 `docker` 命令）

任务三 (10分)

微服务架构已经成为现代应用开发的主要范式之一。在这个架构中，一个完整的微服务系统由多个微服务构成，而每个微服务都在各自的 Docker 容器中运行。Docker 提供了理想的解决方案，使得微服务的构建、部署和管理变得简单而高效。

针对这些容器的部署、编排、管理和通信等问题，Kubernetes (K8s) 已经提供了很好的解决方案。但在这里，我们首先了解相对更简单的 **Docker Compose**。通过 Docker Compose，我们可以通过 **YAML 文件** 对多个容器构成的容器组进行统一的部署和配置。

[Jaeger](#) 是一个开源的分布式跟踪系统，它提供了端到端的分布式跟踪功能。通过 Jaeger，开发人员可以轻松追踪跨多个微服务的请求路径和并定位故障及性能问题。

[这里](#)提供了一个最简单的微服务系统，它由 2 个基于 Spring Boot 的 Web 应用 + 1 个 MySQL 数据库构成。

在本任务中，请你参考相关资料（如 [Docker Compose Specification](#), [Docker Compose - 菜鸟教程](#)），理解项目中 **docker-compose.yaml** 文件中各个配置项的作用，并按照以下步骤修改相关 dockerfile 文件及 docker-compose.yaml 文件，为这个微服务系统添加 Jaeger 链路追踪功能。

请你：

1. 部署并运行上述微服务系统：

- 下载[仓库源码](#)（已包含编译好的 jar 包，你无需自行编译），并阅读 README。
- 进入目录，执行 `docker-compose up --build -d` 命令（在你的机器上也可能是 `docker compose up --build -d`），它将自动构建并启动整个微服务应用（即启动 3 个容器构成的容器组）。

```
PS C:\Users\18372\Desktop\task3> docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
5a781942a9e1   demo/ticket-service:base            "java -jar /demo-tic-" 17 minutes ago Up 17 minutes 0.0.0.0:8082->8082/tcp   task3-ticket-service-1
4da032b72249   demo/user-service:base              "java -jar /demo-use-" 17 minutes ago Up 17 minutes 0.0.0.0:8081->8081/tcp   task3-user-service-1
aa5f8051b62c   mysql:8.0.33                        "docker-entrypoint.s..." 17 minutes ago Up 17 minutes 3306/tcp, 33060/tcp      task3-mysql-1
```

- 按照 README 测试它是否能正常响应请求。如果一切正常，应显示 200 响应代码及相关内容：

```
PS C:\Users\18372\Desktop\task3> curl http://localhost:8082/buyTicket/feign/1

StatusCode      : 200
StatusDescription :
Content         : User(id=1, name=Alice), station 上海
RawContent      : HTTP/1.1 200
                  Keep-Alive: timeout=60
                  Connection: keep-alive
                  Content-Length: 38
                  Content-Type: text/plain; charset=UTF-8
                  Date: Sun, 14 Apr 2024 06:09:04 GMT

                  User(id=1, name=Alice), station 上海
Forms           : {}
Headers         : {[Keep-Alive, timeout=60], [Connection, keep-alive], [Cont
Images          : {}
InputFields     : {}
Links           : {}
ParsedHtml      : mshtml.HTMLDocumentClass
RawContentLength : 38
```

2. 准备 OpenTelemetry Java Instrumentation

- 将目录中的 `opentelemetry-javaagent.jar` 文件分别复制到 `demo-user-service` 和 `demo-ticket-service` 目录下。

这个 jar 文件来自 [Opentelemetry-Java-instrumentation](#)，如有兴趣可进一步了解

- 修改 `demo-user-service` 和 `demo-ticket-service` 目录下的 Dockerfile 文件，令这个 jar 包能随 Web 服务一起启动，自动插桩采集数据：

```
# 添加以下配置
COPY opentelemetry-javaagent.jar opentelemetry-javaagent.jar
ENV JAVA_TOOL_OPTIONS "-javaagent:./opentelemetry-javaagent.jar"
ENV OTEL_EXPORTER_OTLP_ENDPOINT=http://jaeger-service:4318
# 采集到的数据直接导出给 Jaeger
```

3. 参考以下提示，修改 `docker-compose.yaml` 文件，在已有内容的基础上，添加 `jaeger-service` 容器配置。

- 服务名必须为 `jaeger-service`
- 该容器应使用镜像: `jaegertracing/all-in-one:latest`
- 该容器需要端口映射: `16686:16686`
- 需要配置一项环境变量: `COLLECTOR_OTLP_ENABLED=true`
- 该容器需要加入到文件中已有的容器网络

YAML 基本语法要求如下，修改文件时请注意缩进方式是否正确

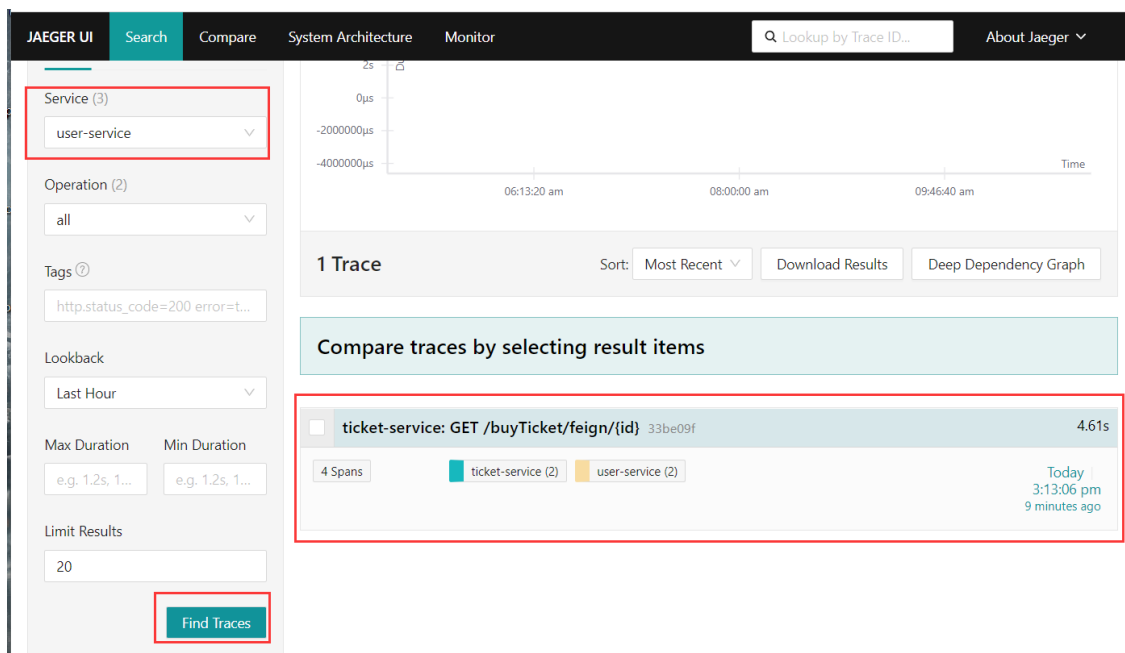
- 大小写敏感
- 使用缩进表示层级关系
- 缩进不允许使用 `tab`，只允许空格
- 缩进的空格数不重要，只要相同层级的元素左对齐即可
- `'#'` 表示注释

4. 修改完成后，重新执行 `docker-compose up --build -d`;

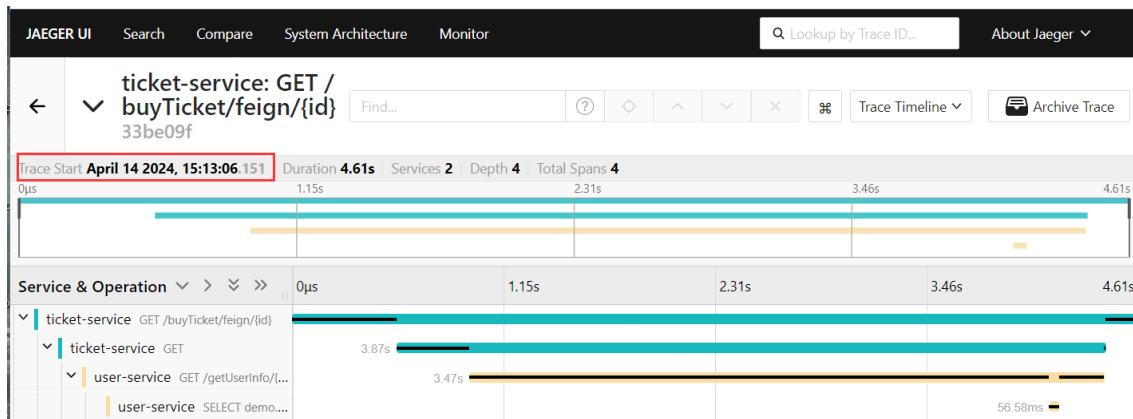
5. 待容器内服务全部启动完毕后，再次向微服务系统发送请求;

- 善用 `docker log` 命令查看启动进度、排查 Bug

6. 访问 Jaeger UI: <http://localhost:16686>，在左上角选择 `ticket-service` 或 `user-service`，点击 Find Traces，应能显示刚才发送的请求链路信息:



点击右侧任意一个 Item，进入详情页:



这表明 Jaeger 链路追踪功能已经成功地添加到微服务系统中，它将捕获并展示每一次请求的链路信息。

你的作业中需要包含：

- 你修改后的 docker-compose.yaml 文件的完整内容
 - 在 Jaeger UI 中查看到的 Trace 详情页截图（如上图所示）
-

提交 DDL 及提交方式：

请于【2024.04.29 23:59】前，将作业 PDF 文件（文件命名为 **姓名-学号-第二次作业**，如 张三-23232323-第二次作业.pdf）发送至邮箱【dengzl11@163.com】，邮件标题格式：**姓名-学号-第二次作业**