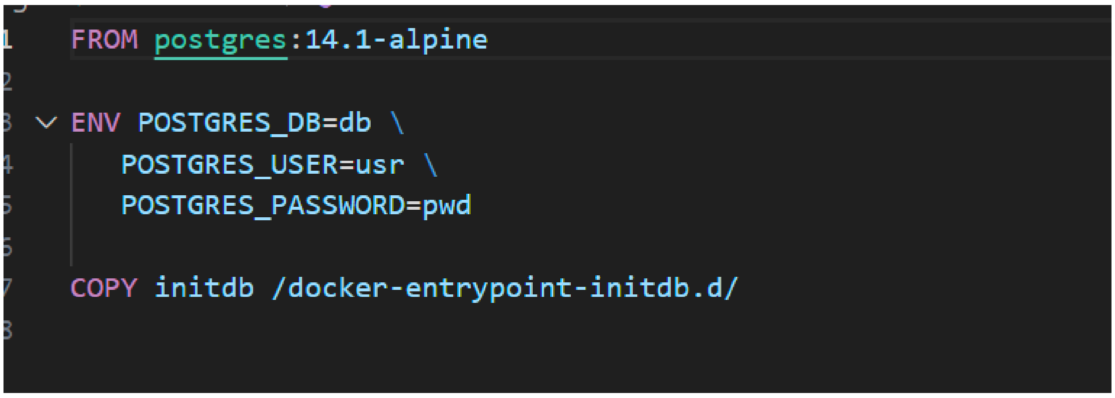
TP1 Devops

Database

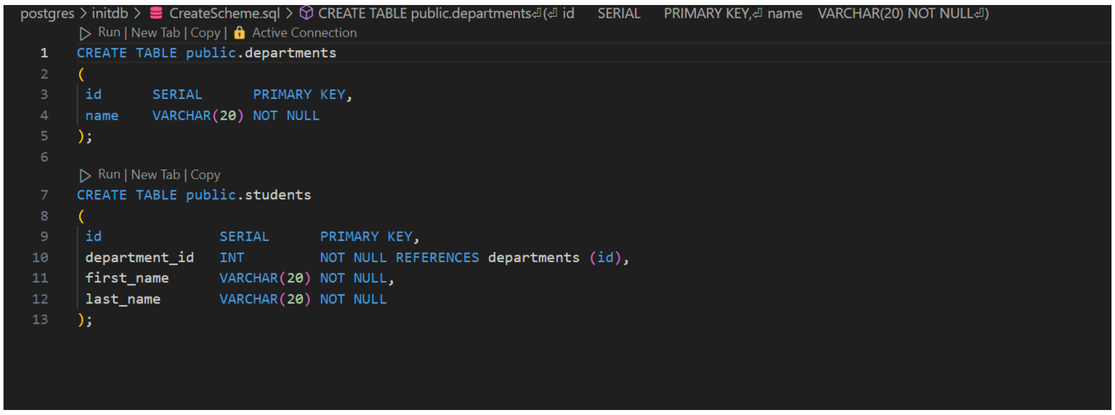
Basics

Question

1-1 Document your database container essentials: commands and Dockerfile.

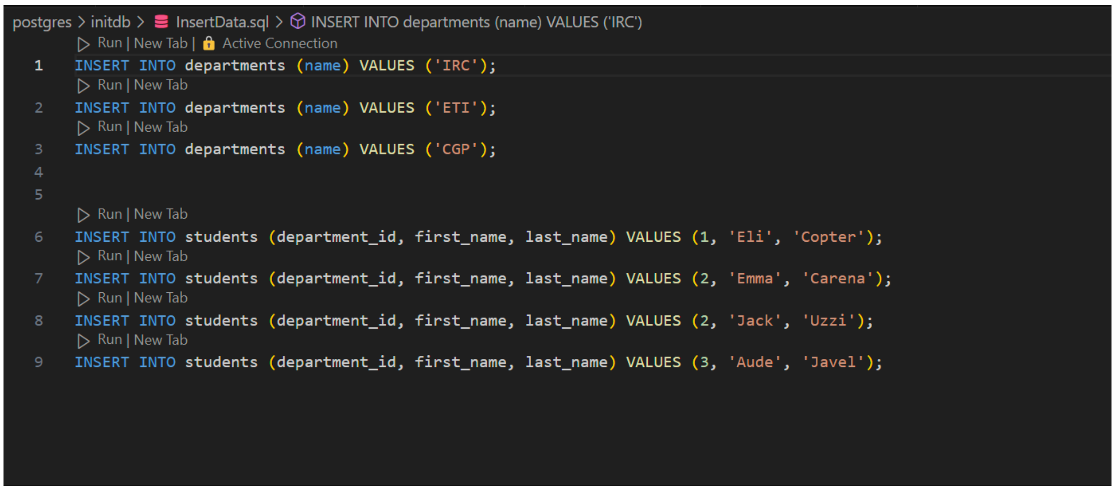


```dockerfile
FROM postgres:14.1-alpine

ENV POSTGRES_DB=db \
    POSTGRES_USER=usr \
    POSTGRES_PASSWORD=pwd

COPY initdb /docker-entrypoint-initdb.d/
```



postgres > initdb > CreateScheme.sql > CREATE TABLE public.departments⏎(⏎ id   SERIAL   PRIMARY KEY,⏎ name   VARCHAR(20) NOT NULL⏎)

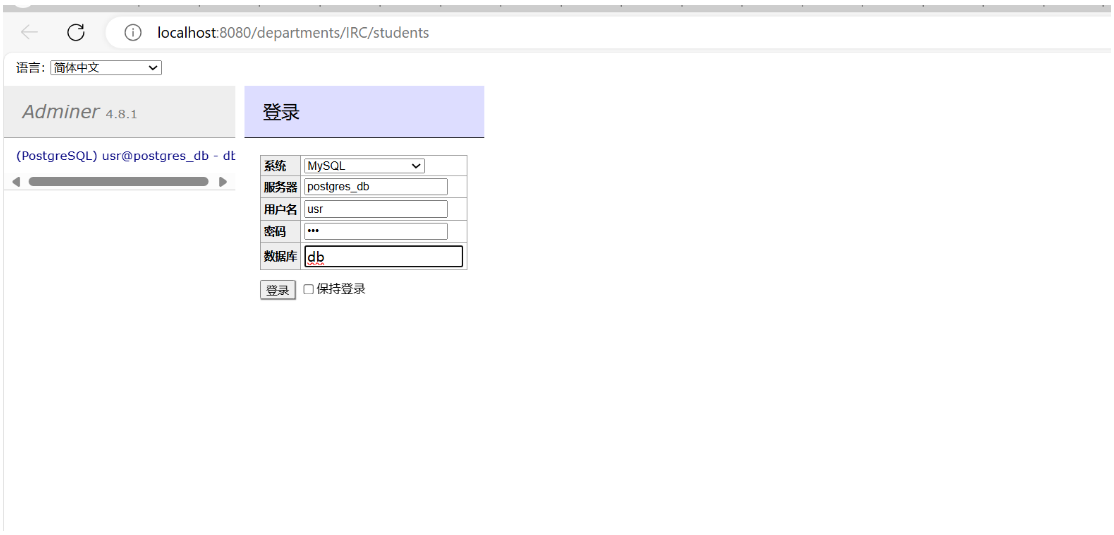
```sql
CREATE TABLE public.departments
(
 id      SERIAL      PRIMARY KEY,
 name    VARCHAR(20) NOT NULL
);

CREATE TABLE public.students
(
 id              SERIAL      PRIMARY KEY,
 department_id   INT         NOT NULL REFERENCES departments (id),
 first_name      VARCHAR(20) NOT NULL,
 last_name       VARCHAR(20) NOT NULL
);
```


postgres > initdb > InsertData.sql > INSERT INTO departments (name) VALUES ('IRC')

```sql
INSERT INTO departments (name) VALUES ('IRC');
INSERT INTO departments (name) VALUES ('ETI');
INSERT INTO departments (name) VALUES ('CGP');


INSERT INTO students (department_id, first_name, last_name) VALUES (1, 'Eli', 'Copter');
INSERT INTO students (department_id, first_name, last_name) VALUES (2, 'Emma', 'Carena');
INSERT INTO students (department_id, first_name, last_name) VALUES (2, 'Jack', 'Uzzi');
INSERT INTO students (department_id, first_name, last_name) VALUES (3, 'Aude', 'Javel');
```

docker network create app-network

docker-compose up -d –build


localhost:8080/departments/IRC/students

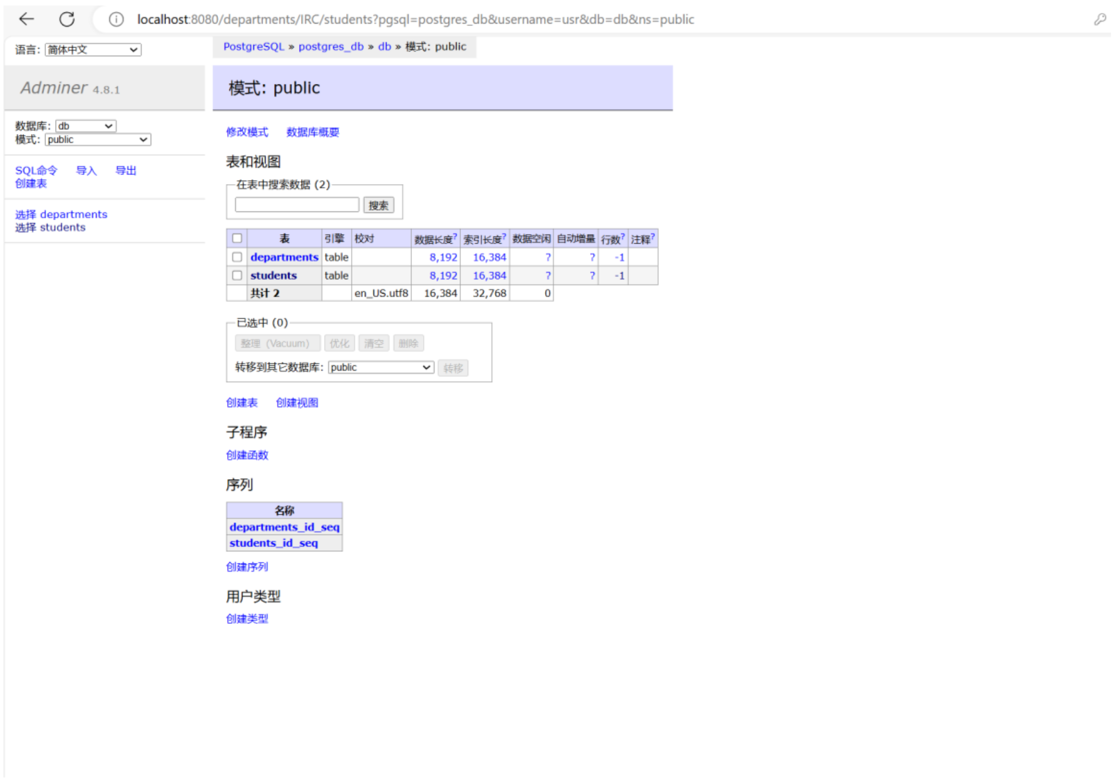
语言: 简体中文

Adminer 4.8.1

(PostgreSQL) usr@postgres_db - db

登录

系统	MySQL
服务器	postgres_db
用户名	usr
密码	•••
数据库	db



localhost:8080/departments/IRC/students?pgsql=postgres_db&username=usr&db=db&ns=public

语言: 简体中文

Adminer 4.8.1

数据库: db
模式: public

SQL命令 导入 导出
创建表

选择 departments
选择 students

PostgreSQL » postgres_db » db » 模式: public

模式: public

修改模式 数据库概要

表和视图

在表中搜索数据 (2)

搜索

☐	表	引擎	校对	数据长度?	索引长度?	数据空闲	自动增量	行数?	注释?
☐	**departments**	table		8,192	16,384	?	?	-1	
☐	**students**	table		8,192	16,384	?	?	-1	
	共计 2		en_US.utf8	16,384	32,768	0			

已选中 (0)

整理 (Vacuum) 优化 清空 删除

转移到其它数据库: public 转移

创建表 创建视图

子程序

创建函数

序列

名称
departments_id_seq
students_id_seq

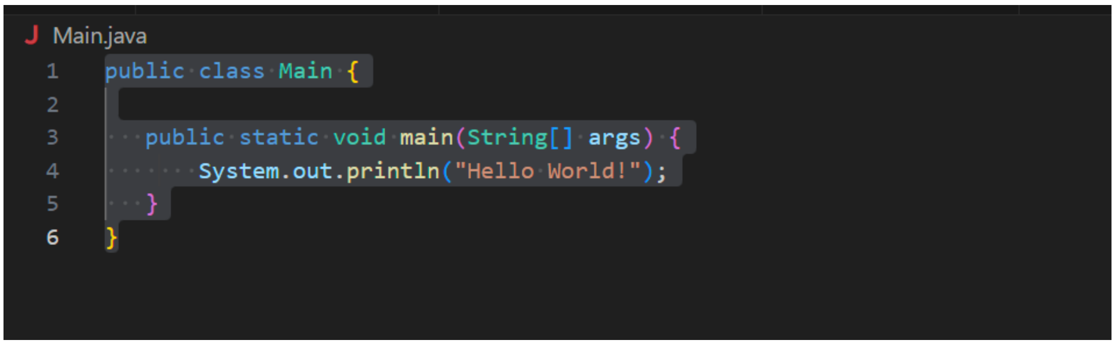
创建序列

用户类型

创建类型

Backend API

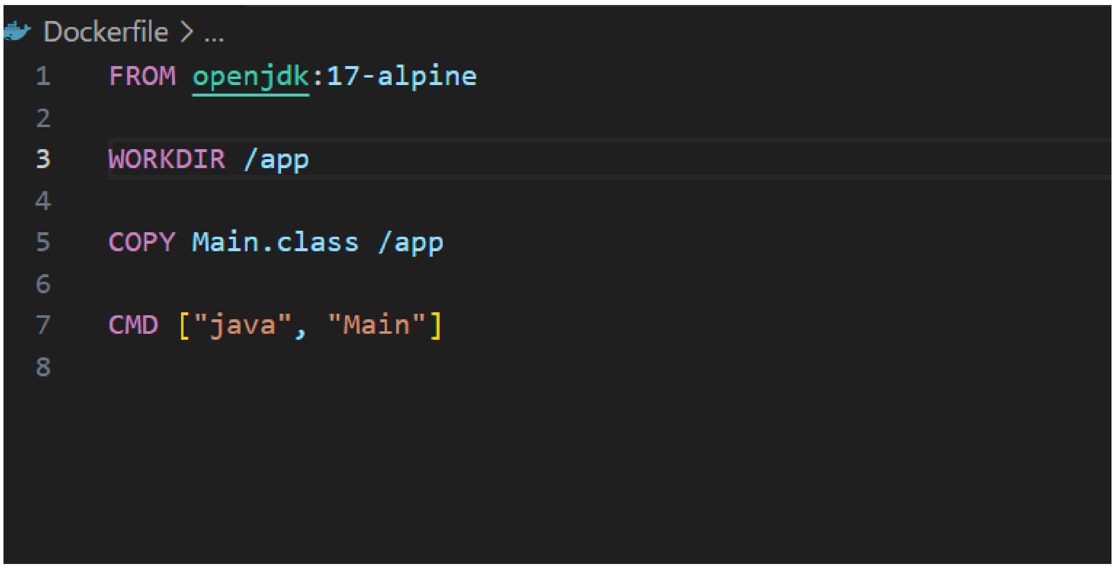
Main. java



Main.java

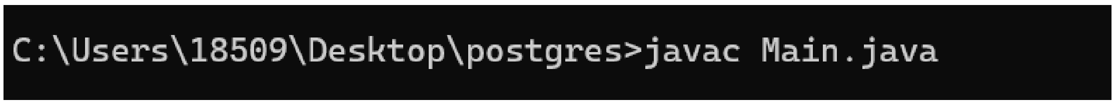
```java
public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

dockerfile

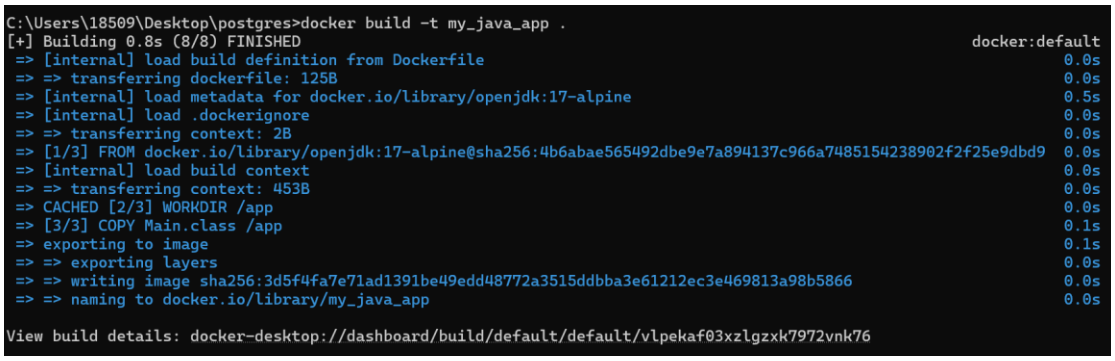


```
Dockerfile > ...
FROM openjdk:17-alpine

WORKDIR /app

COPY Main.class /app

CMD ["java", "Main"]
```

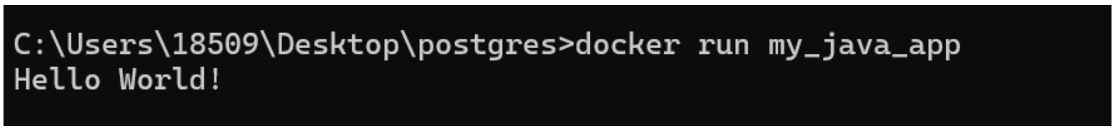
launch app



```
C:\Users\18509\Desktop\postgres>javac Main.java
```



```
C:\Users\18509\Desktop\postgres>docker build -t my_java_app .
[+] Building 0.8s (8/8) FINISHED                                                                        docker:default
 => [internal] load build definition from Dockerfile                                                              0.0s
 => => transferring dockerfile: 125B                                                                              0.0s
 => [internal] load metadata for docker.io/library/openjdk:17-alpine                                              0.5s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [1/3] FROM docker.io/library/openjdk:17-alpine@sha256:4b6abae565492dbe9e7a894137c966a7485154238902f2f25e9dbd9  0.0s
 => [internal] load build context                                                                                 0.0s
 => => transferring context: 453B                                                                                 0.0s
 => CACHED [2/3] WORKDIR /app                                                                                     0.0s
 => [3/3] COPY Main.class /app                                                                                    0.1s
 => exporting to image                                                                                            0.1s
 => => exporting layers                                                                                           0.0s
 => => writing image sha256:3d5f4fa7e71ad1391be49edd48772a3515ddbba3e61212ec3e469813a98b5866                      0.0s
 => => naming to docker.io/library/my_java_app                                                                    0.0s

View build details: docker-desktop://dashboard/build/default/default/vlpekaf03xzlgzxk7972vnk76
```

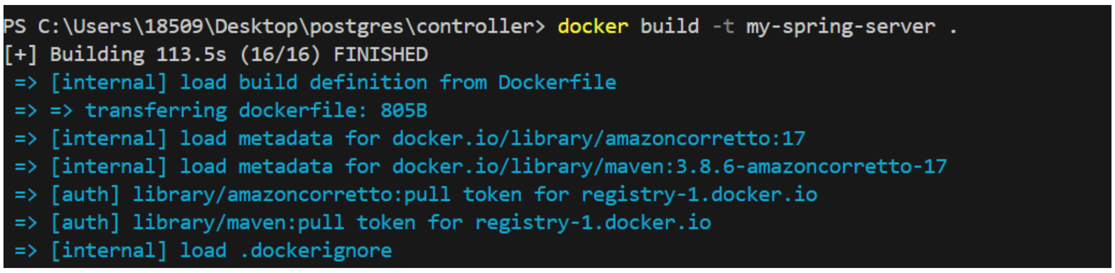


```
C:\Users\18509\Desktop\postgres>docker run my_java_app
Hello World!
```

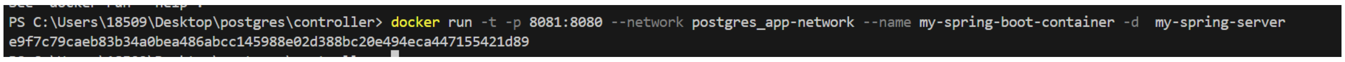
We prefer to use Multistage build because he doesn't require us to have the java

JDK installed on our computers, he just needs us to have docker to use it.

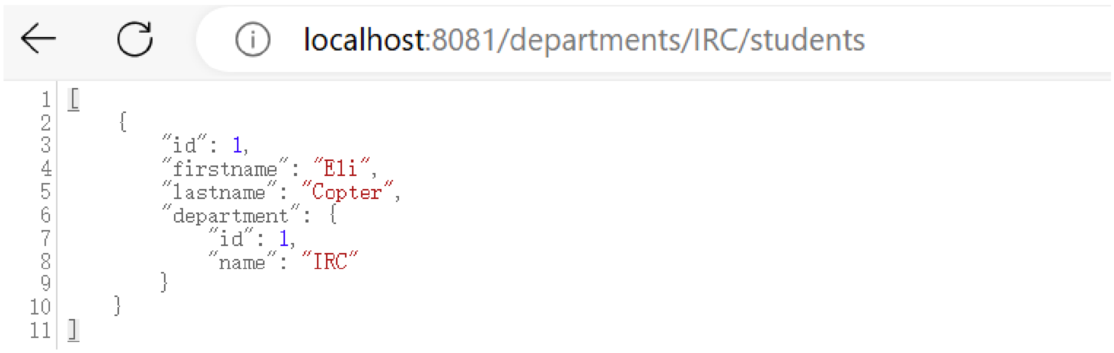
Backend simple api



```
PS C:\Users\18509\Desktop\postgres\controller> docker build -t my-spring-server .
[+] Building 113.5s (16/16) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 805B
 => [internal] load metadata for docker.io/library/amazoncorretto:17
 => [internal] load metadata for docker.io/library/maven:3.8.6-amazoncorretto-17
 => [auth] library/amazoncorretto:pull token for registry-1.docker.io
 => [auth] library/maven:pull token for registry-1.docker.io
 => [internal] load .dockerignore
```



```
PS C:\Users\18509\Desktop\postgres\controller> docker run -t -p 8081:8080 --network postgres_app-network --name my-spring-boot-container -d  my-spring-server
e9f7c79caeb83b34a0bea486abcc145988e02d388bc20e494eca447155421d89
```

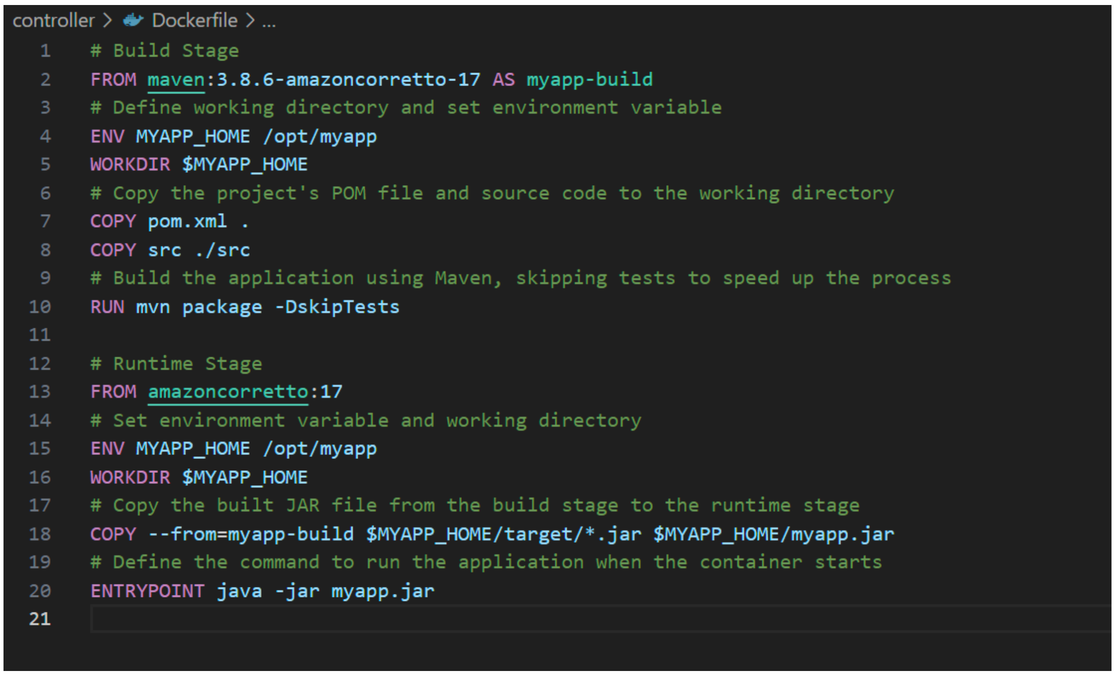


localhost:8081/departments/IRC/students

```
[
    {
        "id": 1,
        "firstname": "Eli",
        "lastname": "Copter",
        "department": {
            "id": 1,
            "name": "IRC"
        }
    }
]
```

Question

1-2 Why do we need a multistage build? And explain each step of this dockerfile.



```dockerfile
# Build Stage
FROM maven:3.8.6-amazoncorretto-17 AS myapp-build
# Define working directory and set environment variable
ENV MYAPP_HOME /opt/myapp
WORKDIR $MYAPP_HOME
# Copy the project's POM file and source code to the working directory
COPY pom.xml .
COPY src ./src
# Build the application using Maven, skipping tests to speed up the process
RUN mvn package -DskipTests

# Runtime Stage
FROM amazoncorretto:17
# Set environment variable and working directory
ENV MYAPP_HOME /opt/myapp
WORKDIR $MYAPP_HOME
# Copy the built JAR file from the build stage to the runtime stage
COPY --from=myapp-build $MYAPP_HOME/target/*.jar $MYAPP_HOME/myapp.jar
# Define the command to run the application when the container starts
ENTRYPOINT java -jar myapp.jar
```

Multistage builds are essential in Docker for several reasons:

1.Image Size Reduction: By separating the build and runtime environments, unnecessary build dependencies are discarded, resulting in smaller final images. This optimization is crucial for efficient image distribution and storage.

2.Improved Build Efficiency: Docker can cache intermediate build stages, speeding up subsequent builds by reusing unchanged layers. This reduces build times, especially for large projects with complex dependencies, leading to faster development cycles.

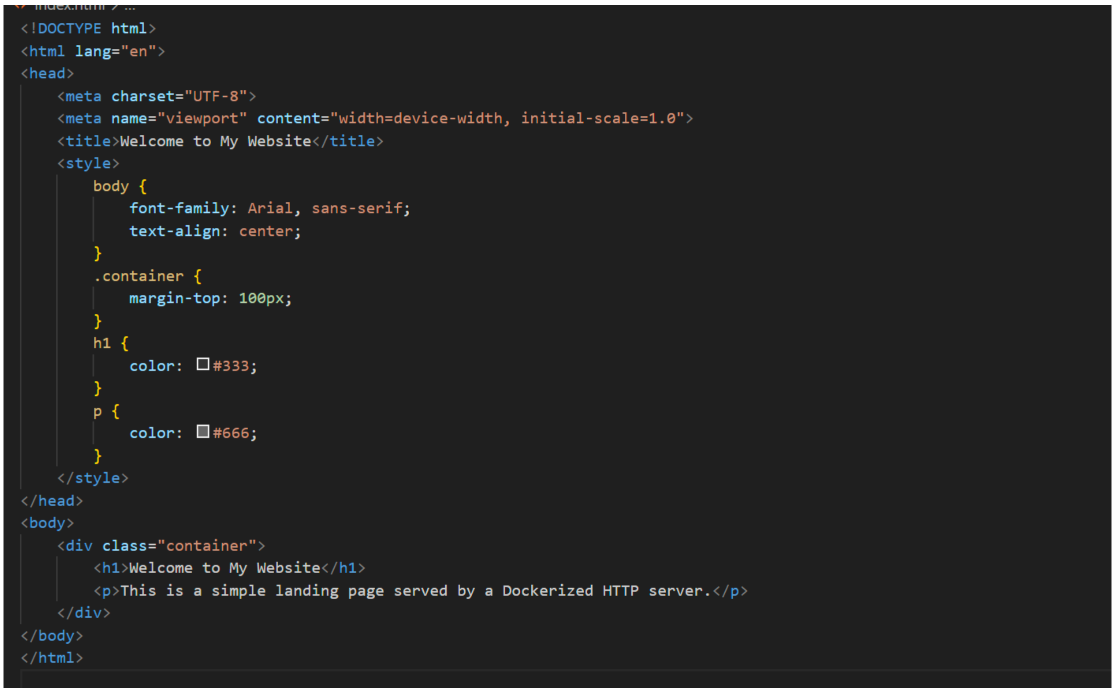
3.Dependency Isolation: Multistage builds provide a clean separation between build-time and runtime dependencies. This isolation enhances security by ensuring that only necessary runtime artifacts are included in the final image, reducing potential vulnerabilities.

4.Simplified Dockerfiles: Multistage builds streamline Dockerfiles by removing unnecessary build artifacts and keeping only essential runtime components. This simplification enhances readability, maintainability, and understanding of Dockerfile structures.

Overall, multistage builds optimize Docker image creation by minimizing size, improving efficiency, enhancing security, and simplifying development workflows.

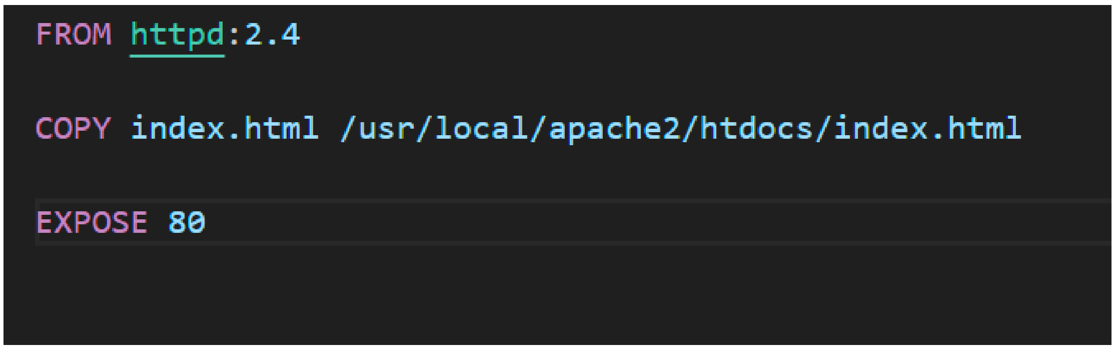
Http server

`index.html`

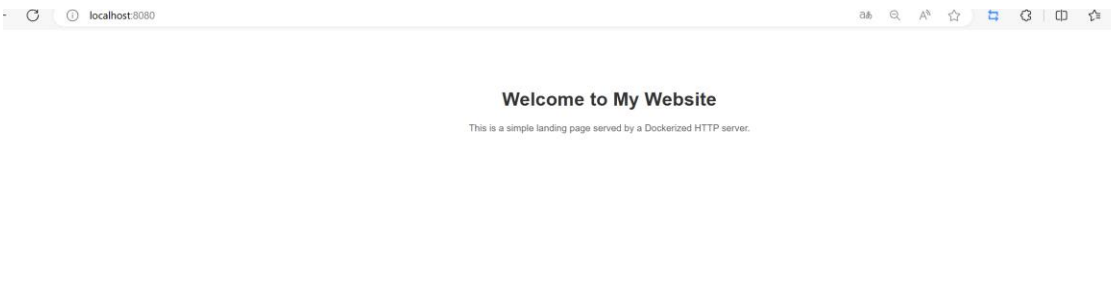


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome to My Website</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
        }
        .container {
            margin-top: 100px;
        }
        h1 {
            color: #333;
        }
        p {
            color: #666;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome to My Website</h1>
        <p>This is a simple landing page served by a Dockerized HTTP server.</p>
    </div>
</body>
</html>
```

dockerfile



```dockerfile
FROM httpd:2.4

COPY index.html /usr/local/apache2/htdocs/index.html

EXPOSE 80
```



localhost:8080

Welcome to My Website

This is a simple landing page served by a Dockerized HTTP server.

Tip

Why is **docker-compose** so important?

1. Simplifying multi-container application management

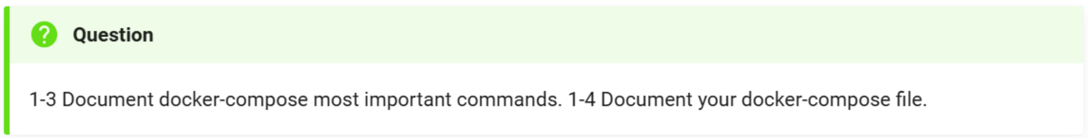
Docker Compose allows we to define and manage multiple containers in a single file (docker-compose.yml). This simplifies the process of launching and managing complex applications, eliminating the need to manually write multiple docker run commands.

2. Consistency and Repeatability

With the docker-compose.yml file, the configuration of the entire application is explicitly documented. This ensures that the deployment of the application is consistent and repeatable across environments such as development, test, and production. Anyone can quickly deploy the same application environment using the same configuration.

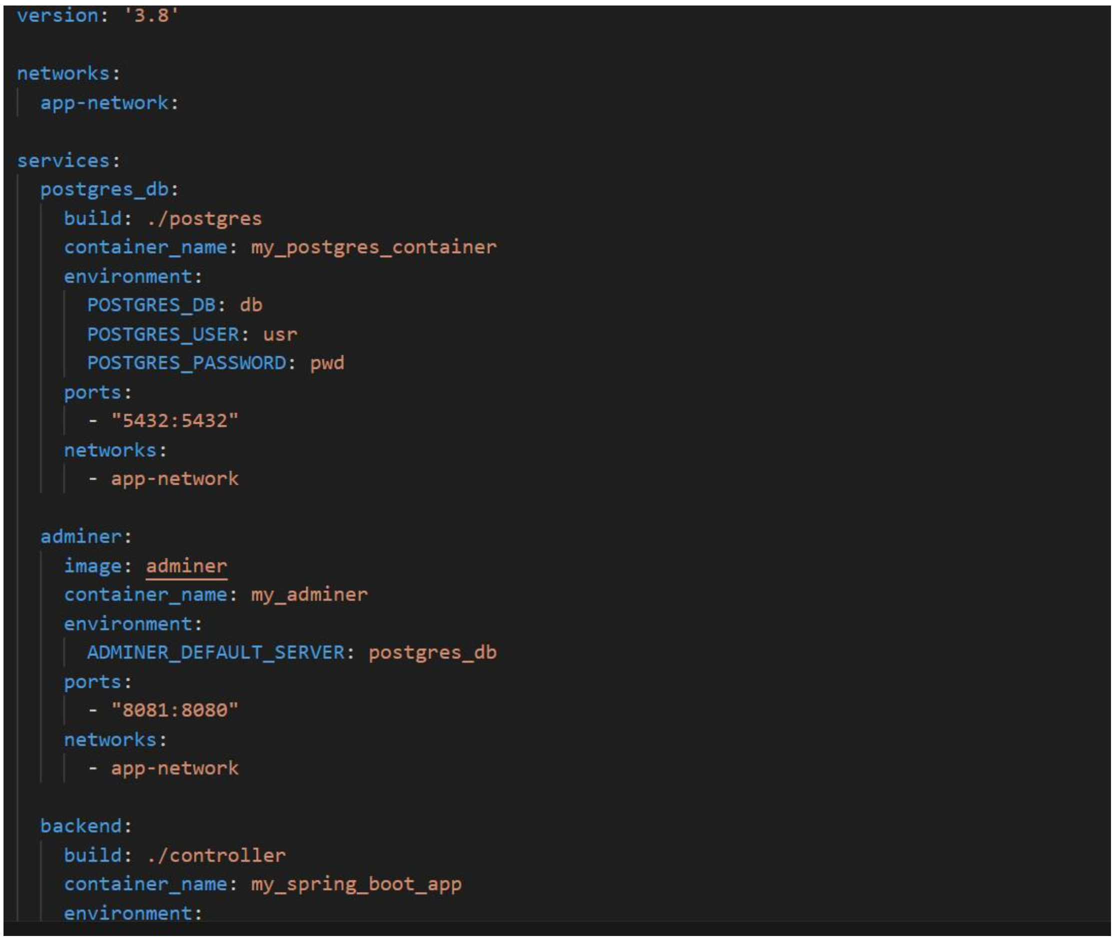
3. Network Management

Docker Compose automatically creates a default network for all containers in an application, which allows containers to communicate with each other via service names. There is no need to manually configure the network, making network management simple and intuitive.

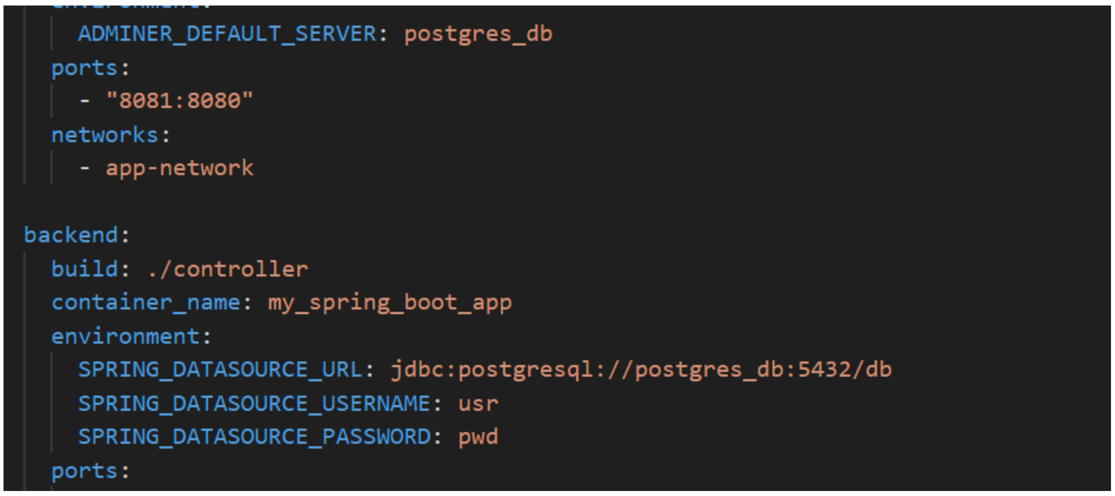


Question

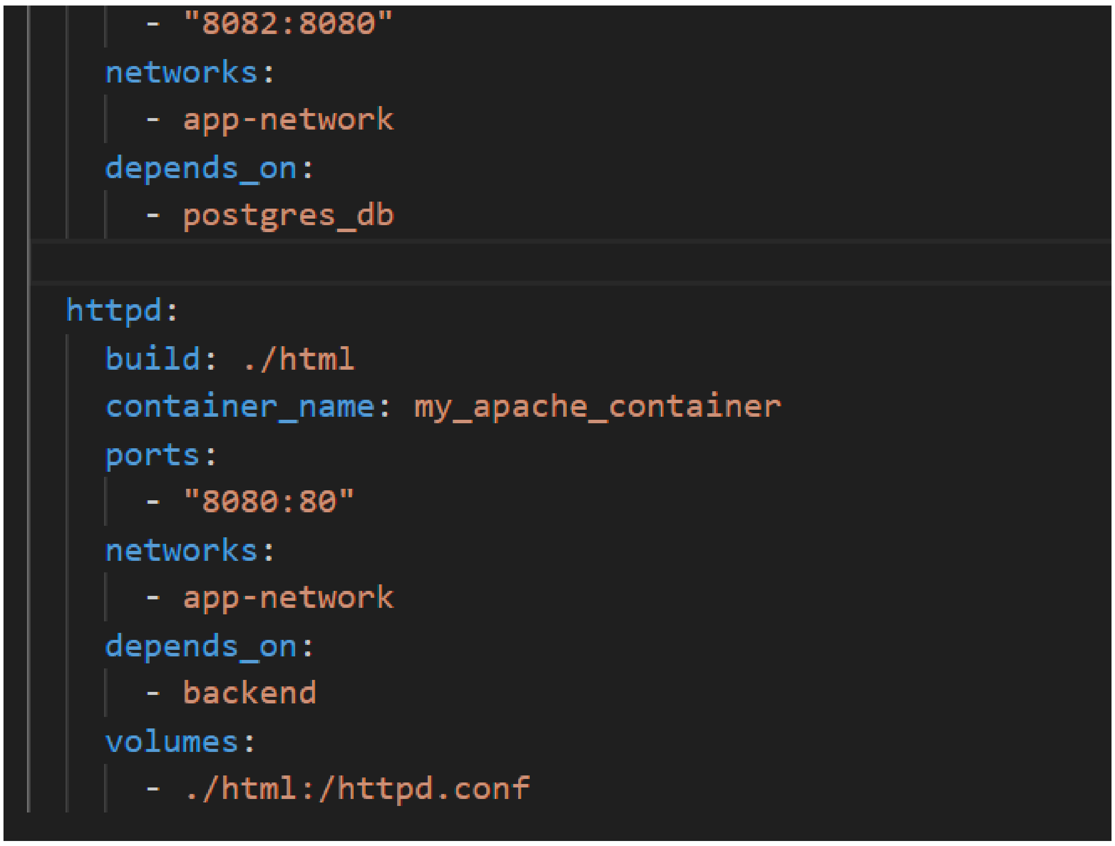
1-3 Document docker-compose most important commands. 1-4 Document your docker-compose file.



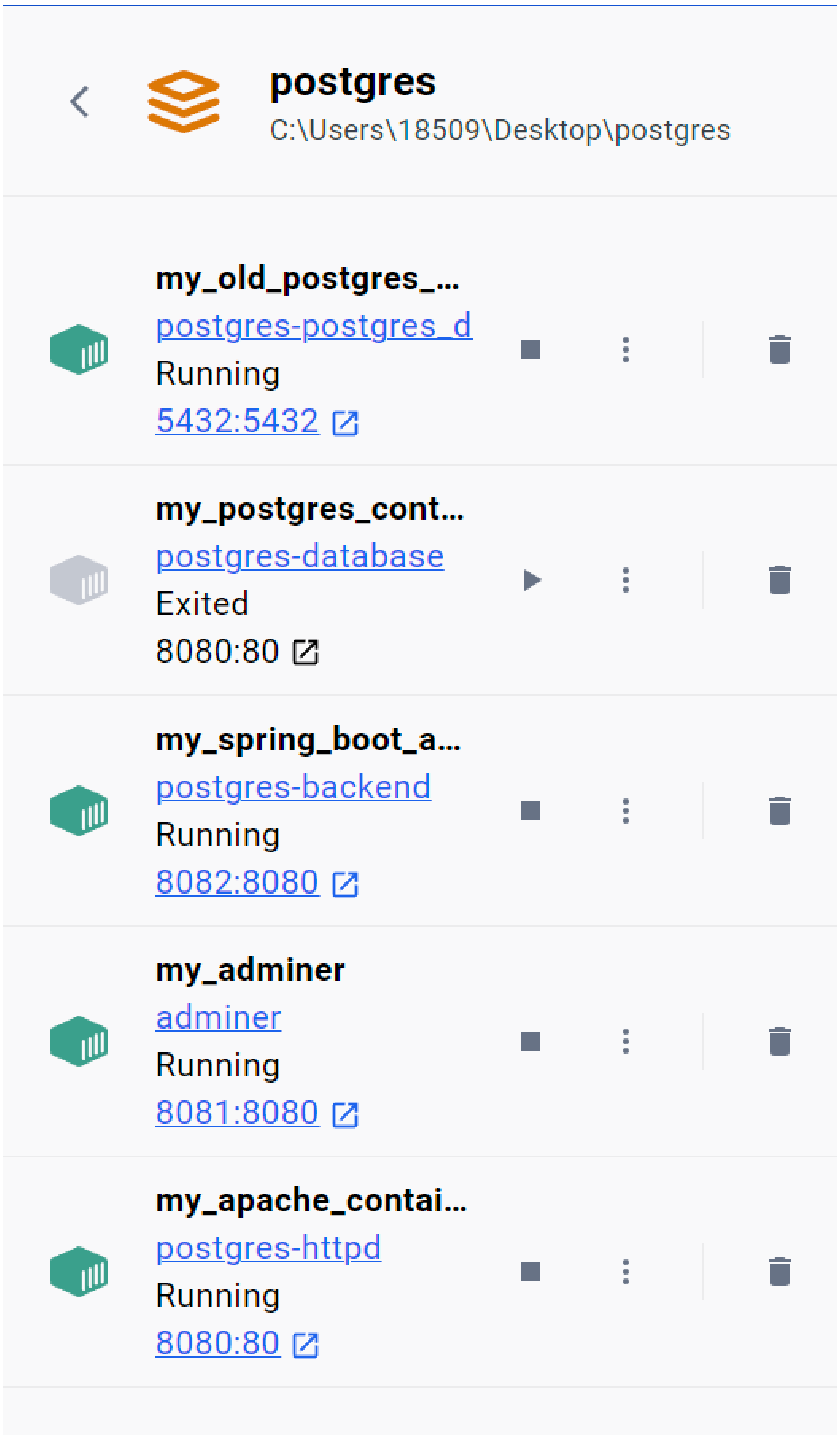
```
version: '3.8'

networks:
  app-network:

services:
  postgres_db:
    build: ./postgres
    container_name: my_postgres_container
    environment:
      POSTGRES_DB: db
      POSTGRES_USER: usr
      POSTGRES_PASSWORD: pwd
    ports:
      - "5432:5432"
    networks:
      - app-network

  adminer:
    image: adminer
    container_name: my_adminer
    environment:
      ADMINER_DEFAULT_SERVER: postgres_db
    ports:
      - "8081:8080"
    networks:
      - app-network

  backend:
    build: ./controller
    container_name: my_spring_boot_app
    environment:
```



```
      ADMINER_DEFAULT_SERVER: postgres_db
    ports:
      - "8081:8080"
    networks:
      - app-network

  backend:
    build: ./controller
    container_name: my_spring_boot_app
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres_db:5432/db
      SPRING_DATASOURCE_USERNAME: usr
      SPRING_DATASOURCE_PASSWORD: pwd
    ports:
```



```yaml
      - "8082:8080"
    networks:
      - app-network
    depends_on:
      - postgres_db

  httpd:
    build: ./html
    container_name: my_apache_container
    ports:
      - "8080:80"
    networks:
      - app-network
    depends_on:
      - backend
    volumes:
      - ./html:/httpd.conf
```



postgres
C:\Users\18509\Desktop\postgres

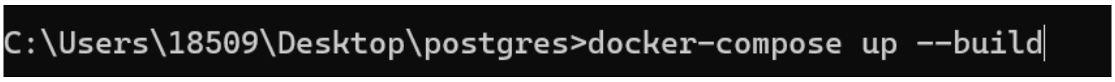
my_old_postgres_...
postgres-postgres_d
Running
5432:5432

my_postgres_cont...
postgres-database
Exited
8080:80

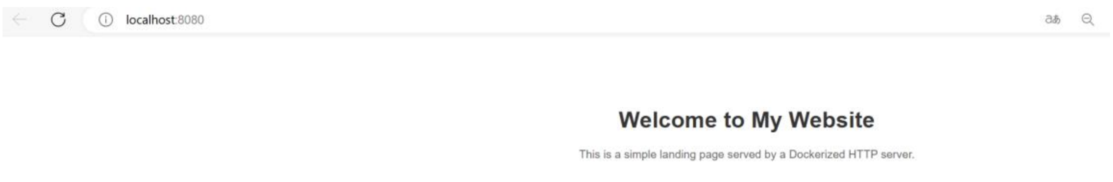
my_spring_boot_a...
postgres-backend
Running
8082:8080

my_adminer
adminer
Running
8081:8080

my_apache_contai...
postgres-httpd
Running
8080:80



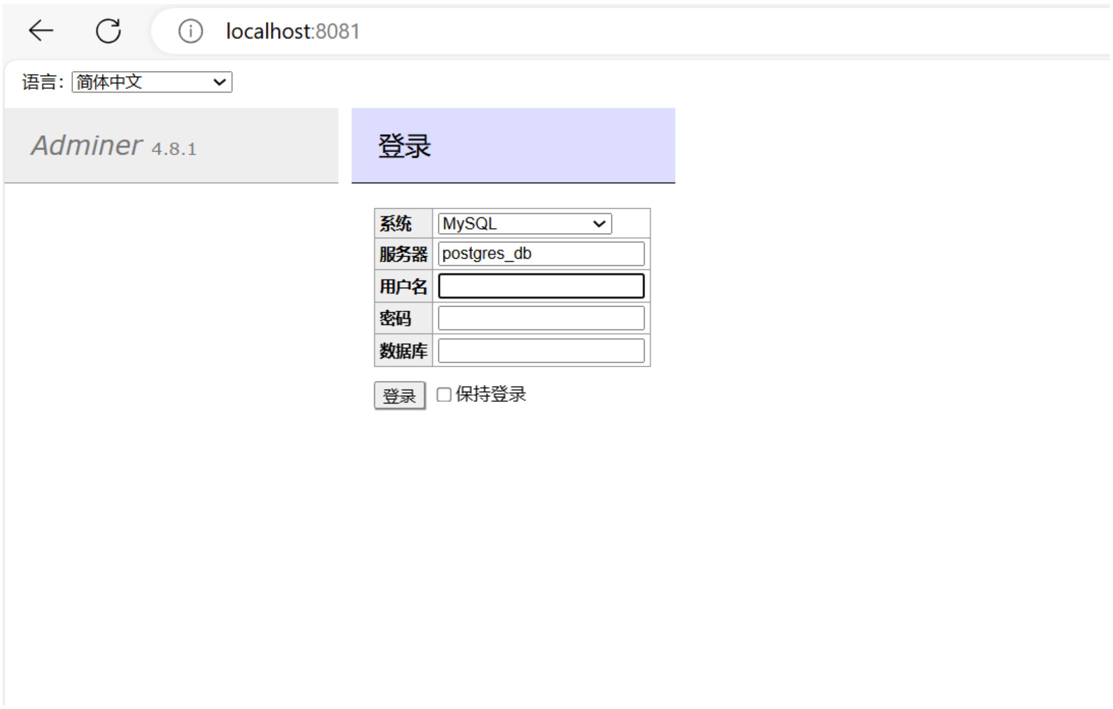
```
C:\Users\18509\Desktop\postgres>docker-compose up --build
```



localhost:8080

Welcome to My Website

This is a simple landing page served by a Dockerized HTTP server.



localhost:8081

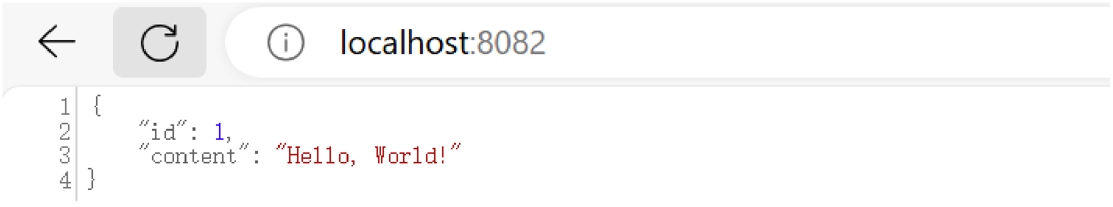
语言: 简体中文

Adminer 4.8.1

登录

系统	MySQL
服务器	postgres_db
用户名	
密码	
数据库	

登录 ☐ 保持登录



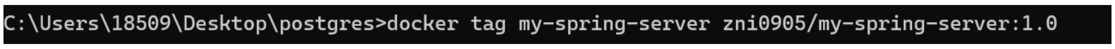
localhost:8082

```
{
    "id": 1,
    "content": "Hello, World!"
}
```

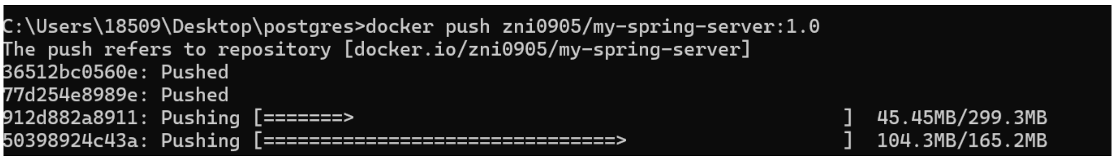
Publish

Question

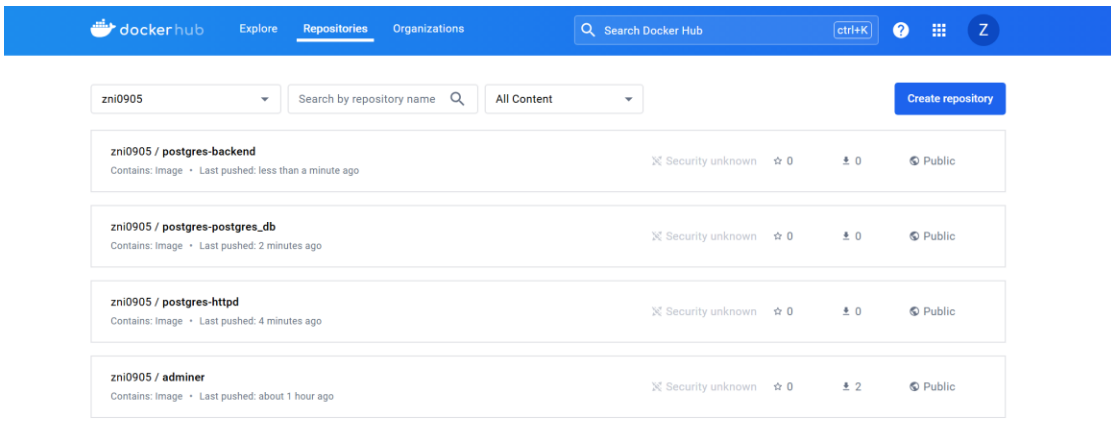
1-5 Document your publication commands and published images in dockerhub.



```
C:\Users\18509\Desktop\postgres>docker tag my-spring-server zni0905/my-spring-server:1.0
```



```
C:\Users\18509\Desktop\postgres>docker push zni0905/my-spring-server:1.0
The push refers to repository [docker.io/zni0905/my-spring-server]
36512bc0560e: Pushed
77d254e8989e: Pushed
912d882a8911: Pushing [=======>                                            ]  45.45MB/299.3MB
50398924c43a: Pushing [==============================>                     ]  104.3MB/165.2MB
```



zni0905 / postgres-backend
Contains: Image • Last pushed: less than a minute ago — Security unknown ☆ 0 ⬇ 0 Public

zni0905 / postgres-postgres_db
Contains: Image • Last pushed: 2 minutes ago — Security unknown ☆ 0 ⬇ 0 Public

zni0905 / postgres-httpd
Contains: Image • Last pushed: 4 minutes ago — Security unknown ☆ 0 ⬇ 0 Public

zni0905 / adminer
Contains: Image • Last pushed: about 1 hour ago — Security unknown ☆ 0 ⬇ 2 Public

Tip

Why do we put our images into an online repo?

Storing images in an online repository increases team productivity, simplifies the deployment process, and ensures image reliability and availability.

TP2 Github Action

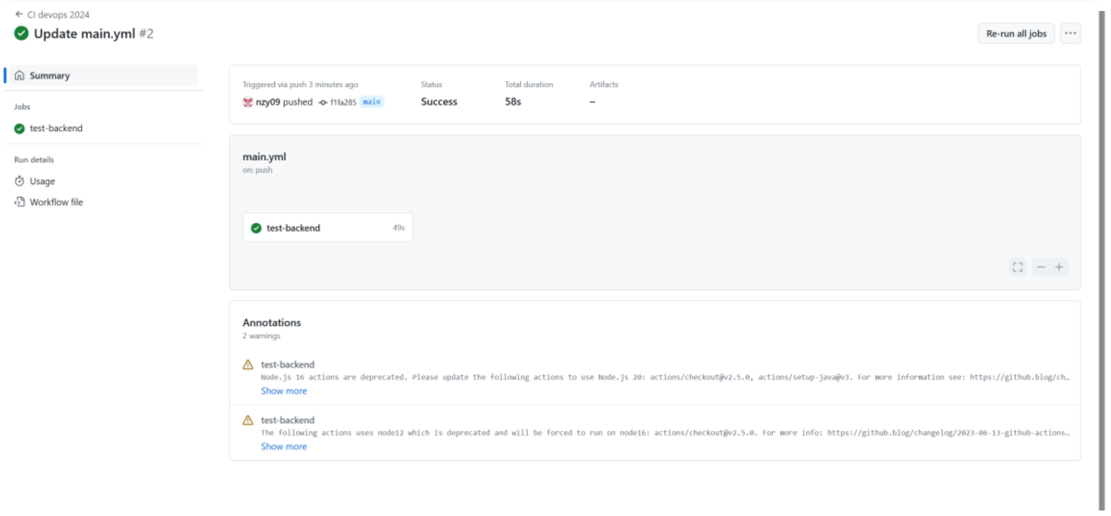
Question

2-1 What are testcontainers?

Testcontainers is a Java library for simplifying the use of Docker containers in testing. It provides flexible container management for integration testing, allowing you to use real databases, message queues, and other services in your test environment.

Question

2-2 Document your Github Actions configurations.



← CI devops 2024

Update main.yml #2

Re-run all jobs

Summary

Jobs
- test-backend

Run details
- Usage
- Workflow file

Triggered via push 3 minutes ago	Status	Total duration	Artifacts
nzy09 pushed f1fa285 main	Success	58s	–

main.yml
on: push

test-backend 49s

Annotations
2 warnings

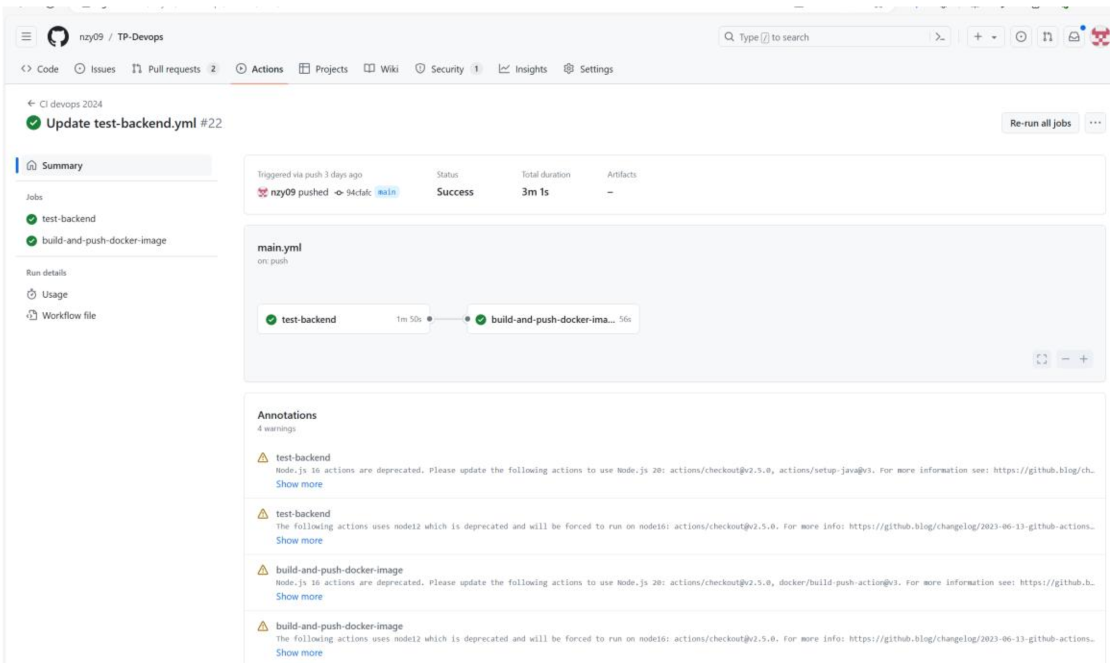
test-backend
```
Node.js 16 actions are deprecated. Please update the following actions to use Node.js 20: actions/checkout@v2.5.0, actions/setup-java@v3. For more information see: https://github.blog/ch…
```
Show more

test-backend
```
The following actions uses node12 which is deprecated and will be forced to run on node16: actions/checkout@v2.5.0. For more info: https://github.blog/changelog/2023-06-13-github-actions…
```
Show more

Note

For what purpose do we need to push docker images?

Pushing Docker images is crucial for enabling efficient sharing and collaboration, facilitating automated and scalable deployments, managing application versions and rollbacks, ensuring global distribution, providing disaster recovery, and enhancing security through vulnerability scanning and controlled access.



nzy09 / TP-Devops

Code · Issues · Pull requests 2 · Actions · Projects · Wiki · Security 1 · Insights · Settings

← CI devops 2024

Update test-backend.yml #22

Re-run all jobs

Summary

Jobs
- test-backend
- build-and-push-docker-image

Run details
- Usage
- Workflow file

Triggered via push 3 days ago	Status	Total duration	Artifacts
nzy09 pushed 94cfafc main	Success	3m 1s	–

main.yml
on: push

test-backend 1m 50s → build-and-push-docker-ima… 56s

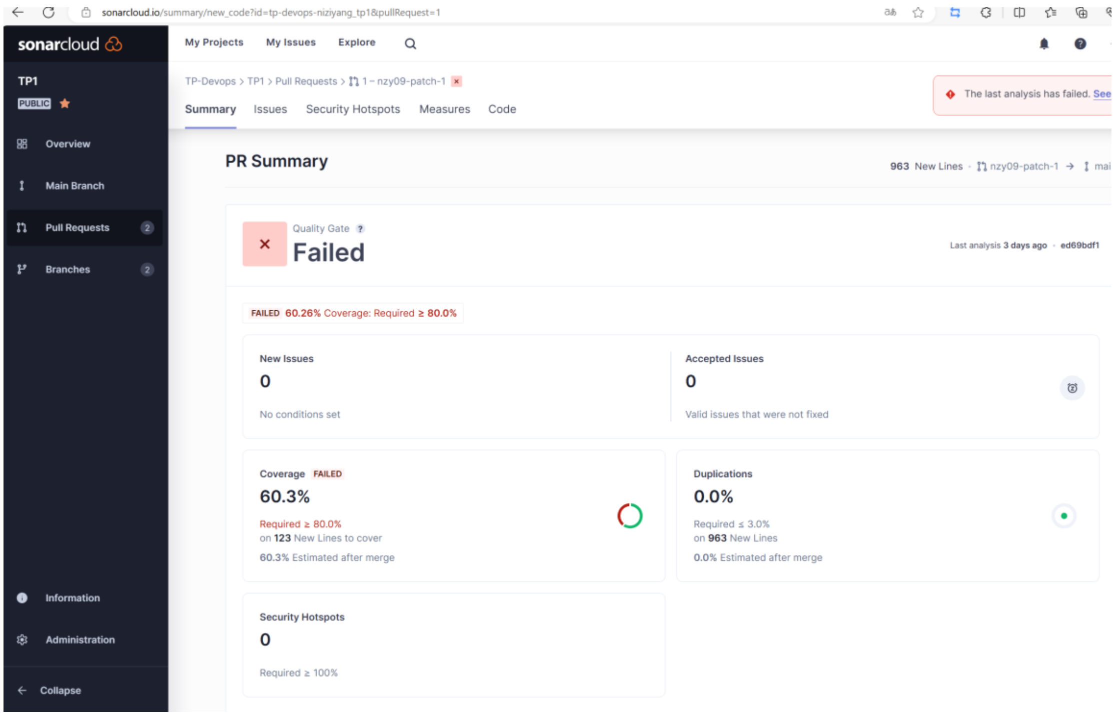
Annotations
4 warnings

test-backend
```
Node.js 16 actions are deprecated. Please update the following actions to use Node.js 20: actions/checkout@v2.5.0, actions/setup-java@v3. For more information see: https://github.blog/ch…
```
Show more

test-backend
```
The following actions uses node12 which is deprecated and will be forced to run on node16: actions/checkout@v2.5.0. For more info: https://github.blog/changelog/2023-06-13-github-actions…
```
Show more

build-and-push-docker-image
```
Node.js 16 actions are deprecated. Please update the following actions to use Node.js 20: actions/checkout@v2.5.0, docker/build-push-action@v3. For more information see: https://github.b…
```
Show more

build-and-push-docker-image
```
The following actions uses node12 which is deprecated and will be forced to run on node16: actions/checkout@v2.5.0. For more info: https://github.blog/changelog/2023-06-13-github-actions…
```
Show more


sonarcloud.io/summary/new_code?id=tp-devops-niziyang_tp1&pullRequest=1

My Projects · My Issues · Explore

TP1 PUBLIC

Overview · Main Branch · Pull Requests 2 · Branches 2 · Information · Administration · Collapse

TP-Devops > TP1 > Pull Requests > 1 – nzy09-patch-1

Summary · Issues · Security Hotspots · Measures · Code

The last analysis has failed. See

PR Summary

963 New Lines · nzy09-patch-1 → mai

Quality Gate
Failed

Last analysis 3 days ago · ed69bdf1

FAILED 60.26% Coverage: Required ≥ 80.0%

New Issues
0
No conditions set

Accepted Issues
0
Valid issues that were not fixed

Coverage FAILED
60.3%
Required ≥ 80.0%
on 123 New Lines to cover
60.3% Estimated after merge

Duplications
0.0%
Required ≤ 3.0%
on 963 New Lines
0.0% Estimated after merge

Security Hotspots
0
Required ≥ 100%

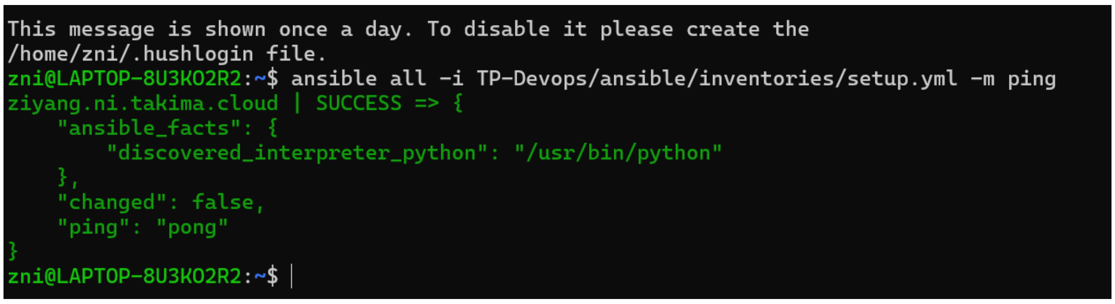
Question

Document your quality gate configuration.

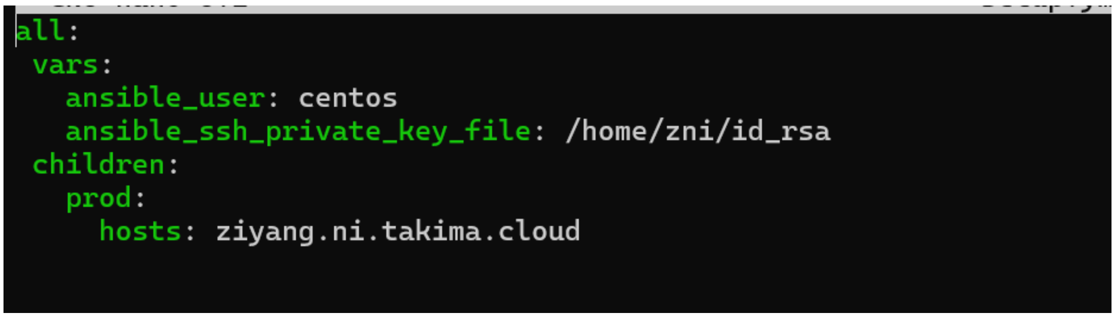
TP3 Discover Ansible

Question

3-1 Document your inventory and base commands


```
This message is shown once a day. To disable it please create the
/home/zni/.hushlogin file.
zni@LAPTOP-8U3KO2R2:~$ ansible all -i TP-Devops/ansible/inventories/setup.yml -m ping
ziyang.ni.takima.cloud | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
zni@LAPTOP-8U3KO2R2:~$ |
```

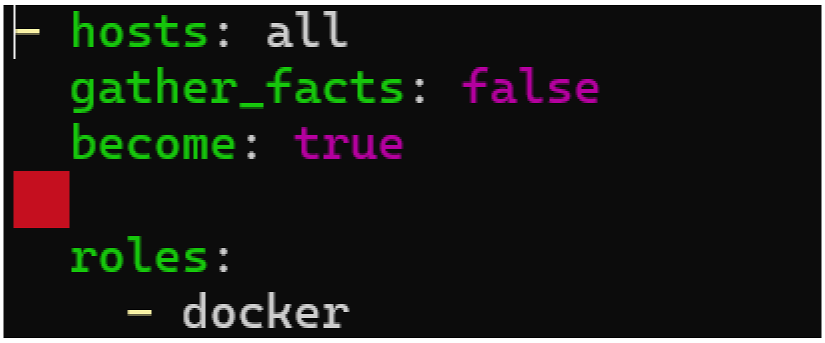
First playbook



```
all:
 vars:
   ansible_user: centos
   ansible_ssh_private_key_file: /home/zni/id_rsa
 children:
   prod:
     hosts: ziyang.ni.takima.cloud
```

Question

3-2 Document your playbook

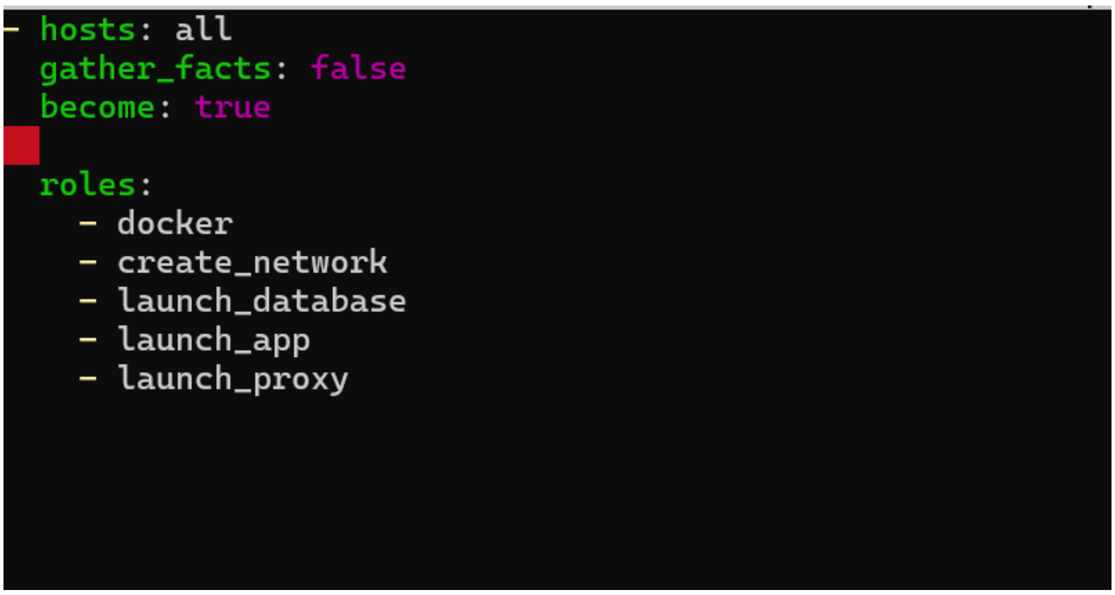


```
- hosts: all
  gather_facts: false
  become: true

  roles:
    - docker
```

Question

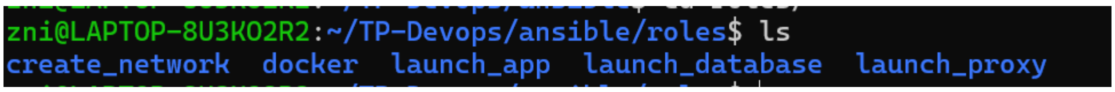
Document your docker_container tasks configuration.

Advanced Playbook



```
- hosts: all
  gather_facts: false
  become: true

  roles:
    - docker
    - create_network
    - launch_database
    - launch_app
    - launch_proxy
```

Using roles

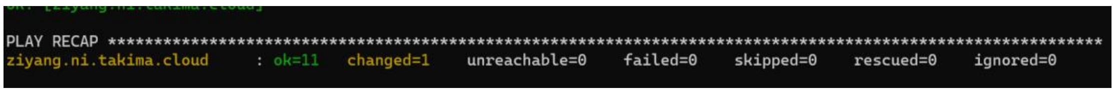


```
zni@LAPTOP-8U3K02R2:~/TP-Devops/ansible/roles$ ls
create_network  docker  launch_app  launch_database  launch_proxy
```

Deploy your App

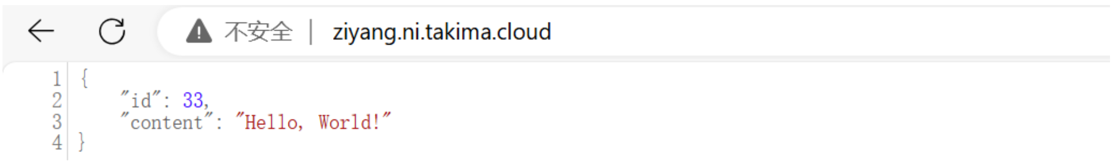
Question

Document your docker_container tasks configuration.



```
PLAY RECAP *********************************************************************************************************
ziyang.ni.takima.cloud     : ok=11   changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Front



```
{
    "id": 33,
    "content": "Hello, World!"
}
```