Bloodhound CE and Docker

0ut3r.space/2024/04/22/bloodhound-ce-and-docker

hoek

April 22, 2024

Yo yo yo my dear readers. A chaotic article today about several things at once. Because why not.

I wasn't sure if I could handle this month's article, if I'd write something meaningful or just a quick entry in the worth checking series. Whenever I have a busy month, I always leave an entry until the last minute and then something falls out, like this month's health problems and surgery in the next few days. Fortunately I'm not dying yet. But if you don't see the next entry next month, that means I've died. Well, it will happen one day anyway. But to the point.

I generally use virtual machines for everything, or install binaries or compile from source as a last resort. I was also ignorant about containers. I mean, I have a container manager installed on my NAS (Synology) and sometimes I would test something there, download images or pre-built containers and install tools into them or use pre-built solutions. Most of the time I would only use a container as a last resort, when I knew there would be a lot of weird dependencies, packages, junk and complications compared to installing on my own home system. No one wants to mess up their main system for the sake of testing, only to have to reinstall a dying, overgrown system a year later. For some things, a virtual machine is too much, and a small container will do. In addition, many solutions, applications and services are increasingly being offered as ready-made containers - no unnecessary configuration, just download, run and it works out of the box (container).

Many developers use containers, for app developing and testing, but there are also people who containerise everything they use every day to separate applications from the system. Crazy, but if it works and someone is using it, then why not.

I recently had to revisit the Bloodhound tool. I discovered that a new version of the community edition has been created. Even on the official repository it says "*This repository will be archived in the near future*". To get the latest version of Bloodhound, you can follow this link to the BloodHound Community Edition repository. My guess is that it's all about money, that someone has taken over Bloodhound and that there will be an enterprise version and a community version. But I haven't looked into it because it won't change anything, and if I can use the community version, why bother. The latest release of the official version for today is 4.3.1 from 23 May 2023 and the community version is 5.8.1 from 12 April 2024. I immediately thought, ok, I'll download the binaries and install them. It's just that the binaries aren't quite there yet. You can compile from source or use a container. That's how I got interested in containers, again.

I want Bloodhound on Kali because that's where I had the previous version. Here are instructions on how to do it the old way. My virtual machine of the inept bounty hunter is based on Kali, so that's where I decided to install Docker.

In Windows it's trivial, you install Docker Desktop and that's it, the console commands are the same plus there's a graphical overlay. I'll be honest, I don't know how to use it and I felt lost there, so after a few tests I found I'd sit in the console because it was easier for me to learn a few commands than click in gui. The Docker Desktop itself on Windows is also a lot for one machine, and it's worth having good hardware if you want to run lots of different containers at the same time.

Docker on Kali

In Linux it is even easier. Two commands:

```
sudo apt install -y
docker.io
sudo apt install docker-
compose
```

You can also use `docker-ce` from the official Docker repository, but nowadays `docker.io` in Debian-based distros and `docker-ce` are the same. In the past, the Debian version was outdated but this has changed.

Anyway if you for some reason want to use `docker-ce` follow these steps.

Add repository:

```
echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian bookworm stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list
```

import key

```
curl -fsSL
https://download.docker.com/linux/debian/gpg |
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

update and install:

```
sudo apt update
sudo apt install -y docker-ce docker-ce-cli
containerd.io
```

If you really want to know what's the difference between the two, check out this great answer on Stack Overflow. Yes, it's still the best place for answers, not just GPT chat.

Docker and Docker Composer

A brief explanation if you are a beginner like me.

Docker and Docker Compose are both tools related to application containerization, but they differ in their scope and functionalities:

1. **Docker**:
    - Docker is a platform for building, deploying, and running applications in containers.
    - It allows for building, running, and managing containers.
    - It enables creating container images using Dockerfiles and building and sharing images via Docker Hub.
    - It allows for managing individual containers and their configurations.
2. **Docker Compose**:
    - Docker Compose is a tool for defining and running multi-container applications.
    - It allows for defining multiple containers as part of a single application and managing them as a whole.
    - It enables defining application configurations in a YAML file, including various containers, networks, volumes, dependencies, etc.
    - It ensures that all containers in the application are started and connected properly.

In summary, Docker is a general-purpose platform for containerizing applications, whereas Docker Compose is a tool more focused on managing multi-container applications and their configurations.

Docker commands and Bloodhound

Now let's learn the Docker using Bloodhound as an example.

For my Docker apps I like to keep each configuration separate like:

```
Documents/
└── Docker/
    ├── BloodHound/
    │   └── docker-
compose.yml
    ├── OtherApp/
    │   └── docker-
compose.yml
    └── ...
```

As you know from the previous sections, we need a configuration file. Here is the file for Bloodhound. Download it to the folder.

It looks like:

```
# Copyright 2023 Specter Ops, Inc.
#
# Licensed under the Apache License, Version 2.0
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# SPDX-License-Identifier: Apache-2.0

version: '3'
services:
  app-db:
    image: docker.io/library/postgres:13.2
    environment:
      - POSTGRES_USER=${POSTGRES_USER:-bloodhound}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD:-bloodhoundcommunityedition}
      - POSTGRES_DB=${POSTGRES_DB:-bloodhound}
    # Database ports are disabled by default. Please change your database
password to something secure before uncommenting
    # ports:
    #   - 127.0.0.1:${POSTGRES_PORT:-5432}:5432
    volumes:
      - postgres-data:/var/lib/postgresql/data
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "pg_isready -U ${POSTGRES_USER:-bloodhound} -d ${POSTGRES_DB:-
bloodhound} -h 127.0.0.1 -p ${POSTGRES_PORT:-5432}"
        ]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 30s

  graph-db:
    image: docker.io/library/neo4j:4.4
    environment:
      - NEO4J_AUTH=${NEO4J_USER:-neo4j}/${NEO4J_SECRET:-
bloodhoundcommunityedition}
      - NEO4J_dbms_allow__upgrade=${NEO4J_ALLOW_UPGRADE:-true}
    # Database ports are disabled by default. Please change your database
password to something secure before uncommenting
    ports:
      - 127.0.0.1:${NEO4J_DB_PORT:-7687}:7687
      - 127.0.0.1:${NEO4J_WEB_PORT:-7474}:7474
    volumes:
      - ${NEO4J_DATA_MOUNT:-neo4j-data}:/data
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "wget -O /dev/null -q http://localhost:${NEO4J_WEB_PORT:-7474} ||
exit 1"
        ]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 30s

  bloodhound:
    image: docker.io/specterops/bloodhound:${BLOODHOUND_TAG:-latest}
    environment:
      - bhe_disable_cypher_qc=${bhe_disable_cypher_qc:-false}
      - bhe_database_connection=user=${POSTGRES_USER:-bloodhound}
password=${POSTGRES_PASSWORD:-bloodhoundcommunityedition}
dbname=${POSTGRES_DB:-bloodhound} host=app-db
      - bhe_neo4j_connection=neo4j://${NEO4J_USER:-neo4j}:${NEO4J_SECRET:-
bloodhoundcommunityedition}@graph-db:7687/
      ### Add additional environment variables you wish to use here.
      ### For common configuration options that you might want to use
environment variables for, see `.env.example`
      ### example: bhe_database_connection=${bhe_database_connection}
      ### The left side is the environment variable you're setting for
bloodhound, the variable on the right in `${}`
      ### is the variable available outside of Docker
    ports:
      ### Default to localhost to prevent accidental publishing of the service
to your outer networks
      ### These can be modified by your .env file or by setting the
environment variables in your Docker host OS
      - ${BLOODHOUND_HOST:-127.0.0.1}:${BLOODHOUND_PORT:-8080}:8080
    ### Uncomment to use your own bloodhound.config.json to configure the
application
    # volumes:
    #   - ./bloodhound.config.json:/bloodhound.config.json:ro
    depends_on:
      app-db:
        condition: service_healthy
      graph-db:
        condition: service_healthy

volumes:
  neo4j-data:
  postgres-data:
```

This file is a Docker Compose configuration file that defines three container services:

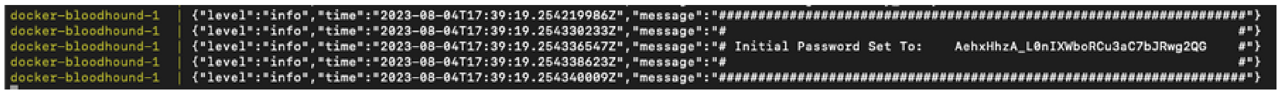
1. **app-db**:
   - It utilizes the PostgreSQL image version 13.2.
   - It sets environment variables for the PostgreSQL user, password, and database name.
   - Configures a volume for storing PostgreSQL data.
   - Defines a health check to verify the availability of the PostgreSQL database.
2. **graph-db**:
   - It utilizes the Neo4j image version 4.4.
   - Sets environment variables for Neo4j authentication and allows database upgrade.
   - Configures access ports for Neo4j.
   - Creates a volume for storing Neo4j data.
   - Defines a health check to verify the availability of the Neo4j web interface.

3. **bloodhound**:
    - It utilizes the Bloodhound image.
    - Sets environment variables for configuring Bloodhound, including connections to PostgreSQL and Neo4j.
    - Specifies the port on which the Bloodhound application should be accessible.
    - Depends on the `app-db` and `graph-db` services and requires them to be running and healthy.

Now in the folder with config file run command:

```
sudo docker-
compose up
```

This will download the correct images, build containers and run them. Scroll up a little bit and locate the randomly generated password in the terminal output of Docker Compose. Look for something like:



```
docker-bloodhound-1  | {"level":"info","time":"2023-08-04T17:39:19.254219986Z","message":"######################################################################"}
docker-bloodhound-1  | {"level":"info","time":"2023-08-04T17:39:19.254330233Z","message":"#                                                                    #"}
docker-bloodhound-1  | {"level":"info","time":"2023-08-04T17:39:19.254336547Z","message":"# Initial Password Set To:    AehxHhzA_L0nIXWboRCu3aC7bJRwg2QG    #"}
docker-bloodhound-1  | {"level":"info","time":"2023-08-04T17:39:19.254338623Z","message":"#                                                                    #"}
docker-bloodhound-1  | {"level":"info","time":"2023-08-04T17:39:19.254340009Z","message":"######################################################################"}
```

In a browser, navigate to http://localhost:8080/ui/login. Login with a username of admin and the randomly generated password from the logs. You will be asked to change the password to your own. In the terminal you can press ctrl+c to close the containers.

Now when you want to run Bloodhound, simply go to the folder with the configuration file and run Bloodhound with the command:

```
sudo docker-compose
up -d
```

`-d` - parameter will allow containers to run in the background, without the need to hold the terminal window.

Now you are a happy user of the new Bloodhound Community Edition. You can start your AD reconnaissance. Oh actually, I haven't told anyone what Bloodhound is. Hmm, I'm sure there will be an article about it in the near future.

Now some important commands for using Docker:

```
sudo
docker ps
```

to display currently running containers.

To log in interactively to the console of a container, use the command:

```
sudo docker exec -it <container_name_or_ID>
bash
```

To stop container use:

```
sudo docker stop
<container_ID>
```

you can also specify several id’s separated by a space, or close all active by:

```
sudo docker stop $(sudo docker
ps -q)
```

If you need to send or download a file to or from a container, use:

```
docker cp /path/to/file.txt
container_name:/path/inside/container
```

```
docker cp container_name:/path/inside/container/plik.txt
/local/path/
```

You can also mount the folder automatically when the container starts by adding a corresponding entry to the configuration. Edit you `docker-compose.yml` file and add inside proper container a volume path:

```
volumes:
      -
/path/on/contener:/local/path/to/folder
```

That’s all, Dockers and their use are not as scary as they may seem. Of course, the topic is very complex and the environment is complicated, but as a regular end user, you only need a few commands and basic knowledge. As I understand it, seriously, everyone will understand it.

Bloodhound Team Work

Sometimes, if we are working in a team and we want access to Bloodhound to be not only local but also hosted on our server within our network for a few users, we can run a container with the IP address of the local machine, which is described in the official documentation, or, what is easier and probably even safer, we can use a reverse proxy.

Using Nginx (it is already installed in Kali), add another page:

```
sudo nano /etc/nginx/sites-
available/bloodhound.conf
```

with configuration:

```
server {
    listen 80;
    server_name IP;  # Replace with your actual IP address

    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

enable it:

```
sudo ln -s /etc/nginx/sites-available/bloodhound.conf /etc/nginx/sites-
enabled/
```

verify configuration:

```
sudo
nginx -t
```

start Nginx:

```
sudo systemctl start
nginx
```

Thanks to this, after entering the IP address of the computer, Nginx will redirect us to the locally running Bloodhound application. Don’t forget to make changes to the firewall and port forwarding in VirtualBox.