The application Drupal is selected here, and lab tasks are implemented on it. A brief summary of performed tasks, what commands have been used and how it got implemented is provided in the report.

TASK-1

The application Drupal is selected from the Docker Hub for the lab, The Drupal application is free and an open-source web content management framework, Acts as backend framework for websites. Here, we focused on showing Drupal functionality with user able to post an article or message on the website, the user be able to view listed messages/articles on the website. The application is using Postgres as the database.

The Drupal application comprises of both web server and database server, with the showing the existing articles and messages on the website for user and the option for the user to able to add/edit articles and messages as shown in figure 1.1.

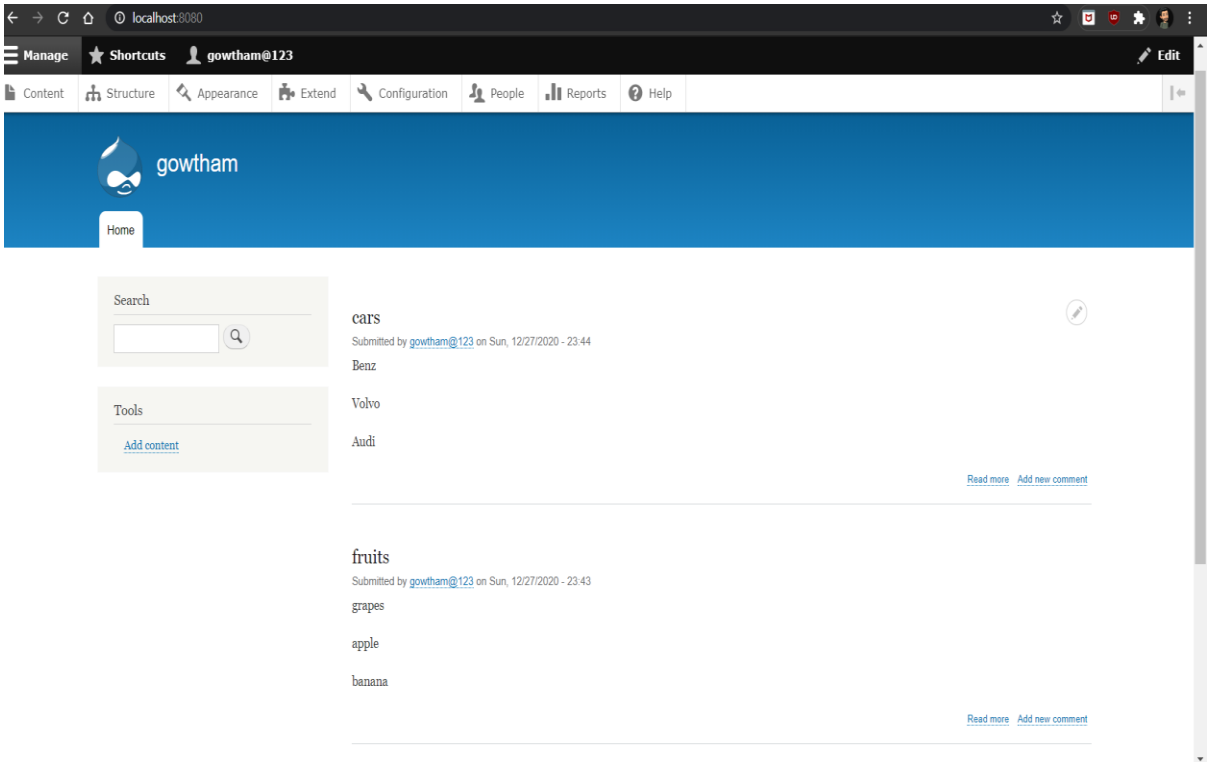

Figure 1.1: Drupal application

TASK-2

Firstly, we pull the Drupal application from the Docker hub. Using command

docker pull drupal

Secondly, the docker file for the drupal is provided to us from the docker hub. The docker file used is shown in the below figure 1.2.

```
MINGW64:/c/Users/govar
  GNU nano 4.9.3                                                                                              New Buffer
FROM php:7.4-fpm-alpine3.12

# install the PHP extensions we need
# postgresql-dev is needed for https://bugs.alpinelinux.org/issues/3642
RUN set -eux; \
        \
        apk add --no-cache --virtual .build-deps \
                coreutils \
                freetype-dev \
                libjpeg-turbo-dev \
                libpng-dev \
                libzip-dev \
                postgresql-dev \
        ; \
        \
        docker-php-ext-configure gd \
                --with-freetype \
                --with-jpeg=/usr/include \
        ; \
        \
        docker-php-ext-install -j "$(nproc)" \
                gd \
                opcache \
                pdo_mysql \
                pdo_pgsql \
                zip \
        ; \
        \
        runDeps="$( \
                scanelf --needed --nobanner --format '%n#p' --recursive /usr/local \
                        | tr ',' '\n' \
                        | sort -u \
                        | awk 'system("[ -e /usr/local/lib/" $1 " ]") == 0 { next } { print "so:" $1 }' \
        )"; \
        apk add --virtual .drupal-phpexts-rundeps $runDeps; \
        apk del .build-deps

# set recommended PHP.ini settings
# see https://secure.php.net/manual/en/opcache.installation.php
RUN { \
                echo 'opcache.memory_consumption=128'; \
                echo 'opcache.interned_strings_buffer=8'; \
                echo 'opcache.max_accelerated_files=4000'; \
                echo 'opcache.revalidate_freq=60'; \
                echo 'opcache.fast_shutdown=1'; \
        } > /usr/local/etc/php/conf.d/opcache-recommended.ini

# https://github.com/drupal/drupal/blob/9.0.1/composer.lock#L4052-L4053
COPY --from=composer:1.10 /usr/bin/composer /usr/local/bin/

# https://www.drupal.org/node/3060/release
ENV DRUPAL_VERSION 9.0.10

WORKDIR /opt/drupal
RUN set -eux; \
        export COMPOSER_HOME="$(mktemp -d)"; \
        composer create-project --no-interaction "drupal/recommended-project:$DRUPAL_VERSION" ./; \
        chown -R www-data:www-data web/sites web/modules web/themes; \
        rmdir /var/www/html; \
        ln -sf /opt/drupal/web /var/www/html; \
        # delete composer cache
        rm -rf "$COMPOSER_HOME"

ENV PATH=${PATH}:/opt/drupal/vendor/bin
```

Figure 1.2: Docker file

Now, the command **run** is used to run the image. The image is already built when the drupal is pulled. The command **run** activates the container. The port 80 is used as network port on the local host port 8080. The command is,

docker run –name lab01-drupal -p 8080:80 -d drupal

The command **inspect** is used to inspect the following image or container.

docker inspect lab01-drupal

Volumes: The data been generated by and used by the containers are stored in Volumes. The following is used to assign the volume to container over network bridge.

docker run –name gow-drupal1 -network bridge -d -v c:/var/www/html/modules\ drupal

The created volume remains even if the container is terminated.

Bind mount: It is similar to the way how volumes works but differ in the way memory is stored, it stores in host OS local drive.

docker run –name our-drupal1 -network bridge -d \-v /c/docker/modules:/var/www/html/modules drupal

Network: For the application bridge network is used for containers to communicate and for accessing the application.

The following command lists all networks

docker network ls

we can inspect the bridge network in detail by using following command.

docker network inspect bridge

TASK-3

Firstly, the docker-compose file with yml extension is taken, for the container orchestration. Both Drupal and Postgres are run together through the docker-compose file. The docker-compose file is provided below.

Version: ’3.1’

Services:

Drupal:

Image:drupal:8-apache

Ports:

-8080:80

Volumes:

-/var/www/html/modules

```
        -/var/www/html/profiles
        -/var/www/html/themes
        -/var/www/html/sites
    restart:  always
postgres:
  image:  postgres;10
  environment:
    Postgres_password:   example
  restart:  always
```

The following command used to run and stop the services using docker-compose file,

docker-compose up -d

docker-compose down

For manually scaling the services, we use the -scale flag as shown below.

docker-compose up -d -scale web=2

Docker Swarm: A Docker Swarm is a collection or group of machines configured to join together in a cluster that are either physical or virtual machines running the docker program. To control the cluster's activities, a swarm manager is used, and the machines in the cluster are known as nodes. We used the following command to generate a swarm manager node.

docker swarm init -advertise-addr < ip address >

for creating the worker nodes, the following command is used.

docker swarm join -token <token id>

for create and scale the services of the application, the following command is used.

docker service create -replicas 2 -p 8083:8080 drupal

Motivation: Initially, we found it difficult to understand the docker, how things worked around it. We tried out reading the docker documentation and videos to understand the docker concepts. A lot of times, we found the commands were not running how we intended them to be. After many trial or error, we able to successfully the run the commands and got the expected results.