Docker Image for Grav (in development)

TL;DR

[Updated: 07.12.2022]

Index

1.0 Prerequisites

This project is cloned from the official GRAV GitHub repository. If you want work with me, feel free to download it from my GitHub repository.

It contains the original packages based on PHP 8.1:

ORIGINAL	PACKAGES
docker-ce	vim editor
apache-2.4.38	php8.1
GD library	php8.1-opcache

ORIGINAL	PACKAGES
Unzip library	php8.1-acpu
cron	php8.1-yaml

In addition other packages are included:

REQUIRED	PACKAGES			
ca-certificates	php8.1_pdo	openssh-client	yq (>= 4.29)	wget (>= 1.20)
ccache	php8.1_pdo_mysql	rsync	uuid (>= -v4)	
iputils-ping	php8.1_pdo_pqsql	sudo	openssl (>= 1.1.1)	
net-tools	php8.1-pgsql	tree (>= 1.8.0)	git (>= 2.17)	
dropbear	php8.1_xdebug	jq (>= 1.5)	getssl (>= 2.32)	

> NOTE: Please ensure that the PHP base docker image is an actual version (e.g. php8.1) and not End-of-Life, otherwise application security is highly impacted.

1.1 Packages

This project needs the following prerequisites on the HOST machine:

- Install at least uuid -v4 >= 1.5 (macOS: brew install ossp-uuid) (Ubuntu: sudo apt install uuid) (Alpine: sudo apk add -U ossp-uuid)
- Install at least jq >= 1.5 (macOS: brew install jq) (Ubuntu: sudo apt install jq) (Alpine: sudo apk add -U jq)
- Install at least yq >= 4.29 (macOS: brew install yq) (Ubuntu: sudo apt install yq) (Alpine: sudo apk add -U yq)
- Install at least openssl >= 1.1.1 (macOS: brew install openssl@1.1) (Ubuntu: sudo apt install openssl) (Alpine: sudo apk add -U openssl)
- Install at least docker-ce >= 20.10 (Ubuntu: sudo apt install docker-ce) (Alpine: sudo apk add -U docker) (See https://docs.docker.com/engine/install) or Docker Desktop for Mac which includes docker-ce
  (macOS: brew install --cask docker)
- Install at least docker buildx plugin >= 0.5.0 (See https://docs.docker.com/buildx/working-with-buildx)
  NOTE: It is already included in a recent version of docker.
- Install at least getssl >= 2.32 (See https://github.com/srvrco/getssl) (macOS: `curl --silent https://raw.githubusercontent.com/srvrco/getssl/master/getssl > getssl && chmod 700 getssl && sudo mv getssl /usr/local/bin`)
- Install at least git 2.x >=2.17 (macOS: brew install git) (Ubuntu: sudo apt install git) (Alpine: sudo apk add -U git)
- Install tree >=1.8.0 (macOS: brew install tree) (Ubuntu: sudo apt install tree) (Alpine: sudo apk add -U tree)
- Install vim (macOS: brew install vim) (Ubuntu: sudo apt install vim) (Alpine: sudo apk add -U vim)
- Install wget (macOS: brew install wget) (Ubuntu: sudo apt install wget) (Alpine: sudo apk add -U wget)

- Install vscode for development (macOS: brew install vscode)
- Add the following vscode extensions: - Docker - EditorConfig for VS Code - Remote - WSL - Remote - Containers - Remote - SSH - Remote - SSH:Editing

This prerequisites are checked automatically with `mkinit.sh init`. Execute it with `${GRAV_HOME}/bin/mkinit.sh init`. After that reload your shell with `source ${HOME}/.bashrc`, now the home path variable is `${GRAV_HOME}`.

> NOTE: If you have installed all the above mentioned packages earlier, you can update it with:
>
> - **brew upgrade `**
> - **getssl --upgrade**.

2.0 Proje t stru ture

The project consists of different directories, each one has a specific role:

```
${GRAV_HOME}
|-- [ ]  bin                |-- (Directory for bash scripts)
|-- [*]  cache              |-- (Directory for cache files)
|-- [*]  cfg                |-- (Directory for config files)
|-- [*]  cert               |-- (Directory for certificate files)
|-- [*]  data               |-- (Directory for data files)
|-- [ ]  docker             |-- (Directory for docker files)
|-- [*]  key                |-- (Directory for SSH & user keys)
|-- [ ]  lib                |-- (Library for shell scripts)
|-- [*]  rootfs             |-- (Repository for packages and files)
|-- [*]  .context
|-- [ ]  .dockerignore
|-- [ ]  .editorconfig
|-- [ ]  .gitattributes
|-- [ ]  .gitignore
|-- [ ]  Dockerfile -> ./docker/Dockerfile
`-- [ ]  README.md
```

> **Note:** The files in directories marked with `[*]` are not uploaded to Git. They must be build with the appropriate `<PROJECT_ROOT>/bin/grav-mk*` script.
>
> To initialize the project, execute `./bin/grav-mkinit.sh init` first from the `${GRAV_HOME}` directory, then activate it with `source ${HOME}/.bashrc`. From this moment you can also use the short form without appending `.sh`, e.g. `grav-mkinit`.

2.1 Project features

This project includes the following features:

- Use docker init as signal forwarder and process reaper
- Use docker runlets for easier dockerfile development
- Use local context key/value files for project configuration settings `${GRAV_HOME}/.context.*`
- Use local cache directory for injecting files at buildtime `${GRAV_HOME}/rootfs/*`

- Use some sophisticated bash shell scripts for build, runtime and configuration `${GRAV_HOME}/bin/grav-*.sh`
- Use the latest docker buildx builder for external docker image storage `--cache-from, --cache-to` in a local directory `${GRAV_HOME}/cache/.ccache`
- Use the latest docker buildx builder for specific platform builds, here `linux/amd64`
- Ability to create a named user `grav` with SSH keys for vscode development over remote SSH over port 2222
- Ability to create a user password for SSH login securely
- Ability to create and add the SSH keys for automatic logins and cache retrieval from local or remote host securely
- Ability to create SSL certificates from letsencrypt.org with getssl
- Use external certificate volume for certificate persistence
- Use external cache volume for faster C/C++ compilation `${GRAV_HOME}/cache/.ccache`
- Use external cache volume for faster PHP compilation `${GRAV_HOME}/cache/.phpcache`
- Mount a docker named volume `grav_data` to a specific host directory `${GRAV_HOME}/data`
- Create a repository `${GRAV_HOME}/rootfs/tmp/grav/core` for caching grav core packages
- Use a bunch of local bash shared library `libgrav*` for all local bash scripts

2.2 Work in progress

- (WIP) Install PHP xdebug for vscode debugging over remote xdebug port
- (WIP) Support letsencrypt SSL keys with getssl bash script
- (TBD) Create a multistage dockerfile with base, compile and release stage
- (TBD) Implement multiarch images wit QEMU static support
- (TBD) Create an alpine container for smaller footprint
- (TBD) Use NGINX instead of Apache web server
- (TBD) Use buildx with docker composer file
- (TBD) Ability to install grav skeletons and plugins

3.0 Installation pro edure

- Install the prerequisite software (See Prerequisites
- Download the project with git `git clone https://github.com/giminni/docker-grav`
- Change into the current project directory with `cd docker-grav`
- `docker-grav` is now your `<PROJECT_HOME>` directory
- Initialize the project with `<PROJECT_HOME>/bin/grav-mkinit.sh init`
- Reload bash shell with `source ${HOME}/.bashrc`
- Set the current grav core production and development package version with `grav-core.sh set all`, older grav core packages version can be set manually, for example with `grav-core.sh set 1.6.0` for production package version or `grav-core.sh set 1.7.0-rc.19` for development package version.
- Download the grav core production packages with `grav-core.sh get all grav` or the core development packages with `grav-core.sh get all grav-admin`, older grav core packages can be set manually, for example with `grav-core.sh get 1.6.0 grav` for production package version or `grav-core.sh get 1.7.0-rc.19 grav-admin` for development package version.
- Create the encrypted password for user `grav` with `grav-mkpass.sh <user-password> grav`, the password must contain at least 11 characters

- Create new or use your own SSH private and public key with `grav-mkssh.sh <email-address>` by answering with `1` for create new SSH key or `2` for use own SSH key. The latter case will copy the key from your `${HOME}/.ssh` directory.
- Create the cache directory with `grav-mkcache.sh cache`
- Build the docker image with `grav-build.sh grav grav-admin testing` for the development version or `grav-build.sh grav` for the production version.
- Create the data directory with `grav-mkdata.sh data`
- Create the certificate directory with `grav-mkcert.sh cert`
- Run the docker image with `grav-run.sh grav grav-admin testing` for the development version or `grav_run.sh grav` for the production version.
- Enter the command line of the running grav image, with `grav-shell.sh grav-admin` for the development version or `grav-shell.sh grav` for the production version.

3.1 Installation checklist

- Check if scripts are available by entering `grav-` and pressing the TAB-key.
- Check aliases from the command line with `alias`.
- Check libraries from the command line with `func`.
- Check if the `.context` file is created in the project directory with `cat ${GRAV_HOME}/.context`.
- Check if the configuration directory `cfg` is populated with `.config.*` files with `ls -las ${GRAV_HOME}/cfg`.
- Check `grav_pass.key` file under the key directory `key` with `cat ${GRAV_HOME}/key/grav_pass.key`.
- Check if the SSH keys exists with `ls -las ${GRAV_HOME}/key/grav_rsa*` if you are using the `rsa` algorithm. Other algorithm that can be used are `dsa` and `ecdsa`.
- Check if the grav core file was downloaded correctly into the `rootfs` directory, with `ls -las ${GRAV_HOME}/rootfs/tmp/grav/core`.
- Check if the cache directories exists with `ls -las ${GRAV_HOME}/cache`. A subdirectory `.ccache` and `.phpcache` must exists, otherwise the `grav-build.sh` script does not start.
- Chek if the certificate directory exists, with `ls -las ${GRAV_HOME}/cert`.
- Check if the docker grav image exists, with `sudo docker images`.
- Check if the docker grav image is running, with `sudo docker ps -a`.

3.2 Using local key/value files for configuration

To persist some project configuration data a couple of key/value files are created in the `${GRAV_HOME}/cfg` directory. A `${GRAV_HOME}/.context` file will be generated with `<PROJECT_HOME/bin/grav-mkinit.sh init` at init time holding the configuration directory where all configuration files are stored.

E.g. `.context` file in `${GRAV_HOME}/` directory:

```
GRAV_CTX="${GRAV_HOME}/cfg"
```

E.g. `.config.bin` file in `${GRAV_HOME}/cfg` directory:

```
GRAV_BIN="${GRAV_HOME}/bin"
```

> **Note:** Every configuration files can be changed manually by expert user or use the handy local bash scripts that starts with `${GRAV_HOME}/bin/grav-mk*.sh` for novice user.

3.3 Using docker multiarch environment

Using the extended docker build features of `buildx` this project is prepared for multiarch images. That means it uses one name for different target architectures `linux/amd64, linux/arm64, linux/armv7, ...`. Currently only the `linux/amd64` architecture is supported.

3.4 Using local docker cache repository

In addition to the build and compile cache environment, there is another local directory `./${GRAV_HOME}/rootfs/*` that holds cached artefacts. This directory can be used to store for example the grav core zip files to reduce bandwith and avoid a lengthy download time from the internet.

In this case store the `grav-admin.zip` file under `${GRAV_HOME}/rootfs/tmp`. If the name is correct the file will be inserted into the docker buildtime context and used instead of downloading the file from the internet.

3.5 Handling user password and SSH secrets

The extended docker build features of `buildx` allows injecting sensitive data without leaving any history trace. The user password is generated externally with openssl `SHA512` encryption by a provided bash script `${GRAV_HOME}/bin/mkssh.sh`. The encrypted password is then stored under `${GRAV_HOME}/key/grav_pass.key` and injected into the container at buildtime.

The same thing occures for the SSH private and public key. The key are stored under `${GRAV_HOME}/key/grav_rsa` and `${GRAV_HOME}/key/grav_rsa.pub` respectively.

> **Note:** Ensure that the SSH keys and user match the SSH keys of an external user on the local or remote host. Otherwise the user autologin over SSH and cache synchronization over github, rsync does not work.

3.6 Caching docker buildtime

The extended docker build features of `buildx` allows to store the docker buildtime cache into a local project directory `${GRAV_HOME}/cache/.dcache`. This can be of course changed to push/pull from a pubic/private registry if needed.

3.7 Running services as non-root user

To increase the overall security the privilege for required services (SSH, Cron and Apache) are deescalated to a non root user (grav). This is realized with `su-exec` `dropbear` and `go-cron`.

3.8 Persisting build cache using ccache and rsync

`CCache` and `rsync` are used to speedup the building of PHP extensions. At buildtime and before the PHP compilation is started, the external cache directory `${GRAV_HOME}/cache/.ccache` is read with `rsync` into the docker container `<CONTAINER_ROOT>/tmp/.ccache`. CCache will reroute the compiler call to this specific directory for faster compilation. Before all build artefacts are removed the cache directory `<CONTAINER_ROOT>/tmp/.ccache` is exported with `rsync` using incremental backup to preserve the compiled data for a next build `${GRAV_HOME}/cache/.ccache`.

> **Note:** Ensure that the SSH keys and user match the SSH keys of an external user on the local or remote host.

3.9 Working with vscode locally or remotely

To avoid direct access to the docker container a SSH user is fully provided and configured. The SSH server is listening on port `2222` to avoid collision with other primary SSH server. Point your vscode remote-SSH plugin to the localhost host or to the designated IP address and port `2222` to access the docker image for development.

3.10 Managing a container from the command line

There are a couple of local bash scripts to create, run and delete a container:

- `grav-build.sh` is used for building a container
- `grav-core.sh` is used to set and get the grav core packages locally
- `grav-run.sh` is used for running a container
- `grav-shell.sh` is used for accessing the command line inside a container
- `grav-purge.sh` is used for deleting all docker cached data, container and image artefacts.

4.0 Configuring a ontainer from the ommand line

The following data is needed to be able to build or run a container:

- Grav binary directory path `.config.bin`, e.g. `GRAV_BIN=${GRAV_HOME}/bin"`
- Grav cache directory path `.config.cache`, e.g. `GRAV_CACHE="${GRAV_HOME}/cache"`
- Grav certificate directory path `.config.cert`, e.g. `GRAV_CERT="${GRAV_HOME}/cert"`
- Grav config directory path `.config.cfg`, e.g. `GRAV_CFG=${GRAV_HOME}/cfg"`
- Grav data volume directory path `.config.data`, e.g. `GRAV_DATA="${GRAV_HOME}/data"`
- Grav development core version `.config.dev`, e.g. `GRAV_DEV=1.7.0-rc.20`
- Grav docker directory path `.config.docker`, e.g. `GRAV_DOCK=${GRAV_HOME}/docker"`
- Grav home directory path `.config.home`, e.g. `GRAV_HOME=${GRAV_HOME}"`
- Grav key directory path `.config.key`, e.g. `GRAV_KEY="${GRAV_HOME}/key"`
- Grav library directory path `.config.lib`, e.g. `GRAV_LIB="${GRAV_HOME}/lib"`
- Grav password file `.config.pass`, e.g. `GRAV_PASS="${GRAV_HOME}/key/grav_pass.key"`
- Grav production core version `.config.prod`, e.g. `GRAV_PROD=1.6.1`
- Grav rootfs directory path `.config.root`, e.g. `GRAV_ROOT="${GRAV_HOME}/rootfs"`
- Grav SSH key directory path `.config.ssh`, e.g. `GRAV_SSH=${GRAV_HOME}/key/grav_rsa"`
- Grav username `.config.user`, e.g. `GRAV_USER=grav`

This information is stored into local project connfig files that begins with `${GRAV_HOME}/cfg/.*`. To insert this data locally some local bash scripts are used `grav-mk*`. Every file is filled with a default value, however

feel free to change it to suite your needs.

- `${GRAV_HOME}/bin/grav-build.sh` = Build the grav docker image from the specified values
- `${GRAV_HOME}/bin/grav-core.sh get`= Download the corresponding production/development core file into `${GRAV_HOME}/rootfs` directory
- `${GRAV_HOME}/bin/grav-mkcache.sh` = Configures the local cache volume path `${GRAV_HOME}/cache/*`
- `${GRAV_HOME}/bin/grav-mkcert.sh` = Configures the local certificate volume path `${GRAV_HOME}/cert`
- `${GRAV_HOME}/bin/grav-mkdata.sh` = Configures the local data volume path `${GRAV_HOME}/data`
- `${GRAV_HOME}/bin/grav-mkinit.sh` = Initialize project, must run first. (See Installation procecure)
- `${GRAV_HOME}/bin/grav-mkpass.sh` = Configures the named container user and password
- `${GRAV_HOME}/bin/grav-mkssh.sh` = Configures the SSH private and public files for rsync, git, ...
- `${GRAV_HOME}/bin/grav-purge.sh` = Remove all grav artefacts, build cache, container and images
- `${GRAV_HOME}/bin/grav-run.sh` = Run the grav docker container from the specified values
- `${GRAV_HOME}/bin/grav-core.sh set` = Configures the grav production/development core version string
- `${GRAV_HOME}/bin/grav-shell.sh` = Access the container locally by opening a shell

> **Note:** Please consult the usage information of each local bash script by executing the command without arguments.

4.1 Downloading files to be cached into the rootfs directory

To be able to create the project in offline situation or minimize the download time from the internet, two tasks must be executed:

- Define wich grav version is needed to be installed from the grav download site using a local script `${GRAV_HOME}/bin/grav-core.sh set`. Insert as first argument `prod` or `dev`. To download a specific version use `<PROJECT_HOME/bin/grav-core.sh get`. Use the same arguments like `${GRAV_HOME}/bin/grav-core.sh set`

E.g. to download a specific version of grav-admin core `1.6.0` enter:

```
${GRAV_HOME}/bin/grav-core.sh get 1.6.0 grav-admin
```

> **Note:** The files are stored into the `${GRAV_HOME}/rootfs/tmp`. To reduce the container size, remove all superfluous artefacts before starting the build.

4.2 Persisting data into an external storage

To save the Grav site data to the host file system (so that it persists even after the container has been removed), simply map the container's `/var/www/html` directory to a named Docker volume `data`. This

named docker volume `data` is mapped into the project directory on the host `${GRAV_HOME}/data`.

> **Note:** If the mapped directory or named volume is empty, it will be automatically populated with a fresh install of Grav the first time that the container starts. However, once the directory/volume has been populated, the data will persist and will not be overwritten the next time the container starts.

4.3 Building the image from Dockerfile

To build the image from the command line a local bash script `${GRAV_HOME}/bin/grav-build.sh` is used.

This script as a lot of presetted arguments. The first argument is mandatory if not set, the script emits a usage string.

Here an example, how to create a user `grav` and build the latest grav+admin development package.

```
${GRAV_HOME}/bin/grav-build.sh grav grav-admin testing
```

Here an example how to create a user `grav` and build the latest grav+admin production package. Observe that the last two arguments are omitted while presetted.

```
${GRAV_HOME}/bin/grav-build.sh grav
```

Here the complete usage string of `${GRAV_HOME}/bin/grav-build.sh` script:

```
${GRAV_HOME} $ ./bin/grav-build.sh
grav-build: Error: Arguments are not provided!

grav-build:  Args: grav-build.sh grav_user [grav_imgname] [grav_tagname]
[grav_passfile] [grav_privfile] [grav_pubfile]
grav-build:  Note: (*) are default values, (#) are recommended values

grav-build: Arg1:   user-name: any|(#)         - (#=grav)"
grav-build: Arg2:  [img-name]: grav|grav-admin - (*=grav)"
grav-build: Arg3:  [tag-name]: latest|testing  - (*=latest)"
grav-build: Arg4: [pass-file]: any|(*)         - (*=
<PROJECT_HOME>/key/grav_pass.key)"
grav-build: Arg5: [priv-file]: any|(*)         - (*=
<PROJECT_HOME>/key/grav_rsa)"
grav-build: Arg6:  [pub-file]: any|(*)         - (*=
<PROJECT_HOME>/key/grav_rsa.pub)"

grav-build: Info: grav-build.sh grav grav-admin latest
/home/rpiadmin/Workspace/docker-grav/key/grav_pass.key
/home/rpiadmin/Workspace/docker-grav/key/grav_rsa
/home/rpiadmin/Workspace/docker-grav/key/grav_rsa.pub
```

```
grav-build: Help: grav-build.sh: Builds the docker file from some entered
arguments. (See Note, Info and Args)
```

5.0 Running the image from Do kerfile

To run the image from the command line a local bash script `${GRAV_HOME}/bin/grav-run.sh` is needed. This script as a lot of presetted arguments. The first argument is mandatory if not set the script emits a usage string. The default run mode is `n`ormal if there is a need to start only a bash command line and test something inside, run with the `d`ebug flag set.

Here an example how to run as user `grav` and use the latest `grav-admin` development package in debug mode.

```
${GRAV_HOME}/bin/grav-run.sh grav grav-admin testing d
```

Here an example how to run as user `grav` and use the **latest** `grav-admin` production package. Observe that the last two arguments are omitted while presetted with `n`ormal and `data`.

```
${GRAV_HOME}/bin/grav-run.sh grav grav-admin latest
```

Here the complete usage string of `${GRAV_HOME}/bin/grav-run.sh` script:

```
${GRAV_HOME} $ ./bin/grav-run.sh
grav-run: Error: Arguments are not provided!

grav-run:  Args: grav-run.sh grav_user [grav_imgname=grav]
[grav_imgtag=latest] [grav_voldata=data]
grav-run:  Note: (*) are default values, (#) are recommended values

grav-run:  Arg1:  user-name: any|(#) - (#=grav)
grav-run:  Arg2: [img-name]: any|(*) - (*=grav-admin)
grav-run:  Arg3: [tag-name]: any|(*) - (*=latest)
grav-run:  Arg4: [run-mode]: n|d|(*) - (*=(n)ormal|(d)ebug)
grav-run:  Arg5: [vol-data]: any|(*) - (*=data)

grav-run:  Info: grav-run.sh grav grav-admin latest n data

grav-run:  Help: grav-run.sh: Instantiate a docker container depending
from some entered arguments. (See Note, Info and Args)
```

IF you installed the `grav-admin` package then point the browser to `http://localhost:9080/admin` and create a user account, otherwise point the browser to `http://localhost:9080/` directly.

> **Note:** The following external <-> internal docker ports are exposed:

> - `2222 <-> 22`: for SSH external host access using the named user
> - `9080 <-> 80`: for HTTP external host access
> - `9443 <-> 443`: for HTTPS external host access (WIP)

The docker image has the following scheme:

- `<grav-user=grav>/<grav-name=<grav|grav-admin>:<grav-tag=latest|testing>`

E.g. `grav/grav:latest` for production images or `grav/grav-admin:testing` for development images.

6.0 Referen es

- Grav v1.7 Documentation
- Docker multiple architectures
- Working with buildx
- Visual Studio Code Tips & Trics
- Visual Studio Code macOS Shortcuts
- Visual Studio Code Linux Shortcuts
- Visual Studio Code Windows Shortcuts