NAVIGATING THE ARCHITECTURE

TABLE OF CONTENTS

HOST NAVIGATION

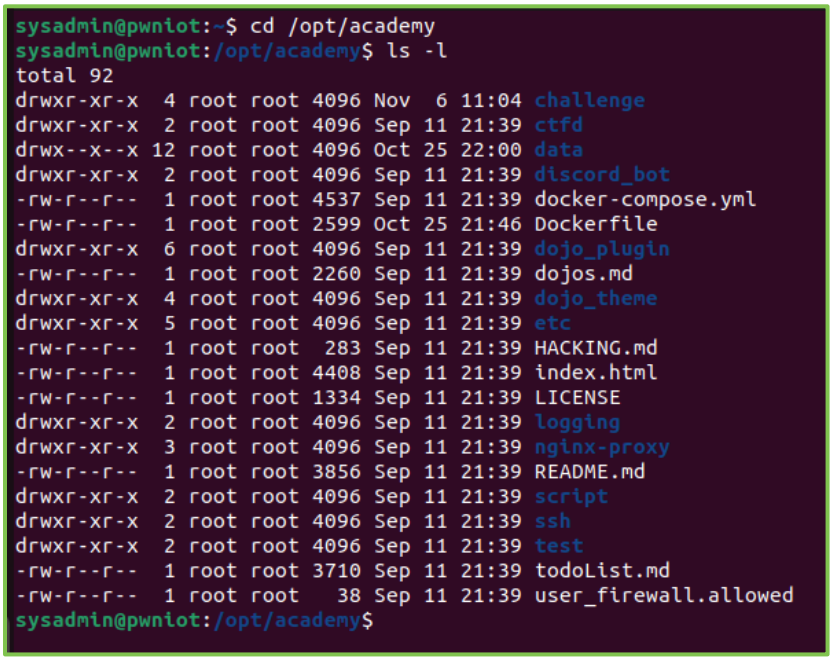
This guide assumes that the platform has been installed (see the "Installing PwnIoT Locally" document).

In the virtual machine, open a terminal. Use it to ensure the platform has been started:

- `docker start pwniot.academy`

In the terminal, navigate to the following directory, which is where the platform's files are located after installation, and list the files using the following:

- `cd /opt/academy`
- `ls -l`



ENTERING THE PRIMARY CONTAINER

While on the host machine, run the following command to enter a shell into the primary container.

- `docker exec -it pwniot.academy bash`

THE PRIMARY CONTAINER

Within the root folder of the repository are two major files that act as the entry point to building the architecture: `Dockerfile` and `docker-compose.yml`.

DOCKERFILE

The `Dockerfile` defines the primary image to build the initial "pwniot.academy" container from. This can be used to prepare data for the eventual nested containers, as well as default data for the user home directories.

Importantly, it will expose three ports: 22 (SSH), 80 (HTTP), and 443 (HTTPS).

BUILDING HOME DIRECTORIES

At the end of the `Dockerfile`, the "start" command in the `/script/dojo` script is executed, which in turns run the `/script/container-setup.sh` script. Of note, that latter script will initialize the default user home filesystem, which is subsequently copied as the current user's home directory whenever that user starts a challenge for the very first time.

Everything from the `/etc/skel` folder **in the docker image** (not from the repository's `/etc/skel` folder) is copied into the default home filesystem; files can be added to that folder via the `Dockerfile`. That is, additional content added to `/etc/skel` in the repository will not be automatically copied over. There is currently no built-in mechanism for modifying existing user home directories after-the-fact.

BUILDING NESTED CONTAINERS

After the initial setup, the `/etc/systemd/system/pwn.college` service is started, which builds and starts the nested containers from `docker-compose.yml`.

REBUILDING

The following command can be used to pull from GitHub and rebuild the architecture:

- `docker exec pwniot.academy dojo update`

DOCKER-COMPOSE.YML

`docker-compose.yml` defines the multitude of nested containers that will be used within the primary container. Once the primary container is set up, the nested containers will follow. This file that would be modified to add additional containers, though the image would have to be rebuilt.

SECONDARY CONTAINERS

LISTING THE SECONDARY CONTAINERS

Enter the primary container. The containers nested within the primary container may be listed with the following:

- `docker container ls`

CHALLENGE CONTAINERS (USER_X)

Enter these containers from the pwniot.academy container using the following, replacing **x** with the ID of the user on the CTFd platform:

- `docker exec -it user_x bash`

Additionally, any running challenges can be entered from the host using the following, replacing **x** with the ID of the user on the CTFd platform:

- `docker exec pwniot.academy dojo enter x`

These containers are initially defined as the "challenge" service in docker-compose.yml. Each user that initiates a challenge will cause a new challenge container to spawn. Multiple challenge containers may be running at the same time, at most one for each user. The names of the containers will follow the pattern of **user_x**, where **x** is the ID of the user on the CTFd platform.

`/challenge/Dockerfile` from the repository will additionally be used for building the image. This is where, for instance, the desktop background or default shell for the user can be changed.

CTFD CONTAINER (CTFD)

Enter this container from the pwniot.academy container using the following:

- `docker exec -it ctfd bash`

Additionally, the Flask shell can be directly accessed from the host machine using the following:

- `docker exec pwniot.academy dojo flask [args]`

This container holds the web server that runs Flask[1] on port 8000 (see the nginx-proxy container for the reverse proxy on port 80 & 443). Importantly, this container builds from the CTFd 3.4.0 source[2], which was pulled from GitHub in the primary Dockerfile to the primary container's `/opt/CTFd` folder (which is additionally copied to

[1] https://flask.palletsprojects.com/en/3.0.x/
[2] https://github.com/CTFd/CTFd/releases/tag/3.4.0

the nested CTFd container's `/opt/CTFd` folder). That source includes the administrative back-end that is otherwise not found within the PwnIoT repository.

Instead, the `/dojo_plugin` and `/dojo_theme` folders from the PwnIoT repository will be linked to `/opt/CTFd/CTFd/plugins/dojo_plugin` and to `/opt/CTFd/CTFd/themes/dojo_theme` respectively. Many other files and folders are copied from the primary container to the CTFd container. Check the "volumes" section of the "ctfd" service in `docker-compose.yml` for more.

The `/dojo_plugin` and `/dojo_theme` folders will be where additional web files can be placed.

When a file is uploaded via Flask, by default it will be placed within the `/var/uploads` folder in the CTFd container. That folder's contents are linked to the host's `/opt/academy/data/CTFd/uploads` folder for the ease of backing up.

DATABASE CONTAINER (DB)

Enter this container from the pwniot.academy container using the following:

- `docker exec -it db bash`

Additionally, the MariaDB shell can be directly accessed from the host machine using the following:

- `docker exec pwniot.academy dojo db [args]`

This container holds the database server running MariaDB 10.4.12. The default MariaDB credentials are as follows:

- **Root Username**: root
- **Username**: ctfd
- **Password**: ctfd
  - For both the root user and the ctfd user.
- **Database**: ctfd

CACHE CONTAINER (CACHE)

Enter this container from the pwniot.academy container using the following:

- `docker exec -it cache bash`

This container holds the Redis cache server, which is used by the web server for caching.

NGINX PROXY CONTAINER (NGINX)

Enter this container from the pwniot.academy container using the following:

- `docker container exec -it nginx bash`

This container uses nginx-proxy[3] as a reverse-proxy to serve files from the CTFd container's HTTP server. This container is where ports 80 and 443 are exposed from. The nginx-certs container accompanies it to aid in serving HTTPS certificates.

NGINX CERTIFICATES CONTAINER (NGINX-CERTS)

Enter this container from the pwniot.academy container using the following:

- `docker container exec -it nginx-certs bash`

This container uses acme-companion[4] to aid in generating HTTPS certificates, which are then served by the nginx-proxy container.

DISCORD BOT CONTAINER (DISCORD-BOT)

Enter this container from the pwniot.academy container using the following:

- `docker container exec -it discord-bot bash`

This container runs a Discord bot using discord.py[5], which is used for logging and grading. Environment variables, including the client, bot, and guild data, are defined within the `docker-compose.yml` file. The bot's source code itself can be found in `/discord_bot/bot.py` within the PwnIoT repository.

[3] https://github.com/nginx-proxy/nginx-proxy
[4] https://github.com/nginx-proxy/acme-companion
[5] https://github.com/Rapptz/discord.py