Container Bootcamp

Docker

Exploring Docker

INNOQ

Hello World Container: BusyBox

```
$ docker run busybox echo hello world
hello world
```

- **size-optimization and limited resources in mind**
- **It is also extremely modular**
- **provides a fairly complete environment for any small or embedded system**
- **We ran a single process and echo'ed `hello world`.**

Ubuntu Container

```
$ docker run -it ubuntu
root@3d027a42be4c:/#
```

- Run a Ubuntu container
- `-it` is short for `-i -t`
  - `-i` connects to containers STDIN
  - `-t` allocates a pseudo-tty (text terminal)

Execute something in our Container

```
root@3d027a42be4c:/# figlet hello
bash: figlet: command not found
```

- Seems that figlet isn’t installed
- Need to install it...

Execute something in our Container

```
root@3d027a42be4c:/# apt-get update
. . .
Fetched 24.3 MB in 5s (4445 kB/s)
Reading package lists... Done
root@3d027a42be4c:/# apt-get install figlet
. . .
Reading state information... Done
The following NEW packages will be
installed:
  figlet
```

- **Seems that figlet isn’t installed**
- **Need to install it**

Execute something in our Container

```
root@3d027a42be4c:/# figlet innoQ
 _                    ___
(_)_ __  _ __   ___  / _ \
| | '_ \| '_ \ / _ \| | | |
| | | | | | | | (_) | |_| |
|_|_| |_|_| |_|\___/ \__\_\
```

- Execute figlet again
- Prints out a beautiful innoQ

Exit a container

```
root@3d027a42be4c:/# exit
$ docker ps -a
CONTAINER ID  IMAGE    COMMAND      CREATED          STATUS
3d027a42be4c  ubuntu  "/bin/bash"  12 minutes ago  Exited (0) About a minute ago
```

- Type `exit` or `^d`
- Container is now in Exited state
- Still exists, but does not use any compute resource anymore

Exercise

1. Run a ubuntu container
2. Install figlet with apt-get
3. Try if figlet works as expected
4. Exit from container and check its status
5. Try to start another ubuntu container and test if figlet is there….

A non-interactive container

```
$ docker run innoq/clock
Mon Oct  3 14:13:42 UTC 2016
Mon Oct  3 14:13:43 UTC 2016
Mon Oct  3 14:13:44 UTC 2016
Mon Oct  3 14:13:45 UTC 2016
. . .
```

- **Will run forever**
- **Stop with `^c`**
- **`innoq/clock` is a user created image**

A non-interactive container

```
$ docker run -d innoq/clock
aa3cc3b06a93795956cbd3bed77e310314e856e68a059c54b05c90d504e13937
```

- `-d` starts container in daemon mode
- There is no output
- Run command returns the container id
- How can I access my container?

A non-interactive container

```
$ docker ps
CONTAINER ID IMAGE          COMMAND                  CREATED              STATUS              NAMES
aa3cc3b06a93 innoq/clock    "/bin/sh -c 'while da"   About a minute ago   Up About a minute   desperate_saha
```

- `docker ps` will show running containers
- It returns:
  - The truncated container id
  - Image name
  - Startup command
  - ...

Show Container Logs

```
$ docker logs desperate_saha
Mon Oct  3 14:27:15 UTC 2016
Mon Oct  3 14:27:16 UTC 2016
Mon Oct  3 14:27:17 UTC 2016
Mon Oct  3 14:27:18 UTC 2016
. . .
```

- **Will show the entire container log (could be too much)**

Show tailed Container Logs

```
$ docker logs --tail 3 desperate_saha
Mon Oct  3 14:27:15 UTC 2016
Mon Oct  3 14:27:16 UTC 2016
Mon Oct  3 14:27:17 UTC 2016
```

- **Will show the tailed container log**
- **3 is the number of lines to be printed**
- **Use `^c` to exit**

Ways to stop a container

- **`docker kill`**
  - Send a `KILL` signal to the container
  - `KILL` forces the container to terminate
- **`docker stop`**
  - Sends a `TERM` signal and after 10 Sec a `KILL` signal
  - More graceful way

Exercise

1. Start several non-interactive clock container instances
2. Try docker ps flags, like
   a. -l (last)
   b. -q (only ID)
   c. -a (all)
3. Display the container logs
4. Provide the prefix of the container id (instead of desperate_saha)
5. Try log --follow (or -f) to follow the log
6. Stop our container (how long does it take?)
7. Kill a container

Concepts Container rely on

Namespaces

- **Isolates Linux processes into their own little system environments**
- **allow aspects of the operating system to be independently modified**
  - pid: Isolated “nested” process trees
  - net: Entirely different set of networking interfaces
  - mnt : Independant mount points per process
  - uts: Used for isolating kernel and version identifiers.
  - ipc: inter-process communication
  - user: allows to have a different view of the uid and gid ranges than the host system
  - . . .

Control Groups

- **is a Linux kernel feature to limit, police and account the resource usage for a set of processes.**
- **Groups are organized hierarchically**
- **Child cgroups inherit some of the attributes of their parents**
- **Associates a set of tasks with a set of parameters for one or more subsystems.**
- **A subsystem is typically a "resource controller" for**
  - Block IO
  - CPU
  - Network Bandwidth
  - Memory
  - . . .

Union file systems

- Are file systems that operate by creating layers
- Layers are conceptually stacked on top of each other
- Very lightweight and fast
- Optimized for disk usage, transfer times, and memory use

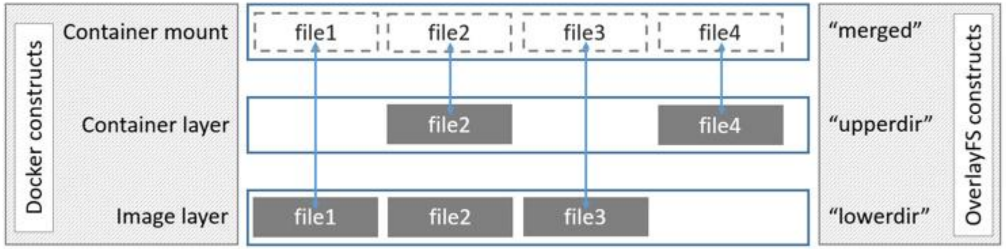
Overlayfs: History

- Newer / modern implementation of a union filesystem
- Since 2014 part of the kernel mainline (since Kernel 3.18)
- Promise to be
  - faster and has a simple architecture than aufs

Overlayfs: How It Works

- Works only with two layers
- multilayered images

  cannot be implemented.



- Higher files obscure lower files.
- Each layer create its own directory
- Hard links are used to reference data with lower directories.
- File location
- /var/lib/docker/overlay/*

Overlayfs2

- Support natively multiple lower OverlayFS layers
- The newest
- With docker 1.12 Overlayfs2 is available works only with kernel 4.0 and later
- Recommended for production, preferred choice

Container Bootcamp

Container Runtime

What is a Container

A container uses the resource isolation features like cgroups, kernel namespaces, aufs (layered file system), etc. of the Linux Kernel

At the end, a running container is a simple Linux process

Advantages:

- lightweight “virtualization”
- Networking (exposing ports)
- Stackable Images (easy to use shipment artefact)
- Ecosystem (Image Registries etc.)

Container Runtime

- Docker
- rkt (CoreOS Inc.)
- LXD (Canonical, based on LXC)
- Flockport (based on LXC)
- LXC (low level infrastructure for container projects)
- Windocks (Docker in Windows)
- OpenVZ
- systemd-nspawn

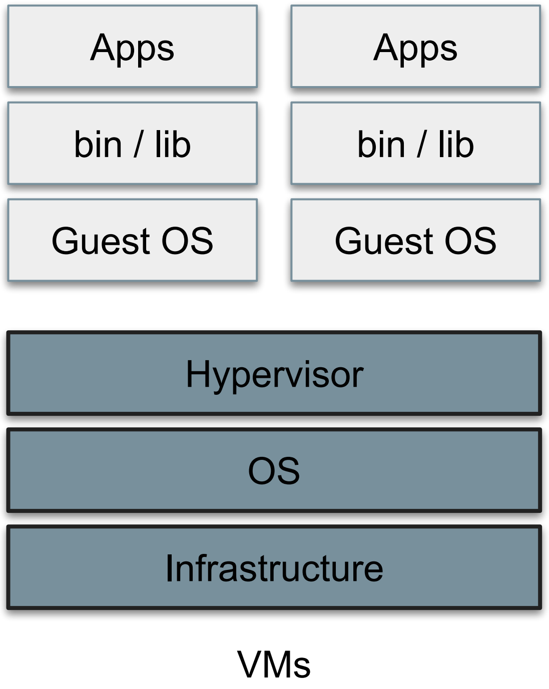
Docker

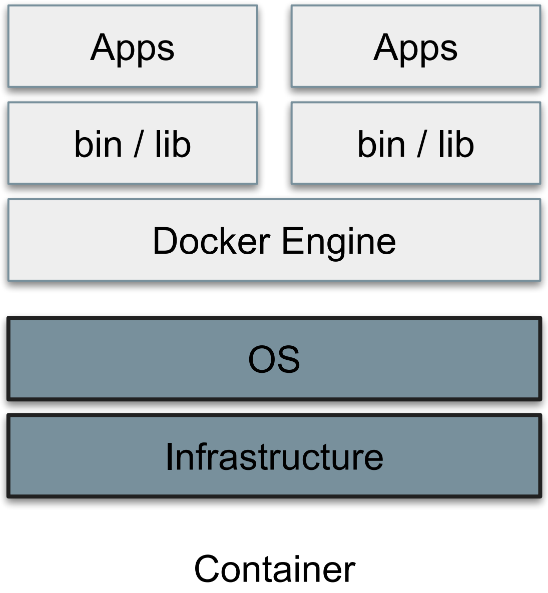
- Standard
- Widely used
- Execution of containers is now handled by containerd (starts the container using runC).

Rkt (CoreOS)

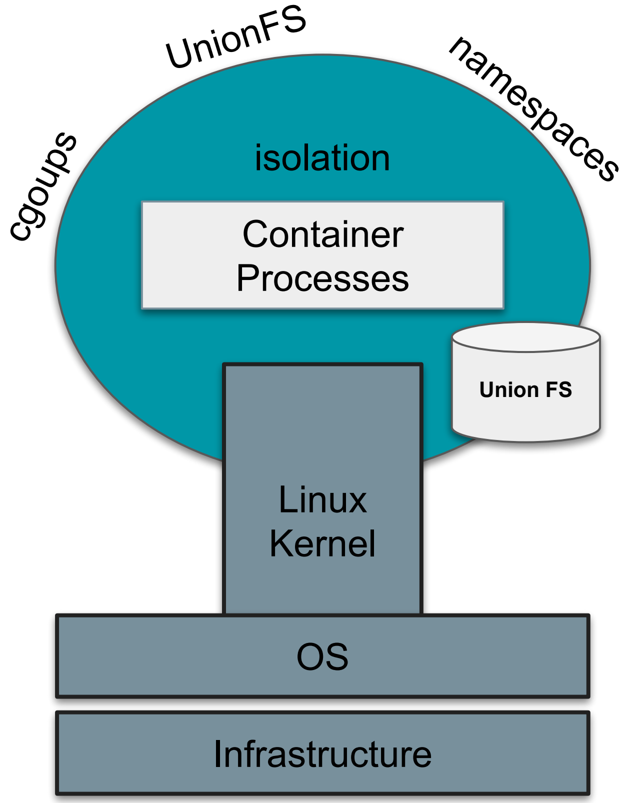
- the only stable container runtime engine that meets the Open Container Initiative (OCI)’s container image specification
- Runs Docker images
  (converts the Docker image on the fly to the standard spec)
- Runs container as real virtualization (optional)
- API, that makes it easier to integrate with scheduling and orchestration platforms
- enhanced with more granular visibility and controls for handling security configurations
- No “init” daemon
  launching containers directly from client commands,

What is Docker

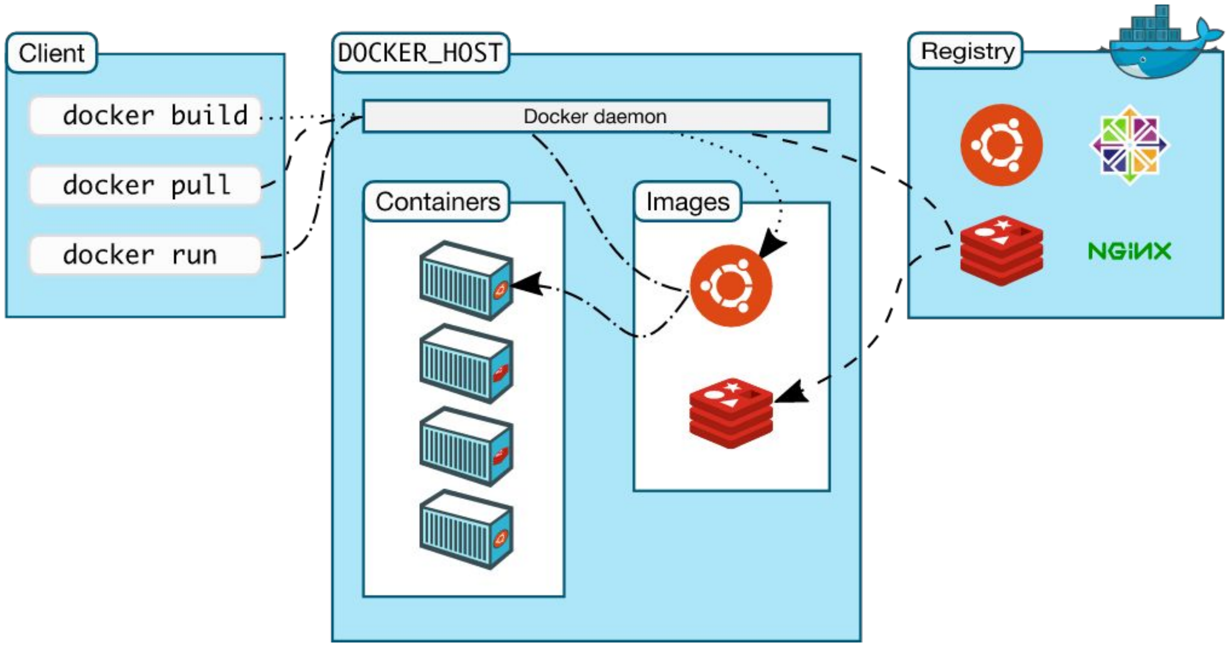




What is Docker



Docker Architecture



https://docs.docker.com/v1.8/introduction/understanding-docker/

Image Size

Official Java Image	->	~ 640 MB
Scala App on custom Java Image	->	~ 350 MB
Java Image on Alpine	->	~ 125 MB
Binary on Alpine Linux	->	~ 10 MB

Docker History example

```
$ docker history my-curator
IMAGE               CREATED             CREATED BY                                      SIZE
7d13b0f92b34        38 seconds ago      /bin/sh -c #(nop) CMD ["curator"]               0 B
f79af82e16a1        38 seconds ago      /bin/sh -c #(nop) ENTRYPOINT ["/docker-entryp   0 B
e08c288cc2cb        39 seconds ago      /bin/sh -c #(nop) ENV ELASTICSEARCH_HOST=elas   0 B
9d667780e3d3        39 seconds ago      /bin/sh -c #(nop) ENV OLDER_THAN_IN_DAYS=20     0 B
44783eb19e90        39 seconds ago      /bin/sh -c #(nop) ENV INTERVAL_IN_HOURS=24      0 B
60af6dafd3e4        39 seconds ago      /bin/sh -c #(nop) COPY file:fcc5b305d95782af1   552 B
c61906248ce2        39 seconds ago      /bin/sh -c pip install elasticsearch-curator=   3.584 MB
ea1f19ee32e2        43 seconds ago      /bin/sh -c groupadd -r curator && useradd -r    330.4 kB
1f6f8180a401        44 seconds ago      /bin/sh -c arch="$(dpkg --print-architecture)   2.267 MB
fdde9931a4e8        50 seconds ago      /bin/sh -c gpg --keyserver ha.pool.sks-keyser   134.5 kB
6b494b5f019c        2 weeks ago         /bin/sh -c #(nop) CMD ["python2"]               0 B
<missing>           2 weeks ago         /bin/sh -c pip install --no-cache-dir virtual   7.898 MB
<missing>           2 weeks ago         /bin/sh -c set -ex  && buildDeps='   tcl-dev    49.37 MB
<missing>           2 weeks ago         /bin/sh -c #(nop) ENV PYTHON_PIP_VERSION=8.1.   0 B
<missing>           2 weeks ago         /bin/sh -c #(nop) ENV PYTHON_VERSION=2.7.12     0 B
<missing>           2 weeks ago         /bin/sh -c #(nop) ENV GPG_KEY=C01E1CAD5EA2C4F   0 B
<missing>           2 weeks ago         /bin/sh -c apt-get update && apt-get install    7.751 MB
<missing>           2 weeks ago         /bin/sh -c #(nop) ENV LANG=C.UTF-8              0 B
<missing>           2 weeks ago         /bin/sh -c #(nop) ENV PATH=/usr/local/bin:/us   0 B
<missing>           2 weeks ago         /bin/sh -c apt-get update && apt-get install    319.1 MB
<missing>           6 weeks ago         /bin/sh -c apt-get update && apt-get install    122.6 MB
<missing>           6 weeks ago         /bin/sh -c apt-get update && apt-get install    44.3 MB
<missing>           6 weeks ago         /bin/sh -c #(nop) CMD ["/bin/bash"]             0 B
<missing>           6 weeks ago         /bin/sh -c #(nop) ADD file:0e0565652aa852f620   125.1 MB
```

Container Bootcamp

Docker Images

INNOQ

Container vs. images

- **An image is a read-only filesystem**
- **A Container**
  - Is a set of processes created in that file system
  - Is working with a read-write layer added to the image (copy on write)

- **OOP analogy**
  - Images are similar to **classes**
  - Layers are similar to **inheritance**
  - Containers are similar to **instances**

Creating an Image from Scratch

- A special empty image called `scratch` allows to build images from scratch
- `docker import`:
  Imports the contents from a tarball to create a filesystem image
- The imported tarball becomes a standalone image
- That new image has a single layer

Usually such kind of tarball is called Root File System, is script generated and forms a Base Linux Distribution image (e.g. Alpine, Ubuntu, etc.)

Other commands

`docker commit`

- Saves all container’s file changes or settings into a new layer
- Creates a new image

`docker build`

- Build an image from a Dockerfile
- A Dockerfile is a repeatable build sequence

Sorts of Images

- **Official Images (e.g. `ubuntu`, `centos`)**
  - Curated set of Docker repositories that are promoted on Docker Hub
  - Provide essential base OS repositories
  - Provide drop-in solutions (data stores, runtimes, …)
  - Exemplify Dockerfile best practices
  - Provide a channel for software vendors to redistribute up-to-date and supported versions of their products.
- **User Images (e.g. `innoq/clock`)**
  - holds images for Docker Hub users and organizations
- **Self-hosted images (e.g. `my.reg.io:2345/ssl-proxy`)**
  - holds images which are not hosted on Docker Hub, but on third party registries

Display the list of pulled images

```
$ docker images
REPOSITORY                        TAG               IMAGE ID       CREATED          SIZE
ubuntu                            latest            c73a085dc378   7 days ago       127.1 MB
python                            2.7               6b494b5f019c   5 weeks ago      676.1 MB
nats                              latest            182553468429   8 weeks ago      7.534 MB
anapsix/alpine-java               8_server-jre      839a695b9726   8 weeks ago      125.2 MB
ches/kafka                        latest            d6179d756438   11 weeks ago     697.3 MB
busybox                           latest            2b8fd9751c4c   3 months ago     1.093 MB
digitalwonderland/zookeeper       latest            deadd4d18859   11 months ago    437.9 MB
digitalwonderland/base            latest            51fae42da917   11 months ago    214.3 MB
anapsix/nyancat                   latest            5bd1209bb5e2   13 months ago    2.769 MB
innoq/clock                       latest            12068b93616f   19 months ago    2.433 MB
```

Pulling an image

```
$ docker pull anapsix/alpine-java:jre7
jre7: Pulling from anapsix/alpine-java
e110a4a17941: Pull complete
8130b3fa5614: Pull complete
Digest: sha256:14afb54bfdb3f0d2c267aac477301c816771d91f8a89a499a589764d378ed498
Status: Downloaded newer image for anapsix/alpine-java:jre7
```

- **Images are made of layers (two in this case)**
- **User `anapsix` provided image `alpine-java` with version tag `jre7`**
- **Tags define image versions, revisions or variants**
- **Not tag refers to `:latest` tag**
- **`:latest` is generally updated often**

Exercise

1. Search for images
   a. With docker search <whatever> (f.e. java)
   b. https://hub.docker.com/
2. Pull images with docker pull
3. Try different tags

Building Images Interactively

Building Images Interactively

- In a former example we used a `ubuntu` image
- Missed and installed `figlet`
- We now want to have an image with `figlet`
- First we'll do it manually with `docker commit`
- Later, we'll use Dockerfile

Building Images Interactively: install

```
$ docker run -it ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
. . .
9f03ce1741bf: Pull complete
Digest: sha256:28d4c5234db8d5a634d5e621c363d900f8f241240ee0a6a978784c978fe9c737
Status: Downloaded newer image for ubuntu:latest
root@071c8c200564:/# apt-get update && apt-get install figlet
. . .
Fetched 24.3 MB in 12s (1897 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  figlet
. . .
root@071c8c200564:/# exit
exit
```

Building Images Interactively: diff

```
$ docker diff boring_carson
C /.wh..wh.plnk
A /.wh..wh.plnk/98.1577025
C /etc
C /etc/alternatives
A /etc/alternatives/figlet
A /etc/alternatives/figlet.6.gz
. . .
```

`docker diff`: Inspect changes on the container's filesystem we’ve made

- A -> Add
- D -> Delete
- C -> Change

Building Images Interactively: commit

```
$ docker commit boring_carson
sha256:696f3dbc163339d7c8e92f970b2f6ff57db5fc9025d39689166c5a5841f486b2
$ docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
<none>                <none>              696f3dbc1633        4 seconds ago       167.1 MB
$ docker run -it 696f3dbc1633
root@392789ad0b3c:/# figlet innoQ again
 _                    ___                    _
(_)_ __  _ __   ___  / _ \    __ _  __ _  __ _(_)_ __
| | '_ \| '_ \ / _ \| | | |  / _` |/ _` |/ _` | | '_ \
| | | | | | | | (_) | |_| | | (_| | (_| | (_| | | | | |
|_|_| |_|_| |_|\___/ \__\_\  \__,_|\__, |\__,_|_|_| |_|
                                   |___/
```

`docker commit`: commits a container’s file changes or settings into a new image

Output is the newly created image ID

Building Images Interactively: tag

```
$ docker images
REPOSITORY             TAG                    IMAGE ID             CREATED              SIZE
<none>                 <none>                 696f3dbc1633         4 seconds ago        167.1 MB
$ docker tag 696f3dbc1633 figlet
$ docker images
REPOSITORY             TAG                    IMAGE ID             CREATED              SIZE
figlet                 latest                 696f3dbc1633         7 minutes ago        167.1 MB
```

`docker tag`: tags an image into a repository

Exercise

1. Run an interactive ubuntu container
2. Install figlet
3. Test if it works as expected
4. Stop the container
5. Check what has changed and commit the changes
6. Tag the new image and run a new container with this image
7. Test if figlet is installed and working

Container Bootcamp

Dockerfile

Dockerfile

- **Instructions . . .**
  - tell docker how to construct an image
  - are commands a user could call on the command line to assemble an image

- **The `docker build` command builds an image from a Dockerfile**

Our first Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
```

- **This is our `figlet` example as Dockerfile**
- **The `FROM` instruction sets the Base Image for subsequent instructions**
- **The `RUN` instruction will execute a commands in a new layer on top of the current image and commit the results**
    - `RUN` **needs to be non interactive, you cannot answer apt-get questions (use `-y` flag)**

Build the Image

```
$ docker build -t figlet .
```

- `docker build` builds Docker images from a Dockerfile and a "context"
- `-t` indicates name and optionally a tag in the '`name:tag`' format
- `.` is the location of the build context

Build the Image

```
$ docker build -t figlet .
Sending build context to Docker daemon 2.048
kB
Step 1 : FROM ubuntu
 ---> c73a085dc378
Step 2 : RUN apt-get update
 ---> Running in 5de7bd1b6892
. . .
 ---> 3775c7adf3df
Removing intermediate container 5de7bd1b6892
Step 3 : RUN apt-get install figlet
 ---> Running in 930f62dccb73
. . .
 ---> ab7f493af20e
Removing intermediate container 930f62dccb73
Successfully built ab7f493af20e
```

context is send (as an archive) to docker daemon

Ubuntu image id

R/W image id

committed image id

final image id that gets named figlet

Image history

```
$ docker history figlet
IMAGE           CREATED           CREATED BY                                          SIZE
ab7f493af20e    12 minutes ago    /bin/sh -c apt-get install figlet                   1.008 MB
3775c7adf3df    12 minutes ago    /bin/sh -c apt-get update                           39.02 MB
c73a085dc378    8 days ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]                0 B
<missing>       8 days ago        /bin/sh -c mkdir -p /run/systemd && echo 'doc       7 B
<missing>       8 days ago        /bin/sh -c sed -i 's/^#\s*\(deb.*universe\)$/       1.895 kB
<missing>       8 days ago        /bin/sh -c rm -rf /var/lib/apt/lists/*              0 B
<missing>       8 days ago        /bin/sh -c set -xe   && echo '#!/bin/sh' > /u       745 B
<missing>       8 days ago        /bin/sh -c #(nop) ADD file:cd937b840fff16e04e       127.1 MB
```

`c73a085dc378` -> ubuntu base image

`3775c7adf3df` -> 1st `RUN`

`ab7f493af20e` -> 2nd `RUN`

Build the Image again

```
$ docker build -t figlet .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM ubuntu
 ---> c73a085dc378
Step 2 : RUN apt-get update
 ---> Using cache
 ---> 3775c7adf3df
Step 3 : RUN apt-get install figlet
 ---> Using cache
 ---> ab7f493af20e
Successfully built ab7f493af20e
```

?

Build Cache

Why is the second build faster?

- **existing image in cache?**
- **instruction string is cache key**
- **For the `ADD` and `COPY`, checksum of the file(s) is used**

- **Things like `RUN apt-get update` is not re-executed!**

If you do not want to use the cache at all, use the `--no-cache` option

RUN Instruction

RUN has 2 forms:

- `RUN <command>`
  (shell form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)
- `RUN ["executable", "param1", "param2"]`
  (exec form)

Exercise

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
```

1. *Create a Dockerfile with the above content*
2. *Build the image with tag figlet*
3. *Try to understand the image history*
4. *Build the image again (check the "using cache" entries)*
5. *Try the no-cache option and build the figlet image again*
6. *Change `apt-get install figlet` to exec form, build image*
7. *Check docker history, what has changed?*

CMD Instruction

Allows us to set the default command to run in a container

- `CMD ["executable","param1","param2"]`
  (exec form, this is the preferred form)
- `CMD ["param1","param2"]`
  (as default parameters to ENTRYPOINT <- see later)
- `CMD command param1 param2`
  (shell form, command wrapped by `/bin/sh -c`)

Only last CMD will take effect

Exercise: CMD Instruction

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
CMD figlet hello innoQ
```

1. *Use the above Dockerfile to create a new figlet image*
2. *Run image figlet*
3. *What happens if you run `docker run figlet figlet Hello` ?*
4. *What happens if you run `docker run -it figlet bash` ?*

ENTRYPOINT Instruction

An `ENTRYPOINT` allows you to configure a container that will run as an executable. It allows arguments to be passed to the entry point.

- `ENTRYPOINT ["executable", "param1", "param2"]`
  (exec form, preferred)
- `ENTRYPOINT command param1 param2`
  (shell form, command wrapped by `/bin/sh -c`)

ENTRYPOINT Instruction

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
ENTRYPOINT ["figlet", "-f", "script"]
```

- **ENTRYPOINT defines a base command (and its parameters) for the container**
- **Command line arguments are appended to those parameters**

Only last ENTRYPOINT will take effect

CMD and ENTRYPOINT Instruction

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
ENTRYPOINT ["figlet", "-f", "script"]
CMD ["hello innoQ"]
```

- **ENTRYPOINT is the base command**
- **CMD are the default parameter**

CMD and ENTRYPOINT Instruction

remember:

- **Shell form replaces environment parameter**
  Exec form doesn’t
- **Shell form does not deliver signals**
  Exec form does
- **To ensure that docker stop will signal any long running executable correctly use `exec`, f.e.:**

```
ENTRYPOINT exec top -b
```

Exercise: CMD and ENTRYPOINT

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
ENTRYPOINT ["figlet", "-f", "script"]
```

1. *Build the figlet image with the ENTRYPOINT instruction*
2. *Run image with*

   `docker run figlet Image with ENTRYPOINT`
3. *What happens (if we use the ENTRYPOINT Shell Form)?*
4. *Append `CMD ["hello innoQ"]` to Dockerfile and rebuild figlet image*
5. *Execute container with `docker run figlet`*
6. *What happens with `docker run figlet HiHo` ?*

BTW: Some useful cleanup commands

- `docker ps -a | grep ago | awk '{print $1}' | xargs docker rm -f`
- `docker ps -a | grep registry.com | awk '{print $1}' | xargs docker rm -f`
- `docker images | grep "<none>" | awk '{print $3}' | xargs docker rmi`
- `docker images | grep deployment01 | awk '{print $1 ":" $2}' | xargs docker rmi`
- `docker run --rm . . .`
- `docker rm $(docker ps -a -q)`
- `docker rmi $(docker images -q)`

COPY Instruction

The `COPY` instruction copies new files or directories from `<src>` and adds them to the filesystem of the container at the path `<dest>`

- `COPY <src>... <dest>`
- `COPY ["<src>",... "<dest>"]`
  (this form is required for paths containing whitespace)

```
COPY hom* /mydir/          # adds all files starting with "hom"
COPY hom?.txt /mydir/      # ? is replaced with any single character, e.g., "home.txt"
COPY test relativeDir/     # adds "test" to `WORKDIR`/relativeDir/
COPY test /absoluteDir/    # adds "test" to /absoluteDir/
```

Exercise: COPY

1. *Create a File `hello.c` with the following content:*

```
int main () {
  puts("Hello, innoQ!");
}
```

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y build-essential
COPY hello.c /
RUN make hello
CMD /hello
```

2. *Build the image with `build -t hello .`*
   and rebuild it (wo/w modification of `hello.c`)
3. *Run the image (`docker run --rm hello`)*

Container Bootcamp

Advanced Dockerfile

INNOQ

Dockerfile Usage

- Instructions are executed in order (except CMD and ENTRYPOINT)
- Each instruction adds a new image layer
- There is an instruction cache
- Only the last CMD / ENTRYPOINT is used
- There are best practices (-> later)

FROM Again

```
FROM <image>
FROM <registry/image>:<tag>
FROM <image>@<digest>
```

- `FROM` must be the first non-comment instruction in the Dockerfile.
- `FROM` can appear multiple times within a single Dockerfile in order to create multiple images (`build` fails as soon as an instructions fails)
- The `tag` or `digest` values are optional. If you omit either of them, the builder assumes a latest by default.

RUN Again

- **Used during image creation**
  - Install libraries, packages etc.
- **Do not start a process in a container with `RUN`**

```
RUN apt-get update && apt-get install -y \
    aufs-tools \
    . . . \
    ruby1.9.1-dev \
    s3cmd=1.1.* \
 && rm -rf /var/lib/apt/lists/*
```

- **One `RUN` instruction (using `&&` and `\`) creates one image layer**
- **removing the apt cache `/var/lib/apt/lists` helps keep the image size down**

EXPOSE

- **Indicates the ports on which the container will listen for connections**
- **EXPOSE does not make the ports of the container accessible to the host**
  - Use the `-p` flag to publish a range of ports (even if they are not EXPOSEd)

    ```
    docker run -p 1234-1236:1234/tcp ...
    ```
  - Use the `-P` flag to publish all of the exposed ports
    Docker binds each exposed port to a random port on the host

    ```
    docker run -P ...
    ```

```
EXPOSE 8080
EXPOSE 80 443
EXPOSE 53/tcp 53/udp
```

ADD

```
ADD hom* /mydir/
ADD hom?.txt /mydir/
ADD http://www.example.com/webapp.jar /opt/
ADD ./assets.zip /var/www/htdocs/assets/
```

Almost like COPY

- `ADD <src>... <dest>`
- `ADD ["<src>",... "<dest>"]`
  (this form is required for paths containing whitespace)

Copies new files, directories or remote file URLs from `<src>` and adds them to the filesystem of the container at the path `<dest>`.

If `<src>` is a local tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory

Resources from remote URLs are not decompressed.

WORKDIR

Sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY` and `ADD` instructions that follow it in the Dockerfile.

It can be used multiple times in the one Dockerfile

```
WORKDIR /a
WORKDIR b
WORKDIR c
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
```

ENV

Sets the environment variable `<key>` to the value `<value>`

- **`ENV <key> <value>`**

  sets a single variable to a value

- **`ENV <key>=<value>` …**

  allows multiple variables to be set at one time

Environment can be changed with docker `run -e <key>=<value>`

```
ENV myName="John Doe" myDog=Rex\ The\ Dog \
    myCat=fluffy
ENV myDog Rex The Dog
```

USER

Sets the user `name` or `UID` to use when running the image and for any `RUN`, `CMD` and `ENTRYPOINT` instructions

Can be used multiple times

```
USER daemon
```

Dockerfile Examples

Dockerfile Examples

```
FROM       ubuntu
MAINTAINER Victor Vieux <victor@docker.com>

LABEL Description="start FBar" Vendor="ACME Products" Version="1.0"
RUN apt-get update && apt-get install -y inotify-tools nginx
apache2
```

Dockerfile Examples

```
FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and firefox
RUN apt-get update && apt-get install -y x11vnc xvfb firefox
RUN mkdir ~/.vnc
# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the trick)
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]
```

Best practices

Best practices for writing Dockerfiles

- **Containers should be ephemeral**
  - Can be stopped and destroyed and a new one built and put in place with an absolute minimum of set-up and configuration
- **Avoid installing unnecessary packages**
  - No texteditor in a DB image
  - clean caches and unneeded directories
- **Only one process per container**
  - Decouple applications into multiple containers
- **Minimize the number of layers**
  - Combine multiple similar commands into one by using `&&` to continue commands and `\` to wrap lines

Best practices for writing Dockerfiles

- `ADD` and `COPY`
  - `COPY` is preferred, because it's more transparent than ADD
- `ADD`
  - Because image size matters, using `ADD` to fetch packages from remote URLs is strongly discouraged; use `curl` or `wget` instead

```
ADD http://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C
/usr/src/things
RUN make -C /usr/src/things all
```

```
RUN mkdir -p /usr/src/things \
    && curl -SL http://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

Best practices for writing Dockerfiles

- `COPY`
  - COPY dependency lists (package.json, requirements.txt, etc.) by themselves to avoid reinstalling unchanged dependencies every time.

```
FROM python
COPY . /src/
WORKDIR /src
RUN pip install -qr requirements.txt
EXPOSE 5000
CMD ["python", "app.py"]
```

🚫

```
FROM python
COPY ./requirements.txt /tmp/requirements.txt
RUN pip install -qr /tmp/requirements.txt
COPY . /src/
WORKDIR /src
EXPOSE 5000
CMD ["python", "app.py"]
```