



HOW TO SECURELY CONFIGURE A LINUX HOST TO RUN CONTAINERS



Twistlock
Enterprise Security. DevOps Agility.

HOW TO SECURELY CONFIGURE A LINUX HOST TO RUN CONTAINERS

To run containers securely, one must go through a multitude of steps to ensure that a) the environment – the host & the operating system – is set up correctly and hardened, b) the images and containers are configured in a robust fashion, and c) the system has the necessary integration with components like enterprise directories, SIEM systems, etc.

This guide focuses on concrete steps to configure a host to run Docker containers securely in production.

An out-of-the box installation of just about any Linux distribution would be capable of running the Docker daemon and Docker containers, but it would leave your host exposed to many security and performance concerns. In this guide, we will use a minimal install of CentOS 7, as an example host, to illustrate specific configuration steps to harden the host and the OS. CentOS has benefited from the Red Hat Enterprise development ecosystem and has proven to be a stable and secure Linux distribution for data centers.

If you choose to use a different distribution, the practices suggested in this guide are still relevant but will require that the provided sample commands be translated to your target environment.

There are three main steps that are required to configure a basic CentOS host for production.

- Strip the operating system of any extra services and software so that only the tools and services required to run Docker are up and running.
- Install and configure the Docker daemon to run your containers. This includes both performance and security settings that configure Docker to be more suitable for a production environment rather than a development environment.
- Configure the firewall to only allow incoming traffic for SSH by default and open ports on-demand that are required by the containers for external communication.

INSTALL CENTOS 7

Minimal Install

This guide assumes a minimal install of CentOS 7 from the official image. If you have previously installed CentOS, there are some additional steps you might want to consider:

- Remove all development tools (compilers, etc.)
- Remove all services listening so that only port 22 is exposed for SSH access. Our firewall configuration will stop traffic intended for any other running services but it is best practice to stop and remove these additional services.

Install nmap

Nmap is an open source security tool that can be used to gather information about a network or host. We will use it to scan for open ports and listening services on our host.

sudo yum install nmap -y

The -y option answers [yes] silently for any prompts from yum. Typically these are confirmation messages relating to downloading and installing packages.

```
Installed:
  nmap.x86_64 2:6.40-7.el7

Dependency Installed:
  nmap-ncat.x86_64 2:6.40-7.el7

Complete!
[root@localhost ~]#
```

Update system sources

Updating the system sources ensures that all of your libraries and programs are running the latest versions. This is good practice to do for any new Linux installation, not just for production servers.

sudo yum update -y

```
nss-util.x86_64 0:3.21.0-2.2.el7_2
openldap.x86_64 0:2.4.40-9.el7_2
openssh-clients.x86_64 0:6.6.1p1-25.el7_2
openssl.x86_64 1:1.0.1e-51.el7_2.5
pcrpe.x86_64 0:8.32-15.el7_2.1
procps-ng.x86_64 0:3.3.10-5.el7_2
python-libs.x86_64 0:2.7.5-39.el7_2
python-pyudev.noarch 0:0.15-7.el7_2.1
selinux-policy.noarch 0:3.13.1-60.el7_2.9
sudo.x86_64 0:1.8.6p7-17.el7_2
systemd-libs.x86_64 0:219-19.el7_2.13
teamd.x86_64 0:1.17-7.el7_2
tzdata.noarch 0:2016f-1.el7
numactl-libs.x86_64 0:2.0.9-6.el7_2
openssh.x86_64 0:6.6.1p1-25.el7_2
openssh-server.x86_64 0:6.6.1p1-25.el7_2
openssl-libs.x86_64 1:1.0.1e-51.el7_2.5
polkit.x86_64 0:0.112-7.el7_2
python.x86_64 0:2.7.5-39.el7_2
python-perf.x86_64 0:3.10.0-327.36.1.el7
rdma.noarch 0:7.2.4.1_rc6-2.el7
selinux-policy-targeted.noarch 0:3.13.1-60.el7_2.9
systemd.x86_64 0:219-19.el7_2.13
systemd-sysv.x86_64 0:219-19.el7_2.13
tuned.noarch 0:2.5.1-4.el7_2.6
util-linux.x86_64 0:2.23.2-26.el7_2.3

Complete!
[root@localhost ~]#
```

Create a new user

We don't want to do everything on this host using our root user, so it is a good practice to add a new user and execute the rest of commands as that user.

For example: Add a new user with the username dockeruser

- 1 | Add a new user to the host.
adduser dockeruser
- 2 | Set a password for the new user
passwd dockeruser
- 3 | Add user to the wheel group to give sudo access
usermod -a -G wheel dockeruser

```
[root@localhost ~]# adduser dockeruser
[root@localhost ~]# passwd dockeruser
Changing password for user dockeruser.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]# usermod -a -G wheel dockeruser
[root@localhost ~]#
```

Generate keys for SSHD

A security best practice is to disable password based authentication for SSH. To do this we will generate public/private keys and copy them over to the server as authenticated keys. The steps below will show how to generate keys for a linux or OSX machine. (To generate keys for Windows please see one of the great guides online)

- 1 | On your client machine generate ssh key
ssh-keygen -t rsa
- 2 | Running this command will create two files in your home directory's .ssh directory
 - a. id_rsa -- This file is your private key
 - b. id_rsa.pub -- This file is your public key
- 3 | Log in to your Docker host as dockeruser
mkdir ~/.ssh
sudo chmod 700 ~/.ssh
- 4 | Copy the public key to your Docker host. For Example
scp ~/.ssh/id_rsa.pub dockeruser@10.0.0.37:~/.ssh/authorized_keys
- 5 | You should now be able to login to your host without using a password from the client system
ssh dockeruser@10.0.0.37

Disable root login and password based authentication ssh

We don't want people to be able to login to our system using the root user. Additionally, we don't want to allow people to login using a password either. This helps us protect against brute force attacks against our users. In the last section, we added a new user to the system and copied their public keys to the server. This user should be used for the rest of your configuration.

- 1 | Login to your host using the user added in the last section
- 2 | Edit the SSH daemon configuration to disable root logins
sudo vi /etc/ssh/sshd_config
- 3 | Find the line with the following text
#PermitRootLogin yes
- 4 | Change the line to read
PermitRootLogin no
- 5 | Find the line with the following text
#PasswordAuthentication yes
- 6 | Change the line to read
PasswordAuthentication no
- 7 | Restart SSHD
sudo systemctl restart sshd.service

Stop any services other than SSHD

Ideally, you only want ssh listening on your container host when no containers are running. This host is dedicated to running our containers and as such should not be running any additional services if possible. It is advisable to change the port ssh listens on to further enhance security. For this guide we will leave to listen on the default port 22.

- 1| List any open and listening ports

sudo nmap -sU -sS -p 1-65535 localhost

```
[dockeruser@localhost ~]$ sudo nmap -sU -sS -p 1-65535 localhost
[sudo] password for dockeruser:

Starting Nmap 6.40 ( http://nmap.org ) at 2016-09-25 16:01 PDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000050s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 131064 closed ports
PORT      STATE      SERVICE
22/tcp    open       ssh
25/tcp    open       smtp
68/udp    open|filtered dhcpc
323/udp   open|filtered unknown
25054/udp open|filtered unknown
55993/udp open|filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 3.23 seconds
[dockeruser@localhost ~]$
```

In this case, we have two open and listening TCP ports. Port 22 is open to run our ssh listener and port 25 is open for SMTP traffic. By default, CentOS 7 comes with postfix installed.

- 2| Stop and Remove postfix

- a. Stop postfix service

sudo systemctl stop postfix

- b. Check that postfix is no longer listening on port 25

sudo nmap -sU -sS -p 1-65535 localhost

```
Starting Nmap 6.40 ( http://nmap.org ) at 2016-09-25 16:05 PDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000050s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 131065 closed ports
PORT      STATE      SERVICE
22/tcp    open       ssh
68/udp    open|filtered dhcpc
323/udp   open|filtered unknown
25054/udp open|filtered unknown
55993/udp open|filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 3.11 seconds
[dockeruser@localhost ~]$
```

- c. Remove postfix from our host

sudo yum remove postfix

Installing and Configuring Docker

Install Docker

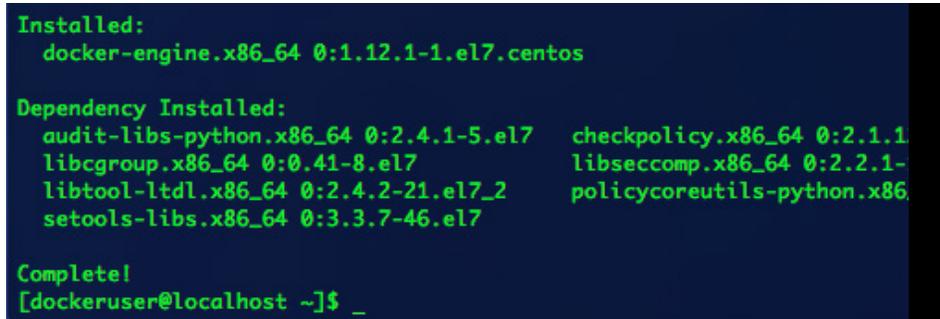
1 | Configure yum to find the Docker repository

- a. Create a new file to hold the Docker repository's information
`sudo vi /etc/yum.repos.d/docker.repo`
- b. Add the following contents to the file:

```
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
enabled=1
gpgcheck=1
gpgkey= https://yum.dockerproject.org/gpg
```
- c. Save the file

2 | Install the Docker package

sudo yum install docker-engine -y



```
Installed:
docker-engine.x86_64 0:1.12.1-1.el7.centos

Dependency Installed:
audit-libs-python.x86_64 0:2.4.1-5.el7      checkpolicy.x86_64 0:2.1.1
libcgroup.x86_64 0:0.41-8.el7              libseccomp.x86_64 0:2.2.1-
libtool-ltdl.x86_64 0:2.4.2-21.el7_2      policycoreutils-python.x86
setools-libs.x86_64 0:3.3.7-46.el7

Complete!
[dockeruser@localhost ~]$ _
```

3 | Start the docker engine

sudo service docker start

Alternate: systemctl start docker

4 | Verify the docker engine is running correctly

sudo docker run hello-world

```
[dockeruser@localhost ~]$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdc9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

[dockeruser@localhost ~]$
```

5 | Configure the Docker daemon to start at boot

sudo chkconfig docker on

Alternate: sudo systemctl enable docker

```
[dockeruser@localhost ~]$ sudo chkconfig docker on
Note: Forwarding request to 'systemctl enable docker.service'.
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service
to /usr/lib/systemd/system/docker.service.
[dockeruser@localhost ~]$ _
```

Configure Docker

We need to configure Docker for use in a production environment by configuring increased security and adding container resource consumption limits. We do this by passing command line arguments to the Docker Daemon. The command line operands we will use are summarized below.

--icc=false

Disable inter-container communication. This is a more secure default than allowing containers to communicate with each other.

- If you know your application topology, you can enable specific containers to talk to each other using the `--link` parameter (`--link=CONTAINER_NAME_or_ID:ALIAS`)
- If you need to have containers talking to each other but are unsure of your application topology, you can set this to true but this is somewhat insecure in that it allows full network communication between all containers.

--log-level="info"

Set the log level to info. Logging is available at several levels, but can sometimes be very verbose. Log level info is the preferred logging level to get the information you need while not logging information that won't be as useful and takes up a lot of space on disk.

--iptables=true

Enable addition of iptables rules. We want to allow Docker to change iptables rules based on the requirements of the containers. For example, we start a container that is supposed to listen on port 80 to server web requests. By default, in a secure system, we aren't allowing any traffic on port 80 due to our firewall rules. This will allow Docker to decide that traffic on port 80 should be allowed while the container is running.

--default-ulimit

Set default ulimits for your containers. This operand will help you set up soft and hard limits on the number of processes and files. This helps ensure that a container doesn't greedily consume the host's resources rendering the machine (and other containers) inaccessible. These numbers need to be tuned for your specific application / containers but below is an example with sensible defaults.

- `--default-ulimit nproc=1024:2048 --default-ulimit nofile=1020:2048`

--disable-legacy-registry=false

Limit communication to registries that are running version 2 of the Docker protocol. Version 2 has many features that enhance security such as image provenance and image signing, but also brings performance improvements as well.

- 1 | Edit the docker.service file
sudo vi /usr/lib/systemd/system/docker.service
- 2 | Find the line that reads
ExecStart=/usr/bin/dockerd
- 3 | Modify that line to read
**ExecStart=/usr/bin/dockerd --icc=false --log-level="info" --iptables=true
--default-ulimit nproc=1024:2408 --default-ulimit nofile=1024:2048
--disable-legacy-registry=false**
- 4 | Reload units
sudo systemctl daemon-reload
- 5 | Restart the Docker daemon
sudo service docker restart
- 6 | Verify command line operands are now passed to Docker
ps -eaf | grep docker

```
root      38367      1  0 16:57 ?        00:00:00 /usr/bin/dockerd --icc=false --log-level=
"info" --iptables=true --default-ulimit nproc=1024:2408 --default-ulimit nofile=100:200 --d
isable-legacy-registry=false
```

Configure iptables

Iptables is the firewall we will use to secure this system. At a high level, the firewall configuration should drop all input and forwarding traffic, and then whitelist ssh traffic. By default, CentOS 7 ships with firewalld for configuring iptables. For our installation, we will disable firewalld and use iptables services for firewall configuration.

- 1 | Disable firewalld
sudo systemctl disable firewalld
- 2 | Install iptables-services
sudo yum install iptables-services -y
- 3 | Enable iptables services
sudo systemctl enable iptables
- 4 | Configure the iptables rules (In the final rule replace <serverip> with your server's ip-address without the <>
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -A INPUT -p tcp -s 0/0 -d <serverip> --sport 513:65535 --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT
- 5 | Save the iptables rules
sudo service iptables save
- 6 | Verify iptables rules
sudo iptables --list

TESTING BY INSTALLING NGINX

Let's test our Docker installation by downloading and running nginx. Nginx is a popular lightweight webserver.

- 1 | Download image
sudo docker pull nginx (alpine / nginx:latest)
- 2 | Start the nginx container
sudo docker run --name docker-nginx -p 80:80 nginx
Note: -p 80:80 maps the container's port 80 to the hosts external port 80.
This will modify our iptables rules to allow for traffic to flow over port 80
- 3 | Connect to the host's ip-address on port 80
For example: `http://10.0.0.160`

You should see the following in your browser:

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

SUMMARY

Configuring a host to run containers in production in a secure fashion requires attention to both security and performance concerns. This guide presents a starting point for a production host and it is up to the system administrator to keep the system up to date and tuned going forward. As new security issues and vulnerabilities emerge, you need to stay on top of security patches and keep your system updated, which is a topic that we will cover in a future guide. It is also good practice to incorporate some kind of auditing to ensure you know who is doing what on your host. We hope you find this guide useful in helping you take the steps necessary to run Docker securely in your production environment.

ABOUT TWISTLOCK



ENTERPRISE SECURITY. DEVOPS AGILITY.

Twistlock protects today's applications from tomorrow's threats with advanced intelligence and machine learning capabilities. Automated policy creation and enforcement along with native integration to leading CI/CD tools provide security that enables innovation by not slowing development. Robust compliance checks and extensibility allow full control over your environment from developer workstations through to production. As the first end-to-end container security solution, Twistlock is purpose-built to deliver modern security.

LEARN MORE AT

Twistlock.com