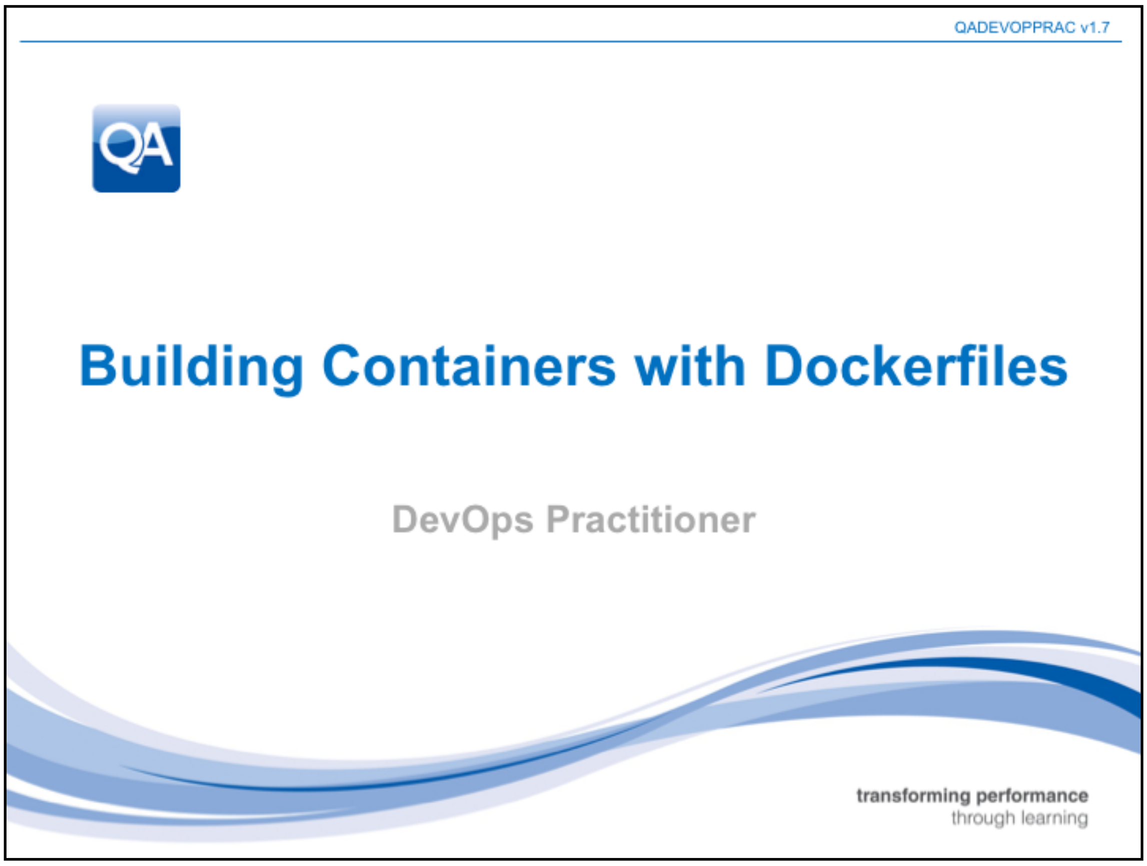


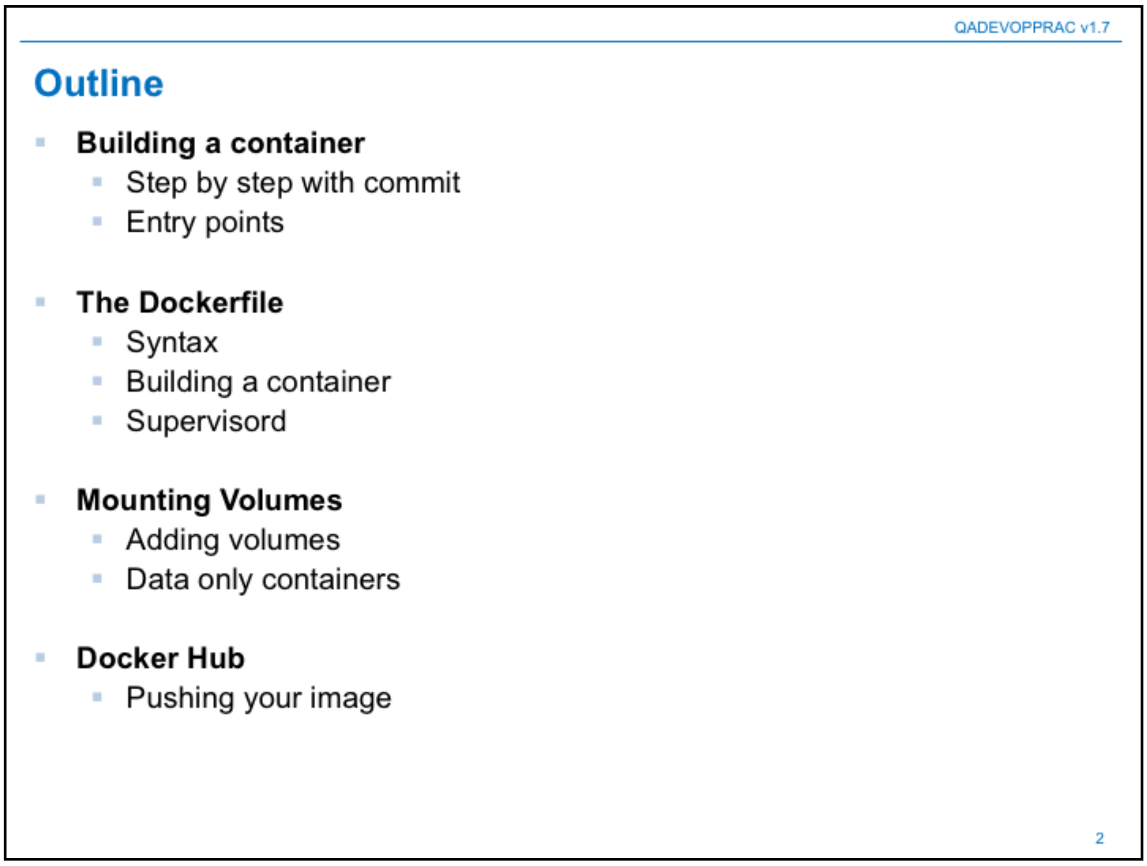
QADEVOPPRAC v1.7

Building Containers with Dockerfiles

DevOps Practitioner

transforming performance
through learning

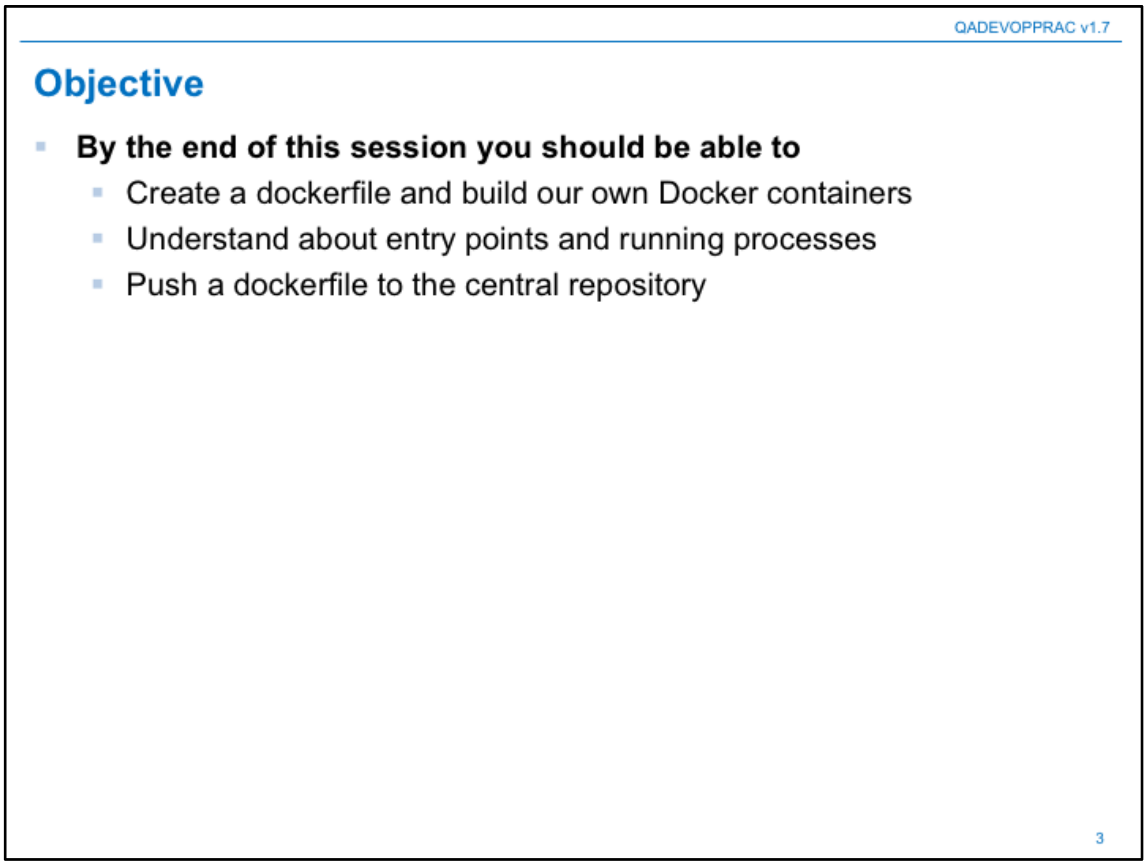
http://neyto.blogspot.co.uk/?view=magazine – summary for docker



Outline

- **Building a container**
  - Step by step with commit
  - Entry points
- **The Dockerfile**
  - Syntax
  - Building a container
  - Supervisord
- **Mounting Volumes**
  - Adding volumes
  - Data only containers
- **Docker Hub**
  - Pushing your image

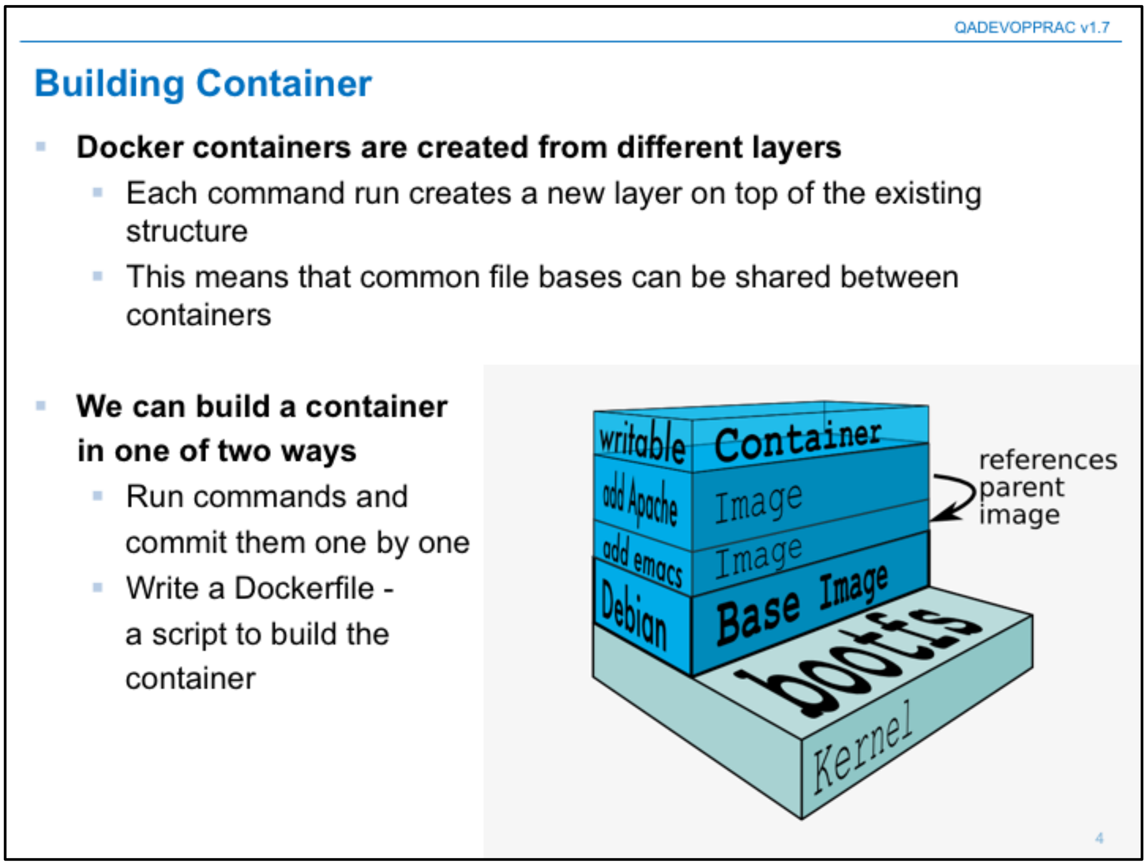
2



Objective

- **By the end of this session you should be able to**
  - Create a dockerfile and build our own Docker containers
  - Understand about entry points and running processes
  - Push a dockerfile to the central repository

3



Building Container

- **Docker containers are created from different layers**
  - Each command run creates a new layer on top of the existing structure
  - This means that common file bases can be shared between containers
- **We can build a container in one of two ways**
  - Run commands and commit them one by one
  - Write a Dockerfile - a script to build the container

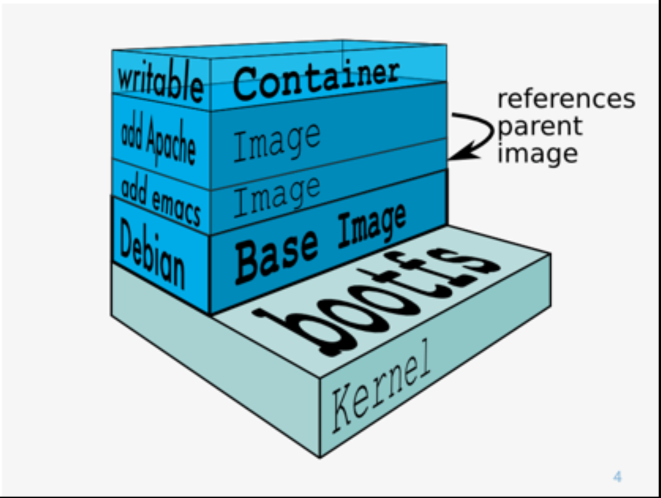
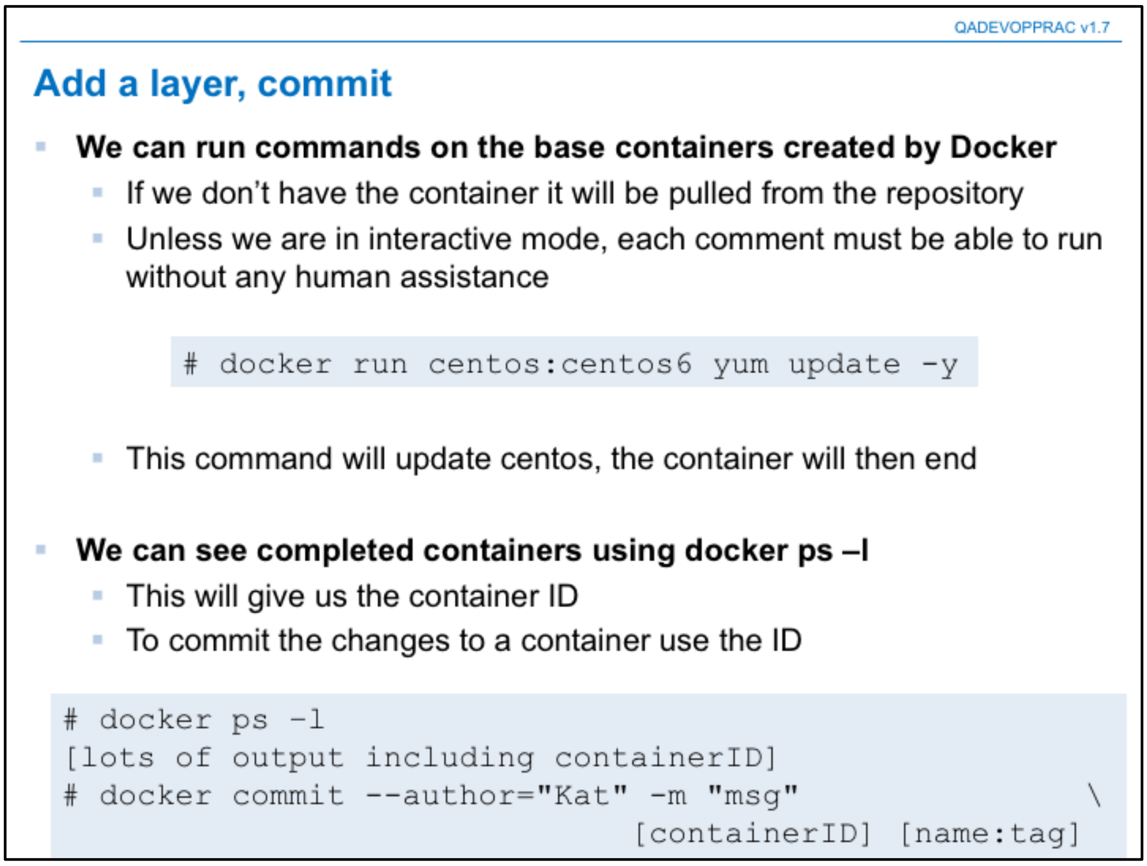


Image from the Docker website: https://docs.docker.com/terms/layer/



QADEVOPPRAC v1.7

Add a layer, commit

- **We can run commands on the base containers created by Docker**
  - If we don't have the container it will be pulled from the repository
  - Unless we are in interactive mode, each comment must be able to run without any human assistance

```
# docker run centos:centos6 yum update -y
```

  - This command will update centos, the container will then end

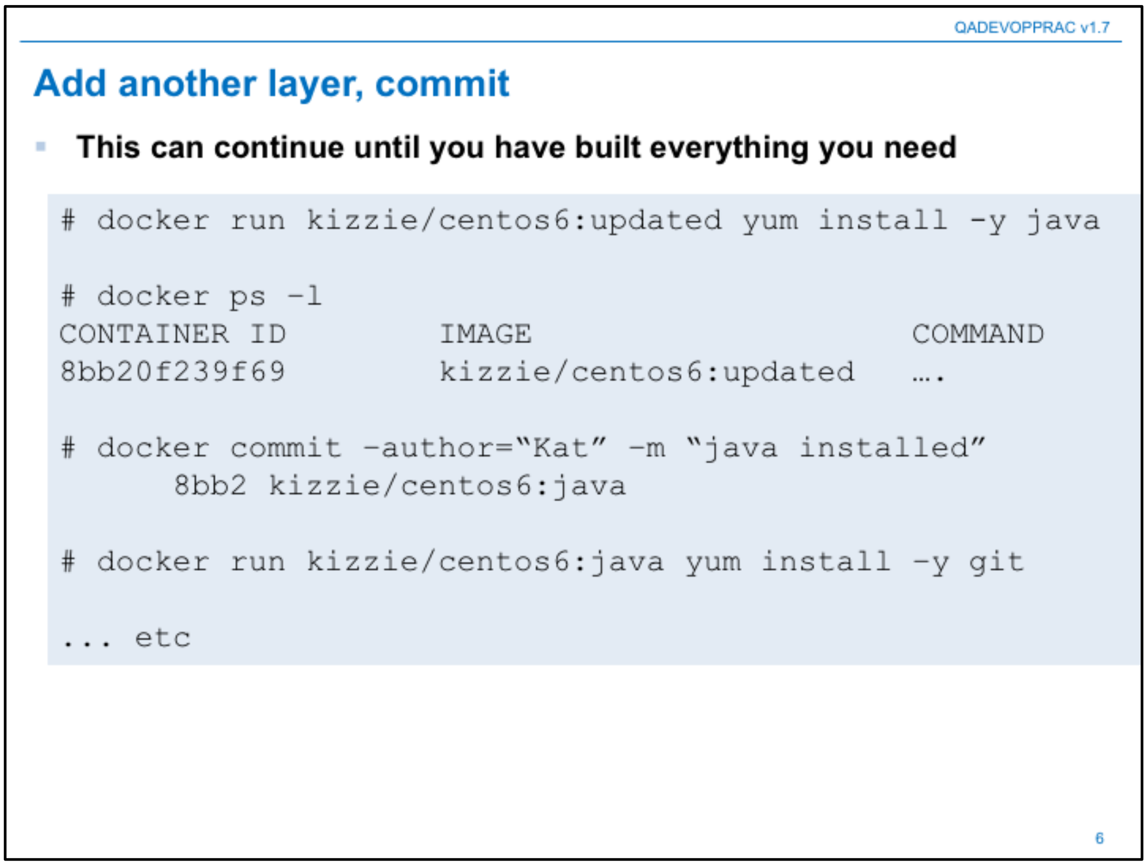
- **We can see completed containers using docker ps –l**
  - This will give us the container ID
  - To commit the changes to a container use the ID

```
# docker ps -l
[lots of output including containerID]
# docker commit --author="Kat" -m "msg"                \
                                [containerID] [name:tag]
```

The naming convention is usually to use your Docker hub username and a tag to give it a version, this doesn't need to be a number. For example

docker commit --author="Kat" -m "Updated Centos" a3e8 kizzie/centos6:updated

When you use the container ID you do not need to put the full hash, usually the first four characters will suffice.



Add another layer, commit

- This can continue until you have built everything you need

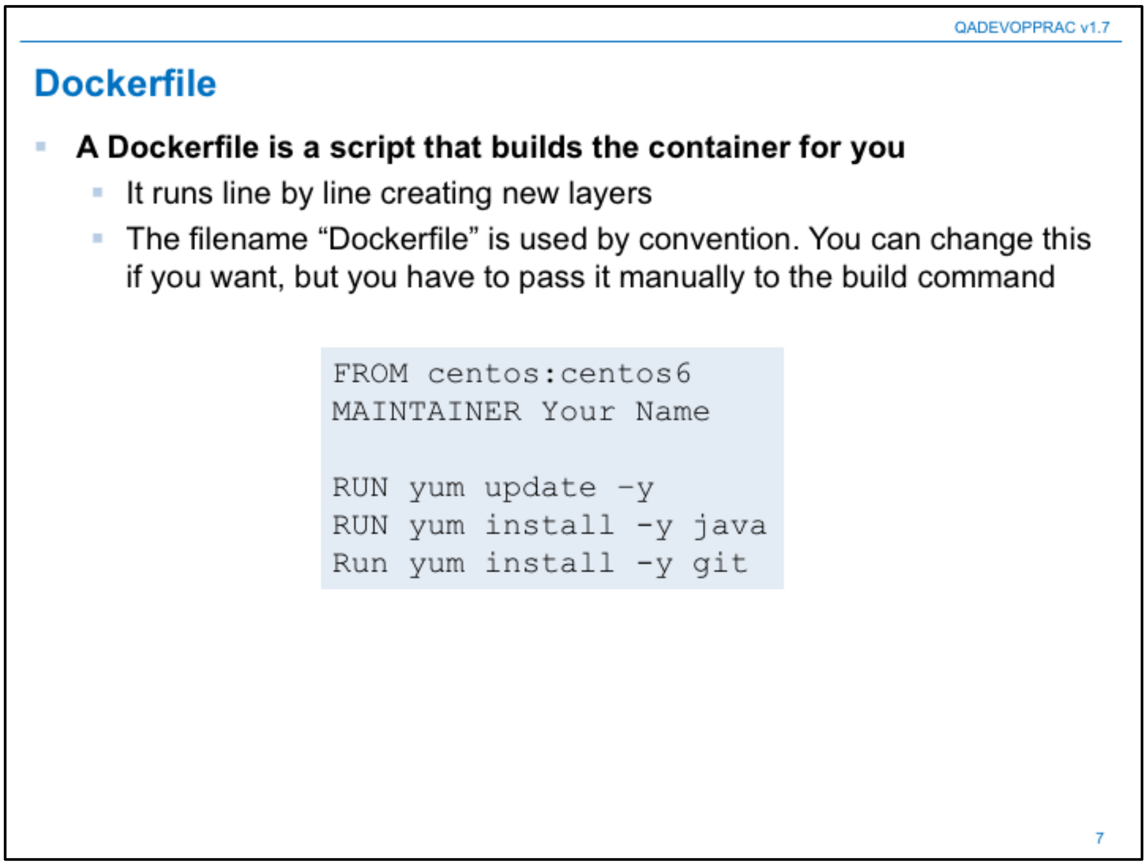
```
# docker run kizzie/centos6:updated yum install -y java

# docker ps -l
CONTAINER ID         IMAGE                     COMMAND
8bb20f239f69         kizzie/centos6:updated    ….

# docker commit -author="Kat" -m "java installed"
      8bb2 kizzie/centos6:java

# docker run kizzie/centos6:java yum install -y git

... etc
```

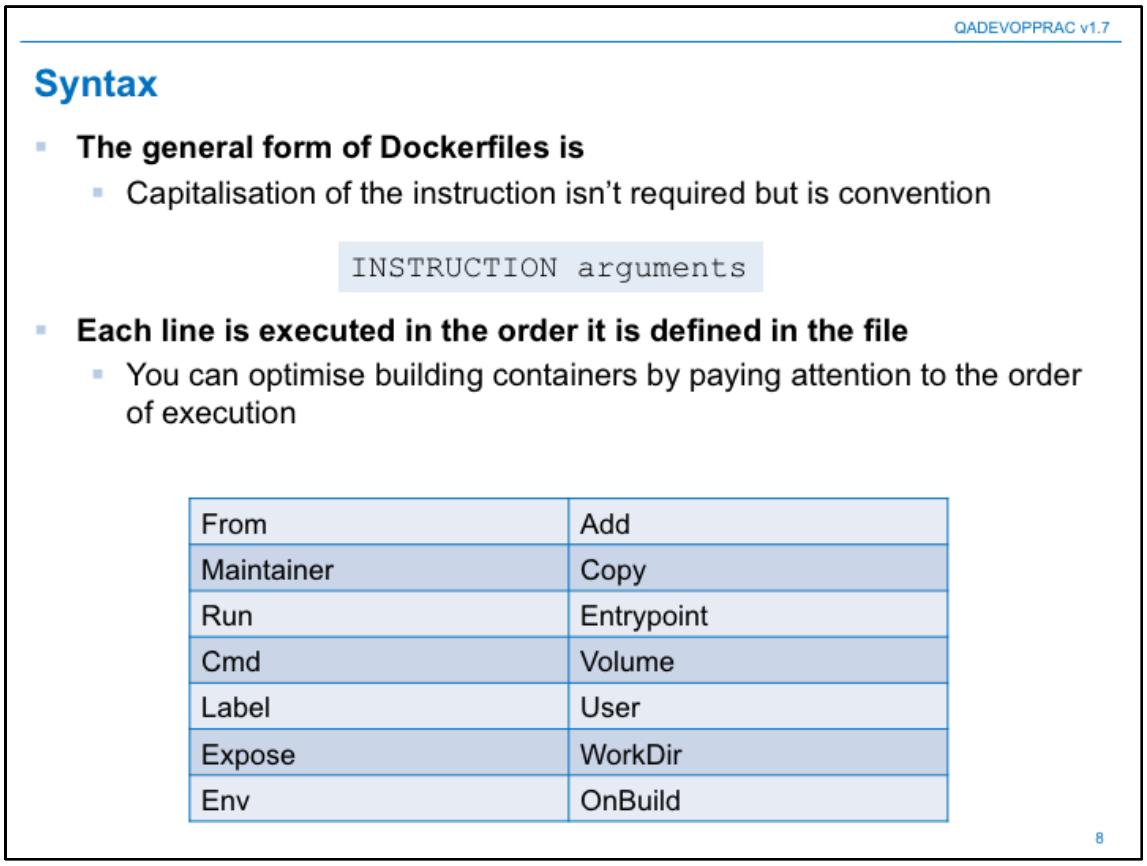


Dockerfile

- **A Dockerfile is a script that builds the container for you**
  - It runs line by line creating new layers
  - The filename “Dockerfile” is used by convention. You can change this if you want, but you have to pass it manually to the build command

```
FROM centos:centos6
MAINTAINER Your Name

RUN yum update -y
RUN yum install -y java
Run yum install -y git
```



Syntax

- **The general form of Dockerfiles is**
  - Capitalisation of the instruction isn't required but is convention

```
INSTRUCTION arguments
```

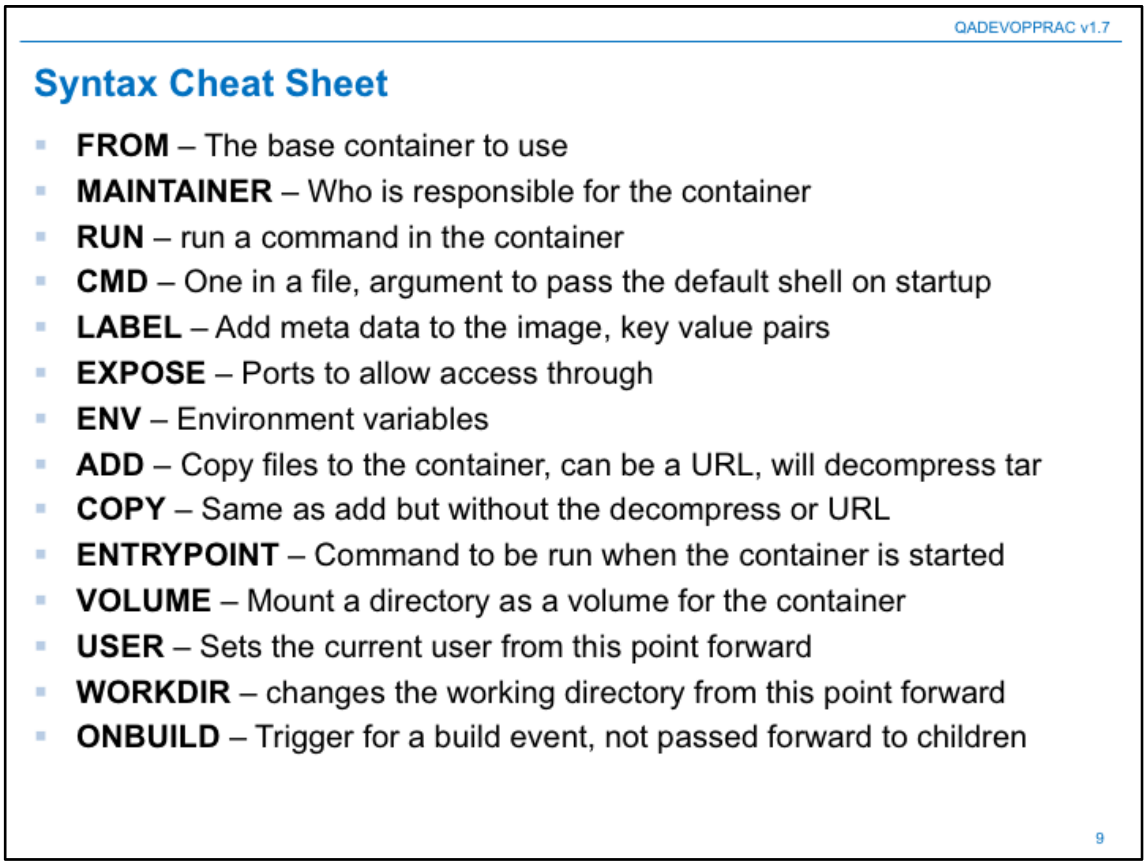
- **Each line is executed in the order it is defined in the file**
  - You can optimise building containers by paying attention to the order of execution

From	Add
Maintainer	Copy
Run	Entrypoint
Cmd	Volume
Label	User
Expose	WorkDir
Env	OnBuild

8

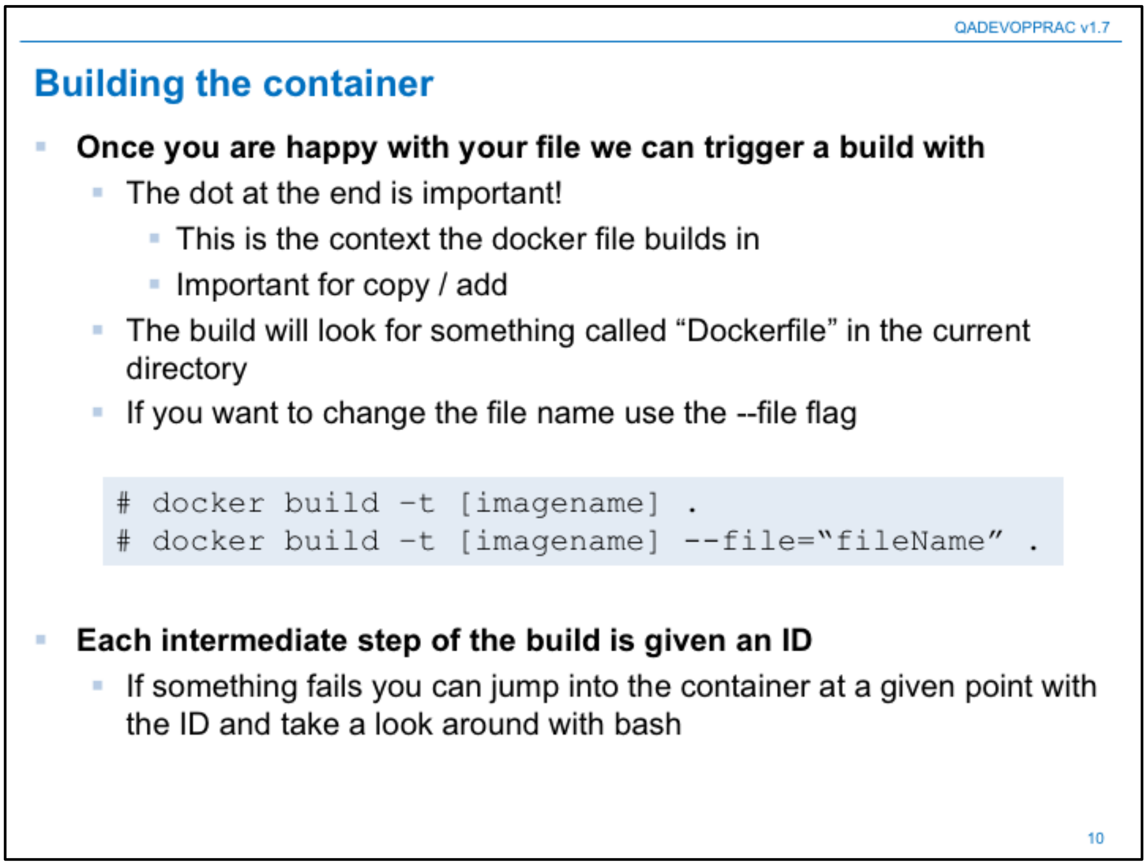
The full documentation for the build files is at:

https://docs.docker.com/reference/builder/



Syntax Cheat Sheet

- **FROM** – The base container to use
- **MAINTAINER** – Who is responsible for the container
- **RUN** – run a command in the container
- **CMD** – One in a file, argument to pass the default shell on startup
- **LABEL** – Add meta data to the image, key value pairs
- **EXPOSE** – Ports to allow access through
- **ENV** – Environment variables
- **ADD** – Copy files to the container, can be a URL, will decompress tar
- **COPY** – Same as add but without the decompress or URL
- **ENTRYPOINT** – Command to be run when the container is started
- **VOLUME** – Mount a directory as a volume for the container
- **USER** – Sets the current user from this point forward
- **WORKDIR** – changes the working directory from this point forward
- **ONBUILD** – Trigger for a build event, not passed forward to children

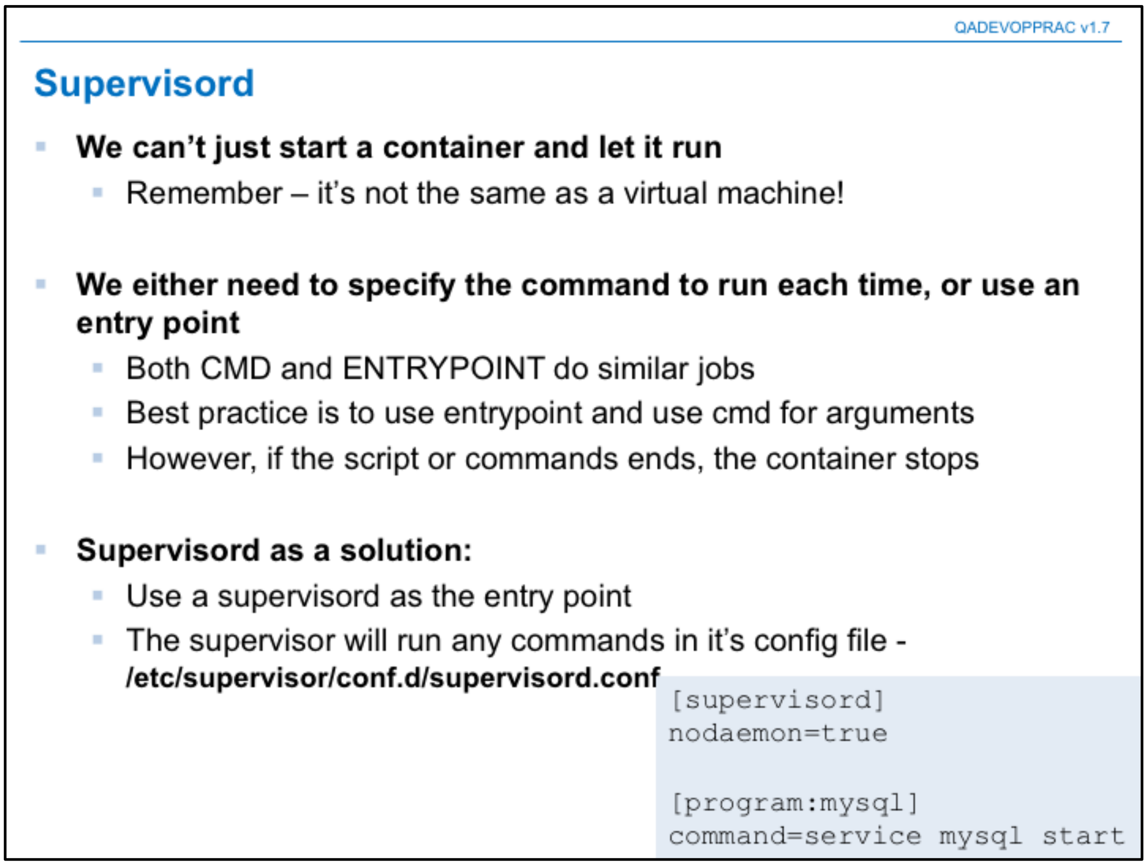


Building the container

- **Once you are happy with your file we can trigger a build with**
  - The dot at the end is important!
    - This is the context the docker file builds in
    - Important for copy / add
  - The build will look for something called “Dockerfile” in the current directory
  - If you want to change the file name use the --file flag

```
# docker build -t [imagename] .
# docker build -t [imagename] --file="fileName" .
```

- **Each intermediate step of the build is given an ID**
  - If something fails you can jump into the container at a given point with the ID and take a look around with bash



Supervisord

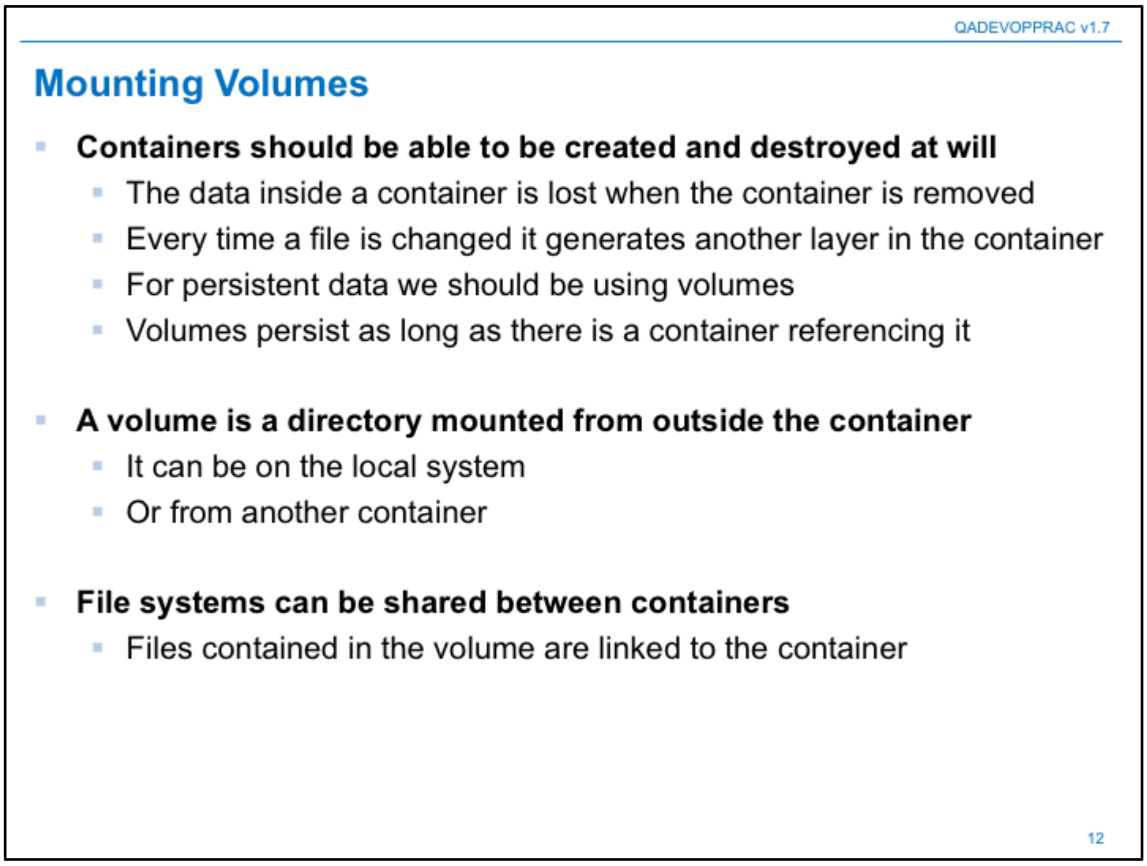
- **We can’t just start a container and let it run**
  - Remember – it’s not the same as a virtual machine!

- **We either need to specify the command to run each time, or use an entry point**
  - Both CMD and ENTRYPOINT do similar jobs
  - Best practice is to use entrypoint and use cmd for arguments
  - However, if the script or commands ends, the container stops

- **Supervisord as a solution:**
  - Use a supervisord as the entry point
  - The supervisor will run any commands in it’s config file - **/etc/supervisor/conf.d/supervisord.conf**

```
[supervisord]
nodaemon=true

[program:mysql]
command=service mysql start
```



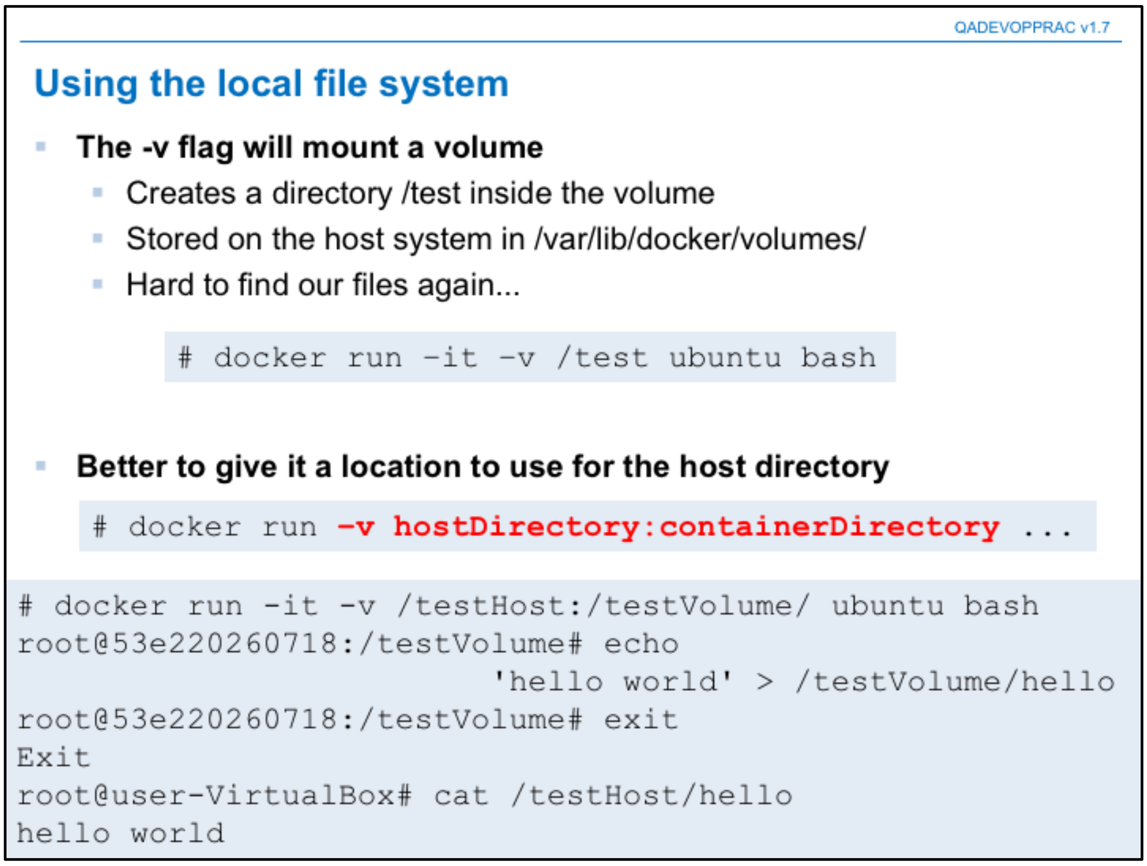
QADEVOPPRAC v1.7

Mounting Volumes

- **Containers should be able to be created and destroyed at will**
  - The data inside a container is lost when the container is removed
  - Every time a file is changed it generates another layer in the container
  - For persistent data we should be using volumes
  - Volumes persist as long as there is a container referencing it
- **A volume is a directory mounted from outside the container**
  - It can be on the local system
  - Or from another container
- **File systems can be shared between containers**
  - Files contained in the volume are linked to the container

12

See http://container-solutions.com/understanding-volumes-docker/ for a very well put together explanation of how the layers file system and volumes interact.



Using the local file system

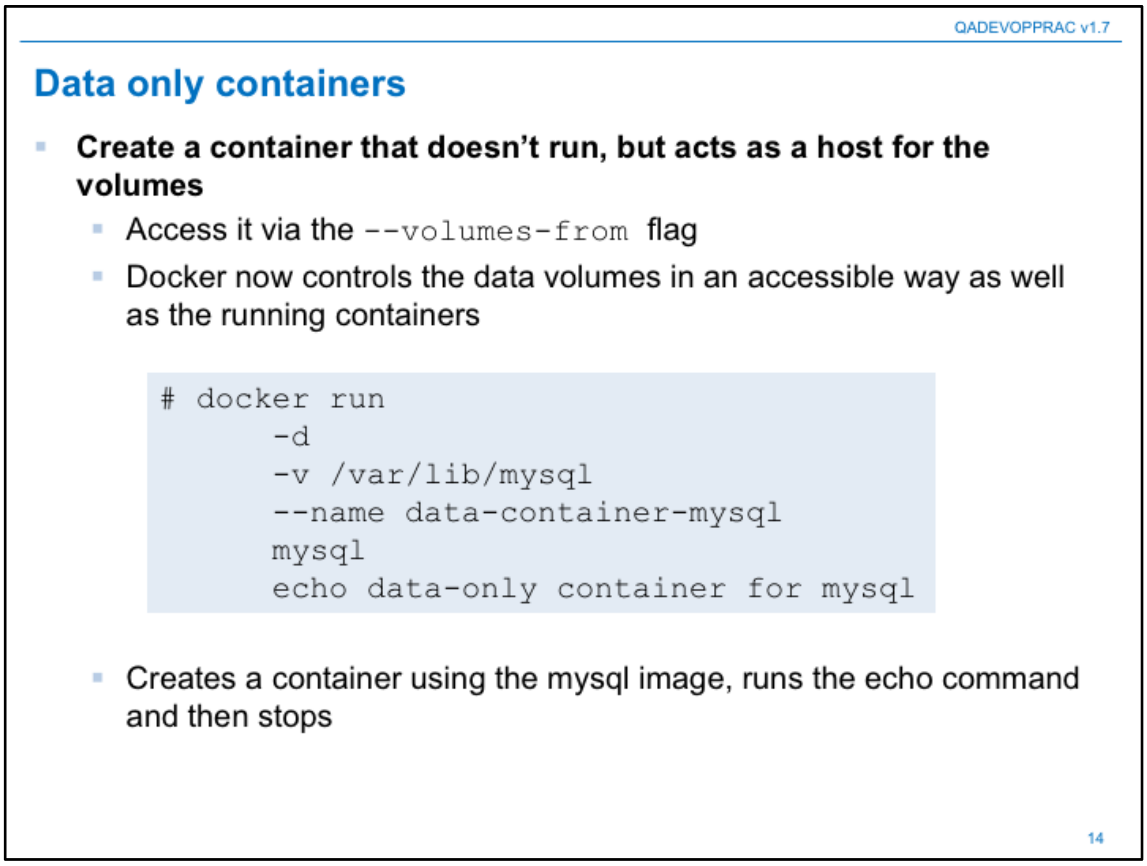
- **The -v flag will mount a volume**
  - Creates a directory /test inside the volume
  - Stored on the host system in /var/lib/docker/volumes/
  - Hard to find our files again...

```
# docker run -it -v /test ubuntu bash
```

- **Better to give it a location to use for the host directory**

```
# docker run -v hostDirectory:containerDirectory ...
```

```
# docker run -it -v /testHost:/testVolume/ ubuntu bash
root@53e220260718:/testVolume# echo
                         'hello world' > /testVolume/hello
root@53e220260718:/testVolume# exit
Exit
root@user-VirtualBox# cat /testHost/hello
hello world
```

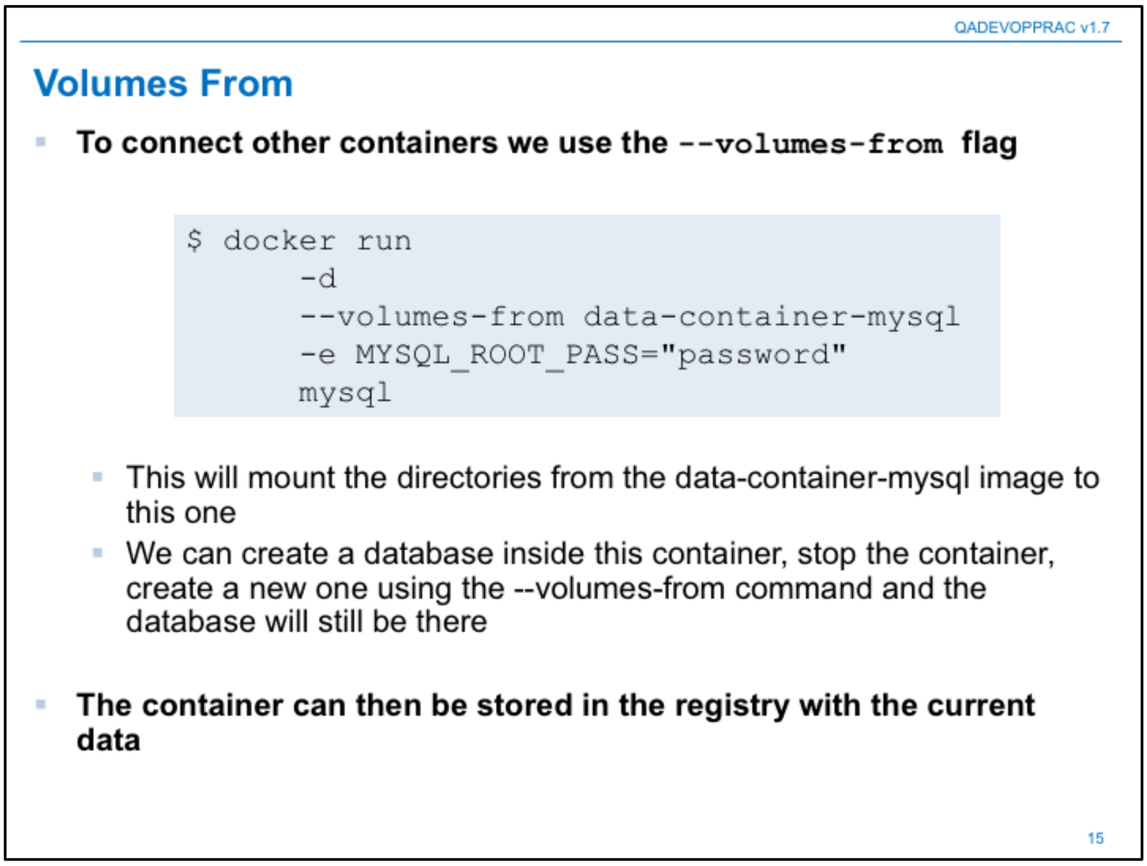


Data only containers

- **Create a container that doesn't run, but acts as a host for the volumes**
  - Access it via the `--volumes-from` flag
  - Docker now controls the data volumes in an accessible way as well as the running containers

```
# docker run
      -d
      -v /var/lib/mysql
      --name data-container-mysql
      mysql
      echo data-only container for mysql
```

  - Creates a container using the mysql image, runs the echo command and then stops



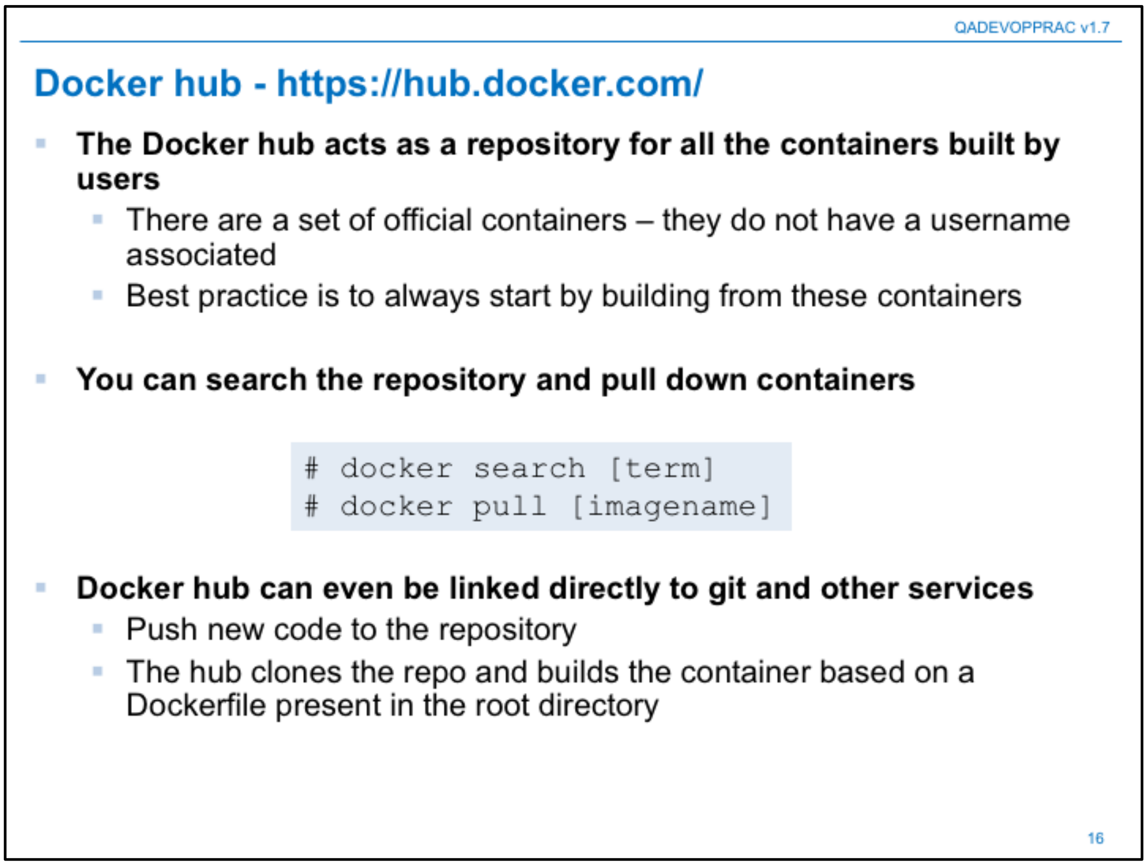
Volumes From

- **To connect other containers we use the `--volumes-from` flag**

```
$ docker run
      -d
      --volumes-from data-container-mysql
      -e MYSQL_ROOT_PASS="password"
      mysql
```

  - This will mount the directories from the data-container-mysql image to this one
  - We can create a database inside this container, stop the container, create a new one using the --volumes-from command and the database will still be there

- **The container can then be stored in the registry with the current data**

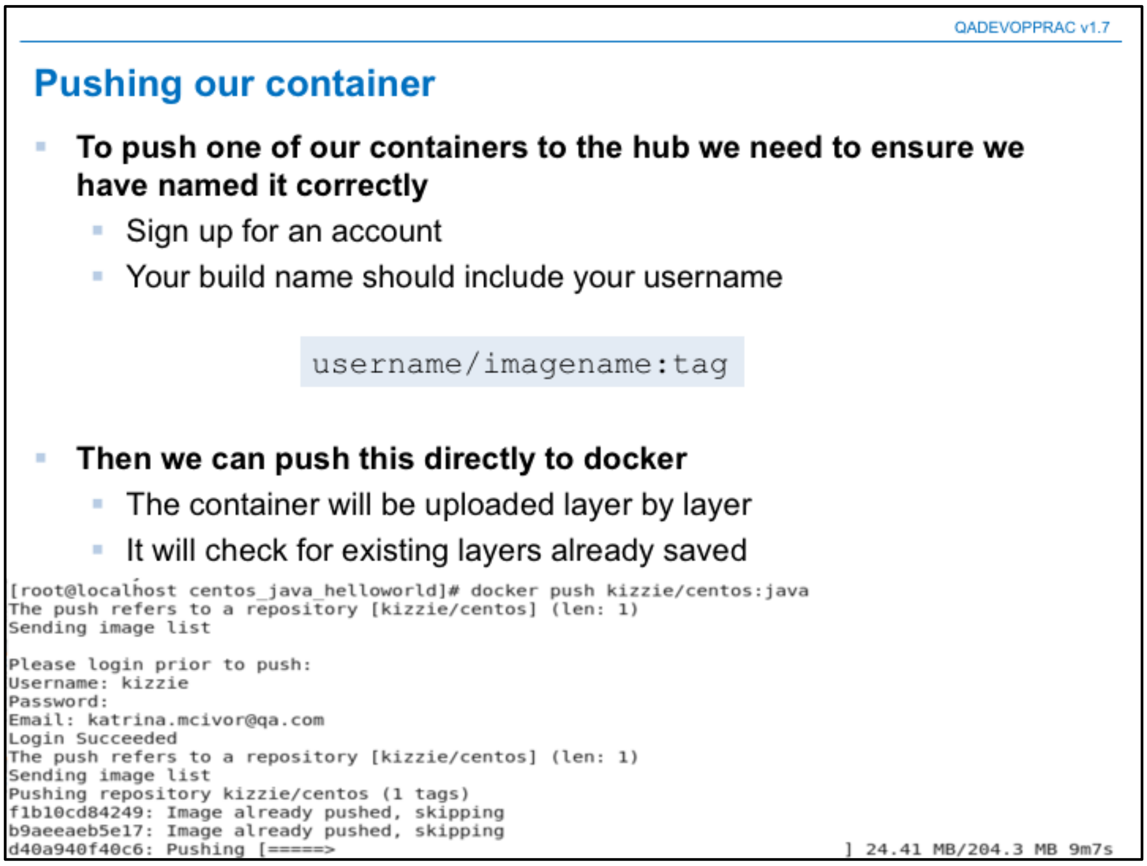


Docker hub - https://hub.docker.com/

- **The Docker hub acts as a repository for all the containers built by users**
  - There are a set of official containers – they do not have a username associated
  - Best practice is to always start by building from these containers
- **You can search the repository and pull down containers**

```
# docker search [term]
# docker pull [imagename]
```

- **Docker hub can even be linked directly to git and other services**
  - Push new code to the repository
  - The hub clones the repo and builds the container based on a Dockerfile present in the root directory



Pushing our container

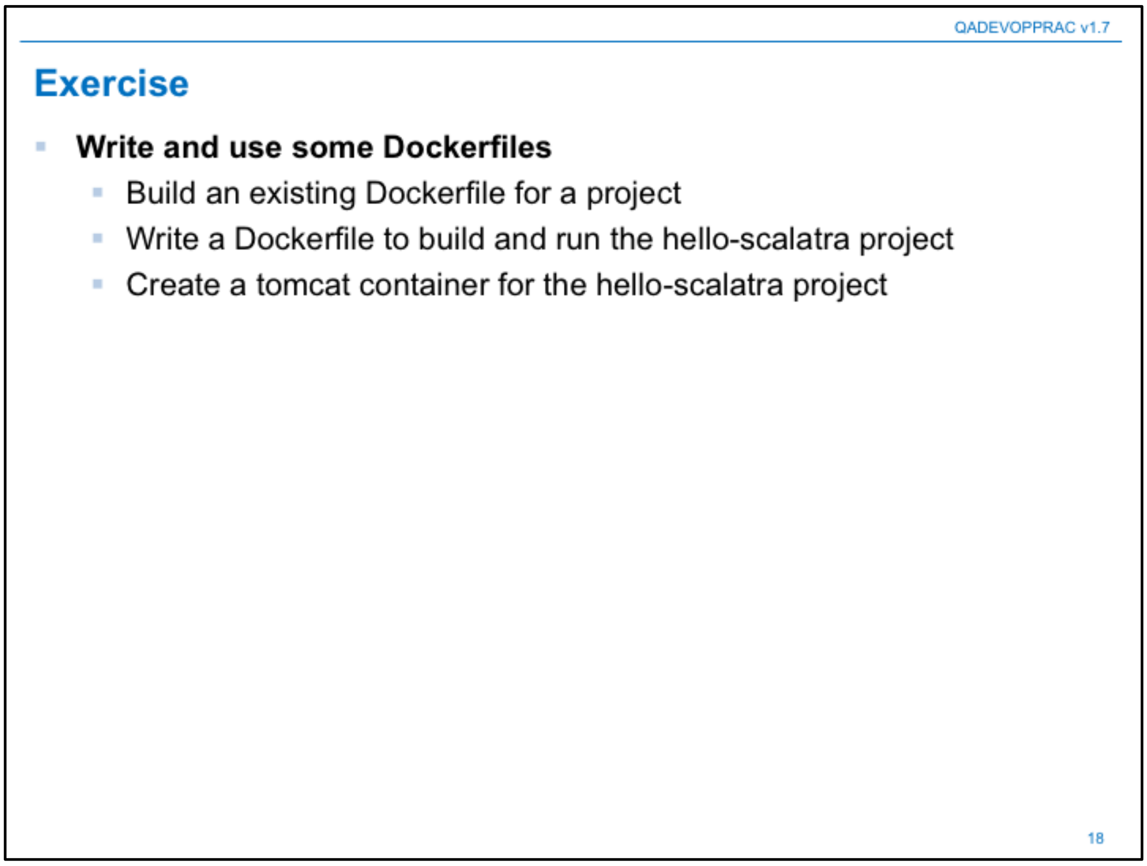
- **To push one of our containers to the hub we need to ensure we have named it correctly**
  - Sign up for an account
  - Your build name should include your username

```
username/imagename:tag
```

- **Then we can push this directly to docker**
  - The container will be uploaded layer by layer
  - It will check for existing layers already saved

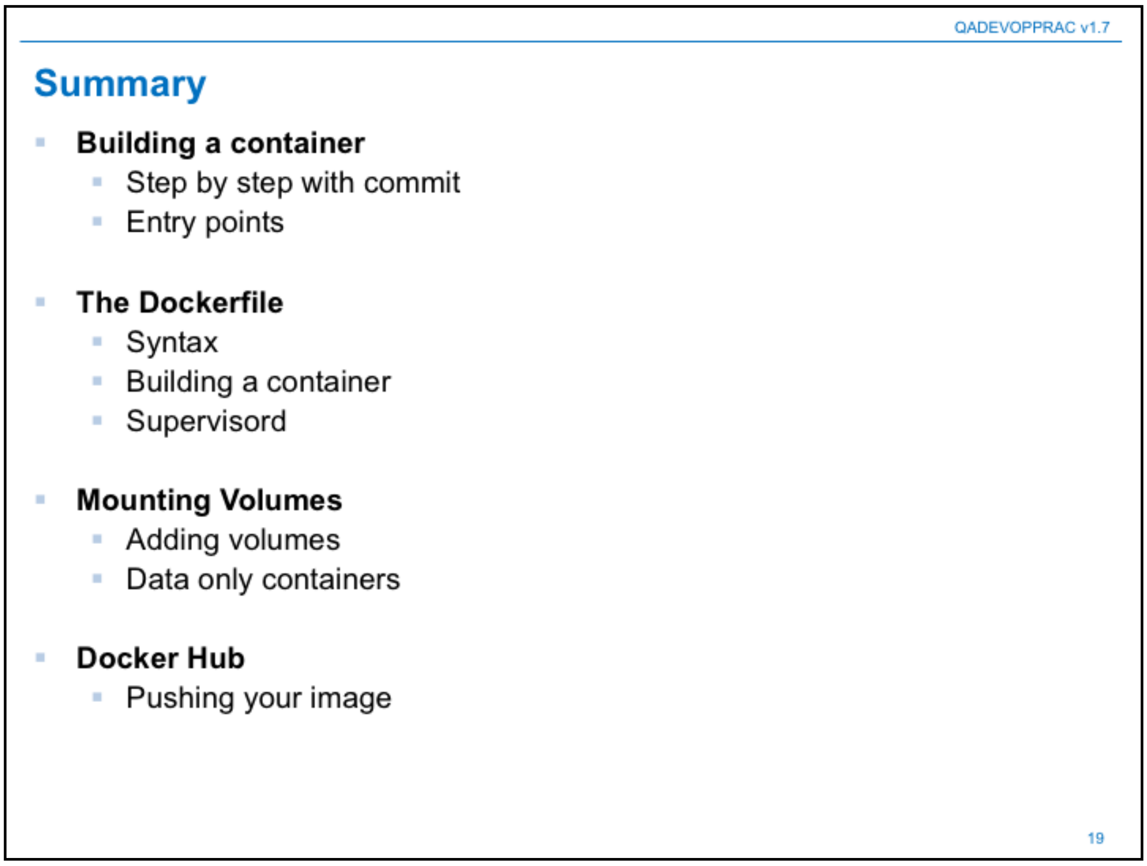
```
[root@localhost centos_java_helloworld]# docker push kizzie/centos:java
The push refers to a repository [kizzie/centos] (len: 1)
Sending image list

Please login prior to push:
Username: kizzie
Password:
Email: katrina.mcivor@qa.com
Login Succeeded
The push refers to a repository [kizzie/centos] (len: 1)
Sending image list
Pushing repository kizzie/centos (1 tags)
f1b10cd84249: Image already pushed, skipping
b9aeeaeb5e17: Image already pushed, skipping
d40a940f40c6: Pushing [=====>                                             ] 24.41 MB/204.3 MB 9m7s
```



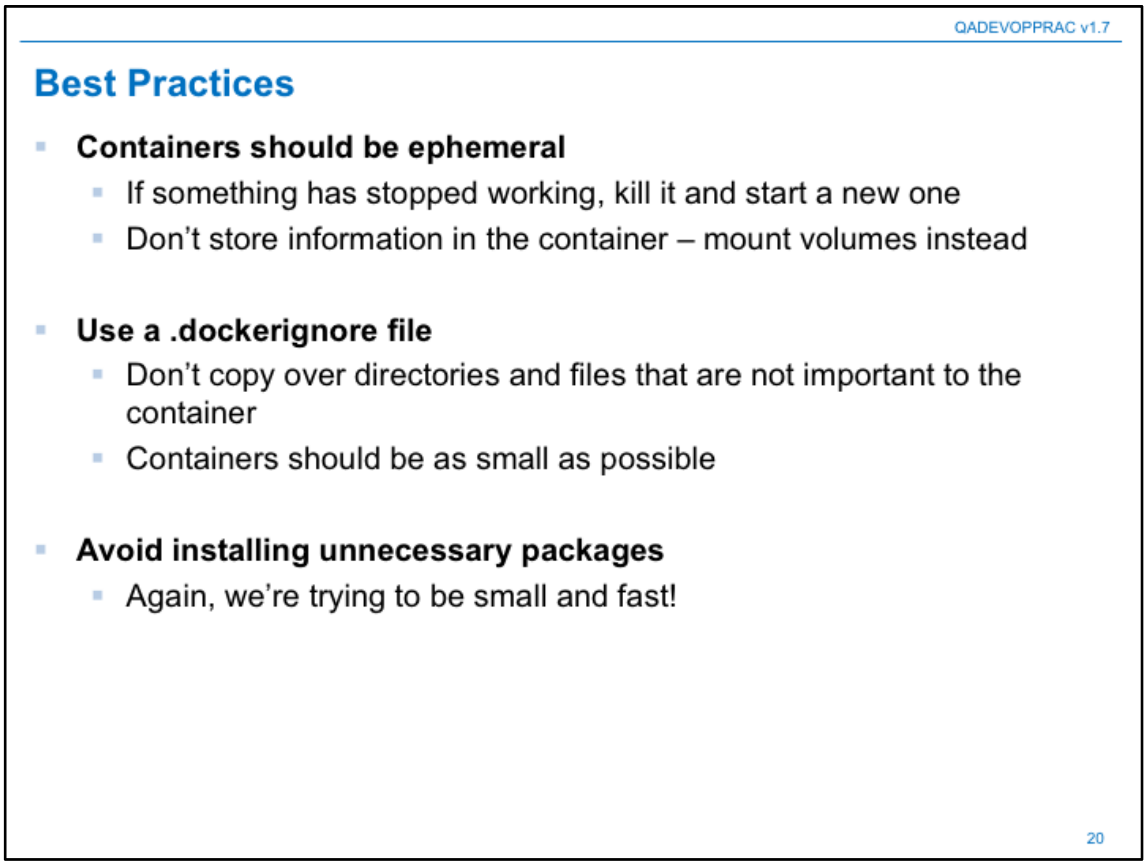
Exercise

- **Write and use some Dockerfiles**
  - Build an existing Dockerfile for a project
  - Write a Dockerfile to build and run the hello-scalatra project
  - Create a tomcat container for the hello-scalatra project



Summary

- **Building a container**
  - Step by step with commit
  - Entry points
- **The Dockerfile**
  - Syntax
  - Building a container
  - Supervisord
- **Mounting Volumes**
  - Adding volumes
  - Data only containers
- **Docker Hub**
  - Pushing your image



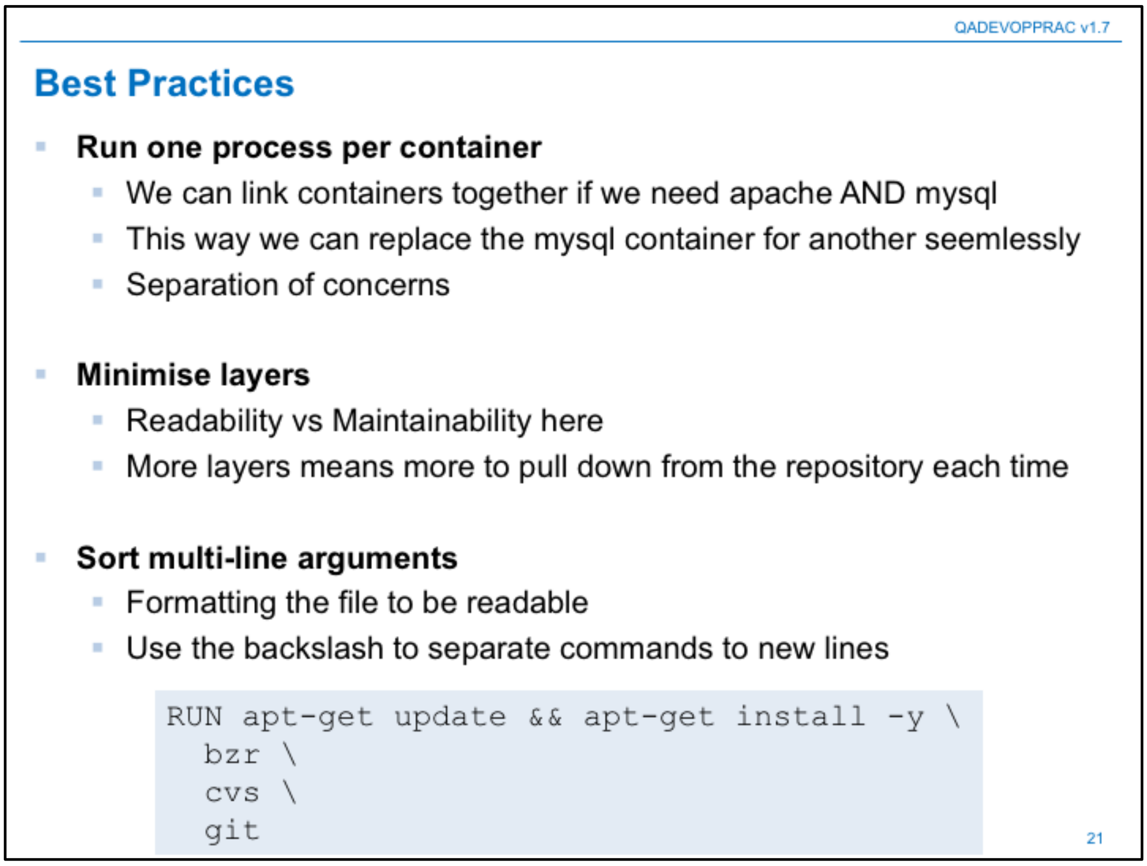
QADEVOPPRAC v1.7

Best Practices

- **Containers should be ephemeral**
  - If something has stopped working, kill it and start a new one
  - Don't store information in the container – mount volumes instead
- **Use a .dockerignore file**
  - Don't copy over directories and files that are not important to the container
  - Containers should be as small as possible
- **Avoid installing unnecessary packages**
  - Again, we're trying to be small and fast!

20

https://docs.docker.com/articles/dockerfile_best-practices/



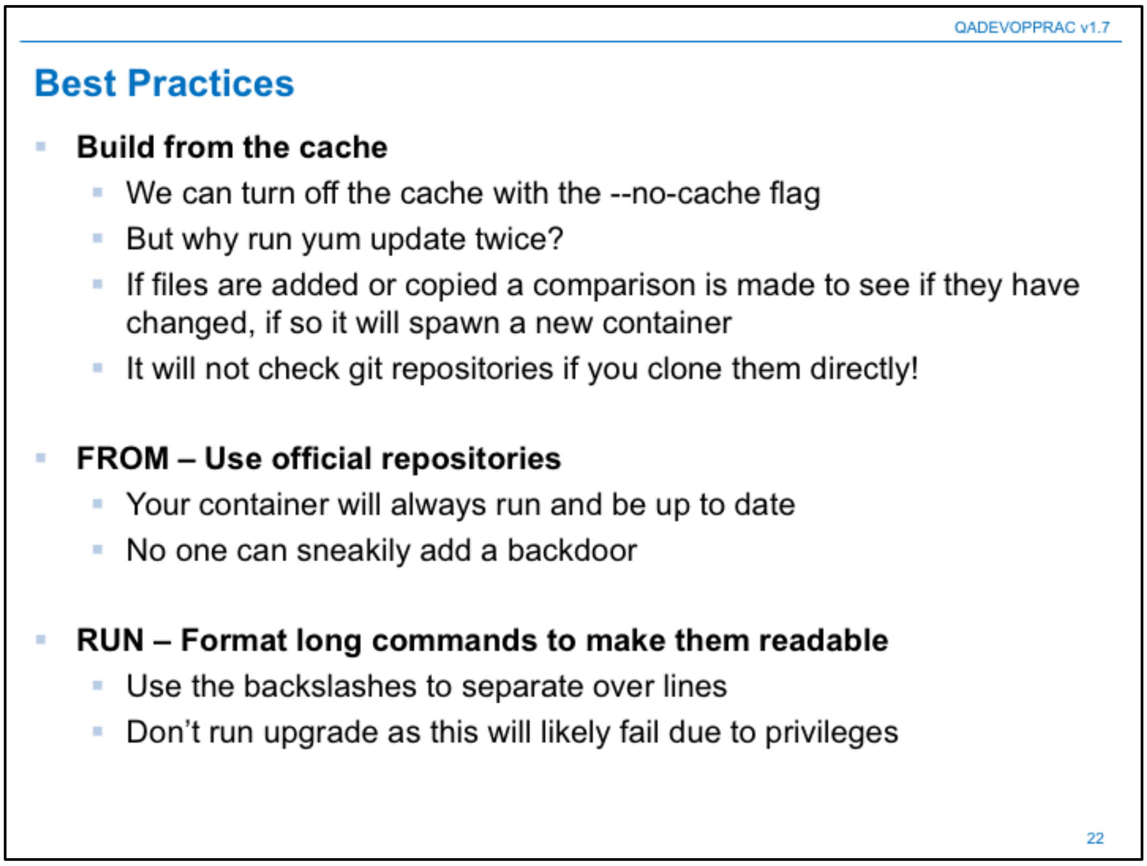
QADEVOPPRAC v1.7

Best Practices

- **Run one process per container**
  - We can link containers together if we need apache AND mysql
  - This way we can replace the mysql container for another seemlessly
  - Separation of concerns
- **Minimise layers**
  - Readability vs Maintainability here
  - More layers means more to pull down from the repository each time
- **Sort multi-line arguments**
  - Formatting the file to be readable
  - Use the backslash to separate commands to new lines

```
RUN apt-get update && apt-get install -y \
  bzr \
  cvs \
  git
```

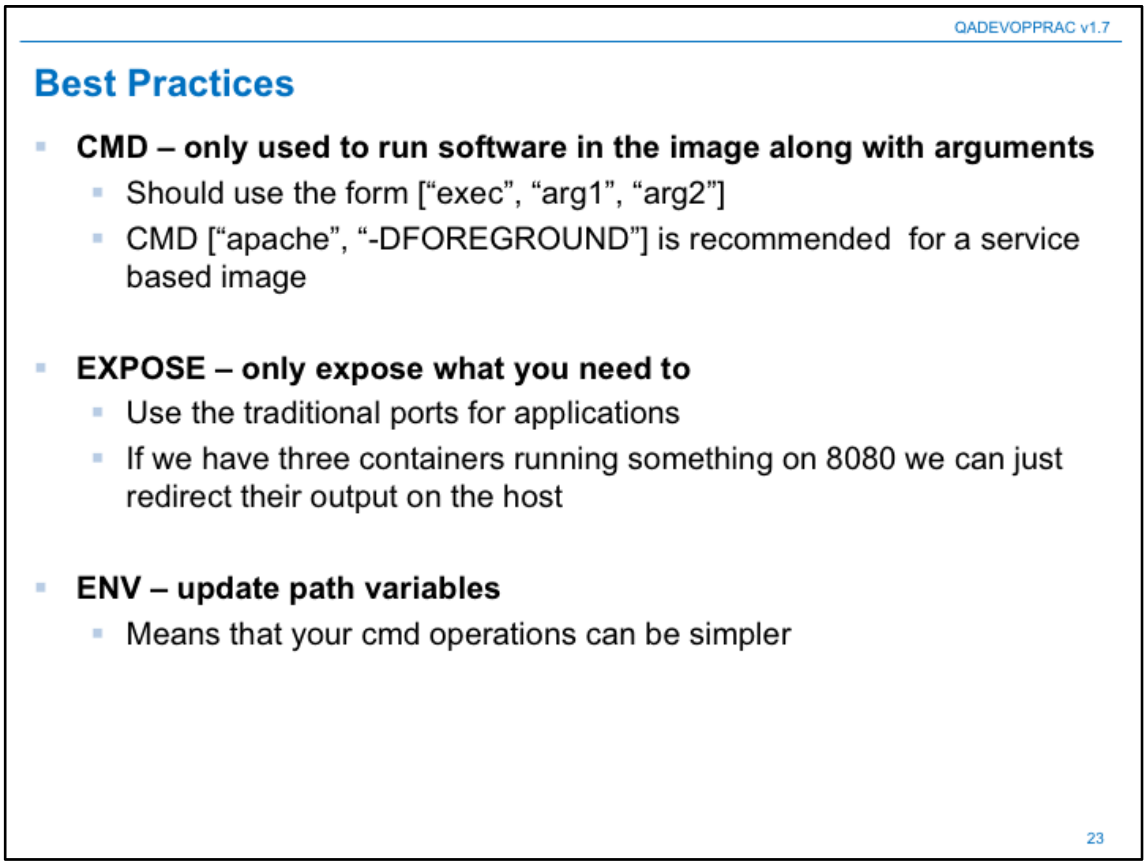
21



Best Practices

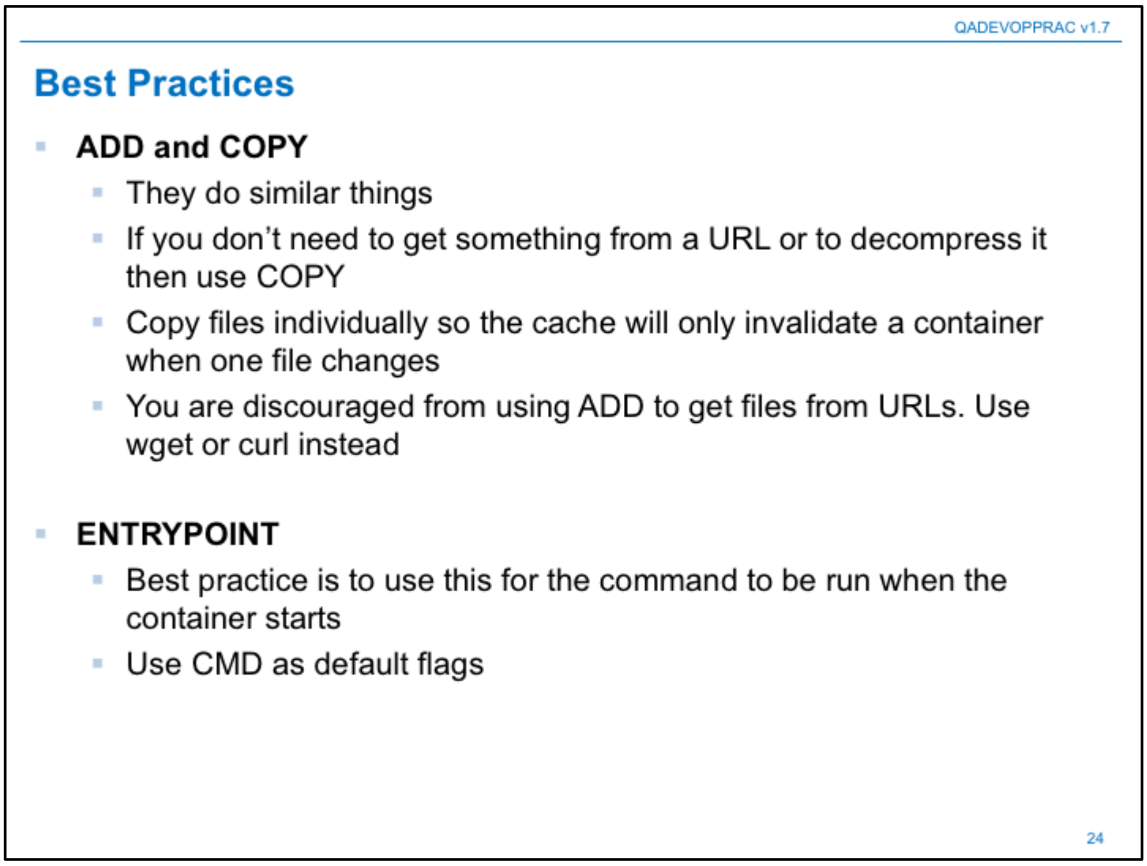
- **Build from the cache**
  - We can turn off the cache with the --no-cache flag
  - But why run yum update twice?
  - If files are added or copied a comparison is made to see if they have changed, if so it will spawn a new container
  - It will not check git repositories if you clone them directly!
- **FROM – Use official repositories**
  - Your container will always run and be up to date
  - No one can sneakily add a backdoor
- **RUN – Format long commands to make them readable**
  - Use the backslashes to separate over lines
  - Don't run upgrade as this will likely fail due to privileges

22



Best Practices

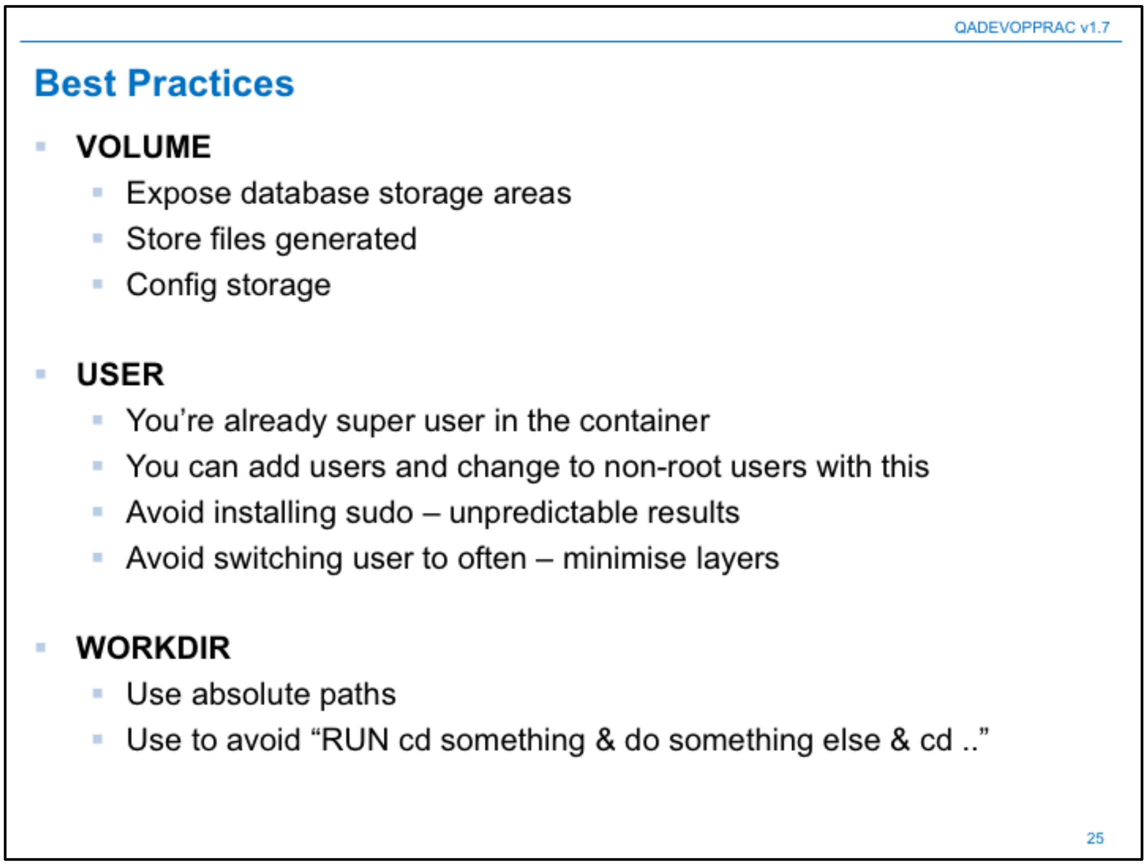
- **CMD – only used to run software in the image along with arguments**
  - Should use the form [“exec”, “arg1”, “arg2”]
  - CMD [“apache”, “-DFOREGROUND”] is recommended for a service based image
- **EXPOSE – only expose what you need to**
  - Use the traditional ports for applications
  - If we have three containers running something on 8080 we can just redirect their output on the host
- **ENV – update path variables**
  - Means that your cmd operations can be simpler



Best Practices

- **ADD and COPY**
  - They do similar things
  - If you don’t need to get something from a URL or to decompress it then use COPY
  - Copy files individually so the cache will only invalidate a container when one file changes
  - You are discouraged from using ADD to get files from URLs. Use wget or curl instead

- **ENTRYPOINT**
  - Best practice is to use this for the command to be run when the container starts
  - Use CMD as default flags



QADEVOPPRAC v1.7

Best Practices

- **VOLUME**
  - Expose database storage areas
  - Store files generated
  - Config storage
- **USER**
  - You’re already super user in the container
  - You can add users and change to non-root users with this
  - Avoid installing sudo – unpredictable results
  - Avoid switching user to often – minimise layers
- **WORKDIR**
  - Use absolute paths
  - Use to avoid “RUN cd something & do something else & cd ..”

25