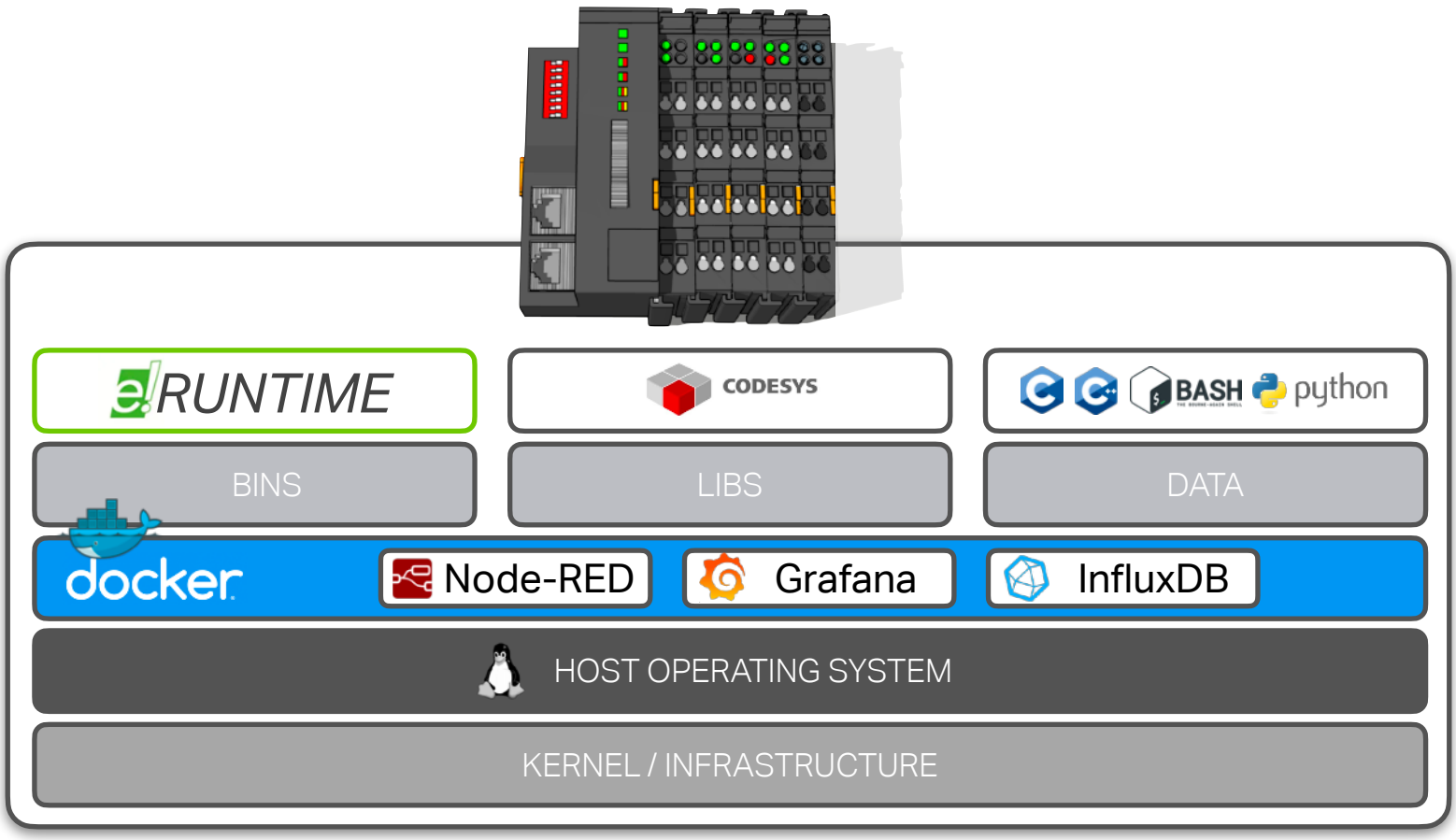
GETTING STARTED WITH:


# WHAT IS DOCKER

## BACKGROUND

Docker is an open containerization platform for developing, shipping, and running applications. Docker allows developers to package applications for scaled deployment, swift installation, and isolation in operation.
Alongside the modularity that Docker brings, there is also a virtualization component.  This means that very light weight virtual environments can be used to supplement platform features and functions.




## WHY USE DOCKER

On the WAGO Platform, Docker is used to build and run applications without the need for complex compilers or cross-build tools.  Take advantage of simple commands and orchestration tools to install images and run containers.

Popular Docker applications include:
- Node-RED
- InfluxDB
- Python Framework
- VPN / Networking Platforms

## DOCKER TERMINOLOGY

It is important to understand some general terms that we will use in this guide.

**Image**:
Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.

**Container**:
A Docker container is a runtime instance of a docker image consisting of:

- A Docker image
- An execution environment
- A standard set of instructions

**Volume**:
Volumes are specially-designated directories within one or more containers designed to persist data, independent of the container's life cycle.  There are three types of volumes:

- A **host** volume lives on the Docker host's filesystem and can be accessed from within the container
- A named **volume** is a volume which Docker manages where on disk the volume is created, but it is given a name
- An **anonymous** volume is similar to a named volume, however, it can be difficult to refer to the same volume over time when it is an anonymous volume. Docker handles where the files are stored

## DOCKER HUB

The Docker Hub (https://hub.docker.com/) will provide the platform where we will "pull" images and run as containers.  It is possible to use other repositories for this as well like AWS Elastic Container Registry or private repositories.

For simplicity we will not modify the configured registries.  See third-party registry documentation for instructions on using their platform with Docker.

## COMPATIBLE WAGO HARDWARE

WAGO supports the Docker engine on two different Linux systems:
WAGO Embedded Linux (with Real Time patch):

PFC200 Gen 2

- 750-8210
- 750-8211
- 750-8212
- 750-8213
- 750-8214
- 750-8215
- 750-8216
- 750-8217

Touch Panel 600

- 762-410__ (all sizes)
- 762-420__/8000-0001 (all sizes)
- 762-430__/8000-0002 (all sizes)
- 762-520__/8000-0001 (all sizes)
- 762-530__/8000-0002 (all sizes)
- 762-620__/8000-0001 (all sizes)
- 762-630__/8000-0002 (all sizes)

Edge Controller

- 753-8303/8000-002

WAGO Debian Linux:

Edge Computers

- 752-9400
- 752-9401
- 752-9800

# SSH / SHELL INTERFACE

This guide will use a terminal to access the shell.  The terminal is just that, a window that allows you to access and interact with the shell servers interface.  There are several terminal programs.  Popular programs include:
Windows Based:

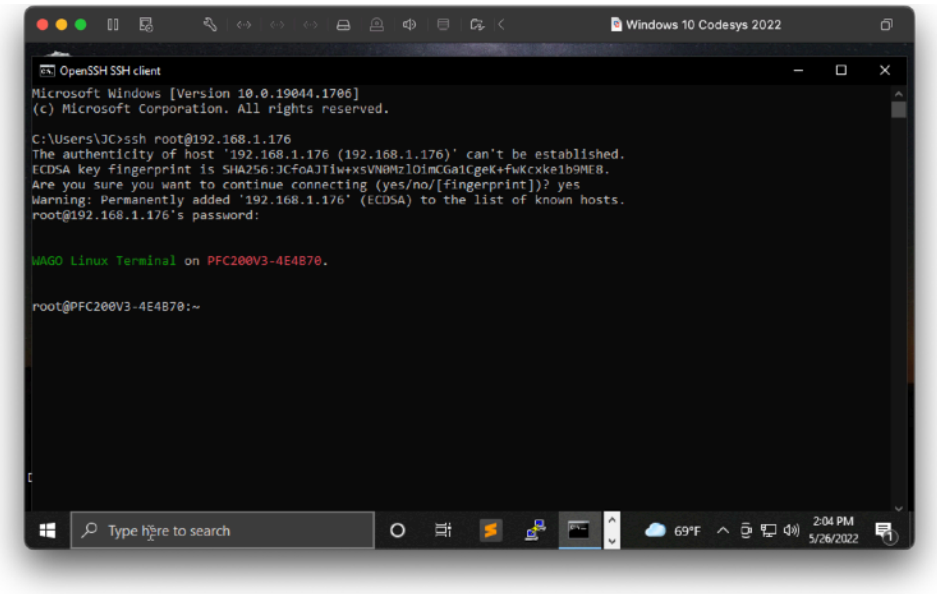
- Command Prompt(cmd **if enabled)
- PuTTY (https://www.putty.org/)

Linux:

- Terminal

Mac:

- Terminal
- Terminus (https://termius.com/mac-os)




A Windows 10 Command Terminal

## CREDENTIALS

It is important to know that the credentials for the Linux shell are not the same for all WAGO platforms.  Use the following default credentials for the respective platform:
PFC100, PFC200, Edge Controller, Touchpanel 600
- username: **root** password: **wago**
- username: **admin** password: **wago** (admin user does not have docker exec privileges)

Edge Computer (752-9400, 750-9401, 750-9800
- username: **edge** password: **wago**

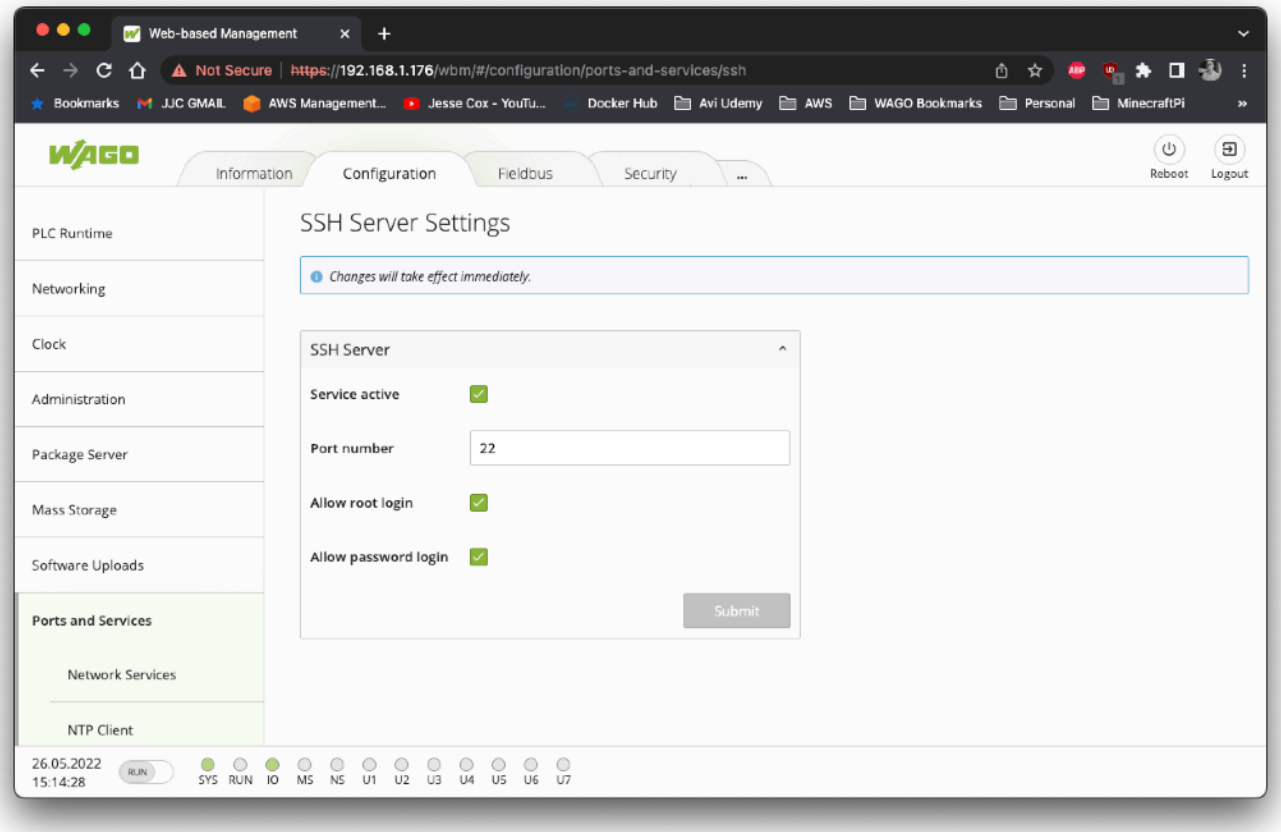
## SECURITY

Because the shell makes deep access to the Linux resources, you will need to take steps to keep this environment secure.  Change all default password and disable the shell when not needed.  Practice proper security policy as highlighted with (https://www.nist.gov)

# GENERAL NETWORKING

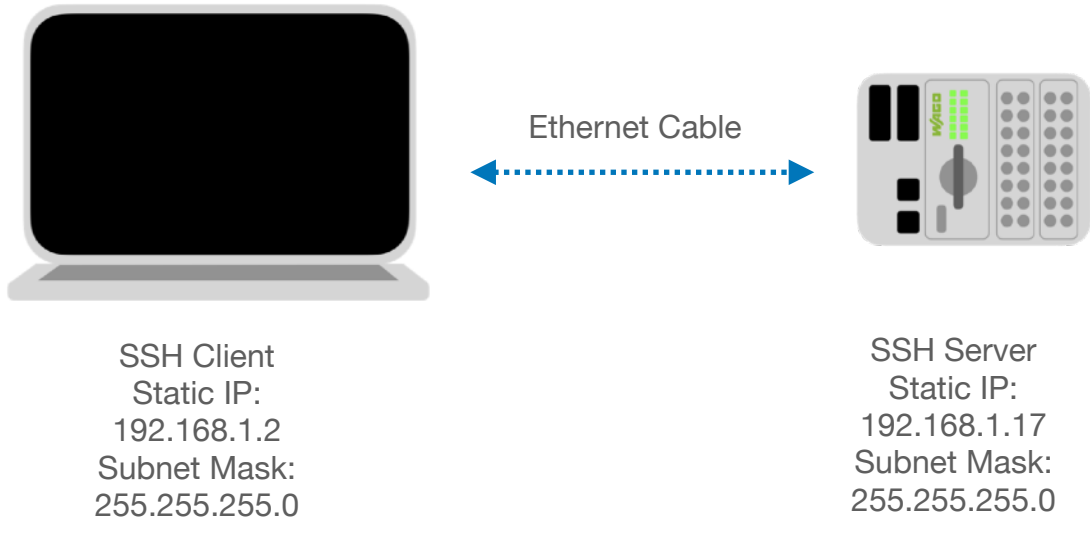
Connecting to the shell server (WAGO device) from the client (PC) requires both devices to be on the same Ethernet network, configured to send/receive connections.  A typical network follows the following rules:

1. Both devices are on the same network IP range
2. Both devices are configure with the same subnet mask
3. The SSH port is open and the service is active on the server ( this is enabled by default on the WAGO standard firmware)



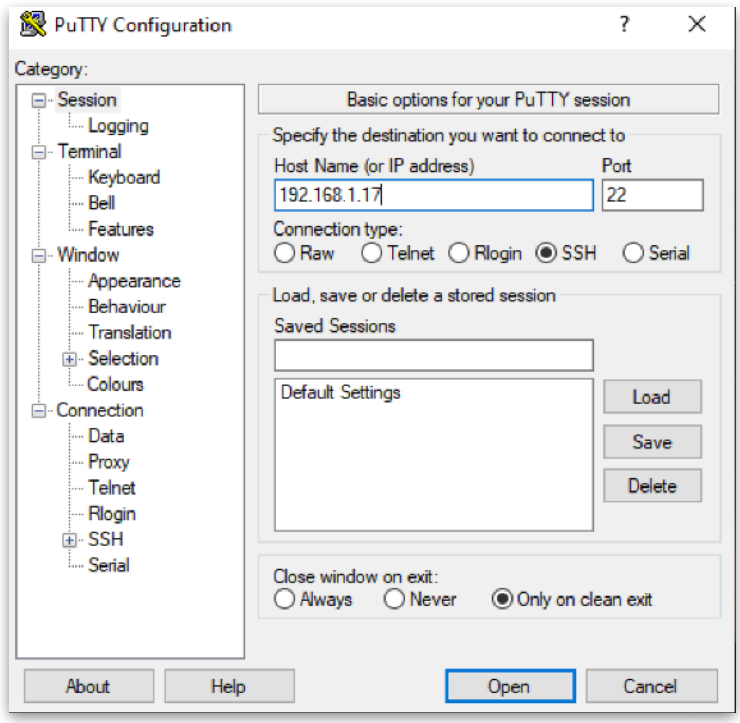

***See WBM >> Configuration >> Ports and Services >> SSH to customize settings

An example for network configuration / settings could be:

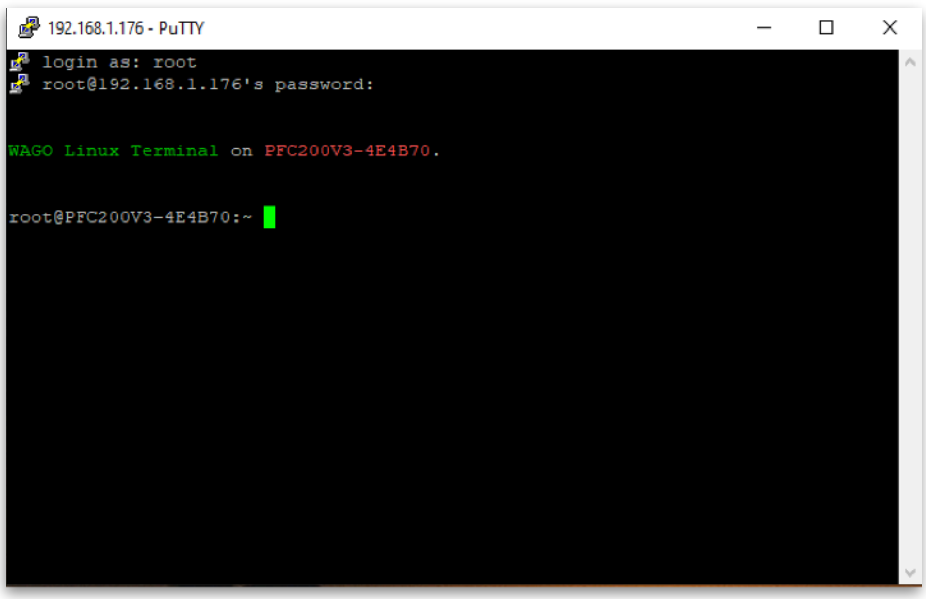


# CONNECTING THE TERMINAL (WINDOWS)

Because not every Windows system can use the SSH client in the Command Prompt, this guide will use the PuTTY Software (https://www.putty.org/). Download and install PuTTY to your Windows machine.




**Step 1:** Open the program and access the WAGO device by entering the controller IP address. The default port should be 22, this is standard for most SSH servers.




**Step 2:** Use default login credentials:

user: **root**

password: **wago**

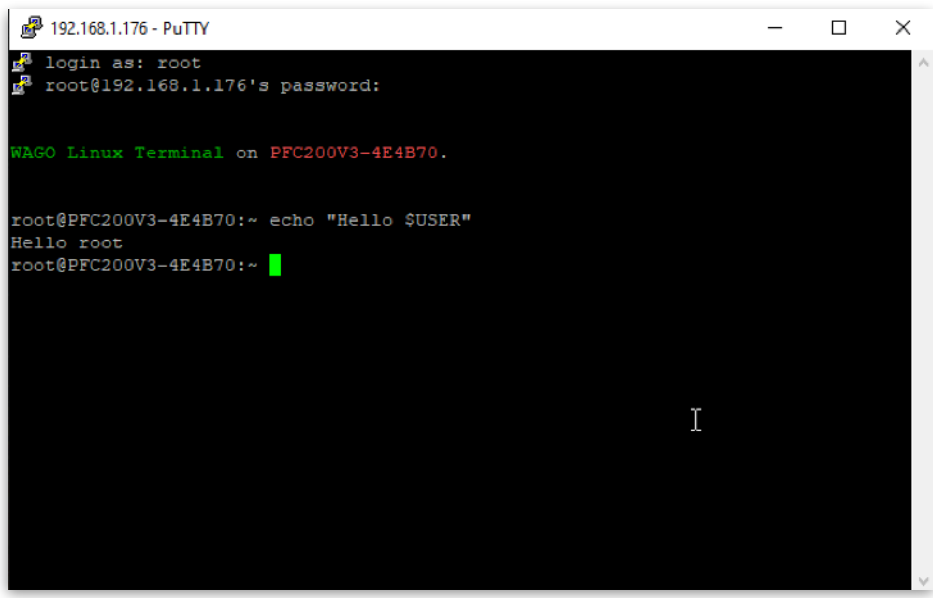
You will be prompted to change the password. Make a note of this and keep it safe.

**Step 3:** You should now be connected to the SSH server. Run the following command:

**echo "Hello $USER"**

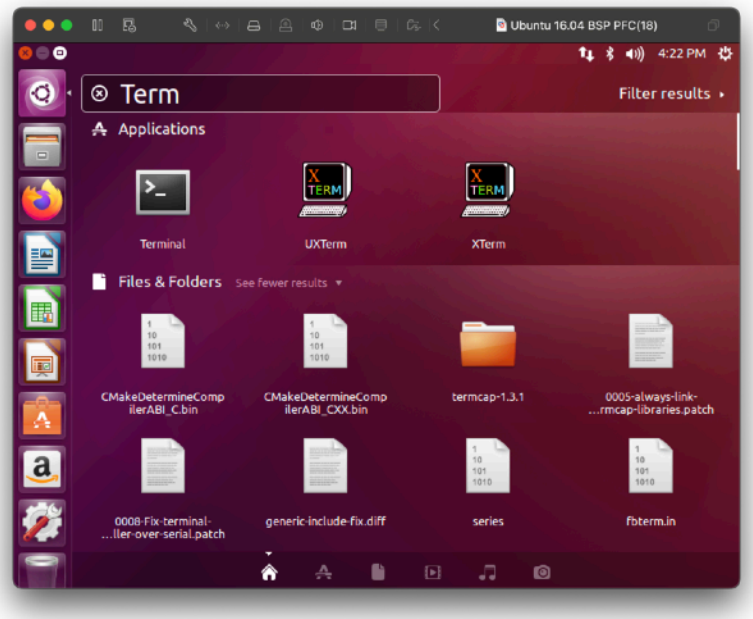
You should see a print out in the terminal:

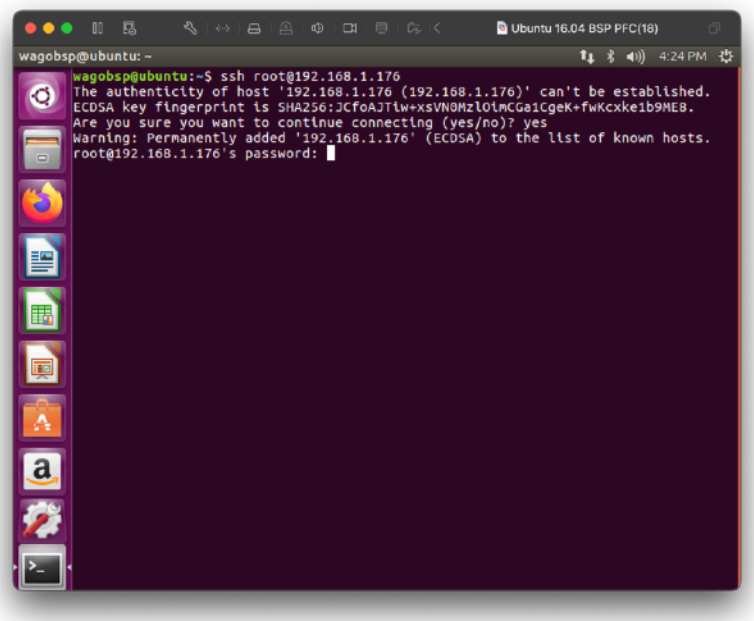
**Hello root**



# CONNECTING THE TERMINAL (LINUX & MAC)

**Step 1:** Linux and Mac systems are equipped with both a terminal and an SSH client. In either platform, launch the application “Terminal”




**Step 2:** Invoke the SSH client connection from the shell with the following command:

```
ssh root@192.168.1.17
```

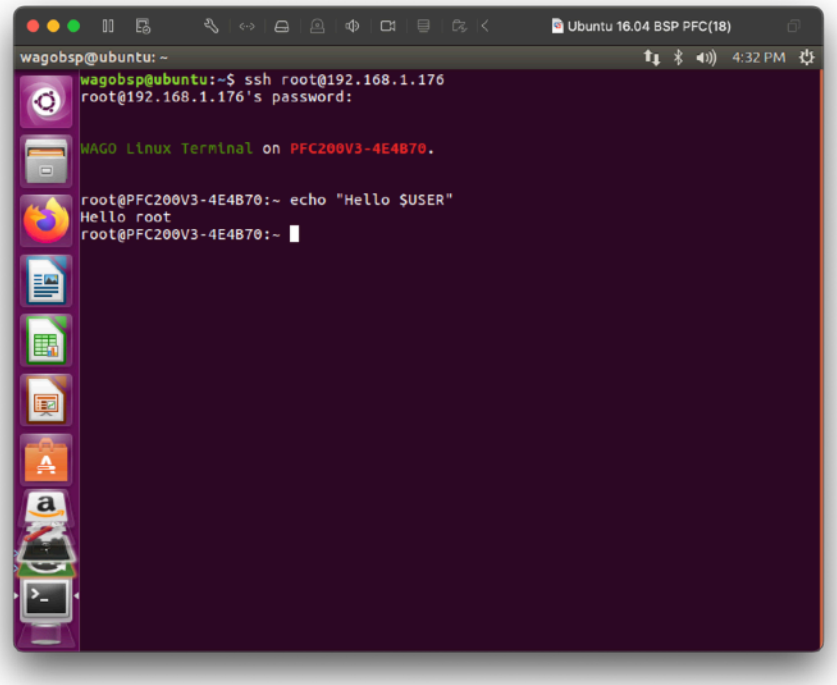
Answer “yes” if prompted to add the user and key to known fingerprint. Enter your password when prompted (default: “wago”).

**Step 3:** You should now be connected to the SSH server. Run the following command:

```
echo “Hello $USER”
```

You should see a print out in the terminal:

```
Hello root
```

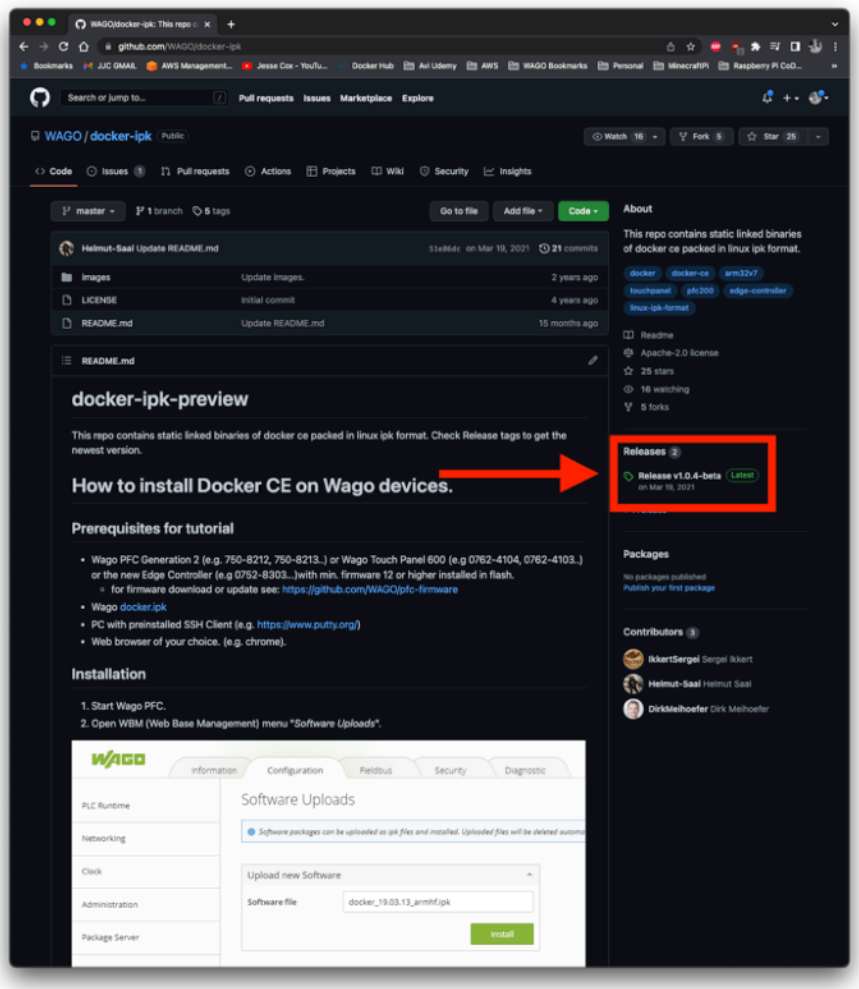


# INSTALLING DOCKER

WAGO began delivering controllers with Docker pre-installed with firmware version 20. This means any device with firmware 19 and lower, requires installing the Docker engine with an ipk package.

## DOCKER ENGINE ON GITHUB

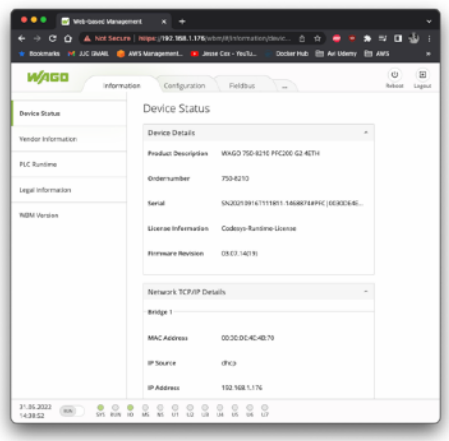
Many valuable resources can be found on WAGO's GitHub Organization page (https://github.com/wago). Here you will find a repository containing the Docker ipk (https://github.com/WAGO/docker-ipk) Current install files can be found under releases. Download the ipk file from the current release.

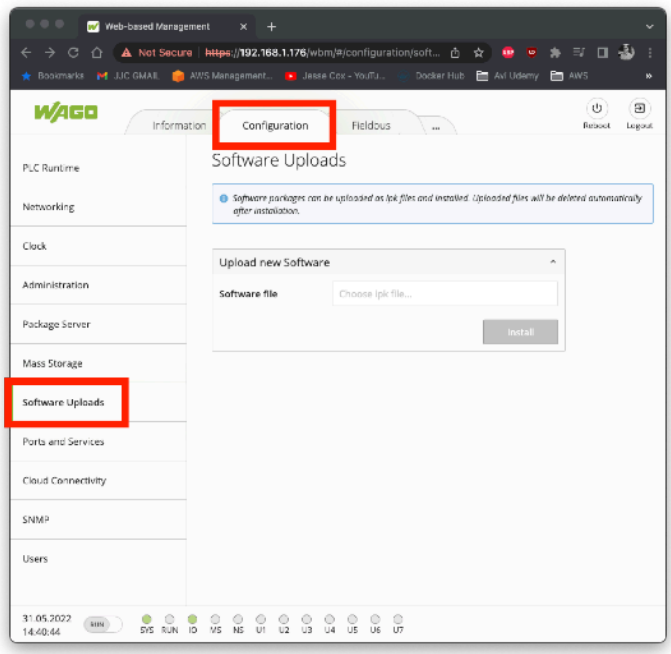



## INSTALLING DOCKER VIA WEB BASED MANAGEMENT

To install the Docker engine ipk file via WBM:

**Step 1:** Open the WBM by navigating to the controller's IP address in your web browser.

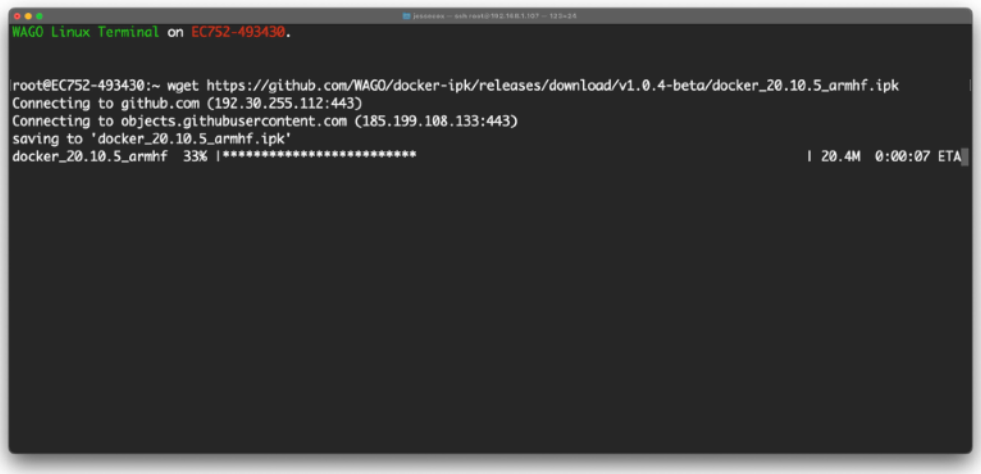



**Step 2:** Navigate to Configuration >> Software Uploads. Click the "Software File" box and select the Docker ipk file. Click "Install". ** You may need to force the install if updating the Docker version.

# INSTALLING DOCKER VIA CLI (COMMAND LINE INTERFACE)

Installing the Docker engine using the CLI has advantages. You can see the install process workflow, force a reinstall, or force a downgrade for compatibility.

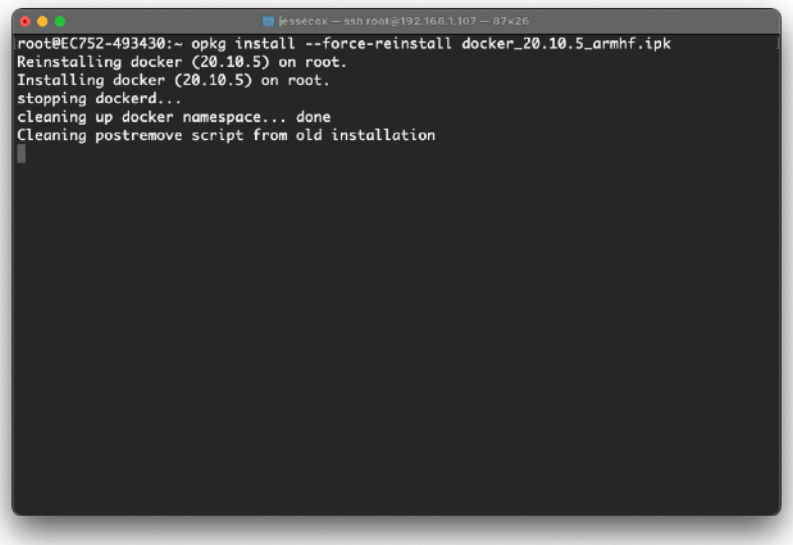


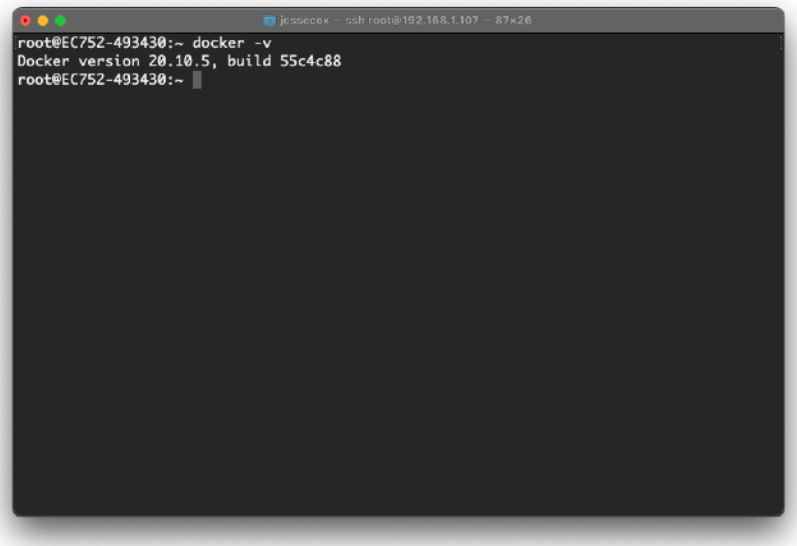

**Step 1**: Use the wget command to download the ipk from GitHub

```
wget https://github.com/WAGO/docker-
ipk/releases/download/v1.0.4-beta/
docker_20.10.5_armhf.ipk
```

**Step 2**: Run the install command on the downloaded ipk

```
opkg install docker_20.10.5_armhf.ipk
     ** optional flags   --force-reinstall
                         --force-downgrade
```




**Step 3**: Test the Docker install

```
docker -v
```

This should print out the current installed version of the Docker engine.

**Troubleshooting** : If you are having trouble installing Docker, check the following:

- PLC has connection to the internet (test with `ping www.google.com`)
- PLC time and timezone are set correctly
- There is sufficient storage capacity for the Docker package (run `df -h` to verify)
- You are using a Docker compatible device (see page 5)

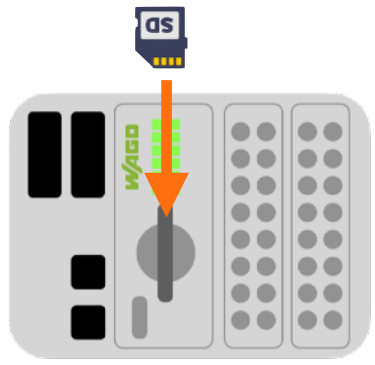
# CONFIGURING DOCKER

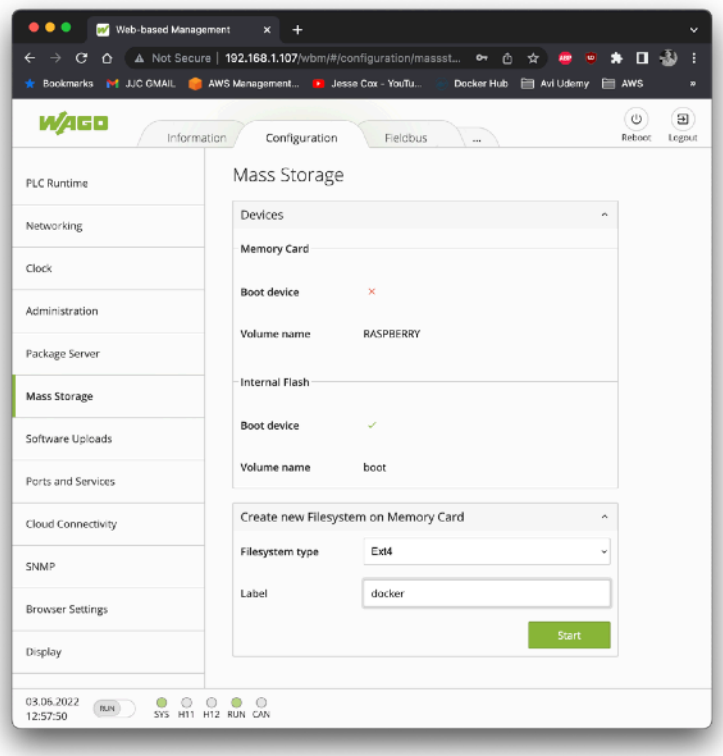
Docker uses a configuration file to determine where resources are stores, what format logs are create in, and how big those logs can be.  No configuration is needed if sing default location.

It may be necessary to change the data-root location when images, or volumes exceed the available size of the local file system.

For example - to move the docker data-root to an SD card follow these steps:

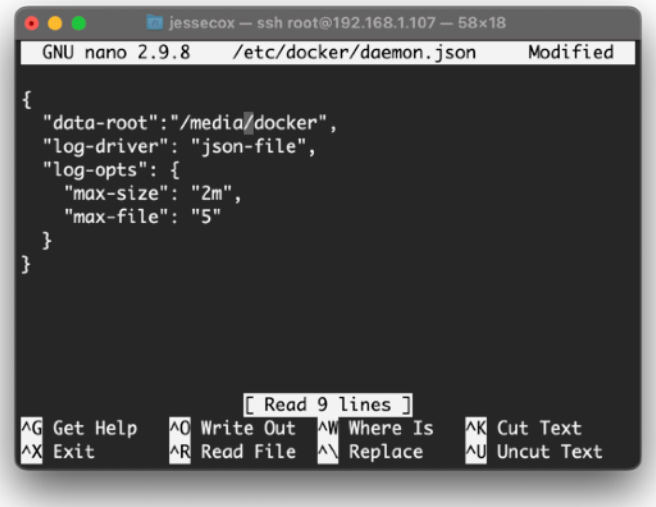
**Step 1**: Insert a compatible SD card into the controller

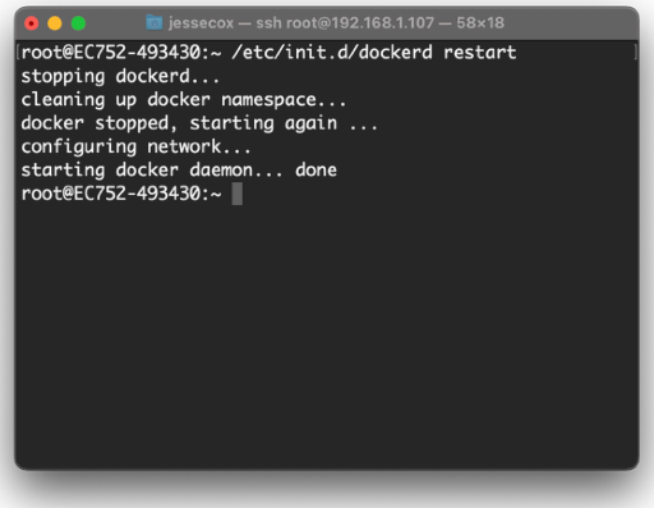





**Step 2**: In WBM navigate to Configuration >> Mass Storage and format the SD card to “Ext4” with the Volume Name “docker”

**Step 3**: use “nano” in the shell to change the docker “data-root” location to “/media/docker”




**Step 4**: Restart the docker daemon in the shell with:

```
/etc/init.d/dockerd restart
```

# WORKING WITH DOCKER

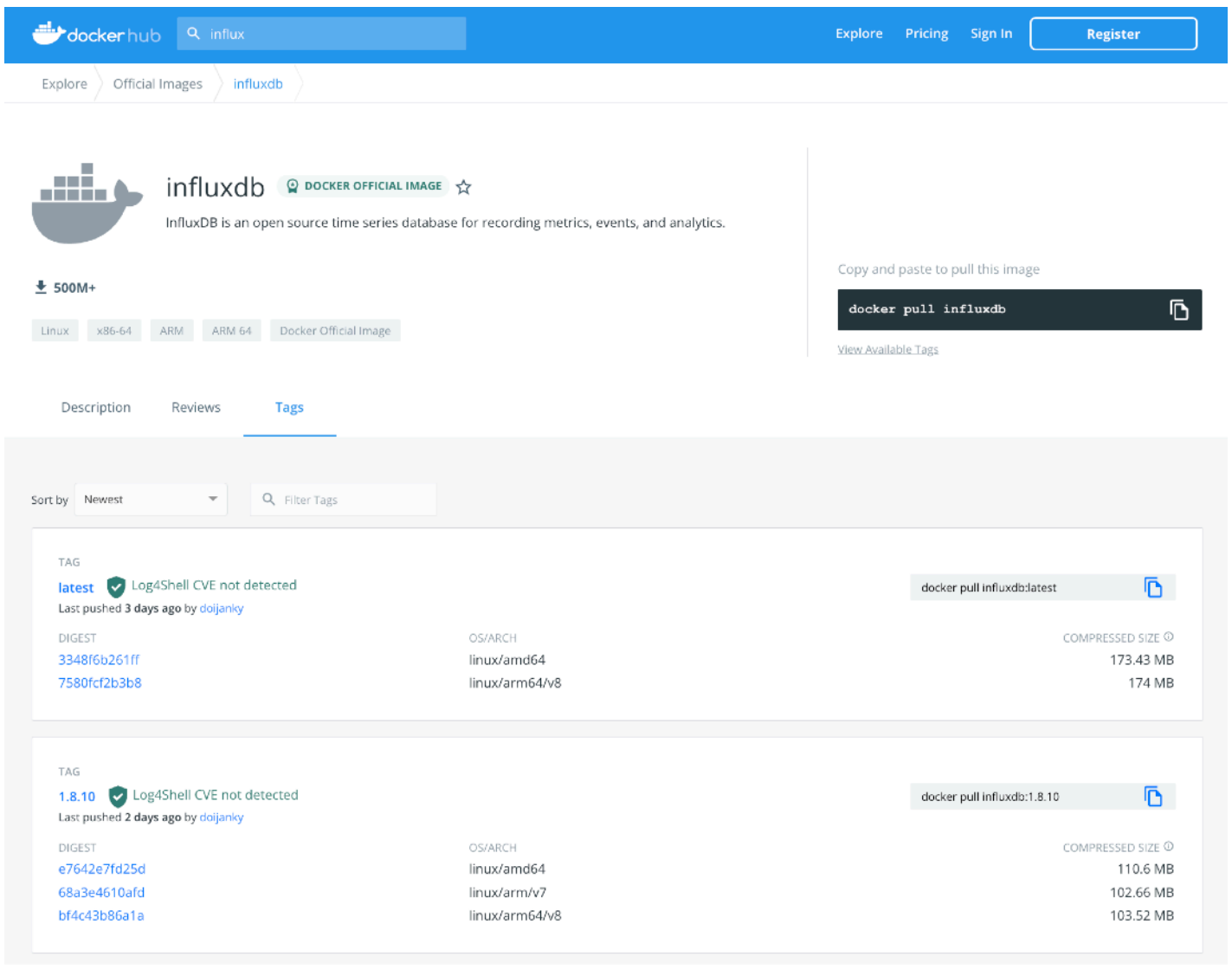
With Docker installed, you can begin working with the Docker API, pre-built Docker images, and building your own Docker resources. This guide will step through a simple workflow to pull a Docker package, run this as a container, and attach storage to allow data to persist, even when the container is stopped and removed

## DOCKER PULL

**Syntax**: `docker pull [IMAGE NAME]:[TAG NAME]`

Docker pull command is a command to download an image from the Docker Hub and save it to local images.  It is very simple but attention should be paid to the compatible architectures.  If you do not specify a tag, “latest” will be assumed.  This can cause problems at runtime.
For example, this influxdb image has wide platform support, but if you look closely, the latest tag does not support arm architecture.  For this images you would need to specify tag “1.8.10”.

**Example:** `docker pull influxdb:1.8.10`



# DOCKER IMAGE COMMANDS

There are specific sets of commands that can he used with Docker images.

---

Command: **`docker image ls [OPTIONS]`**

Description: List all images. Use this command to see container status, get image ID, and other information.

Options: **`-a`** (all)
**`-q`** (quiet)

Example: `docker image ls -a`

---

Command: **`docker image rm [OPTIONS] [IMAGE_ID]`**

Description: Remove one or more images.

Options: **`-f`** (force)

Example: `docker image rm -f 84db8c18bd54`

---

Command: **`docker image inspect [IMAGE_ID]`**

Description: Display detailed information on one or more images.

Example: `docker image inspect 84db8c18bd54`

---

Command: **`docker image save [OPTIONS] [IMAGE_ID]`**

Description: Save one or more images to a tar archive.

Options: **`-o`** (output file)

Example: `docker image save -o mysavedimage.tar 84db8c18bd54`

---

Command: **`docker image load [OPTIONS] [IMAGE_ID]`**

Description: Save one or more images to a tar archive.

Options: **`-i`** (input tar archive file)

Example: `docker image load -i mysavedimage.tar`

---

Command: **`docker image prune [OPTIONS]`**

Description: Remove all unused images.

Options: **`-a`** (remove all unused - not just the dangling ones)
**`-f`** (force - don't prompt for confirmation)

Example: `docker container prune -f`

---

Command: **`docker image --help`**

Description: List all commands that can be invoked against docker images. For a complete list of docker container commands, see the Docker documentation found here: https://docs.docker.com/engine/reference/commandline/image/.

# DOCKER RUN

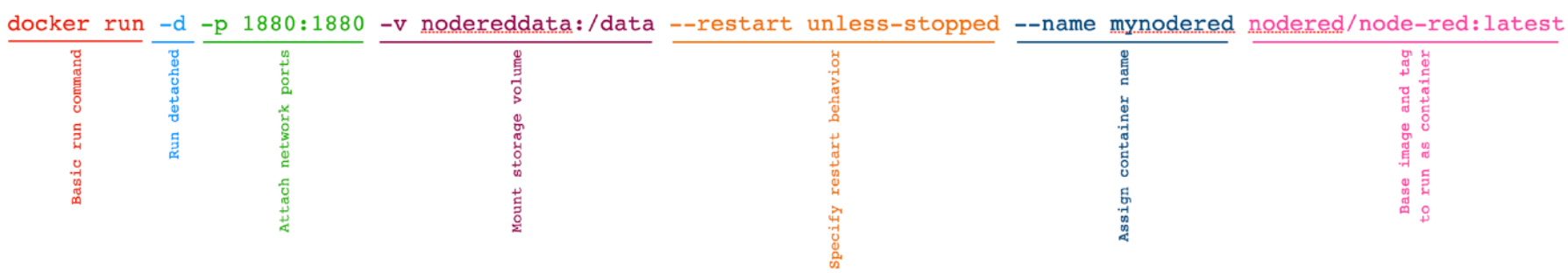
Syntax: **`docker run [OPTIONS] [IMAGE NAME]:[TAG NAME] [COMMAND`**(optional)**`]`**

Docker run command is a cli instruction to tell the Docker engine how you want to run a container. This includes information about container lifecycle, interactivity, networking, storage, name, etc… This guide will cover basics but more extensive instructions can be found here: https://docs.docker.com/engine/reference/run/

Example:

```
docker run -d -p 1880:1880 -v nodereddata:/data --restart unless-stopped --name mynodered nodered/node-red:latest
```

This diagram shows a breakdown of the above run example:




The order of the option flags does not matter as long as the image name, and command are called at the end. If you are invoking a “long-lived” container (to sustain a runtime or application) you will not need to add an explicit. If you are simply running a single action or function, you can call the command with the “run” command.
Common Optional Flags:

lifecycle + labeling

| | |
|---|---|
| **`-d`** | (run detached) |
| **`-it`** | (run in interactive terminal) |
| **`--rm`** | (clean up: remove container when stopped) |
| **`--init`** | (init process should be used as PID 1 in container) |
| **`--name [LABEL]`** | (adds a named label to container) |
| **`--restart on-failure:[MAX_RETRIES]`** | (Restart only if container exits with a non-zero status) |
| **`--restart unless-stopped`** | (container will restart unless explicitly stopped) |
| **`--restart always`** | (container will restart unless removed (forced) |

networking

| | |
|---|---|
| **-p [HOST_PORT]:[CONT_PORT]** | (attach network port to container) |
| **--network [DOCKER NETWORK]** | (attach custom docker network to container) |
| **--network host** | (attach entire host network to container) |
| **--network none** | (isolate the container from all networks) |
| **--network container:[ID]** | (attach to specified container's network stack) |

storage

| | |
|---|---|
| **-v [HOST_PATH]:[CONT_PATH]** | (attach storage to container. This can be either a specified docker volume or a standard linux direct path) |

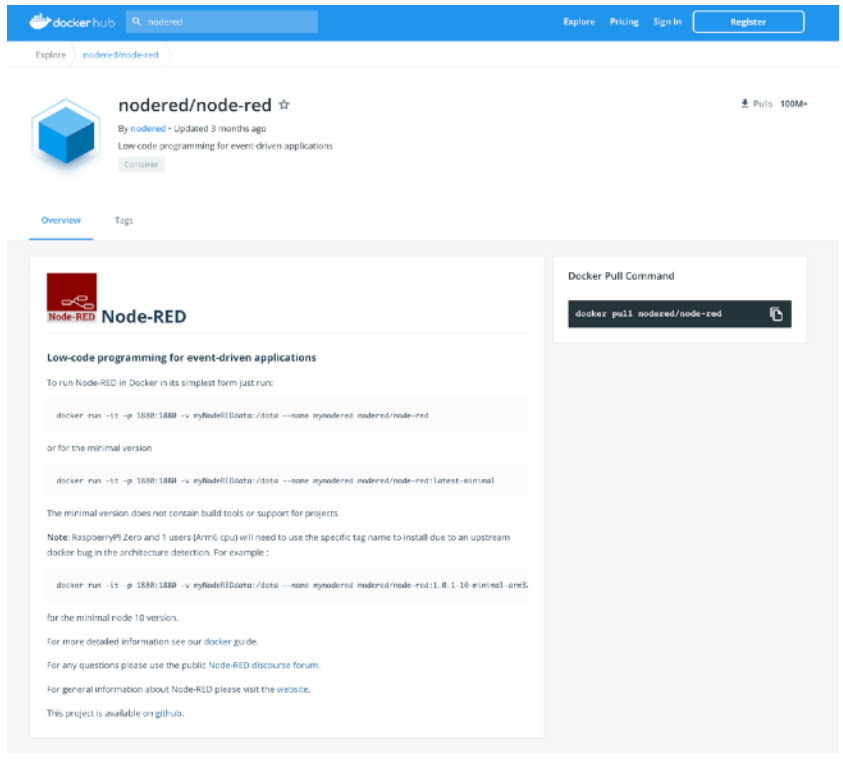
cpu management

| | |
|---|---|
| **-m [NUMBER[FORMAT]]** | (Memory limit (format: <number>[<unit>]). Number is a positive integer. Unit can be one of b, k, m, or g. Minimum is 4M, example "`-m 4m`") |
| **--cpus=[NUMBER(0.000)]** | (Number of CPUs. Number is a fractional number. 0.000 means no limit.) |
| **--cpuset-cpus=[NUMBER(S)/ RANGE]** | (CPUs in which to allow execution (0-3, 0,1)) |

system

| | |
|---|---|
| **--privileged** | (Give extended system privileges to this container i.e. access to hardware devices and drivers) |
| **--device=[HOST_DEV]:[CONT_DEV]** | (Allows you to run devices from inside the container without the --privileged flag) |

**Tip:** In addition to this guide, well constructed Docker images in the Docker Hub will have detailed instructions and examples on how to run the images as a container and any specific flags that should be used.




*This figure shows the instructions and example run command for the container*

# DOCKER CONTAINER COMMANDS

There are specific sets of commands that can he used with Docker containers.

---

Command: **`docker container ls [OPTIONS]`**

Description: List all containers running or stopped. Use this command to see container status, get container ID, and other information.

Options: **`-q`** (quiet)

Example: `docker container ls -a`

---

Command: **`docker container rm [OPTIONS] [CONTAINER_ID]`**

Description: Remove a container. Use the force command of the container is running or attached to a process.

Options: **`-f`** (force)

Example: `docker container rm -f 3756a204b1e4`

---

Command: **`docker container start|stop|restart [CONTAINER_ID]`**

Description: Manage the status of a container. This is used when you need to change the run status and not remove the container.

Example: `docker container restart 3756a204b1e4`

---

Command: **`docker container exec [OPTIONS] [CONTAINER_ID] [COMMAND]`**

Description: Execute a command in a running container. Tip: use the /bin/sh to enter a container's shell.

Options:

| | |
|---|---|
| **`-i`** | (interactive) |
| **`-u [USER]`** | (user) |
| **`--privileged`** | (grant extended privileges) |

Example: `docker container exec -i -u root 3756a204b1e4 /bin/sh`

---

Command: **`docker container update [OPTIONS] [CONTAINER_ID]`**

Description: List all containers running or stopped. Use this command to see container status, get container ID, and other information.

Options:

| | |
|---|---|
| **`-m [NUMBER[FORMAT]]`** | (Memory limit **see run command |
| **`--cpus=[NUMBER(0.000)]`** | (Number of CPUs. Number is a fractional number. 0.000 means no limit.) |
| **`--cpuset-cpus=[NUMBER]`** | (CPUs in which to allow execution (0-3, 0,1)) |
| **`--restart-string`** | (Update the restart behavior of the running container **see docker run command) |

Example: (update a container to limit memory access to 4mb)

```
docker container update -m 4m 3756a204b1e4
```

Command: **`docker container prune [OPTIONS]`**

Description: Remove all stopped containers.

Options: **`-f`** (force - don't prompt for confirmation)

Example: `docker container prune -f`

---

Command: **`docker container logs [OPTIONS] [CONTAINER_ID]`**

Description: Fetch the logs for a specific container.

Options: **`-f`** (follow)

**`-n`** (number of lines)

Example: `docker container logs 3756a204b1e4 -n 100`

---

Command: **`docker container export [OPTIONS] [CONTAINER_ID]`**

Description: Export a container file system to tar archive.

Options: **`-o`** (file to write)

Example: `docker container export -o my export.tar 3756a204b1e4`

---

Command: **`docker container cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-`**

Description: Copy files/folders between a container and the local filesystem.

Example: (copy from local file system into container)

```
docker cp ./some_file 3756a204b1e4:/work
```

(copy from container to local file system)

```
docker cp 3756a204b1e4:/var/logs/ /tmp/app_logs
```

---

Command: **`docker container top [CONTAINER_ID]`**

Description: Display the running processes of a container.

Example: `docker container top 3756a204b1e4`

---

Command: **`docker commit [OPTIONS] [CONTAINER_ID] REPOSITORY:TAG`**

Description: Create a new image from a container's changes.

Options: **`-a`** (author string)

**`-c`** (apply Dockerfile instruction to the created image)

**`-m`** (add message to committed image)

**`-p`** (pause container during commit process)

Example: (copy from local file system into container)

```
docker commit -p 3756a204b1e4 myimage:1.0.0
```

---

Command: **`docker image --help`**

Description: List all commands that can be invoked against docker images. For a complete list of docker container commands, see the Docker documentation found here: https://docs.docker.com/engine/reference/commandline/container/.

# DOCKER VOLUME COMMANDS

Docker volumes are simply dedicated areas where data and resources are stored outside of containers. This allows data to persist regardless of the container state. There are specific sets of commands that can he used with Docker volumes.

---

Command: **`docker volume create [OPTIONS] [VOLUME_NAME]`**

Description: Create a volume for persistent container storage.

Options: **`-d`** (driver ex. *local*)

Example: `docker volume create -d local nodereddata`

---

Command: **`docker volume ls`**

Description: Display a list of created volumes.

Example: `docker volume ls`

---

Command: **`docker volume inspect [VOLUME_NAME]`**

Description: Inspect the specific configuration of a volume.

Example: `docker volume inspect nodereddata`

---

Command: **`docker volume rm [VOLUME_NAME]`**

Description: Remove one or more volumes. You cannot remove a volume that is in use by a container unless forced.

Options: **`-f`** (force)

Example: `docker volume rm -f nodereddata`

---

Command: **`docker volume prune [OPTIONS]`**

Description: Remove all unused local volumes. Unused local volumes are those which are not referenced by any containers.

Options: **`-f`** (force)

Example: `docker volume rm -f nodereddata`

---

Command: **`docker volume --help`**

Description: List all commands that can be invoked against docker volumes. For a complete list of docker volumes commands, see the Docker documentation found here: https://docs.docker.com/engine/reference/commandline/volume/.

# DOCKER NETWORK COMMANDS

Command: **`docker network create [OPTIONS] NETWORK`**

Description: Creates a new network.

Options: **`--driver`** (driver to manage network)
**`--subnet`** (subnet in CIDR format)
**`--ip-range`** (allocate container IP from subrange)
**`--gateway`** (gateway of subnet master)

Example: `docker network create --ip-range=172.28.5.0/24 –gateway=172.28.5.254 br0`

Command: **`docker network ls`**

Description: List all of the networks that the Docker Engine knows about.

Example: `docker network ls`

Command: **`docker network connect|disconnect [OPTIONS] NETWORK CONTAINER`**

Description: Attach or detach an existing network to a container.

Options: **`--alias`** (add network-scoped alias for the container)
**`--ip`** (IPv4 address (e.g., 172.30.100.104)
**`--link`** (add link to another container)

Example: `docker network connect --ip 10.10.36.122 multi-host-network container2`

Command: **`docker network inspect [OPTIONS] NETWORK`**

Description: Display detailed information on one or more networks.

Example: `docker network inspect v –f JSON multi-host-network container2`

Command: **`docker network rm NETWORK`**

Description: Remove one or more networks.

Options: **`-f`** (force)
Example: `docker network rm  -f multi-host-network`

Command: **`docker network prune`**

Description: Remove all unused networks.

Options: **`-f`** (force)
Example: `docker network prune -f`

Command: **`docker network --help`**

Description: List all commands that can be invoked against docker networks. For a complete list of docker network commands, see the Docker documentation found here: https://docs.docker.com/engine/reference/commandline/network/.

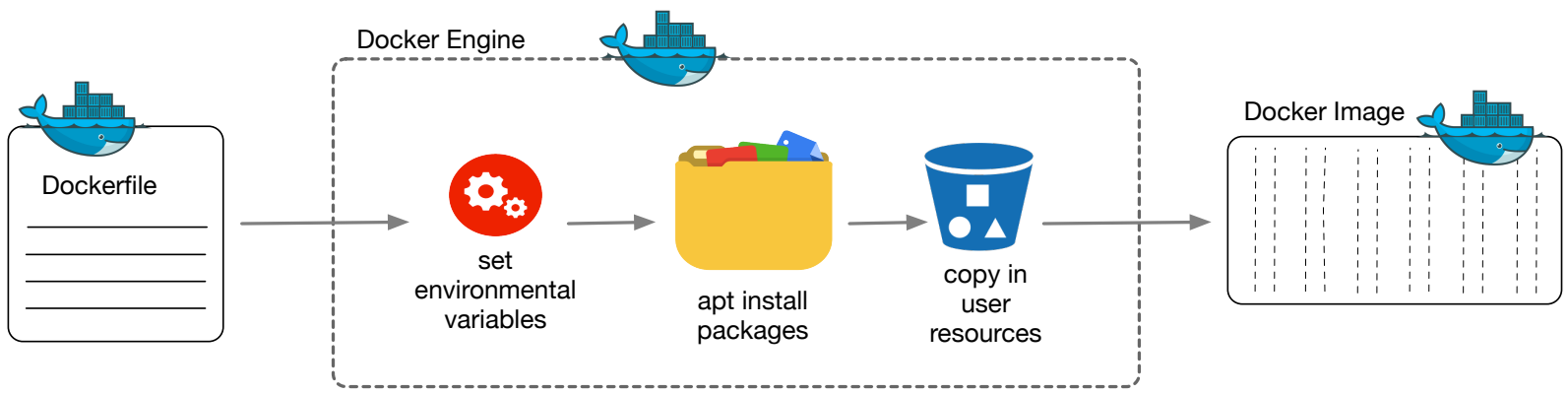
# CREATING DOCKER IMAGES

One of the most powerful features with Docker is the ability to automate the building of images with Dockerfiles and the docker build command.  The next part of this guide will explain the workflow, required resources, and instructions.

## THE DOCKERFILE

The Dockerfile is a very simple structured file with explicit instructions to build a Docker image.  The Dockerfile starts with a base image, then supplements the base image with resources, and outputs an image.  This file should always be named **`Dockerfile`** with no extension.

A docker build command and a Dockerfile create an image based on the instructions in the Dockerfile




For example: This is a Dockerfile example to build a web based ssh client to connect to an ssh server.  This begins with a base Ubuntu image, installs python, and the pip package manager, and uses pip to install the webssh package. The instructions open port 8080 when run as a container. Finally it runs the “wssh” executable when container is invoked.

```
# FROM denotes the base image to work from
FROM ubuntu:latest

# LABEL adds metadata to image
LABEL maintainer=“John Smith” version=“1.0”

# RUN the command to install dependencies
RUN apt-get update && apt-get install -y python3.4
python3-pip

# RUN pip install (we just installed pip)
RUN pip3 install webssh

# EXPOSE defaults the container to open 8080
EXPOSE 8080

# CMD runs this command on invocation of container
CMD ["wssh", "--address=0.0.0.0", “--port=8080"]
```

# DOCKERFILE INSTRUCTION REFERENCE

**FROM -** This will create the base layer for the subsequent instructions.

Syntax `FROM <image>:<tag>`

Example: `FROM debian:buster-slim`

---

**WORKDIR -** Set the working directory. RUN, CMD, ENTRYPOINT, COPY and ADD instructions will use this while execution. We can have multiple WORKDIR changes in a single Dockerfile.

Syntax: `WORKDIR <direct_path>`

Example: `WORKDIR /home/`

---

**COPY -** Copy files from source file system to container's file system.

Syntax: `COPY <source_path> <dest_path>`

Example: `COPY resources/test.py /home/test.py`

---

**ENV -** Sets environment variable.

Syntax: `ENV <key>=<value>`

Example: `ENV app_config=/app.properties`

---

**LABEL -** Adds meta-data to image.

Syntax: `LABEL <key>=<value>`

Example: `LABEL Author="John Smith"`

---

**USER -** Sets the username to use when running the image, all subsequent RUN, CMD and ENTRYPOINT instructions use this user when running.  Add a group is optional but it is not required.

Syntax: `SER <user>[:<group>]`

Example: `USER wago-user`

---

**ARG -** Define a variable to pass the value at build time; subsequent instructions will use this value.  Add a default value but it is not required.

Syntax: `ARG <name>[=<default-value>]`

Example: `ARG NODE_VER="16"`

---

**RUN -** Executes given command in a new layer on top of the current image and commits the result; this resulted image will be used as a base image by the next instructions. Also, note that RUN is executed at build time, for example; you can use RUN to setup environment before application execution, like updating software or installing required tools.

Syntax: `RUN <command> <param1> <param2>` (shell form)

`RUN ["executable", "param1", "param2"]` (exec form)

Example: `RUN apt-get update && apt-get upgrade`

**EXPOSE** - Expose port to allow incoming traffic to the container.

Syntax: `EXPOSE <port-number>`

Example: `EXPOSE 8080`

---

**VOLUME** - Mount file system to the container.

Syntax: `VOLUME <direct-path>`

Example: `VOLUME /etc/mosquitto`

---

**CMD** - Executes given command when the container is launched with 'docker run'. Provide only one CMD instruction; if you provide more, then the last one will be used to execute.

Syntax: `CMD ["executable", "param1", "param2"]` (exec form *preferred)

`CMD <command> <param1> <param2>` (shell form)

Example: `CMD java -jar inventory-1.0.jar`

---

**ENTRYPOINT** - Executes given command when container is launched with 'docker run'. When CMD and ENTRYPOINT, both are provided then ENTRYPOINT will be used to execute container and all the arguments of CMD will be passed to ENTRYPOINT.

Syntax: `ENTRYPOINT ["executable", "param1", "param2"]` (exec form *preferred)

`ENTRYPOINT <command> <param1> <param2>` (shell form)

Example: `ENTRYPOINT ["java", "-jar", "inventory-1.0.jar"]`

---

**HEALTHCHECK** - Docker will run this command to check if the container is working properly.

Syntax: `HEALTHCHECK [OPTIONS] CMD command` (checks container health by running it inside the container)

`HEALTHCHECK NONE` (disables health check)

Example: `HEALTHCHECK --interval=5m CMD curl -f http://localhost:8080/ || exit 1`

---

**STOPSIGNAL** - This instruction sets the system call signal that will be sent to the container to exit.

Syntax: `STOPSIGNAL <signal>`

Example: `STOPSIGNAL signal`

---

**#** - This is the character used in a Dockerfile for comments.

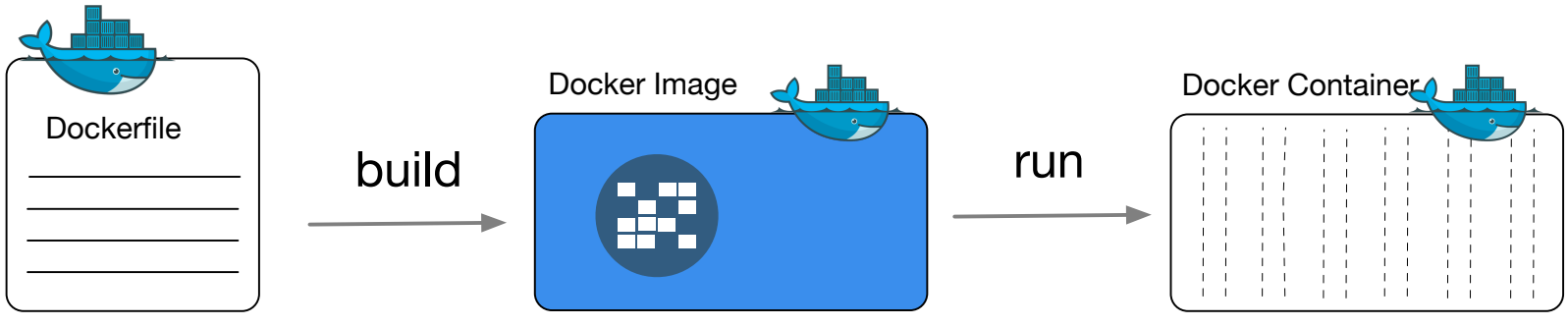
Syntax: `# <comment>`

Example: `# this is a comment`

---

For a more complete reference to Dockerfile instructions and formatting see the official Docker documentation found here: https://docs.docker.com/engine/reference/builder/.

# DOCKER BUILD




With a complete Dockerfile, the build process is triggered with a “docker build” command. This command starts the workflow to step through the instructions in the Dockerfile and report status back to the terminal.

---

Command: **`docker build [OPTIONS] [DOCKERFILE_PATH]`**

Description: Build an image from instructions in a Dockerfile.

| | | |
|---|---|---|
| Options: | **`-t [NAME]:[TAG]`** | (Set the name and tag of image) |
| | **`-q`** | (quiet build) |

Example: `docker build -t mynewimage:1.0 .`

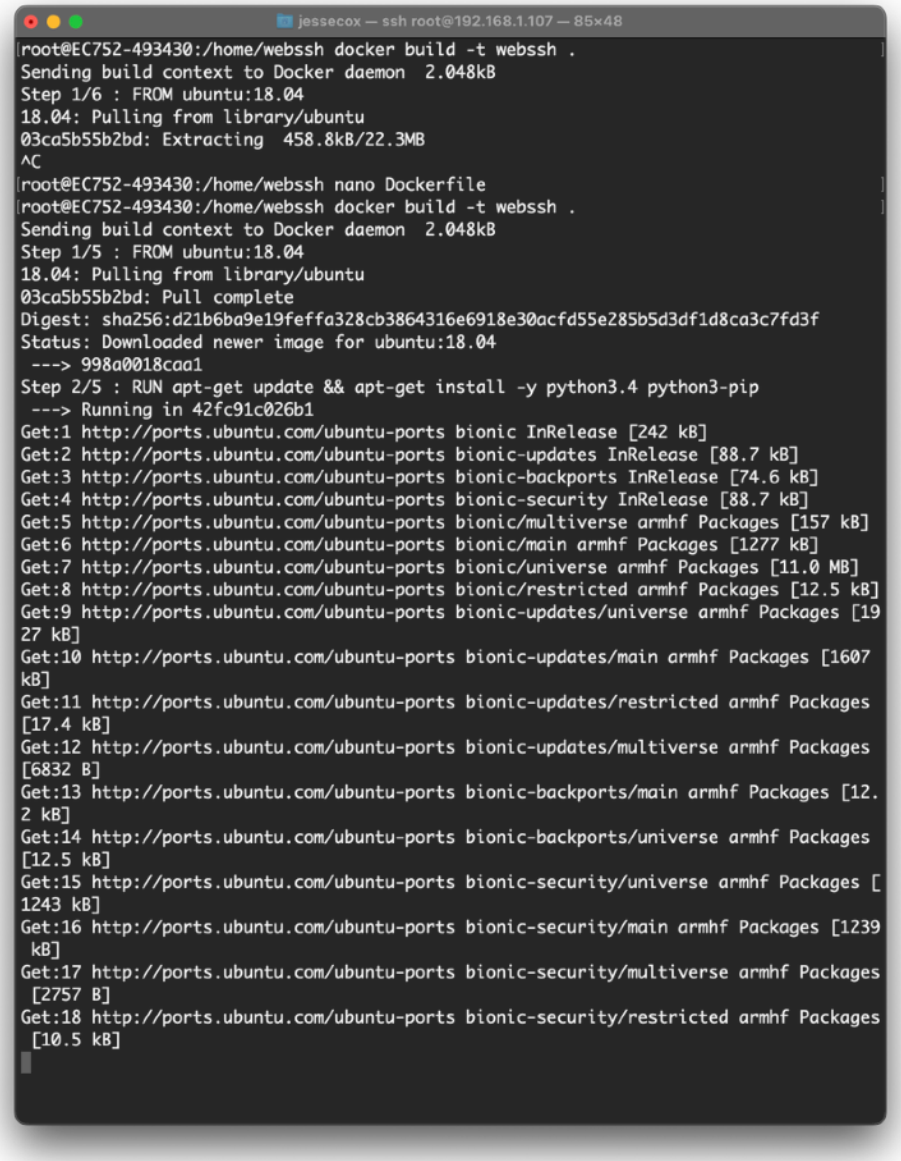
---

There are several options to pass the docker build command but it is possible (and common) to use this command in its most simple form.

Without adding the quiet flag, you can monitor the progress of the build and any errors that might be thrown.

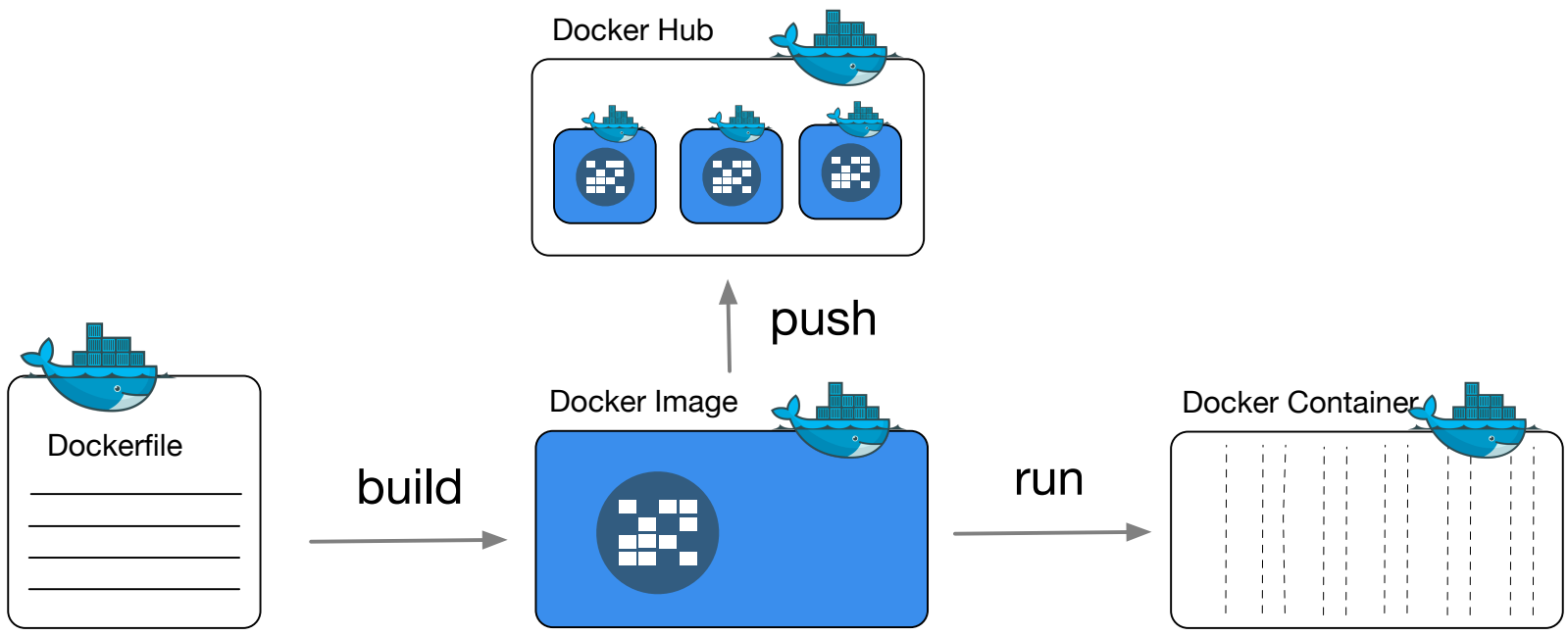
For a complete and detailed list of options for the docker build command see the Docker documentation link found here:

https://docs.docker.com/engine/reference/commandline/build/



## PUBLISHING IMAGES TO DOCKER HUB

Docker is so powerful because of its community contributions.  In addition to the convenience of getting pre-built images, you may want to make your images available either to the public or privately.  Publishing images is a simple process and free for public images.




To publish you will first be required to login to your Docker Hub account, then you can push your image to your repositories.

You can set up a free account here: https://hub.docker.com/

After your account is created, you can drive everything else from the command line.  Here are the two most used commands.

---

Command: **`docker login [OPTIONS]`**

Description: Log in to the Docker Hub to enable the push of your images.

Options: **`-u [USERNAME]`** (add your Docker Hub username)

**`-p [PASSWORD]`** (add your Docker Hub username)

Example: `docker login -u johndoe -p supersecretpassword`

---

Command: **`docker push [OPTIONS] [IMAGE]:[TAG]`**

Description: Commit your image to the Docker Hub after logging in.

Options: **`-q`** (quiet)

Example: `docker push myimage:1.0`

A complete description of the push process can be found in the documentation here: https://docs.docker.com/engine/reference/commandline/push/.

# TROUBLESHOOTING

| Symptom | Potential Cause | Suggested Action |
|---|---|---|
| INSTALLATION | | |
| Installation failed on “incompatible kernel version” | The target device firmware is not current enough to support the install package. | Update the firmware and try the install again |
| Installation failed with “device out of memory” error | The controller does not have enough free space to add the packages | Remove unused files or use an sd card to expand the file system. |
| PULLING IMAGES | | |
| docker pull fails with 404 invalid link error | Your controller is likely not connected to the internet. | See the nwtworking section of this guide for optional configurations |
| docker pull results n “authentication error” | Controller likely has some bad configurations. The most common cause is network and/or time configs | Verify the network connection and clock settings for accuracy for the local timezone |
| RUNNING CONTAINERS | | |
| Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running? | There is an issue with the daemon.json configuration file | Verify settings in the config file and restart docker daemon |
| docker run results in “linux_go” error | A container with incompatible architecture is being run. | Verify the image architecture is compatible with the host device (i.e. armv7 on PFC/EdgeCtrl/TP) |