School of Electrical Engineering and Computer Science
INFS3208 – Cloud Computing

Programming Assignment Task I (10 Marks)

Task description:

You are a developer at a leading software development company tasked with creating a scalable and efficient deployment solution for your company's PHP-based web applications. The solution uses Nginx as the web server, and PHP-FPM for server-side scripting, all managed within a single Docker container. Composer will be used for dependency management, similar to npm for JavaScript or pip for Python, enabling easy addition, updating, and removal of libraries through the `composer.json` file. The Dockerfile you create will be shared with PHP developers, who will seamlessly upload their PHP websites to the container and conduct testing. The container should support dynamic code updates and include health checks to ensure reliability.

Objectives:

1. **Provide Maintainer Information and Define PHP Version Variable (1 mark)**:

   - Include maintainer information (version, name, email) using appropriate Dockerfile instructions (0.5 mark).
   - Define a variable `PHP_FPM_VER` with a default value (8.2) for the PHP version, allowing developers to specify a different version when building the image (0.5 mark).

2. **Implement Multi-stage Build and Install Dependency using Composer (2 marks)**:

   - Use a multi-stage build (1 mark).
   - In the first stage, named `builder`, use Composer to install PHP dependencies defined in `composer.json` (provided in the `conf` folder) (1 mark). For the installation process, use the following options:
     - Install only production dependencies (no dev version).
     - Optimize the autoloader for better performance.
     - Run the installation without requiring user input.

   Note: Store the installed dependencies in the `vendor` folder to be used in the second stage.

3. **Choose an Alpine-based Base Image in the Second Stage (0.5 marks)**:

   - In the second stage, choose an Alpine-based image as the base for the final build and ensure Nginx and PHP-FPM are installed on the chosen base.

     Note: Ubuntu-based images (e.g. php:8.2-fpm) may still be preferred in certain scenarios where a larger default package set and a familiar environment are beneficial. However, Alpine-based images (e.g. php:8.2-fpm-alpine) are often chosen for their minimalism, smaller size, improved security, and better performance, making them a preferred choice for many Dockerized applications. In this task, you can **ONLY** use alpine-based images

as the base image for the final build. You can choose **either** an Alpine-based Nginx image **or** an Alpine-based PHP-FPM image as the base for this programming task.

4. **Configure Nginx for PHP-FPM (1 mark)**:

   - Ensure Nginx is configured to support PHP-FPM by copying the provided `nginx.conf` from the `conf` folder to `/etc/nginx/nginx.conf` (0.5 marks).
   - Install PHP database extensions: `pdo` and `pdo_mysql` (0.5 marks).
   - [Optional] Install process manager `supervisord` for managing nginx and php-fpm.

5. **Copy Application Files (0.5 marks)**:

   - Copy PHP files from the `src` folder to `/var/www/html` in the container (0.25 marks).
   - Copy the `vendor` folder (the installed PHP dependencies in the first stage) to `/var/www/vendor` (0.25 marks).

6. **Implement Health Checks (1.5 marks)**:

   - Use the `curl` command for health checks (`curl -f http://localhost/ || exit 1`) (0.5 marks).
   - The configurations of the checks are set to:
     - Wait 10 seconds before the first health check (0.25 marks).
     - Run health checks every 30 seconds (0.25 marks).
     - Allow each health check to complete within 10 seconds (0.25 marks).
     - Mark the container as unhealthy if the health check fails three times consecutively (0.25 marks).

7. **Configure and Expose the Container (2 marks)**:

   - Expose and publish port 80 for Nginx (0.5 marks).
   - Designate the root folder of the website as a volume, ensuring PHP files in the root folder sync instantly with changes in the host's mounted folder (0.5 marks).
   - Set the working directory to `/var/www/html` (0.5 marks).
   - Start Nginx and PHP-FPM, ensuring Nginx runs in the foreground (0.5 marks).

8. **Good Practices and Completeness (1 mark):**

   - In addition to the multi-stage build, follow good practices of writing a Dockerfile to make the built image as lean as possible (0.5 marks).
   - To test the completeness of your solution, visit the testing PHP webpage at `http://public_ip/index.php` (0.5 marks).

9. **Provide Docker Commands (0.5 marks)**:

   - Document the commands to build and run the container, tagging the image as `a1:<your surname>` and the container as `cca1_nginx_php`.
     Note: To assess your assignment correctly, you need to provide the related docker commands with appropriate options (e.g., *docker build [options]* and *docker run [options])* in **a separate txt file**. [**IMPORTANT**] Lack of the related docker commands may lead to the failure of the assessment, which will be **marked as a zero!**

Preparation:

In this individual coding assignment, you will put your Docker command skills, Dockerfile instruction knowledge (from Lecture 3), and relevant fields to practical use. The first step is to comprehend the task and understand the technical specifications required. Next, you should utilize Linux commands to install Nginx, Composer, PHP, and PHP-FPM on a Virtual Machine (VM), following the guidance provided in the Practical/Tutorial sessions. Lastly, these Linux commands need to be transformed into Dockerfile instructions to accomplish the task at hand. You can practice on either the VM from Google Cloud Platform (GCP) or your local machine (requiring Oracle VirtualBox or Windows Subsystem for Linux), depending on your access to GCP. Please refer to the Dockerfile writing example below for more detailed instructions.

Assignment Submission:

- You must compress the **Dockerfile** and **a txt file** that contains docker commands to properly build the image and run it.
- The name of the compressed file should be named as "FirstName_LastName_StudentNo.zip".
- You must make an online submission to Blackboard **before 3:00 PM on Friday, 29/08/2025**.
- Only one extension application could be approved due to medical conditions.

Main Steps:

Step 1:

Log in your VM and change to your home directory.

Step 2:

```
git clone https://github.com/csenw/cca1.git && cd cca1
```

Run this command to download the required configuration files and testing web pages.

```
uqteaching@instance-1:~$ git clone https://github.com/csenw/cca1.git && cd cca1
Cloning into 'cca1'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 23 (delta 4), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (23/23), 13.64 KiB | 1.95 MiB/s, done.
uqteaching@instance-1:~/cca1$
```

In this folder (cca1), please **write your Dockerfile**.

Step 3:

Build your Dockerfile and tag your image as a1:<your surname>.

```
 ---> df8ca524271d
Successfully built df8ca524271d
Successfully tagged a1:wang
```

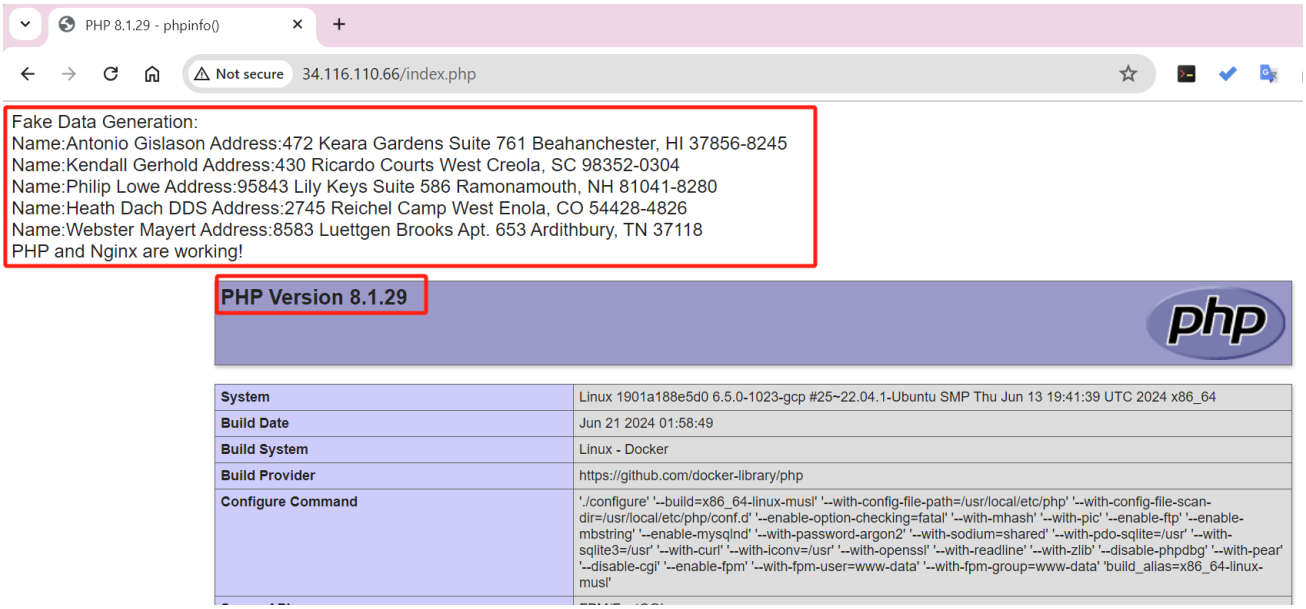
Step 4:

Run the image as a container ‘cca1_nginx_php’:

```
uqteaching@instance-1:~/cca1$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                                   NAMES
f976ae62abea   a1:wang    "/bin/bash -c '/usr/…"   2 minutes ago    Up 2 minutes    0.0.0.0:80->80/tcp, :::80->80/tcp       cca1_nginx_php
```

Step 5:

Test the container on your VM with the preloaded index.php.
PHP webpage test: `http://public_ip/index.php`



Note: If your docker works well, the dependency fakerphp should work and generate fake data (in the red box).

Appendix

Nginx (pronounced "engine X"), stylized as NGINX or Nginx or NginX, is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. The software was created by Igor Sysoev and publicly released in 2004. Nginx is free and open-source software, released under the terms of the 2-clause BSD license. A large fraction of web servers uses NGINX, often as a load balancer. Nginx can be deployed to serve dynamic HTTP content on the network using FastCGI, SCGI handlers for scripts, WSGI application servers, and it can serve as a software load balancer. Note that Nginx cannot directly deal with PHP but can serve PHP applications through the FastCGI protocol. Nginx employs PHP-FPM (FastCGI Process Manager) that is running in the background as a daemon and listening for CGI requests. Nginx uses an asynchronous event-driven approach, rather than threads, to handle requests. Nginx's modular event-driven architecture can provide more predictable performance under high loads. Nginx default configuration file is nginx.conf

PHP [2] is a general-purpose scripting language that is especially suited to web development. It was created by Danish-Canadian programmer Rasmus Lerdorf in 1994; the PHP reference implementation is now produced by The PHP Group. PHP originally stood for Personal Home Page, but it now stands for the recursive initialism PHP: Hypertext Preprocessor. PHP code is usually processed on a web

server by a PHP interpreter implemented as a module, a daemon or as a Common Gateway Interface (CGI) executable. On a web server, the result of the interpreted and executed PHP code – which may be any type of data, such as generated HTML or binary image data – would form the whole or part of an HTTP response. Various web template systems, web content management systems, and web frameworks exist which can be employed to orchestrate or facilitate the generation of that response. Additionally, PHP can be used for many programming tasks outside of the web context, such as standalone graphical applications and robotic drone control. Arbitrary PHP code can also be interpreted and executed via the command-line interface (CLI).

PHP-FPM (FastCGI Process Manager) [2] is an alternative FastCGI implementation for PHP, bundled with the official PHP distribution since version 5.3.3. When compared to the older FastCGI implementation, it contains some additional features, mostly useful for heavily loaded web servers. Figure 1 shows how PHP-FPM helps Nginx process PHP pages.

Figure 1. How PHP and Nginx work together (Image credit: DataDog)

Composer [3] is a dependency manager for PHP, similar to npm for JavaScript or pip for Python. It allows developers to declare the libraries their project depends on, and it manages (install/update) them for the project. Composer simplifies the process of managing third-party libraries and dependencies. By defining dependencies in a composer.json file, you can easily add, update, or remove libraries. Composer allows you to specify versions or version ranges for each dependency, ensuring that your project uses compatible library versions. By using a composer.lock file, Composer ensures that the same versions of libraries are installed every time, providing consistency across different environments (development, staging, production). Composer provides an autoloader for your PHP classes, making it easy to use namespaces and class autoloading without manually including files. Composer taps into the Packagist repository, which hosts thousands of PHP packages. This allows developers to find and integrate community-supported libraries and tools easily.

[1] Nginx, https://en.wikipedia.org/wiki/Nginx
[2] PHP, https://en.wikipedia.org/wiki/PHP
[3] Composer, https://getcomposer.org/