Docker Tutorial

Lukas Heinrich 2019/10/24

HEP Computing — Software Distribution

High Energy Physics is a very computing-intensive science

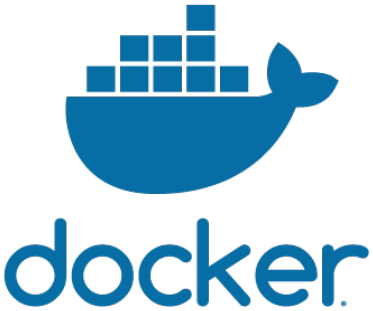
Often we cannot do a computation on a single computer — go distributed, use many computers

Basic Problem: How do these computers get the same software environment?

Docker Introduction

Docker is an open-source linux *container technology* that packages code with **all its dependencies**, into a standardized, portable package, that can run on a **minimally configured ("any")** Host

Docker Image. Think: executable filesystem snapshot



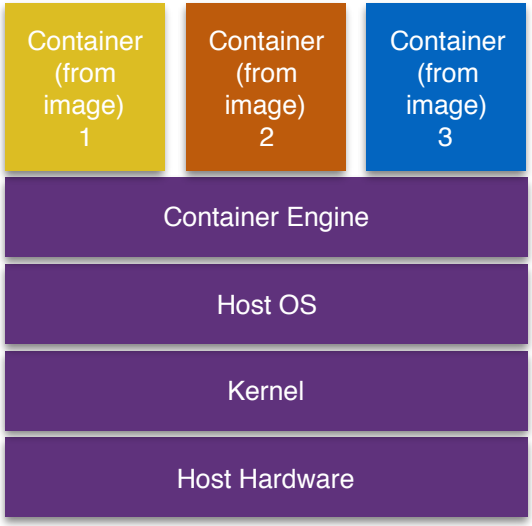
Grew out of need from IT community to efficiently ship & quickly launch applications on cloud computing resources. Supported and used internally by most big cloud players (Amazon, Google, Microsoft, …)

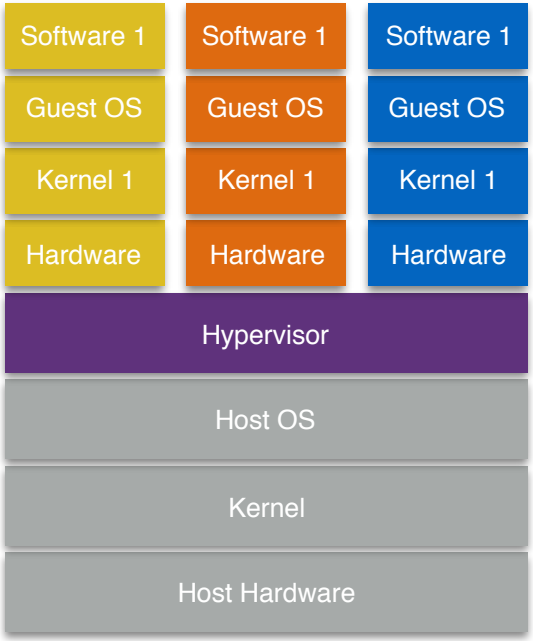
Increasingly used in sciences (not only HEP, but other software-heavy fields like bio-informatics) to package reproducible software environment (bit-by-bit the same environment on large HPC and on your laptop)

Docker Introduction

Docker is **not a Virtual Machine**. VMs simulate both hardware and software. Docker will **share** OS kernel, but isolate different environments (“containers”) w.r.t. data, code, networking, etc.

- containers “start” much quicker than VMs O(ms) vs O(min)
- run as native(*), but isolated processes
- a bit like anaconda envs or virtualenv on steroids

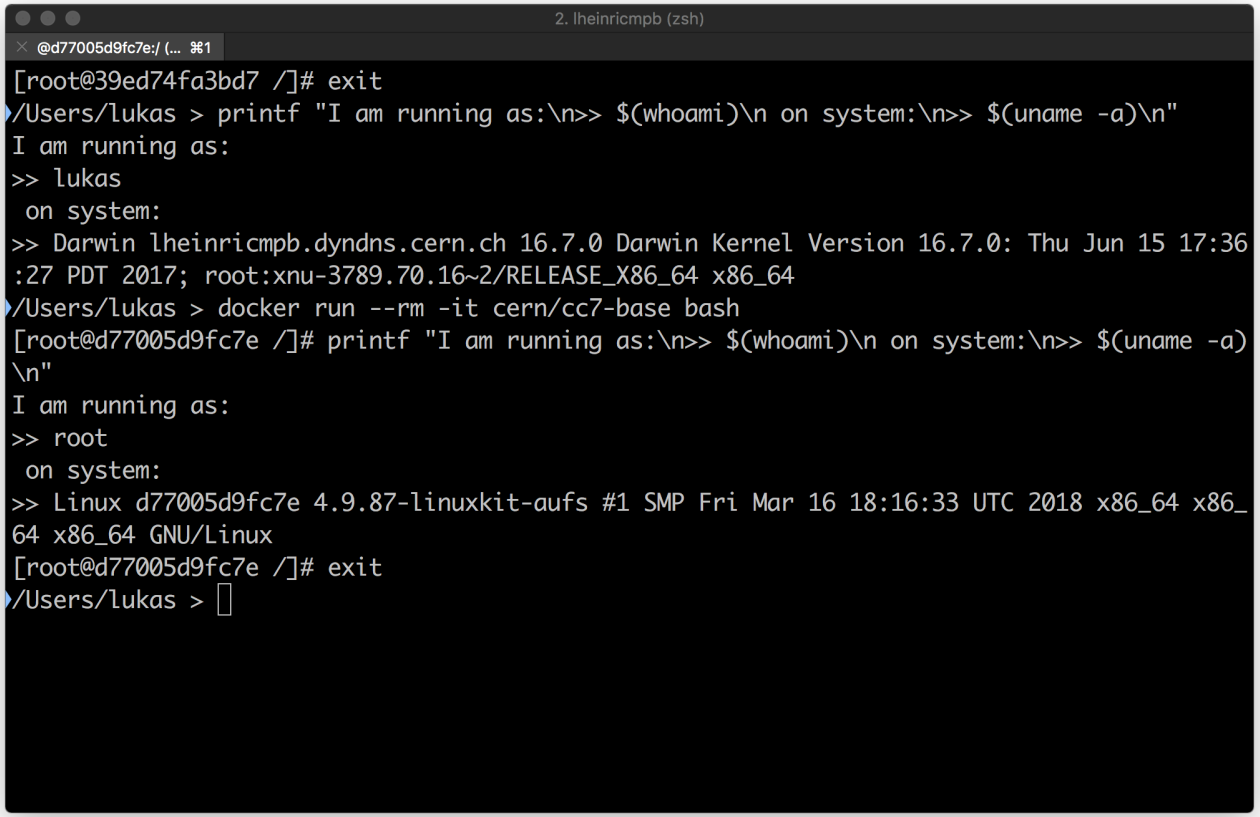




Docker Introduction

Docker Hello World:

dropping into CERN CentOS from my mac within milliseconds



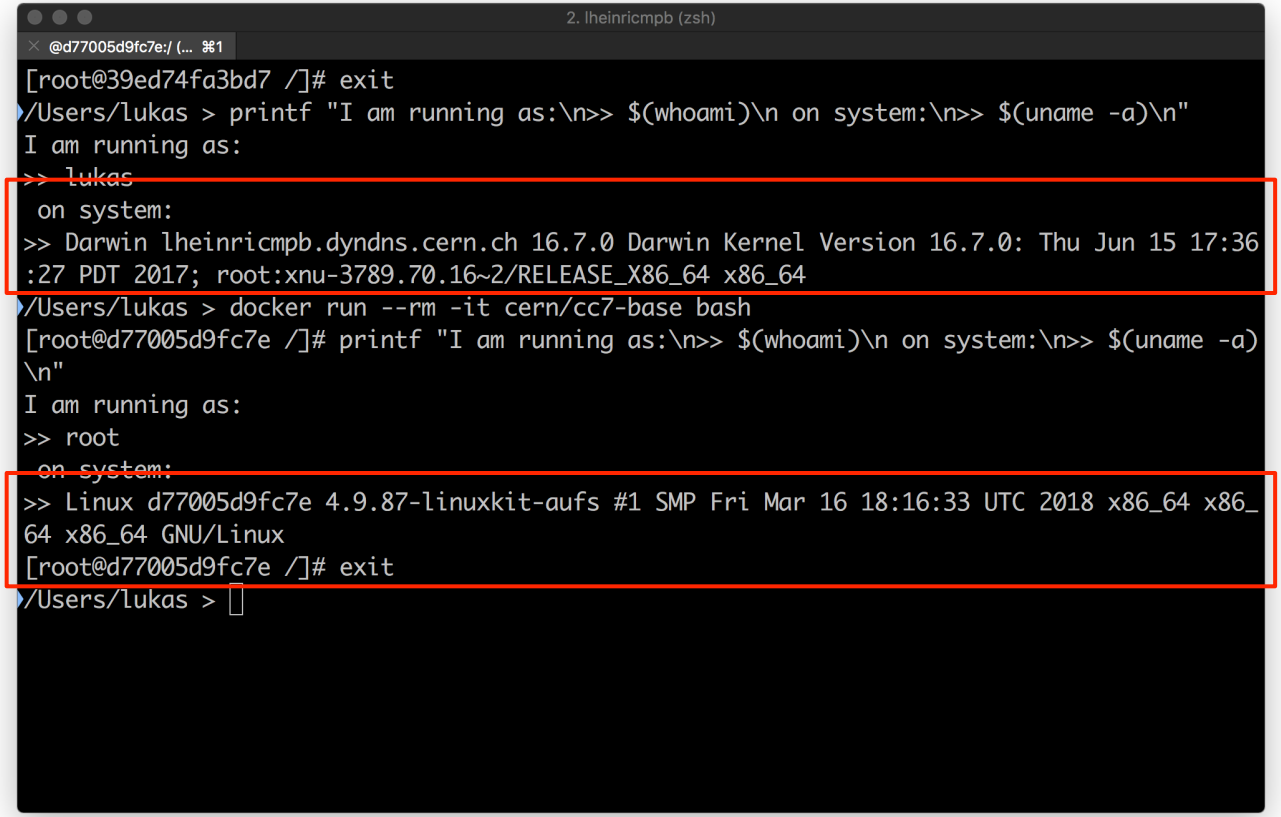
Docker Introduction

Docker Hello World:

dropping into CERN CentOS from my mac within milliseconds

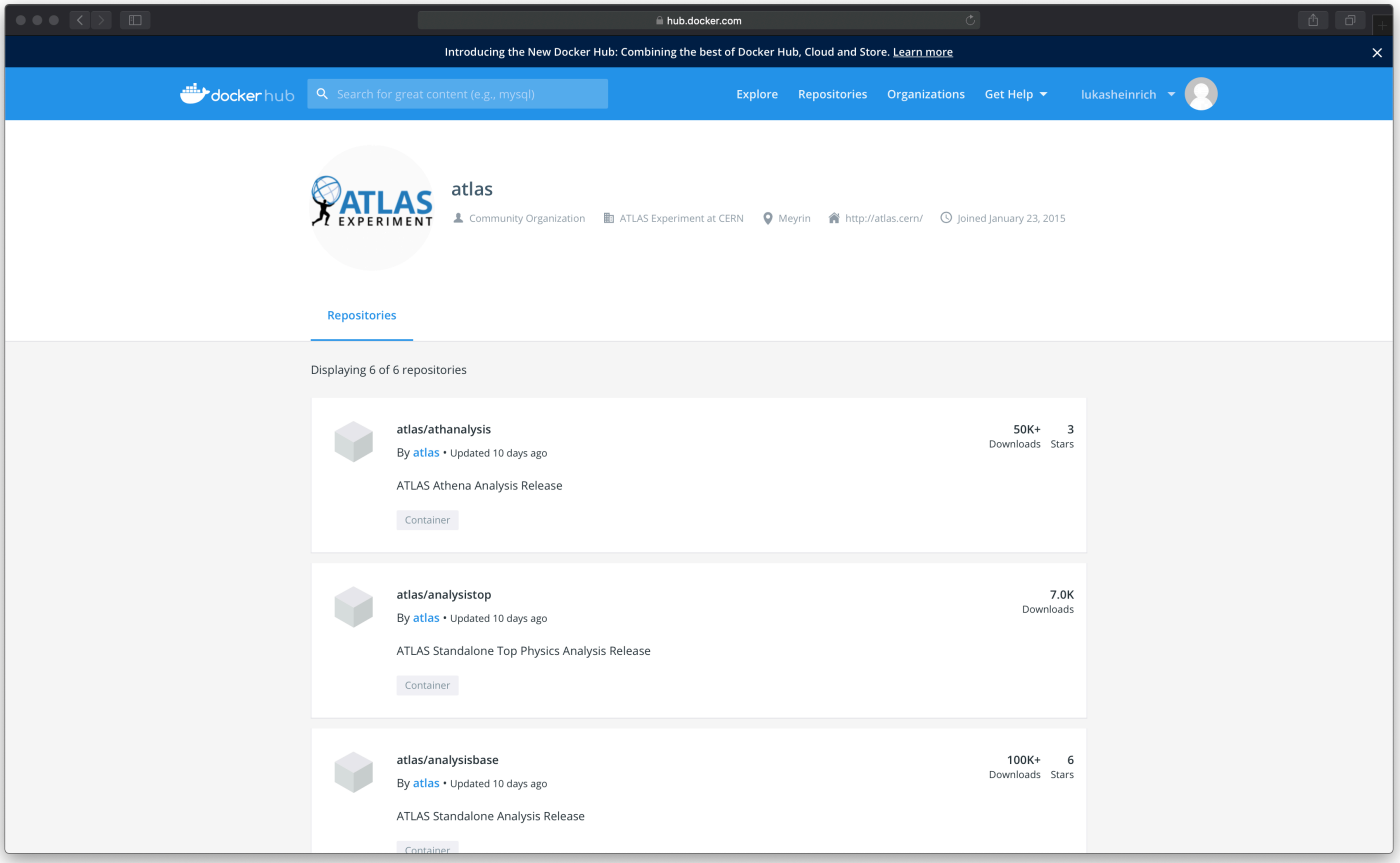
Mac OS X

CentOS7



Official Images

ATLAS provides Docker Images for software releases that you can use



Official Images

Main use-cases for using Docker images in ATLAS

1. **Continuous Integration**
    - With your analysis on GitLab, you can run complex tests on every single commit instantly
    - increase the trust in your analysis code

2. **Analysis Preservation** (package your analysis code together with the release, re-run the analysis later w/ minimal setup/compilation)
    - your analysis is precious, — it should be preserved once it's done
    - building images that include your code, allows re-use later on

3. **Running on Machines without CVMFS** (also on non-Linux OSes)
    - do local development without network access
    - debugging within team across guaranteed identical environments
    - debug you continuous integration jobs locally
    - athena on Macs
    - HEP on the HPCs, Commercial Clouds, Windows (???)

Official Images

Images are available for both AnalysisBase, AthAnalysisBase, AnalysisTop releases. **Note, these images contain only a single release, there is no setupATLAS, etc — kept minimal to reduce size**

Users should only use images published on Docker Hub

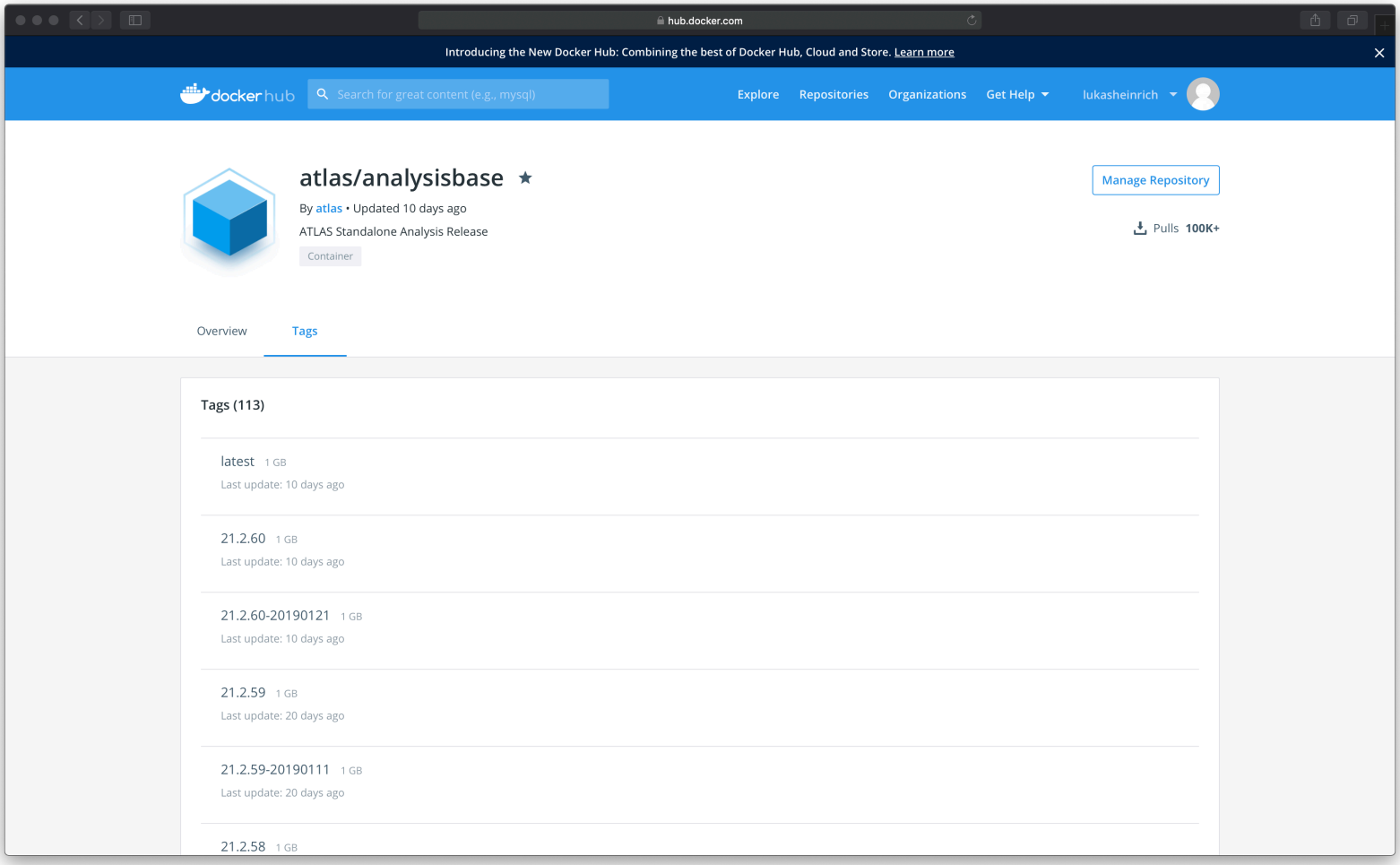
https://hub.docker.com/u/atlas/

See available releases by clicking on image name > Tags

naming convention: (same for atlas/athanalysis)

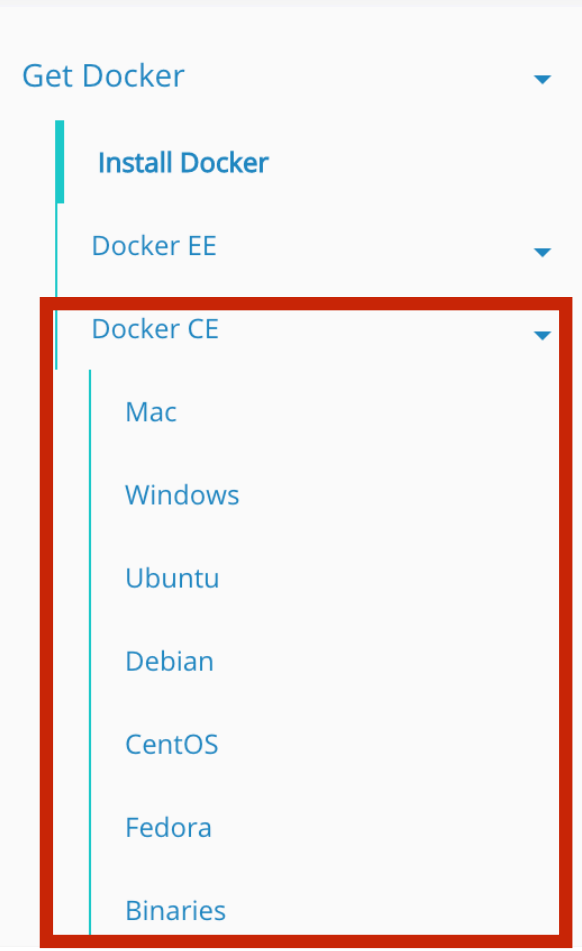
atlas/analysisbase[:<release>-][<date>]	most concrete build (timestamped)
atlas/analysisbase[:<release>]	alias for latest build of that release
atlas/analysisbase = atlas/analysisbase:latest	alias for latest build of latest release

Official Images



Official Images

Installing Docker on your Laptop:



Docker supported on many
Linux distros / Mac / Windows(!)

For Linux, probably you'll find it in your package manager.

For Mac, install Docker for Mac

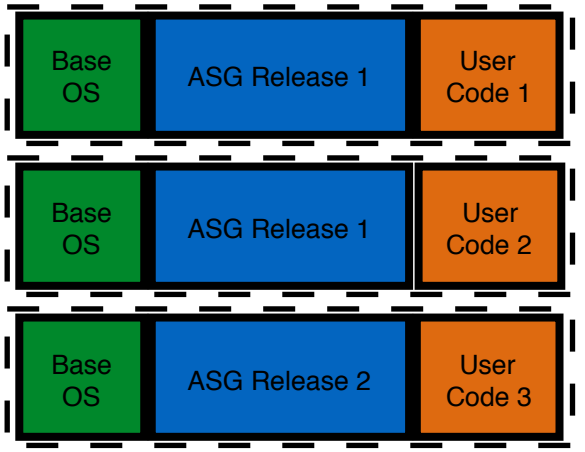
Windows: apparently also works :)

On fast release schedule. Install Docker 18.X

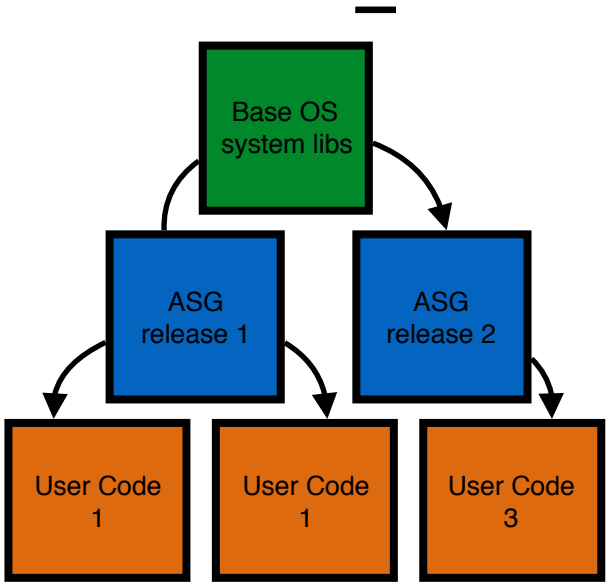
Official Images

Downloading an ATLAS release image:

```
$> docker pull atlas/analysisbase
Using default tag: latest
latest: Pulling from atlas/analysisbase
a75635ba8c7c: Pulling fs layer
c5992bef8b48: Pulling fs layer
9cc33f810ca6: Pulling fs layer
```



Images are ***layered,*** which allows efficient caching. once you have a base release all subsequent updates
(e.g. your analysis code layer) are fast.



Official Images

Starting the container and setting up the release

- remember, there is only one release
- sufficient to do source ~/release_setup.sh

```
$> docker run --rm -it atlas/analysisbase
            _ _____ _      _   ___
           /_\_   _| |    /_\ / __|
          / _ \| | | |__ / _ \\__ \
         /_/ \_\_| |____/_/ \_\___/

This is a self-contained ATLAS AnalysisBase image.
To set up the analysis release of the image, please
execute:

      source /home/atlas/release_setup.sh

[bash][atlas]:~ > source release_setup.sh
Configured GCC from: /opt/lcg/gcc/6.2.0/x86_64-slc6
Configured AnalysisBase from: /usr/AnalysisBase/21.2.10/InstallArea/x86_64-slc6-gcc62-opt
[bash][atlas AnalysisBase-21.2.10]:~ >
```

Official Images

Docker Images are meant to be ephemeral. The environment get destroyed after you exit the shell. Of course, we want to keep some of the work we did in the container

Where is my data ?

While a Docker image is it's own *entire filesystem* to stay reproducible, you can make outside data available

```
$> docker run --rm -it -v /path/to/my/data:/data atlas/analysisbase bash
container> ls -lrt /data
```

Official Images

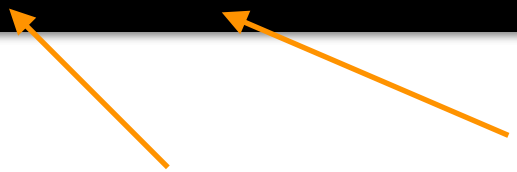
Common Pattern:

- Checkout the source code you are developing on your Laptop
- bind mount source into the container.
- compile using in-container shell
- run your test jobs on local or remote data
- edit source using your laptops Editor (e.g. on a Mac you can use a native Mac editor)
- exit container when done, code is still there, environment is gone

Official Images

Example from XAMPP framework developers:

```
host> mkdir MyProject
host> mkdir build run
host> git clone ssh://git@gitlab.cern.ch:7999/atlas-mpp-xampp/XAMPPmonoH.git --recursive source
host> docker run --rm -it -v $PWD:$PWD -w $PWD atlas/athanalysis
container>
```



makes source code available in container

changes directory immediatly to your source, so you are exactly where you started

Now compile the code

```
container> source ~/release_setup.sh
container> cd ../build && cmake ../source && make -j4
```

useful alias

```
alias docker_here='docker run --rm -it -w $PWD -v $PWD:
$PWD'
host> docker_here atlas/analysisbase
```

Official Images

The Images have Kerberos installed, so you can access EOS files that are not on your laptop via XrootD

```
container> kinit <your CERN Username>
container> source ~/release_setup.sh
container> root -b -l
root [0] TFile::Open("root://eosatlas.cern.ch///eos/...")
```

Also scp etc work as normal inside the container

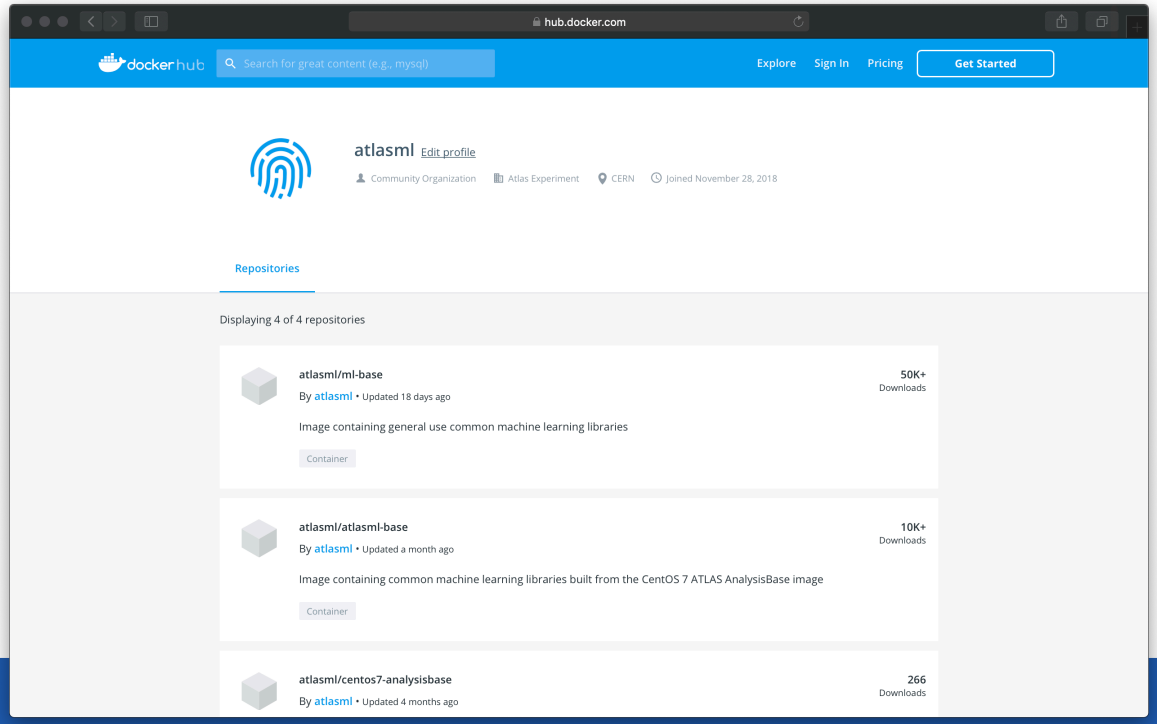
Machine Learning Images

One interesting use-case for containers is Machine Learning.

- The Software Stack is very different from the traditional one

The ATLAS Machine Learning Group is also provides images under

`atlasml/*`



Continuous Integration

Major use-case: Continuous Integration. Very easy to setup using the images

If you do one thing with the docker images make it be using it in continuous integration of your analysis (can do it in the hands-on, takes ~5 minutes)

HWW Analysis (Higgs)

```
build:
  image: atlas/analysisbase:21.2.23
  stage: build
  script:
    - source /home/atlas/release_setup.sh
    - mkdir ../build
    - cd ../build
    - cmake ../CAFExample
    - make -j4
    - cd ../
    - source build/*/setup.sh
```

XAMMP monoH (Exotics)

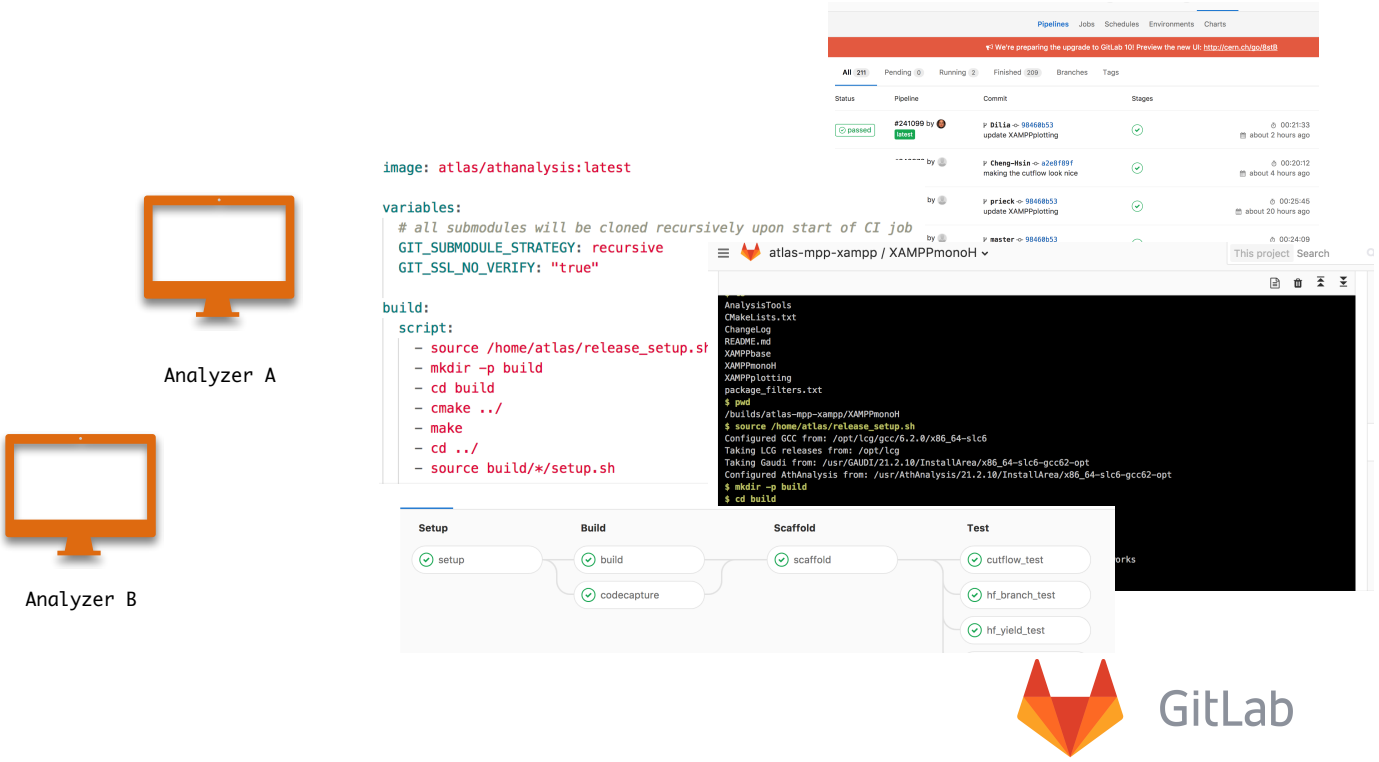
```
build:
  stage: build
  image: atlas/athanalysis:latest
  script:
    # check current working environment
    - ls
    - pwd
    # setup athena release
    - source /home/atlas/release_setup.sh
    # setup working space and build the code
    - mkdir -p build
    - cd build
    - cmake ../
    - make
    - cd ../
    - source build/*/setup.sh
```

Multi-Bjet (SUSY Analysis)

```
.analysis_image: &image
  image: atlas/analysisbase:21.2.18
  tags:
    - cvmfs
  before_script:
    - pwd
    - ls
    - echo "Project Directory    ${CI_PROJECT_DIR}"
    - echo "Source Directory     ${SRC_DIR_ABS}"
    - echo "      Directory Name ${SRC_DIR}"
    - echo "Build Directory      ${BUILD_DIR_ABS}"
    - echo "      Directory Name ${BUILD_DIR}"
    - source /home/atlas/release_setup.sh
    - echo $SERVICE_PASS | kinit $CERN_USER
```

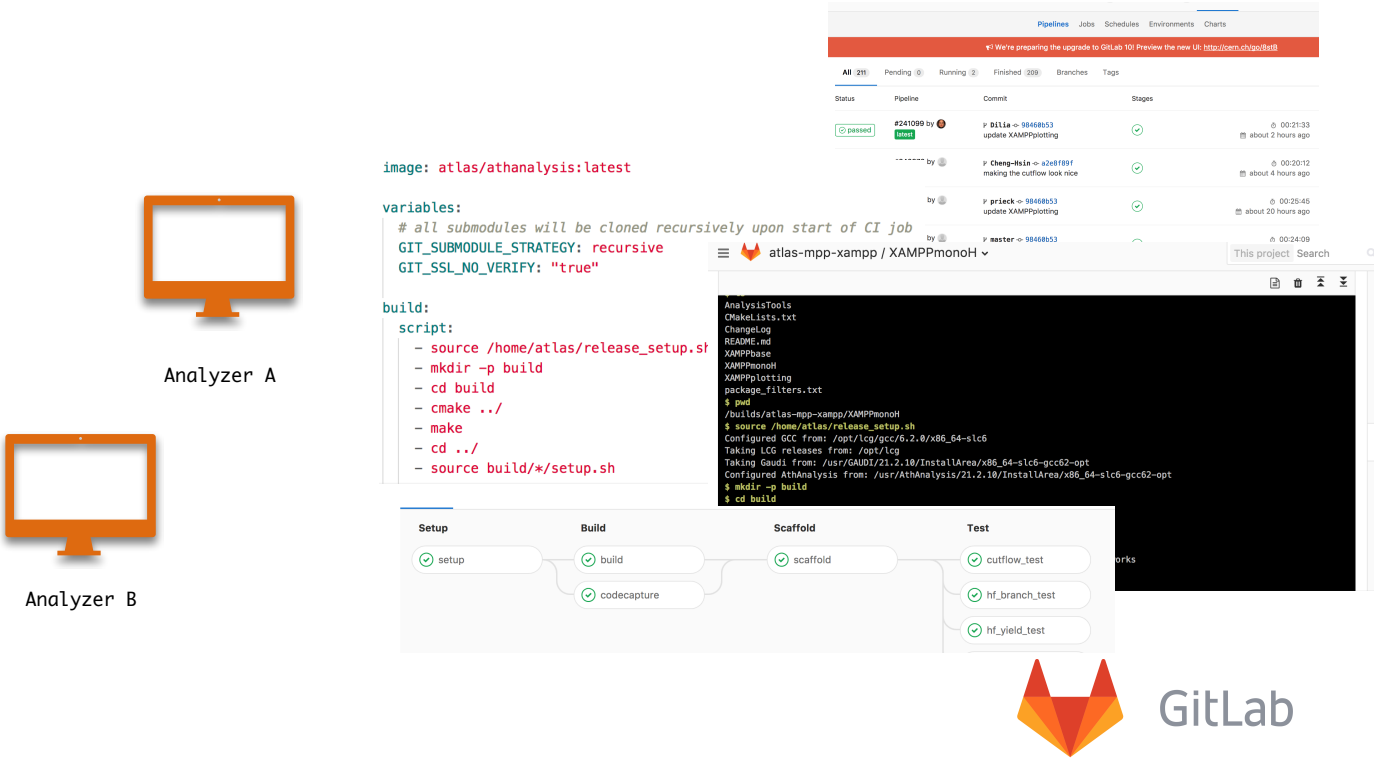
Continuous Integration

- immediate feedback on new code (test compilation, test simple single-file jobs, etc)
- using atlas/* iages makes it easy to reproduce/debug failures locally since you get exactly the same environment, without always going through CI
- sets up you analysis nicely for analysis preservation



Continuous Integration

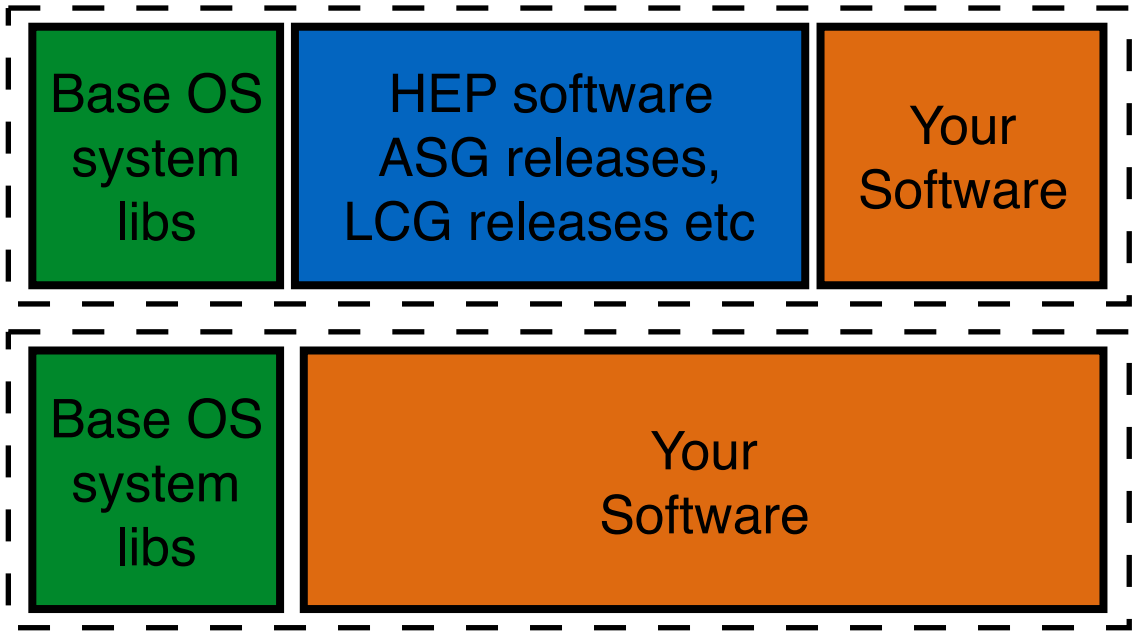
- immediate feedback on new code (test compilation, test simple single-file jobs, etc)
- using atlas/* iages makes it easy to reproduce/debug failures locally since you get exactly the same environment, without always going through CI
- sets up you analysis nicely for analysis preservation



Building your Own Images

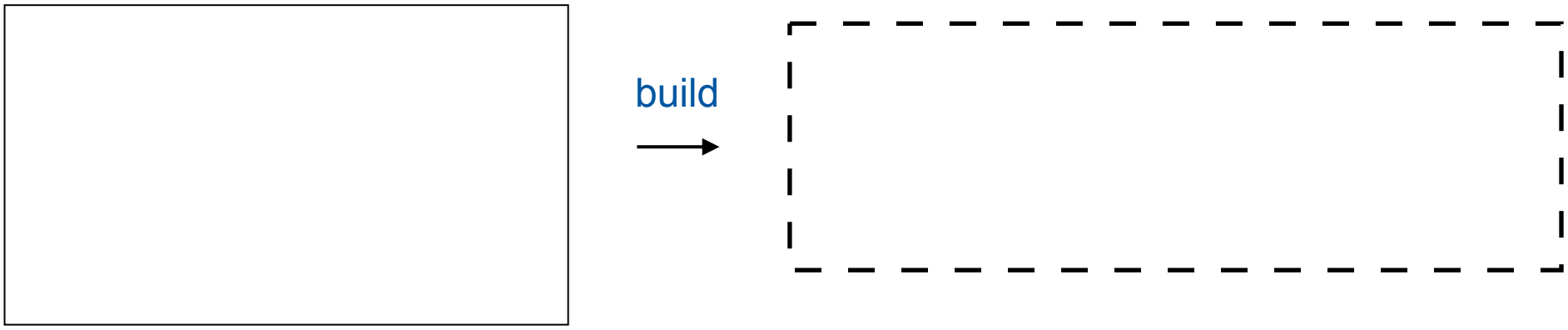
Docker gives you freedom to assemble your favourite software environment and use it everywhere in exactly the same way.

You can build your own Docker Images — by building on top of some existing base collection of software (provided by ATLAS or someone else)



The easiest way to build an Image is using a "`Dockerfile`"

- a text file with instructions what to do.. starting from a "Base Image"

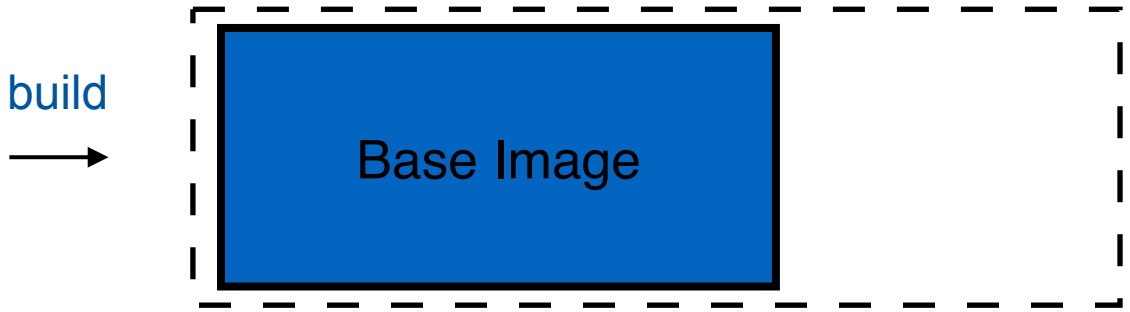



```
docker build -t <myimage> -f Dockerfile /path/to/source
```

The easiest way to build an Image is using a "`Dockerfile`"

- a text file with instructions what to do.. starting from a "Base Image"

```
FROM <baseimage>
```

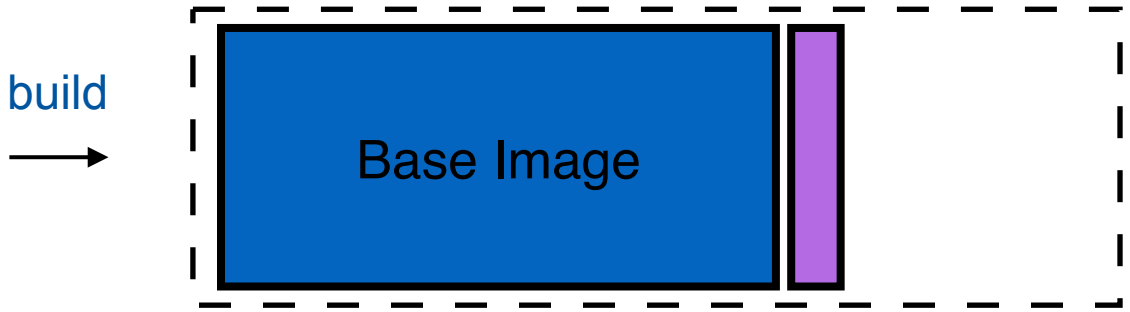



```
docker build -t <myimage> -f Dockerfile /path/to/source
```

The easiest way to build an Image is using a "Dockerfile"

- a text file with instructions what to do.. starting from a "Base Image"

```
FROM <baseimage>
RUN <instruction 1>
```

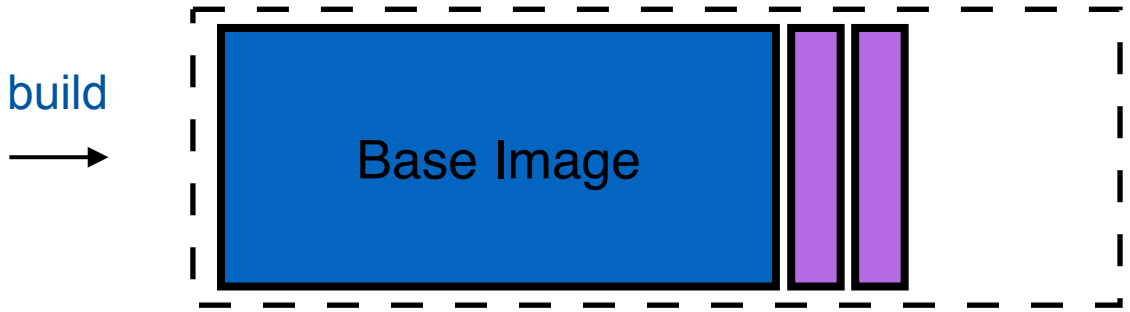



```
docker build -t <myimage> -f Dockerfile /path/to/source
```

The easiest way to build an Image is using a "Dockerfile"

- a text file with instructions what to do.. starting from a "Base Image"

```
FROM <baseimage>
RUN <instruction 1>
RUN <instruction 2>
```

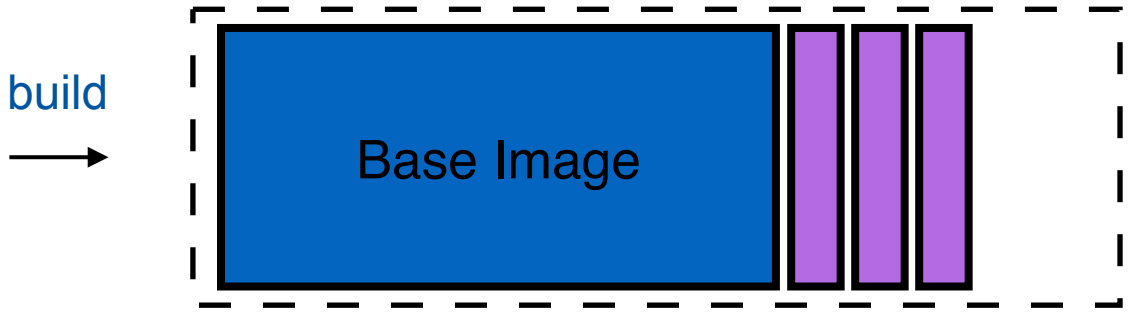



```
docker build -t <myimage> -f Dockerfile /path/to/source
```

The easiest way to build an Image is using a "Dockerfile"

- a text file with instructions what to do.. starting from a "Base Image"

```
FROM <baseimage>
RUN <instruction 1>
RUN <instruction 2>
RUN <instruction 3>
```

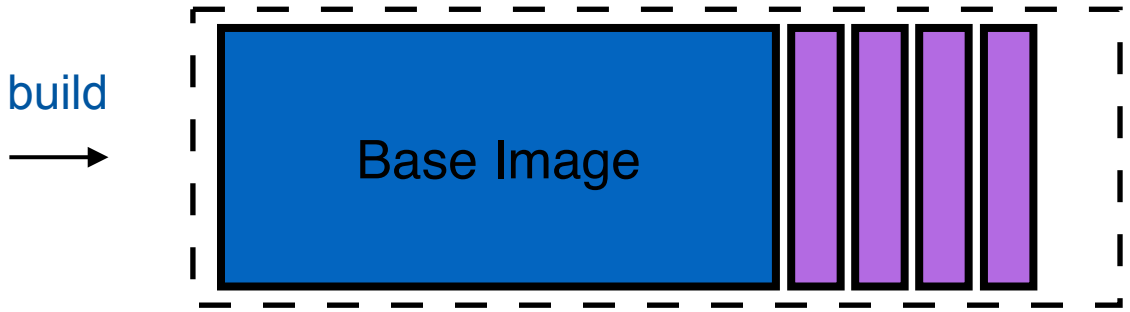



```
docker build -t <myimage> -f Dockerfile /path/to/source
```

The easiest way to build an Image is using a "Dockerfile"

- a text file with instructions what to do.. starting from a "Base Image"

```
FROM <baseimage>
RUN <instruction 1>
RUN <instruction 2>
RUN <instruction 3>
COPY <source> <target>
```

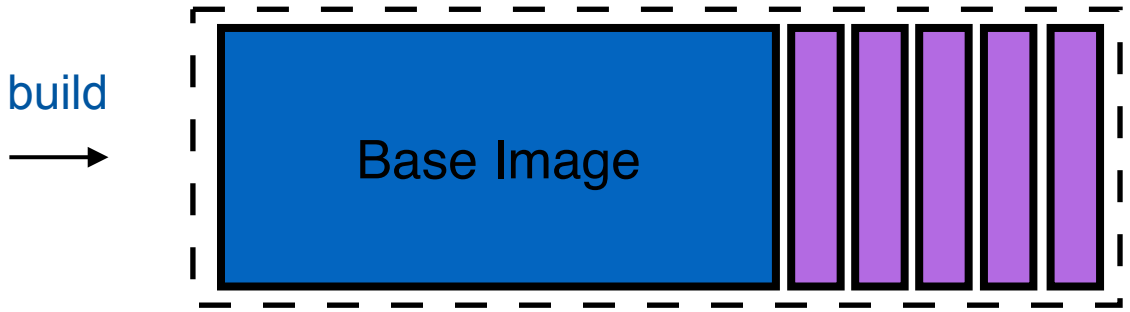



```
docker build -t <myimage> -f Dockerfile /path/to/source
```

The easiest way to build an Image is using a "Dockerfile"

- a text file with instructions what to do.. starting from a "Base Image"

```
FROM <baseimage>
RUN <instruction 1>
RUN <instruction 2>
RUN <instruction 3>
COPY <source> <target>
RUN <instruction 4>
```



- once you have a Dockerfile you can build an image (rel. to a source context)

```
docker build -t <myimage> -f Dockerfile /path/to/source
```

- and archive it in an "image registry"

```
docker push <myimage>
```

Example: MBJ Event Selection

```
FROM atlas/analysisbase:21.2.51
ADD . /MBJ/
WORKDIR /MBJ/build
RUN source ~/release_setup.sh &&  \
    sudo chown -R atlas /MBJ && \
    cmake ../src && \
    make -j4
```

Continuous Integration

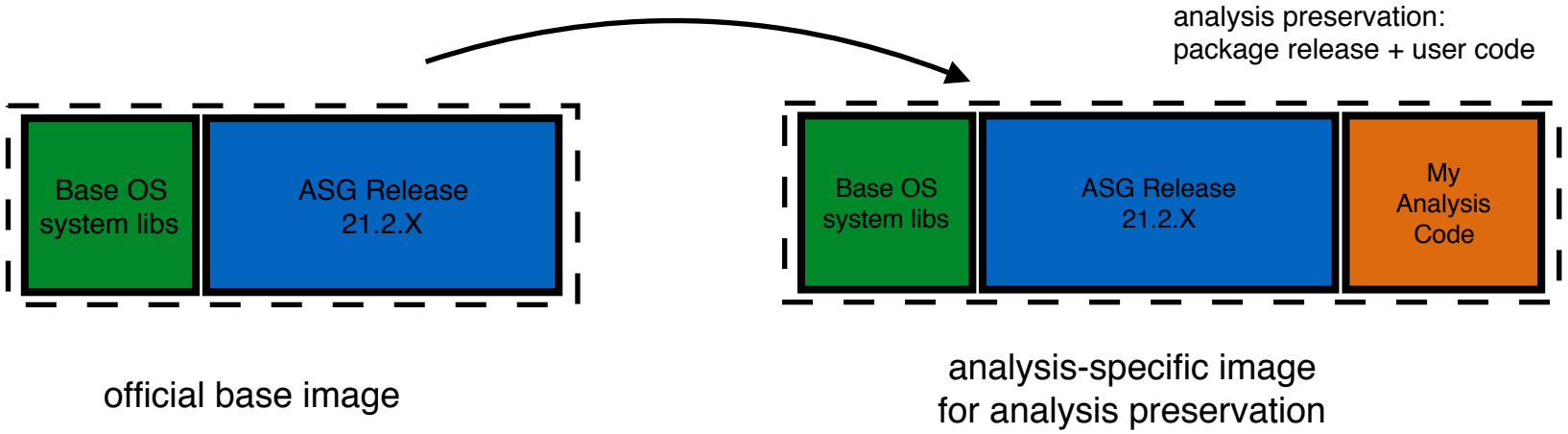
Similar to CI, but slightly different.

Continuous Integration:

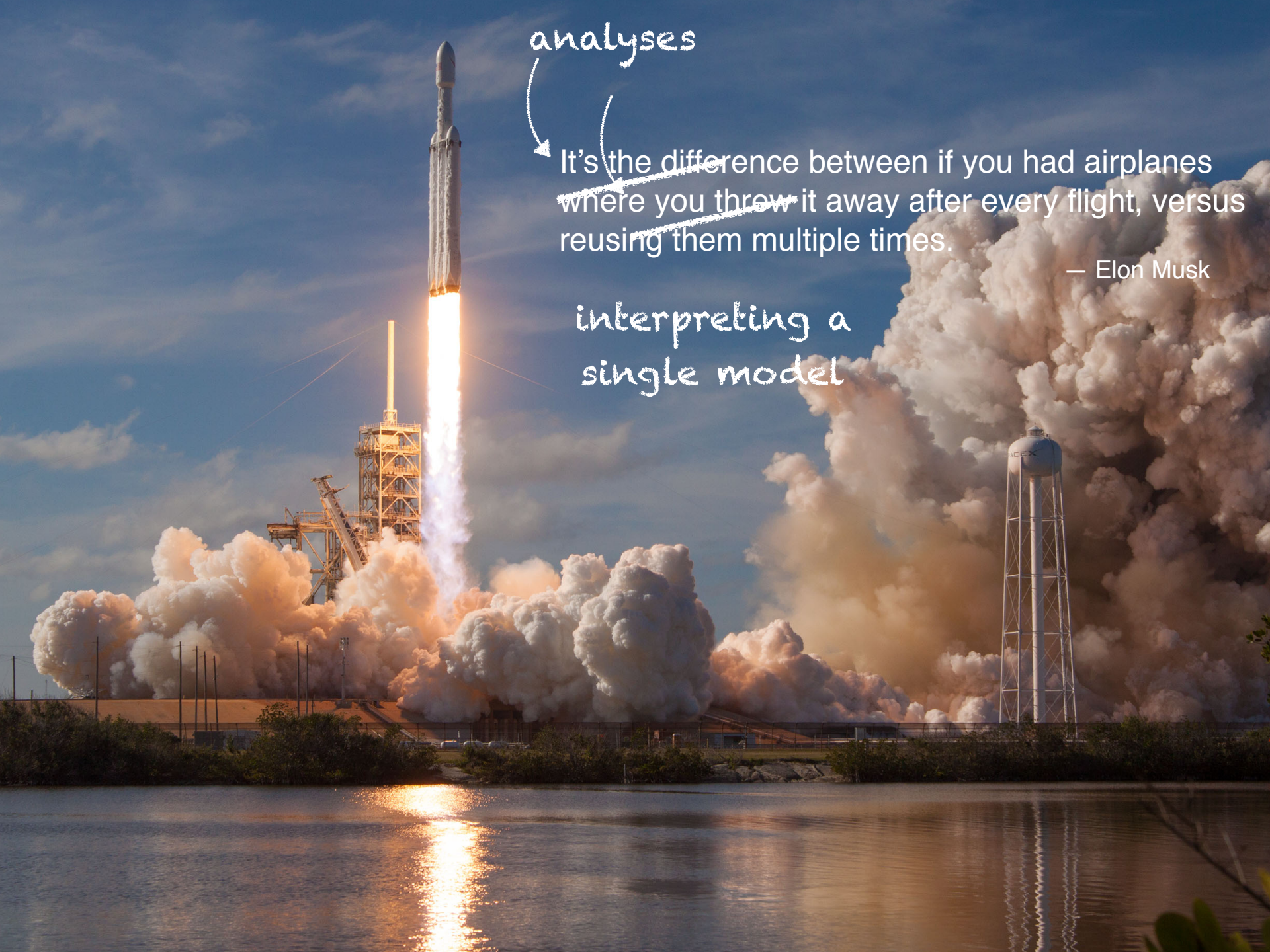
- job runs in official image (atlas/*), analysis code added in dynamically.

Analysis Preservation

- build a **new image** on top of the official image that bakes in the analysis code into the image. **Makes you analysis available to others.**

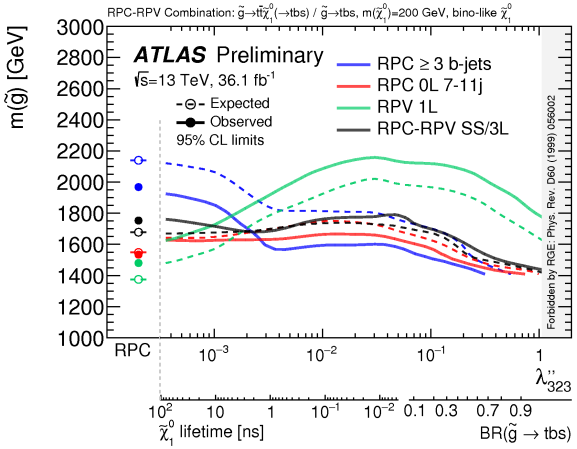



It’s the difference between if you had airplanes where you threw it away after every flight, versus reusing them multiple times.
— Elon Musk

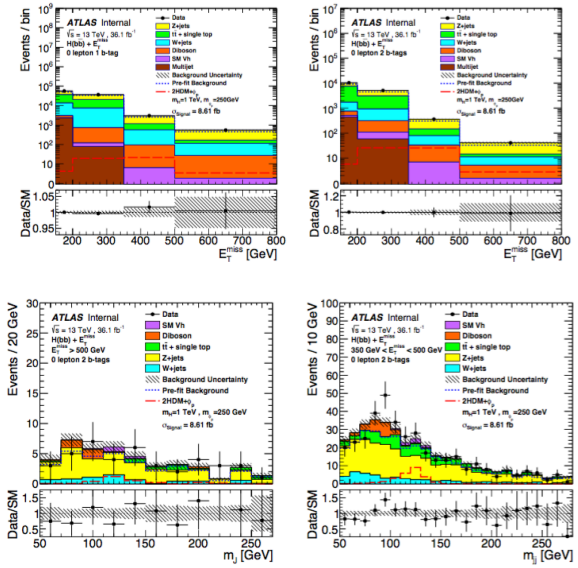

analyses
It's the difference between if you had airplanes where you throw it away after every flight, versus reusing them multiple times.
— Elon Musk
interpreting a single model

Analysis Preservation

- End of Run-2 / LS2 will see a lot of activity on summary papers, reinterpretations etc of analysis based on the full dataset
- Requests for re-running the analysis on new samples **will come,** so prepare now to make you life easier later.
- **Approval Requirement in Exotics to preserve your analysis with Docker Images**



SUSY RPV/RPC analysis
(recast of ~10 analyses)

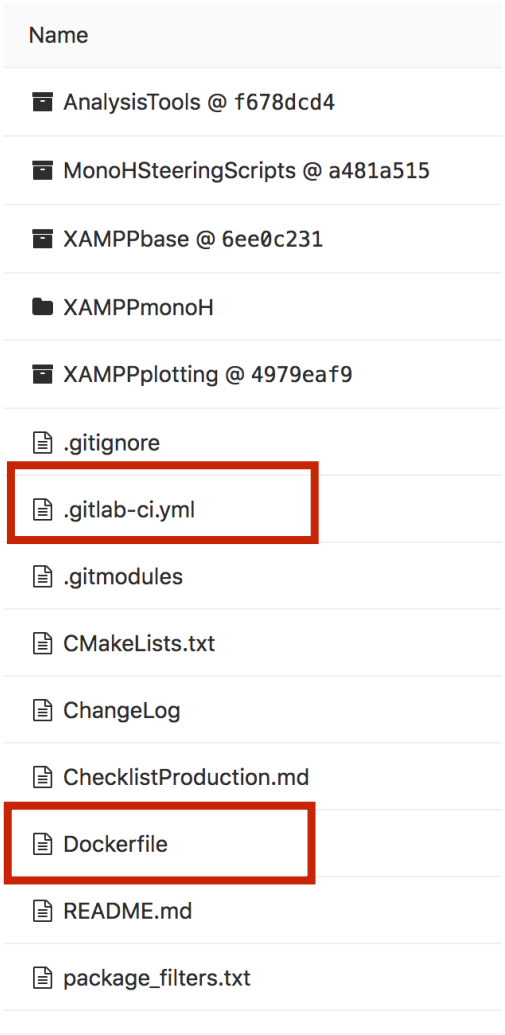


Exotics DM Summary

Analysis Preservation

What you need to do:

- add a text file named “Dockerfile” into the root of your repo
- add “image build” stage into your .gitlab-ci.yml



Analysis Preservation

To preserve your analysis you can almost copy your gitlab-ci job
Minimal Setup:

- add small "docker build" section got .gitlab-ci.yml
- add file called “Dockerfile” to your repo that has build proce

Example: monoH analysis

```yaml
build:
  stage: build
  image: atlas/athanalysis:latest
  script:
    # check current working environment
    - ls
    - pwd
    # setup athena release
    - source /home/atlas/release_setup.sh
    # setup working space and build the code
    - mkdir -p build
    - cd build
    - cmake ../
    - make
    - cd ../
    - source build/*/setup.sh
```

```yaml
build_image:
  stage: docker
  tags:
    - docker-image-build
  script:
    - ignore
```

instruct

Dockerfile 190 Bytes

```
1  FROM atlas/athanalysis:21.2.23
2  ADD . /xampp/XAMPPmonoH
3  WORKDIR /xampp/build
4  RUN source ~/release_setup.sh &&  \
5      sudo chown -R atlas /xampp && \
6      cmake ../XAMPPmonoH && \
7      make -j1
```

Analysis Preservation

To preserve your analysis you can almost copy your gitlab-ci job
Minimal Setup:

- add small "docker build" section got .gitlab-ci.yml
- add file called “Dockerfile” to your repo that has build proce

Example: monoH analysis

```yaml
build:
  stage: build
  image: atlas/athanalysis:latest
  script:
    # check current working environment
    - ls
    - pwd
    # setup athena release
    - source /home/atlas/release_setup.sh
    # setup working space and build the code
    - mkdir -p build
    - cd build
    - cmake ../
    - make
    - cd ../
    - source build/*/setup.sh
```

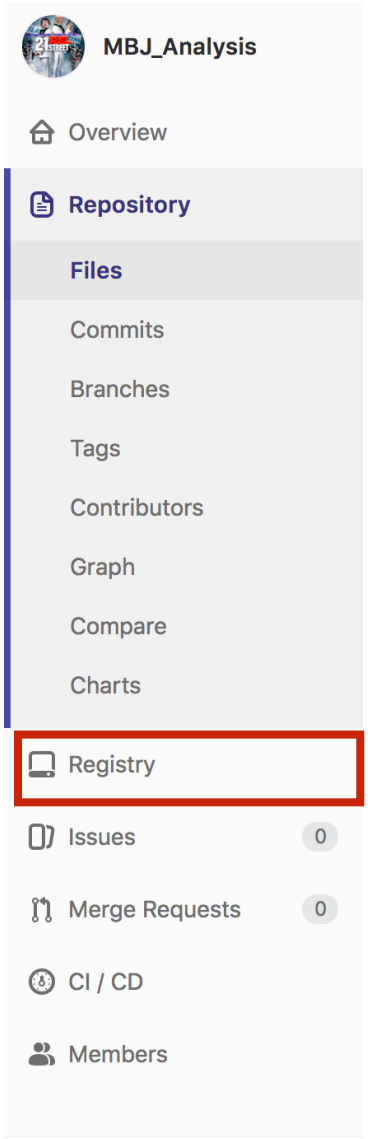
```yaml
build_image:
  stage: docker
  tags:
    - docker-image-build
  script:
    - ignore
```

instruct

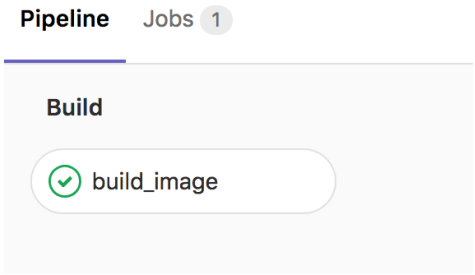
Dockerfile 190 Bytes

```
1  FROM atlas/athanalysis:21.2.23
2  ADD . /xampp/XAMPPmonoH
3  WORKDIR /xampp/build
4  RUN source ~/release_setup.sh &&  \
5      sudo chown -R atlas /xampp && \
6      cmake ../XAMPPmonoH && \
7      make -j1
```

Analysis Preservation



pipeline should now include “build_image” job



your preserved, compiled analysis code

Conclusion

ATLAS provides Docker images to make analysis development easier

- **reproducible software environments**
- **easy continuous integration**
- **easy analysis preservation**

If you have 10 minutes (in the hands-on?)

- **add a .gitlab-ci.yml file and test your analysis**
- **add a Dockerfile to preserve your analysis code**