Docker Cheat Sheet

Author: Anshita Bhasin
LinkedIn: https://www.linkedin.com/in/anshita-bhasin/

Docker is a software platform that allows you to build, deploy, and run applications quickly. It uses OS-level virtualization to deliver software in packages called containers, which allows you to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Docker provides a way to run these containers in a consistent environment, regardless of the host operating system or infrastructure. This makes it easier to run and manage applications, as well as to move them between different environments.

Below are some of the most used docker commands:

1). docker login:

The *docker login* command allows you to authenticate with a registry and specify any required credentials. To log in to a Docker registry, you can use the *docker login* command.

Below is an example of how to use *docker login* to log in to the Docker Hub registry:

```
$ docker login
Username: myusername
Password:
Login Succeeded
```

In this example, you will be prompted to enter your Docker Hub username and password. Once you have entered these credentials, the *docker login* command will log you into the Docker Hub registry.

You can also specify your credentials directly on the command line using the *—username* and *—password* flags:

```
$ docker login --username myusername --password mypassword
myregistry.example.com
Login Succeeded
```

Note that using the *—password* flag is generally not recommended, as it can expose your password in your shell's history and in process listings. Instead, you can use the *—password-stdin* flag to pass your password via standard input or through a file, which can be useful when using the *docker login* command in scripts:

Pass Password through standard input

```
$ echo "mypassword" | docker login --username myusername
--password-stdin myregistry.example.com
Login Succeeded
```

Pass Password through a file

```
$ cat ~/my_password.txt | docker login --username foo --password-stdin
```

2). docker pull:

The *docker pull* command is used to download a Docker image from a registry, such as Docker Hub. You can use it to download an image to your local machine before running a container from that image.

To use the *docker pull* command to download an image from a registry, you will need to specify the name of the image that you want to download. For example, to download the latest version of the official Ubuntu Docker image, you can use the following command:

```
$ docker pull ubuntu
```

This will download the latest version of the Ubuntu image from the Docker Hub registry.

You can also specify a specific version of the image by including the image's tag name. For example, to download version 18.04 of the Ubuntu image, you can use the following command:

```
$ docker pull ubuntu:18.04
```

By default, the *docker pull* command will download the image from the Docker Hub registry. However, you can also specify a different registry path by including the registry's URL as part of the image name. For example, to download *test/my-image* image from a local registry listening on port 5000, you can use a command like the following:

```
$ docker pull registry.local:5000/test/my-image
```

This will download the *latest* version of the *my-image* image from the ACME registry.

3). docker build:

The *docker build* command is used to build a Docker image from a Dockerfile (A Dockerfile is a text file that contains instructions for building a Docker image). It reads the instructions in the Dockerfile and creates an image based on those instructions.

To build a Docker image from a Dockerfile, you can use the *docker build* command. Here is an example of how to use this command:

```
$ docker build -t myimage:latest .
```

This command will build an image named "*myimage*" with the "*latest*" tag, using the Dockerfile in the current directory (indicated by the . at the end of the command).

4). docker push:

The *docker push* command is used to upload a Docker image to a registry. You can use it to publish an image that you have built locally or to update an image that you have previously pushed to a registry.

To push a Docker image to a registry (such as Docker Hub), you can use the *docker push* command. Below is an example:

```
$ docker push myimage:latest
```

This command will push the image named "*myimage*" with the "*latest*" tag to the registry that is currently logged in to.

Make sure, before you push an image to a registry, you will need to log in to the registry using the *docker login* command. For example:

```
$ docker login
Username: myusername
Password: ********
Login Succeeded
```

You can also specify the registry URL as an argument when pushing the image, like this:

```
$ docker push myimage:latest
```

This can be useful if you want to push the image to a registry other than the one you are currently logged in to.

```
$ docker push registry.example.com/myimage:latest
```

Note: When pushing an image to a registry, you may need to include the registry URL in the image name. For example, if you are pushing an image to Docker Hub, the full image name would be in the format ***username/image:tag*.**

5). docker run

The *docker run* command is used to run a Docker container. It allows you to specify various options and arguments to control how the container is run.

Below is an example of how to use the *docker run* command

```
$ docker run hello-world
```

In this example, the *hello-world* image is pulled from the Docker Hub registry, and a new container is created from it. The container is then run and the output is displayed on the terminal.

6). docker run -d

The *-d* flag is used with the *docker run* command to run a container in detached mode. In detached mode, the container runs in the background and the command prompt is returned to the user immediately after the

container is started.

Below is an example of how to use the *-d* flag with the *docker run* command:

```
$ docker run -d mycontainer
```

In this example, the *docker run* command starts a container from the *mycontainer* image in detached mode. The container will run in the background and the command prompt will be returned to the user immediately.

7). docker run -p

The *-p* flag is used with the *docker run* command to expose/map a container's ports to the host ports:

For example, if you have an image named *myapp* and you want to run a container from that image on port 8080, you can use the following command:

```
$ docker run -p 8080:8080 myapp
```

This will start a container based on the *myapp* image and expose port 8080 on the container to port 8080 on the host.

You can also specify a range of ports to expose. For example, the following command will expose ports 8000–8100 on the host to the same range of ports on the container:

```
$ docker run -p 8000-8100:8000-8100 myapp
```

8). docker run -v

The *-v* flag is used with the *docker run* command to mount a volume in a Docker container. A volume is a persistent storage location that exists outside the container's file system and can be used to store data that needs to be preserved across container restarts.

To mount a volume using the *-v* flag, you need to specify the host path and the container path, separated by a colon. For example:

```
$ docker run -v /host/path:/container/path myapp
```

This will mount the directory at /host/path on the host machine as a volume in the container at /container/path.

You can also specify read-only access to the volume by adding*:ro* to the end of the host path. For example:

```
$ docker run -v /host/path:/container/path:ro myapp
```

This will mount the volume in read-only mode, which means that the container will not be able to make any changes to the files in the volume.

9). docker images

The *docker images* command is used to list the images that are available on your Docker host. After running the *docker image* command, it will display a list of all images available on your host, along with their repository, tag, image ID, creation date, and size.

Below is an example of how to use the *docker images* command to list all images:

```
$ docker images
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
myimage         latest       1234567890ab   2 hours ago    1GB
otherimage      v1           0987654321cd   3 days ago     500MB
```

In this example, the *docker images* command lists all images available on the Docker host, along with their repository, tag, image ID, creation date, and size.

10). docker images -a

The *-a* flag is used with the *docker images* command to show all images, including intermediate images (images that are used as a base for other images but are not directly used to create containers)

Below is an example of how to use the *-a* flag with the *docker images* command:

```
$ docker images -a
REPOSITORY      TAG          IMAGE ID       CREATED        SIZE
myimage         latest       1234567890ab   2 hours ago    1GB
otherimage      v1           0987654321cd   3 days ago     500MB
<none>          <none>       0123456789ef   4 days ago     300MB
```

In this example, the *docker images* command lists all images available on the Docker host, including the top-level images and the intermediate images. The top-level images are *myimage* and *otherimage* , and the *intermediate image* is the one with the <*none*> repository and tag.

11). docker images—format

The *--format* flag is used with the *docker images to* specify a custom output format for the images list:

```
$ docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
IMAGE ID          REPOSITORY         TAG
1234567890ab      myimage            latest
0987654321cd      otherimage         v1
0123456789ef      <none>             <none>
```

In this example, the *–format* flag is used to specify a custom output format that includes the image ID, repository, and tag for each image. The output is displayed in a table format, with each field separated by a tab character.

You can use various fields in the Go template to customize the output. For example, you can use *{{.Created}}* to include the creation date of the images, or *{{.Size}}* to include the size of the images. You can also use *{{.ID}}, {{.Repository}},* and *{{.Tag}}*, as in the example above.

Here is another example that shows how to use the *—format* flag to list the image IDs and sizes in a comma-separated list:

```
$ docker images --format "{{.ID}}, {{.Size}}"
1234567890ab, 1GB
0987654321cd, 500MB
```

12). docker images -q

The *-q* flag is used with the *docker images* command to show only the IDs of the images. When you use the *-q* flag, the *docker images* command will display a list of image IDs, one per line.

Here is an example of how to use the *-q* flag with the *docker images* command:

```
$ docker images -q
1234567890ab
0987654321cd
```

In this example, the *docker images* command lists the IDs of all images available on the Docker host.

You can use the *-q* flag in combination with other flags and arguments to further filter and customize the output. For example, you can use the *-f* flag to filter the images by a specific pattern, or the *—filter* flag to filter the images by a specific key-value pair.

Below is an example that shows how to use the *-q* flag with the *-f* flag to list the IDs of the images that match a specific pattern:

```
$ docker images -q -f "label=mylabel"
1234567890ab
```

13). docker rm

The *docker run* command is used to remove one or more Docker containers. When you remove a container, it is stopped if it is running and then deleted along with any associated resources, such as volumes and networks.

To remove a single container, you can use the *docker rm* command followed by the name or ID of the container you want to delete. For example:

```
$ docker rm mycontainer
```

To remove multiple containers at once, you can specify the names or IDs of the containers separated by a space. For example:

```
$ docker rm mycontainer1 mycontainer2 mycontainer3
```

You can also use the *-f* flag to force the removal of a running container. This can be useful if you want to delete a container that is stuck in a running state and cannot be stopped normally.

```
$ docker rm -f mycontainer
```

Make sure, once a container is removed, any data stored in the container's file system is deleted and cannot be recovered.

14). docker start

The *docker start* command is used to start one or more stopped containers. It is followed by the name or ID of the container.

Below is an example to use *docker start* command:

```
$ docker start myapp
```

This will start the container with the name *myapp*

15). docker start -a

The *-a* flag is used with the *docker start* command to attach to the container and see its output. The *-a* flag stands for "attach," and it allows you to attach to a running Docker container and see its output. When you start a container with the *-a* flag, the command will not return until you stop the container or detach from it.

Below is an example of how to start a container and attach to it:

```
$ docker start -a myapp
```

This will start the container with the name *myapp* and attach to it, allowing you to see its output. To detach from the container and leave it running in the background, you can use the *Ctrl-C* keyboard shortcut.

Note, *-a* flag only works with running containers. If the container is stopped, the *-a* flag will have no effect.

16). docker attach

The *docker attach* command is used to attach your terminal to a running container. It allows you to interact with the container and execute commands inside the container in real-time.

To use docker attach, you need to specify the container name or ID as an argument. For example:

```
$ docker attach myapp
```

This will attach to the container with the name *myapp* and display its output. To detach from the container and leave it running in the background, you can use the Ctrl-C keyboard shortcut.

17). docker ps -a

The *docker ps -a* command is used to list all Docker containers, including stopped ones, on a host. It displays a list of containers with their names, IDs, and current status (e.g., "Up" for running containers and "Exited" for stopped containers).

Below is an example of how to use the *docker ps -a* command:

```
$ docker ps -a
```

This will list all Docker containers, including stopped ones, on the host.

Tip: If you don't know the name or ID of the container you want to start, you can use the *docker ps -a* command to list all stopped and running containers, along with their names and IDs.

18). docker ps -l

The *-l* flag with the *docker ps* command is used to show only the latest created container. It displays a list of the latest created container with its name, ID, and current status.

Below is an example of how to use the -l flag:

```
$ docker ps -l
```

19). docker ps -n

The *-n* flag with the *docker ps* command is used to show the n last created containers with their names, IDs, and current statuses.

Below is an example to show last 2 created containers:

```
$ docker ps -n 2
```

20). docker ps—format

The—*format* flag for the *docker ps* command is used to customize the output of the command. It allows you to specify a Go template to control the formatting of the output.

Below is an example of how to use the—*format* flag:

```
$ docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Command}}\t{{.RunningFor}}\t{{.Status}}\t{{.Ports}}"
```

This will display the containers in a table format, with the ID, image, command, running time, status, and ports of each container.

You can use the following placeholders in the Go template to include specific information in the output:

- .ID: the container ID
- .Image: the image used to create the container
- .Command: the command used to run the container
- .CreatedAt: the time the container was created
- .RunningFor: the amount of time the container has been running
- .Status: the current status of the container (e.g., "Up" for running containers and "Exited" for stopped containers)
- .Ports: the ports exposed by the container

21). docker stop

The *docker stop* command is used to stop a running Docker container. When you stop a container, the process running inside the container is terminated, and the container is no longer running.

To stop a container, you can use the *docker stop* command followed by the name or ID of the container.

Below is an example to use *docker stop* command:

```
$ docker stop myapp
```

This will stop the container with the name *myapp*.

22). docker stop -t

The *-t* flag with *docker stop is used to specify* the time to wait for the container to stop before sending the SIGKILL signal. The time is specified in seconds, and the default value is 10 seconds.

Below is an example of how to use the *docker stop -t* command:

```
$ docker stop -t 20 myapp
```

This will stop the container with the name *myapp* and wait 20 seconds before killing it.

23). docker stats

The *docker stats* command is used to display resource usage statistics for one or more running Docker containers. It shows real-time resource usage information, such as CPU, memory, and network usage, for each container.

Below is an example of how to use the *docker stats* command:

```
$ docker stats myapp
```

This will show resource usage statistics for the container with the name *myapp*.

You can also use the *docker stats* command without specifying a container name to show resource usage statistics for all running containers.

Below is an example to use *docker stats* for all the running containers:

```
$ docker stats
```

This will show resource usage statistics for all running containers.

24). docker stats—format

The—*format* flag with the *docker stats* command is used to customize the output of the *docker stats* command.

For example, to display only the names and CPU usage of the containers, the below command, is used:

```
$ docker stats --format "table {{.Name}}\t{{.CPUPerc}}"
```

25). docker kill

The *docker kill* command is used to kill one or more running Docker containers. This command sends a SIGKILL signal to the main process inside the container, which terminates the process and all its subprocesses.

To use *docker kill*, you need to specify the name or ID of the container you want to kill. For example:

```
$ docker kill container_name
```

```
$ docker kill container_id
```

26). docker version

The *docker version* command is used to display the version number of Docker and the other relevant information, such as the build date and the version of the Docker Engine.

Below is an example of how to use *docker version* command:

```
$ docker --version
Docker version 19.03.13, build 4484c46d9d
```

27). docker logs

The *docker logs* command allows you to view the logs of a running Docker container. By default, it shows the standard output (stdout) and standard error (stderr) of the container's main process.

Below is an example of how to use the *docker logs* command:

```
$ docker logs myapp
```

This command will display the logs of the container with the name or ID *myapp*.

28). docker logs -f

The *docker logs -f* command allows you to view the logs of a running Docker container in real-time. It is similar to the *tail -f* command, which follows the end of a file and displays new lines as they are added.

Below is an example of how to use the *docker logs -f* command:

```
$ docker logs -f mycontainer
```

This command will display the logs of the container with the name or ID “mycontainer” as they are generated by the container.

29). docker logs -f—tail

The—*tail* flag with the command *docker logs* is used to specify the number of lines to show from the end of the logs from a running Docker container

Below is an example of how to use the *docker logs -f—tail n* command:

```
$ docker logs -f --tail 10 mycontainer
```

This will display the last 10 lines of the logs of the container with the name or ID “mycontainer”and then follow the logs in real-time.

Tip: You can specify a different number of lines to show from the end of the logs by replacing 10 with the desired number.

30). docker inspect

The *docker inspect* command allows you to view detailed information about a Docker container or image. It returns a JSON object that contains metadata about the container or image, such as its ID, name, and state, as well as information about its network settings, mounts, and resource limits.

Below is an example of how to use the *docker inspect* command for a container:

```
$ docker inspect mycontainer
```

This command will display detailed information about the container with the name or ID “mycontainer”.

31). docker image inspect

The *docker image inspect* command allows you to view detailed information about a Docker image. It returns a JSON array that contains metadata about the image, such as its ID, name, and tags, as well as information about its layers, history, and config.

Below is an example of how to use the *docker image inspect* command:

```
$ docker image inspect myimage:latest
```

This command will display detailed information about the image with the name “*myimage*” and the tag “*latest*”.

You can also use the *docker inspect* command for an image, like this:

```
$ docker inspect myimage:latest
```

This will display detailed information about the image with the name “myimage” and the tag “latest”.

32). docker exec

The *docker exec* command allows you to run a command in a running Docker container. It is useful for executing commands inside a container, such as running a shell or starting a new process.

Below is an example of how to use the *docker exec* command:

```
$ docker exec mycontainer command
```

This command will run the *command* in the container with the name or ID "mycontainer".

You can also run a shell inside the container, like this:

```
$ docker exec -it mycontainer bash
```

This will open a Bash shell inside the container, allowing you to run commands interactively.

Conclusion:

Overall, Docker can be a powerful tool for automating tests, as it allows you to easily create and manage consistent, reproducible test environments, and helps you to speed up the testing process. Above shared are the most commonly used docker commands.

Thanks for reading. Happy learning!

Ref: https://docs.docker.com/

Author: Anshita Bhasin