



Search

Write

Sign
up

Sign
in



Apache Guacamole: manual installation with docker-compose



Kenneth KOFFI · [Follow](#)

9 min read · Feb 20, 2023



8



1



Updated on 23/11/2023: This article has been revised to incorporate information relevant to the release of **Apache Guacamole version 1.5.3**.

Apache Guacamole is one of the best open source tools out there. I love it and use it daily. But unfortunately, I've already heard many people say that it's too hard to install. So I decided to break this myth and popularize

the deployment of this gem. Everyone should benefit from it.

What is Apache Guacamole ?

Apache Guacamole is a free, open source clientless remote desktop gateway that allows you to access remote Desktop and Server machines via a web browser. It supports standard protocols like VNC, RDP, and SSH, and use HTML5 for remote connection. It can run on most Linux distributions, and the client runs on any modern web browser. You don't need to install any software on your system. Just browse and connect to any remote server defined on your server.

In this article, I will show you how to install Apache Guacamole with Docker on a Linux server.

Requirements

To follow this tutorial, all you need is a Linux server/laptop.

Deployment

*I run all the commands in this article as **root** user. If this is not the case for you, you must precede each command with **sudo**.*

Install docker and docker-compose

Download and execute the convenience script provided by [docker](#).

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

```
sh get-docker.sh | grep -qE "ERROR: Unsupported distribution 'rocky'|ERROR:"
```

Enable and start the docker service

```
sudo systemctl start docker  
sudo systemctl enable docker
```

Add the current user to the **docker** group to allow him to execute docker commands without `sudo`

```
sudo usermod -a ${USER} -G docker
```

You will need to log out and log in back for the change to take effect.

Install Guacamole

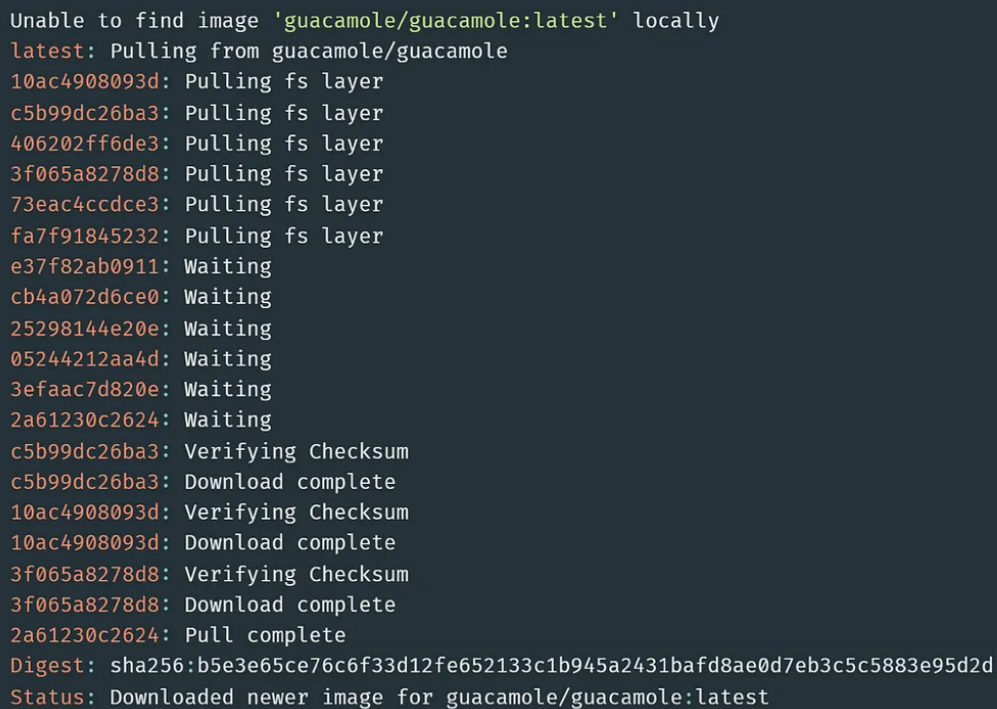
Create a folder for the project

```
mkdir ${HOME}/docker-stack  
cd ${HOME}/docker-stack
```

Initialize the database

```
mkdir -p ${HOME}/docker-stack/guacamole/init
chmod -R +x ${HOME}/docker-stack/guacamole/init
docker run --rm guacamole/guacamole:1.5.3 /opt/guacamole/bin/initdb.sh --pos
```

You should get an output similar to this

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal displays the output of a Docker pull command. It starts with a message indicating the image was not found locally, followed by a series of progress messages for pulling layers and waiting for other containers. The process concludes with checksum verification, download completion, and a final status message indicating a newer image was downloaded.

```
Unable to find image 'guacamole/guacamole:latest' locally
latest: Pulling from guacamole/guacamole
10ac4908093d: Pulling fs layer
c5b99dc26ba3: Pulling fs layer
406202ff6de3: Pulling fs layer
3f065a8278d8: Pulling fs layer
73eac4ccdce3: Pulling fs layer
fa7f91845232: Pulling fs layer
e37f82ab0911: Waiting
cb4a072d6ce0: Waiting
25298144e20e: Waiting
05244212aa4d: Waiting
3efaac7d820e: Waiting
2a61230c2624: Waiting
c5b99dc26ba3: Verifying Checksum
c5b99dc26ba3: Download complete
10ac4908093d: Verifying Checksum
10ac4908093d: Download complete
3f065a8278d8: Verifying Checksum
3f065a8278d8: Download complete
2a61230c2624: Pull complete
Digest: sha256:b5e3e65ce76c6f33d12fe652133c1b945a2431bafd8ae0d7eb3c5c5883e95d2d
Status: Downloaded newer image for guacamole/guacamole:latest
```

Create the `${HOME}/docker-stack/guacamole/docker-compose.yml` file:

```
nano ${HOME}/docker-stack/guacamole/docker-compose.yml
```

Paste the following content into it :

```
version: '3.9'

# networks
# create a network 'guacamole_net' in mode 'bridged'
networks:
  guacamole_net:
    driver: bridge
  haproxy_net:
    external: true

# services
services:
  # guacd
  guacd:
    container_name: guacamole_backend
    image: guacamole/guacd:1.5.3
    networks:
      guacamole_net:
    restart: always
    volumes:
      - ./drive:/drive:rw
      - ./record:/var/lib/guacamole/recordings:rw

  # postgres
  postgres:
    container_name: guacamole_database
    environment:
      PGDATA: /var/lib/postgresql/data/guacamole
      POSTGRES_DB: guacamole_db
      POSTGRES_PASSWORD: '${POSTGRES_PASSWORD}'
      POSTGRES_USER: '${POSTGRES_USER}'
    image: postgres:15.0
    networks:
      guacamole_net:
    restart: always
    volumes:
      - ./init:/docker-entrypoint-initdb.d:ro
      - ./data:/var/lib/postgresql/data:rw

  # guacamole
  guacamole:
    container_name: guacamole_frontend
    depends_on:
      - guacd
      - postgres
    environment:
      GUACD_HOSTNAME: guacd
      POSTGRESQL_DATABASE: guacamole_db
      POSTGRESQL_HOSTNAME: postgres
```

```
POSTGRESQL_PASSWORD: '${POSTGRES_PASSWORD}'
POSTGRESQL_USER: '${POSTGRES_USER}'
POSTGRESQL_AUTO_CREATE_ACCOUNTS: true
image: guacamole/guacamole:1.5.3
links:
- guacd
networks:
- guacamole_net
- haproxy_net
restart: always
volumes:
- ./drive:/drive:rw
- ./record:/var/lib/guacamole/recordings
```

You may wonder what's going on here, so let me explain. This `docker-compose.yml` file contains the declaration of all the docker containers needed to run Apache Guacamole. Guacamole is not a self-contained web application and is made up of many parts. By default, Guacamole need 3 containers to run:

- **guacamole_frontend:** the web UI, the part of Guacamole that a user actually interacts with.
- **guacamole_backend:** guacd is the heart of Guacamole, which dynamically loads support for remote desktop protocols (called “client plugins”) and connects them to remote desktops based on instructions received from the web application.
- **guacamole_database:** the database that Guacamole will use for authentication and storage of connection configuration data.

Create the `${HOME}/docker-stack/guacamole/.env` file to store the database credentials. Like this:

```
POSTGRES_PASSWORD='PleasePutAStrongPasswordHere'  
POSTGRES_USER='guacamole_user'
```

The `docker-compose.yml` file also contains the declaration of two networks:

- **guacamole_net**: the docker network to isolate communication between the different guacamole services.
- **haproxy_net**: the docker network to link **guacamole_frontend** container and **HAProxy** container

You just said HAProxy ? What is that ?

HAProxy is a reverse proxy which can run inside a docker container. We're going to put it in front of our guacamole stack to uplift the Guacamole UI to SSL/HTTPS.

Install HAProxy

Create a folder to hold our HAProxy configuration

```
mkdir -p ${HOME}/docker-stack/haproxy  
cd ${HOME}/docker-stack/haproxy
```

Create the `${HOME}/docker-stack/haproxy/docker-compose.yml` file

```
nano ${HOME}/docker-stack/haproxy/docker-compose.yml
```

and paste the following content inside:

```
version: '3.9'
services:
  haproxy:
    container_name: haproxy
    image: haproxytech/haproxy-alpine:2.4
    ports:
      - 80:80
      - 443:443
      - 8404:8404
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
    networks:
      - haproxy_net
    restart: always
    environment:
      ENDPOINT: '${ENDPOINT}'
networks:
  haproxy_net:
    name: haproxy_net
    driver: bridge
```

Let's define the HAProxy configuration file.

Create the `${HOME}/docker-stack/haproxy/haproxy.cfg` file

```
nano ${HOME}/docker-stack/haproxy/haproxy.cfg
```

and paste the following content into it :


```
global
  stats socket /var/run/api.sock user haproxy group haproxy mode 660 level a
  log stdout format raw local0 info
  maxconn 50000

resolvers docker_resolver
  nameserver dns 127.0.0.11:53

defaults
  mode http
  timeout client 10s
  timeout connect 5s
  timeout server 10s
  timeout http-request 10s
  default-server init-addr none
  log global

frontend stats
  bind *:8404
  stats enable
  stats uri /
  stats refresh 10s

frontend myfrontend
  mode http
  bind :80
  use_backend %[req.hdr(Host),lower]

backend "${ENDPOINT}"
  server guacamole guacamole:8080 check inter 10s resolvers docker_resolver
```

These directives are used to define how the reverse proxy should work.

Let's define the Fully Qualified Domain Name (FQDN) where we want to reach our guacamole web UI.

Create the file `${HOME}/docker-stack/haproxy/.env` with the following template.

```
ENDPOINT="your fqdn"
```

In my case, I want to use `161-35-39-33.traefik.me` as FQDN. Where `161.35.39.33` is the public IP address of my server hosting the guacamole app. So in my case, this is the `${HOME}/docker-stack/haproxy/.env` file content:

```
ENDPOINT='161-35-39-33.traefik.me'
```

Then what about that `.traefik.me` appended at the end ?

That's an excellent question !

traefik.me is a magic domain name that can be used by anyone on the internet. It provides wildcard DNS for any IP address. Say your host IP address is `159.74.28.170`. Using `traefik.me`,

- `159-74-28-170.traefik.me` resolves to `159.74.28.170`
- `159.74.28.170.traefik.me` resolves to `159.74.28.170`
- `www.159.74.28.170.traefik.me` resolves to `159.74.28.170`

...and so on. This avoids you buying domain name to test your projects or editing `/etc/hosts` on your devices. On top of that, a wildcard SSL certificate signed by Let's encrypt is available for `*.traefik.me`. Just grab the files from traefik.me and you're ready to go.

Why don't you just use the IP address of your server to access your application

| ?

That's also a possibility but as I said, traefik.me provides free SSL certificate signed by a public CA and some applications integrations don't work with IP address.

But buddy, feel free to use any FQDN that suits you. I was just giving you a tip. Now let's continue our deployment.

At the end, your folder architecture should look like this:

```

${HOME}/docker-stack
├─ guacamole
│   ├── .env
│   ├── docker-compose.yml
│   ├── init
│   └── `-- initdb.sql
`-- haproxy
    ├── .env
    ├── docker-compose.yml
    └── `-- haproxy.cfg
```

Don't forget to change files ownership

```
chown -R ${USER}:${USER} ${HOME}/docker-stack/
```

Now let's bring everything up

```
docker compose -f ${HOME}/docker-stack/haproxy/docker-compose.yml up -d
docker compose -f ${HOME}/docker-stack/guacamole/docker-compose.yml up -d
```

Your output should look like this:

```
[+] Running 6/6
  :: haproxy Pulled 3.1s
  :: 63b65145d645 Pull complete 0.7s
  :: 730f0f9518c9 Pull complete 1.2s
  :: 5bfeac19ca18 Pull complete 1.5s
  :: d445c7b17883 Pull complete 1.5s
  :: fb9f2d9b4cd8 Pull complete 1.6s
[+] Running 2/2
  :: Network haproxy_net Created 0.1s
  :: Container haproxy Started
```

```
[+] Running 20/20
  :: postgres Pulled 10.8s
  :: 7d63c13d9b9b Pull complete 4.4s
  :: cad0f9d5f5fe Pull complete 4.7s
  :: ff74a7a559cb Pull complete 4.8s
  :: c43dfd845683 Pull complete 4.9s
  :: e554331369f5 Pull complete 5.4s
  :: d25d54a3ac3a Pull complete 5.5s
  :: bbc6df00588c Pull complete 5.6s
  :: d4deb2e86480 Pull complete 5.7s
  :: d4132927c0d9 Pull complete 9.0s
  :: 3d03efa70ed1 Pull complete 9.1s
  :: 645312b7d892 Pull complete 9.1s
  :: 3cc7074f2000 Pull complete 9.2s
  :: 4e6d0469c332 Pull complete 9.3s
  :: guacd Pulled 6.1s
  :: 63b65145d645 Already exists 0.0s
  :: 76b564159444 Pull complete 1.0s
  :: a8f9a235cadc Pull complete 4.7s
  :: a1b1f03ae3d6 Pull complete 4.8s
  :: f20dbdbdbec3 Pull complete 5.0s
[+] Running 4/4
  :: Network guacamole_guacamole_net Created 0.1s
  :: Container guacamole_database Started 1.1s
  :: Container guacamole_backend Started 1.0s
  :: Container guacamole_frontend Started 1.4s
```

Et voilà !! You get your Guacamole instance deployed





Access Apache Guacamole Dashboard

You can now access the Apache Guacamole web interface using the URL <http://your-fqdn/guacamole>. In my case, my instance is reachable at <http://161-35-39-33.traefik.me/guacamole>





APACHE GUACAMOLE

Se connecter

150

The default admin credentials are:

- username: **guacadmin**
- password: **guacadmin**

Enable SSL

We have deployed our application and we are happy. But satisfaction is not yet at its peak. For the moment, we are using the HTTP protocol, which by default is not very secure. So we will add a security layer with an SSL/TLS certificate. There are many possibilities.

Use SSL certificate from traefik.me

As I said in the previous section, traefik.me offers free wildcard SSL certificate issued by [Let's encrypt](https://letsencrypt.org/).

To use it, we will add a new docker container to our HAProxy docker-compose configuration file. This docker container will use alpine image. Its role will be to: download the certificate files from traefik.me, convert them to HAProxy supported format and pass it to a docker volume shared by both containers.

So update your previous `${HOME}/docker-stack/haproxy/docker-compose.yml` file like this:

```
version: '3.9'
services:
  haproxy:
    container_name: haproxy
    image: haproxytech/haproxy-alpine:2.4
    ports:
      - 80:80
      - 443:443
      - 8404:8404
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
      # New line added here
```

```

    - certs:/usr/local/etc/haproxy/certs/
networks:
    - haproxy_net
restart: always
environment:
    ENDPOINT: '${ENDPOINT}'

##### New section added #####
reverse-proxy-https-helper:
    image: alpine
    command: sh -c "cd /etc/ssl/traefik
        && wget traefik.me/cert.pem -O cert.pem
        && wget traefik.me/privkey.pem -O privkey.pem
        && cat cert.pem privkey.pem > traefik.me.pem"
    volumes:
        - certs:/etc/ssl/traefik
#####

networks:
    haproxy_net:
        name: haproxy_net
        driver: bridge

##### New section added #####
volumes:
    certs:
#####

```

I have placed comments to highlight the changes made.

We also need to edit the HAProxy configuration file `${HOME}/docker-stack/haproxy/haproxy.cfg` to instruct HAProxy to use the SSL certificate.

Update the `frontend` section of that file.

```

frontend myfrontend
    mode http
    bind :80
    # New line added here
    bind :443 ssl crt /usr/local/etc/haproxy/certs/traefik.me.pem
    #http-request redirect scheme https code 301 unless { ssl_fc }

```

```
use_backend %[req.hdr(Host),lower]
```

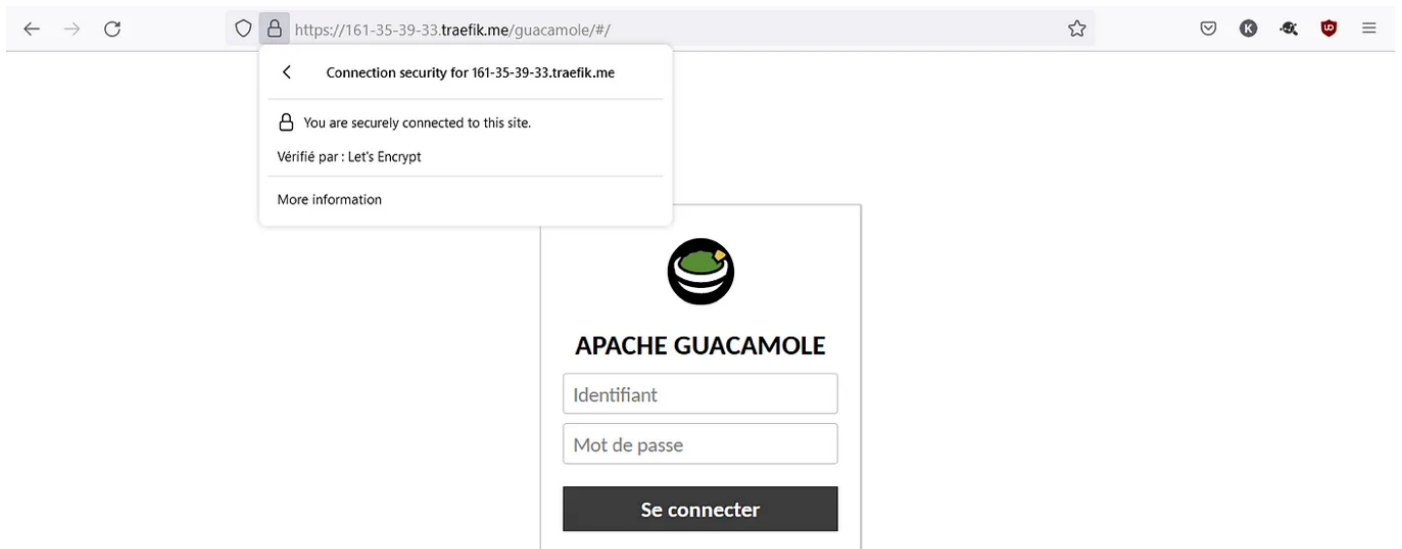
Uncomment the line `#http-request redirect scheme https code 301 unless { ssl_fc }` if you want to redirect all HTTP requests to HTTPS.

Execute these commands to apply the changes made:

```
docker compose -f ${HOME}/docker-stack/haproxy/docker-compose.yml up -d
docker restart haproxy
```

Now you can access your instance with HTTPS protocol.

In my case, the URL is <https://161-35-39-33.traefik.me/guacamole>



Use SSL self-signed certificate

In case you want to use a self-signed certificate, here are the steps to

follow.

Generate the certificate

```
mkdir ${HOME}/docker-stack/haproxy/certs
openssl req -nodes -newkey rsa:2048 -new -x509 -keyout ${HOME}/docker-stack/
```

NOTE: At the end of the above command, make sure to replace `your-fqdn` by the correct value of your FQDN.

Convert the certificate to HAProxy supported format

```
bash -c 'cat ${HOME}/docker-stack/haproxy/certs/self.cert ${HOME}/docker-sta
```

Update the `${HOME}/docker-stack/haproxy/docker-compose.yml` file as below:

```
version: '3.9'
services:
  haproxy:
    container_name: haproxy
    image: haproxytech/haproxy-alpine:2.4
    ports:
      - 80:80
      - 443:443
      - 8404:8404
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
      # New line added here
      - ./certs:/usr/local/etc/haproxy/certs/
```

```

    networks:
      - haproxy_net
  restart: always
  environment:
    ENDPOINT: '${ENDPOINT}'
networks:
  haproxy_net:
    name: haproxy_net
    driver: bridge

```

The difference between the old one and the new one:

```

diff --label 'old' -u --color=always ${HOME}/docker-stack/haproxy/docker-com
--- old
+++ new
@@ -9,6 +9,7 @@
     - 8404:8404
     volumes:
       - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
+    - ./certs:/usr/local/etc/haproxy/certs/
     networks:
       - haproxy_net
     restart: always

```

Update the frontend section of `${HOME}/docker-stack/haproxy/haproxy.cfg` file

```

frontend myfrontend
  mode http
  bind :80
  # New line added here
  bind :443 ssl crt /usr/local/etc/haproxy/certs/haproxy-ssl.pem
  #http-request redirect scheme https code 301 unless { ssl_fc }
  use_backend %[req.hdr(Host),lower]

```

We can use the `diff` command to check the difference between the old and new version:

```
diff -u --color=always --label 'old version' ~/docker-stack/haproxy/haproxy.cfg.a --label 'new version' ~/docker-stack/haproxy/haproxy.cfg
--- old version
+++ new version
@@ -24,6 +24,9 @@
     frontend myfrontend
         mode http
         bind :80
+    + # New line added here
+    + bind :443 ssl crt /usr/local/etc/haproxy/certs/haproxy-ssl.pem
+    + #http-request redirect scheme https code 301 unless { ssl_fc }
       use_backend %[req.hdr(Host),lower]
```

Uncomment the line `#http-request redirect scheme https code 301 unless { ssl_fc }` *if you wish to redirect all HTTP requests to HTTPS.*

Execute these commands to apply the changes made

```
docker compose -f ${HOME}/docker-stack/haproxy/docker-compose.yml up -d
docker restart haproxy
```

Now you can access your Guacamole instance with HTTPS protocol.

Use CA signed SSL certificate

For certificates issued by public certificate authorities, the procedure is the same as for self-signed certificates. Except that you do not generate certificates yourself. You just have to adapt the paths. I mean replacing

`self.crt` and `self-ssl.key` with the filenames of the certificate and associated private key respectively.

In this article, we deployed Apache Guacamole with docker and HAProxy as reverse proxy. Thank you for reading to the end, and see you soon for new articles.

You can reach me here in the comment section or on [LinkedIn via this link](#).

Docker

Docker Compose

Apache Guacamole

Haproxy



Written by Kenneth KOFFI

23 Followers

Follow



Administrateur systèmes avec une appétence pour le Cloud et le DevOps

More from Kenneth KOFFI