Docker Cheat Sheet

wsargent

Inhaltsverzeichnis

1 Docker Cheat Sheet

Want to improve this cheat sheet? See the Contributing section!

1.1 Why Docker

„With Docker, developers can build any app in any language using any toolchain. "Dockerized" apps are completely portable and can run anywhere - colleagues' OS X and Windows laptops, QA servers running Ubuntu in the cloud, and production data center VMs running Red Hat.

Developers can get going quickly by starting with one of the 13,000+ apps available on Docker Hub. Docker manages and tracks changes and dependencies, making it easier for sysadmins to understand how the apps that developers build work. And with Docker Hub, developers can automate their build pipeline and share artifacts with collaborators through public or private repositories.

Docker helps developers build and ship higher-quality applications, faster." – What is Docker

1.2 Prerequisites

I use Oh My Zsh with the Docker plugin for autocompletion of docker commands. YMMV.

1.2.1 Linux

The 3.10.x kernel is the minimum requirement for Docker.

1.2.2 MacOS

10.8 „Mountain Lion" or newer is required.

1.2.3 Windows 10

Hyper-V must be enabled in BIOS VT-D must also be enabled if available (Intel Processors)

1.2.4 Windows Server

Windows Server 2016 is the minimum version required to install docker and docker-compose. Limitations exist on this version, such as multiple virtual networks and linux containers. Windows Server 2019 and later are recommended.

1.3 Installation

1.3.1 Linux

Quick and easy install script provided by Docker:

```
curl -sSL https://get.docker.com/ | sh
```

If you're not willing to run a random shell script, please see the installation instructions for your distribution.

If you are a complete Docker newbie, you should follow the series of tutorials now.

1.3.2 macOS

Download and install Docker Community Edition. if you have Homebrew-Cask, just type `brew cask install docker`. Or Download and install Docker Toolbox. Docker For Mac is nice, but it's not quite as finished as the VirtualBox install. See the comparison.

> **NOTE** Docker Toolbox is legacy. You should to use Docker Community Edition, See Docker Toolbox.

Once you've installed Docker Community Edition, click the docker icon in Launchpad. Then start up a container:

```
docker run hello-world
```

That's it, you have a running Docker container.

If you are a complete Docker newbie, you should probably follow the series of tutorials now.

1.3.3 Windows 10

Instructions to install Docker Desktop for Windows can be found here

Once insalled, open powershell as administrator

```
#Display the version of docker installed:
docker version

##Pull, create, and run 'hello-world' all in one command:
docker run hello-world
```

To continue with this cheat sheet, right click the Docker icon in the system tray, and go to settings. In order to mount volumes, the C:/ drive will need to be enabled in the settings to that information can be passed into the containers (later described in this article).

To switch between Windows containers and Linux containers, right click the icon in the system tray and click the button to switch container operating system Doing this will stop the current containers that are running, and make them unaccessible until the container OS is switched back.

Additionally, if you have WSL or WSL2 installed on your desktop, you might want to install the Linux Kernel for Windows. Instructions can be found here. This requires the Windows Subsystem for Linux feature. This will allow for containers to be accessed by WSL operating systems, as well as the efficiency gain from running WSL operating systems in docker. It is also preferred to use Windows terminal for this.

1.3.4 Windows Server 2016 / 2019

Follow Microsoft's instructions that can be found here

If using the latest edge version of 2019, be prepared to only work in powershell, as it is only a servercore image (no desktop interface). When starting this machine, it will login and go straight to a powershell window. It is reccomended to install text editors and other tools using Chocolatey.

After installing, these commands will work:

```
#Display the version of docker installed:
docker version

##Pull, create, and run 'hello-world' all in one command:
docker run hello-world
```

Windows Server 2016 is not able to run linux images.

Windows Server Build 2004 is capable of running both linux and windows containers simultaneously through Hyper-V isolation. When running containers, use the `--isolation=hyperv` command, which will isolate the container using a seperate kernel instance.

1.3.5 Check Version

It is very important that you always know the current version of Docker you are currently running on at any point in time. This is very helpful because you get to know what features are compatible with what you have running. This is also important because you know what containers to run from the docker store when you are trying to get template containers. That said let see how to know which version of docker we have running currently.

- `docker version` shows which version of docker you have running.

Get the server version:

```
$ docker version --format '{{.Server.Version}}'

1.8.0
```

You can also dump raw JSON data:

```
$ docker version --format '{{json .}}'

{"Client":{"Version":"1.8.0","ApiVersion":"1.20","GitCommit":"f5bae0a",
   "GoVersion":"go1.4.2","Os":"linux","Arch":"am"}
```

1.4 Containers

Your basic isolated Docker process. Containers are to Virtual Machines as threads are to processes. Or you can think of them as chroots on steroids.

1.4.1 Lifecycle

- `docker create` creates a container but does not start it.
- `docker rename` allows the container to be renamed.
- `docker run` creates and starts a container in one operation.
- `docker rm` deletes a container.
- `docker update` updates a container's resource limits.

Normally if you run a container without options it will start and stop immediately, if you want keep it running you can use the command, `docker run -td container_id` this will use the option `-t` that will allocate a pseudo-TTY session and `-d` that will detach automatically the container (run container in background and print container ID).

If you want a transient container, `docker run --rm` will remove the container after it stops.

If you want to map a directory on the host to a docker container, `docker run -v $HOSTDIR:$DOCKERDIR`. Also see Volumes.

If you want to remove also the volumes associated with the container, the deletion of the container must include the `-v` switch like in `docker rm -v`.

There's also a logging driver available for individual containers in docker 1.10. To run docker with a custom log driver (i.e., to syslog), use `docker run --log-driver=syslog`.

Another useful option is `docker run --name yourname docker_image` because when you specify the `--name` inside the run command this will allow you to start and stop a container by calling it with the name the you specified when you created it.

1.4.2 Starting and Stopping

- `docker start` starts a container so it is running.
- `docker stop` stops a running container.
- `docker restart` stops and starts a container.
- `docker pause` pauses a running container, „freezing" it in place.
- `docker unpause` will unpause a running container.
- `docker wait` blocks until running container stops.
- `docker kill` sends a SIGKILL to a running container.
- `docker attach` will connect to a running container.

If you want to detach from a running container, use `Ctrl + p, Ctrl + q`. If you want to integrate a container with a host process manager, start the daemon with `-r=false` then use `docker start -a`.

If you want to expose container ports through the host, see the exposing ports section.

Restart policies on crashed docker instances are covered here.

1.4.2.1 CPU Constraints

You can limit CPU, either using a percentage of all CPUs, or by using specific cores.

For example, you can tell the `cpu-shares` setting. The setting is a bit strange – 1024 means 100% of the CPU, so if you want the container to take 50% of all CPU cores, you should specify 512. See https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/#_cpu for more:

```
docker run -it -c 512 agileek/cpuset-test
```

You can also only use some CPU cores using `cpuset-cpus`. See https://agileek.github.io/docker/2014/08/06/docker-cpuset/ for details and some nice videos:

```
docker run -it --cpuset-cpus=0,4,6 agileek/cpuset-test
```

Note that Docker can still **see** all of the CPUs inside the container – it just isn't using all of them. See https://github.com/docker/docker/issues/20770 for more details.

1.4.2.2 Memory Constraints

You can also set memory constraints on Docker:

```
docker run -it -m 300M ubuntu:14.04 /bin/bash
```

1.4.2.3 Capabilities

Linux capabilities can be set by using `cap-add` and `cap-drop`. See https://docs.docker.com/engine/reference/run/#/runtime-privilege-and-linux-capabilities for details. This should be used for greater security.

To mount a FUSE based filesystem, you need to combine both –cap-add and –device:

```
docker run --rm -it --cap-add SYS_ADMIN --device /dev/fuse sshfs
```

Give access to a single device:

```
docker run -it --device=/dev/ttyUSB0 debian bash
```

Give access to all devices:

```
docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb debian bash
```

More info about privileged containers here.

1.4.3 Info

- `docker ps` shows running containers.
- `docker logs` gets logs from container. (You can use a custom log driver, but logs is only available for `json-file` and `journald` in 1.10).
- `docker inspect` looks at all the info on a container (including IP address).
- `docker events` gets events from container.
- `docker port` shows public facing port of container.
- `docker top` shows running processes in container.
- `docker stats` shows containers' resource usage statistics.
- `docker diff` shows changed files in the container's FS.

`docker ps -a` shows running and stopped containers.

`docker stats --all` shows a list of all containers, default shows just running.

1.4.4 Import / Export

- `docker cp` copies files or folders between a container and the local filesystem.
- `docker export` turns container filesystem into tarball archive stream to STDOUT.

1.4.5 Executing Commands

- `docker exec` to execute a command in container.

To enter a running container, attach a new shell process to a running container called foo, use: `docker exec -it foo /bin/bash`.

1.5 Images

Images are just templates for docker containers.

1.5.1 Lifecycle

- `docker images` shows all images.
- `docker import` creates an image from a tarball.
- `docker build` creates image from Dockerfile.
- `docker commit` creates image from a container, pausing it temporarily if it is running.
- `docker rmi` removes an image.
- `docker load` loads an image from a tar archive as STDIN, including images and tags (as of 0.7).
- `docker save` saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions (as of 0.7).

1.5.2 Info

- `docker history` shows history of image.
- `docker tag` tags an image to a name (local or registry).

1.5.3 Cleaning up

While you can use the `docker rmi` command to remove specific images, there's a tool called docker-gc that will safely clean up images that are no longer used by any containers. As of docker 1.13, `docker image prune` is also available for removing unused images. See Prune.

1.5.4 Load/Save image

Load an image from file:

```
docker load < my_image.tar.gz
```

Save an existing image:

```
docker save my_image:my_tag | gzip > my_image.tar.gz
```

1.5.5 Import/Export container

Import a container as an image from file:

```
cat my_container.tar.gz | docker import - my_image:my_tag
```

Export an existing container:

```
docker export my_container | gzip > my_container.tar.gz
```

1.5.6 Difference between loading a saved image and importing an exported container as an image

Loading an image using the `load` command creates a new image including its history.
Importing a container as an image using the `import` command creates a new image excluding the history which results in a smaller image size compared to loading an image.

1.6 Networks

Docker has a networks feature. Docker automatically creates 3 network interfaces when you install it (bridge, host none). A new container is launched into the bridge network by default. To enable communication between multiple containers, you can create a new network and launch containers in it. This enables containers to communicate to each other while being isolated from containers that are not connected to the network. Furthermore, it allows to map container names to their IP addresses. See working with networks for more details.

1.6.1 Lifecycle

- `docker network create` NAME Create a new network (default type: bridge).

- `docker network rm` NAME Remove one or more networks by name or identifier. No containers can be connected to the network when deleting it.

1.6.2 Info

- `docker network ls` List networks
- `docker network inspect` NAME Display detailed information on one or more networks.

1.6.3 Connection

- `docker network connect` NETWORK CONTAINER Connect a container to a network
- `docker network disconnect` NETWORK CONTAINER Disconnect a container from a network

You can specify a specific IP address for a container:

```
# create a new bridge network with your subnet and gateway for your ip block
docker network create --subnet 203.0.113.0/24 --gateway 203.0.113.254 iptastic

# run a nginx container with a specific ip in that block
$ docker run --rm -it --net iptastic --ip 203.0.113.2 nginx

# curl the ip from any other place (assuming this is a public ip block duh)
$ curl 203.0.113.2
```

1.7 Registry & Repository

A repository is a *hosted* collection of tagged images that together create the file system for a container.

A registry is a *host* – a server that stores repositories and provides an HTTP API for managing the uploading and downloading of repositories.

Docker.com hosts its own index to a central registry which contains a large number of repositories. Having said that, the central docker registry does not do a good job of verifying images and should be avoided if you're worried about security.

- `docker login` to login to a registry.
- `docker logout` to logout from a registry.
- `docker search` searches registry for image.
- `docker pull` pulls an image from registry to local machine.
- `docker push` pushes an image to the registry from local machine.

1.7.1 Run local registry

You can run a local registry by using the docker distribution project and looking at the local deploy instructions.

Also see the mailing list.

1.8 Dockerfile

The configuration file. Sets up a Docker container when you run `docker build` on it. Vastly preferable to `docker commit`.

Here are some common text editors and their syntax highlighting modules you could use to create Dockerfiles: * If you use jEdit, I've put up a syntax highlighting module for Dockerfile you can use. * Sublime Text 2 * Atom * Vim * Emacs * TextMate * VS Code * Also see Docker meets the IDE

1.8.1 Instructions

- .dockerignore
- FROM Sets the Base Image for subsequent instructions.
- MAINTAINER (deprecated - use LABEL instead) Set the Author field of the generated images.
- RUN execute any commands in a new layer on top of the current image and commit the results.
- CMD provide defaults for an executing container.
- EXPOSE informs Docker that the container listens on the specified network ports at runtime. NOTE: does not actually make ports accessible.
- ENV sets environment variable.
- ADD copies new files, directories or remote file to container. Invalidates caches. Avoid `ADD` and use `COPY` instead.
- COPY copies new files or directories to container. By default this copies as root regardless of the USER/WORKDIR settings. Use `--chown=<user>:<group>` to give ownership to another user/group. (Same for `ADD`.)
- ENTRYPOINT configures a container that will run as an executable.
- VOLUME creates a mount point for externally mounted volumes or other containers.
- USER sets the user name for following RUN / CMD / ENTRYPOINT commands.
- WORKDIR sets the working directory.
- ARG defines a build-time variable.
- ONBUILD adds a trigger instruction when the image is used as the base for another build.
- STOPSIGNAL sets the system call signal that will be sent to the container to exit.
- LABEL apply key/value metadata to your images, containers, or daemons.

- SHELL override default shell is used by docker to run commands.
- HEALTHCHECK tells docker how to test a container to check that it is still working.

1.8.2 Tutorial

- Flux7's Dockerfile Tutorial

1.8.3 Examples

- Examples
- Best practices for writing Dockerfiles
- Michael Crosby has some more Dockerfiles best practices / take 2.
- Building Good Docker Images / Building Better Docker Images
- Managing Container Configuration with Metadata
- How to write excellent Dockerfiles

1.9 Layers

The versioned filesystem in Docker is based on layers. They're like git commits or changesets for filesystems.

1.10 Links

Links are how Docker containers talk to each other through TCP/IP ports. Atlassian show worked examples. You can also resolve links by hostname.

This has been deprecated to some extent by user-defined networks.

NOTE: If you want containers to ONLY communicate with each other through links, start the docker daemon with `-icc=false` to disable inter process communication.

If you have a container with the name CONTAINER (specified by `docker run --name CONTAINER`) and in the Dockerfile, it has an exposed port:

```
EXPOSE 1337
```

Then if we create another container called LINKED like so:

```
docker run -d --link CONTAINER:ALIAS --name LINKED user/wordpress
```

Then the exposed ports and aliases of CONTAINER will show up in LINKED with the following environment variables:

```
$ALIAS_PORT_1337_TCP_PORT
$ALIAS_PORT_1337_TCP_ADDR
```

And you can connect to it that way.

To delete links, use `docker rm --link`.

Generally, linking between docker services is a subset of „service discovery“, a big problem if you're planning to use Docker at scale in production. Please read The Docker Ecosystem: Service Discovery and Distributed Configuration Stores for more info.

1.11 Volumes

Docker volumes are free-floating filesystems. They don't have to be connected to a particular container. You can use volumes mounted from data-only containers for portability. As of Docker 1.9.0, Docker has named volumes which replace data-only containers. Consider using named volumes to implement it rather than data containers.

1.11.1 Lifecycle

- `docker volume create`
- `docker volume rm`

1.11.2 Info

- `docker volume ls`
- `docker volume inspect`

Volumes are useful in situations where you can't use links (which are TCP/IP only). For instance, if you need to have two docker instances communicate by leaving stuff on the filesystem.

You can mount them in several docker containers at once, using `docker run --volumes-from`.

Because volumes are isolated filesystems, they are often used to store state from computations between transient containers. That is, you can have a stateless and transient container run from a recipe, blow it away, and then have a second instance of the transient container pick up from where the last one left off.

See advanced volumes for more details. Container42 is also helpful.

You can map MacOS host directories as docker volumes:

```
docker run -v /Users/wsargent/myapp/src:/src
```

You can use remote NFS volumes if you're feeling brave.

You may also consider running data-only containers as described here to provide some data portability.

Be aware that you can mount files as volumes.

1.12 Exposing ports

Exposing incoming ports through the host container is fiddly but doable.

This is done by mapping the container port to the host port (only using localhost interface) using `-p`:

```
docker run -p 127.0.0.1:$HOSTPORT:$CONTAINERPORT --name CONTAINER -t
    someimage
```

You can tell Docker that the container listens on the specified network ports at runtime by using EXPOSE:

```
EXPOSE <CONTAINERPORT>
```

Note that EXPOSE does not expose the port itself – only `-p` will do that. To expose the container's port on your localhost's port:

```
iptables -t nat -A DOCKER -p tcp --dport <LOCALHOSTPORT> -j DNAT --to-
    destination <CONTAINERIP>:<PORT>
```

If you're running Docker in Virtualbox, you then need to forward the port there as well, using forwarded_port. Define a range of ports in your Vagrantfile like this so you can dynamically map them:

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  ...

  (49000..49900).each do |port|
    config.vm.network :forwarded_port, :host => port, :guest => port
  end

  ...
end
```

If you forget what you mapped the port to on the host container, use `docker port` to show it:

```
docker port CONTAINER $CONTAINERPORT
```

1.13 Best Practices

This is where general Docker best practices and war stories go:

- The Rabbit Hole of Using Docker in Automated Tests
- Bridget Kromhout has a useful blog post on running Docker in production at Dramafever.
- There's also a best practices blog post from Lyst.
- Building a Development Environment With Docker
- Discourse in a Docker Container

1.14 Docker-Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see the list of features.

By using the following command you can start up your application:

```
docker-compose -f <docker-compose-file> up
```

You can also run docker-compose in detached mode using -d flag, then you can stop it whenever needed by the following command:

```
docker-compose stop
```

You can bring everything down, removing the containers entirely, with the down command. Pass `--volumes` to also remove the data volume.

1.15 Security

This is where security tips about Docker go. The Docker security page goes into more detail.

First things first: Docker runs as root. If you are in the `docker` group, you effectively have root access. If you expose the docker unix socket to a container, you are giving the container root access to the host.

Docker should not be your only defense. You should secure and harden it.

For an understanding of what containers leave exposed, you should read Understanding and Hardening Linux Containers by Aaron Grattafiori. This is a complete and comprehensive guide to the issues involved with containers, with a plethora of links and footnotes leading on to yet more useful content. The security tips following are useful if you've already hardened containers in the past, but are not a substitute for understanding.

1.15.1 Security Tips

For greatest security, you want to run Docker inside a virtual machine. This is straight from the Docker Security Team Lead – slides / notes. Then, run with AppArmor / seccomp / SELinux / grsec etc to limit the container permissions. See the Docker 1.10 security features for more details.

Docker image ids are sensitive information and should not be exposed to the outside world. Treat them like passwords.

See the Docker Security Cheat Sheet by Thomas Sjögren: some good stuff about container hardening in there.

Check out the docker bench security script, download the white papers.

Snyk's 10 Docker Image Security Best Practices cheat sheet

You should start off by using a kernel with unstable patches for grsecurity / pax compiled in, such as Alpine Linux. If you are using grsecurity in production, you should spring for commercial support for the stable patches, same as you would do for RedHat. It's $200 a month, which is nothing to your devops budget.

Since docker 1.11 you can easily limit the number of active processes running inside a container to prevent fork bombs. This requires a linux kernel >= 4.3 with CGROUP_PIDS=y to be in the kernel configuration.

```
docker run --pids-limit=64
```

Also available since docker 1.11 is the ability to prevent processes from gaining new privileges. This feature have been in the linux kernel since version 3.5. You can read more about it in this blog post.

```
docker run --security-opt=no-new-privileges
```

From the Docker Security Cheat Sheet (it's in PDF which makes it hard to use, so copying below) by Container Solutions:

Turn off interprocess communication with:

```
docker -d --icc=false --iptables
```

Set the container to be read-only:

```
docker run --read-only
```

Verify images with a hashsum:

```
docker pull debian@sha256:a25306f3850e1bd44541976aa7b5fd0a29be
```

Set volumes to be read only:

```
docker run -v $(pwd)/secrets:/secrets:ro debian
```

Define and run a user in your Dockerfile so you don't run as root inside the container:

```
RUN groupadd -r user && useradd -r -g user user
USER user
```

1.15.2 User Namespaces

There's also work on user namespaces – it is in 1.10 but is not enabled by default.

To enable user namespaces („remap the userns") in Ubuntu 15.10, follow the blog example.

1.15.3 Security Videos

- Using Docker Safely
- Securing your applications using Docker
- Container security: Do containers actually contain?
- Linux Containers: Future or Fantasy?

1.15.4 Security Roadmap

The Docker roadmap talks about seccomp support. There is an AppArmor policy generator called bane, and they're working on security profiles.

1.16 Tips

Sources:

- 15 Docker Tips in 5 minutes
- CodeFresh Everyday Hacks Docker

1.16.1 Prune

The new Data Management Commands have landed as of Docker 1.13:

- `docker system prune`
- `docker volume prune`
- `docker network prune`
- `docker container prune`
- `docker image prune`

1.16.2 df

`docker system df` presents a summary of the space currently used by different docker objects.

1.16.3 Heredoc Docker Container

```
docker build -t htop - << EOF
FROM alpine
RUN apk --no-cache add htop
EOF
```

1.16.4 Last Ids

```
alias dl='docker ps -l -q'
docker run ubuntu echo hello world
docker commit $(dl) helloworld
```

1.16.5 Commit with command (needs Dockerfile)

```
docker commit -run='{"Cmd":["postgres", "-too -many -opts"]}' $(dl)
    postgres
```

1.16.6 Get IP address

```
docker inspect $(dl) | grep -wm1 IPAddress | cut -d '"' -f 4
```

or with jq installed:

```
docker inspect $(dl) | jq -r '.[0].NetworkSettings.IPAddress'
```

or using a go template:

```
docker inspect -f '{{ .NetworkSettings.IPAddress }}' <container_name>
```

or when building an image from Dockerfile, when you want to pass in a build argument:

```
DOCKER_HOST_IP=`ifconfig | grep -E "([0-9]{1,3}\.){3}[0-9]{1,3}" | grep -v 127.0.0.1 | awk '{ print $2 }' | cut -f2 -d: | head -n1`
echo DOCKER_HOST_IP = $DOCKER_HOST_IP
docker build \
  --build-arg ARTIFACTORY_ADDRESS=$DOCKER_HOST_IP
  -t sometag \
  some-directory/
```

1.16.7 Get port mapping

```
docker inspect -f '{{range $p, $conf := .NetworkSettings.Ports}} {{$p}} -> {{(index $conf 0).HostPort}} {{end}}' <containername>
```

1.16.8 Find containers by regular expression

```
for i in $(docker ps -a | grep "REGEXP_PATTERN" | cut -f1 -d" "); do echo $i; done
```

1.16.9 Get Environment Settings

```
docker run --rm ubuntu env
```

1.16.10 Kill running containers

```
docker kill $(docker ps -q)
```

1.16.11 Delete all containers (force!! running or stopped containers)

```
docker rm -f $(docker ps -qa)
```

1.16.12 Delete old containers

```
docker ps -a | grep 'weeks ago' | awk '{print $1}' | xargs docker rm
```

1.16.13 Delete stopped containers

```
docker rm -v $(docker ps -a -q -f status=exited)
```

1.16.14 Delete containers after stopping

```
docker stop $(docker ps -aq) && docker rm -v $(docker ps -aq)
```

1.16.15 Delete dangling images

```
docker rmi $(docker images -q -f dangling=true)
```

1.16.16 Delete all images

```
docker rmi $(docker images -q)
```

1.16.17 Delete dangling volumes

As of Docker 1.9:

```
docker volume rm $(docker volume ls -q -f dangling=true)
```

In 1.9.0, the filter `dangling=false` does *not* work - it is ignored and will list all volumes.

1.16.18 Show image dependencies

```
docker images -viz | dot -Tpng -o docker.png
```

1.16.19 Slimming down Docker containers

- Cleaning APT in a RUN layer

This should be done in the same layer as other apt commands. Otherwise, the previous layers still persist the original information and your images will still be fat.

```
RUN {apt commands} \
  && apt-get clean \
  && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

- Flatten an image

```
ID=$(docker run -d image-name /bin/bash)
docker export $ID | docker import - flat-image-name
```

- For backup

```
ID=$(docker run -d image-name /bin/bash)
(docker export $ID | gzip -c > image.tgz)
gzip -dc image.tgz | docker import - flat-image-name
```

1.16.20 Monitor system resource utilization for running containers

To check the CPU, memory, and network I/O usage of a single container, you can use:

```
docker stats <container>
```

For all containers listed by id:

```
docker stats $(docker ps -q)
```

For all containers listed by name:

```
docker stats $(docker ps --format '{{.Names}}')
```

For all containers listed by image:

```
docker ps -a -f ancestor=ubuntu
```

Remove all untagged images:

```
docker rmi $(docker images | grep “^” | awk '{split($0,a," "); print a[3]}')
```

Remove container by a regular expression:

```
docker ps -a | grep wildfly | awk '{print $1}' | xargs docker rm -f
```

Remove all exited containers:

```
docker rm -f $(docker ps -a | grep Exit | awk '{ print $1 }')
```

1.16.21 Volumes can be files

Be aware that you can mount files as volumes. For example you can inject a configuration file like this:

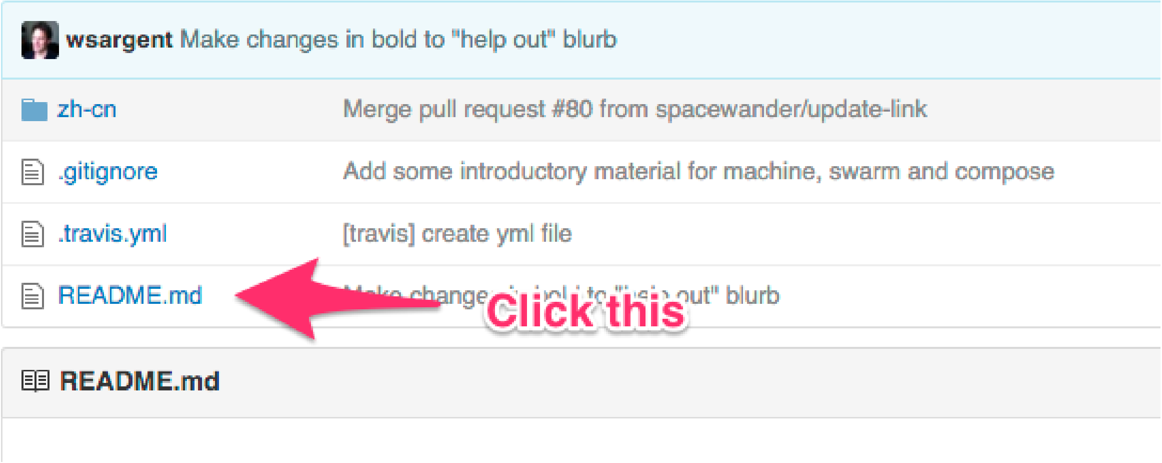
```
# copy file from container
docker run --rm httpd cat /usr/local/apache2/conf/httpd.conf > httpd.conf

# edit file
vim httpd.conf

# start container with modified configuration
docker run --rm -it -v "$PWD/httpd.conf:/usr/local/apache2/conf/httpd.conf:ro" -p "80:80" httpd
```

1.17 Contributing

Here's how to contribute to this cheat sheet.

1.17.1 Open README.md

Click README.md <– this link



wsargent Make changes in bold to "help out" blurb

zh-cn	Merge pull request #80 from spacewander/update-link
.gitignore	Add some introductory material for machine, swarm and compose
.travis.yml	[travis] create yml file
README.md	Make changes in bold to "help out" blurb

README.md

Abbildung 1: Click This

1.17.2 Edit Page

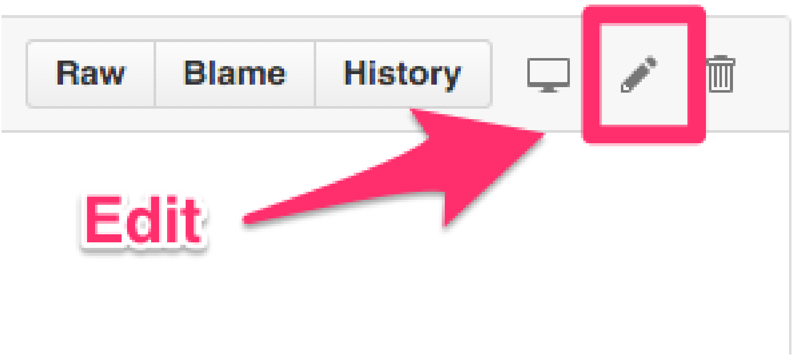
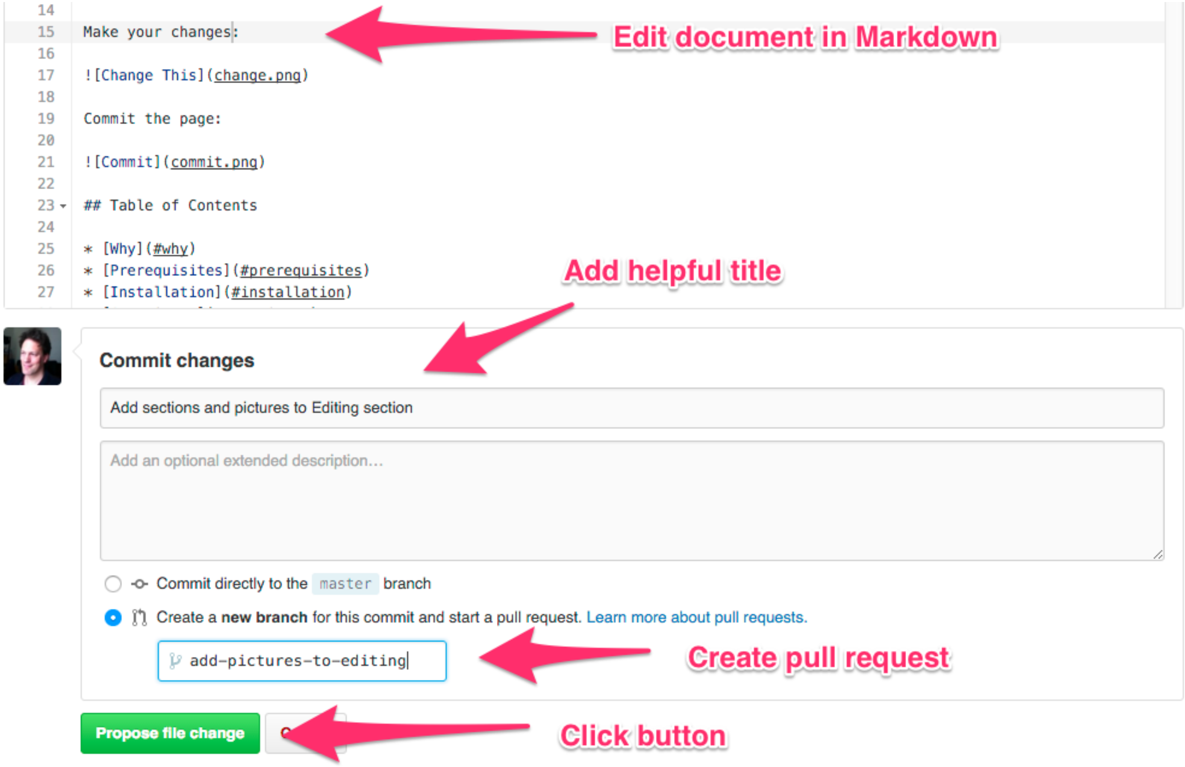


Abbildung 2: Edit This

1.17.3 Make Changes and Commit



```
Make your changes:

![Change This](change.png)

Commit the page:

![Commit](commit.png)

## Table of Contents

* [Why](#why)
* [Prerequisites](#prerequisites)
* [Installation](#installation)
```

Commit changes

Add sections and pictures to Editing section

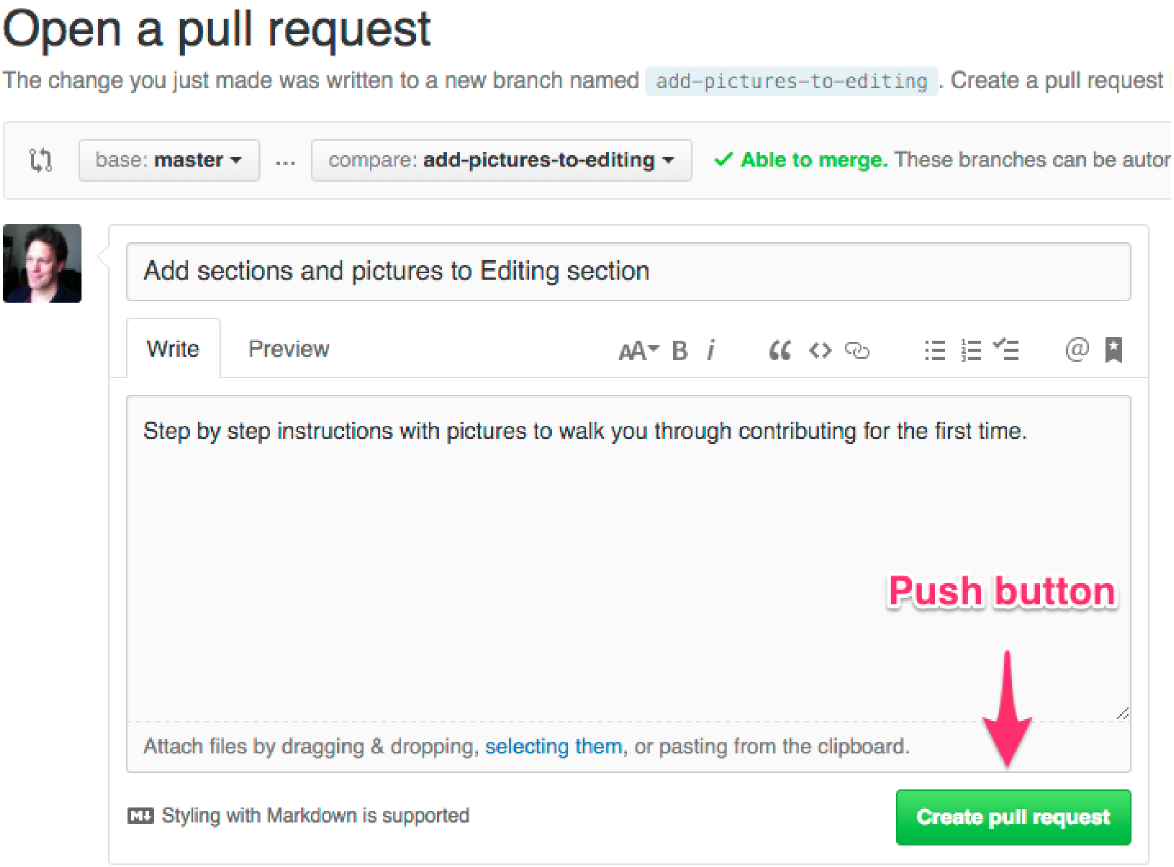
Add an optional extended description…

○ Commit directly to the `master` branch

● Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

add-pictures-to-editing

Abbildung 3: Change This



Open a pull request

The change you just made was written to a new branch named `add-pictures-to-editing`. Create a pull request

base: master ... compare: add-pictures-to-editing ✓ Able to merge. These branches can be autor

Add sections and pictures to Editing section

Step by step instructions with pictures to walk you through contributing for the first time.

Write
Preview
Push button
Attach files by dragging & dropping, selecting them, or pasting from the clipboard.
Styling with Markdown is supported
Create pull request

Abbildung 4: Commit