Cloud-based Cluster Computing for PyData

Olivier Grisel - SUCCES, Paris 2016

Outline

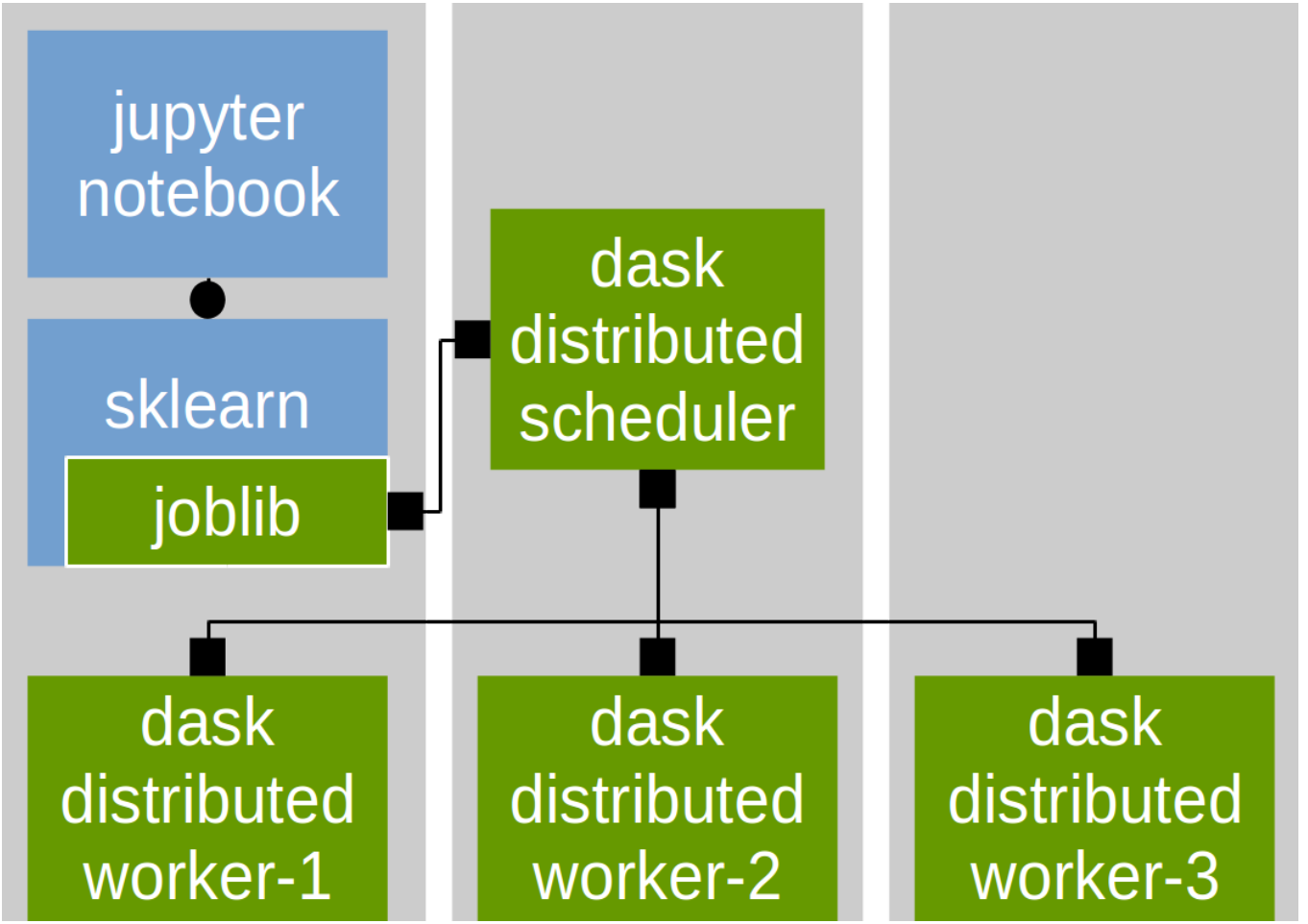
Demo of distributed ML in Python

Some PyData tools for interactive distributed computing

Cluster orchestration with Docker and Kubernetes


Distributed ML with joblib
and dask/distributed

A cluster for sklearn



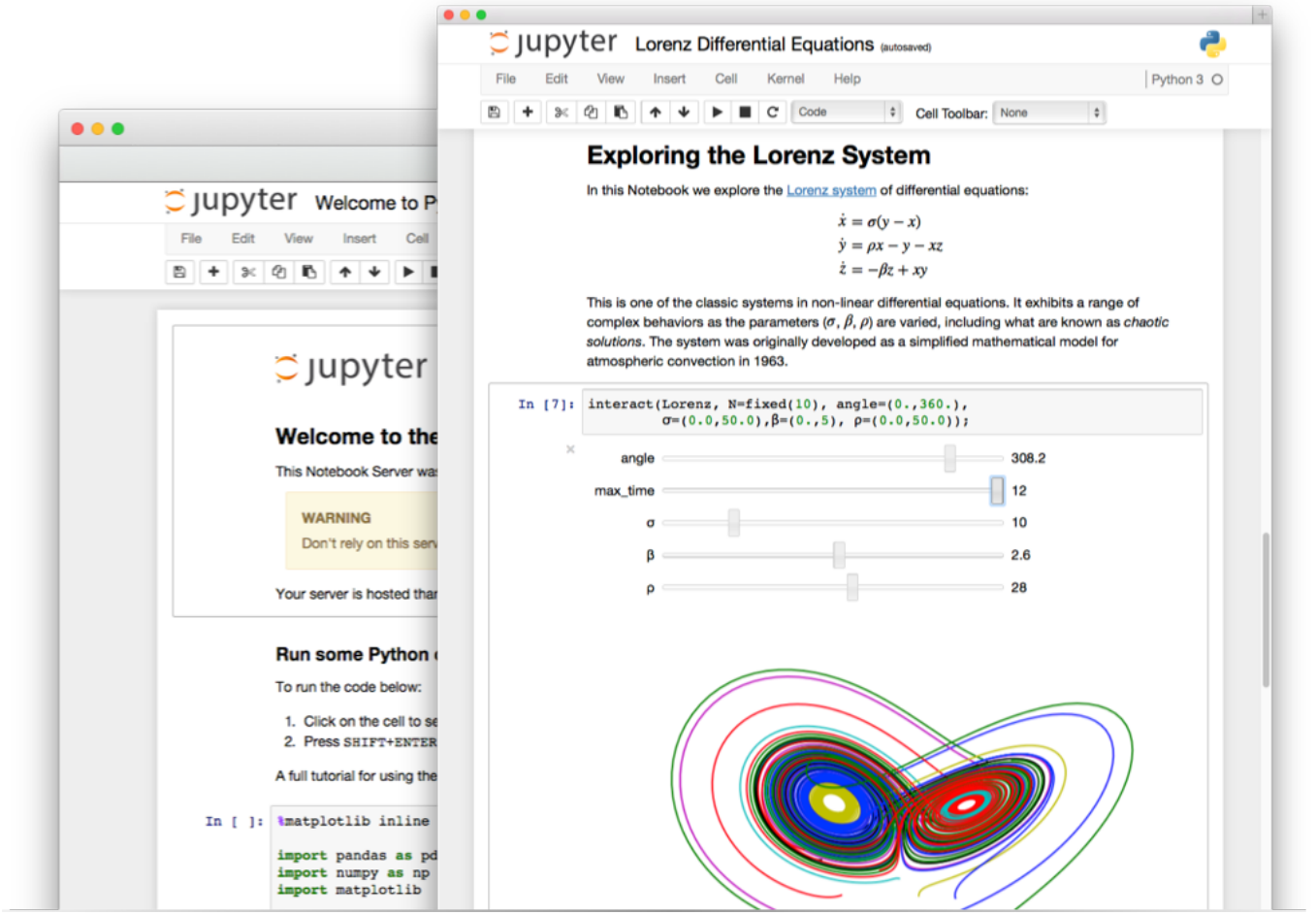
Demo

https://github.com/ogrisel/docker-distributed

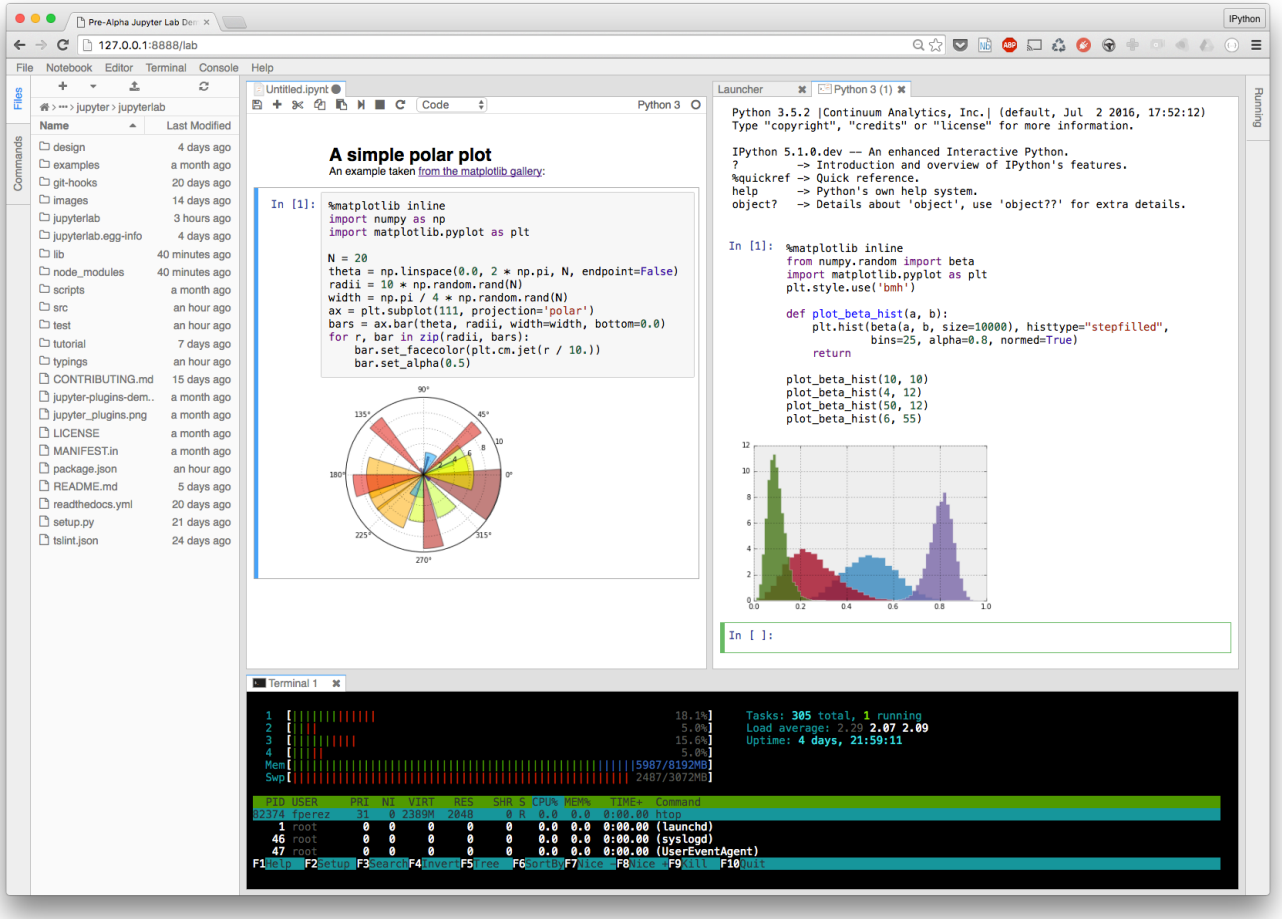
Have a look a the notebooks in the `examples` folder.

https://youtu.be/6mKNSEQ0FIQ

Jupyter notebook



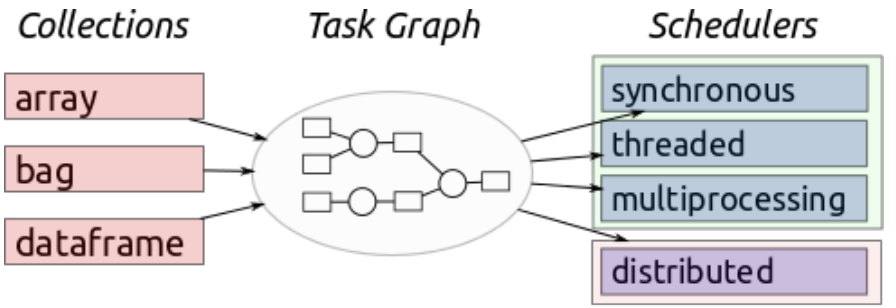
JupyterLab


Pre-Alpha Jupyter Lab Demo
IPython
127.0.0.1:8888/lab
File Notebook Editor Terminal Console Help
Files
Commands
Running
jupyter › jupyterlab
Name
Last Modified
design 4 days ago
examples a month ago
git-hooks 20 days ago
images 14 days ago
jupyterlab 3 hours ago
jupyterlab.egg-info 4 days ago
lib 40 minutes ago
node_modules 40 minutes ago
scripts a month ago
src an hour ago
test an hour ago
tutorial 7 days ago
typings an hour ago
CONTRIBUTING.md 15 days ago
jupyter-plugins-dem.. a month ago
jupyter_plugins.png a month ago
LICENSE a month ago
MANIFEST.in a month ago
package.json an hour ago
README.md 5 days ago
readthedocs.yml 20 days ago
setup.py 21 days ago
tslint.json 24 days ago
Untitled.ipynb
Code
Python 3
A simple polar plot
An example taken from the matplotlib gallery:
In [1]:
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
ax = plt.subplot(111, projection='polar')
bars = ax.bar(theta, radii, width=width, bottom=0.0)
for r, bar in zip(radii, bars):
bar.set_facecolor(plt.cm.jet(r / 10.))
bar.set_alpha(0.5)
Launcher
Python 3 (1)
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul 2 2016, 17:52:12)
Type "copyright", "credits" or "license" for more information.
IPython 5.1.0.dev -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]:
%matplotlib inline
from numpy.random import beta
import matplotlib.pyplot as plt
plt.style.use('bmh')
def plot_beta_hist(a, b):
plt.hist(beta(a, b, size=10000), histtype="stepfilled",
bins=25, alpha=0.8, normed=True)
return
plot_beta_hist(10, 10)
plot_beta_hist(4, 12)
plot_beta_hist(50, 12)
plot_beta_hist(6, 55)
In [ ]:
Terminal 1
Tasks: 305 total, 1 running
Load average: 2.29 2.07 2.09
Uptime: 4 days, 21:59:11
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
22374 fperez 31 0 2389M 2048 0 R 0.0 0.0 0:00.00 htop
1 root 0 0 0 0 0 0.0 0.0 0:00.00 (launchd)
46 root 0 0 0 0 0 0.0 0.0 0:00.00 (syslogd)
47 root 0 0 0 0 0 0.0 0.0 0:00.00 (UserEventAgent)
F1Help F2Setup F3SearchF4InvertF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit

dask & distributed

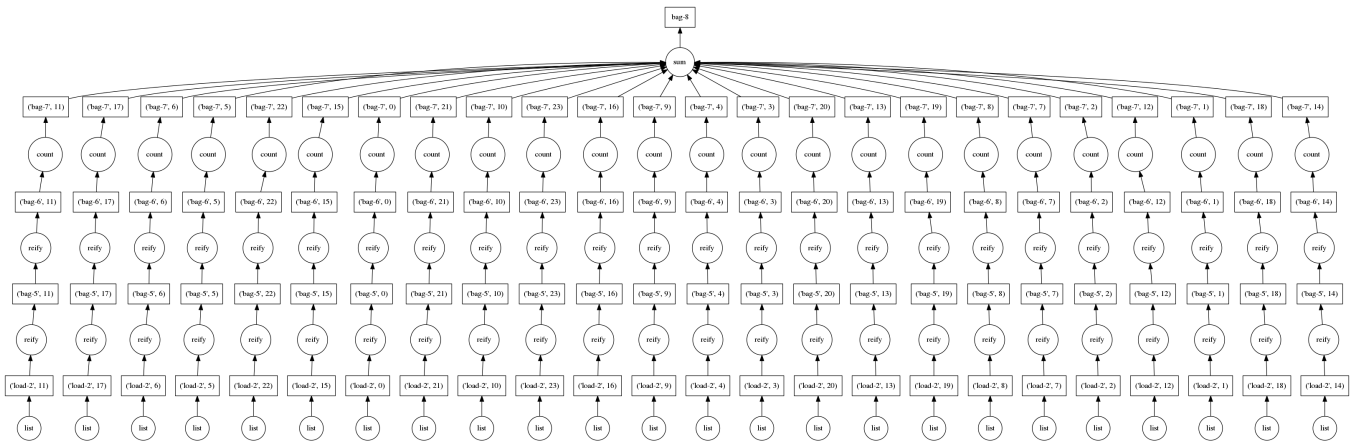
- Collection API similar to NumPy array and Pandas DataFrame objects
- Custom workloads via tasks scheduling API
- Pure python, easily integrated in PyData environment
- Low overhead (scheduling latency: ~1ms per tasks)
- Scales up: runs on clusters with 1000s of cores
- Scales down: runs on a laptop in a single process

dask architecture



dask bags & compute graphs

```
>>> import dask.bag as db
>>> b = db.from_s3('githubarchive-data', '2015-01-01-*.json.gz')
          .map(json.loads)
          .map(lambda d: d['type'] == 'PushEvent')
          .count()
```



dask arrays & compute graphs

```
>>> import dask.array as da
>>> x = da.ones((5000, 1000), chunks=(1000, 1000))
>>> u, s, v = da.linalg.svd(x)
```

dask collection-oriented API

Dask DataFrame mimics Pandas

```
import pandas as pd                     import dask.dataframe as dd
df = pd.read_csv('2015-01-01.csv')      df = dd.read_csv('2015-*-*.csv')
df.groupby(df.user_id).value.mean()     df.groupby(df.user_id).value.mean().compute()
```

Dask Array mimics NumPy

```
import numpy as np                      import dask.array as da
f = h5py.File('myfile.hdf5')            f = h5py.File('myfile.hdf5')
x = np.array(f['/small-data'])          x = da.from_array(f['/big-data'],
                                                          chunks=(1000, 1000))
x - x.mean(axis=1)                      x - x.mean(axis=1).compute()
```

Dask Bag mimics iterators, Toolz, and PySpark

```
import dask.bag as db
b = db.read_text('2015-*-*.json.gz').map(json.loads)
b.pluck('name').frequencies().topk(10, lambda pair: pair[1]).compute()
```

dask task-oriented API

Dask Delayed mimics for loops and wraps custom code

```
from dask import delayed
L = []
for fn in filenames:                    # Use for loops to build up computation
    data = delayed(load)(fn)            # Delay execution of function
    L.append(delayed(process)(data))  # Build connections between variables

result = delayed(summarize)(L)
result.compute()
```

The **concurrent.futures** interface provides general submission of custom tasks:

```
from dask.distributed import Client
client = Client('scheduler:port')

futures = []
for fn in filenames:
    future = client.submit(load, fn)
    futures.append(future)

summary = client.submit(summarize, futures)
summary.result()
```


Container-based
orchestraion with docker &
kubernetes

Docker is:

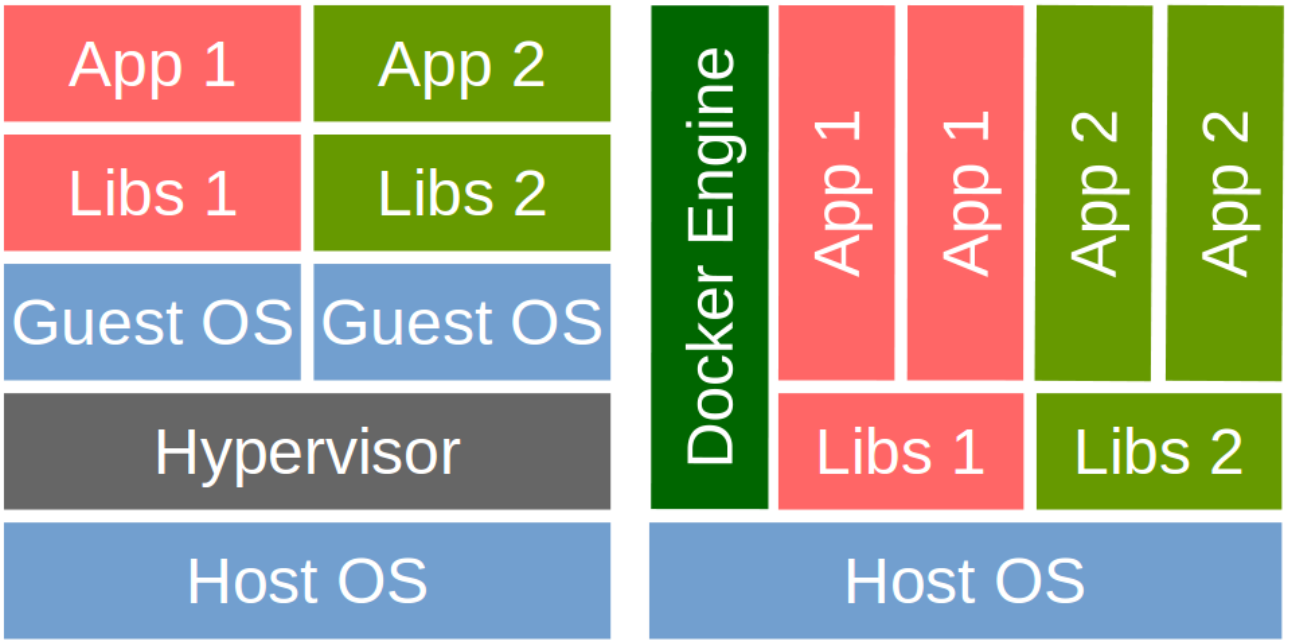
Not a virtual machine system

Linux container + Layered file-system + Abstract network

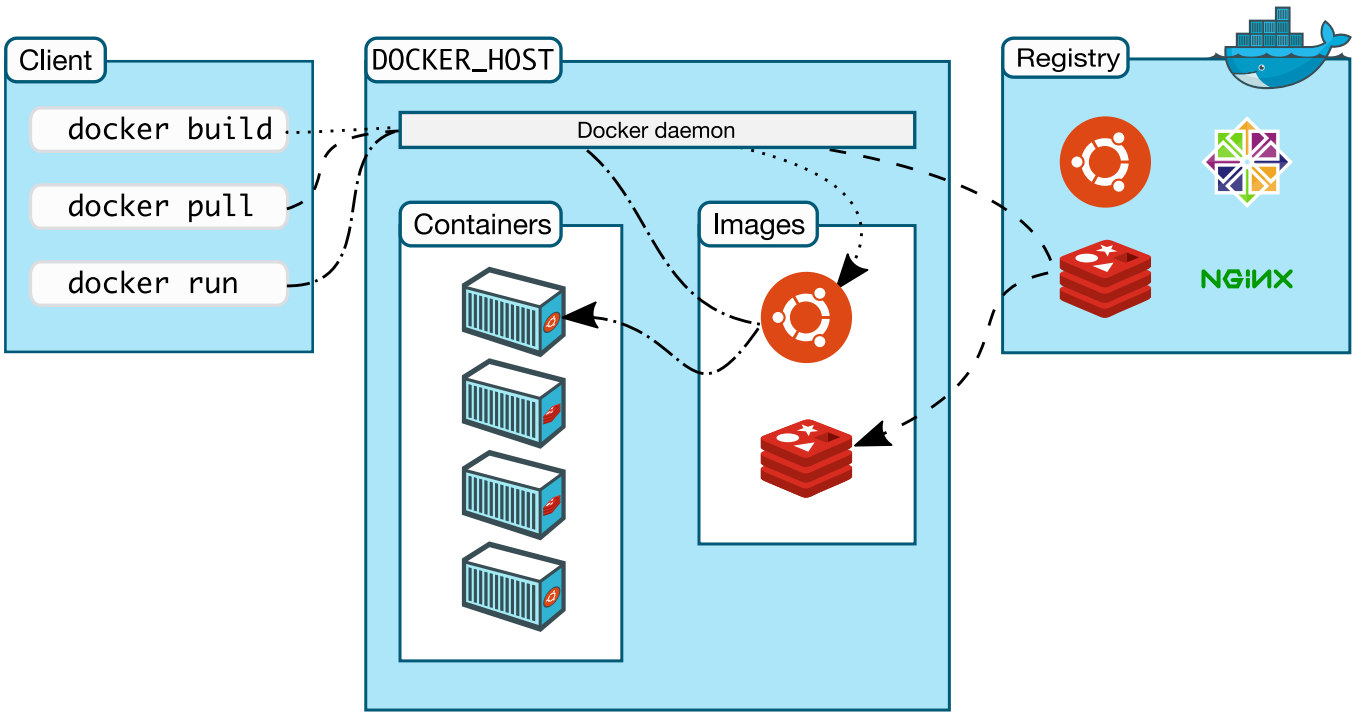
User-friendly command line interface

It can run in a VM (or not)

VM vs Container



Docker architecture



Building an image

```
$ git clone https://github.com/ogrisel/docker-distributed
$ ls docker-distributed
docker-compose.yml
Dockerfile
examples
kubernetes
README.md
requirements.txt
```

```
$ git clone https://github.com/ogrisel/docker-distributed
$ ls docker-distributed
docker-compose.yml
Dockerfile
examples
kubernetes
README.md
requirements.txt
```

```
$ git clone https://github.com/ogrisel/docker-distributed
$ ls docker-distributed
docker-compose.yml
Dockerfile
examples
kubernetes
README.md
requirements.txt
```

```
FROM debian:jessie
MAINTAINER Olivier Grisel <olivier.grisel@ensta.org>

RUN apt-get update -yqq && apt-get install -yqq wget bzip2 git \
  && rm -rf /var/lib/apt/lists/*

# Configure environment
ENV LC_ALL=C.UTF-8 LANG=C.UTF-8

RUN mkdir /work
WORKDIR /work

# Install Python 3 from miniconda
RUN wget -O miniconda.sh \
  https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh \
  && bash miniconda.sh -b -p /work/miniconda \
  && rm miniconda.sh

ENV PATH="/work/bin:/work/miniconda/bin:$PATH"

RUN conda install -y \
  pip \
  notebook \
  pandas \
  scikit-learn \
  && conda clean -tipsy

# Install the master branch of distributed and dask
COPY requirements.txt .
RUN pip install -r requirements.txt && rm -rf ~/.cache/pip/

# Add the example notebooks
COPY examples .
```

docker build

```
$ docker build -t ogrisel/distributed .
Sending build context to Docker daemon 178.2 kB
Step 1 : FROM debian:jessie
 ---> 1b01529cc499
Step 2 : MAINTAINER Olivier Grisel <olivier.grisel@ensta.org>
 ---> Using cache
 ---> 37887ee139f1
Step 3 : RUN apt-get update -yqq  && apt-get install -yqq wget [...]
 ---> Using cache
 ---> 3c2b8caccb80
[...]
```

docker build & docker push

```
$ docker build -t ogrisel/distributed .
Sending build context to Docker daemon 178.2 kB
Step 1 : FROM debian:jessie
 ---> 1b01529cc499
Step 2 : MAINTAINER Olivier Grisel <olivier.grisel@ensta.org>
 ---> Using cache
 ---> 37887ee139f1
Step 3 : RUN apt-get update -yqq  && apt-get install -yqq wget [...]
 ---> Using cache
 ---> 3c2b8caccb80
[...]
$ docker run --rm ogrisel/distributed \
    python -c "import sklearn; print(sklearn.__version__)"
0.18.1
$ docker push ogrisel/distributed
```


Orchestration

Orchestration tools

Docker Swarm and
Docker Compose

Kubernetes

DC/OS and Mesos

Kubernetes and GKE

```
$ gcloud config set compute/zone europe-west1-d
$ gcloud container clusters create cluster-1 \
    --num-nodes 3 \
    --machine-type n1-highcpu-32 \
    --scopes bigquery,storage-rw \
    --preemptible \
    --no-async
Creating cluster cluster-1...done.
Created [https://container.googleapis.com/v1/.../cluster-1].
kubeconfig entry generated for cluster-1.
NAME       ZONE            ... MACHINE_TYPE   NUM_NODES  STATUS
cluster-1  europe-west1-d  ... n1-highcpu-32  3          RUNNING
```

Kubernetes and GKE

```
$ gcloud config set compute/zone europe-west1-d
$ gcloud container clusters create cluster-1 \
    --num-nodes 3 \
    --machine-type n1-highcpu-32 \
    --scopes bigquery,storage-rw \
    --preemptible \
    --no-async
Creating cluster cluster-1...done.
Created [https://container.googleapis.com/v1/.../cluster-1].
kubeconfig entry generated for cluster-1.
NAME       ZONE            ... MACHINE_TYPE   NUM_NODES  STATUS
cluster-1  europe-west1-d  ... n1-highcpu-32  3          RUNNING

$ gcloud container clusters get-credentials cluster-1
Fetching cluster endpoint and auth data.
kubeconfig entry generated for cluster-1.
```

Kubernetes and GKE

```
$ git clone https://github.com/ogrisel/docker-distributed
$ cd docker-distributed
$ kubectl create -f kubernetes/
service "dscheduler" created
service "dscheduler-status" created
replicationcontroller "dscheduler" created
replicationcontroller "dworker" created
service "jupyter-notebook" created
replicationcontroller "jupyter-notebook" created
```

Kubernetes and GKE

```
$ git clone https://github.com/ogrisel/docker-distributed
$ cd docker-distributed
$ kubectl create -f kubernetes/
service "dscheduler" created
service "dscheduler-status" created
replicationcontroller "dscheduler" created
replicationcontroller "dworker" created
service "jupyter-notebook" created
replicationcontroller "jupyter-notebook" created

$ kubectl get services
NAME                CLUSTER-IP       EXTERNAL-IP      PORT(S)
dscheduler          10.115.249.189   <none>           8786/TCP,9786/T
dscheduler-status   10.115.244.201   130.211.50.206   8787/TCP
jupyter-notebook    10.115.254.255   146.148.114.90   80/TCP
kubernetes          10.115.240.1     <none>           443/TCP
```

distributed-worker.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: dworker-controller
spec:
  replicas: 3
  selector:
    name: dworker
  template:
    metadata:
      labels:
        name: dworker
    spec:
      containers:
        - name: dworker
          image: ogrisel/distributed:latest
          args: ["dask-worker", "dscheduler:8786"]
```

Conclusion

Jupyter + Dask a nice interactive distributed environement for PyData

Containers make cluster setup reproducible and cloud agnostic

Docker Swarm vs Kubernetes vs Mesos: learn any one and understand others

Thank you

Rackspace for free cloud resources to support SciPy projects and for helping the Python community in general.

Google for GCP credits to test distributed Python with Kubernetes.

Inria for supporting my work on scikit-learn and related projects.

Thank you!

Sample configuration:

- github.com/ogrisel/docker-distributed

Those slides:

- ogrisel.github.io/decks/2016_succes_paris_pydata_kubernetes

Image credits

- "Supermicro Storage" by Robert: https://flic.kr/p/e17qr3
- "smiiile!" by mooglet: https://flic.kr/p/am9zQe
- "Docker Architecture" https://docs.docker.com/engine/understanding-docker/
- "Container terminal" by Martin Abegglen: https://flic.kr/p/b7rw5D
- "Symphonic Orchestra and Choir of the WYD" by HazteOir.org: https://flic.kr/p/c5B6Cd