WEBSERVER SETUP INSIDE DOCKER

Key Points

- How to Setup Webserver
- Docker Build vs DockerFile
- Docker Volume
- Docker Expose

www.linkedin.com/in/kumar--himanshu/

Follow

Share

WEBSERVER SETUP IN DOCKER

Once upon a time, in the world of web hosting, there lived a diligent webmaster named Alex. Alex was responsible for managing a website for a small business, and this website was the lifeblood of their online presence.

In the early days of web hosting, Alex hosted the website on a traditional operating system, much like how we use our computers at home. Everything seemed to be running smoothly, and the website attracted a steady stream of visitors.

However, there was a problem. Whenever there was an issue with the web server, like a sudden surge in traffic or a software conflict, Alex had to intervene. This meant late nights, frantic troubleshooting, and sometimes even server downtime. The business depended on the website, and this constant maintenance was taking its toll on Alex.

One day, while browsing the internet for solutions to simplify web hosting, Alex stumbled upon a revolutionary concept - containerization. Containerization, as Alex learned, offered an isolated environment for hosting applications, including web servers. It was like creating a little world for the web server to live in, separate from the main operating system.

Excited by the potential benefits, Alex decided to give containerization a try. They began by setting up a container specifically for the web server. The beauty of containerization was that it encapsulated everything needed to run the web server – the code, libraries, and dependencies – all neatly packaged into one container.

With the container in place, Alex could now easily launch and manage the web server with just a few simple commands. If there was ever an issue with the website, instead of frantically troubleshooting the entire server, Alex could simply replace the problematic container with a fresh one.

This newfound simplicity was a game-changer. Alex could now launch a new container within seconds, and the website would be up and running again, as good as new. The business no longer suffered from long periods of downtime, and Alex had more time for other important tasks.

As time went on, Alex marveled at how containerization had transformed their web hosting experience. The days of maintaining a complex web server on a traditional operating system were a distant memory. Containerization had provided an isolated, hassle-free environment where the website thrived, and Alex could finally breathe easy.

And so, in the world of web hosting, the tale of Alex and containerization became a shining example of how technology had made life simpler, allowing webmasters to focus on what mattered most – ensuring their websites ran smoothly and their businesses flourished.

Alex has their website hosted on normal OS. they have installed one software HTTPD and then start the service of httpd and enable it for permanently and then put the content of the webpage in the Document Root of httpd which is in this location **/var/www/html/**

```
[root@ip-172-31-13-44 ~]# yum install httpd -y
Last metadata expiration check: 1:01:58 ago on Fri Sep 22 06:30:28 2023.
Dependencies resolved.
================================================================================
 Package                         Architecture            Version
================================================================================
Installing:
 httpd                           x86_64                  2.4.56-1.amzn2023
```

```
[root@ip-172-31-13-44 ~]# systemctl enable httpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[root@ip-172-31-13-44 ~]# systemctl status httpd
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
     Active: active (running) since Fri 2023-09-22 07:33:07 UTC; 7s ago
```

```
[root@ip-172-31-13-44 ~]# cd /var/www/html/
[root@ip-172-31-13-44 html]# vim index.html
[root@ip-172-31-13-44 html]# ls
index.html
```

After encountering some issues, Alex wanted to host their website in a Docker container. To achieve this, Alex learned the basics of Docker from the following link, "[Docker Basics]" Furthermore, Alex already possesses knowledge on how to set up a web server, so they downloaded a fresh CentOS image from Docker Hub.

```
[root@ip-172-31-13-44 html]# docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a1d0c7532777: Pull complete
Digest: sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
[root@ip-172-31-13-44 html]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
centos       latest    5d0da3dc9764   2 years ago   231MB
```

- **docker pull centos** # Pull centos Images with latest version.
- **docker images** # List all the available Image in the OS.

```
[root@ip-172-31-13-44 html]# docker run -it --name company_webserver centos
[root@e28029ab46d8 /]#
```

- **docker run-it –name company_webserver centos**
  # Alex created a new container with the name "**company_webserver**" using the CentOS image. This system is essentially a fresh operating system. Now, he will proceed with setting up everything as needed.

However, when he attempts to install the Apache HTTP Server (Httpd) software using the `yum` command on the server, he encounters an error.

```
[root@e28029ab46d8 /]# yum install httpd -y
Failed to set locale, defaulting to C.UTF-8
CentOS Linux 8 - AppStream                                                      170
Error: Failed to download metadata for repo 'appstream': Cannot prepare internal mirrorlist: No URLs in mirrorlist
[root@e28029ab46d8 /]#
```

Then He search the solution from web then he Found these 3 Command which solves his problem.

- **sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-***
- **sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-***
- **dnf update -y**

```
[root@e28029ab46d8 ~]# yum install httpd
Last metadata expiration check: 0:03:30 ago on Fri 22 Sep 2023 07:50:59 AM UTC.
Dependencies resolved.
================================================================================
 Package                 Architecture          Version
================================================================================
Installing:
 httpd                   x86_64                2.4.37-43.module_el8.5.0+1022+b541f3b1
```

After successfully installing the webserver software, he has two options for adding content to the webpage:

1. 1. He can transfer the content from the Host OS to the Docker container's document root, which is located at `/var/www/html/`.
2. He can create the content from scratch.

```
[root@e28029ab46d8 html]# ls
[root@e28029ab46d8 html]#
[root@e28029ab46d8 html]# pwd
/var/www/html
[root@e28029ab46d8 html]#
[root@e28029ab46d8 html]#
```

But he decides to put the content from the Host OS to Docker Container by using copy command.

```
[root@ip-172-31-13-44 html]# ls
index.html
[root@ip-172-31-13-44 html]#
[root@ip-172-31-13-44 html]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
e28029ab46d8   centos    "/bin/bash"   14 minutes ago   Up 14 minutes             company_webserver
[root@ip-172-31-13-44 html]# pwd
/var/www/html
```

Alex's content is currently located in the Host OS folder `**/var/www/html/**`, with the filename `**index.html**`. He needs to transfer this content to the container named '**company_webserver**,' and inside the container, it should be placed in the directory `**/var/www/html/**`.

He uses the `**docker cp**` command to transfer the file from the Host OS to inside the container. Here's how he can do it:

- **Syntax: docker cp path-of-file/file-name container-name:/path-to-store**
- **Command: docker cp /var/www/html/index.html company_webserver:/var/www/html/**

This command copies the `**index.html**` file from the Host OS to the **'company_webserver'** container, placing it in the `**/var/www/html/**` directory inside the container.

```
[root@ip-172-31-13-44 html]# docker cp index.html company_webserver:/var/www/html/
Successfully copied 4.1kB to company_webserver:/var/www/html/
[root@ip-172-31-13-44 html]#
```

And when we get inside the container using: **docker attach container-name**

```
[root@ip-172-31-13-44 ~]# docker attach company_webserver
[root@e28029ab46d8 html]# ls
index.html
[root@e28029ab46d8 html]# pwd
/var/www/html
```

Then we can see Alex have successfully copied the content from the Main Host to Docker Container. Now Alex needs to start the HTTPD service using:

- systemctl start httpd.

```
[root@e28029ab46d8 html]# systemctl start httpd
System has not been booted with systemd as init system (PID 1). Can't operate.
Failed to connect to bus: Host is down
[root@e28029ab46d8 html]#
```

Here Alex encounters the problem. Here in docker container we cannot use systemctl command, but Alex is a good developer. He knows that systemctl command behind the seen run one command to start the httpd webserver. And this command is located in this location "**/usr/lib/systemd/system/"** inside the file "**httpd.service**"

```
[root@ip-172-31-13-44 ~]# systemctl status httpd
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
```

```
[root@ip-172-31-13-44 system]# pwd
/usr/lib/systemd/system
[root@ip-172-31-13-44 system]# ls | grep httpd
httpd.service
httpd.service.d
httpd.socket
```

When he open the file "**httpd.service**" then he saw that for starting the service he need to run this command inside the container:

- **/usr/sbin/httpd**

```
[Service]
Type=notify
Environment=LANG=C

ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
```

Then he again go inside the container using: "**docker attach container-name"** and run **/usr/sbin/httpd -DFOREGROUND** here DFOREGROUND is used to run this command in the background.

```
[root@e28029ab46d8 html]# ls
index.html
[root@e28029ab46d8 html]# /usr/sbin/httpd -DFOREGROUND
AH00558: httpd: Could not reliably determine the server's fully qualified
o suppress this message
read escape sequence
```

To check whether web server is running or not, he need the IP address of the container, which can be obtained by using the `**docker inspect <container-name>**` command. Scroll down and search for the IP address within the output to find the container's IP address.

```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "fa62be16e6029c0d955d82c44d6d5d2de31ca9fc364227df3bea5f24961d2aa8",
        "EndpointID": "7e3d38ae125c4263f66b4cd161c8e0ace78f1fc0d07a988d69a28f0f10a6a954",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
```

Or you can format an argument to get the IP address.

- **docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' company_webserver**

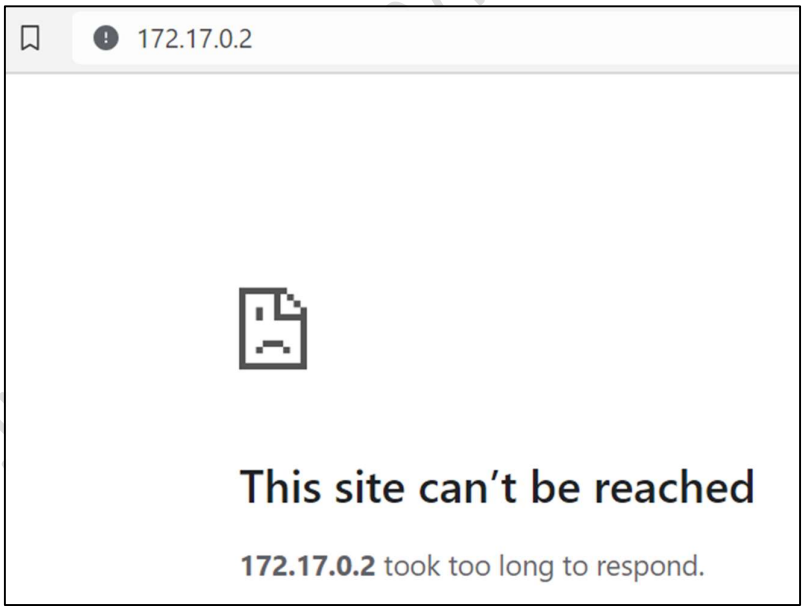
```
[root@ip-172-31-13-44 ~]#  docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' company_webserver
172.17.0.2
[root@ip-172-31-13-44 ~]#
```

Now He is using Curl Command with container IP address to see the Webserver which is hosted inside the container.

```
[root@ip-172-31-13-44 ~]# curl 172.17.0.2
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DevOps Basics</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f0f0f0;
        }
```

Here It is showing the result which means that webserver is successfully hosted.

But When Alex Put the IP of the Container in the normal browser like chrome then the Webpage is not visible to him.



When Alex hosted the webserver inside the container, Docker was installed on top of his machine. By default, Docker and the host OS have connectivity, but Docker does not allow external connections to the container from the outside world. However, Alex found a solution for this issue.

Alex explains, "I want to run the webserver in my container, but Docker doesn't allow direct connections from the public world. Fortunately, there are ways to enable external access. I'm going to open a specific port, like 1234, in this container. If anyone from the public wants to access my web server, they'll need to use my host machine's IP address along with the port number where the container is running. Then they can view the webpage. Essentially, I'm exposing my container to the real world. While it's possible to run multiple programs in one container, it's not considered a good practice. It's better to create one container for each specific purpose."

But here we have an existing Docker container that was created without port 80 exposed, then Alex cannot directly modify the container to expose port 80.

For this reason, I am going to remove the existing container and creating the new container where I am going to expose the port 80.

```
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED       STATUS                      PORTS     NAMES
e28029ab46d8   centos    "/bin/bash"   3 hours ago   Exited (137) 55 minutes ago           company_webserver
[root@ip-172-31-13-44 ~]# docker rm company_webserver
company_webserver
```

```
[root@ip-172-31-13-44 ~]# docker run -it -p 1234:80  --name company_webserver centos
[root@40721d58aaf1 /]#
```

Again, I am going to run those 4 Commands to Install the Webserver

- **sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-***
- **sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-***
- **dnf update -y**
- **yum install httpd -y**

```
[root@40721d58aaf1 /]# rpm -q httpd
httpd-2.4.37-43.module_el8.5.0+1022+b541f3b1.x86_64
[root@40721d58aaf1 /]#
```

The httpd software is installed and also, I have transfer the index.html file from Host OS to Docker Container.

```
[root@ip-172-31-13-44 ~]# docker cp /var/www/html/index.html company_webserver:/var/www/html/
Successfully copied 4.1kB to company_webserver:/var/www/html/
[root@ip-172-31-13-44 ~]# docker attach company_webserver
[root@40721d58aaf1 /]# cd /var/www/html/
[root@40721d58aaf1 html]# ls
index.html
```

Then I have started the HTTPD service

```
[root@40721d58aaf1 html]# /usr/sbin/httpd -DFOREGROUND
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
o suppress this message
```

After that I have removed the index.html file from Document Root of HOST OS

```
[root@ip-172-31-13-44 ~]# cd /var/www/html/
[root@ip-172-31-13-44 html]# ls
index.html
[root@ip-172-31-13-44 html]# rm index.html
rm: remove regular file 'index.html'? y
[root@ip-172-31-13-44 html]# ls
[root@ip-172-31-13-44 html]#
[root@ip-172-31-13-44 html]#
```

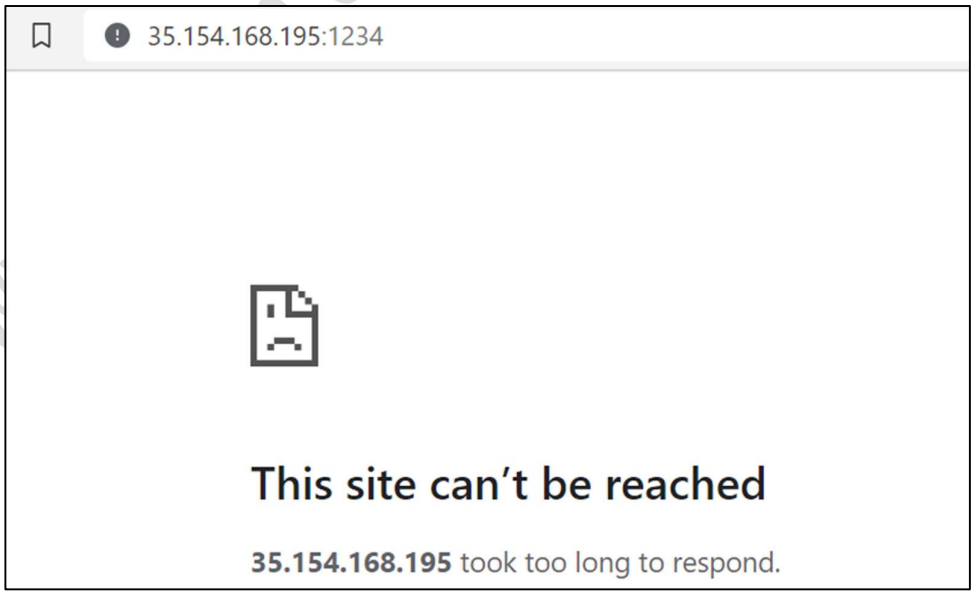
Currently in my HOST OS, The Content is:

But My webserver is running inside the container on the port number 1234. So we need to mention the Port Number with the IP address.

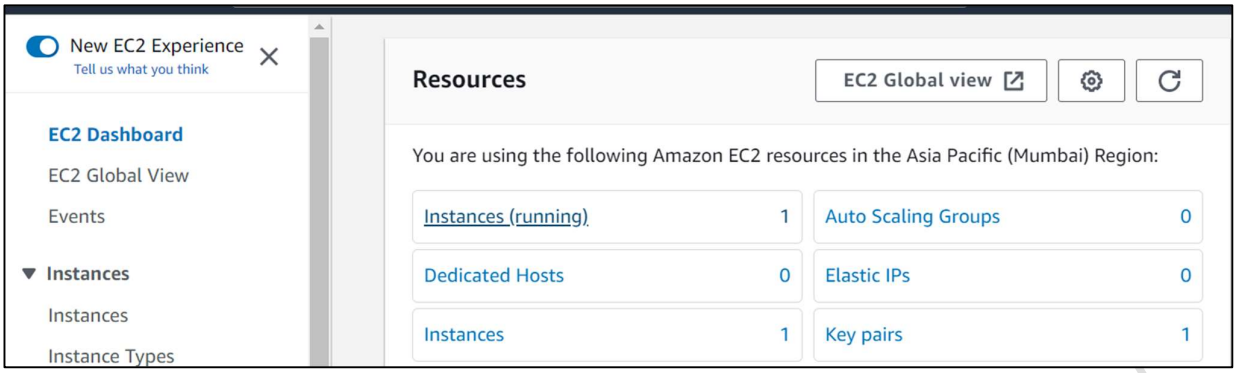
I have checked this with command: **netstat -tnlp**

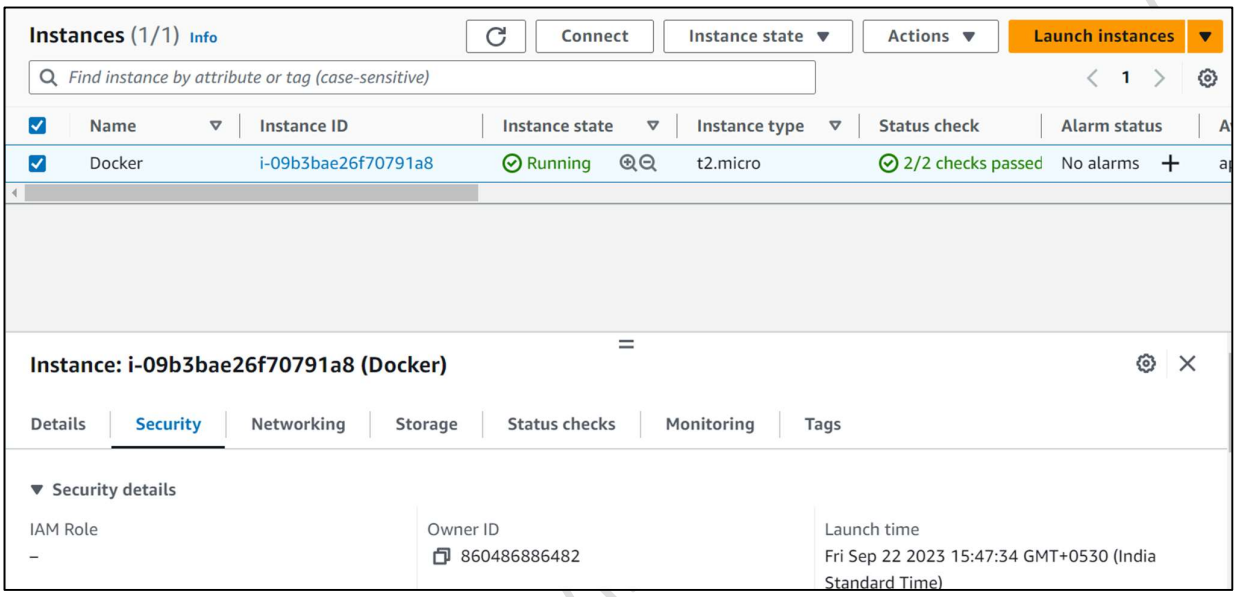
```
[root@ip-172-31-13-44 html]# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2273/sshd: /usr/sbi
tcp        0      0 0.0.0.0:1234            0.0.0.0:*               LISTEN      3402/docker-proxy
tcp        0      0 127.0.0.1:35227         0.0.0.0:*               LISTEN      1946/containerd
tcp6       0      0 :::22                   :::*                    LISTEN      2273/sshd: /usr/sbi
tcp6       0      0 :::80                   :::*                    LISTEN      1953/httpd
tcp6       0      0 :::1234                 :::*                    LISTEN      3407/docker-proxy
[root@ip-172-31-13-44 html]#
```

But when I Try to enter the IP address with the port number then I am not able to see my webpage because of firewall issue.



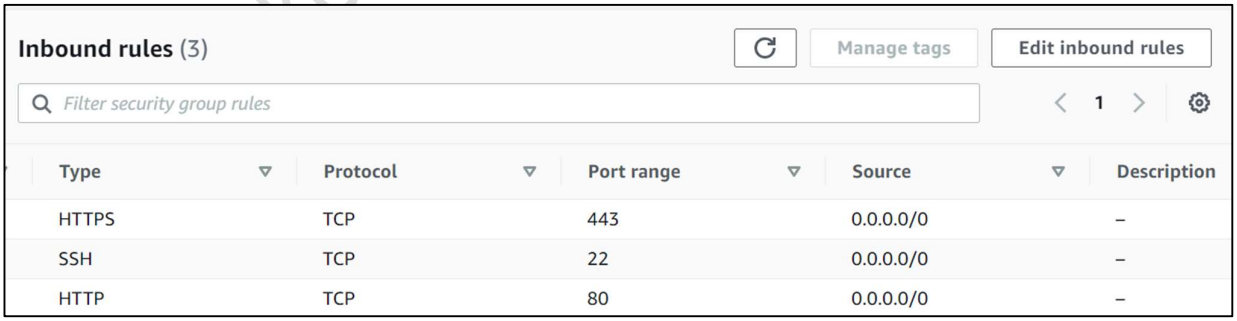
From the AWS Console → Go to EC2 Dashboard



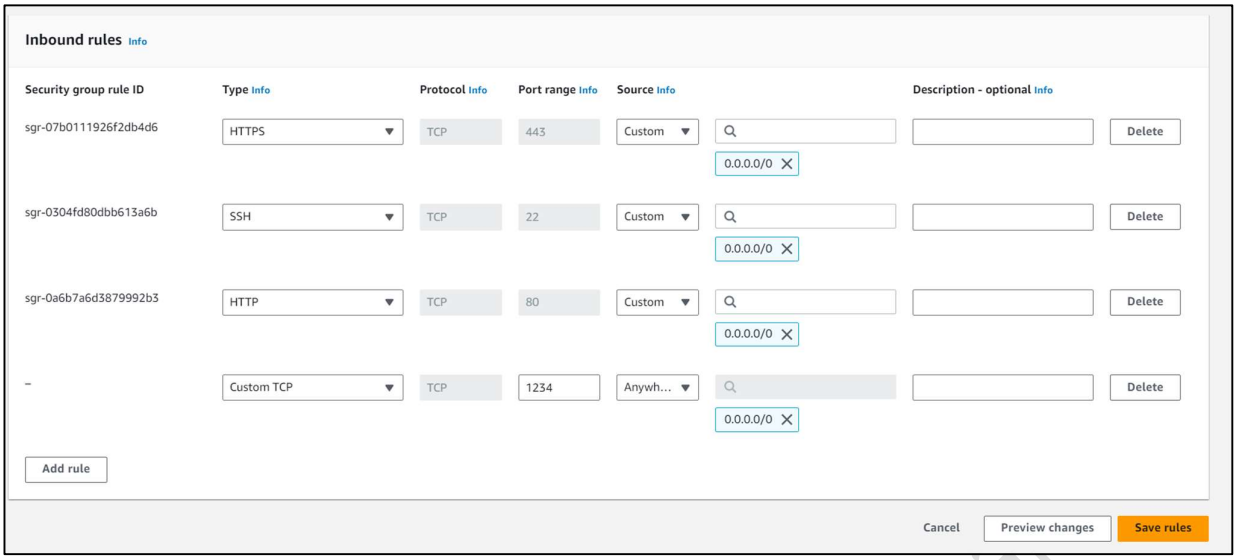


Select your Instance where you have installed Docker → Go to Security Section

Then Click on Security Groups



Here, we have only allowed access from public IP addresses on ports 443, 22, and 80. However, I would like to allow access on port 1234 for the public. To do this, click on "Edit Inbound Rules," which is located in the upper right corner.

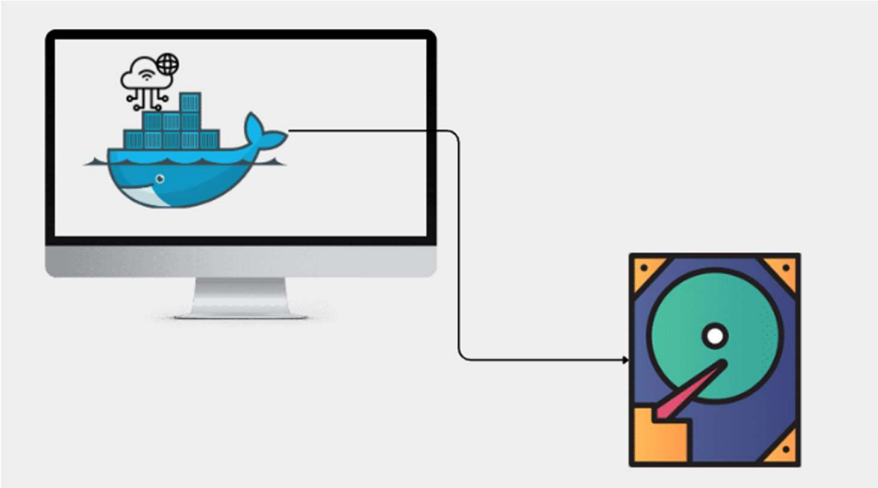


Next, click on "**Add rule,**" then in the "**Type**" section, select "**Custom TCP**." Enter "**1234**" in the "**Port Range**" field and select "**Anywhere**" in the "**Source**" field. Finally, click on "**Save Rules**."

Now if we again try to access the webserver then we can view it.

Imagine if there are problems with the container, and that makes it hard for people to reach our webpage. Or, let's say the container is accidentally deleted. In both cases, nobody can access our webpage anymore. So, how can we fix this issue?

The thing is, this container provides temporary storage. That means if we delete the container, everything inside it is also gone



We can solve this problem using a simple technique. When we start the container, we can connect it to a storage device like a USB drive. This USB drive works as a permanent storage space. Alex can put important things, like the company's web pages, on this USB drive. So, if the container ever stops working, the data on the USB drive stays safe.

Now, you might wonder how to connect this USB drive to the Docker container. Docker lets us do this when we start the container. We can link this storage to a specific folder inside the container, where we want to keep the data safe. Even if the container is deleted, the data remains secure on the USB drive.

Let’s try to implement this thing practically.

```
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED      STATUS          PORTS                                   NAMES
40721d58aaf1   centos    "/bin/bash"   2 days ago   Up 14 seconds   0.0.0.0:1234->80/tcp, :::1234->80/tcp   company_webserver
[root@ip-172-31-13-44 ~]# docker rm company_webserver
Error response from daemon: You cannot remove a running container 40721d58aaf18a629563d3304723224f1b50767ca7b695b03b77703860db22c9.
```

Right now, I have just one container running, and it's named 'company_webserver.' If I try to delete this container, I'll get an error message because it's currently running. You can technically force-delete a running container, but that's not a recommended practice.

```
[root@ip-172-31-13-44 ~]# docker stop company_webserver
company_webserver
[root@ip-172-31-13-44 ~]# docker rm company_webserver
company_webserver
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-13-44 ~]#
```

Here, I stopped the container first, and then I removed it. Now, let's attempt to create a new container where I'll share storage from my base operating system's hard drive and mount it inside one of the container's folders.

```
[root@ip-172-31-13-44 ~]# ls
index.html
[root@ip-172-31-13-44 ~]# mkdir Host_Webpage
[root@ip-172-31-13-44 ~]# mv index.html Host_Webpage/
[root@ip-172-31-13-44 ~]# ls
Host_Webpage
[root@ip-172-31-13-44 ~]# cd Host_Webpage/
[root@ip-172-31-13-44 Host_Webpage]# ls
index.html
[root@ip-172-31-13-44 Host_Webpage]# cd ..
[root@ip-172-31-13-44 ~]#
```

On my host operating system, I have a webpage stored in the main directory. Within this main directory, I made a new folder called 'Host_Webpage.' I moved the webpage from the main directory into this 'Host_Webpage' folder. Now, I'm about to start a container and connect the 'Host_Webpage' directory to it.

```
[root@ip-172-31-13-44 ~]# docker run -it -p 1234:80 -v /root/Host_Webpage/:/var/www/html --name company_webserver1 centos
[root@fd5085470bb6 /]# cd /var/www/html/
[root@fd5085470bb6 html]# ls
index.html
[root@fd5085470bb6 html]#
```

I made a container named 'company_webserver' using the CentOS image, and I've successfully linked it with my host storage.

Now, if I save something in the ' /var/www/html/' directory inside the container, it will also be saved in my host operating system's 'Host_Webpage' directory.

```
[root@fd5085470bb6 html]# ls
index.html
[root@fd5085470bb6 html]# read escape sequence
[root@ip-172-31-13-44 ~]# cd Host_Webpage/
[root@ip-172-31-13-44 Host_Webpage]# ls
index.html
[root@ip-172-31-13-44 Host_Webpage]# docker attach company_webserver1
```

I'm currently inside my container, but I want to access my host operating system. To do that, I use the keyboard shortcut **Ctrl + P + Q**. Once I'm in my host OS, I navigate to the directory '/root/Host_Webpage/' where I have a single file named 'index.html.' Then, to return to the container, I use the **'docker attach container-name**' command.

```
[root@fd5085470bb6 html]# cat > Container-file.txt
Hello User
I am file from the container
Going to store in Host OS
[root@fd5085470bb6 html]# ls
Container-file.txt  index.html
[root@fd5085470bb6 html]# cat Container-file.txt
Hello User
I am file from the container
Going to store in Host OS
[root@fd5085470bb6 html]# read escape sequence
[root@ip-172-31-13-44 Host_Webpage]# ls
Container-file.txt  index.html
```

While I was inside my container, I made a file named **'Container-file.txt**' and added some content to it. Then, I returned to my host OS using the shortcut "**Ctrl + P + Q**." After that, I navigated to the '/root/Host_Webpage' directory, and when I ran the '**ls**' command, you could see that the file is now visible in my host OS.

So, we've solved the issue of temporary storage, but there's another problem we need to address – setting up the container correctly.

```
sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*
dnf update -y
yum install httpd -y
/usr/sbin/httpd -DFOREGROUND
```

Every time I start a new container, I have to set up the web server from scratch. This means I have to install the web server software and then start the services manually. If there's an issue with the web server every time I start a container, it becomes repetitive to run these four commands each time. If I start 100 containers, that's 100 times I'd have to run these commands.

To address this challenge in Docker, we have two solutions:

- Using Docker commit
- Using a Dockerfile

Let me explain how Docker Commit works.

First, we start by downloading a fresh image like CentOS from Docker Hub. Then, we run some commands to install certain software on the base operating system. In my case, it's just httpd.

Once the software is installed, we can shut down the container and turn it into an image. Then, we can use this image to launch new containers. The advantage is that these new containers already have the software pre-installed.

Now, let's create this setup.

```
[root@ip-172-31-13-44 ~]# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS                                          NAMES
fd5085470bb6   centos    "/bin/bash"   32 minutes ago   Up 13 minutes   0.0.0.0:1234->80/tcp, :::1234->80/tcp   company_webserver1
[root@ip-172-31-13-44 ~]# docker stop company_webserver1

company_webserver1
[root@ip-172-31-13-44 ~]#
[root@ip-172-31-13-44 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
centos       latest    5d0da3dc9764   2 years ago   231MB
```

I have a container running named **'company_webserver1.'** First, I stopped this container using the **'docker stop'** command. After that, I checked the list of all the images installed on my operating system. Now, I'm going to turn this container into an image.

```
[root@ip-172-31-13-44 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
centos       latest    5d0da3dc9764   2 years ago   231MB
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                      PORTS     NAMES
fd5085470bb6   centos    "/bin/bash"   35 minutes ago   Exited (137) 2 minutes ago            company_webserver1
[root@ip-172-31-13-44 ~]# docker commit company_webserver1 webserver:v1
sha256:4faafac6067b64ee25f767783fef594b0f12f33d5738770cbbc282cac2e4f24a
[root@ip-172-31-13-44 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
webserver    v1        4faafac6067b   5 seconds ago    541MB
centos       latest    5d0da3dc9764   2 years ago      231MB
```

In my host operating system, I only have the CentOS image. I also have just one container, and it's currently in a stopped state. The container's name is 'company_webserver1.' Then, I ran this command:

- **Syntax: docker commit container-name image-name:tag**
- **Command: docker commit company_webserver1 webserver:v1**

This command transformed the Docker container into an image, and it was created just 5 seconds ago, as indicated by the image's timestamp.

Now, I'm going to remove the existing container and launch a new container using the **'webserver:v1'** image.

```
[root@ip-172-31-13-44 ~]# docker rm company_webserver1
company_webserver1
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-13-44 ~]#
```

For launching the container, I have run this command:

- **docker run -it -p 1234:80 -v /root/Host_Webpage/:/var/www/html/ --name company_webserver2 webserver:v1**

```
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-13-44 ~]# docker run -it -p 1234:80 -v /root/Host_Webpage/:/var/www/html/ --name company_webserver2 webserver:v1
[root@ed8f50e94c6f /]# rpm -q httpd
httpd-2.4.37-43.module_el8.5.0+1022+b541f3b1.x86_64
[root@ed8f50e94c6f /]#
```

When I run the command `**rpm -q httpd**`, it shows that the httpd software is already installed on my operating system. Now, I just need to start the httpd services using the command `**/usr/sbin/httpd -DFOREGROUND**`.

```
[root@ed8f50e94c6f /]# /usr/sbin/httpd -DFOREGROUND
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
o suppress this message
```

When I run this Command then the Httpd service is start and we can see the webpage.

If we have the requirement to share this image to other team mates then for this we need to export this image using command:

- **Syntax: docker save image-name:tag file-name.tar**
- **Command: docker save webserver:v1 server.tar**

```
[root@ip-172-31-13-44 ~]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED       SIZE
webserver     v1        4faafac6067b   3 hours ago   541MB
centos        latest    5d0da3dc9764   2 years ago   231MB
[root@ip-172-31-13-44 ~]# docker save webserver:v1 -o serverimage.tar
[root@ip-172-31-13-44 ~]# ls
Host_Webpage  serverimage.tar
```

Now Let's say this Image I share to one of my teammates and he wants to launch new container using this image for this. He needs to export the image using command:

- Syntax: docker load -i file-name.tar
- Command: docker load -i server.tar

```
[root@ip-172-31-13-44 ~]# ls
Host_Webpage  serverimage.tar
[root@ip-172-31-13-44 ~]# docker load -i serverimage.tar
Loaded image: webserver:v1
```

Here the problem is not 100% solved. For understanding this, Let's try to create this setup from the scratch.

For this I am going to launch one more container with the name of **company_webserver3** with the webserver Image.

```
[root@ip-172-31-13-44 ~]# docker run -it -v /root/Host_Webpage/:/var/www/html/  --name company_webserver3 webserver:v1
[root@328952023d90 /]# ls /var/www/html/
Container-file.txt  index.html
[root@328952023d90 /]# [root@ip-172-31-13-44 ~]#
[root@ip-172-31-13-44 ~]# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2289/sshd: /usr/sbi
tcp        0      0 127.0.0.1:46539         0.0.0.0:*               LISTEN      1962/containerd
tcp6       0      0 :::80                   :::*                    LISTEN      1969/httpd
tcp6       0      0 :::22                   :::*                    LISTEN      2289/sshd: /usr/sbi
[root@ip-172-31-13-44 ~]#
```

Imagine a new developer joins the company and, while trying to launch a new container, forgets to open a specific port for communication. Additionally, after launching the new container, they need to manually start the httpd web server by running a command inside the container or by using the `docker exec` command.

However, my requirement is that when the container starts, I want it to automatically expose port 1234 for communication, and I want the httpd web server service to start without the need for manual intervention. To achieve this, we can use a **Dockerfile**, which is like a set of instructions to tell Docker how to build and configure the container. This way, we can ensure that the container is set up exactly as we want it from the start, saving time and avoiding manual steps.

Dockerfile

A Dockerfile is a plain text configuration file used in Docker to define the steps and instructions for creating a Docker container image. It serves as a blueprint for building and configuring Docker containers. Here's an explanation of the keywords you mentioned within the context of a Dockerfile:

1. **FROM**: The FROM keyword is used to specify the base image upon which your Docker image will be built. It defines the starting point for your image, typically an operating system or another pre-configured image.
2. **LABEL**: The LABEL keyword is used to add metadata labels to your Docker image. These labels provide information about the image, such as the maintainer's name, version, or any other custom metadata.
3. **WORKDIR**: The WORKDIR keyword sets the working directory for any subsequent instructions in the Dockerfile. It specifies the directory path inside the container where commands should be executed.
4. **COPY**: The COPY keyword is used to copy files and directories from the host machine into the container. It is often used to add application code, configuration files, or other assets to the image.
5. **RUN**: The RUN keyword is used to execute commands inside the container during the image build process. It is commonly used for tasks such as installing software packages or running build scripts.
6. **CMD**: The CMD keyword specifies the default command to run when a container is started from the image. It can be overridden by specifying a different command when starting the container.
7. **ENV**: The ENV keyword is used to set environment variables within the container. These variables are used to configure the behavior of the containerized application.
8. **EXPOSE**: The EXPOSE keyword documents which ports the container will listen on at runtime. It does not actually publish the ports but provides information to users about which ports are intended to be exposed.

These keywords are fundamental building blocks in a Dockerfile, allowing you to define the image's base, metadata, working directory, file copying, command execution, environment configuration, and port information. A well-structured Dockerfile is key to creating efficient and reproducible Docker images.

```
# Specify the base image (e.g., an official Linux distribution)
FROM centos

# Set metadata (optional)
LABEL maintainer="Himanshu Kumar<himanshukr0007@gmail.com>"

# Run commands to configure the container
RUN sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*
RUN sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g'
/etc/yum.repos.d/CentOS-*
RUN dnf update -y
RUN yum install -y httpd net-tools
RUN echo httpd >> /root/.bashrc
# Switching Inside the WorkDir
WORKDIR /var/www/html/

# Copy files into the container
COPY ./index.html /var/www/html

# Expose ports
EXPOSE 80
```

Here I have created this Dockerfile which sets up a CentOS-based container with a web server (`httpd`) and an `index.html` file in the web server's document root.

1. **Base Image:** I am using CentOS as my base image.
2. **Metadata (Optional):** I have added some optional metadata like the maintainer's name and email.
3. **Configuration:** I made some changes to CentOS configuration files to use a different package source.
4. **Software Installation:** I have updated the package list and installed `httpd` (web server) and `net-tools` (networking utilities).
5. **Bashrc Modification**: Also added `**httpd**` to the `**.bashrc**` file.
6. **Working Directory:** Also set the working directory inside the container to `**/var/www/html/**`.
7. **File Copy:** Copied the `**index.html**` file from our local machine to the container's `**/var/www/html**` directory.
8. **Port Exposure:** Expose port 80, which is the default port for web traffic.

Now for creating the Image from this docker file we need to run the following command:

- **Syntax: docker build -t image-name:version ./Dockerfile**
- **Command: docker build -t webserver:v1 .**

The Dockerfile present in my current directory that is the reason I mentioned '.' Here.

```
[root@ip-172-31-13-44 Host_Webpage]# docker build -t company_webserver:v1 .
[+] Building 0.1s (13/13) FINISHED                                                              docker:default
 => [internal] load build definition from Dockerfile                                                      0.0s
 => => transferring dockerfile: 711B                                                                      0.0s
 => [internal] load .dockerignore                                                                         0.0s
 => => transferring context: 2B                                                                           0.0s
 => [internal] load metadata for docker.io/library/centos:latest                                          0.0s
 => [1/8] FROM docker.io/library/centos                                                                   0.0s
 => [internal] load build context                                                                         0.0s
 => => transferring context: 99B                                                                          0.0s
 => CACHED [2/8] RUN sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*                        0.0s
 => CACHED [3/8] RUN sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*   0.0s
 => CACHED [4/8] RUN dnf update -y                                                                        0.0s
 => CACHED [5/8] RUN yum install -y httpd net-tools                                                       0.0s
 => CACHED [6/8] RUN echo httpd >> /root/.bashrc                                                          0.0s
 => CACHED [7/8] WORKDIR /var/www/html/                                                                   0.0s
 => CACHED [8/8] COPY ./index.html /var/www/html                                                          0.0s
 => exporting to image                                                                                    0.0s
 => => exporting layers                                                                                   0.0s
 => => writing image sha256:dacd6830e2b90c9b835c27b922727621a1001a0c7f696f52068462d865dc1478              0.0s
 => => naming to docker.io/library/company_webserver:v1                                                   0.0s
```

```
[root@ip-172-31-13-44 Host_Webpage]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED            SIZE
company_webserver   v1        dacd6830e2b9   25 minutes ago     584MB
webserver           v1        4faafac6067b   6 hours ago        541MB
centos              latest    5d0da3dc9764   2 years ago        231MB
```

The Image is built successfully, and Now I am going to launch new container with this image.

docker run -dit -p 1234:80 --name os1 company_webserver:v1

```
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ip-172-31-13-44 ~]# docker run -dit -p 1234:80 --name os1 company_webserver:v1
29b4152705676dac9f8a72d723ad0bcbc5f08fc68731bfaebc60494a4af9ba28
```

When I start this container, the web server will automatically initiate and open port 80. The public can access this web server using port 1234.

```
[root@ip-172-31-13-44 ~]# docker ps -a
CONTAINER ID   IMAGE                  COMMAND       CREATED         STATUS         PORTS                                   NAMES
29b415270567   company_webserver:v1   "/bin/bash"   4 seconds ago   Up 3 seconds   0.0.0.0:1234->80/tcp, :::1234->80/tcp   os1
```

Now let's learn about how Dockerfile help us to maintain the versioning:

If you've already configured the webserver and want to update or modify the content of the webpage, you can achieve this using a Dockerfile. In the webpage file, I've provided an introduction to the fundamental concepts of Docker.

Creating New image with Dockerfile and also maintaining the version.

docker build -t company_webserver:v2

```
[root@ip-172-31-13-44 Host_Webpage]# docker build -t company_webserver:v2 .
[+] Building 0.2s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 711B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/centos:latest
```

We can see we have 2 images with the name of company_webserver but they have different versions.

```
[root@ip-172-31-13-44 Host_Webpage]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
company_webserver   v2        d38340f454af   About a minute ago   584MB
company_webserver   v1        dacd6830e2b9   19 hours ago         584MB
webserver           v1        4faafac6067b   24 hours ago         541MB
centos              latest    5d0da3dc9764   2 years ago          231MB
```

THANK YOU FOR READING