Docker

- Interactive shell (commandline)
- Install ping tools (for diagnostics)
- php8.2-fpm with nginx base
- Large default vdx
- Load image from command line
- Virtual Windows machine
- Step-by-Step GitHub Releases Workaround: dockur example

Interactive shell (commandline)

Sometimes it's difficult to understand problems within a docker. Like network issues or configurations that need to be adjusted.
It's quite easy to log into a docker via command line.

1) open a command line (it can be that you need to activate SSH access) on the hosting system

2) find the docker image you're interested in

```
docker ps
```

3) open a shell to the docker using the id from the previous query

```
docker exec -it c05876343 /bin/bash
```

Now you should be inside the docker. To leave the session just type

```
exit
```

Install ping tools (for diagnostics)

Ping is an utility to see if another host is reachable.
In some cases, dockers can be isolated and it can be daunting to try to understand who can see whom.

Using ping can help diagnosing some problems

Log into the docker (see interactive-shell-commandline)

And execute the following commands, so you can "ping"

```
apt-get update -y
apt-get install -y iputils-ping
```

php8.2-fpm with nginx base

Composed from different sources to fit my purpose.

Requires docker to be running on the machine.

This will create a docker image with PHP8.2 using Nginx - which is usable for Laravel.

It's easiest to put the dockerfile together the script ***Dockerbuild.bat*** and run the script.

Dockerfile

```
FROM ubuntu:latest AS base
ENV DEBIAN_FRONTEND noninteractive

# Install dependencies
RUN apt update
RUN apt install -y software-properties-common
RUN add-apt-repository -y ppa:ondrej/php
RUN apt update
RUN apt install -y php8.2\
    php8.2-cli\
    php8.2-common\
    php8.2-fpm\
    php8.2-mysql\
    php8.2-zip\
    php8.2-gd\
    php8.2-mbstring\
    php8.2-curl\
    php8.2-xml\
    php8.2-bcmath\
    php8.2-pdo

# Install php-fpm
RUN apt install -y php8.2-fpm php8.2-cli

# Install composer
RUN apt install -y curl
```

```
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

# Install nodejs
RUN apt install -y ca-certificates gnupg
RUN mkdir -p /etc/apt/keyrings
RUN curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o
/etc/apt/keyrings/nodesource.gpg
ENV NODE_MAJOR 20
RUN echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg]
https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | tee /etc/apt/sources.list.d/nodesource.list
RUN apt update
RUN apt install -y nodejs
RUN apt install -y git
# Install nginx
RUN apt install -y nginx
RUN echo "\
   server {\n\
      listen 80;\n\
      listen [::]:80;\n\
      root /var/www/BookStack/public;\n\
      add_header X-Frame-Options \"SAMEORIGIN\";\n\
      add_header X-Content-Type-Options \"nosniff\";\n\
      index index.php;\n\
      charset utf-8;\n\
      location / {\n\
         try_files \$uri \$uri/ /index.php?\$query_string;\n\
      }\n\
      location = /favicon.ico { access_log off; log_not_found off; }\n\
      location = /robots.txt  { access_log off; log_not_found off; }\n\
      error_page 404 /index.php;\n\
      location ~ \.php$ {\n\
         fastcgi_pass unix:/run/php/php8.2-fpm.sock;\n\
         fastcgi_param SCRIPT_FILENAME \$realpath_root\$fastcgi_script_name;\n\
         include fastcgi_params;\n\
      }\n\
      location ~ /\.(?!well-known).* {\n\
         deny all;\n\
      }\n\
   }\n" > /etc/nginx/sites-available/default
```

```
RUN echo "\
    #!/bin/sh\n\
    echo \"Starting services...\"\n\
    service php8.2-fpm start\n\
    nginx -g \"daemon off;\" &\n\
    echo \"Ready.\"\n\
    tail -s 1 /var/log/nginx/*.log -f\n\
    " > /start.sh

EXPOSE 80

CMD ["sh", "/start.sh"]
```

Dockerbuild.bat

With environment variables:

```
docker build --no-cache -t phpnginx:latest .
docker save phpnginx > phpnginx.tar
docker run --env-file .env -d -p 5050:80 phpnginx:latest
```

example .env

```
# Database details
DB_HOST='127.0.0.1' #change with  yours
DB_PORT='234' # change with yours
DB_DATABASE='yourdbname'
DB_USERNAME='yourdbuser'
DB_PASSWORD='yourpass'
```

Without environment variables:

```
docker build --no-cache -t phpnginx:latest .
docker save phpnginx > phpnginx.tar
docker run -d -p 5050:80 phpnginx:latest
```

Large default vdx

```
>> wsl --shutdown
```

Verify everything is stopped by:

```
>> wsl.exe --list --verbose
```

Then start diskpart:

```
>> diskpart
```

and inside diskpart type:

```
DISKPART> select vdisk file="<path to vhdx file>"
```

it should respond by saying `DiskPart successfully selected the virtual disk file.`

Then to shrink

```
DISKPART> compact vdisk
```

Load image from command line

docker load -i imagename.tar

Virtual Windows machine

docker run -d --name windows_vm -p 8006:8006 -p 3389:3389/tcp -p 3389:3389/udp --device=/dev/kvm --cap-add=NET_ADMIN -e VERSION="11" -e RAM_SIZE="4G" -e CPU_CORES="2" --stop-timeout 120 dockurr/windows

docker:admin

Step-by-Step GitHub Releases Workaround: dockur example

When you can't use docker pull or the github workspace is blocked by your IT department.

Step-by-Step GitHub Releases Workaround
1. Create the Action Workflow File

1. Open your regular web browser on your Windows host.
2. Go to your GitHub profile and create a **New Repository**. Make it **Private** (e.g., `my-image-fetcher`).
3. Inside your repo, click on the **Actions** tab at the top.
4. Click the link that says **"set up a workflow yourself"**.
5. Delete all default text in the editor and paste this configuration exactly:

yaml

```
name: Export Docker Image to Releases
on: [workflow_dispatch]

permissions:
  contents: write

jobs:
  bundle_image:
    runs-on: ubuntu-latest
    steps:
      - name: Pull and Save Image
        run: |
          docker pull ghcr.io/dockur/windows:latest
          docker save -o windows_image.tar ghcr.io/dockur/windows:latest

      - name: Create Private Release Asset
        uses: softprops/action-gh-release@v2
```

```
      with:
        tag_name: v1.0.0
        name: "Windows Container Download"
        files: windows_image.tar
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

6. Click **Commit changes...** in the top right, then confirm it.

---

2. Trigger the Automated Build

1. Go back to the **Actions** tab.
2. On the left side, click **"Export Docker Image to Releases"**.
3. On the right side, click the **Run workflow** dropdown, and then click the green **Run workflow** button.
4. Wait about **3 to 5 minutes**. GitHub's remote cloud servers will download the image and package it into a single `.tar` archive asset. A green checkmark will appear when it finishes.

---

3. Download the Tar via the Browser

1. Click the main logo of your repository to go back to its home page.
2. Look at the right-hand sidebar. Under the **Releases** section, you will see a fresh `v1.0.0` tag. Click on it.
3. Under the "Assets" heading, you will see **`windows_image.tar`**.
4. Left-click it. Your web browser will download it as a standard static file asset, bypassing the terminal blocks entirely.

---

4. Import the File into local Docker

Once your browser finishes downloading the archive to your Windows machine:

1. Drag or cut-paste `windows_image.tar` out of your Downloads folder and into `C:\PROJECTS\CobolSim\`.
2. Open your local WSL terminal and run:
   bash

```
cd /mnt/c/PROJECTS/CobolSim
docker load -i windows_image.tar
```

3. Verify it is ready by typing `docker images`.
