Building reproducible distributed applications at scale

Fabian Höring, Criteo
@f_hoering

The machine learning platform at Criteo

APACHE Spark™

TensorFlow

jupyter

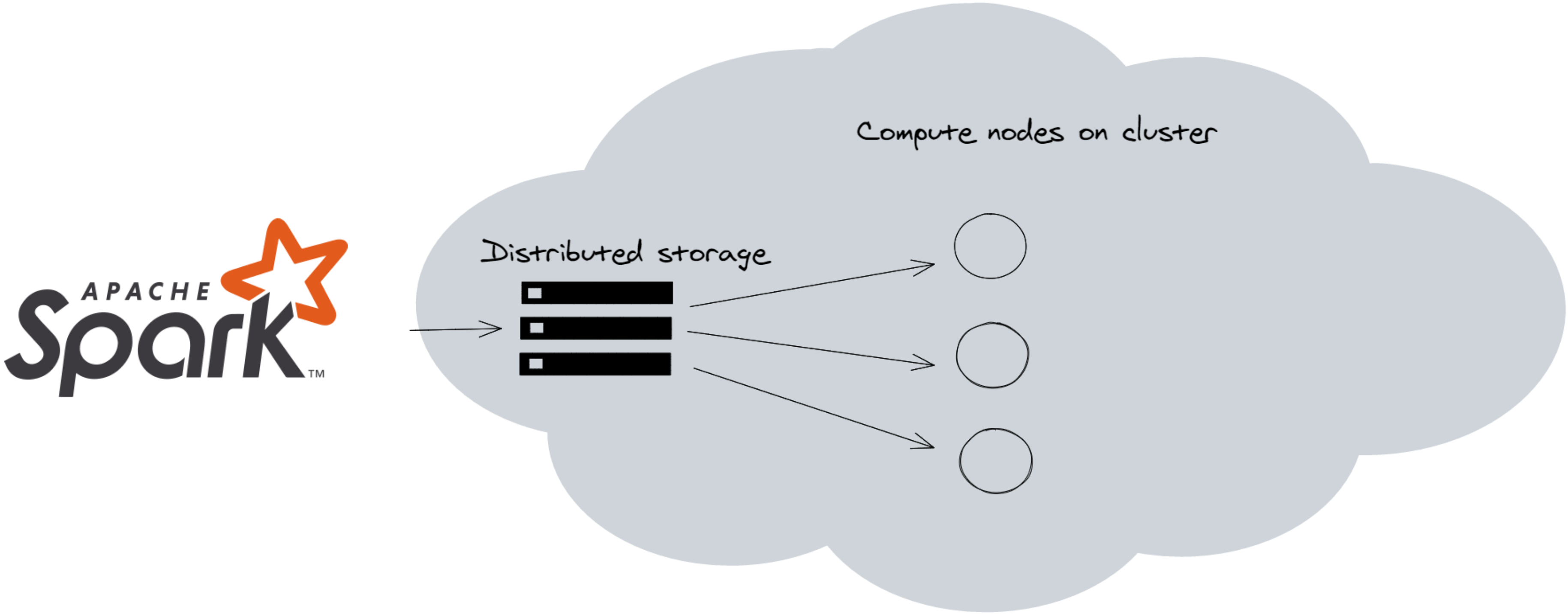
scikit learn

mlflow

pandas

matplotlib

Run a PySpark job on the cluster



PySpark example with Pandas UDF

```
df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), (2, 5.0), (2, 10.0)],
    ("id", "v"))

def mean_fn(v: pd.Series) -> float:
    return v.mean()

mean_udf = pandas_udf(mean_fn,
                      "double", PandasUDFType.GROUPED_AGG)
df.groupby("id").agg(mean_udf(df['v'])).toPandas()
```

Running with a local spark session

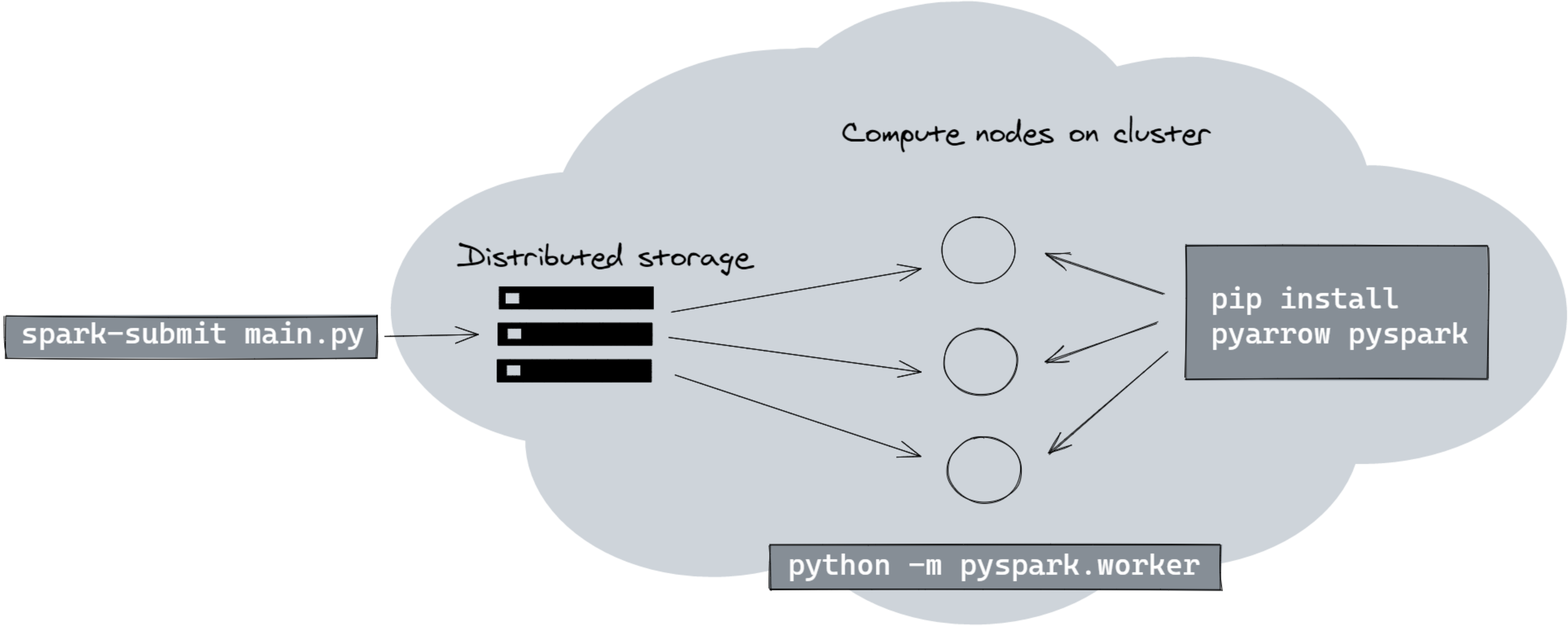
```
(venv) [f.horing]$ pyspark --master=local[1]
--deploy-mode=client
>>> ..
>>> df.groupby("id").agg(
        mean_udf(df['v'])).toPandas()
id  mean_fn(v)
0   1         1.5
1   2         6.0
>>>
```

Running on Apache YARN

```
(venv) [f.horing]$ pyspark --master=yarn
--deploy-mode=client
>>> ..
>>> df.groupby("id").agg(
        mean_udf(df['v'])).toPandas()
```

```
[Stage 1:>
(0 + 2) / 200]20/07/13 13:17:14 WARN
scheduler.TaskSetManager: Lost task 128.0 in stage 1.2 (TID
32, 48-df-37-48-f8-40.am6.hpc.criteo.prod, executor 4):
org.apache.spark.api.python.PythonException: Traceback (most
recent call last):  File "/hdfs/uuid/75495b8a-bbfe-41fb-913a-
330ff6132ddd/yarn/data/usercache/f.horing/appcache/applicatio
n_1592396047777_3446783/container_e189_1592396047777_3446783_
01_000005/pyspark.zip/pyspark/sql/types.py", line 1585, in
to_arrow_type
    import pyarrow as pa
ModuleNotFoundError: No module named 'pyarrow'
```

Running code on a cluster
installed globally

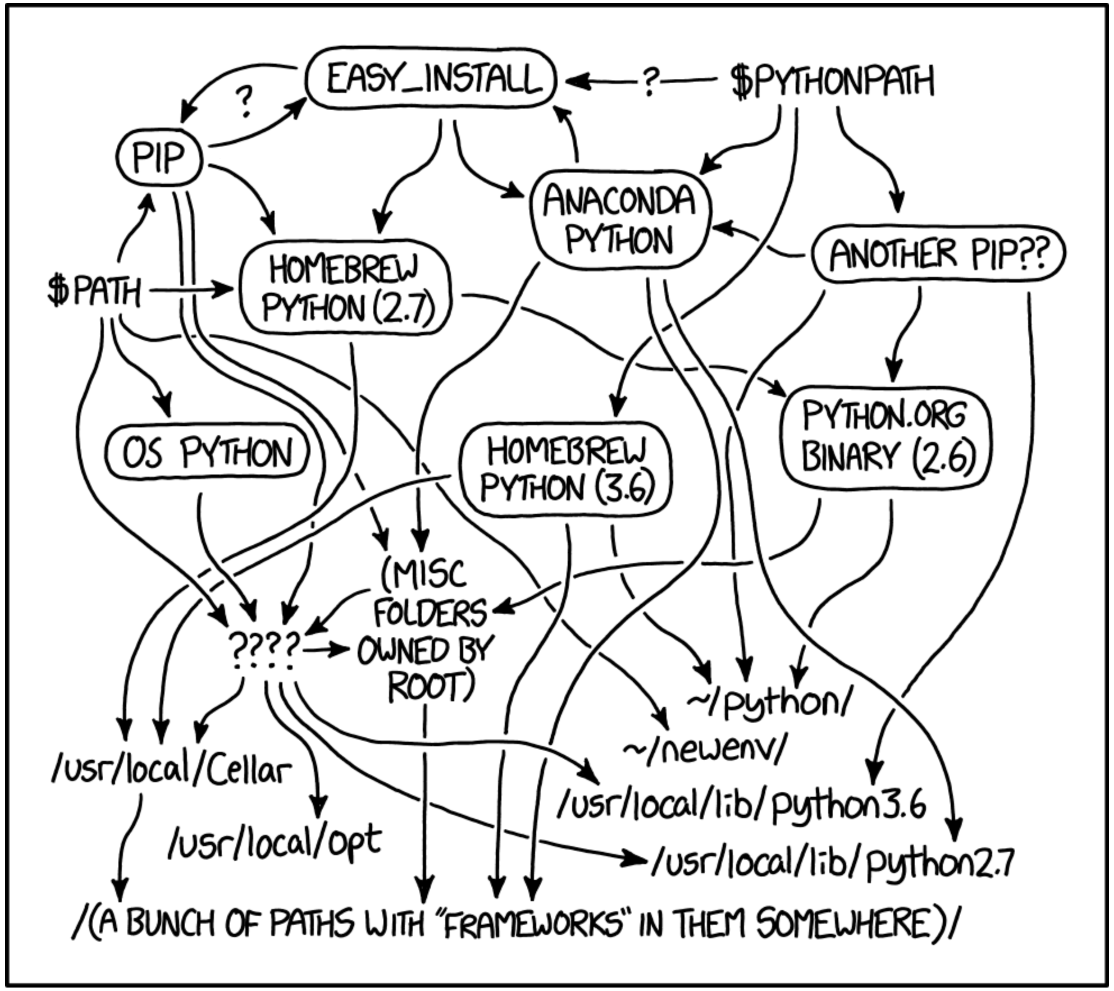


```
spark-submit main.py
```

```
pip install
pyarrow pyspark
```

```
python -m pyspark.worker
```

We want to launch a new application with another version of Spark



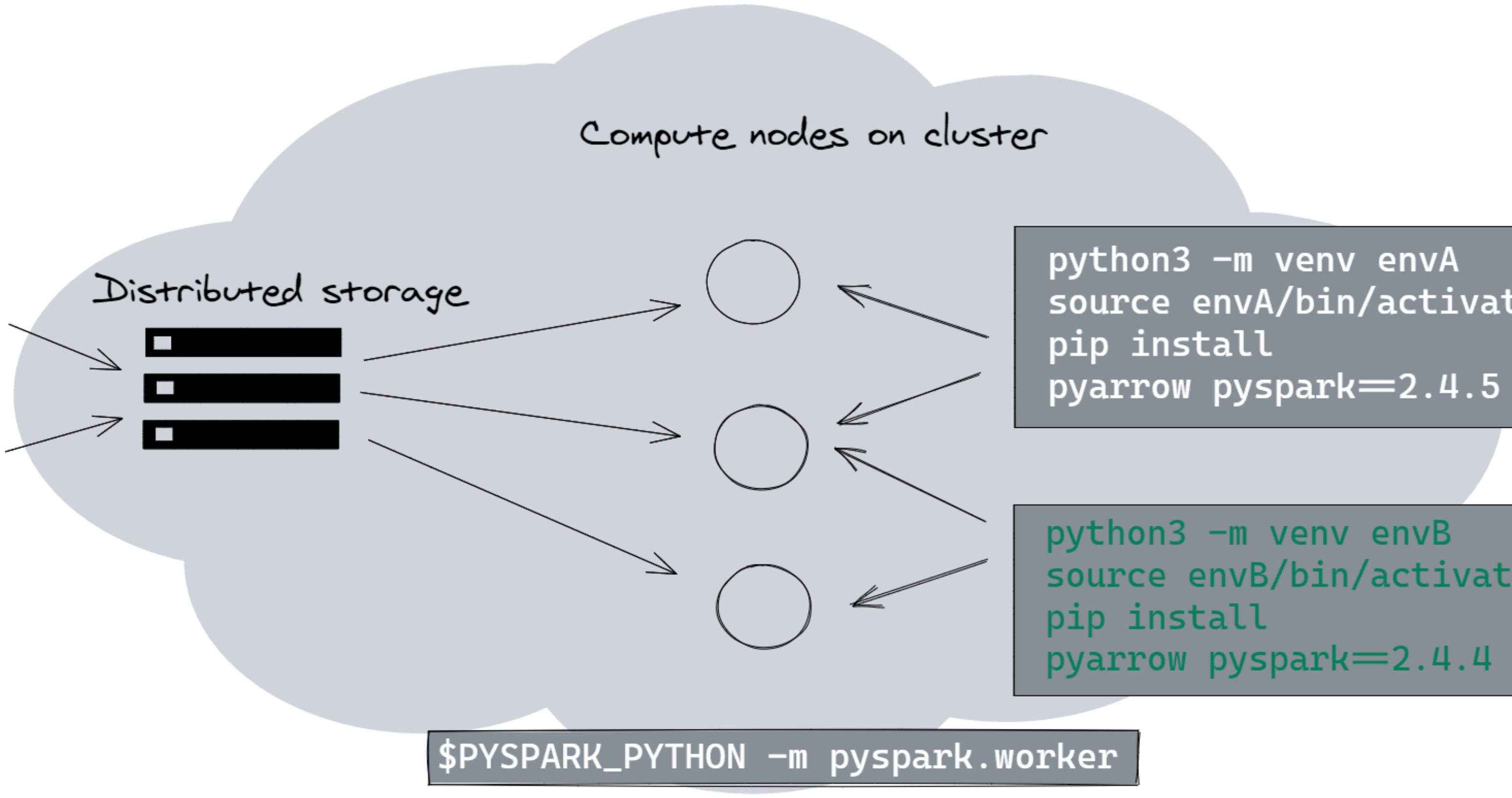
MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

https://xkcd.com/1987/

Running code on a cluster
installed in a Virtual Env

```
export
PYSPARK_PYTHON=envA/bin/python
spark-submit main.py
```

```
export
PYSPARK_PYTHON=envB/bin/python
spark-submit main.py
```



```
python3 -m venv envA
source envA/bin/activate
pip install
pyarrow pyspark==2.4.5
```

```
python3 -m venv envB
source envB/bin/activate
pip install
pyarrow pyspark==2.4.4
```

```
$PYSPARK_PYTHON -m pyspark.worker
```

A new version of Spark is released

```
(env) [f.horing]$ pip install pyspark
Looking in indexes: http://build-
nexus.prod.crto.in/repository/pypi/simple
Collecting pyspark
  Downloading http://build-
nexus.prod.crto.in/repository/pypi/files.pythonhosted.org/ht
tps/packages/8e/b0/bf9020b56492281b9c9d8aae8f44ff51e1bc91b3e
f5a884385cb4e389a40/pyspark-3.0.0.tar.gz (204.7 MB)
```

```
File
"/mnt/resource/hadoop/yarn/local/usercache/livy/appcache/app
lication_XXX/container_XXX/virtualenv_application_XXX/lib/
python3.5/site-
packages/pip/_vendor/lockfile/linklockfile.py", line 31, in
acquire
 os.link(self.unique_name, self.lock_file)
 FileExistsError: [Errno 17] File exists:
 '/home/yarn/XXXXXXXX-XXXXXXXX' ->
'/home/yarn/selfcheck.json.lock'
```

From SPARK-13587 - Support virtualenv in PySpark

Building reproducible **distributed** applications **at scale**

One Machine Learning model is learned with several TB of Data

1000s of jobs are launched every day with Spark, TensorFlow and Dask

Building **reproducible** distributed applications at scale

Non determinism in Machine Learning

Initialization of layer weights

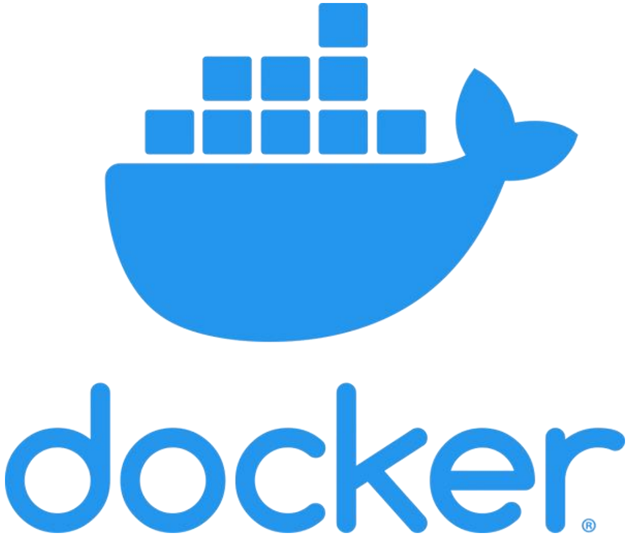
Dataset shuffling

Randomness in hidden layers: Dropout

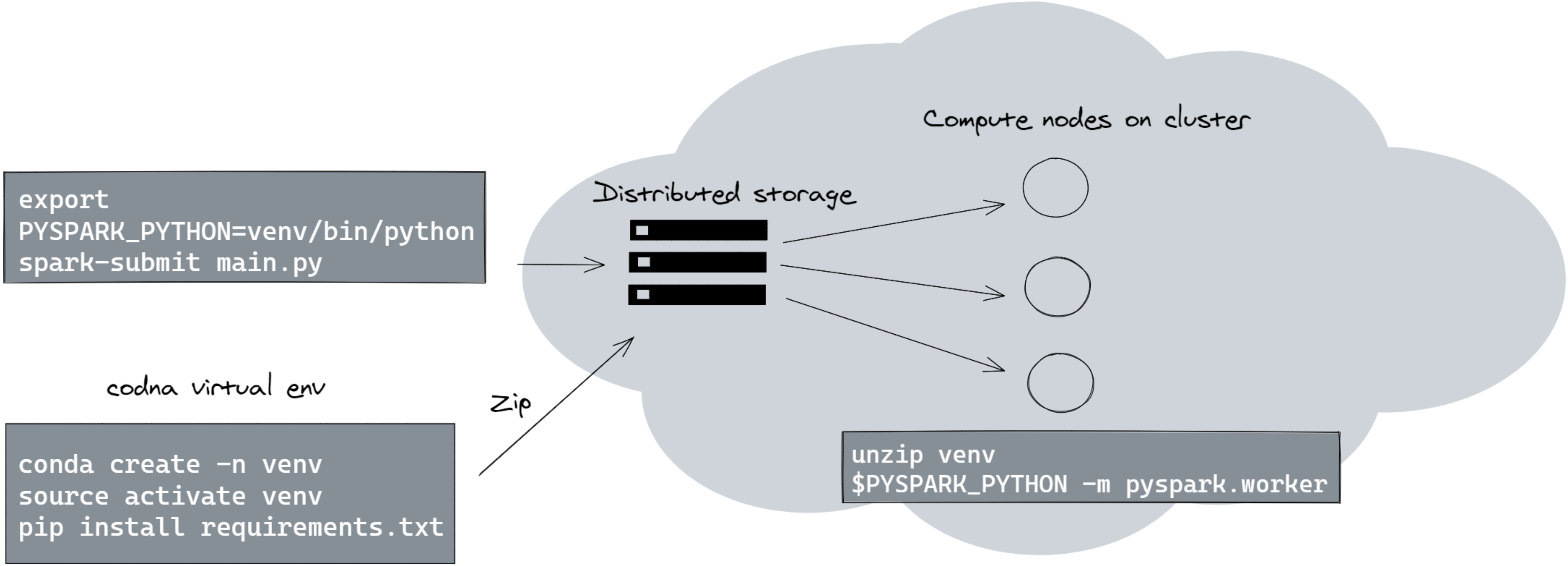
Updates to ML frameworks & libraries

We somehow need to ship the whole environment and then reuse it …

We could use



Using conda virtual envs



```
export
PYSPARK_PYTHON=venv/bin/python
spark-submit main.py
```

```
conda create -n venv
source activate venv
pip install requirements.txt
```

```
unzip venv
$PYSPARK_PYTHON -m pyspark.worker
```

We use our own internal private PyPi package repository

Problems with using conda & pip

> “Use pip only after conda
> Recreate the environment if changes are needed
> Use conda environments for isolation.”

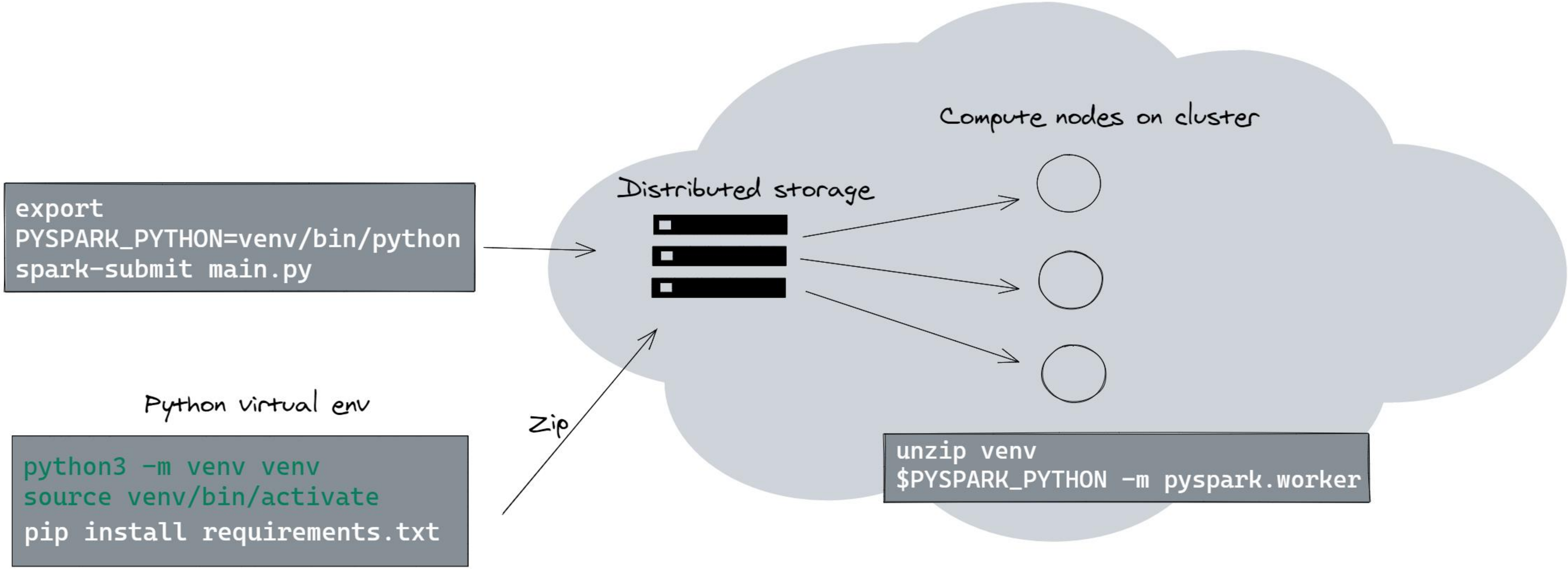
https://www.anaconda.com/blog/using-pip-in-a-conda-environment

Problems with using conda & pip

```
(venv) [f.horing] ~/$ pip install numpy
(venv) [f.horing] ~/$ conda install numpy
(venv) [f.horing] ~/$ conda list
# packages in environment at /home/f.horing/.criteo-conda/envs/venv:
...
mkl               2020.1          217
mkl-service       2.3.0           py36he904b0f_0
mkl_fft           1.1.0           py36h23d657b_0
mkl_random        1.1.1           py36h0573a6f_0
ncurses           6.2             he6710b0_1
numpy             1.19.0          pypi_0    pypi
numpy-base        1.18.5          py36hde5b4d6_0
..
```

> “At Criteo we use & deploy our Data Science libraries with Python standard tools (wheels, pip, virtual envs) without using the Anaconda distribution.”

Using Python virtual envs



```
export
PYSPARK_PYTHON=venv/bin/python
spark-submit main.py
```

```
python3 -m venv venv
source venv/bin/activate
pip install requirements.txt
```

```
unzip venv
$PYSPARK_PYTHON -m pyspark.worker
```

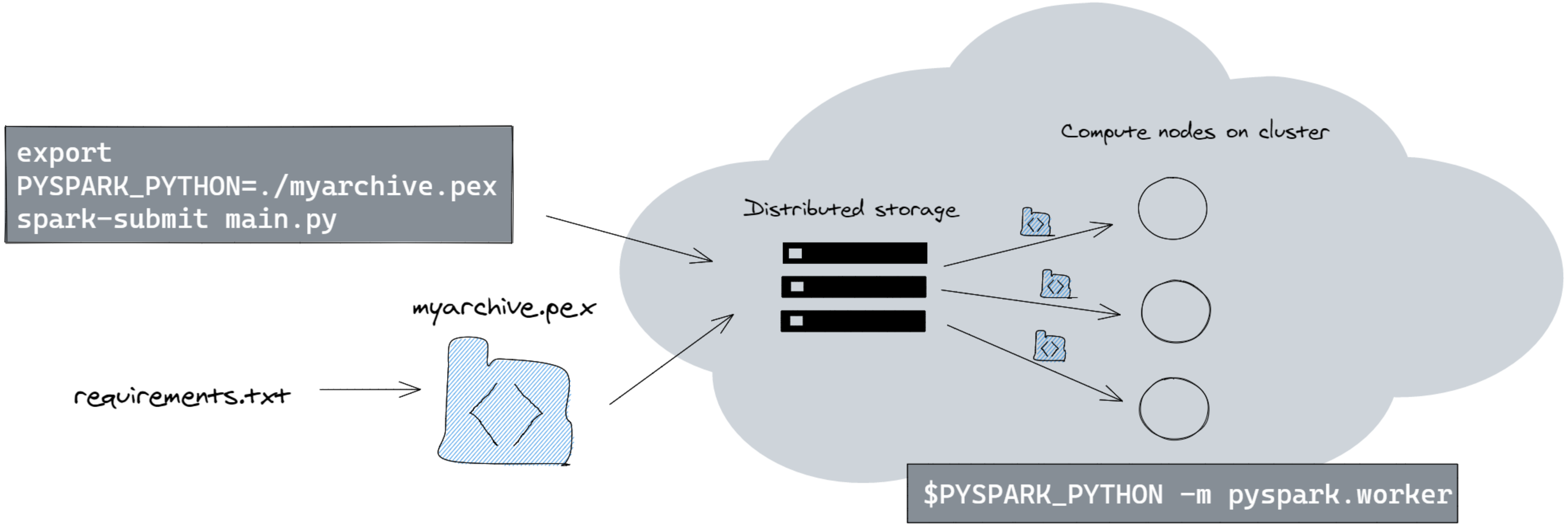
What is PEX ?

A library and tool for generating .pex (Python EXecutable) files

a self executable zip file specified in of PEP-441

```
#!/usr/bin/env python3
#  Python application packed with pex
(binary contents of archive)
```

Using PEX



```
export
PYSPARK_PYTHON=./myarchive.pex
spark-submit main.py
```

```
$PYSPARK_PYTHON -m pyspark.worker
```

Creating the PEX package

```
(pex_env) [f.horing]$ pex pandas pyarrow==0.14.1
pyspark==2.4.4 -o myarchive.pex
(pex_env) [f.horing]$ deactivate
[f.horing]$ ./myarchive.pex
Python 3.6.6 (default, Jan 26 2019, 16:53:05)
(InteractiveConsole)
>>> import pyarrow
>>>
```

How to launch the pex on the Spark executors ?

```
$ export PYSPARK_PYTHON=./myarchive.pex
$ pyspark \
--master yarn --deploy-mode client \
--files myarchive.pex
>>> ..
>>> df.groupby("id").agg(
       mean_udf(df['v'])).toPandas()
```

From spark-submit to Session.builder

```
def spark_session_builder(archive):
  os.environ['PYSPARK_PYTHON'] = \
       './' + archive.split('/')[-1]
  builder = SparkSession.builder
        .master("yarn") \
        .config("spark.yarn.dist.files",
                f"{archive}")
  return builder.getOrCreate()
```

Repackaging Spark code into a function

```
import pandas as pd

def mean_fn(v: pd.Series) -> float:
    return v.mean()

def group_by_id_mean(df):
    mean_udf = pandas_udf(mean_fn, ..)
    return df.groupby("id").agg(
       mean_udf(df['v'])).toPandas())
```

Python api to build & upload pex

```
def upload_env(path):
  # create pex and upload
  return archive
```

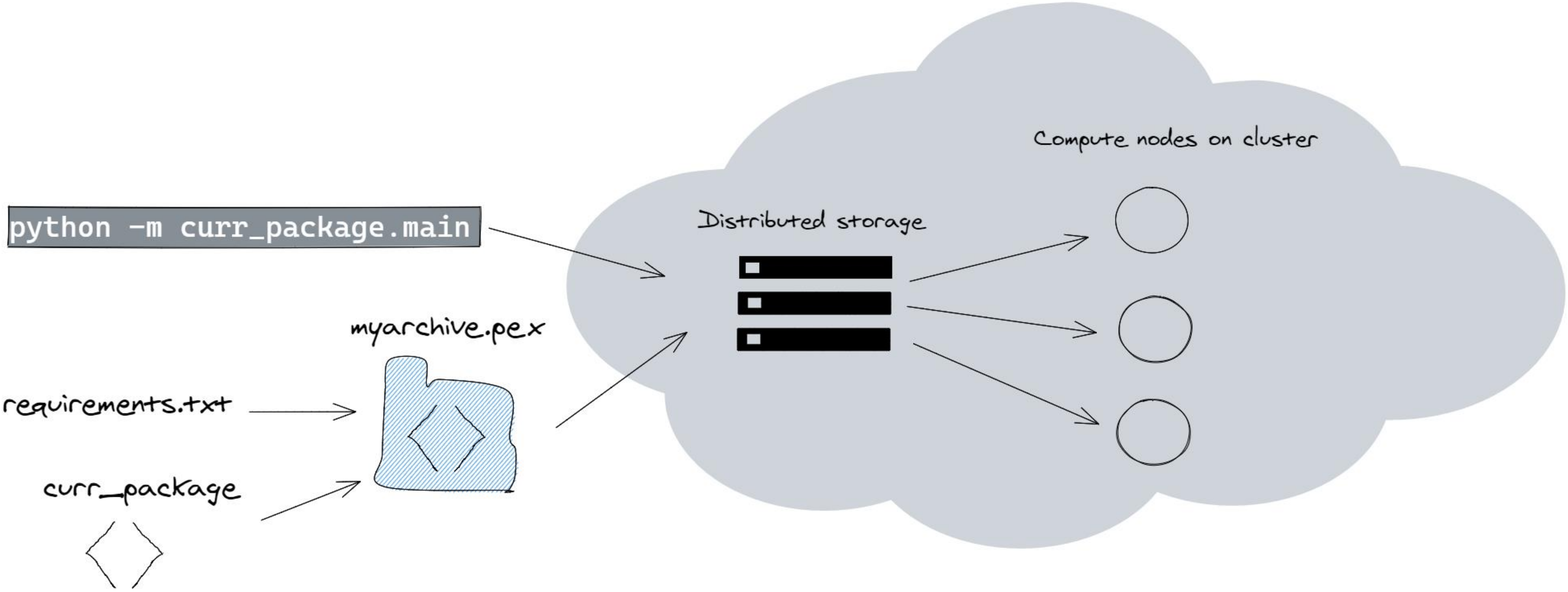
Putting everything to curr_package.main.py

```
archive = upload_env()
spark = spark_session_builder(archive)
df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), ..],
    ("id", "v"))
group_by_id_mean(df)
```

Running main

```
(venv) [f.horing]$ cd curr_package
(venv) [f.horing]$ pip install .
(venv) [f.horing]$ python –m curr_package.main
..
```

Using curr_package.main

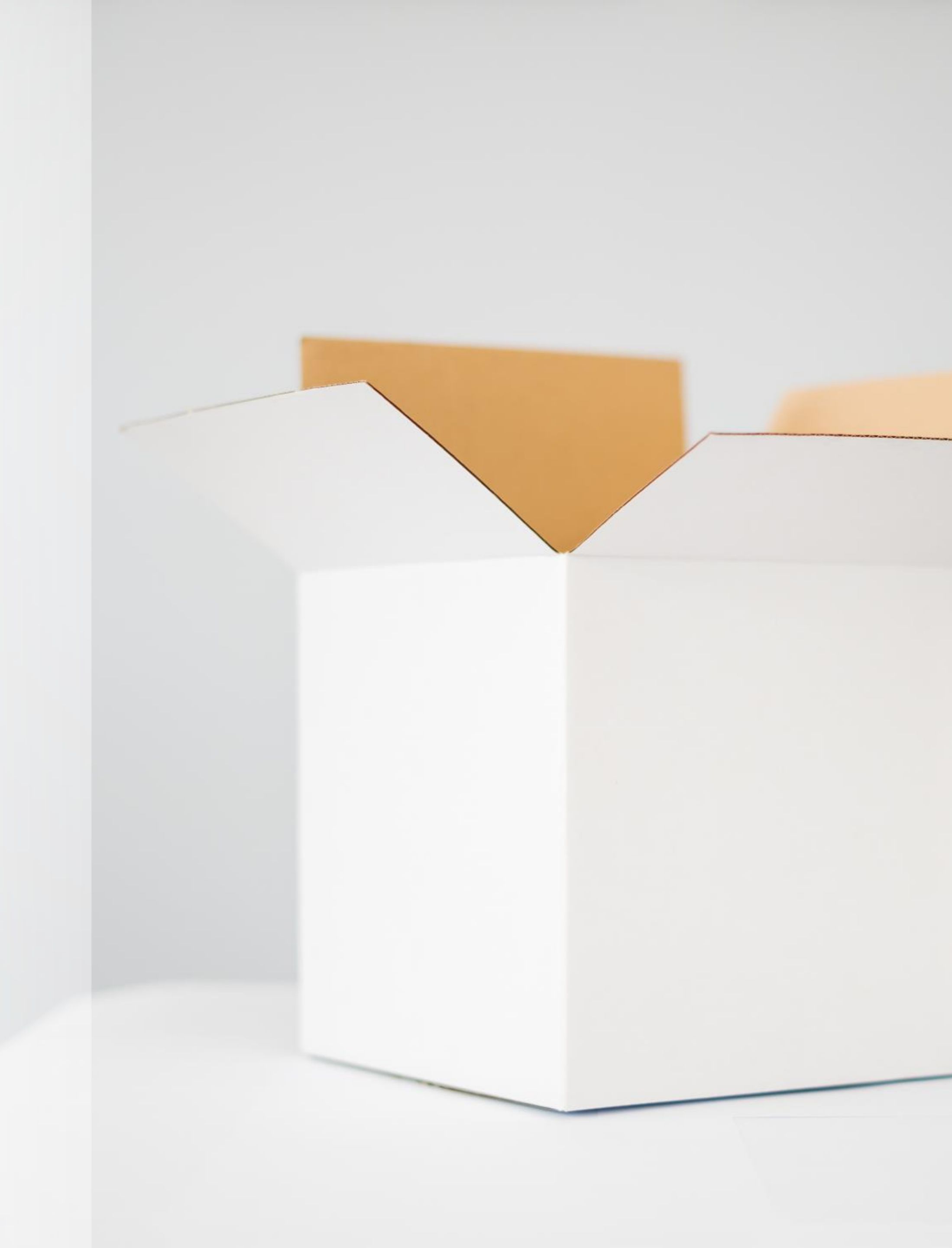


```
python -m curr_package.main
```

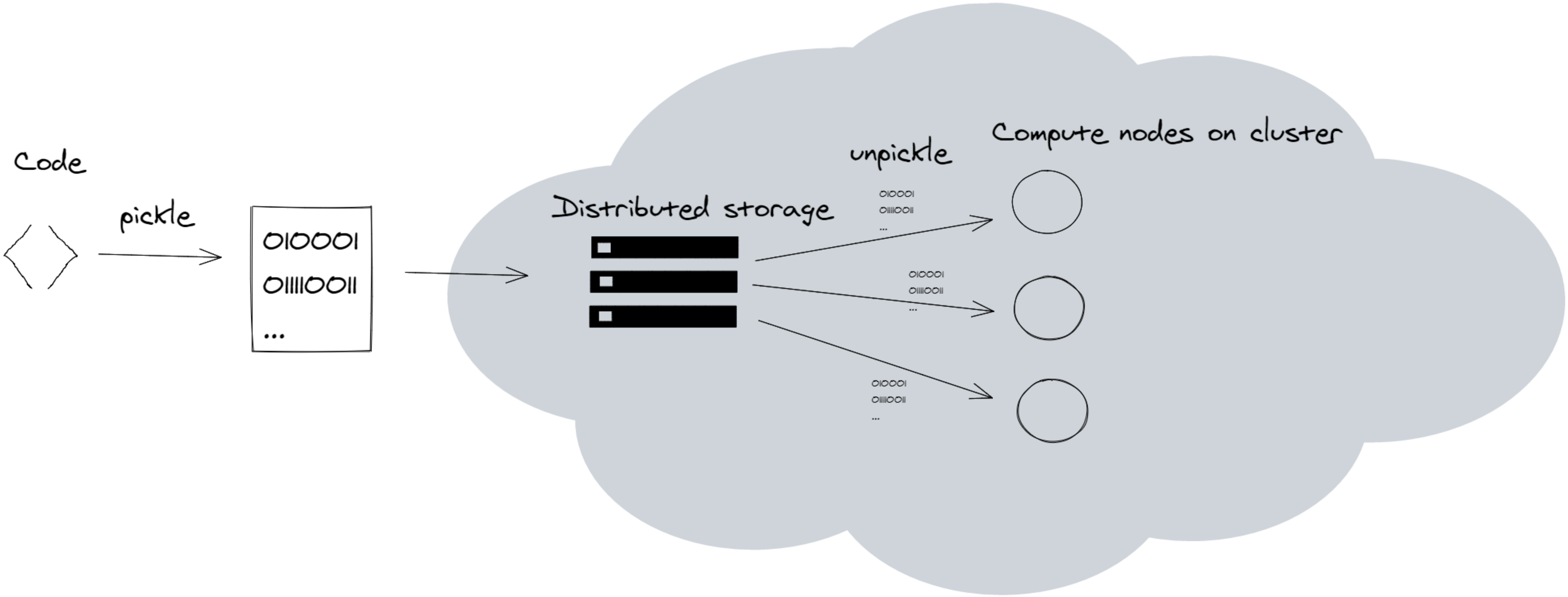
Creating the full package all the time is **reproducable** but **slow**

```
(pex_env) [f.horing]$ time pex curr_package
pandas pyarrow pyspark==2.4.4 -o
myarchive.pex

real    1m4.217s
user    0m43.329s
sys     0m6.997s
```

Separating code under development and dependencies



Pickling with cloudpickle

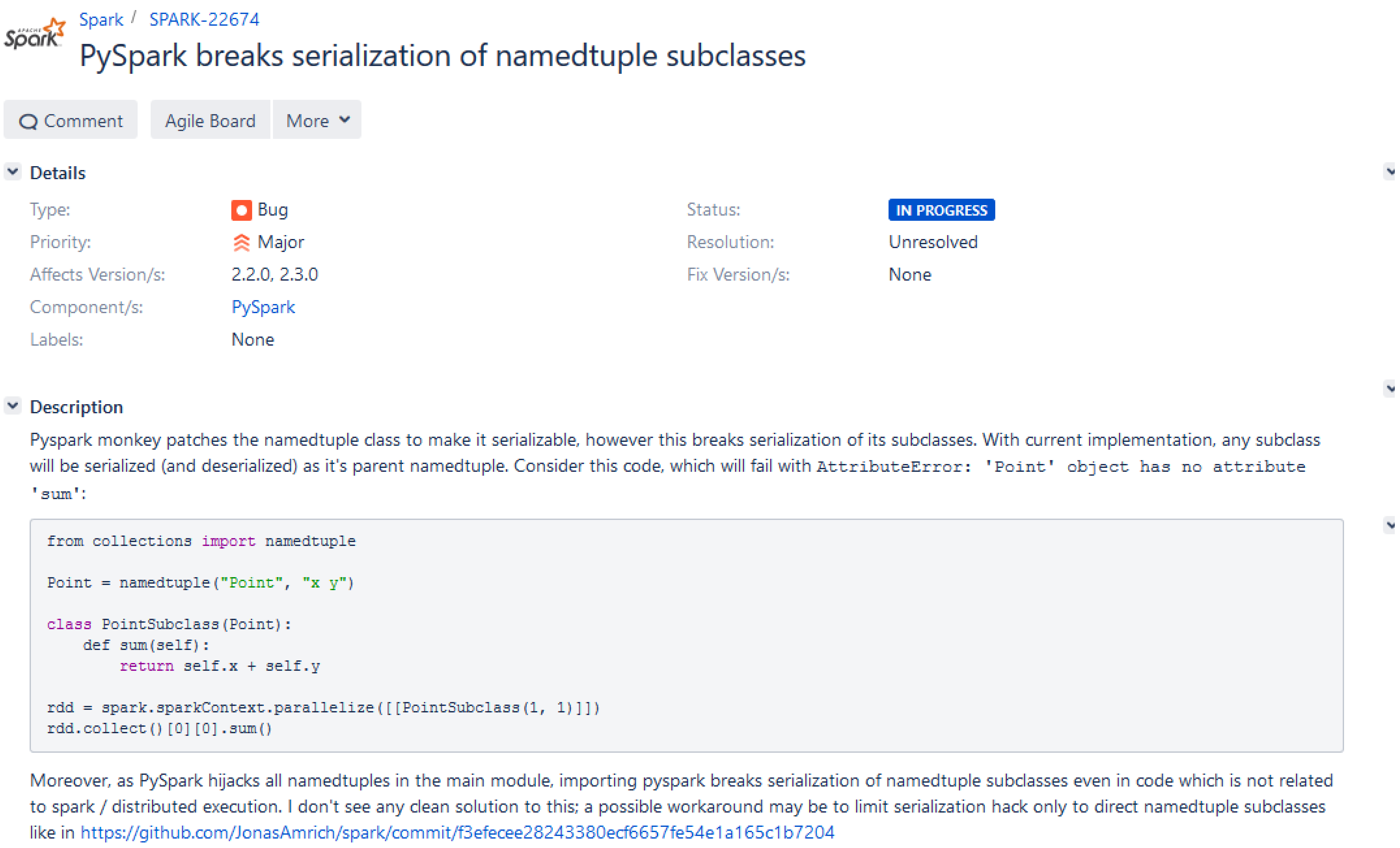


This is how PySpark ships the functions

```
def mean_fn(v: pd.Series) -> float:
    return v.mean()

mean_udf = pandas_udf(mean_fn, ..)
df.groupby("id").agg(
    mean_udf(df['v'])).toPandas()
```

Factorized code won’t be pickled

```
from my_package import main

df.groupby("id").agg(
    main.mean_udf(df['v'])).toPandas()
```



Spark / SPARK-22674

PySpark breaks serialization of namedtuple subclasses

Details

Type:	Bug	Status:	IN PROGRESS
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	2.2.0, 2.3.0	Fix Version/s:	None
Component/s:	PySpark		
Labels:	None		

Description

Pyspark monkey patches the namedtuple class to make it serializable, however this breaks serialization of its subclasses. With current implementation, any subclass will be serialized (and deserialized) as it's parent namedtuple. Consider this code, which will fail with `AttributeError: 'Point' object has no attribute 'sum'`:

```
from collections import namedtuple

Point = namedtuple("Point", "x y")

class PointSubclass(Point):
    def sum(self):
        return self.x + self.y

rdd = spark.sparkContext.parallelize([[PointSubclass(1, 1)]])
rdd.collect()[0][0].sum()
```

Moreover, as PySpark hijacks all namedtuples in the main module, importing pyspark breaks serialization of namedtuple subclasses even in code which is not related to spark / distributed execution. I don't see any clean solution to this; a possible workaround may be to limit serialization hack only to direct namedtuple subclasses like in https://github.com/JonasAmrich/spark/commit/f3efecee28243380ecf6657fe54e1a165c1b7204

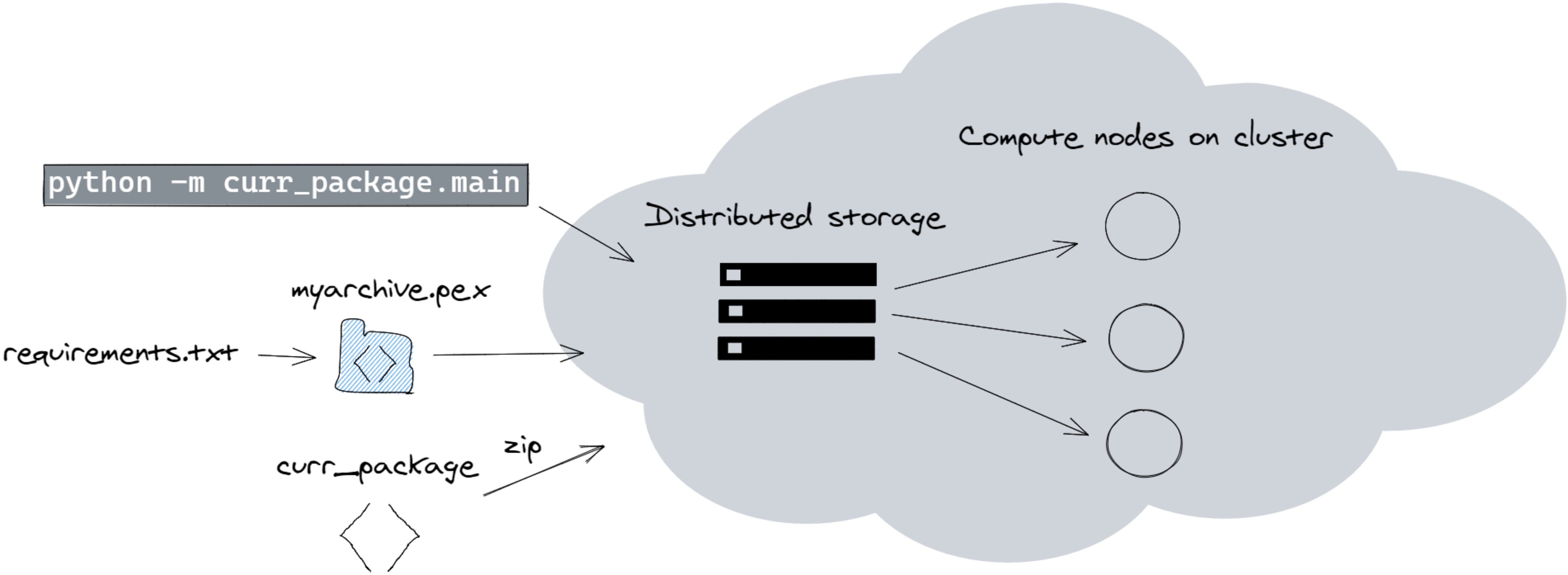
Uploading the current package as zip file

```
def spark_session_builder(archive):

  # upload all but curr_package
  archive = upload_env()
  spark = spark_session_builder(archive)
  spark.sparkContext.addPyFile(
    zip_path("./curr_package"))
  return spark
```

Pip editable mode

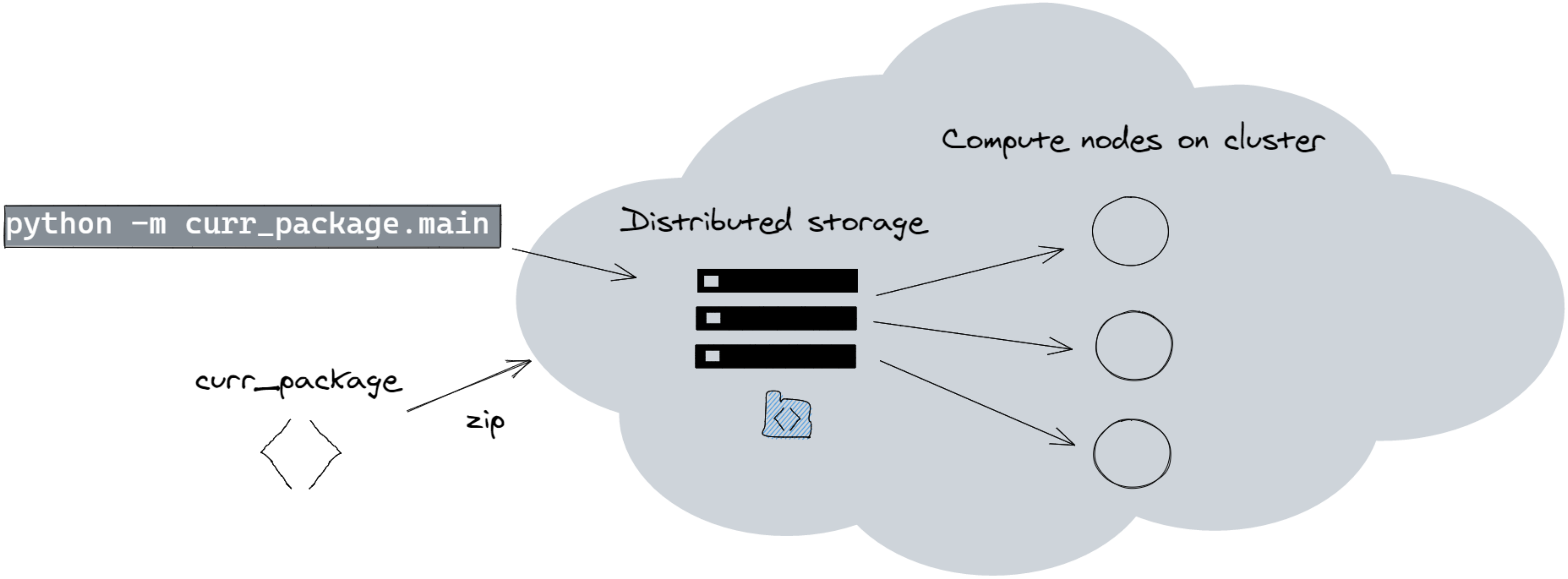
```
(venv) [f.horing]$ pip –e curr_package
(venv) [f.horing]$ pip list
Package         Version         Location
curr_package    0.0.1           /home/f.horing/curr_package
pandas          1.0.0
..
```

Uploading the current package



```
python -m curr_package.main
```

Caching the dependencies on distributed storage



```
python -m curr_package.main
```

How to upload to S3 storage ?

```
>>> s3 = S3FileSystem(anon=False)
>>> with s3.open(
         "s3://mybucket/myarchive.pex",
         "wb") as dest:
...  with open("myarchive.pex", "rb") as source
...    while True:
         out = source.read(chunk)
            if len(out) == 0:
                break
            target.write(out)
```

Listing the uploaded files on S3

```
>>> s3 = S3FileSystem(anon=False)
>>> s3.ls("s3://my-bucket/")
['myarchive.txt']
```

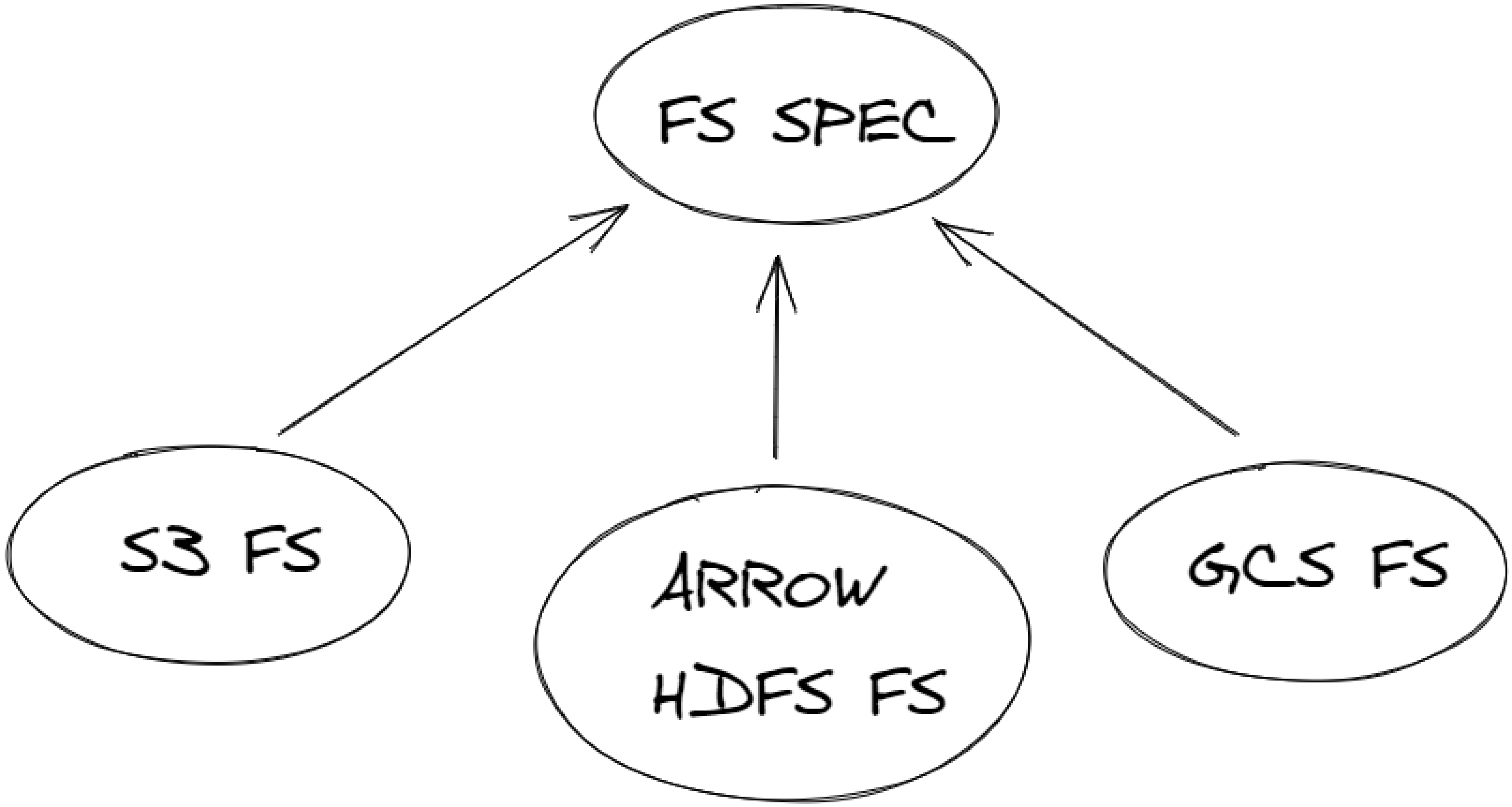
How to connect Spark to S3 ?

```
def add_s3_params(builder):
  builder.config(
       "spark.hadoop.fs.s3a.impl",
       "org.apache.hadoop.fs.s3a.S3AFileSystem")
  builder.config(
       "spark.hadoop.fs.s3a.path.style.access",
       "true")
```

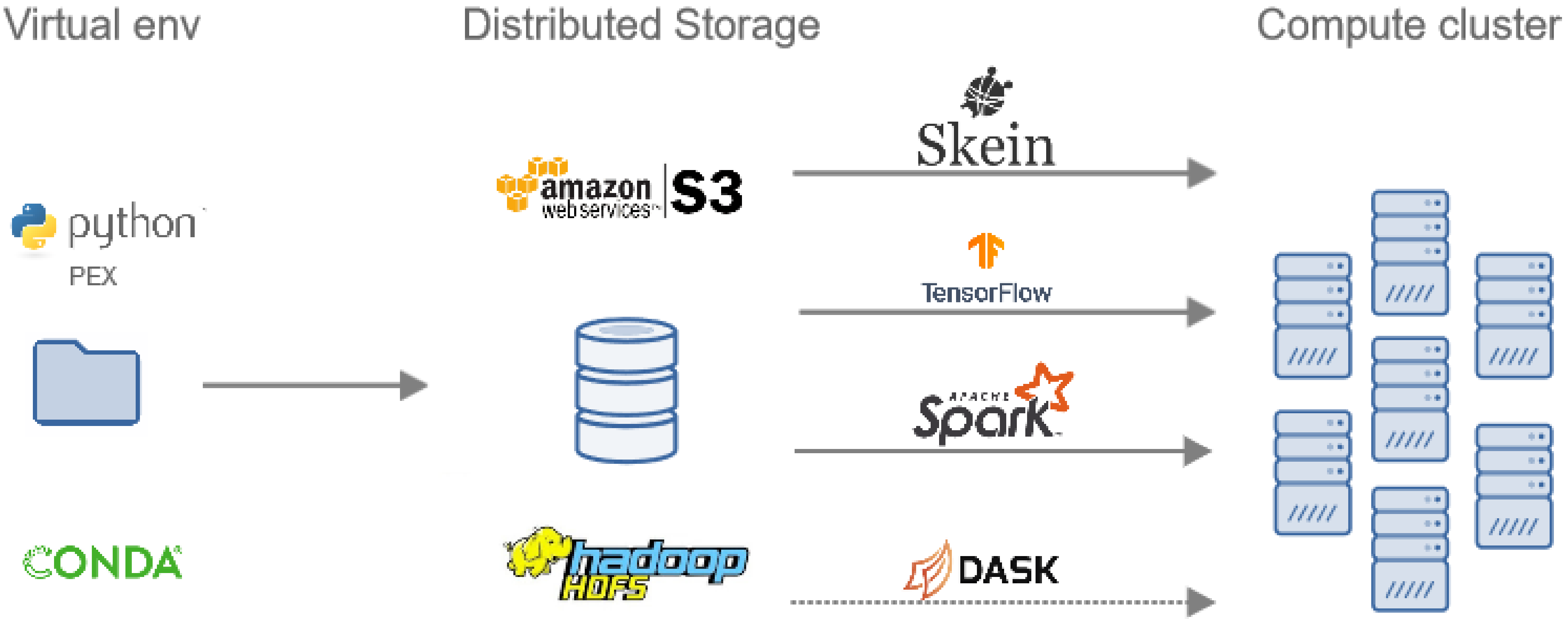
Uploading the zipped current code

```
archive = upload_env(
   "s3://mybucket/myarchive.pex")
builder = spark_session_builder(archive)
add_s3_params(builder)
spark = builder.getOrCreate()
…
group_by_id_mean(df)
```

Using Filesystem Spec
a generic FS interface in Python



cluster-pack



The same example with cluster-pack

```
import cluster_pack
archive = cluster_pack.upload_env(
  package_path="s3://test/envs/myenv.pex")
```

```
from pyspark.sql import SparkSession
from cluster_pack.spark \
  import spark_config_builder as scb

builder = SparkSession.builder
scb.add_s3_params(
  builder,
  s3_args)
```

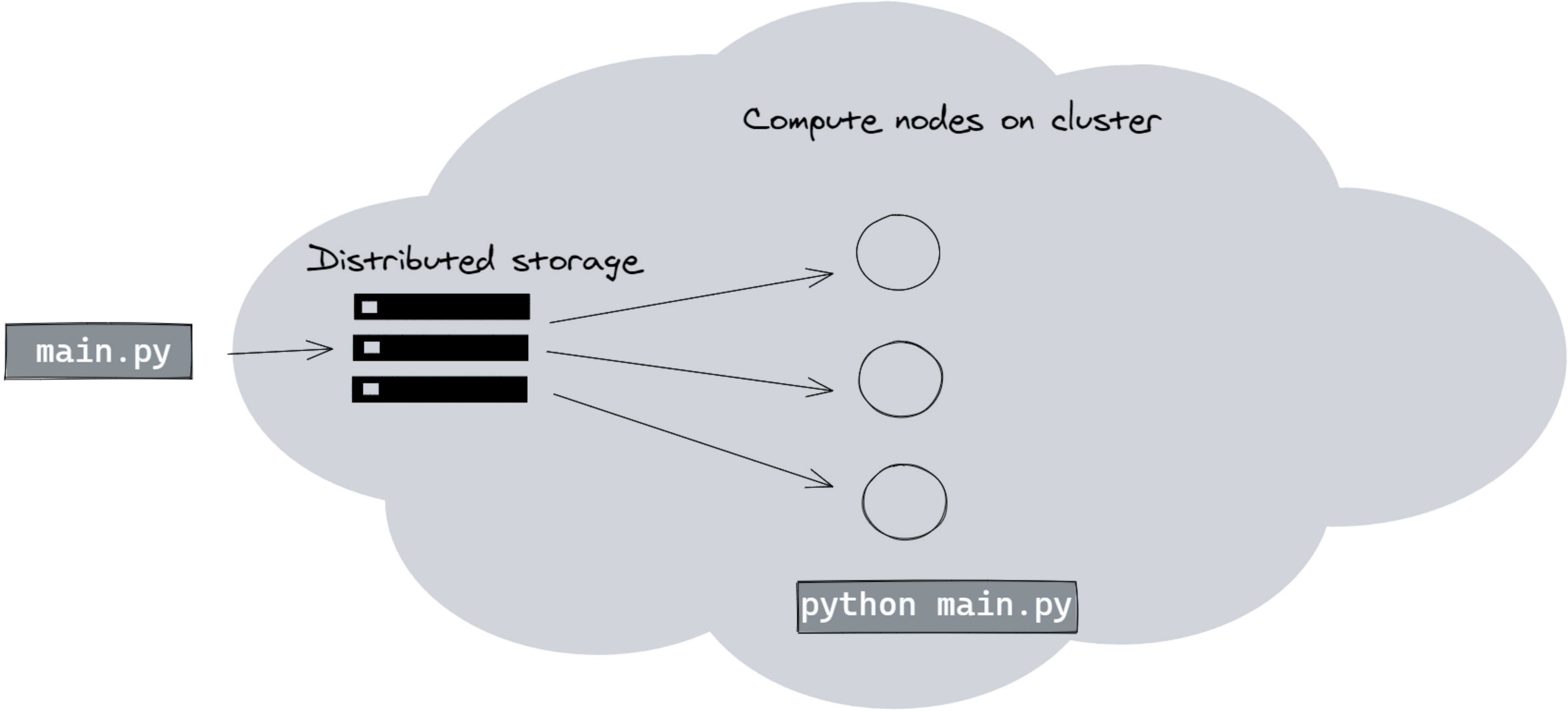
```
scb.add_packaged_environment(
  builder, archive)
scb.add_editable_requirements(
  builder)
spark = builder.getOrCreate()
```

```
df = spark.createDataFrame(
    [(1, 1.0), (1, 2.0), (2, 3.0), ..],
    ("id", "v"))

def mean_fn(v: pd.Series) -> float:
    return v.mean()

mean_udf = pandas_udf(mean_fn, ..)
df.groupby("id").agg(mean_udf(df['v'])).toPandas()
```

What about conda ?

```
import cluster_pack
cluster_pack.upload_env(
   package_path="s3://test/envs/myenv.pex",
   packer = packaging.CONDA_PACKER
)
```

Running TensorFlow jobs



Links & Credits

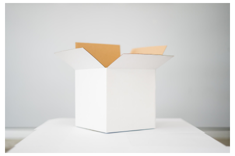


Photo by Kelli McClintock on Unsplash

https://github.com/criteo/cluster-pack/blob/master/examples/spark-with-S3/README.md
https://spark.apache.org/docs/2.4.4/sql-pyspark-pandas-with-arrow.html#grouped-aggregate
https://medium.com/criteo-labs/packaging-code-with-pex-a-pyspark-example-9057f9f144f3
https://github.com/criteo/cluster-pack
https://github.com/dask/s3fs
https://github.com/intake/filesystem_spec