Setup ODC with Docker

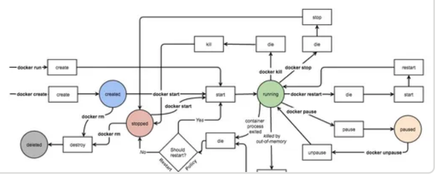
<< Install ODC with Docker >>

Step 0. What is Docker?

Docker筆記 - Docker基礎教學

這篇文章會幫助你搞懂什麼是 image、container、registry、repository 的意思，以及如何使用 image 與 container 的基本功能。 image 的相關操作- 取得 image

https://medium.com/alberthg-docker-notes/docker%E7%AD%86%E8%A8%9...



Docker筆記 - 進入Container，建立並操作 PostgreSQL Container

在 上一篇—基礎教學 的部分我們提到了 image 與 container 的基礎用法，現在將要探討的是如何進入一個正在執行的 container 中並且操作他。 這篇文章一開始會先說

https://medium.com/alberthg-docker-notes/docker%E7%AD%86%E8%A8%9...

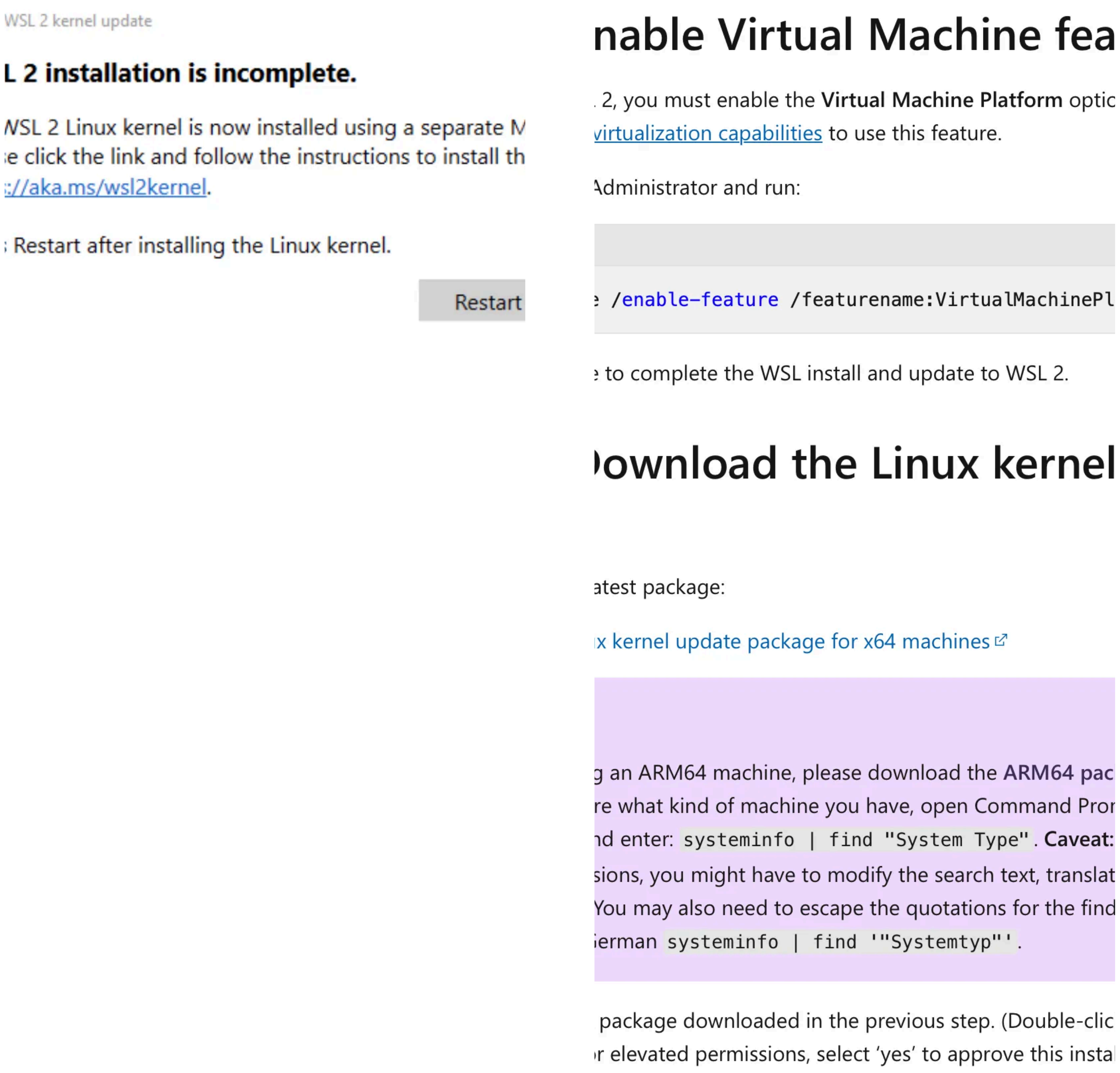
Step 1. Install Docker

Install Docker Engine

Docker Desktop for Linux Docker Desktop helps you build, share, and run containers easily on Mac and Windows as you do on Linux. We are excited to

https://docs.docker.com/engine/install/

! If it brings up a WSL problem, click the link it offered and execute step 3. and step 4. (as below) Then restart your computer if the docker still failed to start.


WSL 2 kernel update

L 2 installation is incomplete.

NSL 2 Linux kernel is now installed using a separate M
e click the link and follow the instructions to install th
://aka.ms/wsl2kernel.

Restart after installing the Linux kernel.

nable Virtual Machine fea

2, you must enable the **Virtual Machine Platform** optic
virtualization capabilities to use this feature.

Administrator and run:

```
e /enable-feature /featurename:VirtualMachinePl
```

to complete the WSL install and update to WSL 2.

ownload the Linux kernel

atest package:

x kernel update package for x64 machines

g an ARM64 machine, please download the **ARM64 pac**
re what kind of machine you have, open Command Pror
nd enter: `systeminfo | find "System Type"`. **Caveat:**
sions, you might have to modify the search text, translat
You may also need to escape the quotations for the find
ierman `systeminfo | find '"Systemtyp"'`.

package downloaded in the previous step. (Double-clic
r elevated permissions, select 'yes' to approve this insta

Step 2. Check if postgres and datacube are in the same network for working together

```
docker network inspect bridge
```

> If it's not linking together, create net for Postgres and datacube

```
docker network create <name of your net> #docker network create my-net
```

Step 3. Pull Postgres and ODC images from Docker Hub

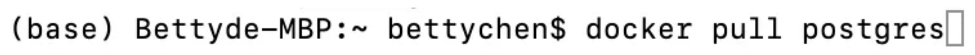
postgres - Official Image | Docker Hub
The PostgreSQL object-relational database system provides reliability and data integrity.

https://hub.docker.com/_/postgres

Docker Hub

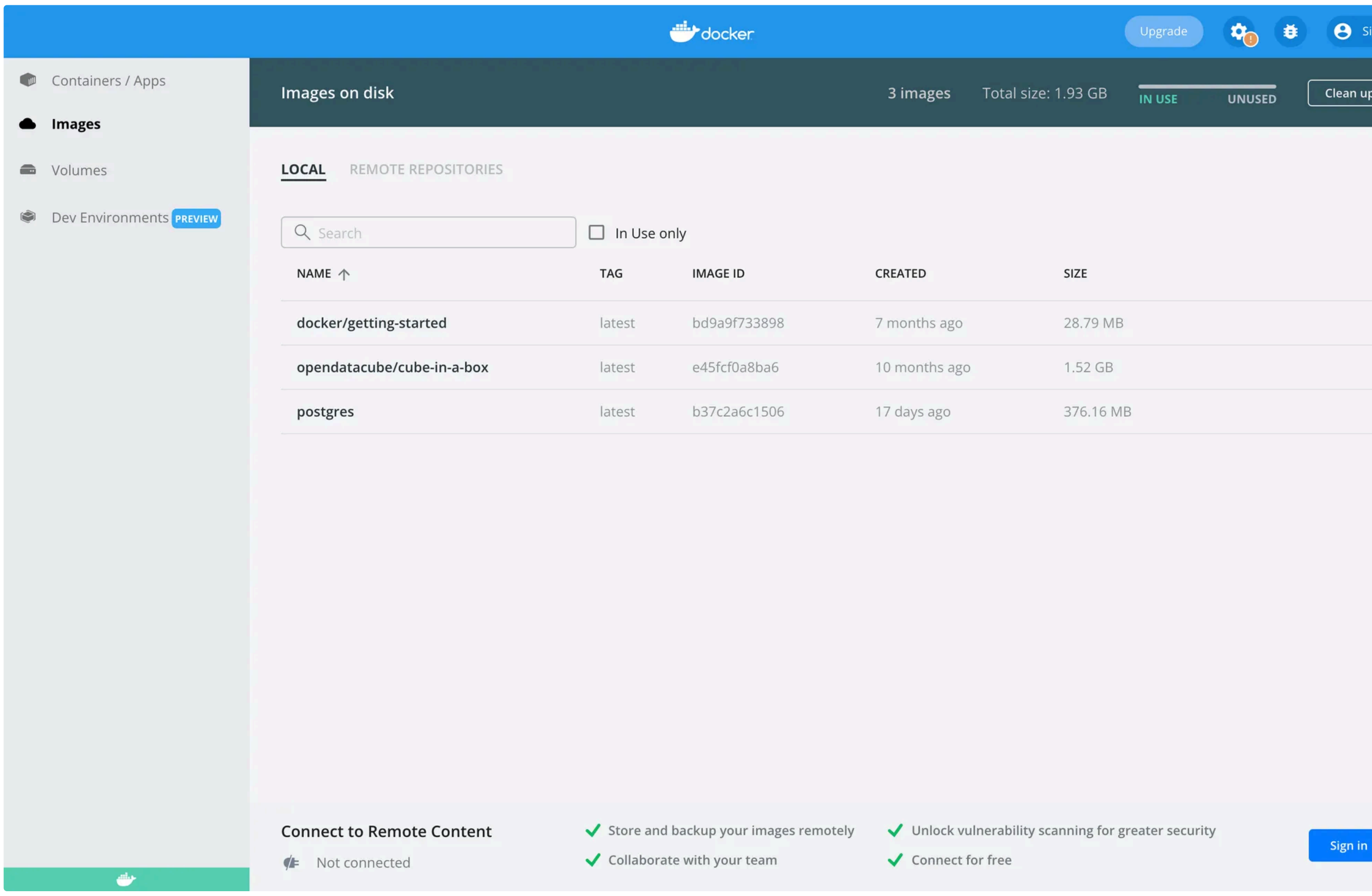
https://hub.docker.com/r/opendatacube/cube-in-a-box

> You can execute it by duplicate the docker pull command to the command line of your computer. (as below)



```
(base) Bettyde-MBP:~ bettychen$ docker pull postgres
```

>> When you done with it you'll get your docker be like this :



Images on disk 3 images Total size: 1.93 GB IN USE UNUSED

NAME	TAG	IMAGE ID	CREATED	SIZE
docker/getting-started	latest	bd9a9f733898	7 months ago	28.79 MB
opendatacube/cube-in-a-box	latest	e45fcf0a8ba6	10 months ago	1.52 GB
postgres	latest	b37c2a6c1506	17 days ago	376.16 MB

Connect to Remote Content
Not connected
✓ Store and backup your images remotely
✓ Collaborate with your team
✓ Unlock vulnerability scanning for greater security
✓ Connect for free

Step 4. Build the container for Postgres and Datacube

- For Postgres

```
docker run -d --name <name for postgres container> -p 8080:5432 -e STGRES_PASSWORD=<your password> postgres #for example : #docker run -d --name opendatacube -p 8080:5432 -e STGRES_PASSWORD=019910 postgres #explaination : #--name : name the container of your postgres #-p 8080:5432 : When it call port 5432 it'll be directed to 8080, so your port won't repeat easily, or it may cause error. #-e STGRES_PASSWORD=<your password>: -e means write in. so it means write in the password of postgres #postgres : The name of your image, here we are building for postgres.
```

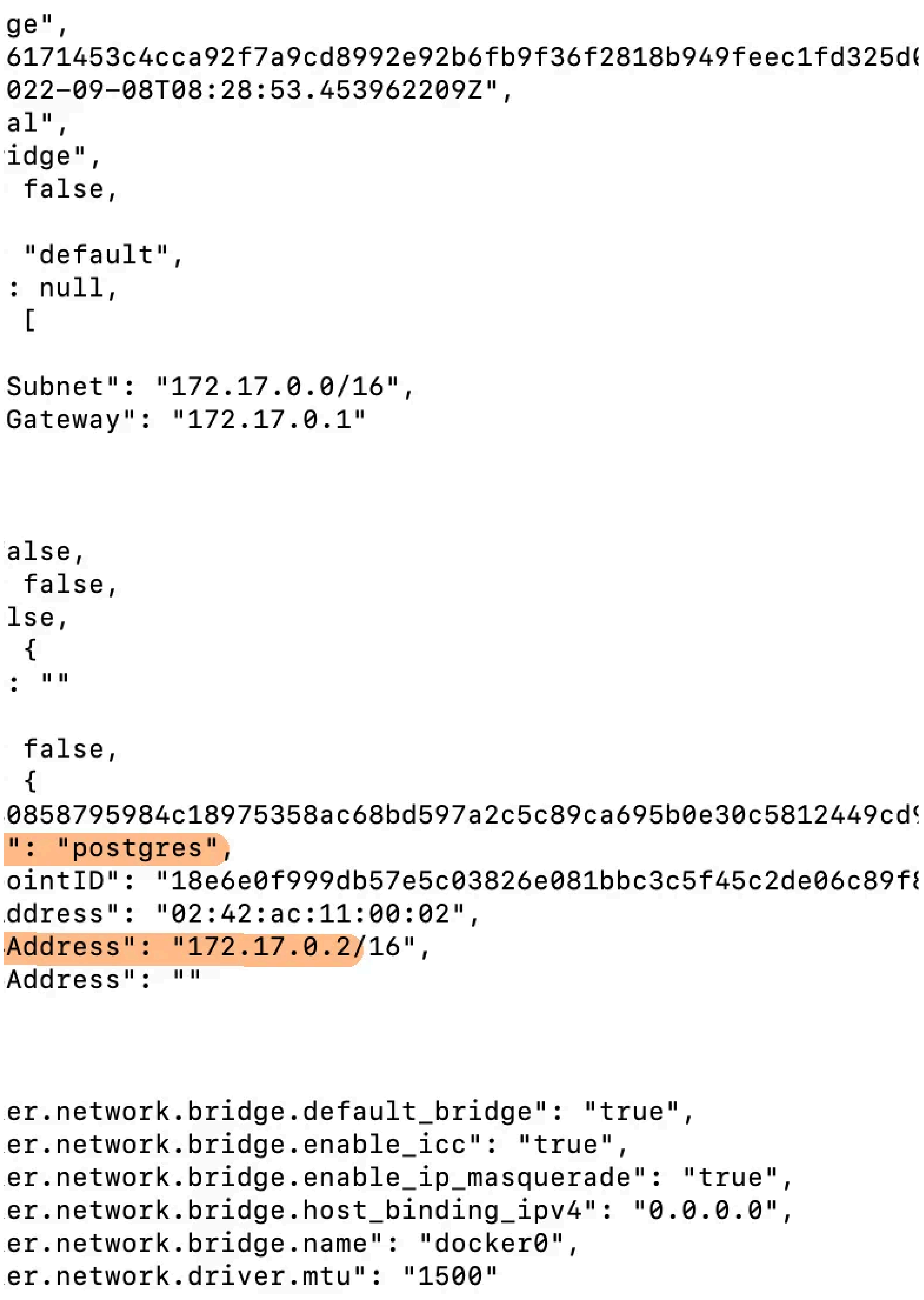
- For Datacube

```
docker run --name <name for your datacube container> -d -p 443:8888 -e DB_HOSTNAME=postgres -e DB_USERNAME=postgres -e DB_PASSWORD=<password for your datadase> -e DB_DATABASE=opendatacube -e AWS_NO_SIGN_REQUEST=true -v <file path of fold for data you want to store>:/<name of your folder> --add-host=postgres:<ip of your postgres in docker, to check it with the way as below> opendatacube/cube-in-a-box #for example : #docker run --name odc -d -p 443:8888 -e DB_HOSTNAME=postgres -e #DB_USERNAME=postgres -e DB_PASSWORD=019910 -e DB_DATABASE=opendatacube #-e AWS_NO_SIGN_REQUEST=true -v #/Users/bettychen/Downloads/NTU/Group/CUBE/ODC:/odctest #--add-host=postgres:172.17.0.2 opendatacube/cube-in-a-box docker run --name odc -d -p 443:8888 -e DB_HOSTNAME=postgres -e DB_USERNAME=postgres -e DB_PASSWORD=019910 -e DB_DATABASE=opendatacube -e AWS_NO_SIGN_REQUEST=true -v Users/bettychen/Downloads/NTU/Group/CUBE/ODC:/odctest --add-host=postgres:172.17.0.2 opendatacube/cube-in-a-box #explaination : #<file path of fold for data you want to store>:/<name of your folder> : The folder to store the data of your datacube.
```

> ***How to check my IP of Postgres?***

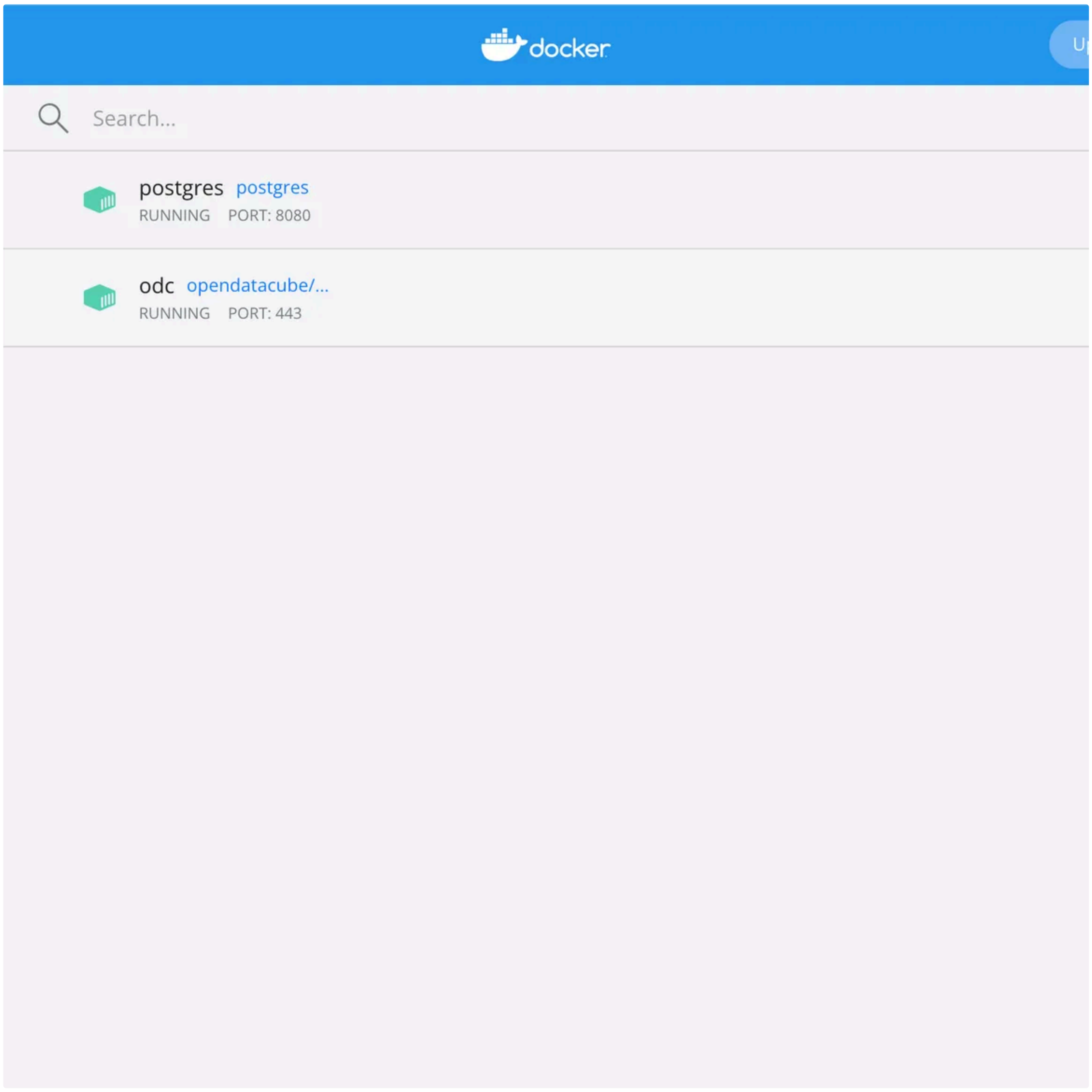
```
#run this code docker network inspect bridge
```

> After execute the code you'll get the following info. The IPv4Address in orange is your ip for postgres in docker.



```
ge",
6171453c4cca92f7a9cd8992e92b6fb9f36f2818b949feec1fd325d
022-09-08T08:28:53.453962209Z",
al",
idge",
 false,

 "default",
: null,
 [

Subnet": "172.17.0.0/16",
Gateway": "172.17.0.1"



alse,
 false,
lse,
 {
: ""

 false,
 {
0858795984c18975358ac68bd597a2c5c89ca695b0e30c5812449cd
": "postgres",
ointID": "18e6e0f999db57e5c03826e081bbc3c5f45c2de06c89f
ddress": "02:42:ac:11:00:02",
Address": "172.17.0.2/16",
Address": ""



er.network.bridge.default_bridge": "true",
er.network.bridge.enable_icc": "true",
er.network.bridge.enable_ip_masquerade": "true",
er.network.bridge.host_binding_ipv4": "0.0.0.0",
er.network.bridge.name": "docker0",
er.network.driver.mtu": "1500"
```

>> When you done with step 4 you should get your containers as below



postgres postgres
RUNNING PORT: 8080

odc opendatacube/...
RUNNING PORT: 443

Step 5. Start Unix shell in the odc container

1. Start your container in the order of postgres then datacube, by clicking the play button on the right hand side. If the order is wrong the port of postgres will be occupied by datacube, which will cause datacube couldn't find postgres and error.
2. run this code

```
docker exec -it <name of your datacube container> bash #for example docker exec -it
odc bash
```

Step 6. Start your datacube

By running this code

```
datacube -v system init
```

> If you failed or got error :

1. Write the datacube.conf by your self

```
vim datacube.conf ####In the datacube.conf write in : [datacube] db_hostname:
postgres db_database: <The name you named for your database> db_username: postgres
db_password: <The password you assigned for datacube> #example : #[datacube]
#db_hostname: postgres #db_database: opendatacube #db_username: postgres
#db_password: 019910
```

2. run these codes

```
export DB_USERNAME=postgres export DB_DATABASE=postgres
```

3. Then try to init database again

```
datacube -v system init
```

How to save datasets to open data cube?

The whole concept is that you will need three elements to save a dataset. The three elements are *“a python script to generate metadata file (.py)”*, *“a metadata file of your dataset (.yaml)”*, and *“a product definition to load your datasets (.yaml)”*.

💡 The definition of a dataset in an open data cube is clarified by their team member as follow:

A dataset is a single time slice of a spatial area defined by storage boundaries. A dataset may be stored as either:

- A single file containing a one-time slice and multiple bands. (one file=one dataset)
- A collection of single-band files covering a single shared time slice and spatial area. (multiple files = one dataset)
- One-time slice from a multi-dimensional file (one file = multiple dataset)

The middle option performs the best if you have a choice.

In addition, according to the team member of the open data cube, do not use NetCDF if you do not have to, it has issues with multi-threading and access from http. So, converting NetCDF to TIFF is recommended, with one timeslice per output image, then generating metadata for each timeslice. Therefore, here we provide three useful modules for you to fix some problems you may encounter.

The first one named "Split_nc.py" is for splitting the time series dataset into time slices but they are still NetCDF files. Then, the second one "Metadata_auto_generater.py", is for you to generate metadata easily and automatically, by converting the NetCDF file to TIFF. Finally, the "Metadata_import.bash" as its name is for importing metadata. An example of using these modules will be mentioned below in Tutorial 3.

❗ Here are the modules :

Split_nc.py 1.8KB

Metadata_auto_generater.py 2.2KB

Metadata_import.bash 0.7KB

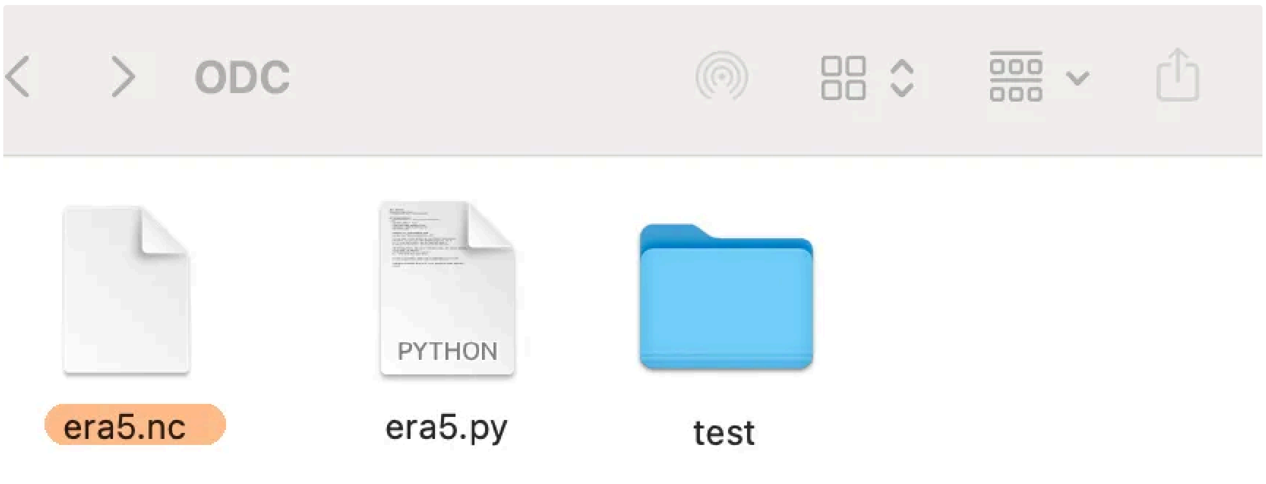
conversion.py 0.9KB

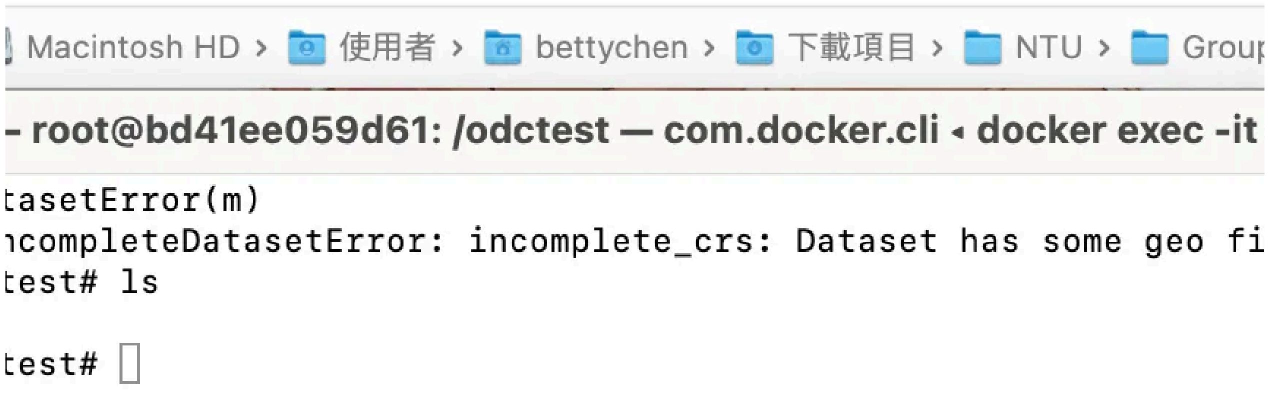
Tutorial 1. Saving data with one timestamp

[1] << Exporting metadata >> = *a python script to generate metadata file (.py)*

> To save your data into data cube, you need to export the metadata of your dataset first.

Step 1. Drag your data file into the shared folder e.g. ODC, then you should be able to find it at odctest folder in docker





```
- root@bd41ee059d61: /odctest — com.docker.cli ◂ docker exec -it
tasetError(m)
ıcompleteDatasetError: incomplete_crs: Dataset has some geo fi
test# ls

test# 
```

Step 2. Prepare your .py file to create dataset documentations in the same path

EO Datasets 3 - eodatasets3 documentation
https://eodatasets.readthedocs.io/en/latest/

for example :

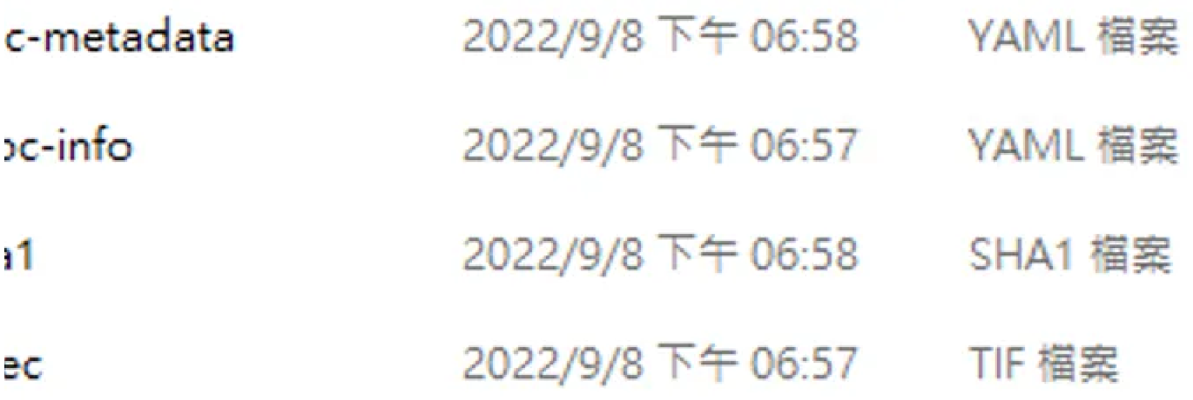
! 1) You may fail to install eodatasets3 due to the lack of psycopg2 in your environment, thus installing psycopg2 may be needed.
2) Be careful !!!! The <output folder name> containing capital letters will raise error.
3) If you have no idea with the epsg of your data, '4326' is relatively recommended due to its general.

```
import datetime from pathlib import Path from eodatasets3 import DatasetAssembler
from conversion import * with DatasetAssembler( Path("<input data path>"),
naming_conventions="default" ) as p: p.product_family = '<product name>' # Add some
common metadata fields. p.datetime = datetime(<datetime of your data>)
p.processed_now() #The "addition mark of output file name" couldn't be blank or
you'll get error conversion('<input file name>', '<espg of your data>', '<output
file name of added crs data>') p.write_measurement("<variable name>", <input data>)
#... # now write more measurements # Validate the dataset and write it to the
destination folder atomically. p.done()
```

Example :

```
import datetime from pathlib import Path from eodatasets3 import DatasetAssembler
from conversion import * with DatasetAssembler( Path("/odctest/."),
naming_conventions="default" ) as p: p.product_family = 'test' p.datetime =
datetime.datetime.now() p.processed_now() #The data type conversion tool #Also a
tool for nc file to add crs #'''def conversion(input data, espg, output data)'''
conversion('era5.nc', '4326', 'tera5.nc') p.write_measurement("prec", 'tera5.nc')
p.done()
```

>> After executing the .py file you'll get the following result in your output path. As the picture below file named "test 2022-09-08.odc-metadata" is a *"metadata file of your dataset (.yaml) "*



```
c-metadata    2022/9/8 下午 06:58    YAML 檔案
oc-info       2022/9/8 下午 06:57    YAML 檔案
a1            2022/9/8 下午 06:58    SHA1 檔案
ec            2022/9/8 下午 06:57    TIF 檔案
```

[2] <<Import your product definition>> = *"a product definition to load your datasets (.yaml)".*

> Before saving the dataset to a data cube, a product to load your dataset is needed. Then create and import your product definition.

Step 1. Create a product definition in the .yaml file by yourself, and make sure it at least includes the following information. In addition, name: should be the same as <product name> in the .py file.

```
name: <product name in .py file> description: <Any description of your dataset> metadata_type: eo3 metadata: product: name: <product name in .py file> measurements: - name: <measurements name in .py file> dtype: float64 nodata: <if there is no data this will replace the blank> units: 'db'
```

Example :

```
name: test description: test metadata_type: eo3 metadata: product: name: test measurements: - name: prec dtype: float64 nodata: -999 units: 'db'
```

Step 2. Import your product definition to your data cube.

Step-by-step Guide to Indexing Data - Open Data Cube 1.8 documentation
Once you have the Data Cube software installed and connected to a database, you can start to load in some data. This step is performed using the datacube command line tool. Note When you load data into the Data Cube using indexing, all you are
https://datacube-core.readthedocs.io/en/latest/installation/indexing-data/step-guide.html

To load Products into your ODC, execute:

```
datacube product add <path-to-product-definition-yml>
```

Example :

```
datacube product add ./era5_2/PD.yaml
```

[3] <<Import your dataset>> = import *“metadata file of your dataset (.yaml) “*

Step 1. Import metadata into your data cube. Please make sure the ‘product name’ and ‘measurements ’ in metadata .yaml are correct. Write them by yourself if it doesn’t exist. (the words in blue as below)

```
--- # Dataset $schema: https://schemas.opendatacube.org/dataset id: 65737ca1-3f9b-
49dc-8f43-8db7ed566498 label: test_2022-09-16 product: name: -> should equal to
<product name in .py file> crs: epsg:4326 geometry: type: Polygon coordinates:
[[[57.64935921607964, 81.94999795321885], [20.747019942012322, 80.84995559689838],
[10.879289137380338, 79.82071067811864], [10.65775572042583, 79.5886138088126],
[-0.049999999152053956, 53.989074735265014], [-0.049999999152053956,
-90.05000000000004], [359.9499938956364, -90.05000000000004], [359.9499938956364,
55.10788652523004], [359.34945255510587, 60.86039120216408], [348.43838331722986,
81.49676905667096], [348.1054641161294, 81.73320502943376], [331.75796889597103,
83.6496815278536], [323.3499945163329, 83.64999999999998], [57.64935921607964,
81.94999795321885]]] grids: default: shape: [1801, 3600] transform:
[0.09999999830410791, 0.0, -0.049999999152053956, 0.0, -0.1, 90.04999999999998, 0.0,
0.0, 1.0] properties: datetime: 2022-09-16 08:52:53.647688Z fmask:cloud_shadow: 42.0
odc:file_format: GeoTIFF odc:processing_datetime: 2022-09-16 08:52:53.647708Z
odc:product_family: test measurements: dm2: -> should equal to <any name you want
for the measurement of your data> in the product definition file path: test_2022-09-
16_prec.tif accessories: metadata:processor: path: test_2022-09-16.proc-info.yaml
checksum:sha1: path: test_2022-09-16.sha1 lineage: {} ...
```

To load dataset into your ODC, execute:

Step-by-step Guide to Indexing Data - Open Data Cube 1.8 documentation

Once you have the Data Cube software installed and connected to a database, you can start to load in some data. This step is performed using the datacube command line tool. Note When you load data into the Data Cube using indexing, all you are

https://datacube-core.readthedocs.io/en/latest/installation/indexing-data/step-guide.html

```
datacube dataset add <path-to-dataset-document-yaml>
```

```
datacube dataset add ./era5_2/2022/01/01/era5_2_2022-01-01.odc-metadata.yaml
```

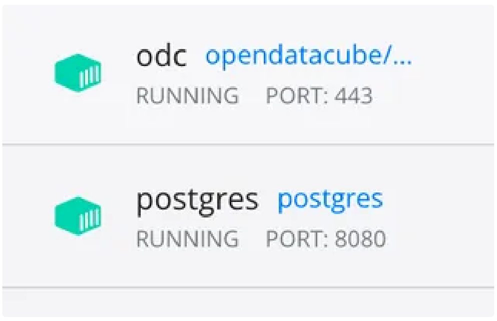
[4] <<Display your data>>

Enter localhost in google

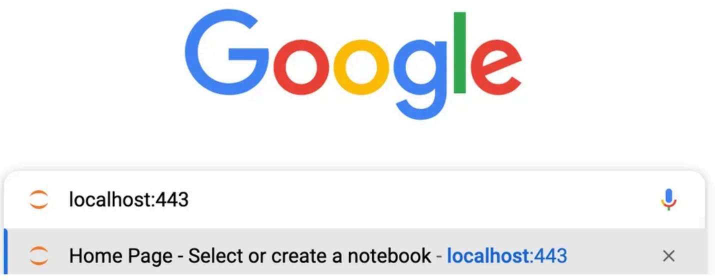
1. Find the port of your datacube and enter notebook by local host

```
localhost:<port of your datacube> #example #localhost:443
```

> You can look up your docker container to find your port


odc opendatacube/...
RUNNING PORT: 443

postgres postgres
RUNNING PORT: 8080


localhost:443

Home Page - Select or create a notebook - localhost:443

2. The password of the local host would be 'secretpassword'
3. Create a notebook and display your data as this file

era5_test.ipynb 86.9KB

Tutorial 2. Saving data with two timestamps

The whole process is similar to Tutorial1. (the one with only one timestamp for each dataset that needs three elements *"a python script to generate metadata file (.py)", "a metadata file of your dataset (.yaml)"*, and *"a product definition to load your datasets (.yaml)"*), however, the setting of product definition and your metadata file should be concerned.

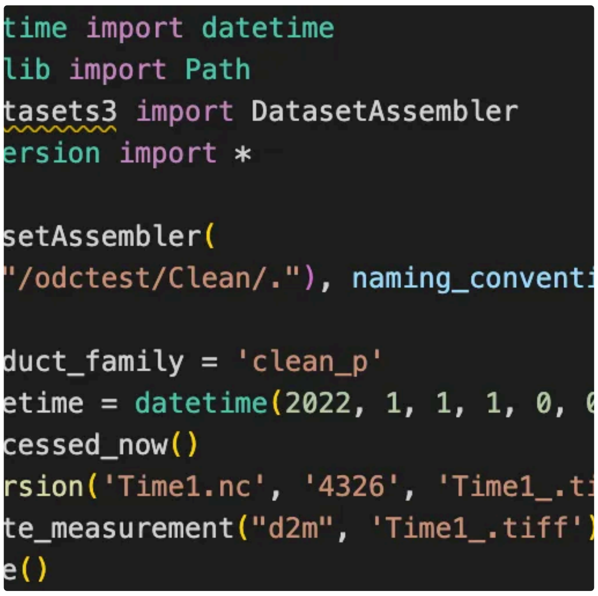
[1] << Exporting metadata >> = *a python script to generate metadata file (.py)*

> To save your data into data cube, you need to export the metadata of your data first.

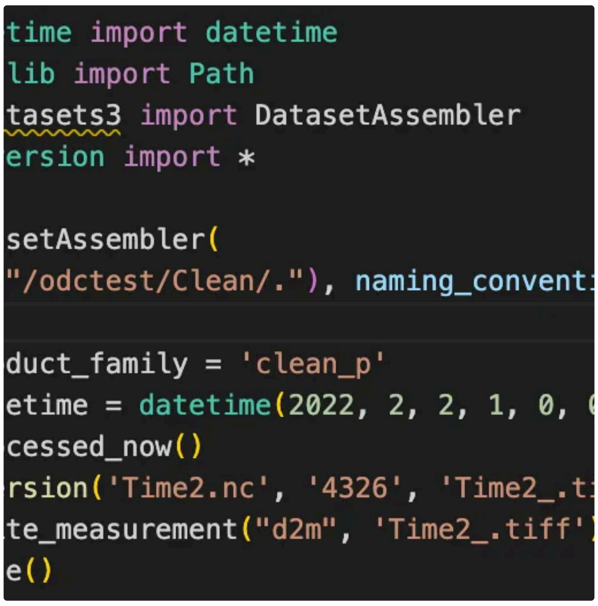
Step 1. Drag your data file into the shared folder e.g. ODC, then you should be able to find it at e.g odctest folder in docker

Step 2. Prepare your .py file to create dataset documentations in the same path

Example :



```
time import datetime
lib import Path
tasets3 import DatasetAssembler
ersion import *

setAssembler(
"/odctest/Clean/."), naming_conventi

duct_family = 'clean_p'
etime = datetime(2022, 1, 1, 1, 0, 0
cessed_now()
rsion('Time1.nc', '4326', 'Time1_.ti
te_measurement("d2m", 'Time1_.tiff')
e()
```



```
time import datetime
lib import Path
tasets3 import DatasetAssembler
ersion import *

setAssembler(
"/odctest/Clean/."), naming_convent

duct_family = 'clean_p'
etime = datetime(2022, 2, 2, 1, 0, 0
cessed_now()
rsion('Time2.nc', '4326', 'Time2_.t
te_measurement("d2m", 'Time2_.tiff')
e()
```

! Careful 1 : The <output folder name> containing capital letter will raise error.

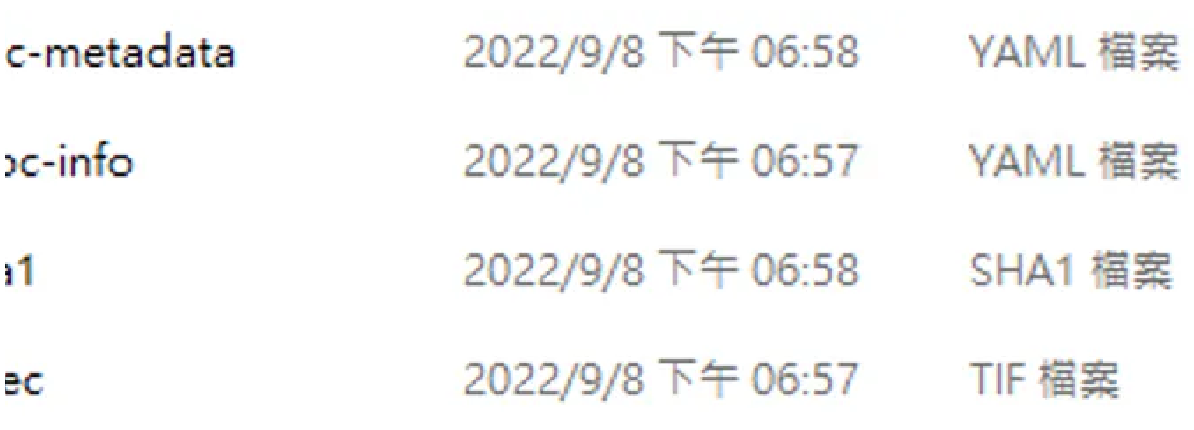
! Careful 2 : The <product name> and <variable name> of the two timestamps should be the same.

for example :

EO Datasets 3 - eodatasets3 documentation
https://eodatasets.readthedocs.io/en/latest/

```
import datetime from pathlib import Path from eodatasets3 import DatasetAssembler
from conversion import * with DatasetAssembler( Path("<input data path>"),
naming_conventions="default" ) as p: p.product_family = '<product name>' p.datetime
= datetime(<datetime of your data>) p.processed_now() p.write_measurement("<variable
name>", <input data>) p.done()
```

>> After executing .py file you'll get the following result in your output path. As the picture below the file named "test_2022-09-08.odc-metadata" is the *"metadata file of your dataset (.yaml) "*



c-metadata 2022/9/8 下午 06:58 YAML 檔案
ɔc-info 2022/9/8 下午 06:57 YAML 檔案
ı1 2022/9/8 下午 06:58 SHA1 檔案
ec 2022/9/8 下午 06:57 TIF 檔案

[2] <<Import your product definition>> = *“a product definition to load your datasets (.yaml)”*.

> Before saving the dataset to a data cube, the product to load your dataset is needed. Then create and import your product definition.

Step 1. Create a product definition in the .yaml file by yourself, and make sure it at least includes the following information. In addition, name: should be the same as <product name> in the .py file.

```
name: <product name in .py file> description: <Any description of your dataset>
metadata_type: eo3 metadata: product: name: <product name in .py file> measurements:
- name: <measurements name in .py file> dtype: float64 nodata: <if there is no data
this will replace the blank> units: 'db'
```

```
name: clean_p description: clean version metadata_type: eo3 metadata: product: name:
clean_p measurements: - name: d2m dtype: float64 nodata: -999 units: 'db'
```

Step 2. Import your product definition to your data cube.

Step-by-step Guide to Indexing Data - Open Data Cube 1.8 documentation
Once you have the Data Cube software installed and connected to a database, you can start to load in some data. This step is performed using the datacube command line tool. Note When you load data into the Data Cube using indexing, all you are
https://datacube-core.readthedocs.io/en/latest/installation/indexing-data/step-guide.html

To load Products into your ODC, execute:

```
datacube product add <path-to-product-definition-yml>
```

Example :

```
datacube product add ./ProDef.yml
```

[3] <<Import your dataset>> = import *“metadata file of your dataset (.yaml) “*

Step 1. Import metadata into your data cube. Please make sure the ‘product name’ and ‘measurements ’ in metadata .yaml are correct. Write them by yourself if it doesn’t exist. (the words in blue as below)

```
--- # Dataset $schema: https://schemas.opendatacube.org/dataset id: 65737ca1-3f9b-
49dc-8f43-8db7ed566498 label: test_2022-09-16 product: name: prec -> should equal to
<product name in .py file> crs: epsg:4326 geometry: type: Polygon coordinates:
[[[57.64935921607964, 81.94999795321885], [20.747019942012322, 80.84995559689838],
[10.879289137380338, 79.82071067811864], [10.65775572042583, 79.5886138088126],
[-0.049999999152053956, 53.989074735265014], [-0.049999999152053956,
-90.05000000000004], [359.9499938956364, -90.05000000000004], [359.9499938956364,
55.10788652523004], [359.34945255510587, 60.86039120216408], [348.43838331722986,
81.49676905667096], [348.1054641161294, 81.73320502943376], [331.75796889597103,
83.6496815278536], [323.3499945163329, 83.64999999999998], [57.64935921607964,
81.94999795321885]]] grids: default: shape: [1801, 3600] transform:
[0.09999999830410791, 0.0, -0.049999999152053956, 0.0, -0.1, 90.04999999999998, 0.0,
0.0, 1.0] properties: datetime: 2022-09-16 08:52:53.647688Z fmask:cloud_shadow: 42.0
odc:file_format: GeoTIFF odc:processing_datetime: 2022-09-16 08:52:53.647708Z
odc:product_family: test measurements: dm2: -> should equal to <any name you want
for the measurement of your data> in the product definition file path: test_2022-09-
16_prec.tif accessories: metadata:processor: path: test_2022-09-16.proc-info.yaml
checksum:sha1: path: test_2022-09-16.sha1 lineage: {}
```

Step 2. load all datasets into your ODC, execute:

Step-by-step Guide to Indexing Data - Open Data Cube 1.8 documentation

Once you have the Data Cube software installed and connected to a database, you can start to load in some data. This step is performed using the datacube command line tool. Note When you load data into the Data Cube using indexing, all you are

https://datacube-core.readthedocs.io/en/latest/installation/indexing-data/step-guide.html

```
datacube dataset add <path-to-dataset-document-yaml>
```

Example :

```
datacube dataset add ./clean_p_2022-01-01.odc-metadata.yaml
datacube dataset add ./clean_p_2022-02-02.odc-metadata.yaml
```

[4] <<Display your data>>

Enter localhost in google

1. Find the port of your datacube and enter the notebook by local host

```
localhost:<port of your datacube> #example #localhost:443
```

> You can find your port by looking up your container

2. The password of the local host would be 'secretpassword'
3. Create a notebook and display your data as this file

display.ipynb 59.4KB

Tutorial 3. Saving time series of images

Here we provide an efficient way with some simple modules to wrap up all the work in this section. Please download the needed modules before you start!

[1] <<Split your NetCDF file>>

Use the module - Split_nc to split your time series nc file into time slices.

> execute this code in python

```
Split_nc('<your time series nc file>', '<output folder path>', '<single parameter in nc file>') #example Split_nc('small.nc', './Split_result', 't2m')
```

[2] << Exporting metadata >>

Execute "Metadata_auto_generater.py" in your odc command line.

> Metadata_auto_generater.py" is below. Please customize it yourself.

```
from datetime import datetime from pathlib import Path from eodatasets3 import
DatasetAssembler import xarray as xr import rioxarray as rio import os import pandas
as pd PATH = <nc files folder path> timestamps = [] outputfiles = [] inputfiles =
os.listdir(PATH) for file in inputfiles: date = [] timelt = [] file_date =
file[-22:-12] file_time = file[-11: -3] for i, t in zip(file_date.split('-'),
file_time.split(':')): if i[0] == '0': i = i[-1] date.append(int(i)) else :
date.append(int(i)) if t[0] == '0': t = t[-1] timelt.append(int(t)) else :
timelt.append(int(t)) timestamps.append(datetime(date[0], date[1], date[2],
timelt[0], timelt[1], timelt[2])) outputfiles.append(str(date[0]) + str(date[1]) +
str(date[2]) + '_' + file[-11:-3]+ '.tiff') print('list ok') tiffoutfile_path =
<Tiffs output path> def conversion(input_filename, epsg, tiffoutfile_path,
out_filename, para): nc_file = xr.open_dataset(input_filename,decode_times=False)
vls = [i for i in nc_file.variables] if para in [i for i in nc_file.variables]: data
= nc_file[para] else: print(f'{para} is not in the variable list! Pleas change the
variable!') data = data.rio.set_spatial_dims(x_dim = vls[0], y_dim = vls[1])
epsgFinal = 'epsg:' + epsg data.rio.write_crs(epsgFinal, inplace=True)
data.rio.to_raster(tiffoutfile_path + '/' + out_filename) for timestamp, inputfile,
outputfile in zip(timestamps, inputfiles, outputfiles): with DatasetAssembler(
Path("/odctest" + "/<metadata output folder path>/"), naming_conventions="default" )
as p: p.product_family = <product name you want> p.datetime = timestamp
p.processed_now() print(f'{PATH}/{inputfile}') conversion(PATH + '/' + inputfile,
<epsg code>, tiffoutfile_path, outputfile, 'value') p.write_measurement(<name you
want for your measurment>, tiffoutfile_path + '/' + outputfile) p.done()
```

example :

```
from datetime import datetime from pathlib import Path from eodatasets3 import
DatasetAssembler import xarray as xr import rioxarray as rio import os import pandas
as pd PATH = './Split_improve' timestamps = [] outputfiles = [] inputfiles =
os.listdir(PATH) for file in inputfiles: date = [] timelt = [] file_date =
file[-22:-12] file_time = file[-11: -3] for i, t in zip(file_date.split('-'),
file_time.split(':')): if i[0] == '0': i = i[-1] date.append(int(i)) else :
date.append(int(i)) if t[0] == '0': t = t[-1] timelt.append(int(t)) else :
timelt.append(int(t)) timestamps.append(datetime(date[0], date[1], date[2],
timelt[0], timelt[1], timelt[2])) outputfiles.append(str(date[0]) + str(date[1]) +
str(date[2]) + '_' + file[-11:-3]+ '.tiff') print('list ok') tiffoutfile_path =
'./TiffsImprove' def conversion(input_filename, epsg, tiffoutfile_path,
out_filename, para): nc_file = xr.open_dataset(input_filename,decode_times=False)
vls = [i for i in nc_file.variables] #para = str(input()) if para in [i for i in
nc_file.variables]: data = nc_file[para] else: print(f'{para} is not in the variable
list! Pleas change the variable!') data = data.rio.set_spatial_dims(x_dim = vls[0],
y_dim = vls[1]) epsgFinal = 'epsg:' + epsg data.rio.write_crs(epsgFinal,
inplace=True) data.rio.to_raster(tiffoutfile_path + '/' + out_filename) for
timestamp, inputfile, outputfile in zip(timestamps, inputfiles, outputfiles): with
DatasetAssembler( Path("/odctest" + "/Clean/"), naming_conventions="default" ) as p:
p.product_family = 'improve' p.datetime = timestamp p.processed_now()
print(f'{PATH}/{inputfile}') conversion(PATH + '/' + inputfile, '4326'
```