Containers 101/a – UCREL NLP Summer School Edition

UCREL NLP Summer School - 2024

John Vidler and Nouran Khallaf

Containers 101/a: Overview

- Why do we care about containers?
  - Some example workflows
- What is Docker? Podman? Containers?
- (Simplified) Anatomy of a Container
- Some examples from the 'real world'…
- Your turn ☺
- Building our own images
  - Dockerfile basics with Python
- Building multi-container images
  - Podman Compose

If you want to view the original versions of this talk and its follow-up visit:

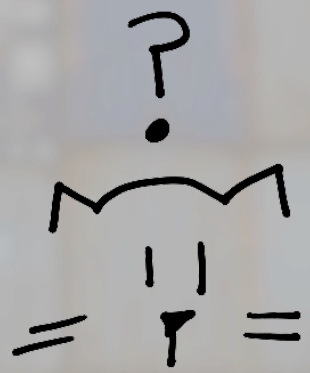
- **Docker 101 -** https://johnvidler.co.uk/blog/docker-101/
- **Docker 102 -** https://johnvidler.co.uk/blog/docker-102/

Moby Dock, the Docker Mascot
Credit: Docker Website

Also cats!

.. lots of cats!

Why do we care about Containers?



Why do we care about Containers?

(…as **developers**)

- Develop once – run everywhere!
  - Desktops, laptops, servers, clusters, etc…
- Isolate the host system from our application requirements
  - Libraries, configurations and runtime environments
- So overall, fewer support headaches (hopefully!)

Why do we care about Containers?

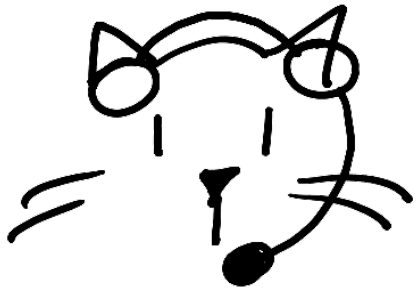
(...as **scientists**)

- Reproducibility

  We have heard in this summer school how reproducibility and verification of results is increasingly important!

- Resource Sharing

  Operating on shared platforms allows us to minimise idle time on expensive compute resources (financially, ecologically)



Some Example Workflows

Some Example Workflows

Running the MEGA-NLP database…

1. Go to the website
2. Download the program or the installer for the program
3. Find out it needs CoolLib24 but you have CoolLib26 and need it for a dozen other programs.
4. Fine… install it on your old laptop then..
5. … but it runs Linux. Ok, so download…

Some Example Workflows

Running the MEGA-NLP database… On a Virtual Machine

1. Ok, so get VMWare, or VirtualBox, or another virtual machine service set up

   (Contact your systems support team for a new managed VM?)

2. Get Linux installed on the VM … ok, we'll wait.

Some Example Workflows

Running the MEGA-NLP database… As a container application

While he’s off doing that, the Podman/Docker/Container way

1. Start Podman Desktop (or Docker Desktop)
2. Run the MEGA-NLP image in a new container
3. Hey, it’s working!

- … we should probably check on the VM cat.

Some Example Workflows

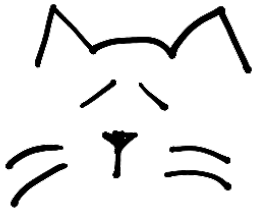
- This obviously a potted example situation, but it really should be this simple with containerised applications
  - Complex programs sometimes need some more complicated configuration to begin with
  - Graphical interfaces for apps are possible, but extremely awkward, and not really the intended use for containers
    - Web interfaces are fine though!

Software Distribution Woes

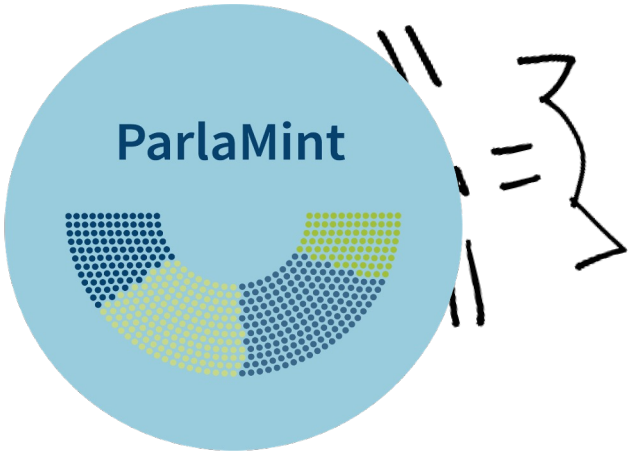
Fundamentally software distribution and deployment is a pain

- …and can be a huge time sink, especially if you find a lot of people start using it!

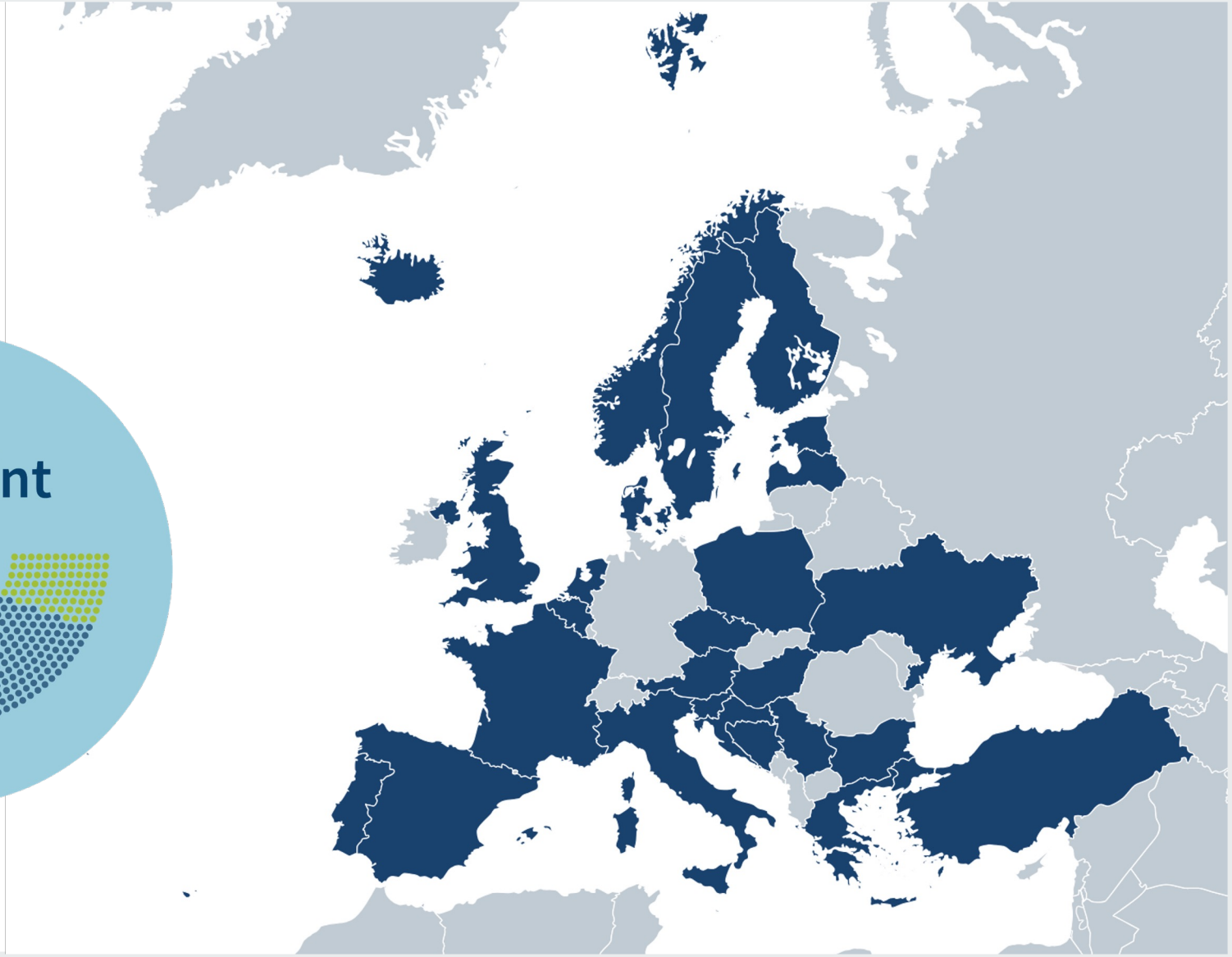
- Using containers simplifies this by moving the host configuration, hardware and runtime requirements to your control without requiring a heavyweight virtual machine around the application.



A Real-World Example!

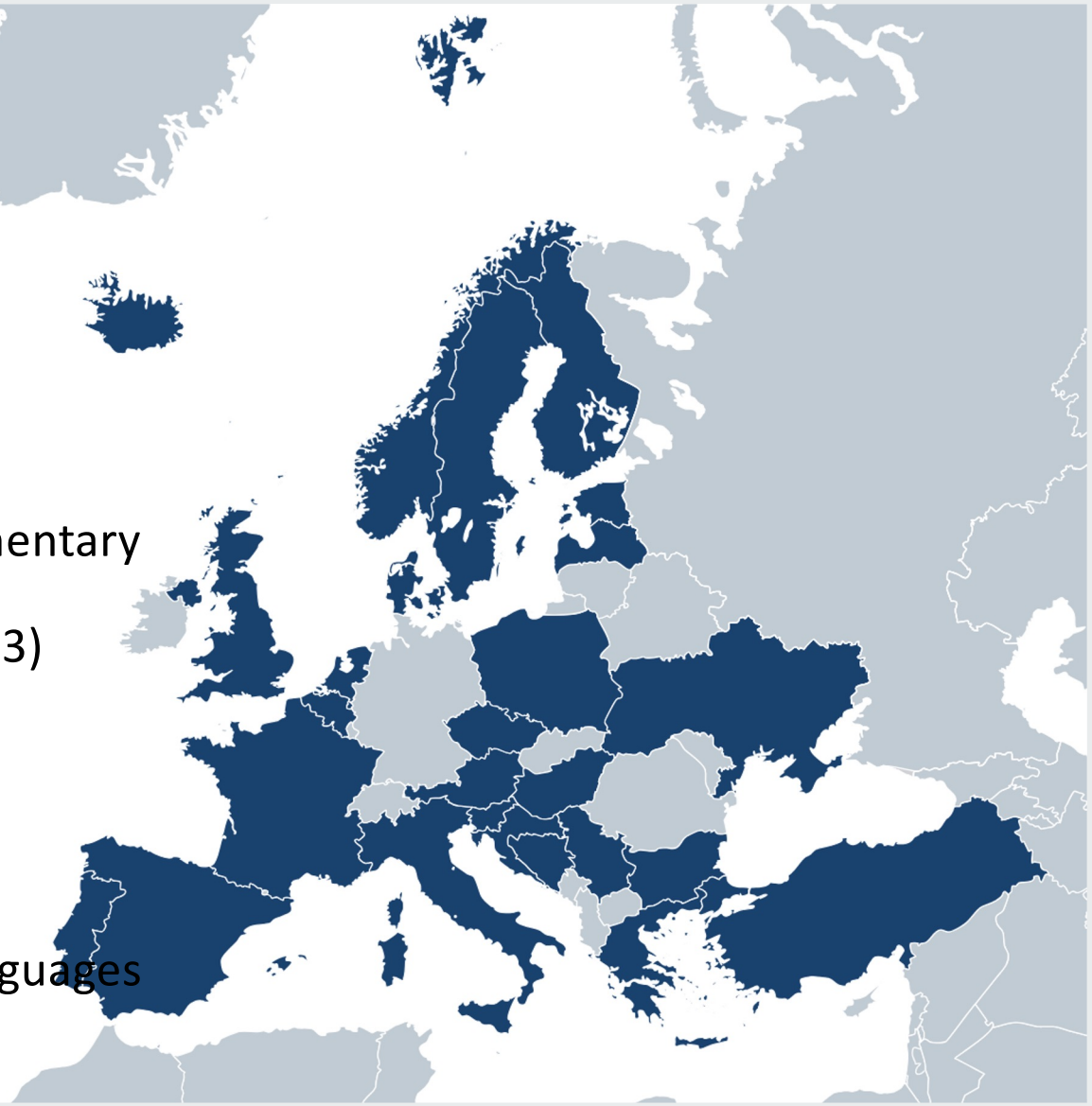


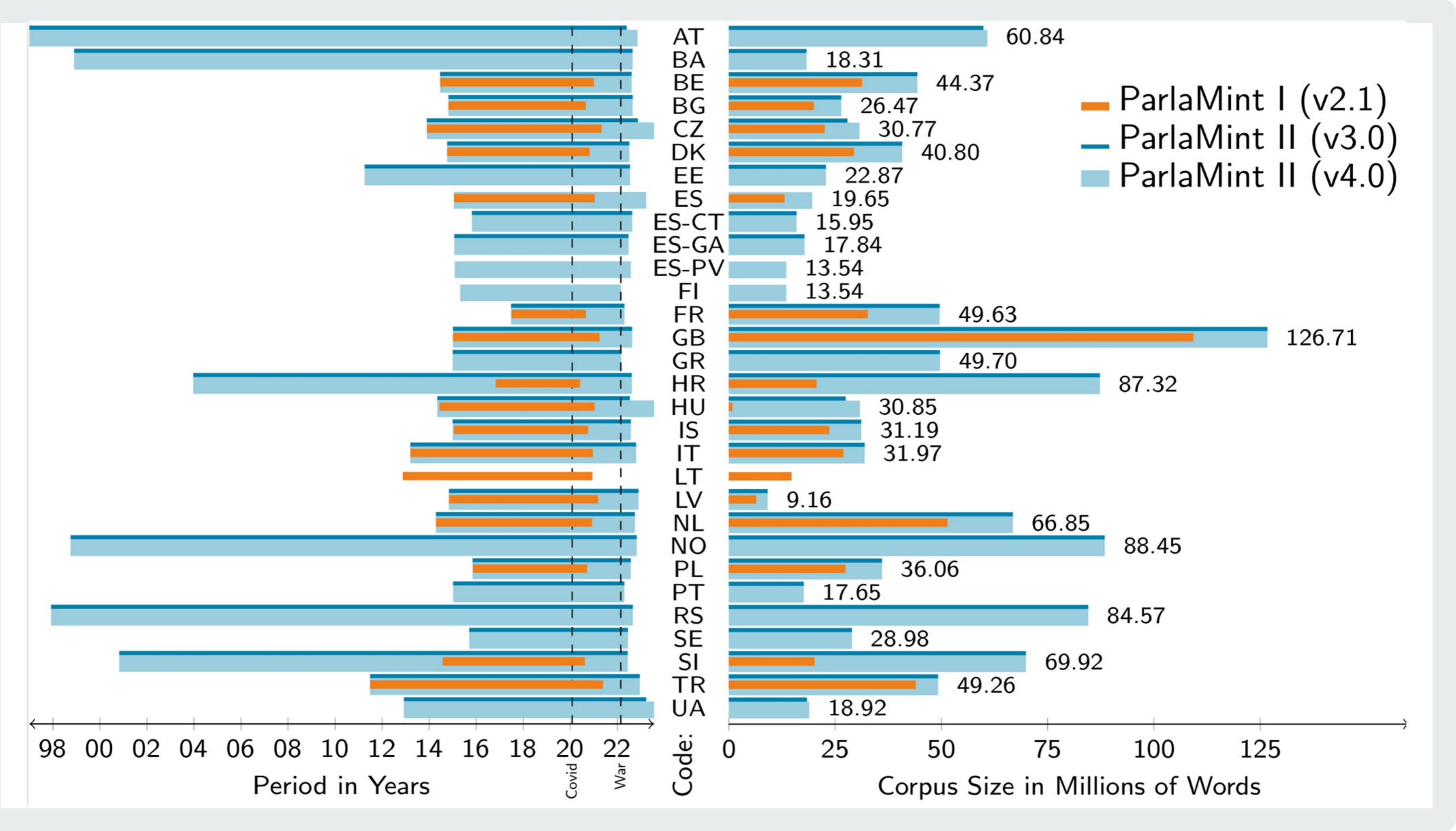
ParlaMint



Introducing ParlaMint …

- CLARIN flagship project (https://www.clarin.eu/parlamint)
- Creation of comparable, uniformly annotated, CC-BY corpora of parliamentary debates across Europe
- Two project stages (2020-21, 2022-23)
- V4.0 released October 2023: http://hdl.handle.net/11356/1859
- English MT annotated version released November 2023: http://hdl.handle.net/11356/1864
- 7.8M utterances; 1.2B words; 29 languages




ParlaMint I (v2.1)
ParlaMint II (v3.0)
ParlaMint II (v4.0)
AT 60.84
BA 18.31
BE 44.37
BG 26.47
CZ 30.77
DK 40.80
EE 22.87
ES 19.65
ES-CT 15.95
ES-GA 17.84
ES-PV 13.54
FI 13.54
FR 49.63
GB 126.71
GR 49.70
HR 87.32
HU 30.85
IS 31.19
IT 31.97
LT
LV 9.16
NL 66.85
NO 88.45
PL 36.06
PT 17.65
RS 84.57
SE 28.98
SI 69.92
TR 49.26
UA 18.92
98 00 02 04 06 08 10 12 14 16 18 20 22
Period in Years
Covid
War
Code:
0 25 50 75 100 125
Corpus Size in Millions of Words

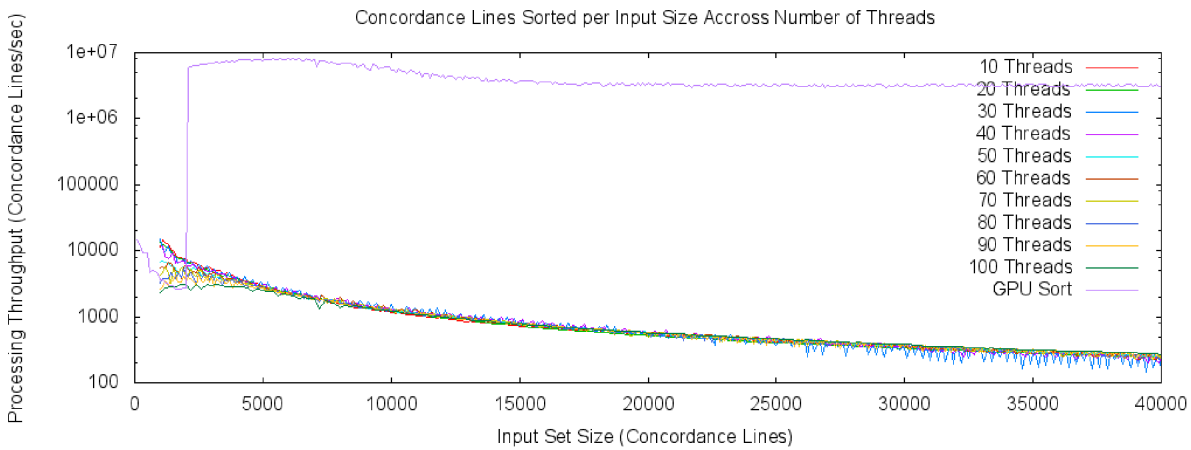
Lancaster's roles …

1. Collect, annotate and convert corpora from UK Hansard
   a) Via Hansard API and Parliamentary Open Data API (see details in LRE journal paper https://doi.org/10.1007/s10579-021-09574-0)
   b) Matt Coole & Amanda Nwadukwe
2. Semantically tag all the data (previously machine translated to English using Helsinki's Opus-MT system)
   a) With thanks to Daisy Lal for the CONLLU format Python tagging script
   b) PyMUSAS (https://pypi.org/project/pymusas/) estimated tagging time on one UCREL VM ≅ 18 days (not including UK corpus)
   c) John to the rescue ☺

Data Dependencies

- If dataset ‘A’ never interacts with dataset ‘B’ we can process them independently
  - We can often cheat this too (kinda!)

- In general, loops over large collections are good candidates for handling in parallel
  - This is formalised as the ‘map’ semantics from map/reduce



Dealing With Big Data Outside Of The Cloud: GPU Accelerated Sort – Vidler et.al

Data Dependencies

- The ParlaMint dataset is *highly* independent
  - Three levels of iteration:
    1. By Language
    2. By Year
    3. By File

- But… the corpus is also *very large™,* so rather than attempting to parallelise everywhere, instead focus on "easy wins"
  - For reference, let us take a quick look at the folder structure

ParlaMint Corpus Folder Structure

One folder for every language

```
ParlaMint-AT-en.conllu
├── 1996
│   ├── ParlaMint-AT-en_1996-01-15-020-XX-NRSITZ-00001.conllu
│   ├── ParlaMint-AT-en_1996-01-15-020-XX-NRSITZ-00002.conllu
│   ├── etc…
├── 1997
│   ├── ParlaMint-AT-en_1997-01-14-020-XX-NRSITZ-00055.conllu
│   ├── ParlaMint-AT-en_1997-01-14-020-XX-NRSITZ-00056.conllu
│   ├── etc…
```

One folder for every year collection

One file for every data set

ParlaMint Corpus Folder Structure

Lancaster University

Easy Win #1 - We can run more than one *language* at once

One folder for every language

```
ParlaMint-AT-en.conllu
├── 1996
│   ├── ParlaMint-AT-en_1996-01-15-020-XX-NRSITZ-00001.conllu
│   ├── ParlaMint-AT-en_1996-01-15-020-XX-NRSITZ-00002.conllu
│   ├── etc…
├── 1997
│   ├── ParlaMint-AT-en_1997-01-14-020-XX-NRSITZ-00055.conllu
│   ├── ParlaMint-AT-en_1997-01-14-020-XX-NRSITZ-00056.conllu
│   ├── etc…
```

One folder for every year collection

Easy Win #2 – We can run more than one *year* at once

One file for every data set

Easy Win #3 – We can run more than one *file* at once

ParlaMint Corpus Folder Structure

Lancaster University

One folder for every language

Easy Win #1 - We can run more than one *language* at once

```
ParlaMint-AT-en.conllu
├── 1996
│   ├── ParlaMint-AT-en_1996-01-15-020-XX-NR
│   ├── ParlaMint-AT-en_1996-01-15-020-XX-NR
│   ├── etc…
├── 1997
│   ├── ParlaMint-AT-en_1997-01-14-020-XX-NR
│   ├── ParlaMint-AT-en_1997-01-14-020-XX-NR
│   ├── etc…
```

One folder for every year collection

Easy Win #2 – We can run more than one *year* at once

One file for every data set

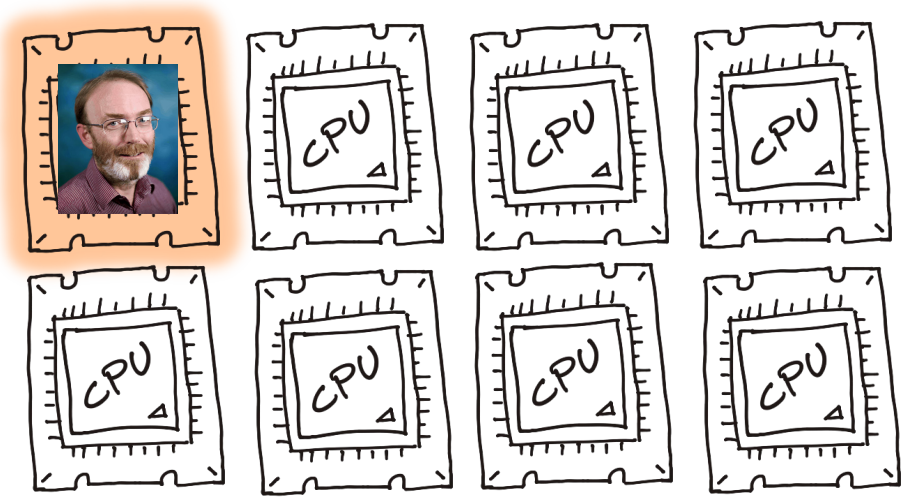
Why not #3?

We may run the risk of over producing very short-lived jobs; which *might* not be a problem, but could easily result in the majority of our time spent running being taken up by starting/stopping the jobs rather than actually running them.

~~Easy Win #3 – We can run more than one *file* at once~~

Hardware Options

- Paul was originally running the jobs on his laptop
  - Paul's Laptop is an 8-core Apple M1!
    - … but only one was being used at once ☹
- The single language jobs were later moved to one of the UCREL VMs
  - … but still only on one core ☹



Hardware Options

- But we also have access to some Ampere, ARM64 server hardware with up to *80 cores!*
  - Just so we don't go completely over the top, we'll just use 64 of them ☺

~ — ubuntu@instance-20230908-2241: ~ — ssh oci-arm

```
nmon-16n                    Hostname=instance-2023Refresh= 2secs ---01:03.30
 CPU Utilisation
--------------------------+-------------------------------------------------+
CPU User%  Sys% Wait%  Idle|0          |25         |50         |75       100|
  1   0.5   0.0   0.0  99.5|                                                  >
  2   0.0   0.0   0.0 100.0|                                                  >
  3   0.5   0.0   0.0  99.5|                                                  >
  4   0.5   0.0   0.0  99.5|                                                  >
  5   0.5   0.0   0.0  99.5|                                                  >
  6   0.5   0.0   0.0  99.5|                                                  >
  7   0.0   0.0   0.0 100.0|                                                  >
  8   0.0   0.0   0.0 100.0|                                                  >
  9   0.5   0.0   0.0  99.5|                                                  >
 10   0.0   0.0   0.0 100.0|                                                  >
 11   0.5   0.0   0.0  99.5|                                                  >
 12   0.5   0.0   0.0  99.5|                                                  >
 13   0.5   0.0   0.0  99.5|                                                  >
 14   0.0   0.0   0.0 100.0|                                                  >
 15   0.5   0.0   0.0  99.5|                                                  >
 16   0.0   0.0   0.0 100.0|                                                  >
 17   0.5   0.0   0.0  99.5|                                                  >
 18   0.0   0.0   0.0 100.0|                                                  >
 19   0.0   0.0   0.0 100.0|                                                  >
 20   0.0   0.0   0.0 100.0|                                                  >
 21   0.5   0.0   0.0  99.5|                                                  >
 22   0.5   0.0   0.0  99.5|                                                  >
 23   0.5   0.0   0.0  99.5|                                                  >
 24   0.5   0.0   0.0  99.5|                                                  >
 25   0.5   0.0   0.0  99.5|                                                  >
 26   0.5   0.0   0.0  99.5|                                                  >
 27   0.0   0.0   0.0 100.0|                                                  >
 28   0.0   0.0   0.0 100.0|                                                  >
 29   0.5   0.0   0.0  99.5|                                                  >
 30   0.5   0.0   0.0  99.5|                                                  >
 31   0.5   0.0   0.0  99.5|                                                  >
 32   0.0   0.0   0.0 100.0|                                                  >
 33   0.0   0.0   0.0 100.0|                                                  >
 34   0.5   0.0   0.0  99.5|                                                  >
 35   0.0   0.0   0.0 100.0|                                                  >
 36   0.5   0.0   0.0  99.5|                                                  >
 37   0.0   0.0   0.0 100.0|                                                  >
 38   0.0   0.0   0.0 100.0|                                                  >
 39   0.0   0.0   0.0 100.0|                                                  >
 40   0.5   0.0   0.0  99.5|                                                  >
 41   0.5   0.0   0.0  99.5|                                                  >
 42   0.0   0.0   0.0 100.0|                                                  >
 43   0.5   0.0   0.0  99.5|                                                  >
 44   0.5   0.0   0.0  99.5|                                                  >
 45   0.0   0.0   0.0 100.0|                                                  >
 46   0.0   0.0   0.0 100.0|                                                  >
 47   0.5   0.0   0.0  99.5|                                                  >
 48   0.5   0.0   0.0  99.5|                                                  >
 49   0.5   0.0   0.0  99.5|                                                  >
 50   0.0   0.0   0.0 100.0|                                                  >
 51   0.0   0.0   0.0 100.0|                                                  >
 52   0.0   0.0   0.0 100.0|                                                  >
 53   0.0   0.0   0.0 100.0|                                                  >
 54   0.0   0.0   0.0 100.0|                                                  >
 55   0.5   0.0   0.0  99.5|                                                  >
 56   0.0   0.0   0.0 100.0|                                                  >
 57   0.5   0.0   0.0  99.5|                                                  >
 58   0.0   0.0   0.0 100.0|                                                  >
 59   0.5   0.0   0.0  99.5|                                                  >
 60   0.5   0.0   0.0  99.5|                                                  >
 61   0.0   0.0   0.0 100.0|                                                  >
 62   0.5   0.0   0.0  99.5|                                                  >
 63   0.5   0.5   0.0  99.0|                                                  >
----------Warning: Some Statistics may not shown----------
```

CPU Load

```
ubuntu@instance-20230908-2241:~$ ls ParlaMint-AT-en.conllu
1996  1998  2000  2002  2004  2006  2008  2010  2012  2014  2016  2018  2020  2022
1997  1999  2001  2003  2005  2007  2009  2011  2013  2015  2017  2019  2021
ubuntu@instance-20230908-2241:~$ ls ParlaMint-AT-en.conllu | wc -l
27
ubuntu@instance-20230908-2241:~$ /opt/pymusas-conllu-parlamint-main/pymusas_parlamint_wrapper.sh /home/ubuntu/ParlaMi
nt-AT-en.conllu
```

Driver Script

2 bash

```
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
```

Docker Stats

Putting the scripts together - /proc/cpuinfo

- Step #1 – Get how many cores we have to work with
- Step #2 – Launch enough processes to keep every core (mostly) busy

- Step 1 is easy (if you're on Linux or Mac) and you can use a Bash script:

```
CORES = $(grep -c ^processor /proc/cpuinfo)
```

regex

count

pattern

our cpu info proc file

Putting the scripts together - xargs

- Step #2 sounds hard, but we can use existing tools common to linux to do this for us ☺

`xargs`

"Construct argument list(s) and execute utility" – linux man page

- This command takes a list, and for every item in the list, runs a command passing the item to the command
  - This gets us 50% of the way already!

Putting the scripts together - xargs

- xargs also has a ”-P” command line option:

  “-P [processes]; Run up to [processes] at once in parallel”

- So now we just need to get a list for xargs to pass through

the processes

```
ls /path/to/files | xargs -P 8 ./job.sh
```

the list

the job

(Check your xargs man page for system specifics!)

Decoupling Context via Docker

- Each job was already being run as a throwaway docker container
  - This helped with replicability and to avoid configuration issues when jobs were run on the various project members computers

- We can reuse this same container here with no modification ☺
  - The original script that launched the container had two nested loops (year/file)
  - We just need it to loop over one year

- So – we modify the driver script to only run over one folder (a year), then wrap that one in a new script to launch it via xargs in parallel mode for every year

~ — ubuntu@instance-20230908-2241: ~ — ssh oci-arm

```
nmon-16n                    Hostname=instance-2023Refresh= 2secs ---01:14.09
 CPU Utilisation
---------------------------+-------------------------------------------------+
CPU User%  Sys% Wait%  Idle|0          |25         |50         |75       100|
  1   0.0   0.0   0.0 100.0|                                                 >
  2   0.0   0.0   0.0 100.0|       >                                         |
  3   0.0   0.0   0.0 100.0|                                   >             |
  4   0.0   0.0   0.0 100.0|>                                                |
  5   0.0   0.0   0.0 100.0|>                                                |
  6   0.0   0.0   0.0 100.0|    >                                            |
  7   0.0   0.0   0.0 100.0|>                                                |
  8   0.0   0.0   0.0 100.0|                                                 >
  9   0.0   0.0   0.0 100.0|                     >                           |
 10   0.0   0.0   0.0 100.0|                                                 >
 11   0.0   0.0   0.0 100.0|  >                                              |
 12   0.0   0.0   0.0 100.0|  >                                              |
 13   0.0   0.0   0.0 100.0|                 >                               |
 14   0.0   0.0   0.0 100.0|>                                                |
 15   0.0   0.0   0.0 100.0| >                                              |
 16   0.0   0.0   0.0 100.0|                                                >|
 17   0.0   0.0   0.0 100.0|                                                 >
 18   0.0   0.0   0.0 100.0|>                                                |
 19   0.0   0.0   0.0 100.0| >                                              |
 20   0.0   0.0   0.0 100.0|    >                                           |
 21   0.0   0.0   0.0 100.0| >                                              |
 22   0.0   0.0   0.0 100.0|      >                                         |
 23   0.0   0.0   0.0 100.0|   >                                            |
 24   0.0   0.0   0.0 100.0|                                                 >
 25   0.0   0.0   0.5  99.5|                   >                             |
 26   0.0   0.0   0.0 100.0|>                                                |
 27   0.0   0.0   0.0 100.0|                >                                |
 28   0.0   0.0   0.0 100.0|>                                                |
 29   0.0   0.0   0.0 100.0|    >                                           |
 30   0.0   0.0   0.0 100.0|                 >                               |
 31   0.0   0.0   0.0 100.0|>                                                |
 32   0.0   0.0   0.0 100.0|    >                                           |
 33   0.0   0.0   0.0 100.0|                                   >             |
 34   0.0   0.0   0.0 100.0|      >                                         |
 35   0.0   0.0   0.0 100.0|      >                                         |
 36   0.0   0.0   0.0 100.0|                             >                   |
 37   0.0   0.0   0.0 100.0|      >                                         |
 38   0.0   0.0   0.0 100.0| >                                              |
 39   0.0   0.0   0.0 100.0|                  >                              |
 40   0.0   0.0   0.0 100.0| >                                              |
 41   0.0   0.0   0.0 100.0| >                                              |
 42   0.0   0.0   0.0 100.0|          >                                     |
 43   0.0   0.0   0.0 100.0|>                                                |
 44   0.0   0.0   0.0 100.0|  >                                             |
 45   0.0   0.5   0.0  99.5|                  >                              |
 46   0.0   0.0   0.0 100.0|                 >                               |
 47   0.0   0.0   0.0 100.0|                >                                |
 48   0.0   0.0   0.0 100.0|   >                                            |
 49   0.0   0.0   0.0 100.0| >                                              |
 50   0.0   0.0   0.0 100.0|         >                                      |
 51   0.0   0.0   0.0 100.0| >                                              |
 52   0.0   0.0   0.0 100.0|         >                                      |
 53   0.0   0.0   0.0 100.0|      >                                         |
 54   0.0   0.0   0.0 100.0|  >                                             |
 55   0.0   0.0   0.0 100.0|                                                 >
 56   0.0   0.0   0.0 100.0|                     >                           |
 57   0.0   0.0   0.0 100.0|   >                                            |
 58   0.0   0.0   0.0 100.0|>                                                |
 59   0.0   0.0   0.0 100.0|>                                                |
 60   0.0   0.0   0.0 100.0|    >                                           |
 61   0.0   0.0   0.0 100.0|>                                                |
 62   0.0   0.0   0.0 100.0|      >                                         |
 63   0.0   0.0   0.0 100.0|      >                                         |
----------Warning: Some Statistics may not shown-----------------------------
```

```
ubuntu@instance-20230908-2241:~$ /opt/pymusas-conllu-parlamint-main/pymusas_parlamint_wrapper.sh /home/ubuntu/ParlaMi
nt-AT-en.conllu
```

```
  1 bash
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK I/O   PIDS
```

What is ‘Podman’ or ‘Docker’?

What is ‘Podman’ or ‘Docker’?

From the Docker website:

“An open platform for developing, shipping and running applications”

… Fancy, but doesn’t really tell us anything.

What is ‘Podman’ or ‘Docker’?

Lancaster University

From the Docker website (Again!)

”Docker packages and runs applications in a **loosely isolated** environment called a container”

Loosely Isolated

- Somewhat like a VM, but much, much lighter
- Keeps the instance isolated, but does the minimum to do so

What is ‘Podman’ or ‘Docker’?

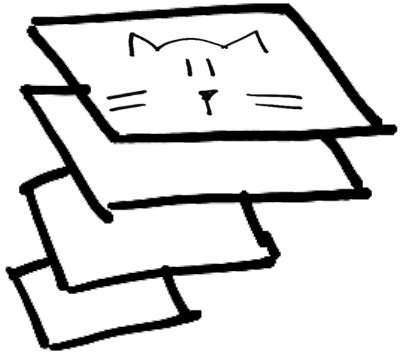
From the Docker website (Again!)

”Docker packages and runs applications in a loosely isolated environment called a **container**”

Container

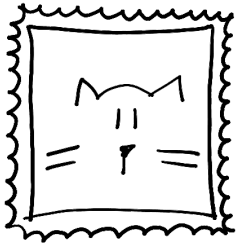
– Describes all the *state* for an **image**

What is an ‘Image’?



Image

- A **read-only** template for creating a container
- Very often, images are based on another image
  - Conceptually, this can be thought of as a stack of cards we ‘play’ in order to build up a complete container
    - When we build our own images, we should aim to keep the stack as short as possible to minimise overall complexity and download sizes



… No cat, not *that* kind of image.

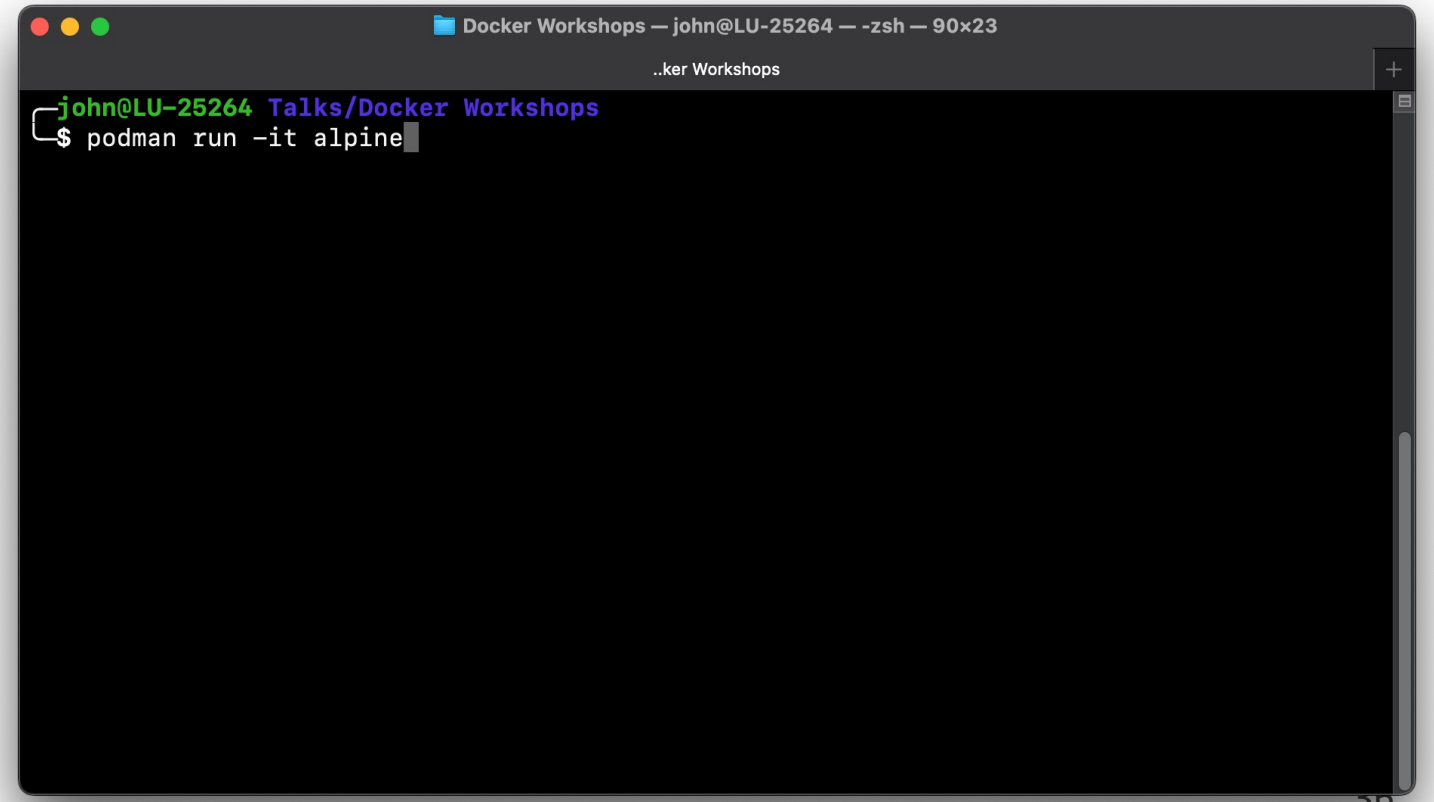
Play Along at Home!

The base layer: Alpine Linux

We'll use Ubuntu later, but Alpine is super fast for playing along with!

This will perform the following steps:

1. Download the "alpine" **image**
2. Create a new **container**
3. Start the container
4. Connect in interactive terminal mode
   (the '`-i`' and '`-t`' flags)



The base layer: Alpine Linux

We're now inside a running Alpine Linux container as root!

Feel free to look around with the usual Linux commands

You can do pretty much anything here, and it will be kept in the container.

Docker Workshops — podman run -it alpine — podman run -it alpine — 90×23

podman

podman run -it alpine

```
john@LU-25264 Talks/Docker Workshops
$ podman run -it alpine
Resolved "alpine" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Copying blob sha256:690e87867337b8441990047e169b892933e9006bdbcbed52ab7a356945477a4d
Copying config sha256:0b4426ad4bf25e13fb09112b9dcb5d5b09b3c5684599654583913b2714a705a2
Writing manifest to image destination
/ #
```

The base layer: Alpine Linux

Alpine Linux is extremely small, so is a popular base image for many other container images

Here's an example of me installing `htop` - the process viewer – in our running container.

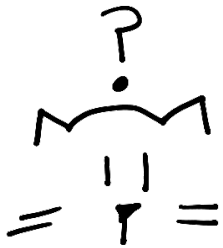
Docker Workshops — podman run -it alpine — podman run -it alpine — 90×23

podman

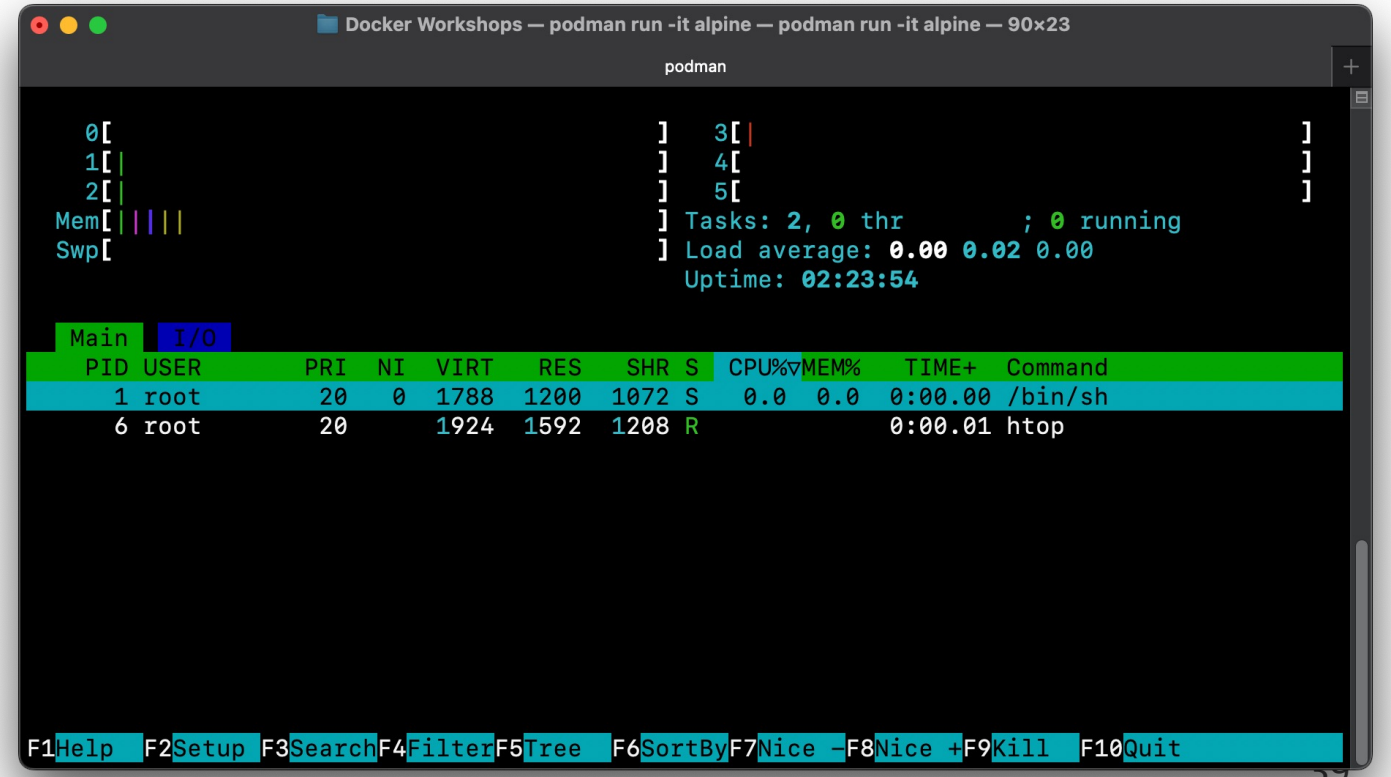
```
john@LU-25264 Talks/Docker Workshops
$ podman run -it alpine
Resolved "alpine" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/alpine:latest...
Getting image source signatures
Copying blob sha256:690e87867337b8441990047e169b892933e9006bdbcbed52ab7a356945477a4d
Copying config sha256:0b4426ad4bf25e13fb09112b9dcb5d5b09b3c5684599654583913b2714a705a2
Writing manifest to image destination
/ # apk update
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/main/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/community/aarch64/APKINDEX.tar.gz
v3.20.2-19-ga9bc4d9eff9 [https://dl-cdn.alpinelinux.org/alpine/v3.20/main]
v3.20.2-21-g30ed104df72 [https://dl-cdn.alpinelinux.org/alpine/v3.20/community]
OK: 24033 distinct packages available
/ # apk add htop
(1/3) Installing ncurses-terminfo-base (6.4_p20240420-r0)
(2/3) Installing libncursesw (6.4_p20240420-r0)
(3/3) Installing htop (3.3.0-r0)
Executing busybox-1.36.1-r29.trigger
OK: 10 MiB in 17 packages
/ #
```

The base layer: Alpine Linux

Hey, this isn't very much…



Normally you would expect to see a lot more in this list on a Linux system, but docker runs on an *already running* host system, so only needs just enough within the container to work



Docker Status

If we open another terminal for a moment, and run ‘podman ps’ we can see our running container, and some basic information about it

This command normally only shows us the *running* containers

If we add the ‘-a’ flag, we can see all containers, including those we have previously stopped (but are still available to re-start later)

Docker Workshops — john@LU-25264 — -zsh — 100×32

..ker Workshops

```
john@LU-25264 Talks/Docker Workshops
$ podman ps
CONTAINER ID  IMAGE                           COMMAND     CREATED        STATUS        PORTS
NAMES
10a9b8b3f7d8  docker.io/library/alpine:latest  /bin/sh     8 seconds ago  Up 8 seconds
sweet_sammet
john@LU-25264 Talks/Docker Workshops
$ podman ps -a
CONTAINER ID  IMAGE                           COMMAND     CREATED         STATUS
  PORTS       NAMES
984b2bb170da  docker.io/library/alpine:latest  /bin/sh     3 minutes ago   Exited (0) 14 seconds ago
              beautiful_meitner
10a9b8b3f7d8  docker.io/library/alpine:latest  /bin/sh     11 seconds ago  Up 11 seconds
              sweet_sammet
john@LU-25264 Talks/Docker Workshops
$
```

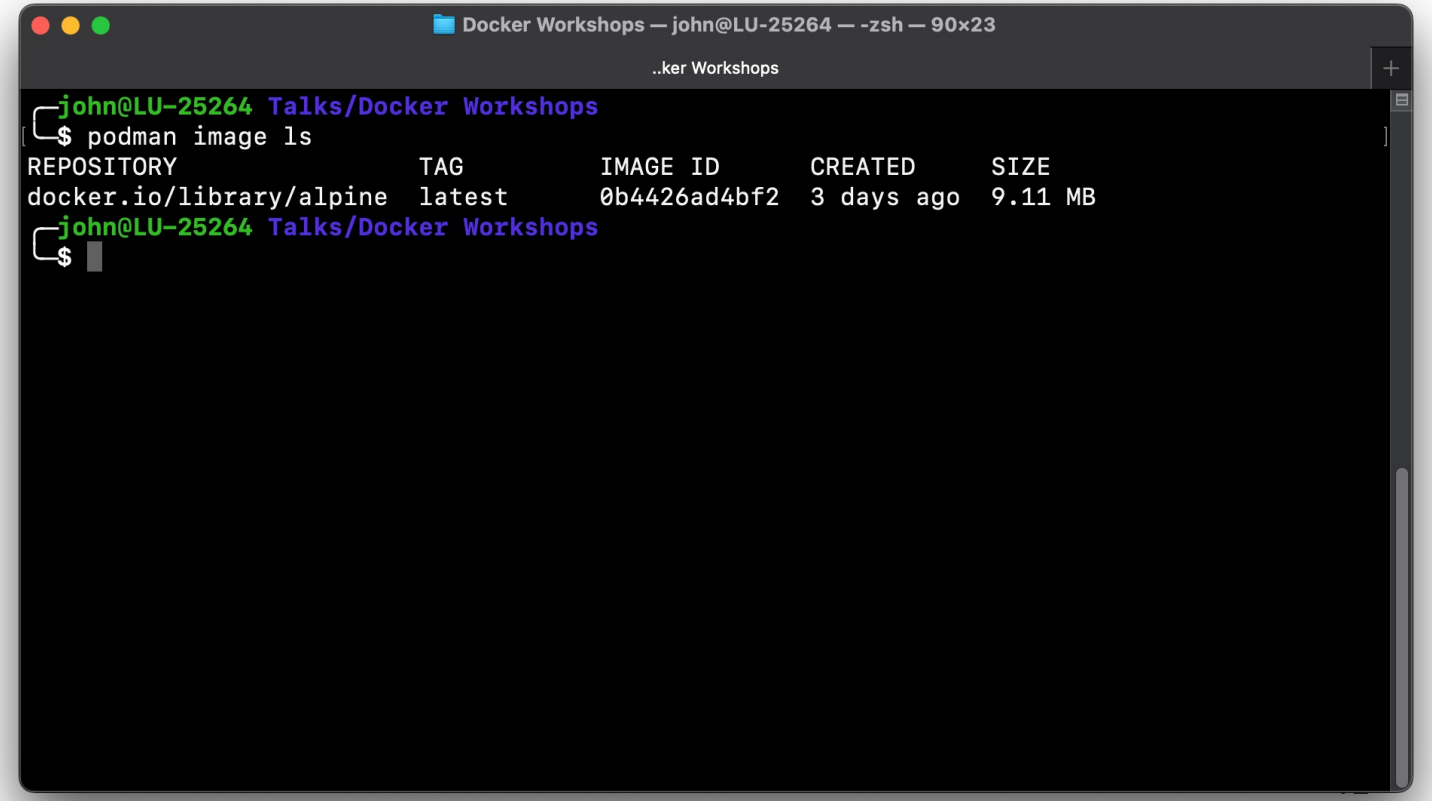
Docker Images

We can also ask podman to tell us which **images** it has downloaded, along with some basic information about each one.

I've cleared my laptop of all old images so we only see our current 'alpine' image.

Note that this one is *technically* the 'alpine:latest' image, where the part after the colon is the 'tag'

(we'll cover tags at a later stage)



Removing Containers

For now, let's clear out both of my existing containers.

We can do this through `podman rm <NAME>` where the name is either the Container ID, OR the Container Name

Here we use one of each form, just because we can.

If we had run our container in daemon (background) mode, we would have also had to stop it first, via 'docker stop <NAME>'

Docker Workshops — john@LU-25264 — -zsh — 90×23

..ker Workshops

```
┌─john@LU-25264 Talks/Docker Workshops
└─$ podman ps -a
CONTAINER ID  IMAGE                           COMMAND     CREATED        STATUS
              PORTS       NAMES
984b2bb170da  docker.io/library/alpine:latest  /bin/sh     5 minutes ago  Exited (0) 2 min
utes ago                  beautiful_meitner
10a9b8b3f7d8  docker.io/library/alpine:latest  /bin/sh     2 minutes ago  Exited (0) About
 a minute ago             sweet_sammet
┌─john@LU-25264 Talks/Docker Workshops
└─$ podman rm 984b2bb170da
984b2bb170da
┌─john@LU-25264 Talks/Docker Workshops
└─$ podman rm sweet_sammet
sweet_sammet
┌─john@LU-25264 Talks/Docker Workshops
└─$ podman ps -a
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
┌─john@LU-25264 Talks/Docker Workshops
└─$
```

Throwaway Containers

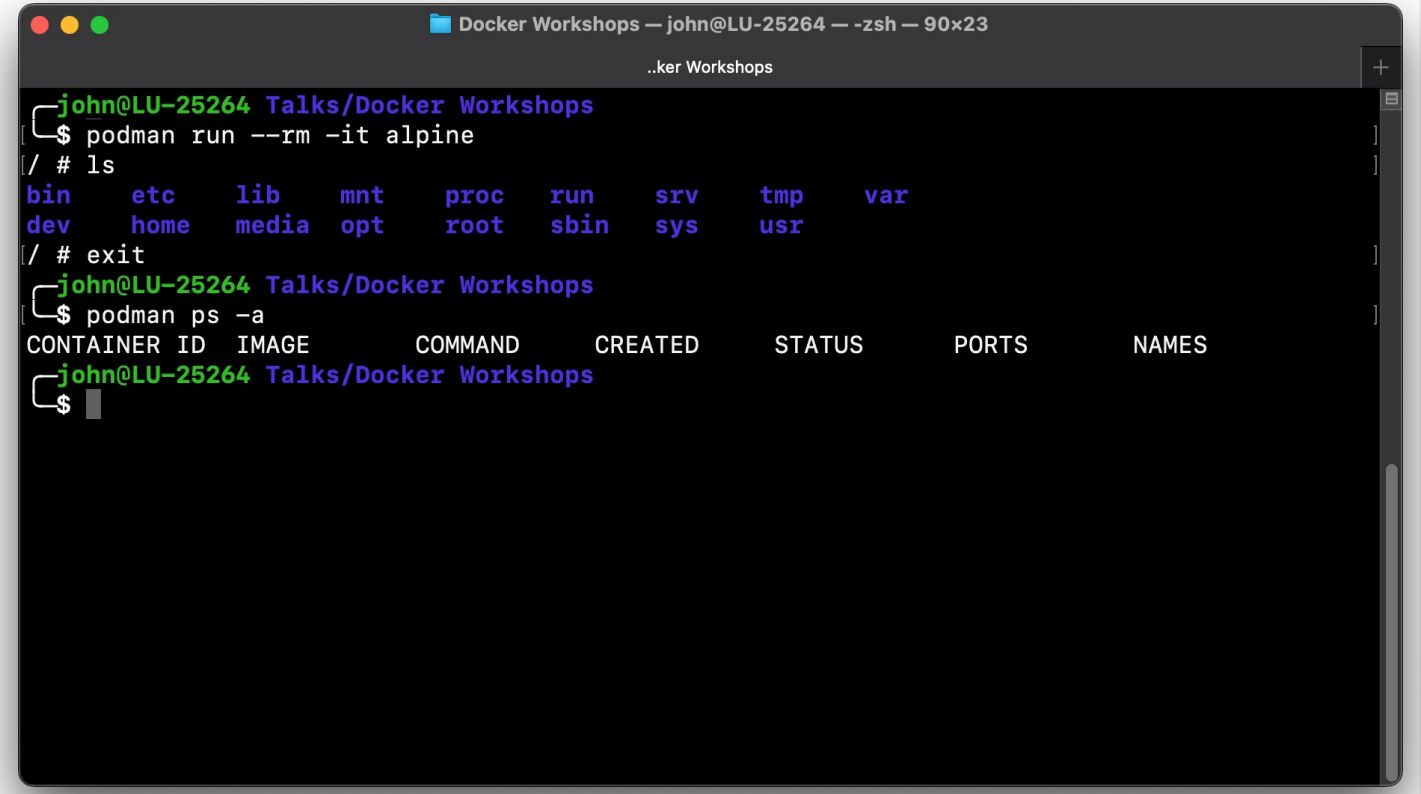
A handy way of trying things in a ‘throwaway’ mode is to include the ‘**--rm**’ flag

This will automatically remove the container when the last process inside it exits
(In this case, when we run ‘exit’)

Note that everything will be lost when this happens, however!

Don’t rely on this for anything important!

This does keep the image, however.



Hello, Python!

We could install Python on the Alpine Linux image, but good practice when using containers is to start from as close an image as possible to your desired final one

Handily, there is a Python image based on Alpine Linux

We can select which variant we want of an image by adding ‘tags’ after a colon in the image name

And voila! A Python REPL

Docker Workshops — podman run --rm -it python:3-alpine — podman run --rm -it python:3-alpine — 90×23

podman

```
john@LU-25264 Talks/Docker Workshops
$ podman run --rm -it python:3-alpine
Resolved "python" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/python:3-alpine...
Getting image source signatures
Copying blob sha256:f37d7261ef4679c56a1bba3f841f0b28fd0b97f68f6be6c515d75a4053aa0b5f
Copying blob sha256:690e87867337b8441990047e169b892933e9006bdbcbed52ab7a356945477a4d
Copying blob sha256:e895e298d43fd334952e9dfd3ecead5d7a971204a21f217b3bca1b8ca350711c
Copying blob sha256:2711f91d5f9c677eb6883af3f381a64f628a585ad46fcfcfeecf39228185c2b2
Copying blob sha256:748a71c342762ea4de72dcae953bae8f85ced09ae9e38089d823a725fbe4b71f
Copying config sha256:378afc3748a3e74c45055fbd0ca1843ad0843da7034ff3c0f15ea0247ca1af1e
Writing manifest to image destination
Python 3.12.4 (main, Jul 24 2024, 04:20:58) [GCC 13.2.1 20240309] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello, Python!")
Hello, Python!
>>>
```

Hello, Python .. Version 2.7.18 !

Lancaster University

A common problem with fast-moving languages like Python is that you can need a specific version to run some software

On Python's Docker Hub page there are tagged images for many versions of Python, so we can get whichever one we need, preconfigured and ready to go

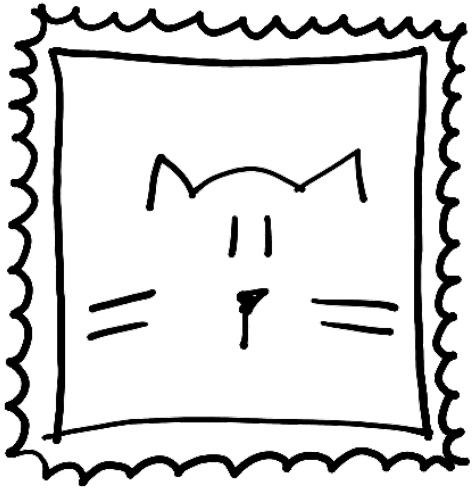
See the link below for many, many other options.
https://hub.docker.com/_/python

Docker Workshops — podman run --rm -it python:2-alpine — podman run --rm -it python:2-alpine — 90×23

podman podman run --rm -it python:2-alpine

```
john@LU-25264 Talks/Docker Workshops
$ podman run --rm -it python:2-alpine
Resolved "python" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/python:2-alpine...
Getting image source signatures
Copying blob sha256:1fb0c72978ae903933f5eacae2eb6455ff072cde77412b254062cd5c1b07291b
Copying blob sha256:8a0637ca1ac98db4cf29f7632449c92801adc80cf0da2cd9c9e39882ce466561
Copying blob sha256:00849b448b02c479ffcb45a0003d5112d8cd987790707b9eb7b1aeb43a9903d3
Copying blob sha256:43a775321ff74ad65fdf9d1705c9244831619bb811974ca8f2e5b8568eb824bf
Copying config sha256:6893769f63c8343bad93e84ce211bda12603a9af2fde17e720e79dc7684d85bd
Writing manifest to image destination
Python 2.7.18 (default, Apr 20 2020, 20:18:02)
[GCC 9.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Building our own Images



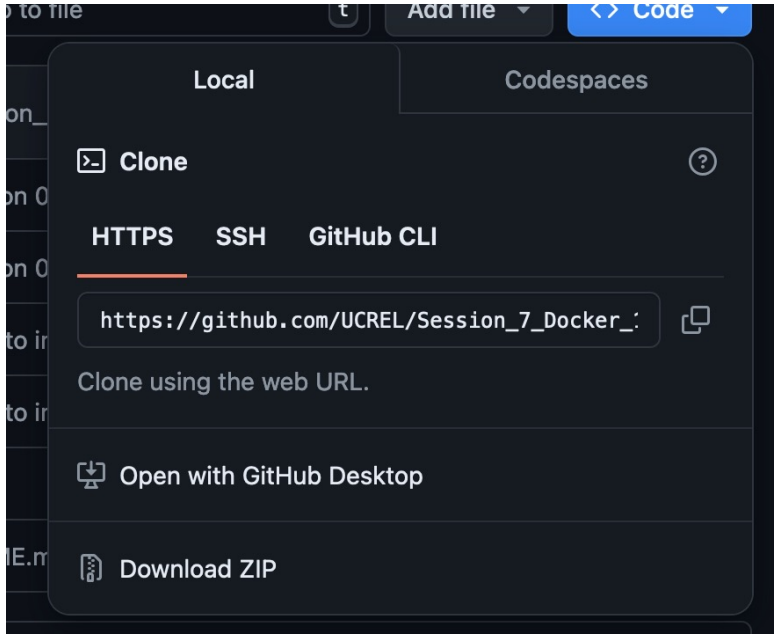
Building a new container image!

I've uploaded the sources for a tiny web service to Github. If you have git installed, you can clone the repository with:

```
git clone
https://github.com/UCREL/Session_7_Docker_101a.git
```

Or you can just download the whole lot as a zip by going to the following URL and using the link shown in the image on the right:

https://github.com/UCREL/Session_7_Docker_101a

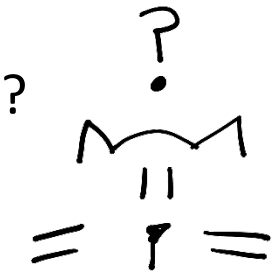


Building Container Images – A Python Web App

Dockerfile

- A “Dockerfile”, much like a Makefile is a recipe for a container **image**
- It describes the steps required to build a new image
  - This should always be the static parts of a container. The runtime commands come later.

- What does this look like?



Building Docker Images – The Dockerfile

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
"Dockerfile" [noeol] 12L, 182B
```

A few new things to unpack here, but hopefully nothing too surprising:

Lets walk through the file explaining each **directive** in turn

- **FROM**
- **WORKDIR**
- **COPY**
- **RUN**
- **EXPOSE**
- **CMD**

Dockerfile – The **FROM** directive

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
-- VISUAL LINE --
```

FROM declares which image this new image should be based on

In the previous examples we used Python 3, on Alpine Linux, but for variety, lets use the 'python:3' tag which is based on Ubuntu Linux

To make things quicker later on, you might want to pull this image now, using:

"podman pull python:3"

Dockerfile – The **WORKDIR** directive

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
-- VISUAL LINE --
```

The **WORKDIR** directive sets the current working directory *inside the image* for subsequent directives to execute relative to

Common ones include /src, /home, but its up to you and the base image to define what is appropriate.

Personally, I'm a fan of using the /opt/appname schema, but here we're going to use /usr/src/app

This directive will auto-create the working directory, if required

Dockerfile – The **COPY** directive

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
-- VISUAL LINE --
```

COPY unsurprisingly copies files, but rather than from path to path within the image or the host, this directive actually copies from the host to the image

The first argument is the path on the **host**, the second is on the **image**

Paths can be individual files, or whole directories! Our second **COPY** call later in this file copies the entire ‘app’ directory on the host to our containers working directory (aka, the ‘.’ shortcut)

Dockerfile – The **RUN** directive

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
-- VISUAL LINE --
```

RUN executes commands within the context of the image

These can be any valid command for the parent image at this point in the build

Here we're running 'pip' to install any dependencies listed in our freshly **COPY**'d requirements.txt file

Each RUN creates a **layer** to the image. We should attempt to keep the number of layers as low as possible to ensure that images don't become too large.

A note on **layers** – an example from my work

MakeCode Local Dev Environment — vim Dockerfile — vim Dockerfile — 77×23

vim

```
# Update and upgrade the system, installing python 3.6 explicitly for yotta
RUN apt update && \
    DEBIAN_FRONTEND=noninteractive TZ=Europe/London apt-get -y install tzdata
 && \
    apt install -y build-essential cmake software-properties-common nodejs np
m git wget srecord libncurses-dev && \
    apt upgrade -y && \
    add-apt-repository ppa:deadsnakes/ppa && \
    apt update && \
    apt install -y python3.6 python3-pip python-setuptools ninja-build python
-dev libffi-dev libssl-dev

# Actually install yotta
WORKDIR /home
COPY requirements.txt /home/requirements.txt
RUN python3 -m pip --no-cache-dir install --upgrade pip && \
    python3 -m pip --no-cache-dir install -r /home/requirements.txt && \
    rm /home/requirements.txt


# Download and install the arm toolchain (arm-non-eabi-*)
```

Because we'd like to have a low number of layers, it's a common pattern in a Dockerfile to pack several commands into one **RUN** directive.

To do this in a mostly readable way, I suggest you have each command on its own line, each terminated with `&& \` to indicate that there are more commands that should be executed in sequence.

I try to make sure that each block of commands are co-related to one another.

Dockerfile – The **EXPOSE** directive

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
-- VISUAL LINE --
```

EXPOSE informs Docker that the container listens on the port(s) listed at runtime, and can be listed multiple times in a single Dockerfile.

This **does not publish** the port, but acts to annotate that this port would present a useful interface should it be published.

Many container orchestrators also use it to identify which ports should be presented on a management interface to be routed somewhere.

I'll show how to **publish** ports shortly

Dockerfile – The **CMD** directive

Python Web Demo — vim Dockerfile — vim Dockerfile — 77×23

vim

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY app .

EXPOSE 8080/tcp

CMD [ "python", "./simple-web-app.py" ]
~
~
~
~
~
~
~
~
~
~
-- VISUAL LINE --
```

Finally, the **CMD** directive tells Docker which commands to run when launching the image as a container.

If other images extend from this one and do not override this directive with their own **CMD** directive, this one will continue to be used.

Note that this has the form of an array or a list, rather than a plain Linux command.

Arguments should be separated into individual array elements!

Hello, WebApp!

Let's build our webapp!

From the new directory (from extracting our .zip file, or cloning the git repository) we should now be able to run:

'podman build -t webapp .'

The dot at the end is important! Its telling podman to build from the current directory (where our Dockerfile is!)

00_simple_dockerfile_example — john@LU-25264 — -zsh — 90×40

..rfile_example

```
john@LU-25264 Session_7_Docker_101a/00_simple_dockerfile_example ‹main›
$ podman build -t webapp .
STEP 1/7: FROM python:3
Resolved "python" as an alias (/etc/containers/registries.conf.d/000-shortnames.conf)
Trying to pull docker.io/library/python:3...
Getting image source signatures
Copying blob sha256:12001ff3ba4b7d43340632be637cdb26f78dfff7064bf6168cb47e3c19e0b90d
Copying blob sha256:0df40ff8dff06855b2dff09ca815eb5044fdfb6861e4d23120e04f07ce113184
Copying blob sha256:9c5ed83eaf5c33e6b2ceb5fe1b2b1300f9117a5dc5eae13b75f9f66dcce43a0f
Copying blob sha256:adacb995432c92df6de0b5690abdd064e095988fac45631ba8fc0a0ffa9be5cc
Copying blob sha256:4e903e4e709d192e5547602a5978c79692063228a98585f33fb02d343bc15719
Copying blob sha256:5173985f496791cc83e90800b73dabe5cb8c60455c3529854ced5412b4809682
Copying blob sha256:6bceaf9865ca98cc73ebc8ea5e3979560fb23bf1f6c89162bcfa4a1f991df9dc
Copying blob sha256:6cb019d8d1e4146e4f421c6f084827a88ff0489d80c339146ed3d04d40787f76
Copying config sha256:933413f5d3cdf8f5007c7d8f54a7871246495401ec140792ac87d53eb8dcbb38
Writing manifest to image destination
STEP 2/7: WORKDIR /usr/src/app
--> 4ff29610c74a
STEP 3/7: COPY requirements.txt ./
--> 8b5c8f41d825
STEP 4/7: RUN pip install --no-cache-dir -r requirements.txt
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting b
ehaviour with the system package manager. It is recommended to use a virtual environment i
nstead: https://pip.pypa.io/warnings/venv

[notice] A new release of pip is available: 24.0 -> 24.1.2
[notice] To update, run: pip install --upgrade pip
--> f0e8b7270d70
STEP 5/7: COPY src/ .
--> 7172ebcc4d2c
STEP 6/7: EXPOSE 8080/tcp
--> 7ee3622fcc13
STEP 7/7: CMD [ "python", "./simple-web-app.py" ]
COMMIT webapp
--> e922e203f222
Successfully tagged localhost/webapp:latest
e922e203f222222a0f992d8d1f95ccd61d1d9eb1f690e3d17b36d77036607ca7
john@LU-25264 Session_7_Docker_101a/00_simple_dockerfile_example ‹main›
$
```

Hello, WebApp!

Now we have an image to work with, we can start a container!

`podman run webapp`

Nothing will appear to be happening, as our web app doesn't print anything unless it gets a web request.

If we run 'podman ps' in another window, we should see that it is, indeed running, but has **no published ports**. To access the port, we need to publish by adding –p

`podman run -p 8080:8080 webapp`

```
00_simple_dockerfile_example — podman run --rm webapp — podman run --rm webapp — 90×40
podman
podman run --rm webapp
john@LU-25264 Session_7_Docker_101a/00_simple_dockerfile_example <main>
$ podman run --rm webapp
```

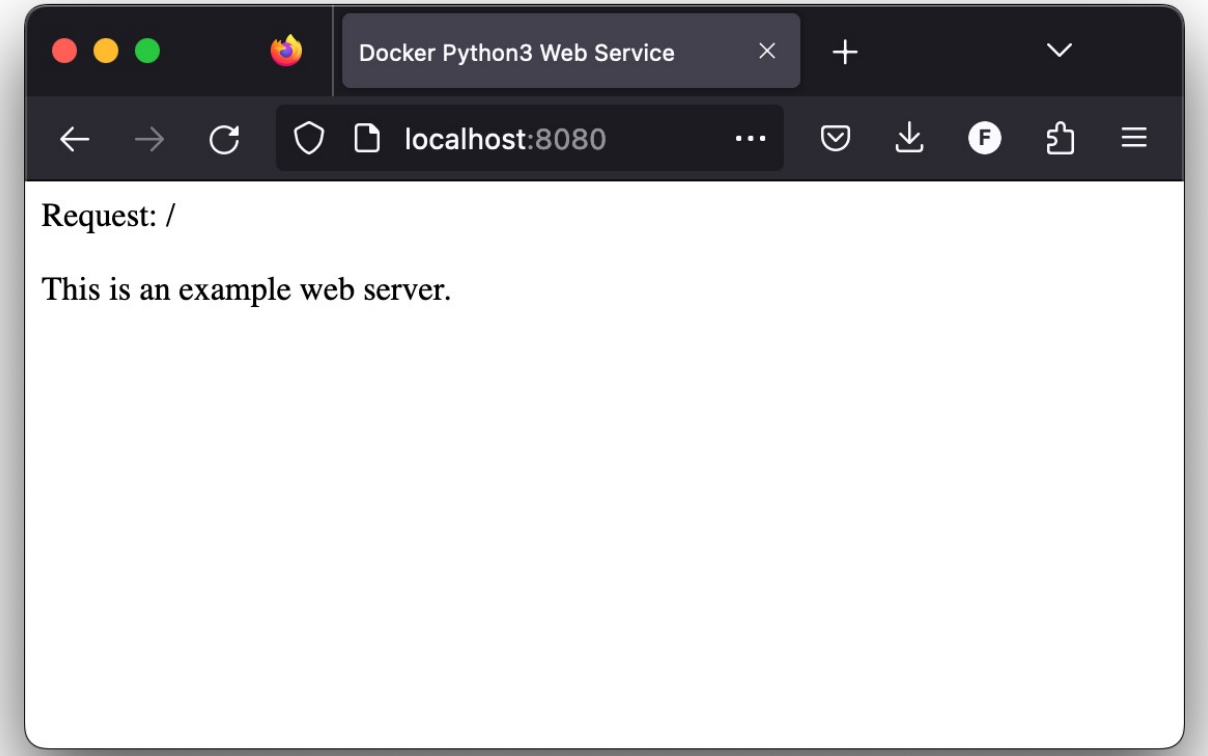
```
00_simple_dockerfile_example — john@LU-25264 — -zsh — 90×22
..rfile_example
Last login: Thu Jul 25 23:37:24 on ttys002
john@LU-25264 Session_7_Docker_101a/00_simple_dockerfile_example <main>
$ podman ps
CONTAINER ID  IMAGE                    COMMAND               CREATED        STATUS
  PORTS       NAMES
b5c0d78cc9ea  localhost/webapp:latest  python ./simple-w...  58 seconds ago  Up 58 seconds
              condescending_almeida
john@LU-25264 Session_7_Docker_101a/00_simple_dockerfile_example <main>
$ podman ps
CONTAINER ID  IMAGE                    COMMAND               CREATED        STATUS
PORTS                   NAMES
9e84cc8ae7f2  localhost/webapp:latest  python ./simple-w...  3 seconds ago  Up 4 seconds
0.0.0.0:8080->8080/tcp  stupefied_ride
john@LU-25264 Session_7_Docker_101a/00_simple_dockerfile_example <main>
$
```

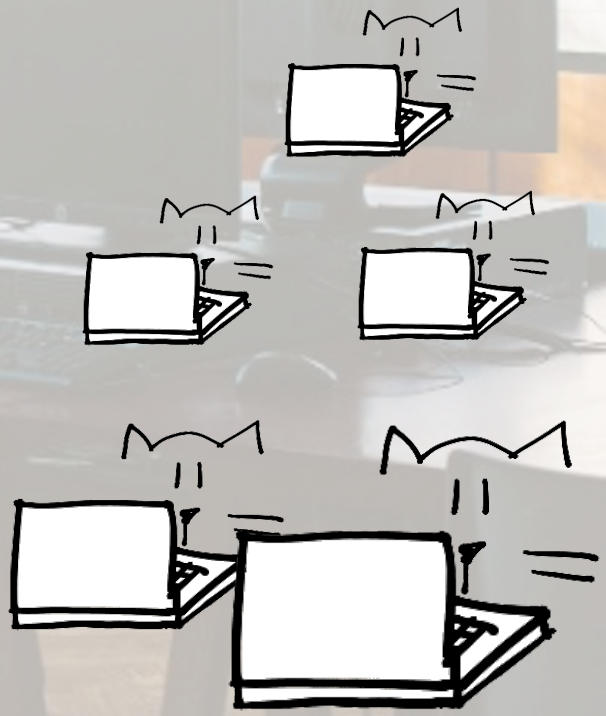
```
PORTS
0.0.0.0:8080->8080/tcp
```

Hello, WebApp!

Leaving our container running, we should now be able to open a web browser and go to ‘localhost:8080’ and we should see our webapp respond!

This is obviously extremely simple, but you should now have enough information to try things out with docker



Your turn

In the Lab!