Joint Research Centre (JRC)

Earth Observation
& Social Sensing
BigData pilot project
EO&SS@BigData

Towards an infrastructure for interactive Earth Observation data analysis and processing

A. Burger and P. Soille

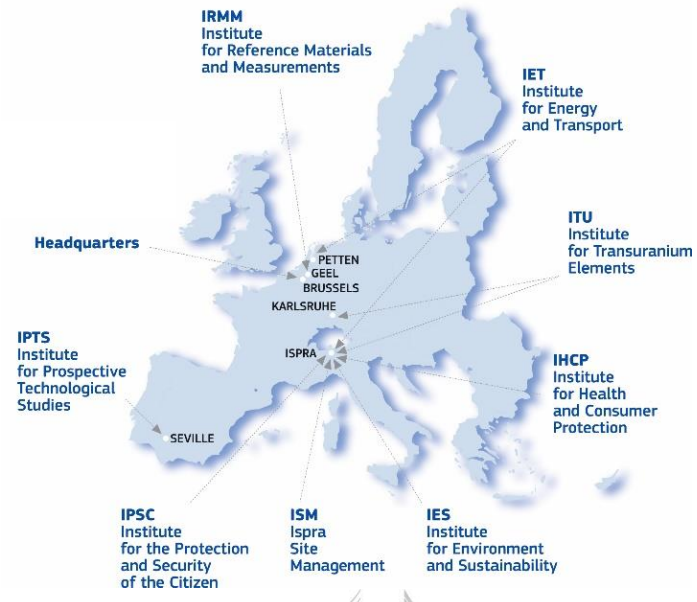
Contacts:Armin.Burger@jrc.ec.europa.eu
Pierre.Soille@jrc.ec.europa.eu

Joint Research Centre

CS3 Workshop, Zurich, 18-19th Jan 2016

The Joint Research Centre (JRC)

- JRC is the science service of the European Commission
- JRC provides independent scientific support to EU policy making
- Wide usage of **Earth Observation [EO] data** as basis for research and policy support



Joint Research Centre

European Commission

"Earth Observation & Social Sensing Big Data Pilot Project"

- The EU ***Copernicus*** Programme with the ***Sentinel*** fleet of satellites acts as a game changer by bringing EO in the Big Data era:
  - *expected 10TB/day of **free and open** data*
  - *Requires new approaches for data management and processing*
- Pilot project launched in January 2015
- Major goal: set up a central infrastructure for storing and processing of Earth Observation and Social Sensing data at JRC

Sentinel-1 (Credits: ESA/P. Carril)

Sentinel-2 (Credits: ESA/P. Carril)

Sentinel-3 (Credits: ESA/J. Huart)

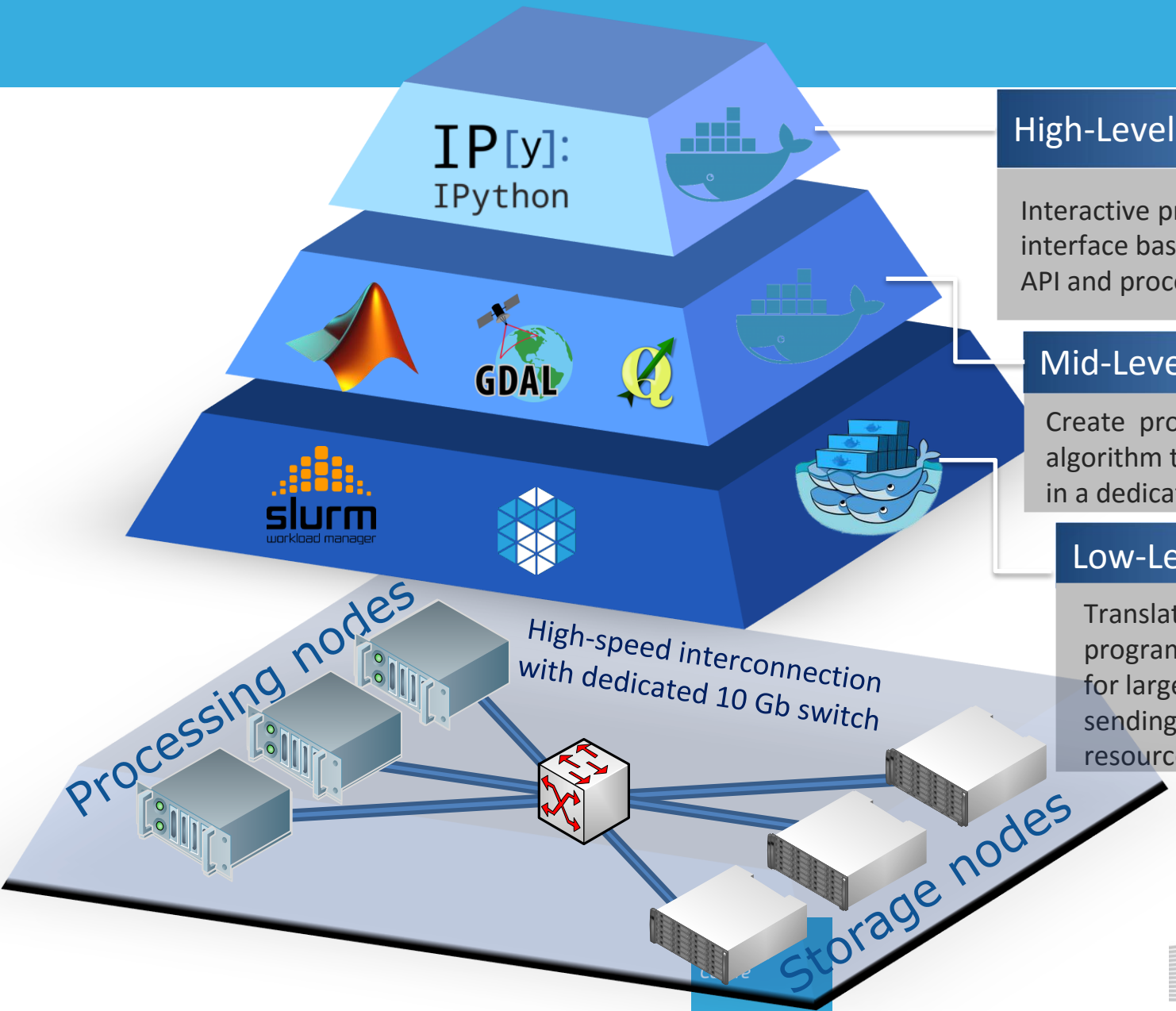
Joint Research Centre

European Commission

Proposal for a "JRC Earth Observation Data Processing Platform" (JEO-DPP)

- Main focus on **satellite image** data
- Shall support existing processing workflows and environments (C/C++, Python, Matlab, Java)
- Provide different processing levels:
  - *Low-level batch processing*
  - *High-level interactive processing*
- Project timeline:
  - *Prototype development: end 2015 – mid 2017*
  - *Scaling-up in 2017/18:*
    JRC Data Centre vs a public cloud solution

JEO-DPP processing components



High-Level Processing

Interactive programming via a Web interface based on Python with custom API and processing modules.

Mid-Level Processing

Create prototype programs for algorithm tests on limited sample data in a dedicated Docker container.

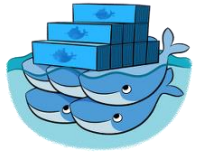
Low-Level Processing

Translate the prototype programs to an operational tool for large-scale processing - sending jobs through the resource manager.

European Commission

Low-level batch processing

- Running large-scale data processing tasks in a cluster environment
- *Docker containers* for flexible management of processing environments
  - *Custom builds for different requirements*
  - *Facilitates upgrades of processing environment (libraries, tools)*
- Run through a workload manager
  - *Using SLURM scheduler*
  - *Usage of MESOS as backend to be evaluated*
    Advantage: better integration with Docker environment



High-level interactive processing

- Web user interface to server-based data processing
- Based on ***IPython Notebook*** (*Jupyter*)
- Development of a data analysis and processing API
  - *Python as core, with C/C++ modules*
  - *Incorporate modules developed by various projects*
- Community building
  - *Sharing expertise and analysis algorithms*
  - *Share and extend existing Notebooks*

IPython/Jupyter interactive analysis

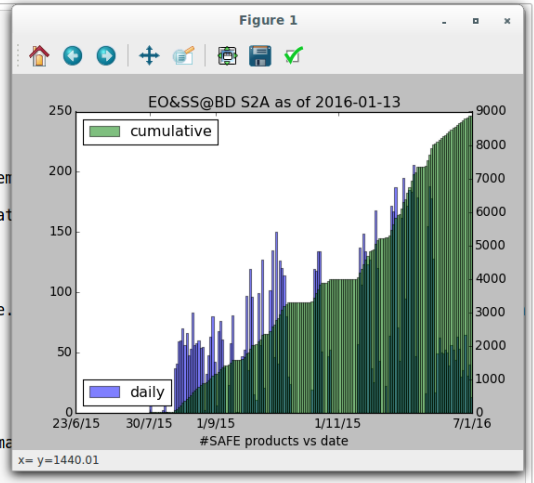
S2A_throughput - Mozilla Firefox

jupyter S2A_throughput Last Checkpoint: 5 hours ago Autosave Failed!

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

```
In [*]: fig, ax1=plt.subplots()
date_array=[]
dateint_array=[]
yday_array=[]
jday_array=[]
for indir in paths:
    for item in os.listdir(indir):
        if item.endswith('.SAFE'):
            adate=re.sub(r'[a-zA-Z_][a-zA-Z_0-9]*_V', '', item
            adate=re.sub(r'T.*', '', adate)
            dt=datetime.datetime.strptime(adate, '%Y%m%d').dat
            dateint_array.append(int(adate))
            date_array.append(dt)
            tt = dt.timetuple()
            yday_array.append(tt.tm_yday)
            jday_array.append(jdcal.gcal2jd(datetime.datetime.
dlaunch=jdcal.gcal2jd(2015, 6, 23)[1]
dmax=max(jday_array)
dmin=min(jday_array)
colors = ['blue', 'green']
ax1.set_xlabel(r'#SAFE products vs date')
#ax1.set_ylabel('daily', color='b')
ax1.hist(jday_array,bins=dmax-dmin+1,alpha=0.5,range=(dmin,dma
plt.legend(loc=3)
ax2=ax1.twinx()
ax2.hist(jday_array,bins=dmax-dmin+1,range=(dmin,dmax),alpha=0.5, cumulative=True, color='g', label='cumulative')
plt.legend(loc=2)
plt.xlim(dlaunch,dmax)
#plt.ylabel('cumulative', color='g')
plt.xlabel('SAFE products')
plt.title('EO&SS@BD S2A as of '+str(datetime.date.today()))
#plt.legend(loc='upper left')
plt.xticks([dlaunch, dmin, jdcal.gcal2jd(2015,9,1)[1], jdcal.gcal2jd(2015,11,1)[1],dmax],
           ['23/6/15',
            str(jdcal.jd2gcal(2400000.5, dmin)[2])+'/'+str(jdcal.jd2gcal(2400000.5, dmin)[1])+'/15',
            str(jdcal.jd2gcal(2400000.5, jdcal.gcal2jd(2015,9,1)[1])[2])+'/'+str(jdcal.jd2gcal(2400000.5, jdcal.gcal2j
            str(jdcal.jd2gcal(2400000.5, jdcal.gcal2jd(2015,11,1)[1])[2])+'/'+str(jdcal.jd2gcal(2400000.5, jdcal.gcal2
```

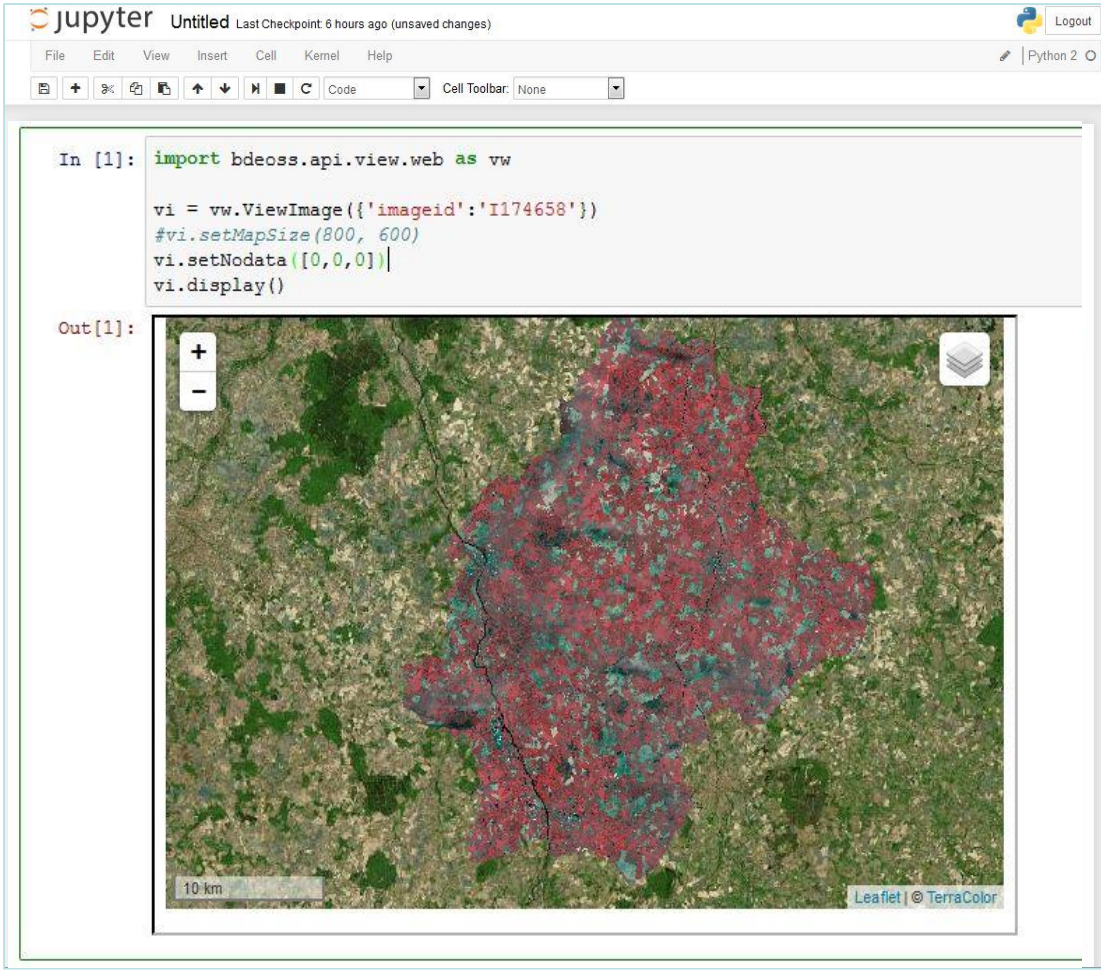


Joint Research Centre

European Commission

Interactive processing - visualisation

- Visualisation of intermediate results
- Interaction with pre-defined processing and visualization Web services



Joint Research Centre

European Commission

Jupyter with GDAL-Python API and rendering in desktop client

jupyter tissot_UTM-4-S2A Last Checkpoint: 12/17/2015 (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

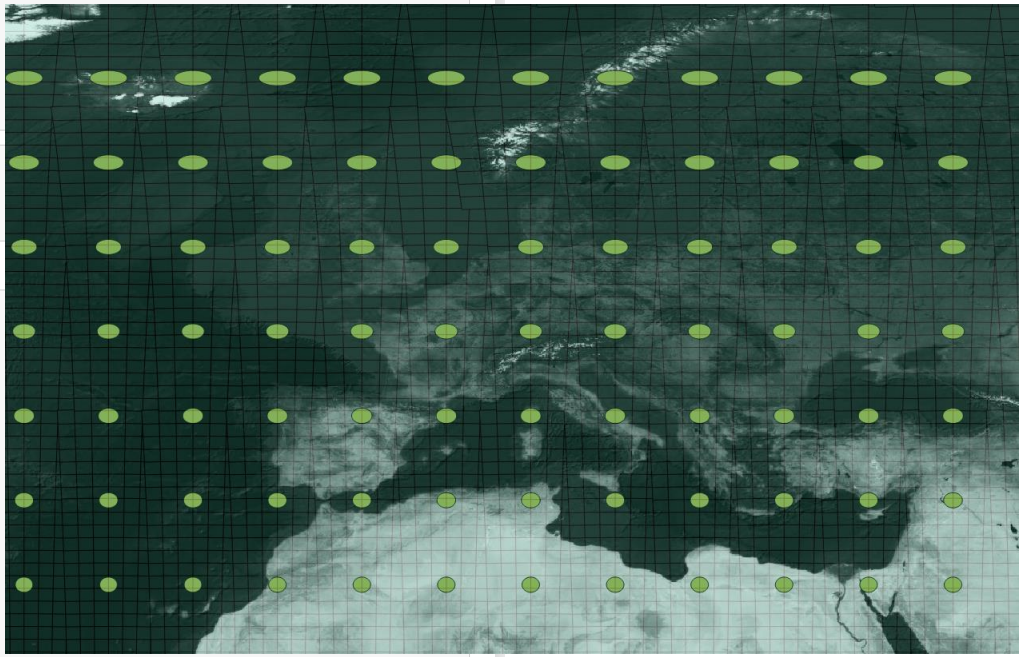
```
Out[3]: array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
```

In [4]:
```
# Create the Shapefile
driver = ogr.GetDriverByName(DriverName)
if os.path.exists(output):
     driver.DeleteDataSource(output)
ds = driver.CreateDataSource(output)
layer = ds.CreateLayer(output, geom_type=ogr.wkbPolygon)
```

In [5]:
```
# Set up spatial reference systems
latlong = osr.SpatialReference()
ortho = osr.SpatialReference()
latlong.ImportFromProj4('+proj=latlong')
```

```
Out[5]: 0
```

In [6]:
```
# For each grid point, reproject to ortho centered on itself,
# buffer by x kilometers, reproject back to latlong,
# and output the latlong ellipse to shapefile
startx=-180+resx/2
starty=-90+resy
for x in np.arange(startx,180,resx):
     for y in np.arange (starty,90,resy):
          f= ogr.Feature(feature_def=layer.GetLayerDefn())
          wkt = 'POINT(%f %f)' % (x, y)
          p = ogr.CreateGeometryFromWkt(wkt)
          p.AssignSpatialReference(latlong)
          proj = '+proj=ortho +lon_0=%f +lat_0=%f' % (x,y)
          ortho.ImportFromProj4(proj)
          p.TransformTo(ortho)
          b = p.Buffer(r*1000)
          b.AssignSpatialReference(ortho)
          b.TransformTo(latlong)
          f.SetGeometryDirectly(b)
          layer.CreateFeature(f)
          f.Destroy()
ds.Destroy()
```



Interactive Processing – set-up and interfacing

- User Notebooks running in *Docker* containers
  - *Separating user environments*
  - *Allows for detailed resource allocation*
  - *Web interface for managing user Notebooks*
  - *Reverse proxy set-up for accessing Docker instances*

- Interaction with workload manager
  - *Launching batch processing from Notebooks*
    => running large-scale processing from your browser

Web services for everyone ?

- Public or restricted ?
- Data view services via standard protocols
- Web processing services
  - *Embeddable in Web applications or called from processing scripts*
  - *For example for*
    - Image sub-setting: areas of interest, band combination,
    - Image compositing
    - Cloud-free image mosaicking
    - Atmospheric correction
    - …

Using public Cloud solutions

- Shall be evaluated for the scaling-up phase
  - *JRC Data Centre vs Public Cloud*
  - *Possible scenarios with mixed environments JRC <-> Public Cloud*
- Interfacing between local and public cloud infrastructure
  - *Docker containers for portability of processing environment*
  - *Seamless distributed processing*
  - *Issue is the availability of input data*
  - *Distributed file system for data sharing*
- Location of processing transparent for users

Thank you for your attention !