Part four

Tips and tricks to optimize container creation.

Nicola Malavasi - Marie Curie Fellow @ MPE - 16/04/2024

Main problems of image creation

The two main bottlenecks of image creation from Dockerfile are the time required to build the image and the memory size of the image.

Memory size can be up to a few GB.

Both are dependent on what is installed inside the image at creation.

```
nmalavasi@ga-lt7982 tutorial_material % docker image ls
REPOSITORY   TAG                 IMAGE ID       CREATED          SIZE
debian       latest              5c2e61c12a03   39 hours ago     139MB
python       3.11.9-bookworm     c9a42713a959   8 days ago       1.01GB
ubuntu       latest              2b7cc08dcdbb   6 weeks ago      69.2MB
nmalavasi@ga-lt7982 tutorial_material %
```

Example

Package installation in Python

Consider the following Dockerfile. End result is python installation with a few packages.

Users > nmalavasi > Desktop > PUNCH_useful >

```
1   # syntax=docker/dockerfile:1
2
3   #Start from python image
4   FROM python:3.11.9-bookworm
5
6   #Use pip to install matplotlib
7   RUN pip install matplotlib
8
9   RUN pip install astropy
10
11  RUN pip install numpy
12
13  RUN pip install scipy
```

Example

Package installation in Python

Consider the following Dockerfile. End result is python installation with a few packages.

Users > nmalavasi > Desktop > PUNCH_useful >

```
1   # syntax=docker/dockerfile:1
2
3   #Start from python image
4   FROM python:3.11.9-bookworm
5
6   #Use pip to install matplotlib
7   RUN pip install matplotlib
8
9   RUN pip install astropy
10
11  RUN pip install numpy
12
13  RUN pip install scipy
```

Example

Package installation in Python

Consider the following Dockerfile. End result is python installation with a few packages.

Users > nmalavasi > Desktop > PUNCH_useful >

```
1   # syntax=docker/dockerfile:1
2
3   #Start from python image
4   FROM python:3.11.9-bookworm
5
6   #Use pip to install matplotlib
7   RUN pip install matplotlib
8
9   RUN pip install astropy
10
11  RUN pip install numpy
12
13  RUN pip install scipy
```

Modules installed sequentially

Example

Package installation in Python

Consider the following Dockerfile. End result is python installation with a few packages.

Users > nmalavasi > Desktop > PUNCH_useful >

```
1   # syntax=docker/dockerfile:1
2
3   #Start from python image
4   FROM python:3.11.9-bookworm
5
6   #Use pip to install matplotlib
7   RUN pip install matplotlib
8
9   RUN pip install astropy
10
11  RUN pip install numpy
12
13  RUN pip install scipy
```

Final size is about 1.5 GB

- Python is installed on top of full OS installation: we don’t need all of this if we are just going to use python.
- Each RUN command is a layer (i.e. an additional intermediate step in image creation). Each layer takes memory.

Example

Package installation in Python

Compare with the following. End result is same as before.

```
# syntax=docker/dockerfile:1

#Start from python image
FROM python:3.11.9-slim-bookworm

#Use pip to install matplotlib
RUN pip install matplotlib astropy numpy scipy

RUN python -m pip cache purge
```

- We use “slim” version of OS: only necessary packages for python installation.
- Packages are installed in same command: only one layer.
- Clean pip cache: once packages are installed they can be used, no need to keep them in cache (installation won’t be repeated within same container use).

Example

Package installation in Python

Compare with the following. End result is same as before.

```
# syntax=docker/dockerfile:1

#Start from python image
FROM python:3.11.9-slim-bookworm

#Use pip to install matplotlib
RUN pip install matplotlib astropy numpy scipy

RUN python -m pip cache purge
```

- We use “slim” version of OS: only necessary packages for python installation.
- Packages are installed in same command: only one layer.
- Clean pip cache: once packages are installed they can be used, no need to keep them in cache (installation won’t be repeated within same container use).

Example

Comparison of light and heavy images

The difference in memory use can be large.

```
nmalavasi@ga-lt7982 tutorial_material % docker image ls
REPOSITORY      TAG         IMAGE ID        CREATED          SIZE
python_small    latest      724bbbcac001    16 hours ago     612MB
python_large    latest      82da345bcbe4    16 hours ago     1.47GB
nmalavasi@ga-lt7982 tutorial_material %
```

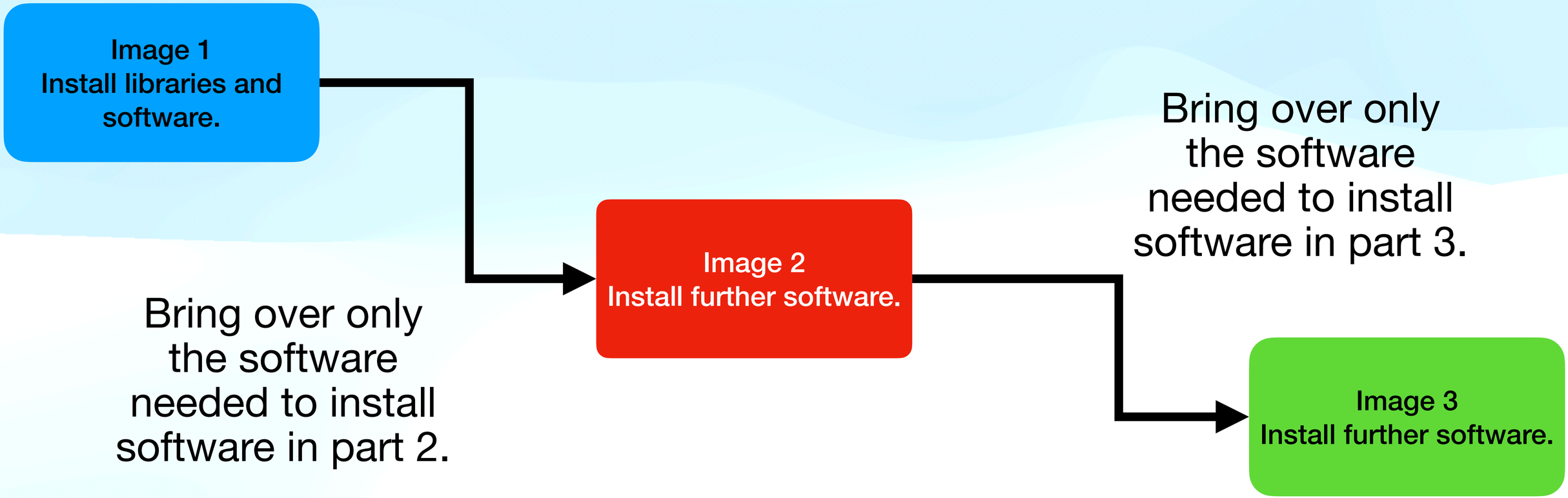
Common practices to optimize containers

Essentially: memory use reduction

- Be mindful of installed code: install only needed libraries, reduce what you install to the bare minimum needed.
- Number of layers increases size: bundle commands when possible (e.g. apt-get update, install, clean).
- Choose correct starting image: use “slim” images if you don’t need full OS functionality.
- Clean up after installation: clear apt-get and pip caches, remove tarballs.

Multi-stage builds

It is possible to use an image as a base for further image creation also within the same Dockerfile.



Only image 3 is kept, others are deleted.

Users > nmalavasi > Desktop > arcane_dockerfiles > OTF_pointing_correction_container > Dockerfile > ...

```
# syntax=docker/dockerfile:1

#Builder image: lighter to download, we use it ot install some stuff- python image
FROM python:3.8.16-slim-bullseye as builder

#Update apt-get and install useful packages: python-is-python3 to symlink python command to python3 command, git, wget, xz-utils, pip3
RUN apt-get update && apt-get -y install wget xz-utils git time

#Download CASA tarball
RUN wget https://casa.nrao.edu/download/distro/casa/release/rhel/casa-6.5.2-26-py3.8.tar.xz

#Extract in home directory
RUN tar -xvf casa-6.5.2-26-py3.8.tar.xz

#Make directory for python packages
RUN mkdir requirements_folder

#Git clone arcane suite
RUN git clone https://github.com/rstofi/arcane_suite.git

#Change folder to the arcane_suite
WORKDIR /arcane_suite

#Set specific commit
RUN git checkout d7d394fc03375b3efa7cc2711834462c812473ad

#Install all requirements
RUN pip3 install --target=/requirements_folder -r ./requirements.txt

```

```
#Use kernsuite image as it contains python-casacore and chgcentre
FROM kernsuite/base:7 as runner

#Install python-casacore and chgcentre
RUN docker-apt-install python3-casacore chgcentre

#Copy what previously installed
COPY --from=builder /requirements_folder /usr/local/src/
COPY --from=builder /requirements_folder/bin /usr/bin/
COPY --from=builder /casa-6.5.2-26-py3.8 /usr/local/src/casa_from_container/
COPY --from=builder /arcane_suite /usr/local/src/arcane_suite/

#Update apt-get and install useful packages: python-is-python3 to symlink python command to python3
command pip3
RUN apt-get update && apt-get -y install python-is-python3 graphviz python3-pydot python3-pip

#Update PYTHONPATH
ENV PYTHONPATH=$PYTHONPATH:/usr/local/src

#Create symbolic link to point to the correct python
RUN ln -s /usr/bin/python3.8 /usr/local/bin/python

#Create symbolic link for casa
RUN ln -s /usr/local/src/casa_from_container/bin/casa /usr/local/bin/casa

#Change folder to the arcane_suite
WORKDIR /usr/local/src/arcane_suite

#Install the arcane arcane_suite
RUN pip3 install -e ./
```

Part five

Apptainer.
Container use in C4P.

Nicola Malavasi - Marie Curie Fellow @ MPE - 16/04/2024

Apptainer

Docker is not the only container engine. Another widely used is Apptainer.

Effectively available only on linux.

Same principle as Docker: can pull/create images, has own version of Dockerfile and syntax, runs processes in isolated way.

Apptainer vs Docker

Apptainer vs Docker

Integration to host

Complete isolation from host

Apptainer vs Docker

Integration to host

- Process is run as current user. USER keyword of Dockerfile is ignored.

Complete isolation from host

- User inside container is completely independent of user outside.

Apptainer vs Docker

Integration to host

- Process is run as current user. USER keyword of Dockerfile is ignored.
- Inside container host file system is still visible (less need for bind mounts).

Complete isolation from host

- User inside container is completely independent of user outside.
- Filesystem inside and outside container are disconnected. Strong need for bind mounts.

Apptainer vs Docker

Integration to host

- Process is run as current user. USER keyword of Dockerfile is ignored.
- Inside container host file system is still visible (less need for bind mounts).
- Image saved as .sif file on host system, easy to transfer.

Complete isolation from host

- User inside container is completely independent of user outside.
- Filesystem inside and outside container are disconnected. Strong need for bind mounts.
- Image saved in non user-friendly place and way.

Apptainer vs Docker

Integration to host

- Process is run as current user. USER keyword of Dockerfile is ignored.
- Inside container host file system is still visible (less need for bind mounts).
- Image saved as .sif file on host system, easy to transfer.
- No root privilege is needed to create image.

Complete isolation from host

- User inside container is completely independent of user outside.
- Filesystem inside and outside container are disconnected. Strong need for bind mounts.
- Image saved in non user-friendly place and way.
- Root privilege is needed to create image.

Apptainer and Docker

Creating Apptainer images and containers from Dockerfiles

- Apptainer can run containers in Docker images
- Apptainer can create images from Dockerfiles

Only the first aspect is needed for C4P.

1. Write Dockerfile
2. Create image from Dockerfile with Docker
3. Have Apptainer read image from Docker daemon and convert it to .sif file
4. Run Apptainer container in Apptainer image

Apptainer and Docker

Creating Apptainer images and containers from Dockerfiles

1. Write Dockerfile
2. Create image from Dockerfile with Docker
3. Have Apptainer read image from Docker daemon and convert it to .sif file
4. Run Apptainer container in Apptainer image apptainer run apptainer exec

apptainer build filename.sif docker-daemon:image_name:image_tag

Apptainer and Docker

Creating Apptainer images and containers from Dockerfiles

1. Write Dockerfile
2. Create image from Dockerfile with Docker
3. Have Apptainer read image from Docker daemon and convert it to .sif file
4. Run Apptainer container in Apptainer image apptainer run apptainer exec

apptainer build filename.sif docker-daemon:image_name:image_tag

Same as Docker

Apptainer and Docker

Creating Apptainer images and containers from Dockerfiles

1. Write Dockerfile
2. Create image from Dockerfile with Docker
3. Have Apptainer read image from Docker daemon and convert it to .sif file
4. Run Apptainer container in Apptainer image apptainer run apptainer exec

apptainer build filename.sif docker-daemon:image_name:image_tag

Same as Docker

.sif file

Apptainer and Docker

Creating Apptainer images and containers from Dockerfiles

1. Write Dockerfile
2. Create image from Dockerfile with Docker
3. Have Apptainer read image from Docker daemon and convert it to .sif file
4. Run Apptainer container in Apptainer image apptainer run apptainer exec

apptainer build | filename.sif | docker-daemon:image_name:image_tag

Same as Docker

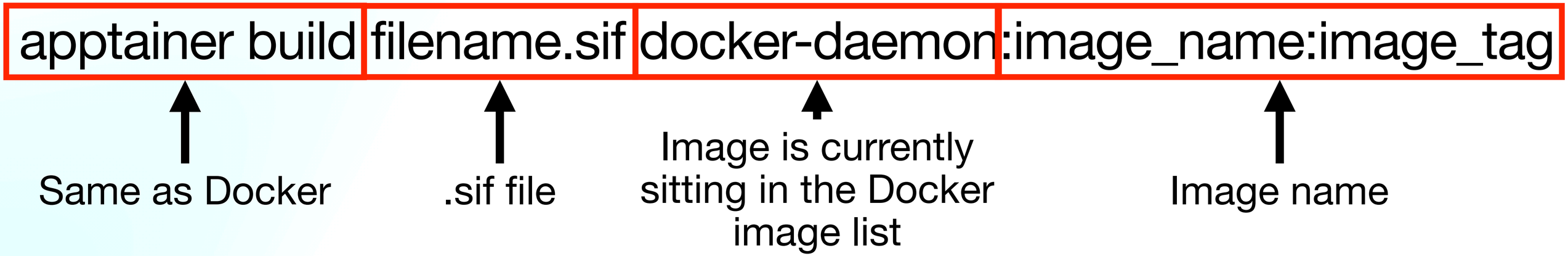
.sif file

Image is currently sitting in the Docker image list

Apptainer and Docker

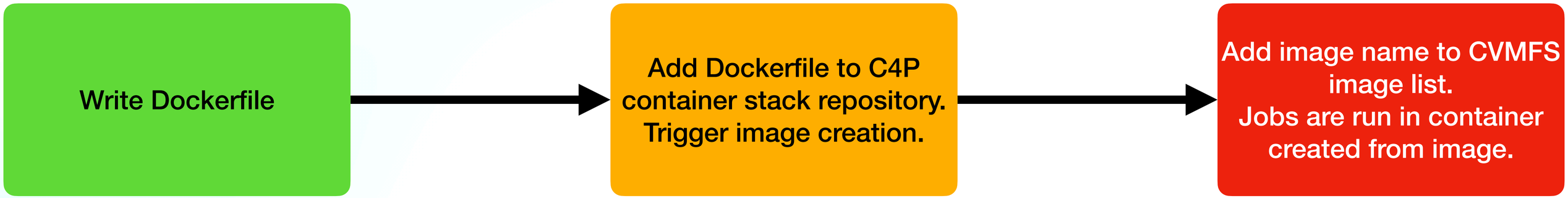
Creating Apptainer images and containers from Dockerfiles

1. Write Dockerfile
2. Create image from Dockerfile with Docker
3. Have Apptainer read image from Docker daemon and convert it to .sif file
4. Run Apptainer container in Apptainer image apptainer run apptainer exec

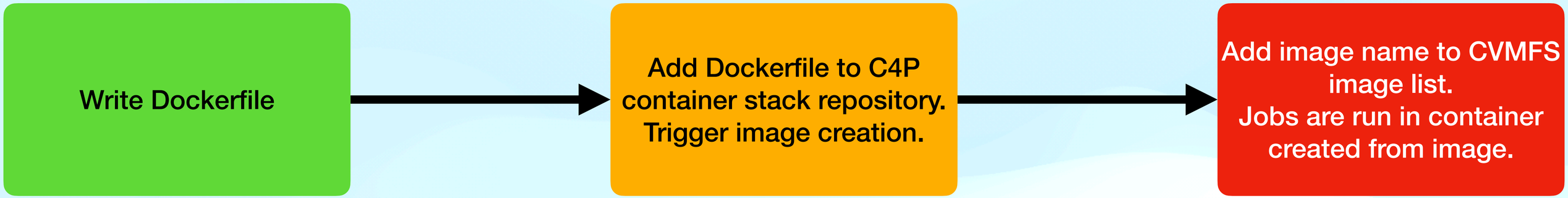


Container use in C4P

- Compute4PUNCH builds images via Docker and runs containers in them via Apptainer.
- Process is automated, only needed input is Dockerfile.



Container use in C4P



- Write Dockerfile, test image creation locally with Docker.
- Make sure it works in Apptainer too (isolation vs integration…).

- Fork correct repository.
- Add Dockerfile.
- Make sure CI/CD pipeline runs to completion.
- Feedback from C4P developers!

- Be mindful of resources used, keep container size small and simple.
- After the image is in CVMFS is very difficult to change it: test before you go through!

From C4P section in PUNCH website

Create your own Singularity Container

Currently the best way to get your software environment running on Compute4PUNCH is to build your own singularity container with your favorite OS and containing your software installation.

You have to do that in the following way.

- Fork https://gitlab-p4n.aip.de/compute4punch/container-stacks into your account
- Create a feature branch and put your Dockerfile and stuff into a meaningful sub-directory
- Submit a merge request, once the CI pipeline runs successful
- Let the Compute4PUNCH team know, when it is ready to be reviewed

Afterwards the new container, which has been to automatically uploaded to the registry at AIP has to be put into https://github.com/cvmfs/images-unpacked.cern.ch via a pull request.

The recipe.yaml inside this repository contains already several available C4P containers. Once merged it will take sometime (~1 hour) until it will appear in
`/cvmfs/unpacked.cern.ch/gitlab-p4n.aip.de\:5005/compute4punch/container-stacks/`
on the login node and you can submit jobs.