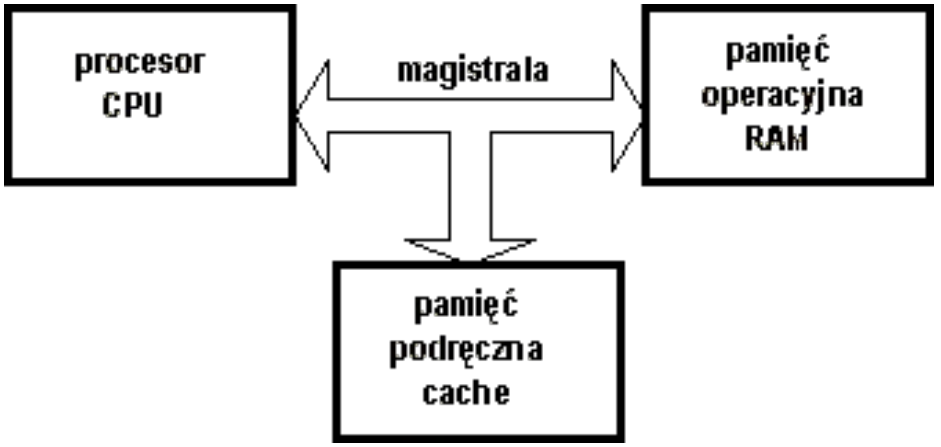
Procesory i5-8250u, ARM Cortex-A53, pamięć podręczna, HDF5, TensorFlow

Pamięć cache

- Lepszy mechanizm przetrzymywania danych w pamięci o lepszych parametrach
- Rodzaje pamięci cache:
  - L1 – pierwszy poziom
  - L2 – drugi poziom
  - L3 – trzeci poziom

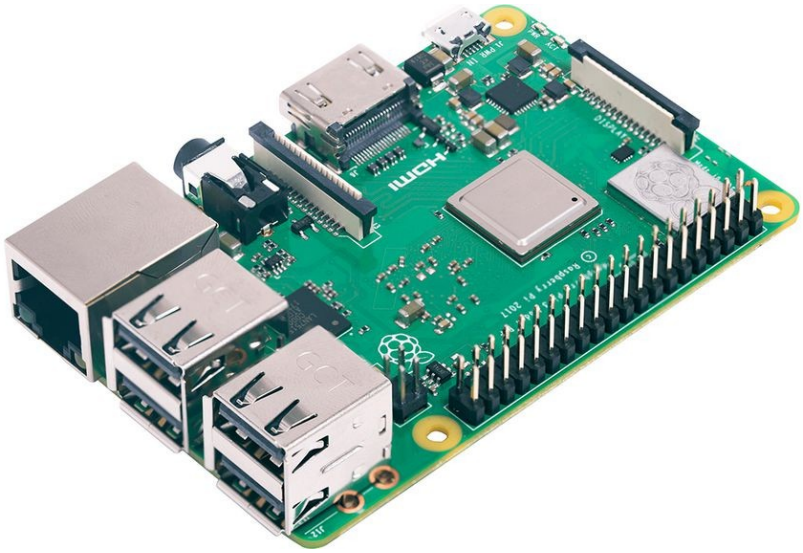


Hp Pavilion 14-ce0000

- Procesor: i5-8250u
- 4 rdzenie, 8 wątków
- 1.6GHz, turbo 3.4GHz
- RAM 16GB DDR4
- 3 poziomowy cache

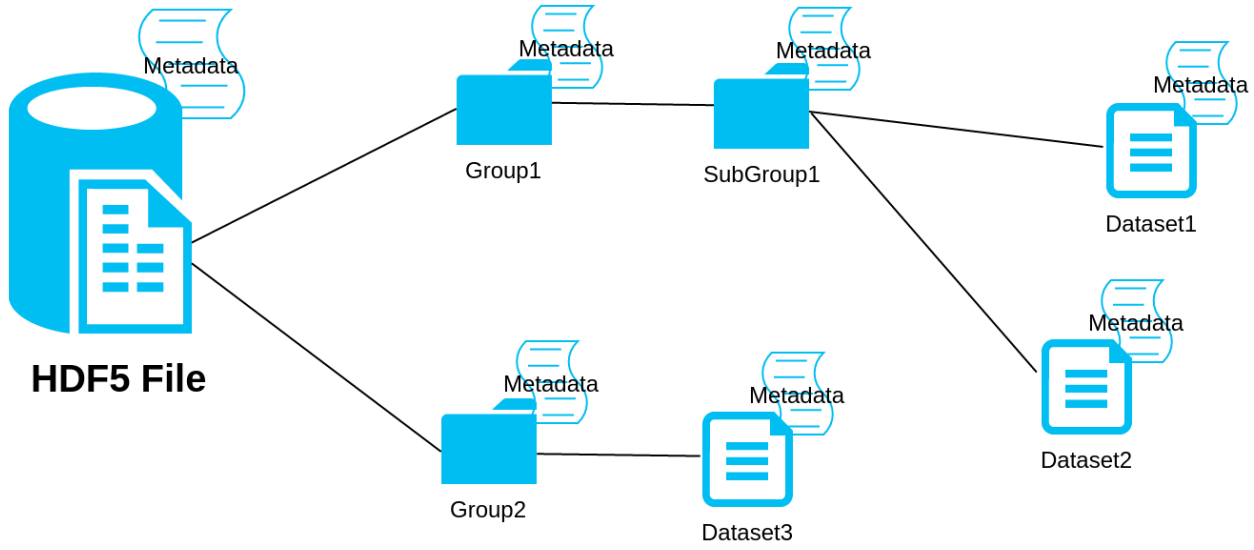
Raspberry PI 3 B+

- Procesor: ARM Cortex-A53
- 4 rdzenie
- 1.2GHz
- RAM 1GB DDR2
- 2 poziomowy cache



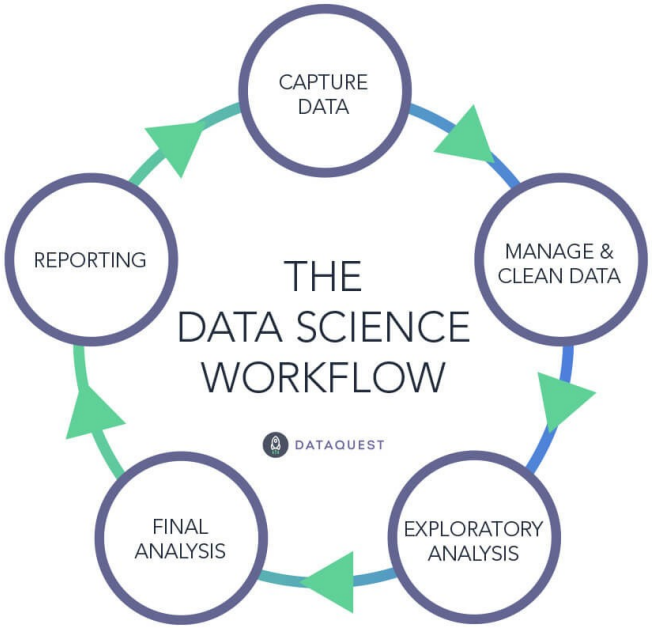
Co to jest HDF5?

HDF5 (Hierarchial Data Format v5) – standard opisujący strukturę pliku, do którego możemy w hierarchiczny i ustrukturowany sposób zapisywać tablicę danych.



W jakim celu został utworzony?

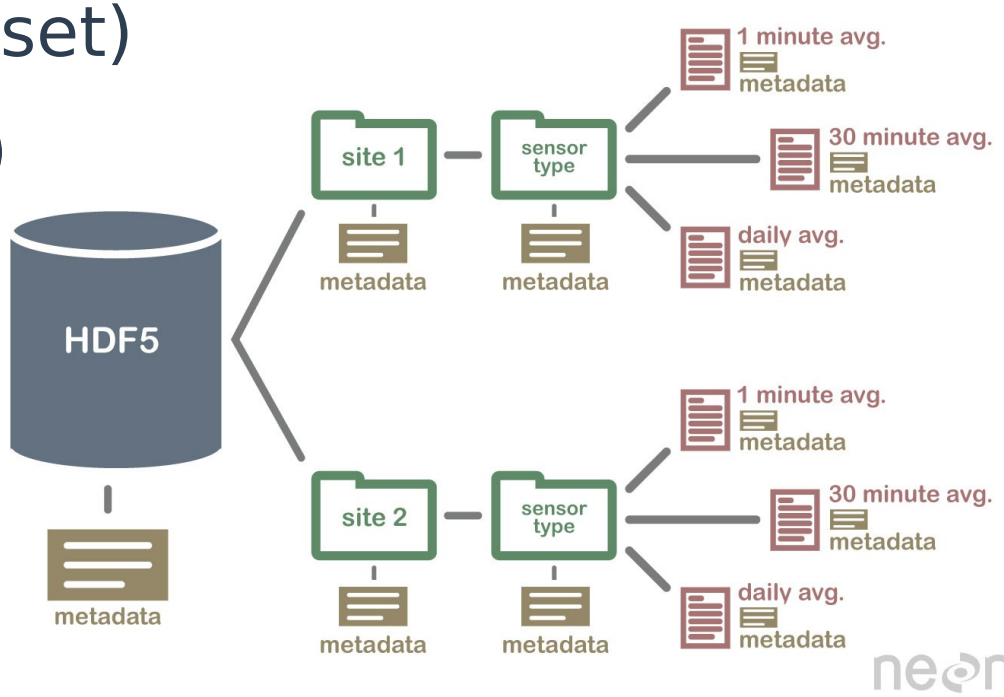
- Powtarzalne workflow pracy **Data Analyst**
- Duże rozmiary plików
- Duże wymagania wolnej pamięci RAM
- Wczytywanie plików partiami



Format pliku

Format pliku przypomina systemu plików.
Wyróżniamy 3 rodzaje obiektów:

- Grupy (groups)
- Zbiory danych (dataset)
- Atrybuty (metadata)



W czym pomaga?

- Katalogowanie danych
- Dodatkowe opisywanie przy pomocy metadanych
- Przechowywanie wielu mały plików w jednym pliku
- Łatwość wykonywania operacji ze względu na zgodność z numpy

Biblioteka python - h5py

Biblioteka ułatwiająca pracę z plikami HDF5 to **h5py:** https://www.h5py.org/

- Aktualna wersja: 2.10.0
- Instalacja: pip install h5py

Otwieranie i tworzenie plików

Dostępne tryby otwarcia pliku:

- **r** – pliku musi istnieć, read-only
- **r+** - pliku musi istnieć, read-write
- **w** – tworzy plik
- **w-** - tworzy plik, jeśli istnieje zwraca błąd
- **a –** tworzy plik jeśli nie istnieje, read-write

```
import h5py


# dobra praktyka
with h5py.File() as hdf_fille:
    hdf_file.create_group('group')
    # inne operacje

# zła praktyka
hdf_file = h5py.File('myfile.hdf5', 'r')
hdf_file.close()
```

Tworzenie grup

```
import h5py

# Tworzenie grupy i podgrupy
group = hdf_file.create_group("gropa")
sub_group = hdf_file.crate_group("podgrupa")

# Grupy korzystają ze słownikowej konwencji pythona
group["podgrupa"]

# Usuwanie
del group["podgrupa"]


# Tworzenie miękkich dowiązań
myfile = h5py.File('foo.hdf5','w')
group = myfile.create_group("somegroup")
myfile["alias"] = h5py.SoftLink('/somegroup')

# Usunięcie spowoduje błędy
del myfile['somegroup']
print myfile['alias']
# KeyError: 'Component not found (Symbol table: Object not found)'
```

Tworzenie zbiorów danych

```
import h5py
import numpy as np


# h5py wspiera większość typów numpy
dataset = hdf_file.create_dataset(name="name", shape=(100,), dtype="f")
dataset_copy = hdf_file["name"]

# Można użyć np.array() zamiast shape i dtype
array = np.arrange(100)
dataset = hdf_file.create_dataset(name="name", data=array)
```

Bloki/części/chunks

Dane mogą być przechowywane w chunkach. Oznacza to, że dane:

- są podzielone na równe części
- przechowywane w różnych miejscach na dysku.
- dostęp odbywa się za pośrednictwem przeszukiwania B-tree

Dzięki blokom możemy:

- tworzyć zbiory danych ze zmiennym rozmiarem
- wykorzystać skompresowane filtrowanie

Rekomendowany rozmiar **chunka** 10KB – 1MB.

Bloki/części/chunks

```
import h5py


# Dane będą trzymane w blokach 100 x 100
dataset = hdf_file.create_dataset("chunked", (1000, 1000), chunks=(100,100))

# Automatyczny dobór rozmiaru chunka
dataset = hdf_file.create_dataset("autochunk", (1000, 1000), chunks=True)
```

Zbiór danych ze zmiennym rozmiarem

```
import h5py


# Z ograniczeniem maksymalnego rozmiaru
dataset = hdf_file.create_dataset("resizable", (10,10), maxshape=(500, 20))

# Potencjalnie nieograniczone(2**64)
dataset = hdf_file.create_dataset("resizable", (10,10), maxshape=(None, 10))

# Zmiana rozmiaru nie jest zaimplementowana tak jak w numpy, jeśli któraś os się
# skurczy, dane w brakującej części są odrzucane, nie następuje przeorganizowanie
# danych.
dataset.resize(new_size, axis)
```

Filter pipline

```
import h5py


# Kompresja danych na dysku
datset = hdf_file.create_dataset("zipped", (100, 100), compression="gzip")

# Z dodatkowymi opcjami
datset = hdf_file.create_dataset(
    "zipped",
    (100, 100),
    compression="gzip",
    compression_opts=9
)
```

Atrybuty

Można dodawać do grupy i zbioru danych. Mają następujące właściwości:

- Mogą być tworzone w dowolnej tablicy skalarnej lub NumPy
- Każdy atrybut powinien być mały mniejszy niż 64K
- Nie ma częściowego I/O , cały atrybut musi zostać odczytany.

Atrybuty

```
import h5py

dataset = hdf_file.create_dataset("resizable", (10,10), dtype="f")
group = hdf_file.create_group("groupa")

dataset.attr["atyrbut1"] = "wartość1"
group.attr["atrybut2"] = "wartość2"
```

Iteracja po trybutach w alfabetycznym porządku. Można to zmienić:

```
import h5py

# Dla wszystkich grup i zbiorów danych
h5.get_config().track_order

# Dla pojedynczego zbioru danych
dataset = hdf_file.create_dataset(
    "resizable", (10,10), dtype="f", track_order=True
)
```

Przykład

HDF5 w TensorFlow

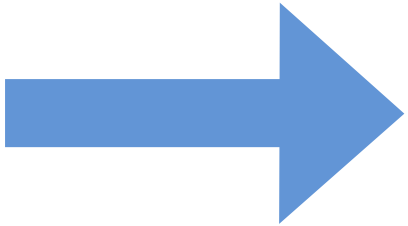
Dostępna jest klasa **tf.Keras.utils.HDF5Matrix**, jest przestarzała i nie powinno się jej użyć, zamiennie:

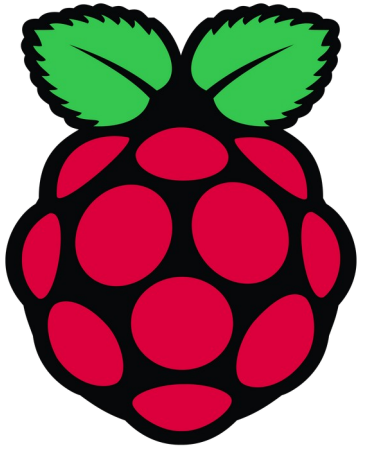
```
import tfio


io_dataset = tfio.IODataset.from_hdf5("nazwa_pliku", "dataset_name",
spec="f", name=None)
```

Budowanie TensorFlow na Raspbian

- **Cel: zainstalowanie tensorflow 2.2 na raspbianie**





Próba 1 – źródła z budowaniem

Zbudowanie ze źródeł.

https://www.tensorflow.org/install/source

Wynik: Niepowodzenie

Powód: brak wsparcia dla mechanizmu budującego **bazel.** Na raspbianie nie ma wsparcia dla zarządzania repozytoriami

Szansa:
https://github.com/samjabrahams/tensorflow-on-raspberry-pi/blob/master/GUIDE.md

Próba 2 – źródła, zbudowana paczka

Znalezienie przygotowanie paczki i zainstalowanie jej przez **pip**.

Udało się znaleźć paczkę, ale dla wersji python 3.5.

Zatem kolejnym krokiem była zmiana środowiska z python 3.7.2 na python 3.5.

Do tego zabiegu należy wykorzystać **conda**.

Na raspbianie, można zainstalować tylko odchudzoną wersję **minconda3**.

Efekt: Utworzenie wirtualnego środowiska z pythonem 3.5

Próba 2 c.d

Próba zainstalowania przygotowanej paczki, źródło:

http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-armv7l.sh

Efekt: Niepowodzenie, występują błędy podczas instalacji modułu **scipy** w najnowszej wersji 1.4.1

Próba 2 - błąd scipy

```
Collecting scipy==1.4.1; python_version >= "3"
  Using cached scipy-1.4.1.tar.gz (24.6 MB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... error

    …

    File "/tmp/pip-build-env-_3rvft_o/overlay/lib/python3.5/site-
    packages/numpy/__init__.py", line 152, in <module>
         test = testing.nosetester._numpy_tester().test
   NameError: name 'testing' is not defined
```

Próba 2 – zainstalowanie scipy

Próby rozwiązania:

1. Instalowanie poprzez pip ze wskazaną wersją

   Efekt: **Niepowodzenie**, taki sam błąd.

2. Zainstalowanie ze zbudowanych paczek

   Efekt: **Niepowodzenie** – najnowsza dostępna wersja dla python3.5 to scipy 1.3.3

3. Zainstalowanie poprzez **conda**

   Efekt: **Niepowodzenie** – instalowania wersja 1.0.0

Próba3 – źródła, budowanie z docker’em

Zbudowanie paczki whl przy pomocy docker’a według tutoriala na stronie tensorflow:

https://www.tensorflow.org/install/source_rpi

Efekt: Niepowodzenie

Próba 3 – problemy z instalacją docker’a

Nie możemy zainstalować docker’a na raspiabnie przy pomocy `set up repositories`. Na raspbianie nie ma możliwości zarządzania repozytoriami przy pomocy poleceni **add-apt-repository**.

Należy zainstalować przy pomocy **convenienve script**.

Efekt: Docker pomyślnie zainstalowany

Próba 3 – powód niepowodzenia

Budowanie obrazu, który ma za zadanie utworzyć paczkę .whl kończy się niepowodzenie spowodowanym przez **scipy** w wersji 1.2.2.

Błąd:

```
Collecting scipy==1.2.2
  Downloading scipy-1.2.2.tar.gz (23.1 MB)
Building wheels for collected packages: scipy
  Building wheel for scipy (setup.py): started
  Building wheel for scipy (setup.py): finished
with status 'error'
```

Próba3 - traceback

```
Traceback (most recent call last):
      File "<string>", line 1, in <module>
      File "/tmp/pip-install-HsRvn4/scipy/setup.py", line 492, in
<module>
        setup_package()
      File "/tmp/pip-install-HsRvn4/scipy/setup.py", line 488, in
setup_package
        setup(**metadata)
      File
"/usr/local/lib/python2.7/dist-packages/numpy/distutils/core.py",
line 135, in setup
        config = configuration()
      File "/tmp/pip-install-HsRvn4/scipy/setup.py", line 395, in
configuration
        raise NotFoundError(msg)
    numpy.distutils.system_info.NotFoundError: No lapack/blas
resources found.
```

Czemu obraz używa python 2.7?

- Nie mam pojęcia czemu to jest tak uruchomione
- Uruchomienie skryptu do zbudowania obrazów dockerowych:

```
CI_DOCKER_EXTRA_PARAMS="-e CI_BUILD_PYTHON=python3 -e \
    CROSSTOOL_PYTHON_INCLUDE_PATH=/usr/include/python3.7" \
    tensorflow/tools/ci_build/ci_build.sh PI-PYTHON3 \
    tensorflow/tools/ci_build/pi/build_raspberry_pi.sh
```