

SKA data challenge workflow + container setup

- Using containers is all about **reproducibility**
- That means your code and workflow also need to be reproducible
- Shipping a container with code that isn't documented and can't be run or understood isn't helpful
- I'll touch on a few things to think about whilst using containers in your workflows

SKA data challenge workflow + container setup

- Last week we saw how to run a full science workflow from inside a container

Link in the program: <https://www.youtube.com/watch?v=pcNzLk7Ohj8>

- This involved primary beam correction, source-finding, training a machine learning model, source classification and scoring results.
- Hopefully you've had a play with that - the best way to learn about containers is to practise using them

```

67 if __name__ == "__main__":
68     """
69     Run through a simple analysis workflow to solve SDC1
70
71     1) Preprocess images (correct PB) and crop out the training area for
72         building ML model
73     2) Find sources in the PB-corrected training images
74     3) Train a classifier for each band to predict the class of each source
75     4) Find sources in the full PB-corrected image
76     5) Predict the class of each source
77     6) Calculate the score for each image band, and write out a short report
78     """
79     time_0 = time()
80     # 1) Create in-memory representation of image and preprocess
81     print("\nStep 1: Preprocessing; elapsed: {:.2f}s".format(time() - time_0))
82     sdc1_image_list = []
83     for freq in FREQS:
84         new_image = Sdc1Image(freq, image_path(freq), pb_path(freq))
85         new_image.preprocess()
86         sdc1_image_list.append(new_image)
87
88     # In data/images, we now have PB-corrected and training images for each band
89
90     # 2) Source finding (training):
91     print("\nStep 2: Source finding (train); elapsed: {:.2f}s".format(time() - time_0))
92     sources_training = {}
93     for sdc1_image in sdc1_image_list:
94         source_finder = SourceFinder(sdc1_image.train)
95         sl_df = source_finder.run()
96         sources_training[sdc1_image.freq] = sl_df

```

SKA data challenge workflow + container setup

- Best practises in code structures
- Best practises in Gitlab repo
- Using make files
- Using aliases
- Using .dockerignore
- Using unit tests

The screenshot shows the GitLab web interface for a repository named 'SDC1 Solution'. The left sidebar contains navigation links for Project information, Repository, Files, Commits, Branches, Tags, Contributors, Graph, Compare, Locked Files, Issues (0), Jira, Merge requests (0), CI/CD, Deployments, Monitor, Packages & Regist..., Analytics, Wiki, and Snippets. The main content area shows the repository's file structure and a list of files with their commit history.

Repository: SDC1 Solution

Branch: master | sdc1-solution

History | Find file | Clone

Adding Dockerfile for CERN DLaaS notebook environment
James Collinson authored 2 months ago

Name	Last commit	Last update
binder	Correcting link in README and add...	4 months ago
docs	ESCAP-161 Set RF kwargs in solutio...	1 year ago
etc	ESCAP-161 Set RF kwargs in solutio...	1 year ago
scripts	Fix bug: create sample_image dir if ...	1 year ago
ska	Adding Dockerfile for CERN DLaaS ...	2 months ago
tests	ESCAP-161 Fixing bugs, adding mo...	1 year ago
.dockerignore	Adding Dockerfile for CERN DLaaS ...	2 months ago
.flake8	ESCAP-122 Adding docs	1 year ago
.gitignore	ESCAP-156 Adding cropped truth c...	1 year ago
LICENSE	Adding license	4 months ago
Makefile	added build/testing architecture	1 year ago
README.md	Adding Zenodo DOI badge to REA...	4 months ago
analyse_sample.ipynb	Remove timing statements from no...	4 months ago
codemeta.json	Adding codemeta.json	4 months ago
pipeline.py	Update pipeline.py	1 year ago
singularity_py3.def	Update singularity_py3.def	1 year ago

README.md

Folder structures

- Your repo is probably capable of doing a variety of things, so organise folders is important
- How you organise things is subjective, but hopefully intuitive
- `/docs` is self explanatory. We implement automatic documentation using the python package Sphinx. This means docs are updated automatically as you update your code
- `/ska` contains the actual code and the docker build files required to make the container
- `/scripts` contains ready to run scripts that could do a variety of things. They are lightweight and easy to interpret, since all the functions are in the `/ska` folder. Users will look at these rather than the code in `/ska`
- `/tests` is used to run unit-tests. These are lightweight functions that ensure your code is working, without having to do any heavy lifting such as accessing large data sets
- `/etc` contains additional code that is not part of your software, but is needed for additional functionality and ease of use, such as aliases

The screenshot shows the GitLab interface for a repository named 'SDC1 Solution'. The left sidebar lists various repository features like Project information, Repository, Files, Commits, Branches, Tags, Contributors, Graph, Compare, Locked Files, Issues, Jira, Merge requests, CI/CD, Deployments, Monitor, Packages & Regist..., Analytics, Wiki, and Snippets. The main content area shows the 'master' branch selected, with a commit history table. The commit history table lists files and their last commit details.

Name	Last commit	Last update
binder	Correcting link in README and add...	4 months ago
docs	ESCAP-161 Set RF kwargs in solutio...	1 year ago
etc	ESCAP-161 Set RF kwargs in solutio...	1 year ago
scripts	Fix bug: create sample_image dir if ...	1 year ago
ska	Adding Dockerfile for CERN DLaaS ...	2 months ago
tests	ESCAP-161 Fixing bugs, adding mo...	1 year ago
.dockerignore	Adding Dockerfile for CERN DLaaS ...	2 months ago
.flake8	ESCAP-122 Adding docs	1 year ago
.gitignore	ESCAP-156 Adding cropped truth c...	1 year ago
LICENSE	Adding license	4 months ago
Makefile	added build/testing architecture	1 year ago
README.md	Adding Zenodo DOI badge to REA...	4 months ago
analyse_sample.ipynb	Remove timing statements from no...	4 months ago
codemeta.json	Adding codemeta.json	4 months ago
pipeline.py	Update pipeline.py	1 year ago
singularity_py3.def	Update singularity_py3.def	1 year ago

Below the table, there is a section for 'README.md' which is currently empty.

Main code folder - /ska/sdc1

- All about splitting your code up into functions related to specific tasks
- Users don't need to come in here and look at things unless they are adapting or adding functionality
- We include the Dockerfile in here
- We also have a version that runs in a Jupyter notebook: /ska/notebook

master sdc1-solution / ska / sdc1 History Find file Clone

Replacing montage-wrappers with newer MontagePy methods
James Collinson authored 4 months ago d0d2421b

Name	Last commit	Last update
..		
models	Replacing montage-wrappers with newer MontagePy m...	4 months ago
utils	ESCAP-161 Fixing bugs, adding more classification feed...	1 year ago
Dockerfile	Replacing montage-wrappers with newer MontagePy m...	4 months ago
requirements.txt	Replacing montage-wrappers with newer MontagePy m...	4 months ago

master sdc1-solution / ska / sdc1 / utils History Find file Clone

ESCAP-161 Fixing bugs, adding more classification feedback, add methods to...
James Collinson authored 1 year ago aa341653

Name	Last commit	Last update
..		
bdsf_utils.py	ESCAP-161 Drop NaNs from truth dataframes	1 year ago
classification.py	ESCAP-161 Fixing bugs, adding more classification feedb...	1 year ago
columns.py	Refactoring SDC1 analysis pipeline	1 year ago
image_utils.py	ESCAP-154 Tidying up Image2D class	1 year ago
postprocess.py	Adding skeleton refactored SDC1 pipeline	1 year ago
source_finder.py	ESCAP-122 Adding docs	1 year ago

Dockerfile

- As you've seen in previous talks, this is the file that is used to build your container
- We use a requirements.txt file which has all the Python packages in it

master


sdcl-solution / ska / sdcl / Dockerfile

Find file

Blame

History

Permalink




 Replacing montage-wrappers with newer MontagePy methods

James Collinson authored 4 months ago

d8d2421b

Dockerfile

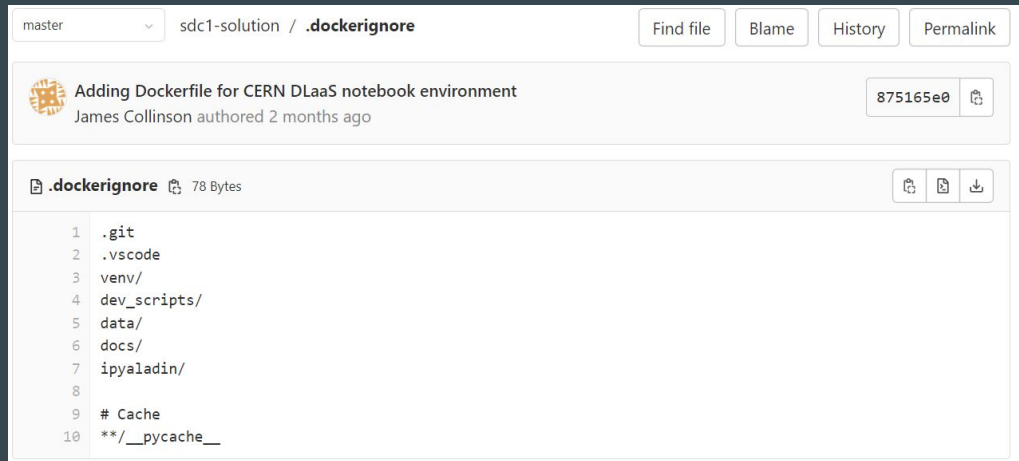
981 Bytes



```
1 FROM ubuntu:18.04
2
3 ENV DEBIAN_FRONTEND=noninteractive
4 RUN apt-get update \
5     && apt-get --yes install --no-install-recommends \
6     bison \
7     build-essential \
8     cmake \
9     eog \
10    flex \
11    g++ \
12    gcc \
13    gettext-base \
14    gfortran \
15    git \
16    libarmadillo-dev \
17    libblas-dev \
18    libcfitsio-dev \
19    libfftw3-dev \
20    libgsl-dev \
21    libgkmm-3.0-dev \
22    libhdf5-serial-dev \
23    liblapack-dev \
24    liblog4plus-1.1-9 \
25    liblog4plus-dev \
26    libncurses5-dev \
27    libpng-dev \
28    libpython3-dev \
29    libreadline-dev \
30    libxml2-dev \
31    openssh-server \
32    python3.6 \
33    python3-pip \
34    python3-tk \
35    python3-setuptools \
36    subversion \
37    vim \
38    wxlib-dev \
39    wget \
40    screen
41
42 # python3 requirements
43 COPY ./requirements.txt /tmp/
44 RUN python3.6 -m pip install --upgrade pip
45 RUN python3.6 -m pip install -r /tmp/requirements.txt
46
47 ENV LD_LIBRARY_PATH=${LD_LIBRARY_PATH}/usr/local/lib
48
49 WORKDIR /opt/
50
51 ENTRYPOINT /bin/bash
```

Dockerignore file

- This lets you specify parts of your repository that don't need to be included when you build your container
- Helps keep your container lightweight

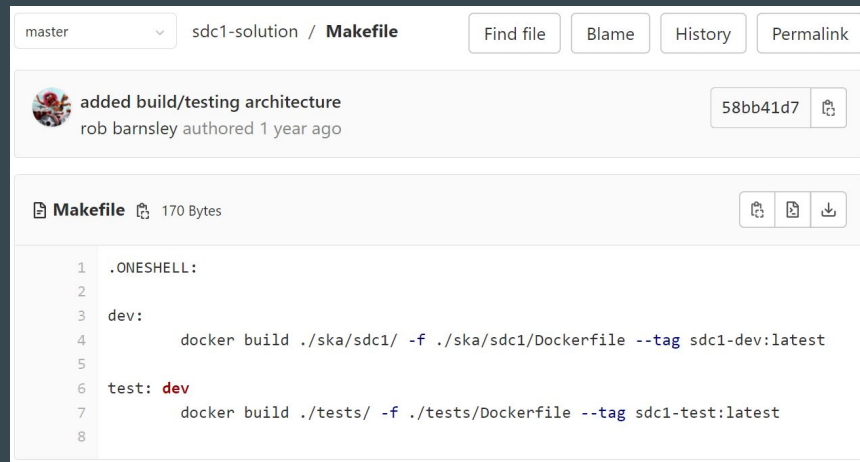


The screenshot shows a GitHub web interface for a repository named 'sdc1-solution'. The file '.dockerignore' is selected, showing its content. Above the file content, there is a commit message: 'Adding Dockerfile for CERN DLaaS notebook environment' by James Collinson, authored 2 months ago. The file size is 78 Bytes. The content of the .dockerignore file is as follows:

```
1 .git
2 .vscode
3 venv/
4 dev_scripts/
5 data/
6 docs/
7 ipyaladin/
8
9 # Cache
10 **/__pycache__
```

Makefile

- The makefile is used to make long Docker build commands simple to implement
- A user just types “make dev” - as stated in the README, to build the container
- Can be used to build the test container for running unit-tests, or other versions for more complex software



The screenshot shows a GitHub web interface for a repository named 'sdc1-solution'. The file 'Makefile' is selected, showing its commit history and content. The commit was made by 'rob barnsley' 1 year ago. The file size is 170 Bytes. The content of the Makefile is as follows:

```
1 .ONESHELL:
2
3 dev:
4     docker build ./ska/sdc1/ -f ./ska/sdc1/Dockerfile --tag sdc1-dev:latest
5
6 test: dev
7     docker build ./tests/ -f ./tests/Dockerfile --tag sdc1-test:latest
8
```


Aliases

- A friendly addition to make your repositories easy to use for people who are less familiar with Docker commands
- They make complex commands easy to execute, avoiding human errors which lead to **reproducibility** issues
- Use them for commands that don't need to be changed, such as building, starting, stopping and executing containers
- Aliases and associated Docker scripts are neatly tucked away in the **/etc** folder, not associated with the code (**/ska** & **/scripts**) of your workflow

sdc1-solution / etc / **aliases**

aliases 435 Bytes

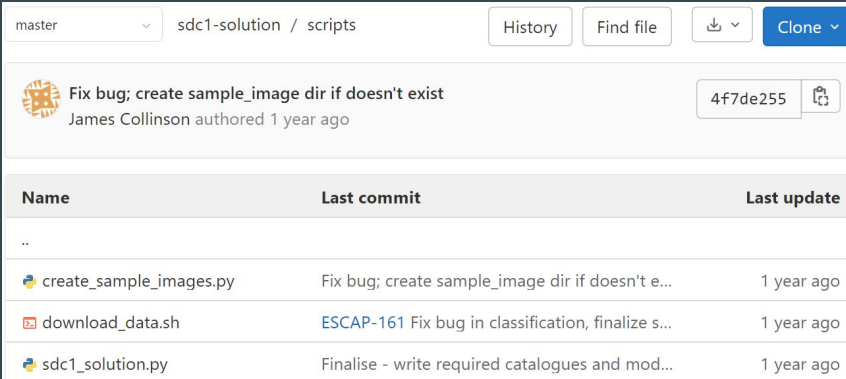
```
1 #!/bin/bash
2
3 alias sdc1-run-unittests="/bin/bash $SDC1_SOLUTION_ROOT/etc/run_unittests.sh"
4 alias sdc1-start-dev="/bin/bash $SDC1_SOLUTION_ROOT/etc/init_dev.sh"
5 alias sdc1-exec-dev="docker exec -it sdc1-dev /bin/bash"
6 alias sdc1-stop-dev="docker stop sdc1-dev"
7 alias sdc1-start-test="/bin/bash $SDC1_SOLUTION_ROOT/etc/init_test.sh"
8 alias sdc1-exec-test="docker exec -it sdc1-test /bin/bash"
9 alias sdc1-stop-test="docker stop sdc1-test"
```

init_dev.sh 391 Bytes




```
1 #!/bin/bash
2
3 docker container stop sdc1-dev >> /dev/null 2>&1
4 docker container rm sdc1-dev >> /dev/null 2>&1
5
6 docker run --name sdc1-dev -d -t \
7   -v $SDC1_SOLUTION_ROOT/data:/opt/data \
8   -v $SDC1_SOLUTION_ROOT/docs:/opt/docs \
9   -v $SDC1_SOLUTION_ROOT/scripts:/opt/scripts \
10  -v $SDC1_SOLUTION_ROOT/ska:/opt/ska \
11  -v $SDC1_SOLUTION_ROOT/tests:/opt/tests \
12  sdc1-dev:latest
```

/scripts

- They are lightweight and easy to interpret, since all the functions are in the `/ska` folder
- You've seen and interacted with these in my demo video from last week
- They can be easily tweaked by the users to get desired outcomes for their specific needs
- Users don't need to go into `/ska` to access the bulk of the code, they can interact with these easier



The screenshot shows the GitHub interface for the 'sdcl-solution / scripts' repository. At the top, there's a navigation bar with 'master' selected, the repository path 'sdcl-solution / scripts', and buttons for 'History', 'Find file', a download icon, and 'Clone'. Below this, a commit message is displayed: 'Fix bug; create sample_image dir if doesn't exist' by James Collinson, authored 1 year ago, with the commit hash '4f7de255'. The main content is a table listing files in the directory.

Name	Last commit	Last update
..		
 create_sample_images.py	Fix bug; create sample_image dir if doesn't e...	1 year ago
 download_data.sh	ESCAP-161 Fix bug in classification, finalize s...	1 year ago
 sdc1_solution.py	Finalise - write required catalogues and mod...	1 year ago

/scripts/sdc1-solution.py

- This runs the whole workflow, and is broken down into steps
- Only ever references functions and data, no “core code” in this Python script, that is all in */ska*
- Designed to be an easy-to-read code, where variables can be changes, or additional steps can be implemented

```

sdcl_solution.py 7.09 KB
1 import os
2 from pathlib import Path
3 from time import time
4
5 import numpy as np
6 from ska.sdc import Sdc1Scorer
7 from sklearn.ensemble import RandomForestClassifier
8 from sklearn.metrics import classification_report
9
10 from ska.sdc1.models.sdc1_image import Sdc1Image
11 from ska.sdc1.utils.bdsf_utils import cat_df_from_sr1_df, load_truth_df
12 from ska.sdc1.utils.classification import SKLClassification
13 from ska.sdc1.utils.source_finder import SourceFinder
14
15 # Challenge frequency bands
16 #
17 FREQS = [560, 1400, 9200]
18
19 # Input data paths; assumes defaults from download_data.sh
20 #
21 def image_path(freq):
22     return os.path.join("data", "images", "{}mhz_1000h.fits".format(freq))
23
24
25 def pb_path(freq):
26     return os.path.join("data", "images", "{}mhz_pb.fits".format(freq))
27
28
29 def train_truth_path(freq):
30     return os.path.join("data", "truth", "{}mhz_truth_train.txt".format(freq))
31
32
33 def full_truth_path(freq):
34     return os.path.join("data", "truth", "{}mhz_truth_full.txt".format(freq))
35
36
37 # Output data paths
38 #
39 def train_source_df_path(freq):
40     return os.path.join("data", "sources", "{}mhz_sources_train.csv".format(freq))
41
42
43 def full_source_df_path(freq):
44     return os.path.join("data", "sources", "{}mhz_sources_full.csv".format(freq))
45
46
47 def submission_df_path(freq):
48     return os.path.join("data", "sources", "{}mhz_submission.csv".format(freq))
49
50

```

```

67 if __name__ == "__main__":
68     ## SKA
69     Run through a simple analysis workflow to solve SDC1
70
71     1) Preprocess images (correct PB) and crop out the training area for
72         building ML model
73     2) Find sources in the PB-corrected training images
74     3) Train a classifier for each band to predict the class of each source
75     4) Find sources in the full PB-corrected image
76     5) Predict the class of each source
77     6) Calculate the score for each image band, and write out a short report
78     ***
79
80     time_0 = time()
81     # 1) Create in-memory representation of image and preprocess
82     print("\nStep 1: Preprocessing; elapsed: {:.2f}s".format(time() - time_0))
83     sdc1_image_list = []
84     for freq in FREQS:
85         new_image = Sdc1Image(freq, image_path(freq), pb_path(freq))
86         new_image.preprocess()
87         sdc1_image_list.append(new_image)
88
89     # In data/images, we now have PB-corrected and training images for each band
90
91     # 2) Source finding (training):
92     print("\nStep 2: Source finding (train); elapsed: {:.2f}s".format(time() - time_0))
93     sources_training = {}
94     for sdc1_image in sdc1_image_list:
95         source_finder = SourceFinder(sdc1_image.train)
96         sl_df = source_finder.run()
97         sources_training[sdc1_image.freq] = sl_df
98
99     # (Optional) Write source List DataFrame to disk
100     write_df_to_disk(sl_df, train_source_df_path(sdc1_image.freq))
101
102     # Remove temp files:
103     source_finder.reset()
104
105     # <Additional feature engineering of the source DataFrames can be performed here>
106
107     # 3) Train classifiers for each frequency's source DataFrame:
108     print("\nStep 3: Training classifiers; elapsed: {:.2f}s".format(time() - time_0))
109     classifiers = {}
110     for freq, source_train_df in sources_training.items():
111         # Load truth catalogue for the training area into memory
112         train_truth_cat_df = load_truth_df(train_truth_path(freq), skiprows=18)
113
114         # Construct and train classifier
115         classifier = SKLClassification(
116             algorithm=RandomForestClassifier,
117             classifier_kwargs={"n_estimators": 100, "class_weight": "balanced"},
118         )
119         srl_df = classifier.train(
120             source_train_df, train_truth_cat_df, regressand_col="class_t", freq=freq
121         )
122
123     # Store model for prediction later

```

Unit tests

- `/tests` is used to run unit-tests. These are lightweight functions that ensure your code is working, without having to do any heavy lifting such as accessing large data sets.
- For this purpose, very small cutout images (**less than 1 MB**) are in this folder, and there is a separate Docker-build to run these.
- Unit-tests should always be shipped as fully working bug-free, so users can test if they have their container set-up properly in minutes before having to execute a workflow that could take days.
- What happens? The expected outputs are tested if they exist, and if so, unit-tests are reported as passed.

In our case, it tests if the primary beam corrected images are created, if source-finding catalogues are created, if machine learning models are created, and if output scoring files are created.

`test_sdc1_image.py` 2.81 KB

```

1  import os
2
3  from ska.sdc1.models.sdc1_image import Sdc1Image
4
5
6  class TestSdc1Image:
7      def test_preprocess_simple_pb(
8          self,
9          images_dir,
10         test_sdc1_image_image_small_name,
11         test_sdc1_image_pb_image_name,
12     ):
13         """
14         Test preprocess with a small test image, with a simple PB correction
15         """
16
17         train_file_expected = test_sdc1_image_image_small_name[:-5] + "_train.fits"
18         pbcor_file_expected = test_sdc1_image_image_small_name[:-5] + "_pbcor.fits"
19         test_image_path = os.path.join(images_dir, test_sdc1_image_image_small_name)
20         pb_image_path = os.path.join(images_dir, test_sdc1_image_pb_image_name)
21
22         # Before running preprocess, the segment and train files shouldn't exist
23         for expected_file in [train_file_expected, pbcor_file_expected]:
24             assert os.path.isfile(os.path.join(images_dir, expected_file)) is False
25
26         sdc1_image = Sdc1Image(560, test_image_path, pb_image_path)
27         sdc1_image.preprocess()
28
29         # Check files have been created
30         for expected_file in [train_file_expected, pbcor_file_expected]:
31             assert os.path.isfile(os.path.join(images_dir, expected_file))
32
33         # Delete them again
34         sdc1_image._delete_train()
35         sdc1_image._delete_pb_corr()
36
37         # Verify
38         for expected_file in [train_file_expected, pbcor_file_expected]:
39             assert os.path.isfile(os.path.join(images_dir, expected_file)) is False

```

Reproducibility

- Reproducibility is more than just presenting your code inside a container
- Think about how a non-expert user sees your Gitlab, how they can reproduce your result, and how they can make use of your workflow for other purposes
- Hopefully you've seen how containers make life a lot easier for you and your users

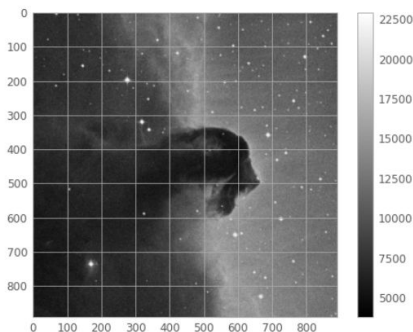
Homework option 1

- I'm sure you all have code + workflows out there that can be implemented inside a container
- Start simple, this could be creating a figure for a paper using some data
- Write your Dockerfile to include the dependencies and build the container
- Try out the code inside, how are you going to get your data to plot? Do you get the expected output?
- Make a Git repository for the workflow with the mindset of a non-expert user coming to reproduce your plot/result
- Add some documentation (at least a README) on how a user can build, run, exec the container, and run code inside
- Can you create makefiles or aliases to make these commands easier to type?
- If you get stuck or want feedback, let us know and we can have a look and try running your code

Homework option 2

- If you would rather test out a really simple workflow, try implementing this plot from the Astropy docs inside a container

https://docs.astropy.org/en/stable/generated/examples/io/plot_fits-image.html



Read and plot an image from a FITS file

This example opens an image stored in a FITS file and displays it to the screen.

This example uses `astropy.utils.data` to download the file, `astropy.io.fits` to open the file, and `matplotlib.pyplot` to display the image.

By: Lia R. Corrales, Adrian Price-Whelan, Kelle Cruz

License: BSD

Set up matplotlib and use a nicer set of plot parameters

```
import matplotlib.pyplot as plt
from astropy.visualization import astropy_mpl_style
plt.style.use(astropy_mpl_style)
```

Download the example FITS files used by this example:

```
from astropy.utils.data import get_pkg_data_filename
from astropy.io import fits

image_file = get_pkg_data_filename('tutorials/FITS-images/HorseHead.fits')
```

Use `astropy.io.fits.info()` to display the structure of the file:

```
fits.info(image_file)
```

Out:

No.	Name	Ver	Type	Cards	Dimensions	Format
0	PRIMARY	1	PrimaryHDU	161	(891, 893)	int16
1	er.mask	1	TableHDU	25	1600R x 4C	[F6.2, F6.2, F6.2, F6.2]

Generally the image information is located in the Primary HDU, also known as extension 0. Here, we use `astropy.io.fits.getdata()` to read the image data from this first extension using the keyword argument `ext=0`:

```
image_data = fits.getdata(image_file, ext=0)
```

The data is now stored as a 2D numpy array. Print the dimensions using the shape attribute:

```
print(image_data.shape)
```

Out:

```
(893, 891)
```

Display the image data:

```
plt.figure()
plt.imshow(image_data, cmap='gray')
plt.colorbar()
```

Good luck have fun

- Thanks for following, we hope you feel more confident with using Containers
- We also hope you feel more inspired about why they are useful and important

Thanks for listening