Prerequisites

Docker requires a 64-bit OS and version 3.10 or higher of the Linux kernel.

To check your current kernel version, open a terminal and use uname -r to display your kernel version:

$ uname -r

3.10.0-229.el7.x86\_64

Install with yum

**You can install using the yum package manager.**

**Log into your machine as a user with sudo or root privileges.**

**Make sure your existing packages are up-to-date.**

$ sudo yum update

**Add the yum repo.**

$ sudo tee /etc/yum.repos.d/docker.repo <<-'EOF'

[dockerrepo]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/main/centos/7/

enabled=1

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg

EOF

**Install the Docker package.**

$ sudo yum install docker-engine

**Enable the service.**

$ sudo systemctl enable docker.service

**Start the Docker daemon..**

$ sudo systemctl start docker

**Verify docker is installed correctly by running a test image in a container.**

$ sudo docker run --rm hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

c04b14da8d14: Pull complete

Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9

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/

Start the docker daemon at boot

**Configure the Docker daemon to start automatically when the host starts:**

$ sudo systemctl enable docker

# Windows

You have two options for installing Docker on Windows:

* [Docker for Windows](https://docs.docker.com/engine/installation/windows/#docker-for-windows)
* [Docker Toolbox](https://docs.docker.com/engine/installation/windows/#docker-toolbox)

## Docker for Windows with InstallDocker.msi

Docker for Windows is our newest offering for PCs. It runs as a native Windows application and uses Hyper-V to virtualize the Docker Engine environment and Linux kernel-specific features for the Docker daemon.

Go to [Getting Started with Docker for Windows](https://docs.docker.com/docker-for-windows/) for download and install instructions, and to learn all about Docker for Windows.

**Requirements**

* **64bit Windows 10 Pro, Enterprise and Education (1511 November update, Build 10586 or later). In the future we will support more versions of Windows 10.**
* The Hyper-V package must be enabled. The Docker for Windows installer will enable it for you, if needed. (This requires a reboot).

## With Docker Toolbox

If you have an earlier Windows system that doesn’t meet the Docker for Windows requirements, [get Docker Toolbox](https://www.docker.com/products/docker-toolbox).

See [Docker Toolbox Overview](https://docs.docker.com/toolbox/overview/) for help on installing Docker with Toolbox.

The Docker Toolbox setup does not run Docker natively on Windows. Instead, it uses docker-machine to create and attach to a virtual machine (VM). This machine is a Linux VM that hosts Docker for you on your Windows system.

**Requirements**

To run Docker, your machine must have a 64-bit operating system running Windows 7 or higher. Additionally, you must make sure that virtualization is enabled on your machine. For details, see the [Toolbox install instructions for Windows](https://docs.docker.com/toolbox/toolbox_install_windows/).

# Install Docker for Windows with Docker ToolBox

Windows users use Docker Toolbox to install Docker software. Docker Toolbox includes the following Docker tools:

* Docker CLI client for running Docker Engine to create images and containers
* Docker Machine so you can run Docker Engine commands from Windows terminals
* Docker Compose for running the docker-compose command
* Kitematic, the Docker GUI
* the Docker QuickStart shell preconfigured for a Docker command-line environment
* Oracle VM VirtualBox

Because the Docker Engine daemon uses Linux-specific kernel features, you can’t run Docker Engine natively in Windows. Instead, you must use the Docker Machine command, docker-machine, to create and attach to a small Linux VM on your machine. This VM hosts Docker Engine for you on your Windows system.

## Step 1: Check your version

To run Docker, your machine must have a 64-bit operating system running Windows 7 or higher. Additionally, you must make sure that virtualization is enabled on your machine. To verify your machine meets these requirements, do the following:

1. Right click the windows message and choose **System**.

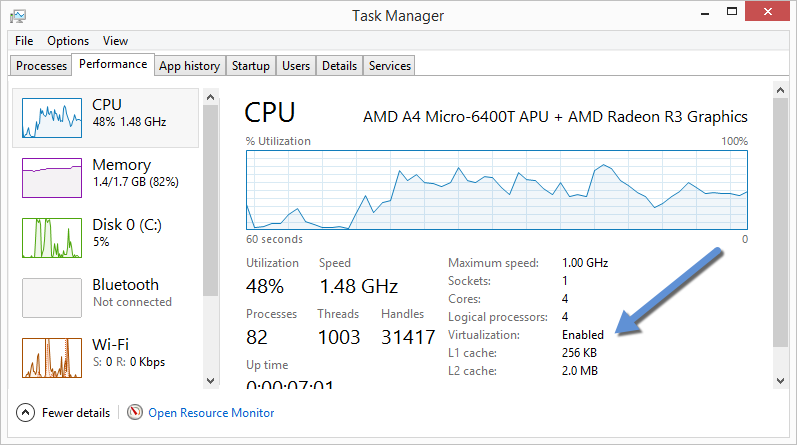
If you aren’t using a supported version, you could consider upgrading your operating system.

If you have a newer system, specifically 64bit Windows 10 Pro, with Enterprise and Education (1511 November update, Build 10586 or later), consider using [Docker for Windows](https://docs.docker.com/docker-for-windows) instead. It runs natively on the Windows, so there is no need for a pre-configured Docker QuickStart shell. It also uses Hyper-V for virtualization, so the instructions below for checking virtualization will be out of date for newer Windows systems. Full install prerequisites are provided in the Docker for Windows topic in [What to know before you install](https://docs.docker.com/docker-for-windows/#what-to-know-before-you-install).

1. Make sure your Windows system supports Hardware Virtualization Technology and that virtualization is enabled.

**For Windows 8 or 8.1**

Choose **Start > Task Manager** and navigate to the **Performance** tab. Under **CPU** you should see the following:



If virtualization is not enabled on your system, follow the manufacturer’s instructions for enabling it.

**For Windows 7**

Run the [Microsoft® Hardware-Assisted Virtualization Detection Tool](http://www.microsoft.com/en-us/download/details.aspx?id=592) and follow the on-screen instructions. 

1. Verify your Windows OS is 64-bit (x64)

How you do this verification depends on your Windows version. For details, see the Windows article[How to determine whether a computer is running a 32-bit version or 64-bit version of the Windows operating system](https://support.microsoft.com/en-us/kb/827218).

## Step 2: Install Docker Toolbox

**Download from**

**https://github.com/docker/toolbox/releases/tag/v1.12.2**

In this section, you install the Docker Toolbox software and several “helper” applications. The installation adds the following software to your machine:

* Docker Client for Windows
* Docker Toolbox management tool and ISO
* Oracle VM VirtualBox
* Git MSYS-git UNIX tools

If you have a previous version of VirtualBox installed, do not reinstall it with the Docker Toolbox installer. When prompted, uncheck it.

If you have Virtual Box running, you must shut it down before running the installer.

1. Go to the [Docker Toolbox](https://www.docker.com/toolbox) page.
2. Click the installer link to download.
3. Install Docker Toolbox by double-clicking the installer.

The installer launches the “Setup - Docker Toolbox” dialog.

If Windows security dialog prompts you to allow the program to make a change, choose **Yes**. The system displays the **Setup - Docker Toolbox for Windows** wizard.

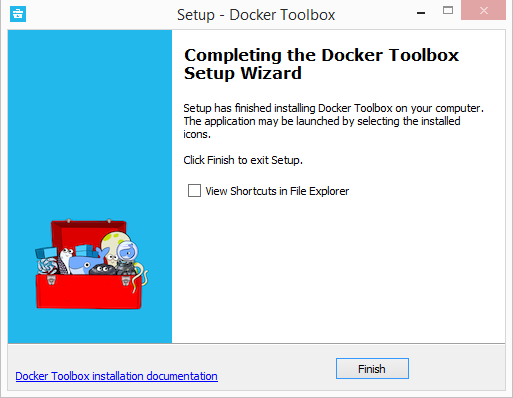


1. Press **Next** to accept all the defaults and then **Install**.

Accept all the installer defaults. The installer takes a few minutes to install all the components:

1. When notified by Windows Security the installer will make changes, make sure you allow the installer to make the necessary changes.

When it completes, the installer reports it was successful:

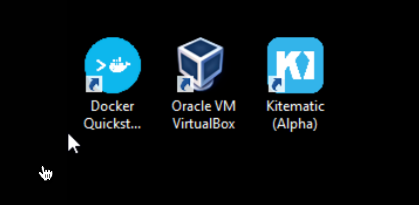


1. Uncheck “View Shortcuts in File Explorer” and press **Finish**.

## Step 3: Verify your installation

The installer places Docker Toolbox and VirtualBox in your **Applications** folder. In this step, you start Docker Toolbox and run a simple Docker command.

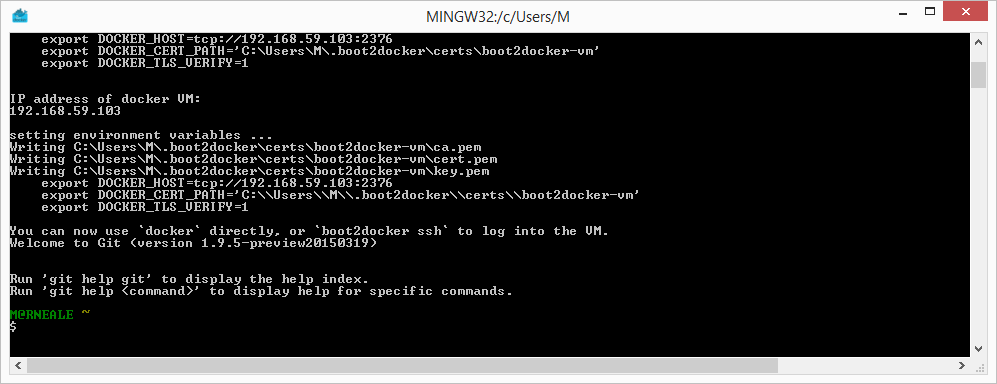
1. On your Desktop, find the Docker Toolbox icon.



1. Click the icon to launch a Docker Toolbox terminal.

If the system displays a **User Account Control** prompt to allow VirtualBox to make changes to your computer. Choose **Yes**.

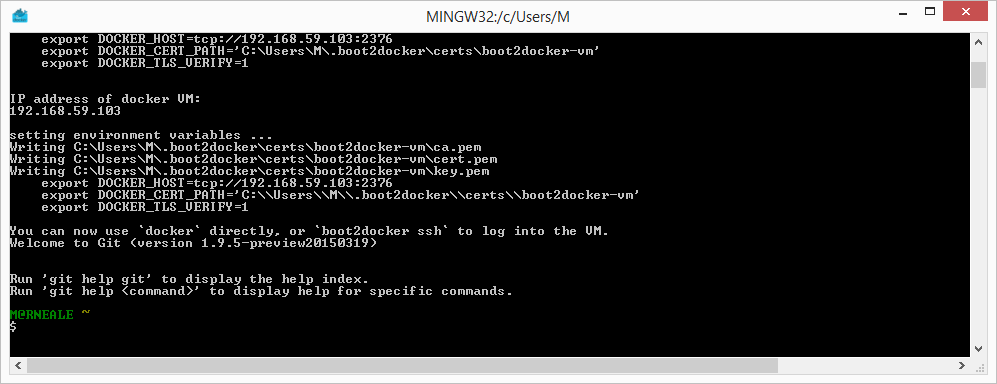
The terminal does several things to set up Docker Toolbox for you. When it is done, the terminal displays the $ prompt.



The terminal runs a special bash environment instead of the standard Windows command prompt. Thebash environment is required by Docker.

1. Make the terminal active by click your mouse next to the $ prompt.

If you aren’t familiar with a terminal window, here are some quick tips.



The prompt is traditionally a $ dollar sign. You type commands into the command line which is the area after the prompt. Your cursor is indicated by a highlighted area or a | that appears in the command line. After typing a command, always press RETURN.

1. Type the docker run hello-world command and press RETURN.

The command does some work for you, if everything runs well, the command’s output looks like this:

$ docker run hello-world

Unable to find image 'hello-world:latest' locally

Pulling repository hello-world

91c95931e552: Download complete

a8219747be10: Download complete

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 Engine CLI client contacted the Docker Engine daemon.

2. The Docker Engine daemon pulled the "hello-world" image **from** the Docker Hub.

(Assuming it was not already locally available.)

3. The Docker Engine daemon created a **new** container **from** that image which runs the

executable that produces the output you are currently reading.

4. The Docker Engine daemon streamed that output to the Docker Engine CLI client, which sent it

to your terminal.

To **try** something more ambitious, you can run an Ubuntu container **with**:

$ docker run -it ubuntu bash

For more examples and ideas, visit:

https:*//docs.docker.com/userguide/*

## Looking for troubleshooting help?

Typically, the above steps work out-of-the-box, but some scenarios can cause problems. If your docker run hello-world didn’t work and resulted in errors, check out [Troubleshooting](https://docs.docker.com/toolbox/faqs/troubleshoot/) for quick fixes to common problems.

A Windows specific problem you might encounter has to do with the NDIS6 host network filter driver, which is known to cause issues on some Windows versions. For Windows Vista systems and newer, VirtualBox installs NDIS6 driver by default. Issues can range from system slowdowns to networking problems for the virtual machine (VM). If you notice problems, **re-run the Docker Toolbox installer**, and select the option to ***install VirtualBox with the NDIS5 driver***.

## How to uninstall Toolbox

Removing Toolbox involves removing all the Docker components it includes.

A full uninstall also includes removing the local and remote machines you created with Docker Machine. In some cases, you might want to keep machines created with Docker Machine.

For example, if you plan to re-install Docker Machine as a part of Docker for Windows you can continue to manage those machines through Docker. Or, if you have remote machines on a cloud provider and you plan to manage them using the provider, you wouldn’t want to remove them. So the step to remove machines is described here as optional.

To uninstall Toolbox on Windows, do the following:

1. List your machines.
2. $ docker-machine ls
3. NAME ACTIVE DRIVER STATE URL SWARM
4. dev \* virtualbox Running tcp:*//192.168.99.100:2376*
5. my-docker-machine virtualbox Stopped
6. **default** virtualbox Stopped
7. Optionally, remove each machine. For example:
8. $ docker-machine rm **my**-docker-machine
9. Successfully removed **my**-docker-machine
10. Uninstall Docker Toolbox using Window’s standard process for uninstalling programs through the control panel.

**Note:** This process does not remove the docker-install.exe file. You must delete that file yourself.

## Learning more

* If you are new to Docker, try out the [Getting Started](https://docs.docker.com/engine/getstarted/) tutorial for a hands-on tour, including using Docker commands, running containers, building images, and working with Docker Hub.
* You can find more extensive examples in [Learn by example](https://docs.docker.com/engine/tutorials/) and in the [Docker Engine User Guide](https://docs.docker.com/engine/userguide/).
* If you are interested in using the Kitematic GUI, see the [Kitematic user guide](https://docs.docker.com/kitematic/userguide/).

**Note**: The Boot2Docker command line was deprecated several releases > back in favor of Docker Machine, and now Docker for Windows.

If you are using Docker Toolbox, you can use the **Docker Quickstart** Terminal to run Docker commands in a pre-configured environment instead of opening a command line terminal.

If you are **using Docker for Mac, Docker for Windows, or Docker on Linux**, you will have Docker running in the background, and your standard command line terminal is already set up to run Docker commands.

# Use the Docker command line

To list available commands, either run docker with no parameters or execute docker help:

$ docker

Usage: docker [OPTIONS] COMMAND [arg...]

docker [ *--help | -v | --version ]*

A self-sufficient runtime for containers.

Options:

*--config=~/.docker Location of client config files*

-D, *--debug Enable debug mode*

-H, *--host=[] Daemon socket(s) to connect to*

-h, *--help Print usage*

-l, *--log-level=info Set the logging level*

*--tls Use TLS; implied by --tlsverify*

*--tlscacert=~/.docker/ca.pem Trust certs signed only by this CA*

*--tlscert=~/.docker/cert.pem Path to TLS certificate file*

*--tlskey=~/.docker/key.pem Path to TLS key file*

*--tlsverify Use TLS and verify the remote*

-v, *--version Print version information and quit*

Commands:

attach Attach to a running container

# […]

Depending on your Docker system configuration, you may be required to preface each dockercommand with sudo. To avoid having to use sudo with the docker command, your system administrator can create a Unix group called docker and add users to it.

For more information about installing Docker or sudo configuration, refer to the [installation](https://docs.docker.com/engine/installation/) instructions for your operating system.

## Environment variables

For easy reference, the following list of environment variables are supported by the docker command line:

* DOCKER\_API\_VERSION The API version to use (e.g. 1.19)
* DOCKER\_CONFIG The location of your client configuration files.
* DOCKER\_CERT\_PATH The location of your authentication keys.
* DOCKER\_DRIVER The graph driver to use.
* DOCKER\_HOST Daemon socket to connect to.
* DOCKER\_NOWARN\_KERNEL\_VERSION Prevent warnings that your Linux kernel is unsuitable for Docker.
* DOCKER\_RAMDISK If set this will disable ‘pivot\_root’.
* DOCKER\_TLS\_VERIFY When set Docker uses TLS and verifies the remote.
* DOCKER\_CONTENT\_TRUST When set Docker uses notary to sign and verify images. Equates to --disable-content-trust=false for build, create, pull, push, run.
* DOCKER\_CONTENT\_TRUST\_SERVER The URL of the Notary server to use. This defaults to the same URL as the registry.
* DOCKER\_TMPDIR Location for temporary Docker files.

Because Docker is developed using ‘Go’, you can also use any environment variables used by the ‘Go’ runtime. In particular, you may find these useful:

* HTTP\_PROXY
* HTTPS\_PROXY
* NO\_PROXY

These Go environment variables are case-insensitive. See the [Go specification](http://golang.org/pkg/net/http/) for details on these variables.

## Configuration files

By default, the Docker command line stores its configuration files in a directory called .docker within your $HOME directory. However, you can specify a different location via the DOCKER\_CONFIG environment variable or the --config command line option. If both are specified, then the --config option overrides the DOCKER\_CONFIG environment variable. For example:

docker --config ~/testconfigs/ ps

Instructs Docker to use the configuration files in your ~/testconfigs/ directory when running the pscommand.

Docker manages most of the files in the configuration directory and you should not modify them. However, you can modify the config.json file to control certain aspects of how the docker command behaves.

Currently, you can modify the docker command behavior using environment variables or command-line options. You can also use options within config.json to modify some of the same behavior. When using these mechanisms, you must keep in mind the order of precedence among them. Command line options override environment variables and environment variables override properties you specify in aconfig.json file.

The config.json file stores a JSON encoding of several properties:

The property HttpHeaders specifies a set of headers to include in all messages sent from the Docker client to the daemon. Docker does not try to interpret or understand these header; it simply puts them into the messages. Docker does not allow these headers to change any headers it sets for itself.

The property psFormat specifies the default format for docker ps output. When the --format flag is not provided with the docker ps command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see the[**Formatting** section in the docker ps documentation](https://docs.docker.com/engine/reference/commandline/ps/)

Once attached to a container, users detach from it and leave it running using the using CTRL-p CTRL-qkey sequence. This detach key sequence is customizable using the detachKeys property. Specify a<sequence> value for the property. The format of the <sequence> is a comma-separated list of either a letter [a-Z], or the ctrl- combined with any of the following:

* a-z (a single lowercase alpha character )
* @ (at sign)
* [ (left bracket)
* \\ (two backward slashes)
* \_ (underscore)
* ^ (caret)

Your customization applies to all containers started in with your Docker client. Users can override your custom or the default key sequence on a per-container basis. To do this, the user specifies the --detach-keys flag with the docker attach, docker exec, docker run or docker start command.

The property imagesFormat specifies the default format for docker images output. When the --formatflag is not provided with the docker images command, Docker’s client uses this property. If this property is not set, the client falls back to the default table format. For a list of supported formatting directives, see the [**Formatting** section in the docker images documentation](https://docs.docker.com/engine/reference/commandline/images/)

Following is a sample config.json file:

{

"HttpHeaders": {

"MyHeader": "MyValue"

},

"psFormat": "table {{.ID}}\\t{{.Image}}\\t{{.Command}}\\t{{.Labels}}",

"imagesFormat": "table {{.ID}}\\t{{.Repository}}\\t{{.Tag}}\\t{{.CreatedAt}}",

"detachKeys": "ctrl-e,e"

}

### Notary

If using your own notary server and a self-signed certificate or an internal Certificate Authority, you need to place the certificate at tls/<registry\_url>/ca.crt in your docker config directory.

Alternatively you can trust the certificate globally by adding it to your system’s list of root Certificate Authorities.

## Help

To list the help on any command just execute the command, followed by the --help option.

$ docker **run** --help

Usage: docker **run** [OPTIONS] IMAGE [COMMAND] [ARG...]

**Run** a command **in** a new container

-a, --attach=[] Attach to STDIN, STDOUT or STDERR

--cpu-shares=0 CPU shares (relative weight)

...

## Option types

Single character command line options can be combined, so rather than typing docker run -i -t --name test busybox sh, you can write docker run -it --name test busybox sh.

### Boolean

Boolean options take the form -d=false. The value you see in the help text is the default value which is set if you do **not** specify that flag. If you specify a Boolean flag without a value, this will set the flag totrue, irrespective of the default value.

For example, running docker run -d will set the value to true, so your container **will** run in “detached” mode, in the background.

Options which default to true (e.g., docker build --rm=true) can only be set to the non-default value by explicitly setting them to false:

$ docker build --rm=**false** .

### Multi

You can specify options like -a=[] multiple times in a single command line, for example in these commands:

$ docker run -a stdin -a stdout -i -t ubuntu /bin/bash

$ docker run -a stdin -a stdout -a stderr ubuntu /bin/ls

Sometimes, multiple options can call for a more complex value string as for -v:

$ docker **run** -v /host:/container example/mysql

**Note:** Do not use the -t and -a stderr options together due to limitations in the ptyimplementation. All stderr in pty mode simply goes to stdout.

### Strings and Integers

Options like --name="" expect a string, and they can only be specified once. Options like -c=0 expect an integer, and they can only be specified once.

**How is Docker different from a normal virtual machine?**

Through this post we are going to draw some lines of differences between VMs and LXCs. Lets first define them.

**VM:**  
A virtual machine emulates a physical computing environment, but requests for CPU, memory, hard disk, network and other hardware resources are managed by a virtualization layer which translates these requests to the underlying physical hardware.

In this context the VM is called as the Guest while the environment it runs on is called the Host

**LXCs:**  
Linux Containers (LXC) are operating system-level capabilities that make it possible to run multiple isolated Linux containers, on one control host (the LXC host). Linux Containers serve as a lightweight alternative to VMs as they donâ€™t require the hypervisors viz. Virtualbox, KVM, Xen etc.

Docker originally used LinuX Containers (LXC), but later switched to runC (formerly known as libcontainer), which runs in the same operating system as its host. This allows it to share a lot of the host operating system resources. Also, it uses a layered filesystem (AuFS) and manages networking.

Containers include the application and all of its dependencies --but share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

**Case 1**  
So, let's say you have a 1GB container image; if you wanted to use a Full VM, you would need to have 1GB times x number of VMs you want. With docker and AuFS you can share the bulk of the 1GB between all the containers and if you have 1000 containers you still might only have a little over 1GB of space for the containers OS (assuming they are all running the same OS image).

**Case 2**  
A full virtualized system gets its own set of resources allocated to it, and does minimal sharing. You get more isolation, but it is much heavier (requires more resources). With docker you get less isolation, but the containers are lightweight (require fewer resources). So you could easily run thousands of containers on a host, and it won't even blink.

**Case 3**  
A full virtualized system usually takes minutes to start, whereas docker/LXC/runC containers take seconds, and often even less than a second.

**Case 4**  
If you want full isolation with guaranteed resources, a full VM is the way to go. If you just want to isolate processes from each other and want to run a ton of them on a reasonably sized host, then docker/LXC/runC seems to be the way to go.

**Case 5**  
Deploying a consistent production environment is easier said than done. Even if you use tools like chef and puppet, there are always OS updates and other things that change between hosts and environments.

Docker gives you the ability to snapshot the OS into a shared image, and makes it easy to deploy on other docker hosts. Locally, dev, qa, prod, etc.: all the same image. Sure you can do this with other tools, but not nearly as easily or fast.

**Case 6**  
LIGHTWEIGHT - Containers running on a single machine share the same operating system kernel; they start instantly and use less RAM. Images are constructed from layered filesystems and share common files, making disk usage and image downloads much more efficient.

Whereas, Virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system -- all of which can amount to tens of GBs.

**What are the differences between a VM image and a Docker image?**

These are some differences between a docker and a VM image which I could list out:

**1. Snapshot process is faster in Docker than VMs**

We generally start with a base image, and then make our changes, and commit those changes using docker, and it creates an image. This image contains only the differences from the base. When we want to run our image, we also need the base, and it layers our image on top of the base using a layered file system. File system merges the different layers together and we get what we want, and we just need to run it. Since docker typically builds on top of ready-made images from aÂ registry, we rarely have to "snapshot" the whole OS ourself. This ability of Dockers to snapshot the OS into a common image also makes it easy to deploy on other docker hosts.

**2. Startup time is less for Docker than VMs**

A virtual machine usually takes minutes to start, but containers takes seconds, and sometime even less than a second.

**4. Docker images have more portability**

Docker images are composed of layers. When we pull or transfer an image, only the layers we haven’t yet in cache are retrieved. That means that if we use multiple images based on the same base Operating System, the base layer is created or retrieved only once. VM images doesn't have this flexibility.

**5. Docker provides versioning of images**

We can use theÂ docker commitÂ command. We can specify two flags:Â -mÂ andÂ -a.Â TheÂ -mÂ flag allows us to specify a commit message, much like we would with a commit on a version control system:

$ sudo docker commit -m "Added json gem" -a "Kate Smith" 0b2616b0e5a8 ouruser/sinatra:v2 4f177bd27a9ff0f6dc2a830403925b5360bfe0b93d476f7fc3231110e7f71b1c

**6. Docker images do not have states**

In Docker terminology, a read-onlyÂ LayerÂ is called an image. An image never changes. Since Docker uses aÂ Union File System, the processes think the whole file system is mounted read-write. But all the changes go to the top-most writeable layer, and underneath, the original file in the read-only image is unchanged. Since images don't change, images do not have state.

**7. VMs are hardware-centric and docker containers are application-centric**

Let's say we have a container image that is 1GB in size. If we wanted to use a Full VM, we would need to have 1GB times x number of VMs you want. In docker container we can share the bulk of the 1GB and if you have 1000 containers we still might only have a little over 1GB of space for the containers OS, assuming they are all running the same OS image.

**8. Supported image formats**

Docker images:

* bare. The image does not have a container or metadata envelope.
* ovf. The OVF container format.
* aki. An Amazon kernel image.
* ari. An Amazon ramdisk image.
* ami. An Amazon machine image.

VM images:

* raw. An unstructured disk image format; if you have a file without an extension it is possibly a raw format
* vhd. The VHD disk format, a common disk format used by virtual machine monitors from VMware, Xen, Microsoft, VirtualBox, and others
* vmdk. Common disk format supported by many common virtual machine monitors
* vdi. Supported by VirtualBox virtual machine monitor and the QEMU emulator
* iso. An archive format for the data contents of an optical disc, such as CD-ROM.
* qcow2. Supported by the QEMU emulator that can expand dynamically and supports Copy on Write
* aki. An Amazon kernel image.
* ari. An Amazon ramdisk image.
* ami. An Amazon machine image

## The Docker command Line Complete Reference

This section contains reference information on using Docker’s command line client. Each command has a reference page along with samples. If you are unfamiliar with the command line, you should start by reading about how to [Use the Docker command line](https://docs.docker.com/engine/reference/commandline/cli/).

You start the Docker daemon with the command line. How you start the daemon affects your Docker containers. For that reason you should also make sure to read the [dockerd](https://docs.docker.com/engine/reference/commandline/dockerd/) reference page.

### Docker management commands

| Command | Description |
| --- | --- |
| [dockerd](https://docs.docker.com/engine/reference/commandline/dockerd/) | Launch the Docker daemon |
| [info](https://docs.docker.com/engine/reference/commandline/info/) | Display system-wide information |
| [inspect](https://docs.docker.com/engine/reference/commandline/inspect/) | Return low-level information on a container or image |
| [version](https://docs.docker.com/engine/reference/commandline/version/) | Show the Docker version information |

### Image commands

| Command | Description |
| --- | --- |
| [build](https://docs.docker.com/engine/reference/commandline/build/) | Build an image from a Dockerfile |
| [commit](https://docs.docker.com/engine/reference/commandline/commit/) | Create a new image from a container’s changes |
| [history](https://docs.docker.com/engine/reference/commandline/history/) | Show the history of an image |
| [images](https://docs.docker.com/engine/reference/commandline/images/) | List images |
| [import](https://docs.docker.com/engine/reference/commandline/import/) | Import the contents from a tarball to create a filesystem image |
| [load](https://docs.docker.com/engine/reference/commandline/load/) | Load an image from a tar archive or STDIN |
| [rmi](https://docs.docker.com/engine/reference/commandline/rmi/) | Remove one or more images |
| [save](https://docs.docker.com/engine/reference/commandline/save/) | Save images to a tar archive |
| [tag](https://docs.docker.com/engine/reference/commandline/tag/) | Tag an image into a repository |

### Container commands

| Command | Description |
| --- | --- |
| [attach](https://docs.docker.com/engine/reference/commandline/attach/) | Attach to a running container |
| [cp](https://docs.docker.com/engine/reference/commandline/cp/) | Copy files/folders from a container to a HOSTDIR or to STDOUT |
| [create](https://docs.docker.com/engine/reference/commandline/create/) | Create a new container |
| [diff](https://docs.docker.com/engine/reference/commandline/diff/) | Inspect changes on a container’s filesystem |
| [events](https://docs.docker.com/engine/reference/commandline/events/) | Get real time events from the server |
| [exec](https://docs.docker.com/engine/reference/commandline/exec/) | Run a command in a running container |
| [export](https://docs.docker.com/engine/reference/commandline/export/) | Export a container’s filesystem as a tar archive |
| [kill](https://docs.docker.com/engine/reference/commandline/kill/) | Kill a running container |
| [logs](https://docs.docker.com/engine/reference/commandline/logs/) | Fetch the logs of a container |
| [pause](https://docs.docker.com/engine/reference/commandline/pause/) | Pause all processes within a container |
| [port](https://docs.docker.com/engine/reference/commandline/port/) | List port mappings or a specific mapping for the container |
| [ps](https://docs.docker.com/engine/reference/commandline/ps/) | List containers |
| [rename](https://docs.docker.com/engine/reference/commandline/rename/) | Rename a container |
| [restart](https://docs.docker.com/engine/reference/commandline/restart/) | Restart a running container |
| [rm](https://docs.docker.com/engine/reference/commandline/rm/) | Remove one or more containers |
| [run](https://docs.docker.com/engine/reference/commandline/run/) | Run a command in a new container |
| [start](https://docs.docker.com/engine/reference/commandline/start/) | Start one or more stopped containers |
| [stats](https://docs.docker.com/engine/reference/commandline/stats/) | Display a live stream of container(s) resource usage statistics |
| [stop](https://docs.docker.com/engine/reference/commandline/stop/) | Stop a running container |
| [top](https://docs.docker.com/engine/reference/commandline/top/) | Display the running processes of a container |
| [unpause](https://docs.docker.com/engine/reference/commandline/unpause/) | Unpause all processes within a container |
| [update](https://docs.docker.com/engine/reference/commandline/update/) | Update configuration of one or more containers |
| [wait](https://docs.docker.com/engine/reference/commandline/wait/) | Block until a container stops, then print its exit code |

### Hub and registry commands

| Command | Description |
| --- | --- |
| [login](https://docs.docker.com/engine/reference/commandline/login/) | Register or log in to a Docker registry |
| [logout](https://docs.docker.com/engine/reference/commandline/logout/) | Log out from a Docker registry |
| [pull](https://docs.docker.com/engine/reference/commandline/pull/) | Pull an image or a repository from a Docker registry |
| [push](https://docs.docker.com/engine/reference/commandline/push/) | Push an image or a repository to a Docker registry |
| [search](https://docs.docker.com/engine/reference/commandline/search/) | Search the Docker Hub for images |

### Network and connectivity commands

| Command | Description |
| --- | --- |
| [network connect](https://docs.docker.com/engine/reference/commandline/network_connect/) | Connect a container to a network |
| [network create](https://docs.docker.com/engine/reference/commandline/network_create/) | Create a new network |
| [network disconnect](https://docs.docker.com/engine/reference/commandline/network_disconnect/) | Disconnect a container from a network |
| [network inspect](https://docs.docker.com/engine/reference/commandline/network_inspect/) | Display information about a network |
| [network ls](https://docs.docker.com/engine/reference/commandline/network_ls/) | Lists all the networks the Engine daemon knows about |
| [network rm](https://docs.docker.com/engine/reference/commandline/network_rm/) | Removes one or more networks |

### Shared data volume commands

| Command | Description |
| --- | --- |
| [volume create](https://docs.docker.com/engine/reference/commandline/volume_create/) | Creates a new volume where containers can consume and store data |
| [volume inspect](https://docs.docker.com/engine/reference/commandline/volume_inspect/) | Display information about a volume |
| [volume ls](https://docs.docker.com/engine/reference/commandline/volume_ls/) | Lists all the volumes Docker knows about |
| [volume rm](https://docs.docker.com/engine/reference/commandline/volume_rm/) | Remove one or more volumes |

### Swarm node commands

| Command | Description |
| --- | --- |
| [node promote](https://docs.docker.com/engine/reference/commandline/node_promote/) | Promote a node that is pending a promotion to manager |
| [node demote](https://docs.docker.com/engine/reference/commandline/node_demote/) | Demotes an existing manager so that it is no longer a manager |
| [node inspect](https://docs.docker.com/engine/reference/commandline/node_inspect/) | Inspect a node in the swarm |
| [node update](https://docs.docker.com/engine/reference/commandline/node_update/) | Update attributes for a node |
| [node ps](https://docs.docker.com/engine/reference/commandline/node_ps/) | List tasks running on a node |
| [node ls](https://docs.docker.com/engine/reference/commandline/node_ls/) | List nodes in the swarm |
| [node rm](https://docs.docker.com/engine/reference/commandline/node_rm/) | Remove one or more nodes from the swarm |

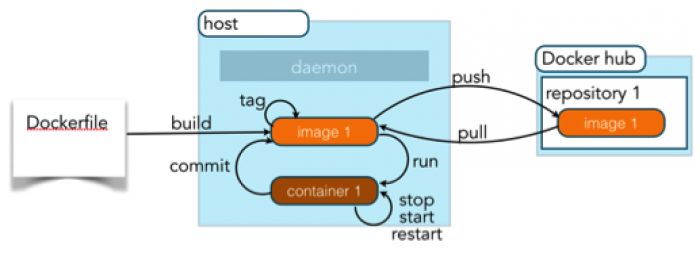
### Swarm swarm commands

| Command | Description |
| --- | --- |
| [swarm init](https://docs.docker.com/engine/reference/commandline/swarm_init/) | Initialize a swarm |
| [swarm join](https://docs.docker.com/engine/reference/commandline/swarm_join/) | Join a swarm as a manager node or worker node |
| [swarm leave](https://docs.docker.com/engine/reference/commandline/swarm_leave/) | Remove the current node from the swarm |
| [swarm update](https://docs.docker.com/engine/reference/commandline/swarm_update/) | Update attributes of a swarm |
| [swarm join-token](https://docs.docker.com/engine/reference/commandline/swarm_join_token/) | Display or rotate join tokens |

### Swarm service commands

| Command | Description |
| --- | --- |
| [service create](https://docs.docker.com/engine/reference/commandline/service_create/) | Create a new service |
| [service inspect](https://docs.docker.com/engine/reference/commandline/service_inspect/) | Inspect a service |
| [service ls](https://docs.docker.com/engine/reference/commandline/service_ls/) | List services in the swarm |
| [service rm](https://docs.docker.com/engine/reference/commandline/service_rm/) | Remove a service from the swarm |
| [service scale](https://docs.docker.com/engine/reference/commandline/service_scale/) | Set the number of replicas for the desired state of the service |
| [service ps](https://docs.docker.com/engine/reference/commandline/service_ps/) | List the tasks of a service |
| [service update](https://docs.docker.com/engine/reference/commandline/service_update/) | Update the attributes of a service |

**Docker work flow**



**Simple Docker workflow - Quick start**

In this tutorials, I am trying to cover the simple quickstart Docker workflow and for the example, I am creating Ubantu containee and using it to showcase this tutorial.

**Step 1 - Download the Ubantu image container from the Docker Hub**

# docker pull -a ubuntu

**Step 2 - Run the ubuntu container and access to ther /bin/bash commands prompt**

# docker run -it ubuntu /bin/bash

**Step 3 - Stop the container**

# docker stop container\_id

**How to get the container id?**

# docker ps -a

**Step 4: Start the container again?**

# docker start container\_id

**Step 5: Exit the running container without stopping the container**

# exit

**Step 6: Login the running container for bash prompt**

# sudo docker exec -i -t 2e56ad1705b1 /bin/bas

**For more - Refer**

<http://www.scmgalaxy.com/scm/package-management/how-to-get-bash-or-ssh-into-a-running-container-in-background-mode.html>

For any issues or errors look at

http://www.devopsschool.com/tutorial/docker/errors-and-resolutions.html

## Location of Dockers images in all Operating Systems

**Location of Dockers images in all Operating Systems**

The location of the images vary depending on the driver Docker is using for storage. Actually, Docker images are stored in two files as shown by following command.

# docker info

**aufs - Most linux version**

/var/lib/docker/aufs/diff/<id> has the file contents of the images.

/var/lib/docker/repositories-aufs is a JSON file containing local image information. This can be viewed with the command docker images.

**btrfs**

TBD

**devicemapper (Redhat)**

/var/lib/docker/devicemapper/devicemapper/data - stores the images

/var/lib/docker/devicemapper/devicemapper/metadata - the metadata

**vfs**

TBD

**Mac OS X**

~/VirtualBox VMs/boot2docker-vm

**Mac OS X using boot2docker**

/Users/rajesh.kumar/.docker/machine/machines/default

File Name - disk.vmdk

**Windows:**

%USERPROFILE%/VirtualBox VMs/boot2docker-vm

You can manually set the storage driver with the -s or --storage-driver= option to the Docker daemon.

/var/lib/docker/{driver-name} will contain the driver specific storage for contents of the images.

/var/lib/docker/graph/<id> now only contains metadata about the image, in the json and layersize files.

**Reference**

<http://stackoverflow.com/questions/19234831/where-are-docker-images-stored-on-the-host-machine>

**Where is the Docker daemon log?**

It depends on your OS. Here are the few locations, with commands for few Operating Systems:

* Ubuntu -Â /var/log/upstart/docker.log
* Boot2Docker -Â /var/log/docker.log
* Debian GNU/Linux -Â /var/log/daemon.log
* CentOS -Â /var/log/daemon.log | grep docker
* CoreOS -Â journalctl -u docker.service
* Fedora -Â journalctl -u docker.service
* Red Hat Enterprise Linux Server -Â /var/log/messages | grep docker
* OpenSuSE -Â journalctl -u docker.service
* OSX -Â ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/log/dâ€Œâ€‹ocker.log

**Note**

* Centos7 seems to log to /var/log/messages
* On an Amazon Linux instance running their Elastic Container Service it's /var/log/docker
* systemd has its own logging system called the journal. The logs for the docker daemon can be viewed using $ journalctl -u docker

## Install Jenkins using Docker

**Step 1: Installing Docker**

# apt-get install docker (Ubuntu)

# yum install docker  (RHEL/CENTOS)

For more info, please following this http://www.scmgalaxy.com/scm/software-containers/how-to-install-docker-in-linux.html

**Step 2:  First, pull the official jenkins image from Docker repository.**

# docker pull jenkins

**Step 3: Next, run a container using this image and map data directory from the container to the host; e.g in the example below /var/jenkins\_home from the container is mapped to jenkins/ directory from the current path on the host. Jenkins 8080 port is also exposed to the host as 49001.**

**Mapping port 8080 on the host to the container (the web ui), port 50000 to port 50000 (for build agents). Run with `-p 50000:50000` so you can connect JNLP slaves. For port 50000. This is to handle connections from JNLP based build slaves. This will store the workspace in /var/jenkins\_home. All Jenkins data lives in there including plugins and configuration.**

> docker run -d -p 8080:8080 -p 50000:50000 jenkins

**This will store the jenkins data in /your/home on the host. Ensure that /your/home is accessible by the jenkins user in container (jenkins user - uid 1000) or use -u some\_other\_user parameter with docker run.**

> docker run -d -p 8080:8080 -p 50000:50000 -u root -v $PWD/jenkins:/var/jenkins\_home jenkins

**Other Example:**

docker run -d -p 49001:8080 -v $PWD/jenkins:/var/jenkins\_home -t jenkins -u root

**This will store the jenkins data in /your/home on the host. Ensure that /your/home is accessible by the jenkins user in container (jenkins user - uid 1000) or use -u some\_other\_user parameter with docker run. This information is also found in the Dockerfile. So all you need to do is to ensure that the directory $PWD/jenkins is own by UID 1000:**

> mkdir jenkins

> chown 1000 jenkins

> docker run -d -p 49001:8080 -v $PWD/jenkins:/var/jenkins\_home -t jenkins

**How to see the Jenkins log?**

> docker exec name tail -f /var/log/jenkins/jenkins.log  
Where name = --name

**Step 3:  Access to j=Jenkins**

As we have successfully run Jenkins Container, we can browse Jenkins Web Interface using our Web Browser by pointint to http://ip-address:49001 or http://localhost:49001 according to the configuration.

## Sonatype Nexus installation using Docker

**1. Download the Docker image using following commands..**

# docker pull sonatype/nexus

**2. Build an image from a Nexus Dockerfile**

# docker build --rm --tag sonatype/nexus oss/

# docker build --rm --tag sonatype/nexus-pro pro/ (For Pro)

**3. To run (if port 8081 is open on your host):**

# docker run -d -p 8081:8081 --name nexus sonatype/nexus:oss

or to assign a random port that maps to port 8081 on the container:

# docker run -d -p 8081 --name nexus sonatype/nexus

**4. To determine the port that the container is listening on:**

# docker ps nexus

**5. To test:**

# curl http://localhost:8081/service/local/status

**Notes:**

1. Default credentials are: admin / admin123

2. It can take some time (2-3 minutes) for the service to launch in a new container. You can tail the log to determine once Nexus is ready:

# docker logs -f nexus

3. Installation of Nexus is to /opt/sonatype/nexus. Notably: /opt/sonatype/nexus/conf/nexus.properties is the properties file.Parameters (nexus-work and nexus-webapp-context-path) definied here are overridden in the JVM invocation.

4. A persistent directory, /sonatype-work, is used for configuration, logs, and storage. This directory needs to be writable by the Nexus process, which runs as UID 200.

5. Four environment variables can be used to control the JVM arguments

CONTEXT\_PATH, passed as -Dnexus-webapp-context-path. This is used to define the

URL which Nexus is accessed.

MAX\_HEAP, passed as -Xmx. Defaults to 768m.

MIN\_HEAP, passed as -Xms. Defaults to 256m.

JAVA\_OPTS. Additional options can be passed to the JVM via this variable.

Default: -server -XX:MaxPermSize=192m -Djava.net.preferIPv4Stack=true.

LAUNCHER\_CONF. A list of configuration files supplied to the

Nexus bootstrap launcher. Default: ./conf/jetty.xml ./conf/jetty-requestlog.xml

6. These can be used supplied at runtime to control the JVM:

# docker run -d -p 8081:8081 --name nexus -e MAX\_HEAP=768m sonatype/nexus  
  
**Reference:**  
https://hub.docker.com/r/sonatype/nexus/

**http://www.devopsschool.com/tutorial/docker/**