Notebooks and Computing Workflows

Ricardo Rocha - IT-CM-RPS

ABP Computing Forum
https://indico.cern.ch/event/976158/

About

Computing Engineer in CERN IT - Cloud Team

Containers, Docker, Kubernetes, Networking

Machine Learning Infrastructure

Cloud Native Computing Foundation (CNCF) TOC

Reach out for any requests on Kubernetes, Prometheus, Helm, Fluentd, …

ricardo.rocha@cern.ch

Today

Workflows

Docker Containers and Reproducibility

Container Orchestration

Notebooks

Pipelines and Distributed Workflows

Some Demos!

Not covered today… but do check them out!

SWAN: https://swan.cern.ch

REANA: https://reana.cern.ch/

Batch / HTCondor: https://batchdocs.web.cern.ch/

Spark

Workflows

It often starts like this... compile, run, analyse, ...

Hard to keep a clean environment

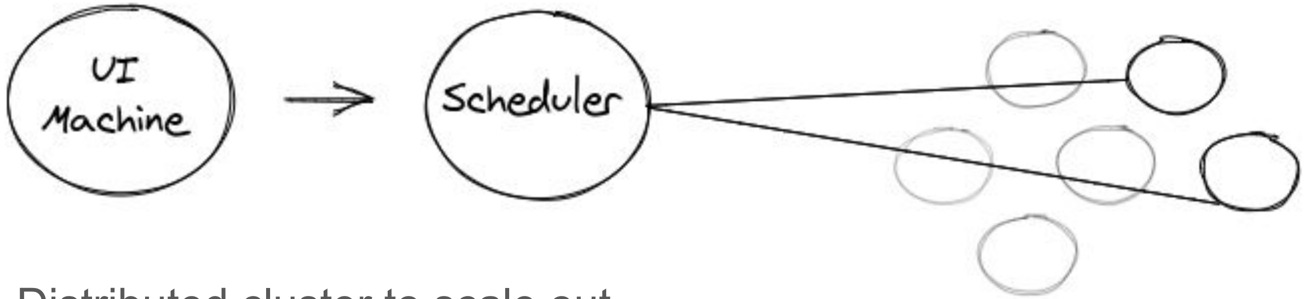
Often hard to find packages for dependencies

  Unless everyone agrees on the same system / distribution

Limited resources, limited scalability

Hard to ensure reproducibility, sharing with colleagues

  “It worked in my machine…”



Distributed cluster to scale out

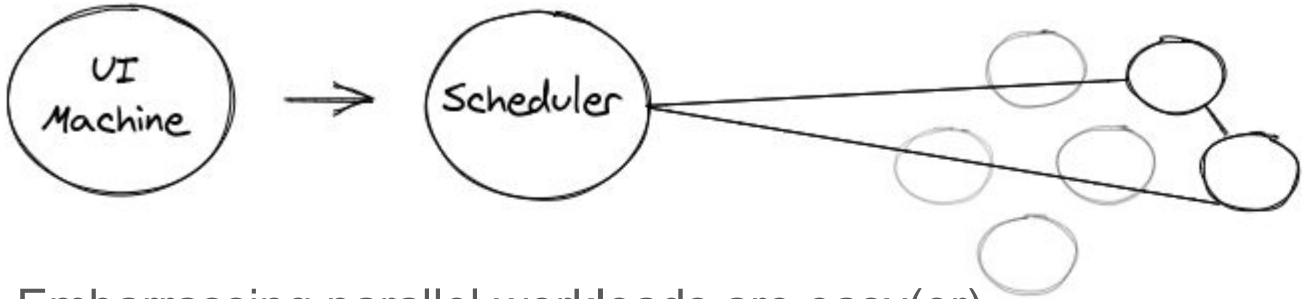
- Typical mode for batch systems: HTCondor, SLURM, …

Worker nodes run one (or a few) sets of systems / distributions

- Not trivial to comply with everyone’s requirements

How to push specific job dependencies

- Shared file-system is the most common solution, but manage / update has issues



Embarrassing parallel workloads are easy(er)

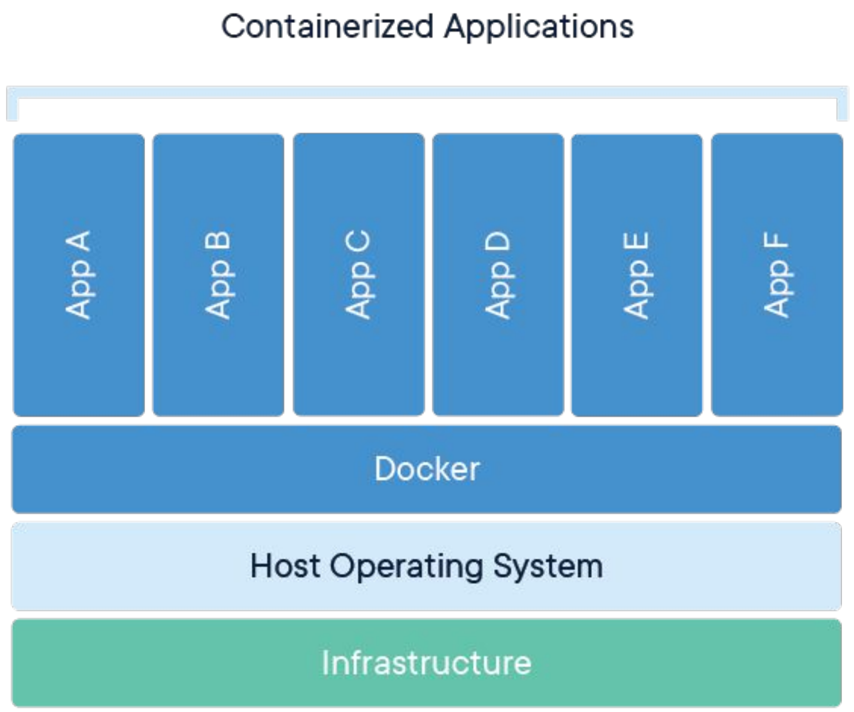
Things get trickier when jobs need to talk to each other

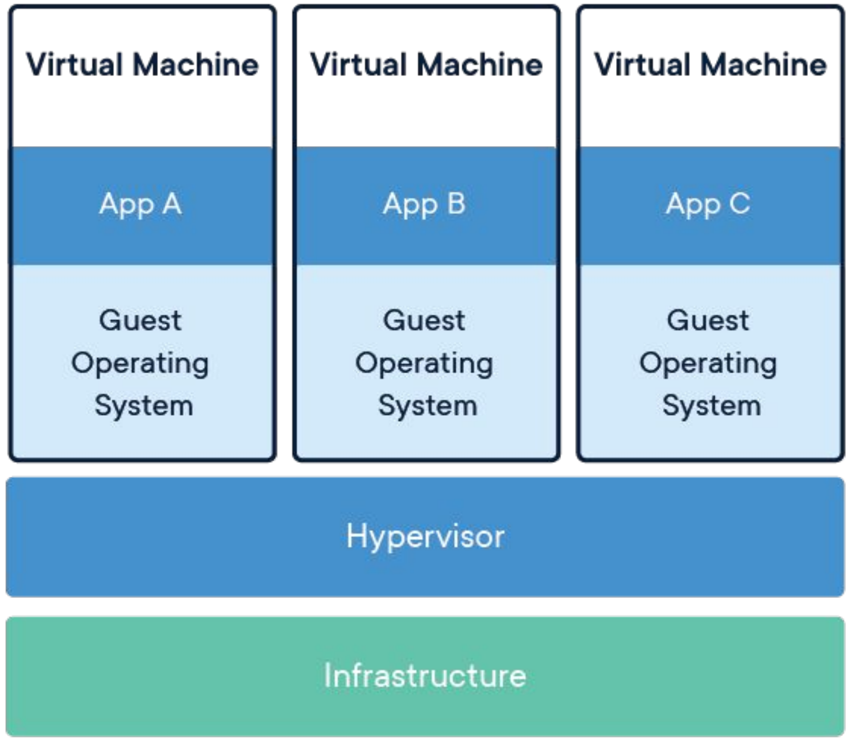
MPI, DAGs, gang scheduling, …

Systems have their own ways to express these dependencies

Docker Containers and Reproducibility

Basic Unit: Docker Container





Basic Unit: Docker Container

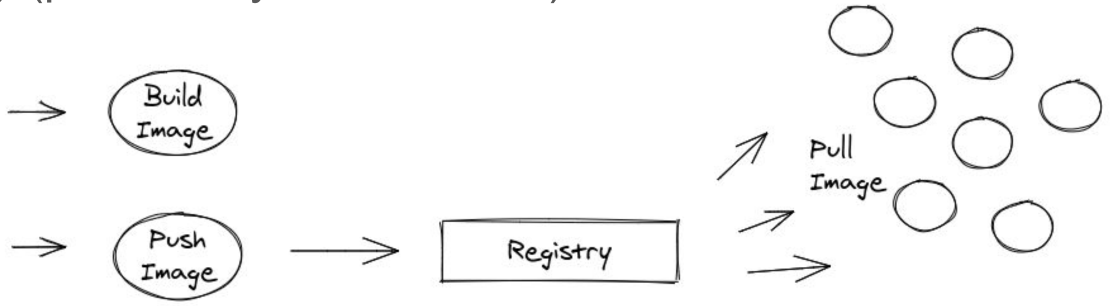
Builds are done locally, or using a CI process, or…

Resulting images are uniquely identified by a sha (immutable) and/or tag

This ensures reproducibility

An image *Push* makes it available to others in a registry

Pulled images are cached locally (pulled only the first time)



Quick Demo

Container Orchestration with Kubernetes

Manage containerized deployments in multi-node clusters

- Support for heterogeneous setups by grouping nodes

Many low level concepts covering compute, networking and storage

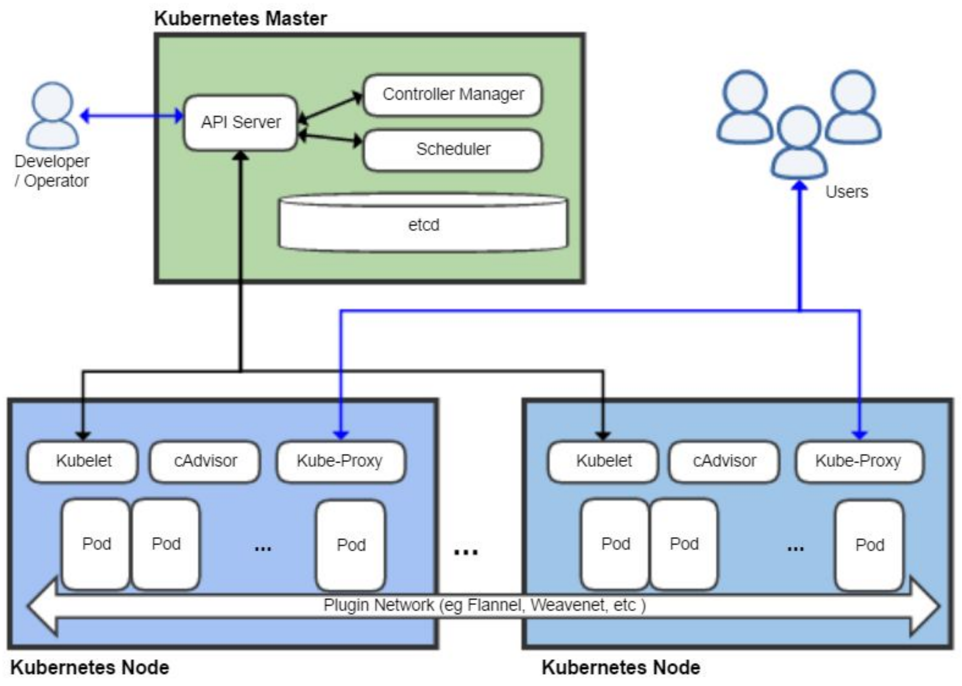
- We need to tell what type of resources are required for a container to run

Ways to express replication, failure domains, disruption budgets, etc

Ways to express policies for auto scaling of workloads

https://clouddocs.web.cern.ch/containers/README.html

Container Orchestration with Kubernetes



Kubernetes

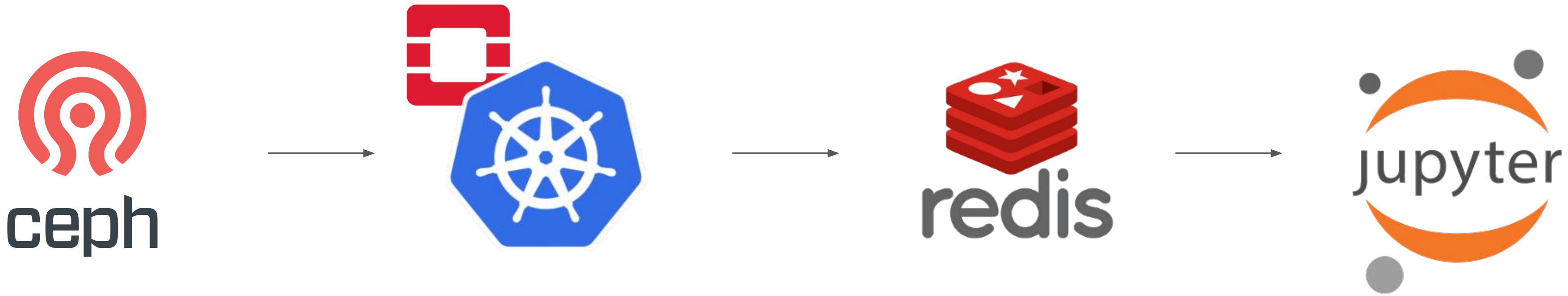
Lingua franca of the cloud

Managed services offered by all major public clouds

Multiple options for on-premise or self-managed deployments

Common declarative API for basic infrastructure : compute, storage, networking

Healthy ecosystem of tools offering extended functionality



70 TB Dataset

OpenStack Magnum

25000 **Kubernetes Jobs**

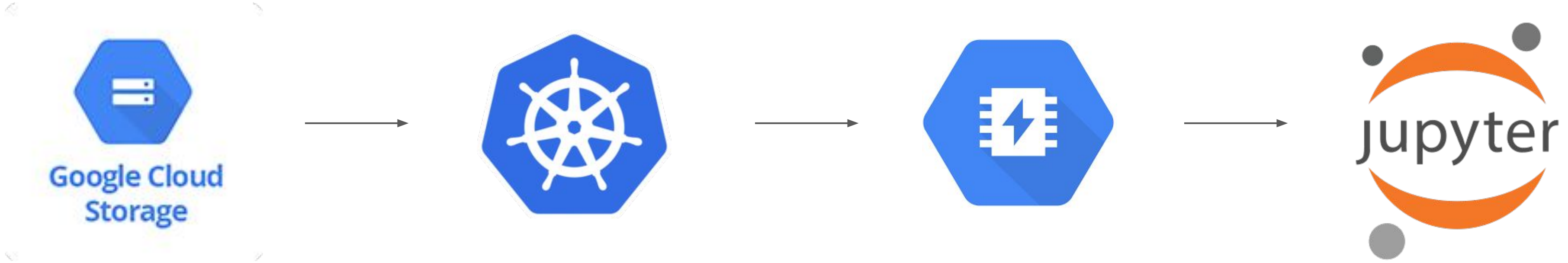
Job Results

Interactive Visualization

Aggregation

CERN openlab

Google Cloud



70 TB Dataset

Cluster on GKE

Max **25000 Cores**

Single Region, 3 Zones

25000 **Kubernetes Jobs**

Job Results

Interactive Visualization

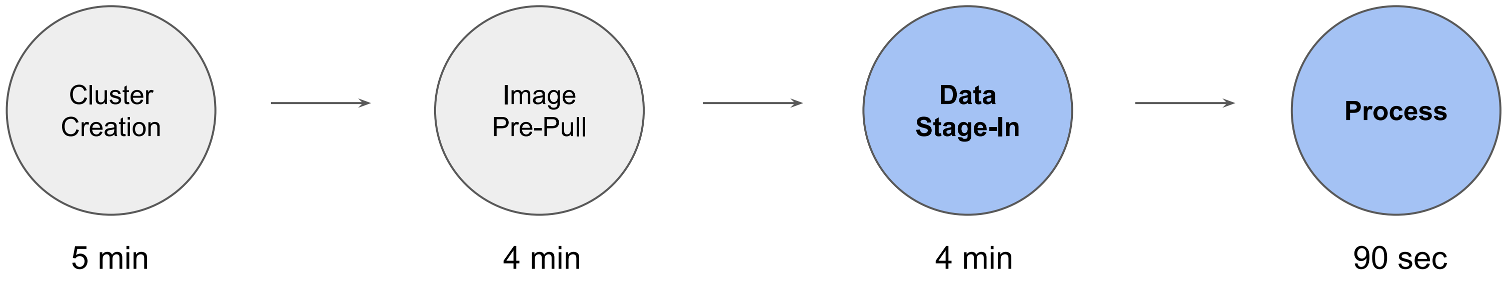
Aggregation

GCP Analysis Run

Kubernetes clusters on GKE (Managed Kubernetes service on GCP)

Run included

- ~1200 nodes: n1-highmem-16, 104 GB RAM
- ~20000 cores, ~120 TB RAM



Quick Demo

Jupyter Notebooks

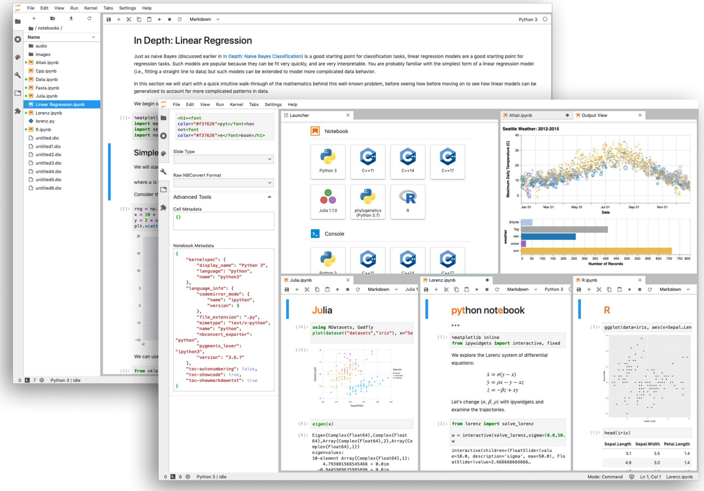
Easy way to create and share documents with code, equations, …

Can be run locally, but popular to run in a central service: JupyterHub

Started with python, but kernel support for R, Go, C++, etc

Extremely popular for data science

And machine learning...



JupyterHub

Central web application managing multi-user environments

Popular to rely on Docker containers for isolation

Common Issue: how to manage dependencies not in an image

- *pip install --user mypkg* has issues when sharing, reproducibility
- Someone needs to rebuild the image, push
- Or you just rely on an alternative to get extra dependencies (shared fs?)
- Some automation is required, we'll see another option later

Dask

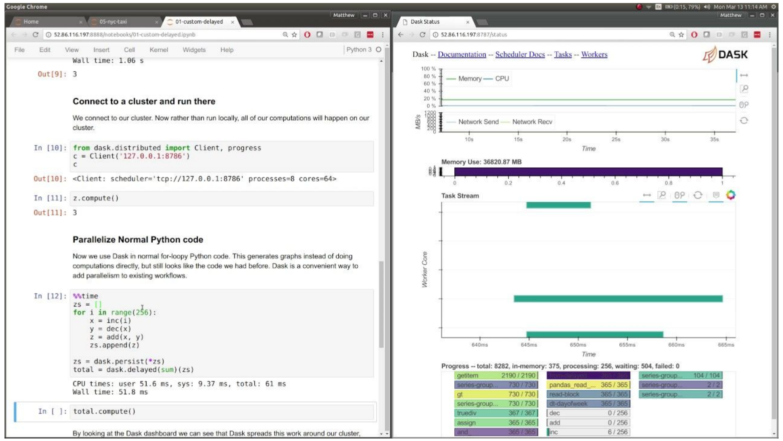
Advanced parallelism for analytics

Integrates well with Pandas, numpy, scikit-learn

Familiar if you’re used to Python

Extreme scalability: thousands of nodes, tens of thousands of workers

Backends for Kubernetes, SLURM, HTCondor, among others


Google Chrome
Home
05-nyc-taxi
01-custom-delayed
52.86.116.197:8888/notebooks/01-custom-delayed.ipynb
File Edit View Insert Cell Kernel Widgets Help
Python 3
Dask Status
52.86.116.197:8787/status
Dask -- Documentation -- Scheduler Docs -- Tasks -- Workers
DASK
Memory
CPU
Network Send
Network Recv
MB/s
Time
Memory Use: 36820.87 MB
Task Stream
Worker Core
Progress -- total: 8282, in-memory: 375, processing: 256, waiting: 504, failed: 0
getitem 2190 / 2190
series-group... 730 / 730
gt 730 / 730
series-group... 730 / 730
truediv 367 / 367
assign 365 / 365
and_ 365 / 365
pandas_read_... 365 / 365
read-block 365 / 365
dt-dayofweek 365 / 365
dec 0 / 256
add 0 / 256
inc 6 / 256
series-group... 104 / 104
series-group... 2 / 2
series-group... 2 / 2

```
    Wall time: 1.06 s

Out[9]: 3
```

Connect to a cluster and run there

We connect to our cluster. Now rather than run locally, all of our computations will happen on our cluster.

```
In [10]: from dask.distributed import Client, progress
         c = Client('127.0.0.1:8786')
         c

Out[10]: <Client: scheduler='tcp://127.0.0.1:8786' processes=8 cores=64>

In [11]: z.compute()

Out[11]: 3
```

Parallelize Normal Python code

Now we use Dask in normal for-loopy Python code. This generates graphs instead of doing computations directly, but still looks like the code we had before. Dask is a convenient way to add parallelism to existing workflows.

```
In [12]: %%time
         zs = []
         for i in range(256):
             x = inc(i)
             y = dec(x)
             z = add(x, y)
             zs.append(z)

         zs = dask.persist(*zs)
         total = dask.delayed(sum)(zs)

         CPU times: user 51.6 ms, sys: 9.37 ms, total: 61 ms
         Wall time: 51.8 ms

In [ ]: total.compute()
```

By looking at the Dask dashboard we can see that Dask spreads this work around our cluster,

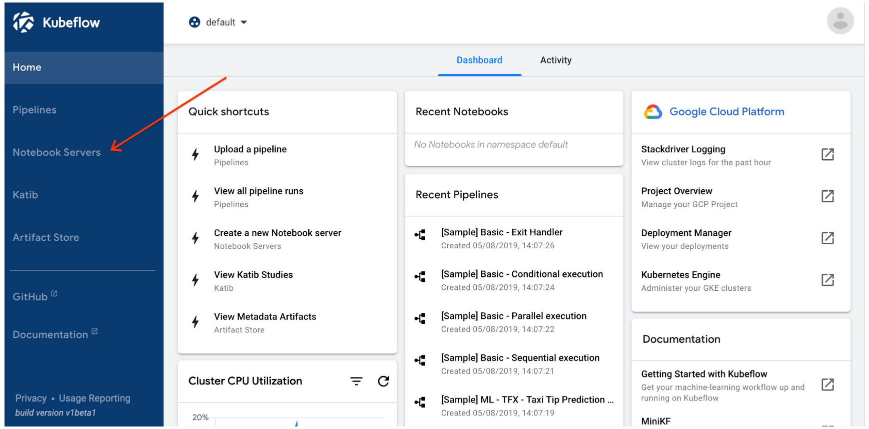
Quick Demo

Kubeflow + Kale

Kubeflow is a platform to do machine learning / data science on Kubernetes

Kale is a Jupyter extension to ease moving from notebooks to pipelines

Early access almost ready for a CERN instance



Quick shortcuts

- Upload a pipeline — Pipelines
- View all pipeline runs — Pipelines
- Create a new Notebook server — Notebook Servers
- View Katib Studies — Katib
- View Metadata Artifacts — Artifact Store

Cluster CPU Utilization

20%

Recent Notebooks

No Notebooks in namespace default

Recent Pipelines

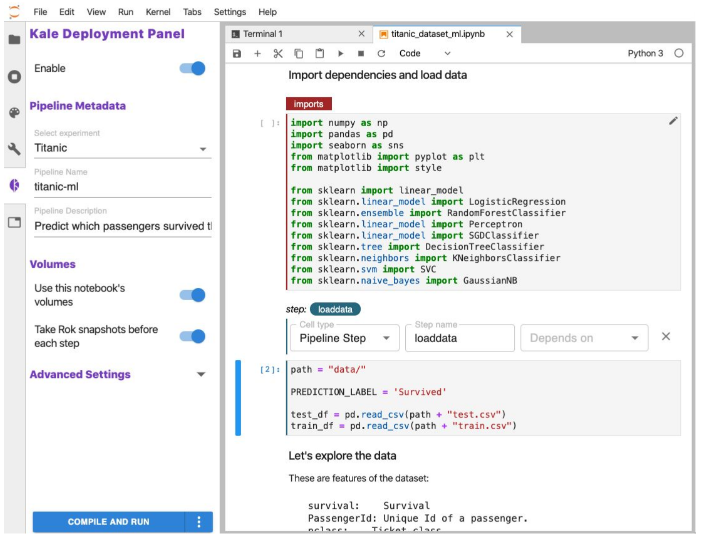
- [Sample] Basic - Exit Handler — Created 05/08/2019, 14:07:26
- [Sample] Basic - Conditional execution — Created 05/08/2019, 14:07:24
- [Sample] Basic - Parallel execution — Created 05/08/2019, 14:07:22
- [Sample] Basic - Sequential execution — Created 05/08/2019, 14:07:21
- [Sample] ML - TFX - Taxi Tip Prediction ... — Created 05/08/2019, 14:07:19

Google Cloud Platform

- Stackdriver Logging — View cluster logs for the past hour
- Project Overview — Manage your GCP Project
- Deployment Manager — View your deployments
- Kubernetes Engine — Administer your GKE clusters

Documentation

- Getting Started with Kubeflow — Get your machine-learning workflow up and running on Kubeflow
- MiniKF



Kale Deployment Panel

Enable

Pipeline Metadata

Select experiment: Titanic

Pipeline Name: titanic-ml

Pipeline Description: Predict which passengers survived t

Volumes

Use this notebook's volumes

Take Rok snapshots before each step

Advanced Settings

Import dependencies and load data

imports

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from matplotlib import style

from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
```

step: loaddata

Cell type: Pipeline Step — Step name: loaddata — Depends on

```
path = "data/"

PREDICTION_LABEL = 'Survived'

test_df = pd.read_csv(path + "test.csv")
train_df = pd.read_csv(path + "train.csv")
```

Let's explore the data

These are features of the dataset:

```
survival:    Survival
PassengerId: Unique Id of a passenger.
```

Quick Demo

Summary

Docker helps with sharing computing environments and reproducibility

Kubernetes is taking a big place in orchestrating computing workflows

- And much more in the service area

Lots of options for data analysis, with notebooks being a very popular option

Platforms like Dask and Kubeflow try to help scaling out these workloads

Questions?

https://clouddocs.web.cern.ch/containers/README.html

https://binder.cern.ch

https://ml.docs.cern.ch

https://swan.cern.ch

https://reana.cern.ch/

https://batchdocs.web.cern.ch/