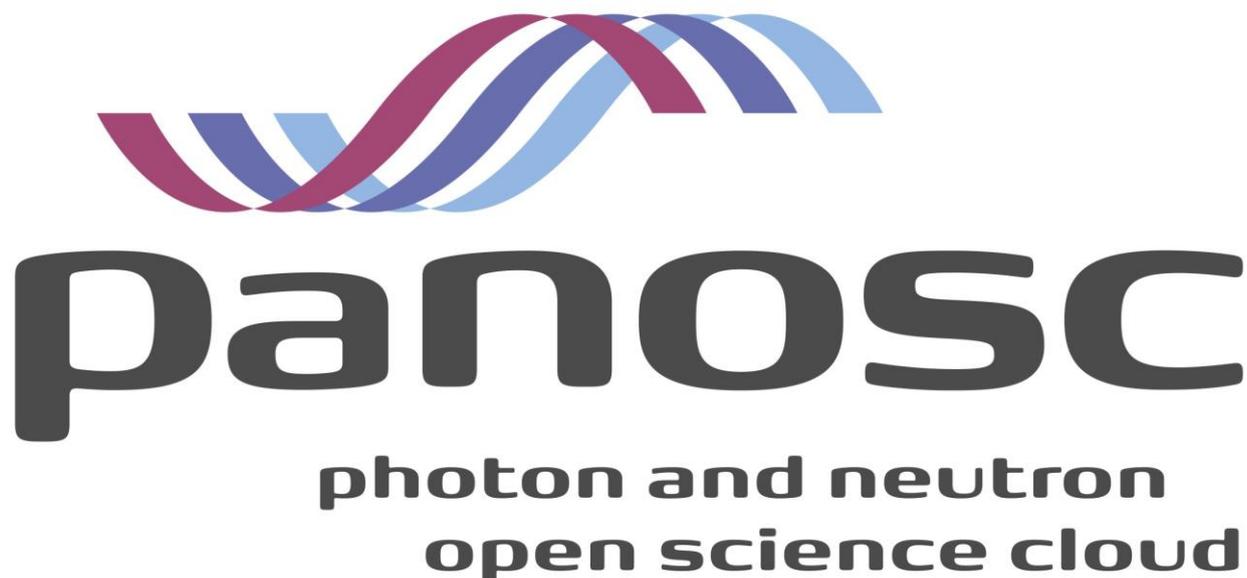


PaNOSC

Photon and Neutron Open Science Cloud

H2020-INFRAEOSC-04-2018

Grant Agreement Number: 823852



Deliverable:

Prototype Remote Desktop and Jupyter Service (4.2)

Project Deliverable Information Sheet

Project Reference No.	823852
Project acronym:	PaNOSC
Project full name:	Photon and Neutron Open Science Cloud
H2020 Call:	INFRAEOSC-04-2018
Project Coordinator	Andy Götz (andy.gotz@esrf.fr)
Coordinating Organization:	ESRF
Project Website:	www.panosc.eu
Deliverable No:	D4.2
Deliverable Type:	Demonstrator
Dissemination Level	Confidential, only for the members for the consortium (including the Commission Services)
Contractual Delivery Date:	Month 18
Actual Delivery Date:	
EC project Officer:	Rene Martins

Document Control Sheet

Document	Title: Prototype Remote Desktop and Jupyter Service
	Version: 1.0
	Available at:
	Files:
Authorship	Written by: GREENWOOD Lottie (ESS), HALL Jamie (ILL), FANGOHR Hans (EuXFEL), KLUYVER Thomas (EuXFEL), ROSCA Robert (EuXFEL)
	Contributors: CAMPBELL Aidan (ESRF), DE SIMONE Marco (CERIC-ERIC), FANGOHR Hans (EuXFEL), GALAL Kareem (ESS), GREENWOOD Lottie (ESS), GROSZ Jakub (ELI), HALL Jamie (ILL), KLUYVER Thomas (EuXFEL), LORENZON Andrea (CERIC-ERIC), ROSCA Robert (EuXFEL), VINCENT Thomas (ESRF)
	Reviewed by: BODERA SEMPERE Jordi (ESRF)
	Approved: GÖTZ Andy (ESRF)

List of participants

Participant No.	Participant organisation name	Country
1	European Synchrotron Radiation Facility (ESRF)	France
2	Institut Laue-Langevin (ILL)	France
3	European XFEL (XFEL.EU)	Germany
4	The European Spallation Source (ESS)	Sweden
5	Extreme Light Infrastructure Delivery Consortium (ELI-DC)	Belgium
6	Central European Research Infrastructure Consortium (CERIC-ERIC)	Italy
7	EGI Foundation (EGI.eu)	The Netherlands

Table of Content

Project Deliverable Information Sheet	2
Document Control Sheet	2
List of participants	2
Table of Content	3
1.0 Introduction	4
2.0 Context	5
2.1 Motivation	5
2.2 Baseline: SSH Access for Cluster	6
3.0 Remote Jupyter Service	9
3.1 Jupyter Notebook	9
3.2 JupyterHub	10
3.3 Jupyter Service Prototype Status	10
3.4 Jupyter Service User Experience with Demonstrators	10
3.5 Technical Deployment	15
4.0 Remote Desktop Services	19
4.1 Remote Desktop	19
4.2 Remote Desktop Prototype Service Status	19
4.3 Remote Desktop User Experience with Demonstrators	19
4.3.1 ILL	20
4.3.2 ESS	25
4.3.3 CERIC-ERIC	26
4.3.4 ISIS	27
4.4 Technical Deployment	27
5.0 Towards a Harmonisation of Services	30
6.0 Summary	33

1.0 Introduction

Due to the increase in data produced by the facilities, an increasing amount of computer power and storage space is required to manage and analyse said data. Remote data analysis will enrich the services provided by European Open Science Cloud (EOSC): if users can explore data from their web browser after having identified a data set of interest, this lowers the barriers towards re-use of the data. Remote data analysis further benefits users and facilities by enabling both pooling, and therefore more efficient use of compute resources, and reducing the need for large data to be transferred.

The Work Package 4.2 deliverable is defined as the three proposed services (virtual machines, remote desktop and Jupyter Notebooks) being available as a remote service via the internet. They are required to have been created and deployed at one partner site minimum and serve as prototypes for the data analysis services. This report covers the status of relevant remote data analysis services at all facilities concerned and discusses how they meet the requirements from the use cases proposed by the facilities.

In this document, we first provide the context for which remote data analysis services are useful for researchers working on Photon and Neutron science (2.0 Context), and describe Secure Shell as the de-facto standard for this (2.2 Baseline: SSH access for cluster). We then have a dedicated section on the Remote Jupyter Service prototypes and demonstrator (3.0 Remote Jupyter Service) and another section on the Remote Desktop Service prototypes and demonstrator (4.0 Remote Desktop Services). In each section, we comment on the status of the service provision across the facilities, the user experience and discuss some aspects of the deployment of the services. We close with a short discussion on the status and plans for the harmonization of all these services (5.0 Towards a Harmonisation of Services) and then a brief summary (6.0 Summary).

2.0 Context

2.1 Motivation

Remote data analysis is the ability to carry out numerical processing from a computer or laptop far away from the (High Performance) computing centre that hosts the data and the computer resources. In the context of Photon and Neutron research facilities, research scientists are the users of the facility and carry out experiments either by physically visiting the facility or sending in samples remotely. Remote data analysis is useful for a range of purposes, including:

- Online data analysis – Remote data analysis accessed during the active running of the experiment. In this case the user is expected to already have a certain workflow and software in place, and only make smaller changes during the experiment. This “online analysis” is important to inform the rest of the experiment. An experiment can last several days or a week. While some of the research team conducting the experiment will be on site, they may be supported by further researchers who act as data experts and can analyse the data while the experiment team sleeps, and the data experts can do their work remotely (i.e. without having to travel to the facility).
- Offline data analysis – Remote data analysis that is necessary after the experiment. At this point, the users have typically returned from the research facility to their home institutions. This includes exploratory data analysis, when the user is trying to find the best way to extract meaning from their data. This is less time sensitive than online data analysis although the user may require quick feedback on smaller sizes of data initially, before running larger analyses across all their experiment results.
- ‘Knowledge sharing’/ Tutorials – Requires that data analysis workflows can be shared across users with a wide spectrum of expertise. This could range from software scientists teaching users how to analyse their offline data, or beamline scientists providing a template for easier analysis of online data for users. Remote data analysis allows users to run examples with real data at the facility when they are not there in person.

We list a number of software tools used in Photon and Neutron research in *Table 1* (collected from suggested use cases by facilities) and compare their compatibility with Jupyter Notebooks and requirements for Graphical User Interfaces (GUIs).

Table 1 - Application use cases from facilities

Application	Description	Usable in Jupyter notebook	Graphical User Interface (GUI)
PyMca	X-ray powder diffraction analysis	Yes	Optional
Mantid	Neutron scattering analysis framework	Partly	Required for full functionality
SCI++	Library for handling multi-dimensional data arrays similar to xarray	Yes	None
Sasview	Small angle scattering analysis	Partly	Required
McStas	Simulation for neutron scattering experiments	Partly	Optional
QENSmodels	Models for fitting Quasi Elastic Neutron Scattering data	Yes	None
PyNX	Tools for nano-structures crystallography	Yes	None
crispy	GUI for calculating core level spectra	No	Required
pyFAI	Fast Azimuthal integration using python	Yes	Optional

2.2 Baseline: SSH Access for Cluster

The baseline provision of remote access to compute services is using secure shell (SSH), and all facilities had this in place before this project started. SSH is a well-established character based interface: commands can be entered and text can be displayed and edited. This works well and can be used to access a supercomputer in Germany from a computer in Japan or the US using the text-based SSH protocol (Figure 1).

```

[[greenwood@rin2 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
long      up 1-00:00:00   30 drain* r1n[1,3-14,16,20,23-24,26-33,36,38],r2n[17,20-21]
long      up 1-00:00:00    3 down* r1n[17,37],r2n18
test      up 1-00:00:00    2 drain* r1n[39-40]
verylong  up 7-00:00:00    1 down* r3n1b7
verylong  up 7-00:00:00   20 alloc r3n1b[1,3-6],r3n5b[1-8],r3n9b[1,3-8]
verylong  up 7-00:00:00    1 down r3n1b8
short     up  4:00:00     1 drain r2n40
short     up  4:00:00     1 drain* r2n38
short     up  4:00:00     2 idle r2n[39,41]
newlong   up 1-00:00:00    1 drain r3n24b1
newlong   up 1-00:00:00   10 alloc r3n26b[1-2,8],r3n28b[1-7]
newlong   up 1-00:00:00   13 idle r3n24b[2-8],r3n26b[3-7],r3n28b8
[[greenwood@rin2 ~]$ █

```

Figure 1 - Example of text based access for submitting jobs to a cluster

However, the presentation of graphical information is not directly possible. SSH can be combined with the use of a graphical interface (so called X-forwarding) where a graphical desktop can be displayed and controlled remotely. However, the performance of this over the Internet is suboptimal and generally not sufficient for graphical remote work across countries and continents.

As outlined in the project's proposal, we are investigating two technologies and approaches to improve the possibilities and user experience of remote data analysis, and to expose the computing and high performance computing resources of the research facilities, and later the European Open Science Cloud, to a wider range of users; including those who cannot carry out their work using a terminal connection.

One technology is based on browser driven Remote Desktop execution, i.e. the provision of virtual machines that are tailored for particular data analysis tasks and which can be controlled remotely through a web browser which hosts a graphical desktop interface. This is the most generic approach for remote data analysis as any application currently used can be hosted in the virtual machine and the 'usual' interface (be it graphical or command line) can be displayed remotely in the browser. We thus take an environment that is already known to the users from working at the facility, and make it accessible remotely. This technology is in particular required for a range of legacy software tools and ready-made GUI based applications.

The second technology is based on the Jupyter Notebook interface and data analysis being carried out within a Jupyter notebook, which is executed within an appropriate software context for the data analysis task at hand. This approach has advantages of being designed for remote access, allowing users to access the service with the webbrowser of their choice on their machine, and allowing better support of the FAIR principles through more intrinsic

reproducibility. While we predict the importance of this approach to grow, the notebook hosted data analysis service is not usable for all data analysis requirements, and thus complementary to the remote Desktop approach.

3.0 Remote Jupyter Service

3.1 Jupyter Notebook

The Jupyter Notebook itself is a web application that allows users to create and share documents containing live code, equations, visualisations and descriptive text (Figure 2). Jupyter notebooks are popular within the scientific community because of their ability to easily share data analysis workflows presented in a clearly defined order.

Code cells show code input and output:

```
In [1]: 1 + 2
```

```
Out[1]: 3
```

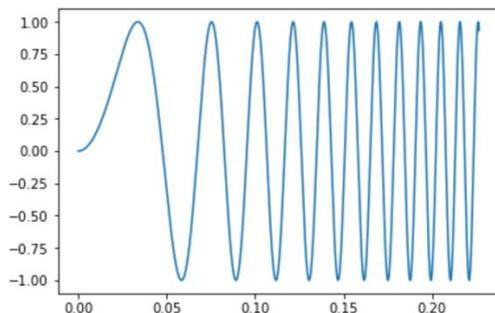
Cells can contain text and latex equations such as $f(x) = \sin(2\pi\omega t^2)$ and $\omega = 220$ Hz. We can use code to define the corresponding functions:

```
In [2]: import numpy as np
def f(t):
    omega = 220
    return np.sin(2 * np.pi * omega * t**2)
```

Let's compute the data and plot the beginning of it:

```
In [3]: t = np.linspace(0, 2, 44100)
y = f(t)
## Show plots inside the notebook
%matplotlib inline
import pylab
pylab.plot(t[0:5000], y[0:5000])
```

```
Out[3]: [<matplotlib.lines.Line2D at 0x1047ada58>]
```



We can integrate media: images, videos, interactive elements and sound:

```
In [4]: from IPython.display import Audio
Audio(y, rate=44100) # plays the data in y as audible signal
```

```
Out[4]: 
```

We can connect other languages and tools, for example execution in bash:

```
In [5]: %%bash
echo "Some shell command, run at `date`"
Some shell command, run at Wed 14 Feb 2018 09:56:18 CET
```

Figure 2 - Jupyter notebook example

3.2 JupyterHub

JupyterHub provides a multiuser implementation of the Jupyter Notebook, enabling users to run notebook servers on shared hardware. JupyterHub allows for user and resource management by system administrators and can run using a variety of backends.

3.3 Jupyter Service Prototype Status

The current status of remote analysis services is shown in Table 2. All facilities have some form of JupyterHub available, whether as a pilot or production service.

In terms of use cases, all facilities have data analysis packages that can be invoked via Jupyter notebooks, with some (EuXFEL/CERIC-ERIC/ESS) expecting it to be available as part of the live data analysis process.

Table 2 - Jupyter Service status at each facility

Facility	JupyterHub	
	Status	Spawner
CERIC-ERIC	Pilot	Docker, Singleuser, Kubernetes, Singularity
ELI	Pilot	Docker, Docker Swarm
ESS	Pilot	Kubernetes
ESRF	Production	Kubernetes, Slurm, Singleuser
ILL	Pilot	Sudo
EuXFEL	Production	Slurm, Singleuser

3.4 Jupyter Service User Experience with Demonstrators

In this section we discuss the experience of users interacting with a typical Jupyter service – the demonstrator max-jhub – and include comments from the instrument scientists at EuXFEL describing their experience with the service.

Max-jhub is a Jupyterhub service that has been in running in production at EuXFEL and DESY since November 2018, seeing around 150-200 users per month. Together with the effort to enable data analysis on this infrastructure, this is the demonstrator for Jupyterhub services (D4.2 in WP4). It is available at <https://max-jhub.desy.de>.



User experience is described below. We show the entry point where a user can access a webpage (Figure 3) and is able to select a particular resource type, dependent on their data analysis requirements. They will then be able to create and execute Jupyter notebooks. An example of this is shown in Figure 4, with part of a notebook which loads and visualizes some analysis results. Throughout this process the user does not need to set up their own Jupyter notebook server for secure remote access.

Maxwell Jupyter Job Options

Maxwell partitions

Choice of GPU

Note: For partitions without GPUs (or choice of GPUs) the GPU selection will be set to 'none'

Constraints

Note: This will override GPU selections!

Number of Nodes

Note: Number of nodes will be set to 1 on shared jhub partition!

Job duration

Note: on the shared Jupyter partition (jhub) the time limit is always 7 days!

Launch modus

Remote Notebook

Node and GPU availability					
Partition	# nodes	# avail	# GPUs avail	# P100 avail	# V100 avail
jhub	3	3	0	0	0
all	427	63	0	0	0
allgpu	75	0	0	0	0
cfel	20	1	0	0	0
cms-desy	4	0	0	0	0
cms-uhh	6	0	0	0	0
cms	10	0	0	0	0
cssb	9	0	0	0	0
exfel	254	50	0	0	0
maxwell	66	8	0	0	0
petra4	26	0	0	0	0
ps	35	1	0	0	0
psx	20	2	0	0	0
uke	16	0	0	0	0
upex	254	50	0	0	0

Spawn

Figure 3 – Launch page of Maxwell Jupyterhub hosted at DESY/EuXFEL



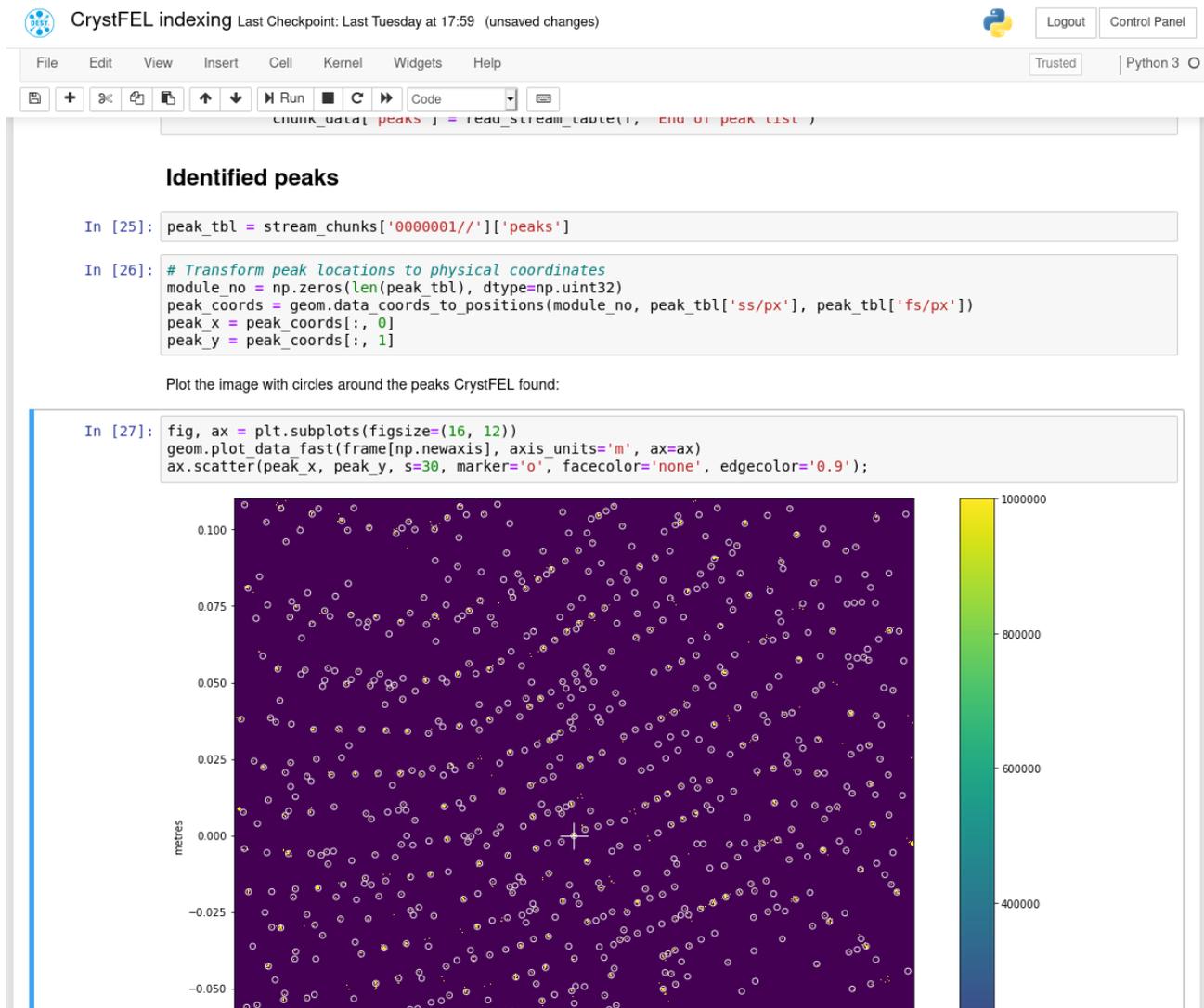


Figure 4 – A notebook in use on max-jhub, examining results from a simulated crystallography experiment

The benefits listed above are further emphasized in the following comments from instrument scientists. It should be highlighted that in addition to the ease of use for less expert users, the improvements for expert users are also mentioned – particularly the flexibility of adding to a workflow for a Jupyter notebook, so that it can cover edge cases other standard applications may not.

“I think the main advantage of max-jhub, in particular compared to direct access to maxwell

nodes, is its accessibility and significantly lower requirements of technical (e.g. SSH, linux command-line, SLURM, etc) expertise.

I can attest to that from my own experience in 2018, where I first joined FLASH experiments with more immediate computing needs as well as the first SASE3 EuXFEL experiments. Here, I had significantly less expertise yet of SSH and working on remote machines. After trying out remote IPython kernels for Spyder and tedious/fragile manual port forwarding, max-jhub made my work significantly easier. It is directly accessible from the (outside!) internet, keeps the session across days "automatically" and networks (beamline + hotel wifi) and still provides ample computing power. At the time, only the jhub partition could be used, but as it was relatively new, those nodes were largely empty. Looking back now, my proficiency in configuring my software stack to my needs makes working on any node as easy as max-jhub. However, I still use max-jhub regularly if all I want to do is take quick look at some previous analysis, remake a plot or discuss over data.

The same applies now to other scientists I support. It is still comparably hard for lots of them to grasp the Linux environment, in particular if they're used to work within Windows and pre-compiled GUI software, and do all the things required to have a working connection to a personal Jupyter. Being able to browse to a page, login, select your allocation and having a running Jupyter notebook within seconds enables lots of these people to actually use maxwell at all. Hence, if jupyterhub is not working, some of these colleagues may be unable to work at all by themselves." – **Phillipp Schmidt**

"If we agree that jupyter notebooks are extremely useful for data analysis—[...]—we should distinguish between expert users and non-expert users of JupyterHub. I would define an expert user as somebody who is comfortable with using the shell, working with python environments, etc.

For expert users, JupyterHub is a convenient tool because it speeds up your workflow. With just a few clicks you can spawn a jupyter server and start working. Moreover, it provides configurations for more complex setups, e.g., using gpus for machine learning, etc. Although you could in principle build your own configuration, you can be sure that the real experts (CAS-DA) make sure that software is properly installed and configured on Maxwell so that you are already good to go.

In case of non-expert users, in my opinion, JupyterHub is essential for conducting successful experiments. First of all, it is super easy to use and lowers the slope of the initial learning curve, which is anyway extremely steep. Secondly, the users can directly start to use example jupyter notebooks, e.g., the ones you provide in the documentation of extra_data. Experience from MID shows that basically all user groups—expert and non-expert users—started their data analysis

based on the jupyter notebooks I provided them. Thirdly, sometimes users are quite familiar with jupyter notebooks because they are using them on their laptops, e.g., with Anaconda. JupyterHub on Maxwell, therefore, allows them to work in a way they are already comfortable with just by opening a browser. This makes it also easier for the beamline scientists because we do not have to explain the users how to allocate a node, spawn a jupyter server, etc.” – **Mario Reiser**

“I have two use cases:

1) fast online analysis of some aspect and during the experiment, which cannot be done with a standard tool. Users at EuXFEL have a lot of freedom in designing their experiment. This sometimes requires a non-standard analysis which is not covered by tools like EXtra-foam. Here, Jhub can be a very convenient. I was moving back and forth between directly working on Maxwell and Jhub depending on what exactly I want to do.

2) Offline data analysis. Currently I'm using Jhub mostly for analysis my FEL FLASH data. The big advantage to me is, that I can share my notebook with colleagues at the university. They are not experts in Python but Jhub helped a lot to make it easier for them. I setup python environment for them and we can work more efficiently together. One nice feature for Jhub would be, if one could share notebooks cross accounts easier.

Jhub makes it easy for non-expert users to get familiar with analysis FEL data. During my beamtimes as PI at FLASH FEL, I first tried the show my team how to access maxwell directly (via ssh etc) and install python analysis codes. This was a difficult task due to different knowledge level, different computers (from mac, PC, Linux) and different python distributions. During a later beamtime, I prepared the software environment on Jhub. Everything was working right from the beginning.

I would strongly recommend to keep Jhub. Experiments at FEL can be very diverse and not all aspects can be implemented in a separate tool. I would investigate ways to share jhub notebooks easier between accounts and also ways, to access data from tools within jhub notebook.” – **Markus Scholz**

“1. Documentation and reproducibility

We've been using jupyterhub-based analysis extensively during our beamtimes as well as when writing reports, preparing talks and, of course, drafting publications. One thing I'd certainly miss would be the convenient documentation of our work in (markup) text and images. I just had to come back to analysis code from about a year ago to prepare a paper submission and the combined code comments and dedicated markup cells make it much easier to understand what we've been doing then. This directly extends to the journals trying to enforce better reproducibility by asking us to make data and analysis code available as part of our publications (which I think is a very important part of scientific credibility). My current experience is, that journal editors are quite happy to get a well-formatted notebook with

figures and extensive explanations (that would obfuscate the actual code in the form of comments). This makes it easier and less effort for us to comply to the journals' data/analysis availability guidelines.

2. Collaboration

Notebooks are easy to prepare/modify for colleagues working on different aspects of the analysis as well as for less experienced users, so that they can explore parts of the data on their own. Especially for the latter group, starting a notebook server on max-jhub is a lower hurdle compared to working on a bare console over ssh and gets them productive much quicker.

At the same time, collaborative features are one thing that I think could be improved upon. Using the jupyter git plugin helps with maintaining different versions of the notebooks among several people, but sometimes a true (real-time) collaborative work flow would be very handy. I personally haven't tested cocalc (<https://cocalc.com/doc/jupyter-notebook.html>) yet, but it seems that something like that could be very useful.

I hope these thoughts are useful for your report. As I said before, the max-jhub service is incredibly useful to our work and I'm very thankful to all of you offering and maintaining it as a service!" – **Michael Schneider**

3.5 Technical Deployment

In this section we discuss different possibilities of deploying and running a JupyterHub installation, and describe approaches of the different facilities to provide this service within their given infrastructure and infrastructure constraints. This discussion is not specific to the demonstrator, max-jhub, but aims to give an overview of the technology used for the range of Jupyter services running at all participating facilities.

In the most basic form, JupyterHub can run on a single server. An example of this already exists at the ESRF, where they have been providing single machine sudo-spawner based JupyterHubs for two beamlines. One of these deployments has been running for over two years in production. It is also possible to use a Docker-spawner for Jupyterhub, as is deployed in both ELI Beamlines as a pilot for remote analysis of particle-in-cell simulations. Of the two JupyterHub distributions supported by the JupyterHub community, one supports running on a single server only, although it is recommended only to be used for under 100 users (<https://tljh.jupyter.org/>).

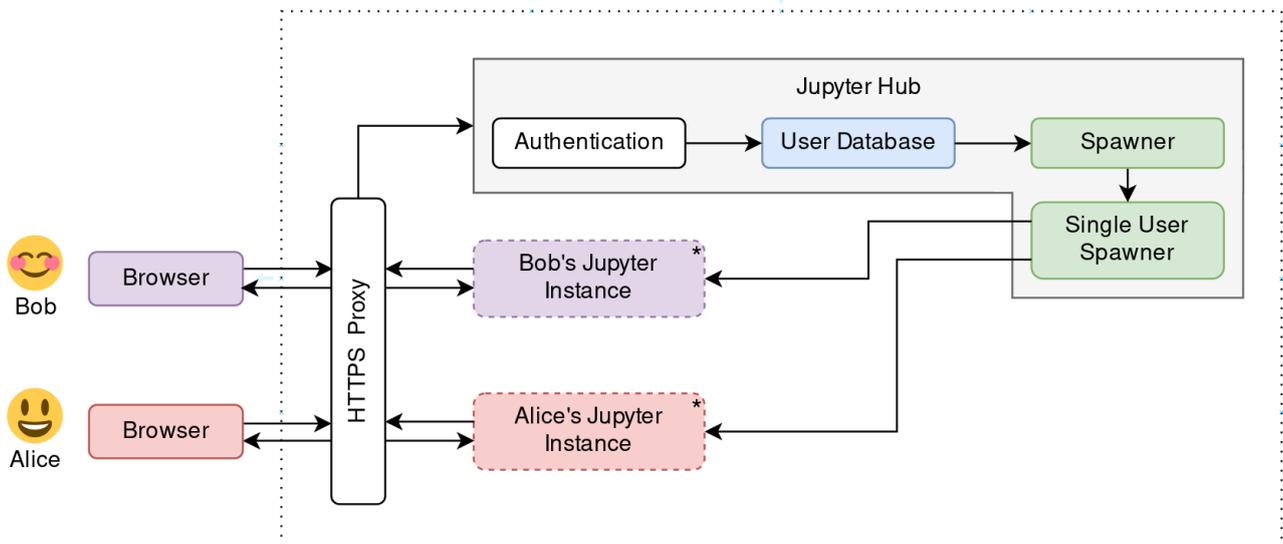


Figure 5 - JupyterHub server architecture

Pre-existing resource managers at facilities can be used for spawning new users notebook servers – such as Kubernetes, Docker Swarm or Slurm. In this case users' Jupyter instances (Figure 5) can be distributed across multiple servers, and it is possible to take advantage of already existing Graphics Processing Unit (GPU) or High-Performance Computing (HPC) resources.

Several facilities have a JupyterHub set up using Kubernetes, either as a pilot for providing scientists/users a place to do exploratory offline data analysis, and as part of tutorial workshops for future users/collaborators (eg in ESS). The second JupyterHub distribution supported by the Jupyterhub community is configured to work with a Kubernetes backend (<https://z2jh.jupyter.org/>).

Docker swarm provides similar functionality to Kubernetes as a resource manager and is used in ELI ALPS to provision notebook instances.

JupyterHub with user instances running in containers - such as with Kubernetes - can be further extended with the `repo2docker` tool. This prepares a container image with the software libraries needed to run a collection of notebooks, based on a requirements file saved alongside the notebooks. This technology forms the basis of the public Binder service (<https://mybinder.org/>). Binder offers anonymous, temporary access to run notebooks with no persistent storage; this is not precisely what we want to offer, but its mechanisms to create and manage container images are of significant interest for facilitating reproducible analysis.

Slurm, along with other batch job schedulers, can be used as the backend provider in the same way as Kubernetes. In this case the user's Jupyter instances are run as jobs on the cluster, and provided the JupyterHub is configured correctly, users may request certain resources for their instance, such as GPU, or a particular number of cores. EuXFEL currently have this setup

working in production as part of their ‘max-jhub’ service (Figure 3).

Other facilities, such as ESRF, have a pilot JupyterHub integrated with Slurm, and are expecting to move it into production before the end of the year.

There is another option for integrating Jupyter Notebooks and Slurm – using python packages such as `dask_jobqueue` or `pyslurm` it is possible to submit slurm jobs directly from the notebook. For this workflow users are expected to do the visualisation within the notebook, and only use the HPC cluster for heavier computation.

Such a configuration might work well for the case of McStas, an application for simulating neutron scattering instruments and experiments. Part of the McStas workflow (generating the instrument definition file) is relatively lightweight and can be done using a Jupyter notebook with fewer resources, whilst the Monte Carlo simulations can require HPC level resources. Therefore the most effective use of resources would be having the user mainly work on a smaller Jupyter instance, whilst submitting specific parts of the workflow as jobs to the cluster. This is shown in Figure 6.

SNAP Simulation with DAC

This notebook simulates the SNAP instrument at SNS with a diamond anvil cell using McStas. The simulation is written using McStasScript, a python API for McStas. The diamond anvil cell is simulated using the Union components, which allow for multiple scattering between all parts of the anvil. The instrument follows the conventions necessary to export the data to a NeXus file and read with Mantid, but McStasScript does not. In order to achieve this, write the instrument file with McStasScript and run separately. Most of the code is found in modules for convenience, these are:

- `SNAP_functions` (contains function for adding powders and function for creating a diamond shape)
- `SNAP_materials` (contains the Union material definitions)
- `SNAP_instrument` (contains the source, choppers, guide)
- `SNAP_DAC` (contains the diamond anvil cell)
- `SNAP_monitors` (contain monitors, Union loggers, Standard monitors and Mantid monitors)
- `SNAP_plotters` (contains functions for plotting returned data)
- `SNAP_cluster` (contains classes for submitting jobs to DMSC cluster)

```
In [1]: import sys

import numpy as np

import SNAP_instr
import SNAP_monitors
import SNAP_cluster

from mcstasinterface import instr, plotter, functions
```

Simulation settings

Here are the overall simulation settings that influence how the McStas file is built.

- `union_loggers`: if True, Union loggers that show scattering in the DAC is enabled
- `standard_monitors`: if True, standard monitors that show for example scattering in 4PI is enabled
- `mantid_monitors`: if True, event mode TOF monitors are enabled
- `enable_guide`: if True, the guide in the 2018 SNAP instrument file is simulated
- `gasket_choice`: "OR_drawing" or "RE_type"

```
In [2]: SNAP = SNAP_instr.generate_SNAP(union_loggers=False,
                                     large_union_loggers=True,
                                     standard_monitors=True,
                                     wavelength_monitors=False,
                                     mantid_monitors=False,
                                     detectors=False,
                                     detector_Al_sheets=False,
                                     enable_guide=False,
                                     gasket_choice="OR_drawing", # OR_drawing or RE_type
                                     sample_material="\NaCalf\\"",
                                     split=100)
```

Create cluster jobs

Here we create jobs for the ESSS DMSC cluster. We want a dataset for each interesting wavelength, but could also investigate other series. The job needs to be uploaded to the cluster, where `compile_all.sh` and `run_all.sh` are to be executed. No modules need to be loaded.

```

In [6]: bragg_wavelengths = [2.37, 1.95, 1.19]

bragg_peak = {"lambda": bragg_wavelengths[0], "d_lambda": 0.04,
              "final_slit_x": 0.0005, "final_slit_y": 0.0005,
              "diamond_mos": 30,
              "diamond_delta_d_d": 1E-3,
              "diamond_up_misalignment": 0.2,
              "diamond_up_rotation": 12.0,
              "diamond_down_misalignment": -0.1,
              "diamond_down_rotation": -40.0 }

rotation = np.linspace(0,88,45) # Which to investigate these rotations

# Create job instance, this can contain many jobs
main_job = SNAP_cluster.job("job_full_rotation")

# Add a instr job, this can contain many tasks but will all be on same instrument and same cluster queue
instr_job = main_job.add_instr(SNAP, "SNAP", queue="newlong")

# Loop over all the tasks to be added to instr_job
for index in range(len(bragg_wavelengths)):
    for mis in range(len(rotation)):
        bragg_peak["lambda"] = bragg_wavelengths[index]
        bragg_peak["diamond_up_rotation"] = rotation[mis]
        instr_job.new_task(bragg_peak, name="bragg_" + str(index) + "_" + str(mis) , ncount=4E7, nodes=1)

# close the main job, this writes the batch files to disk
main_job.close()
  
```

```
In [4]: data = DATA["bragg_0_0"]
```

SNAP is now an instrument object that can for example execute a McStas simulation of SNAP with the following parameters:

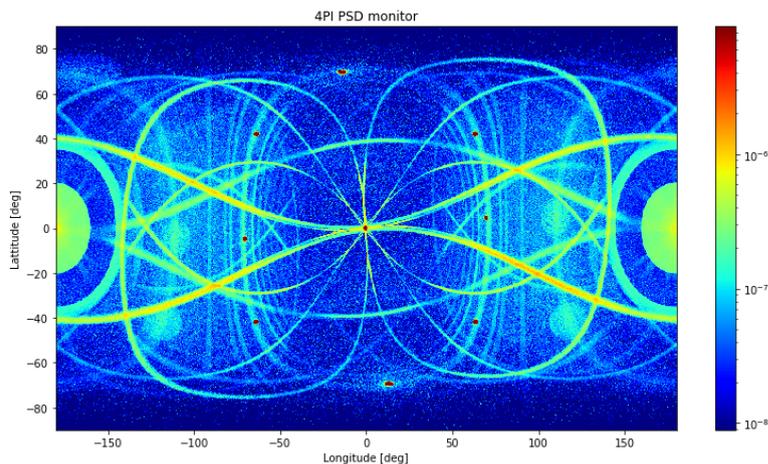
```

In [5]: import SNAP_plotters
import importlib
importlib.reload(SNAP_plotters)

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 10]
index = 0

SNAP_plotters.full_sphere(data, cut_max=0.00005, orders_of_mag=3.0)

number of elements in data list = 1
Plotting data with name standard_monitor_4PI
  
```



Load cluster job

Provide the name of the job, then all data is loaded

```

In [2]: # Provide name of output folder
foldername = "job_full_rotation/output_SNAP"
# Find data sets (folders)
data_folders = os.listdir(foldername)
search_term = "_0_"

print("Found data folders:", data_folders)

# Load each and save in dict
DATA = {}
for data_folder in data_folders:
    if search_term in data_folder:
        DATA[data_folder] = functions.load_data(foldername + "/" + data_folder)
  
```

Figure 6 Part of example McStas notebook demonstrating Instrument generation, submission of a cluster job and

4.0 Remote Desktop Services

4.1 Remote Desktop

A Remote Desktop Service offers a remote data analysis experience that appears as if the display of a local computer is available remotely. This includes the window manager and operating system.

A number of data analysis applications have a requirement for a GUI, or have only partial functionality available from an API and can thus not be driven from a Jupyter Notebook. Users may also be accustomed to a particular user interface, and not wish to move their workflow to a Jupyter notebook. Whilst Jupyter notebooks can be considered more user friendly for inexperienced users than the command line, a well-designed GUI probably has the lowest bar of entry. For these use cases, we anticipate use of the Remote Desktop service for remote data analysis.

There are a variety of technologies used to provide Remote Desktops within the facilities, the main use cases of which are detailed below.

4.2 Remote Desktop Prototype Service Status

All facilities have a remote desktop service available, although in some facilities data analysis is available as a pilot rather than production basis (See Table 3 for overview).

In terms of applications proposed as target use cases by facilities, some are only available via a GUI, and others only provide full functionality when interacted with via a GUI.

Table 3 - Status of remote desktop service

Facility	Remote desktop	
	Status	Technology
CERIC-ERIC	Pilot	VNC, Guacamole, XPRA (X Persistent Remote Applications)
ELI	NA	NA
ESS	Pilot	VNC
ESRF	Production	NoMachine
ILL	Production	VISA (XRDP, Guacamole)
EuXFEL	Production	FastX

4.3 Remote Desktop User Experience with Demonstrators

In this section we discuss the experience of users interacting with a remote desktop service, and describe different approaches from facilities to provide this service, with focus on the demonstrator.

When using a remote desktop to analyse data, the user experience for interacting with the remote desktop itself is similar between all facilities. The differences in user experience occur in how users access the remote desktop (by web browser, or desktop client), and their purpose for doing so (covered in 2.1 Motivation).

4.3.1 ILL

Virtual Infrastructure for Scientific Analysis (VISA) is a remote desktop service in production at ILL. It is proposed as the demonstrator for remote desktop services to fulfil the WP4.2 deliverable requirements and is available internally at <https://visa.ill.eu/>.

A typical remote desktop user experience can be demonstrated by VISA. A user authenticates via a web page (Figure 7) and is able to request an instance (Figure 8). They are then able to select which experiment data to analyse (Figure 10) and several other configuration options (Figure 9 and 11). The virtual machine is deployed on request with access to the results data, and with a set range of software preinstalled. Figure 13 shows the remote desktop view for users.

There are several features available to enable knowledge sharing between proposal group members. These include the ability to share a machine with another user (Figure 14) – either as read-only, or full control and a planned chat room integration for ease of communication.

The user does not need to manage the installation of scientific software, or configuration of their machine and can instead focus on analysing their data. Two months after putting this service into operation, the ILL reported an increasing number of users, with over 30 unique users across the past few weeks, along with positive feedback.

The remote desktop service at ILL is currently only used for offline data analysis (after the experiment), but there are plans to expand it to cover both online data analysis and to use prior to the experiment for simulations.

Data Analysis, in the cloud.

VISA makes it simple create compute instances and analyse your experimental data using just your web browser.



Sign in using your ILL account

Access your data

Access directly your experimental data.

Analyse your data

Create a new compute instance and use your web browser to access a remote desktop to start analysing your data.

No need to install software

The compute instances come with pre-installed data analysis software.

Questions?

Contact the ILL IT services at data@ill.eu for more information.

Figure 7 - VISA Login screen

Figure 8 – VISA computer instance view for authenticated user

NEUTRONS FOR SOCIETY

Home

Create a new computing resource

Define your computing environment

Choose an environment

Basic System

Basic data analysis environment

Standard Analysis

Advanced data analysis environment

Choose hardware requirements

2 Cores

8GB

Small

4 Cores

16GB

Medium

8 Cores

128GB

Large

Figure 9 – VISA compute instance creation environment options

NEUTRONS FOR SOCIETY

Home

Select your experiments

You need to select at least one experiment in order to create you compute resource.

Proposal	Cycle	Instrument	
> CRG-1975	20131	IN12	UNSELECT

Find experiments Cycle All cycles ▼ Instrument All instruments ▼

Proposal	Title	Instrument	
CRG-1975	Possible incommensurate magnetic order of the eu moments in superconducting E uFe ₂ (As _{1-x} P _x) ₂ (x=0.10, 0.15, 0.26) single crystals	IN12	SELECT
7-04-125	Neutron spectroscopy on Sodium-Intercalated Fullerenes	IN1	SELECT

1 - 2 of 2 experiments

Figure 10 – VISA compute instance creation experiment selection



Finalise and create

Choose a name

Give your compute resource an identifying name you will remember them by. Your compute resource name can only contain alphanumeric characters, dashes, and periods.

Choose a screen resolution

You can choose the screen resolution of your compute resource. Your current screen resolution is: **1920 x 1080**

Add some notes

You can add here some comments and notes about this compute resource.

Here are some notes

Terms and Conditions

I accept that I will only use this instance for data analysis purposes and will not engage in any illegal activities. Any user that I share my instance with will have complete access and permissions to my files. I agree to supervise any shared access and remove users once sharing is no longer required.

CREATE

Figure 11 – VISA compute instance creation configuration options



Compute instances

CREATE A NEW INSTANCE

Home

My instances Instances shared with me

My new instance

Basic System

16 GB - 4 VCPUs

Created on 05/02/2020

● active

Connect

⚙ Settings

Figure 12 - VISA authenticated user view with deployed instances

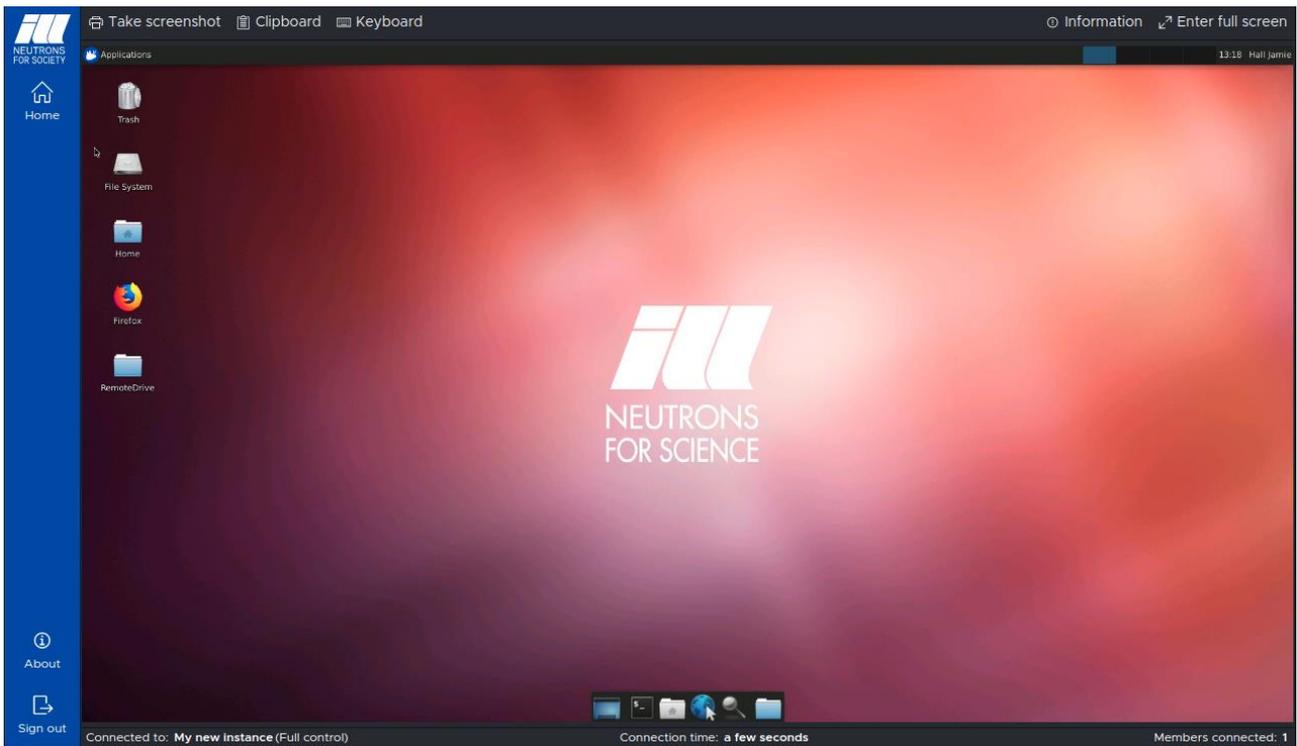


Figure 13 - VISA Remote Desktop view

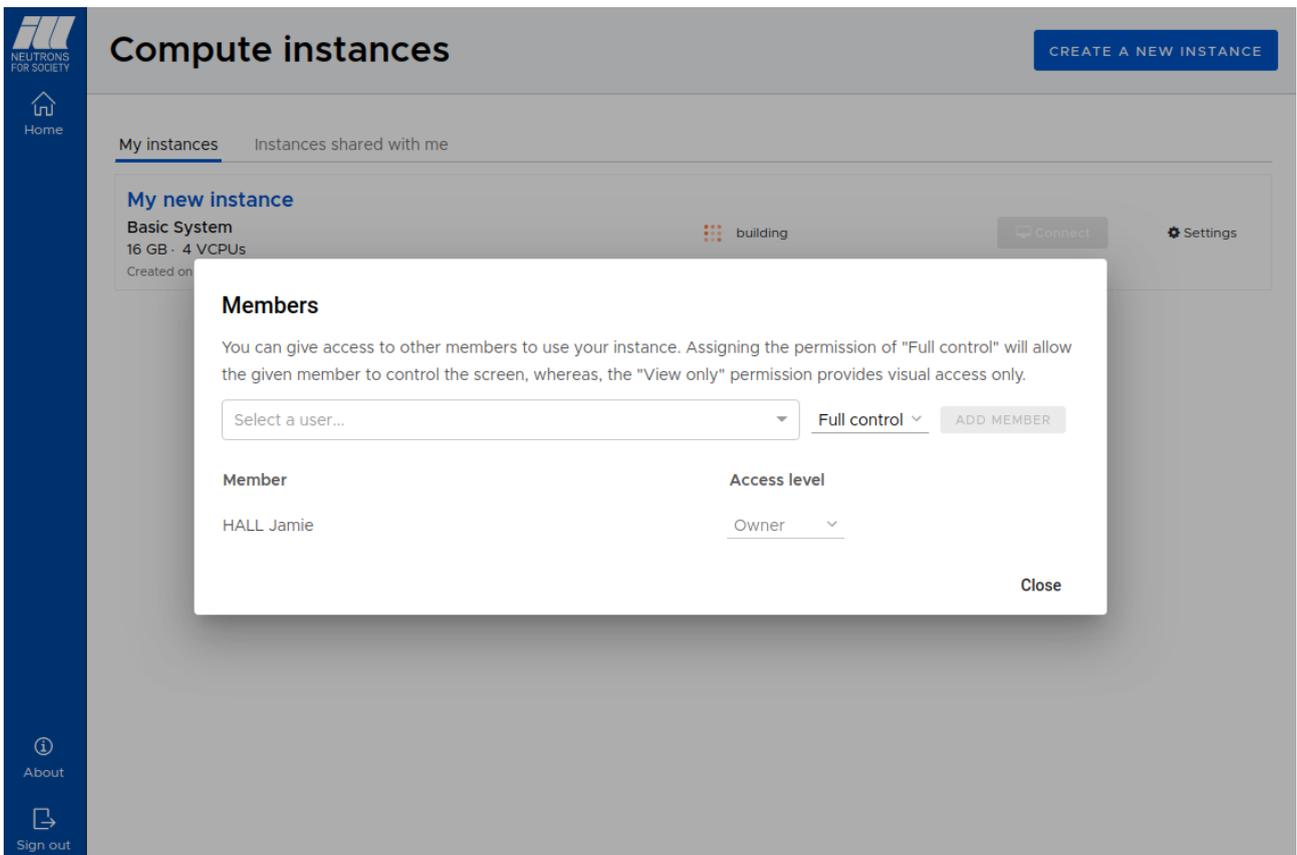


Figure 14 - VISA remote desktop sharing options

4.3.2 ESS

The ESS provides an example of a pilot remote desktop service for online data. Remote desktops would add value as a part of the reduction and live data pipeline that is under development. Whilst the applications that handle data acquisition are command line only, the data reduction application, Mantid (Figure 15), needs a remote desktop view to show relevant graphs with count rates. Whilst not amalgamated into the live data pipeline as yet, some of the data analysis applications that will be used after data reduction also need a remote desktop available for full functionality.

For the pilot set up users are expected to download a desktop client to connect with the server, although it is planned to be provided via the web browser in production.

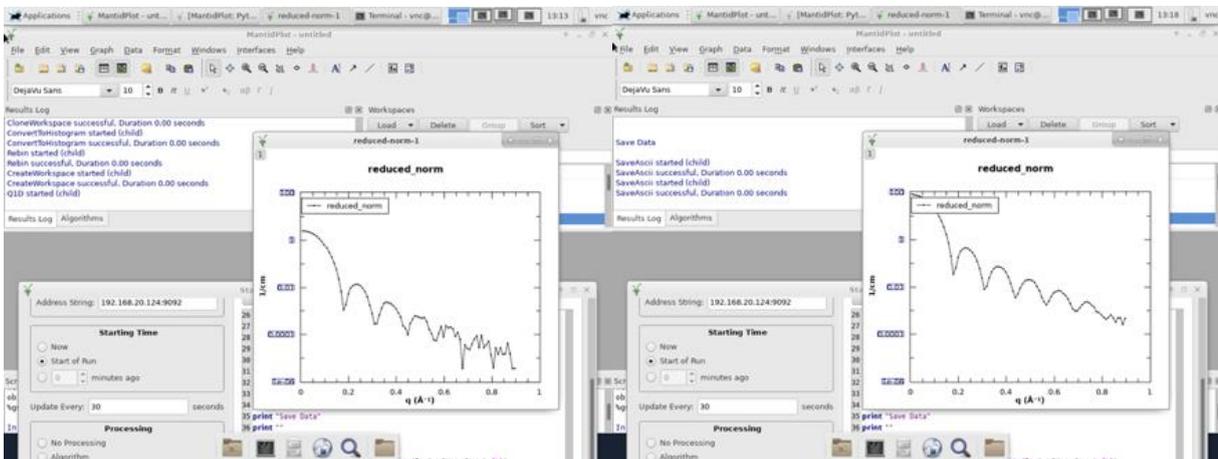


Figure 15 - Mantid GUI showing reduced data at two time points after the data stream has started. Left - +60 seconds after data stream. Right: After all events have been received by Mantid (+306 seconds)

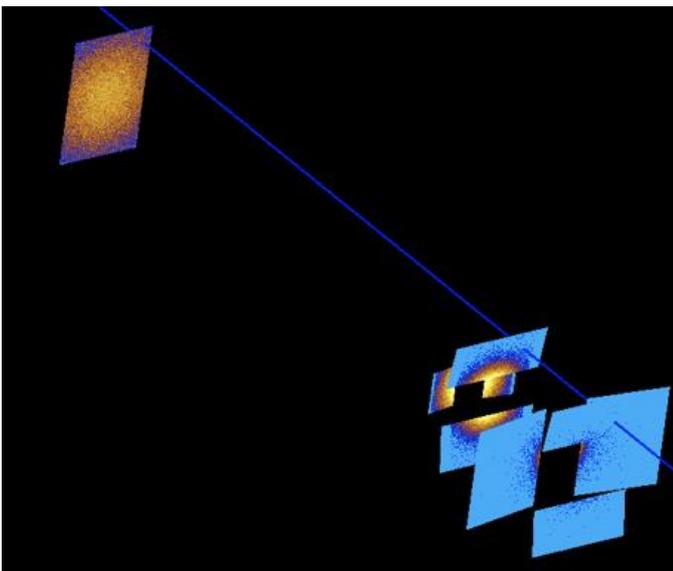


Figure 16 - Mantid instrument view of event data shown in Figure 15

4.3.3 CERIC-ERIC

CERIC-ERIC employs a custom solution for software remotization, developed in house in the context of PaNOSC – ‘RAFEC’ (Remote Application For Edge Computing). This is a DAaaS (data analysis as a service) Linux application and container based spawner. It is accessible via a RESTful API and fully integrated in CERIC VUO (virtual unified office) at the Elettra site and CERIC beamlines workstations. RAFEC acts as bridge for VUO between cloud resources and edge devices with VUO managing the accounts and data access for users.

RAFEC differs from other remote desktop applications discussed within this section in that it does not provide a desktop view from which the user then navigates, but instead directly spawns specific applications. This ranges from Jupyter notebook containerized instances to Linux programs such as PyMCA (an X-ray powder diffraction analysis tool). The software applications are distributed and can reside on either the facility’s cluster resources (ie. ‘the cloud’) or computers located at the facility’s beamlines. This means that data can be processed using a local application with lower latencies. It also enables beamline control systems to be accessible remotely as shown in Figure 17.

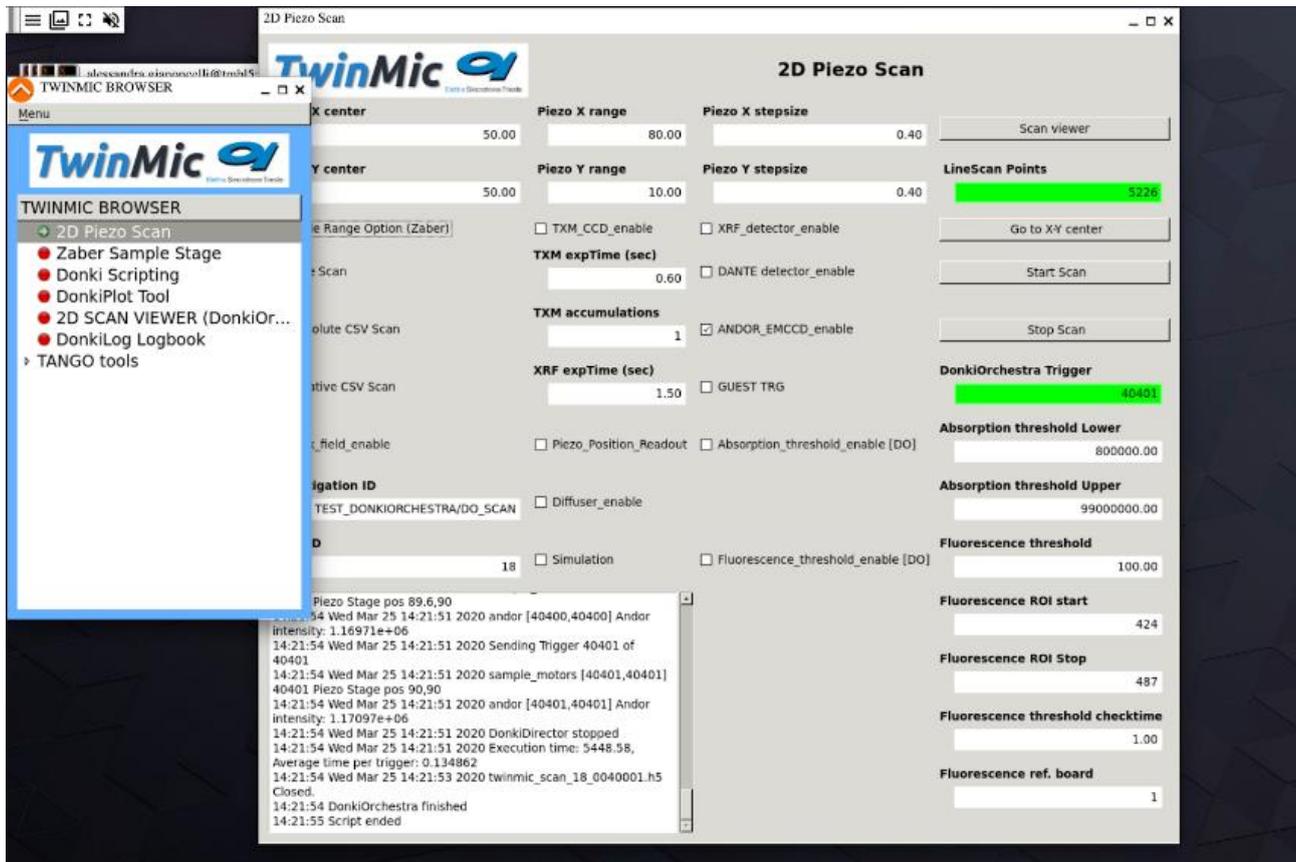


Figure 17 - RAFEC being used to access the beamline controls of the instrument TwinMic remotely

4.3.4 ISIS

ISIS, a facility participating in the sister ExPaNDS project, has a production remote desktop service for online data. Their ‘data analysis as a service’ (ISIS DAaaS) runs on around 10 beamlines. In this set up users are provided with a virtual machine during the experiment. Similarly to VISA, users are able to request an instance of a particular size via a web browser. The machine is then deployed with access to the results data and a Jupyter Notebook single user server is available, along with other relevant data analysis applications. ISIS reports that the service currently sees 20-50 concurrent users and the main limiting factor is availability of resource.

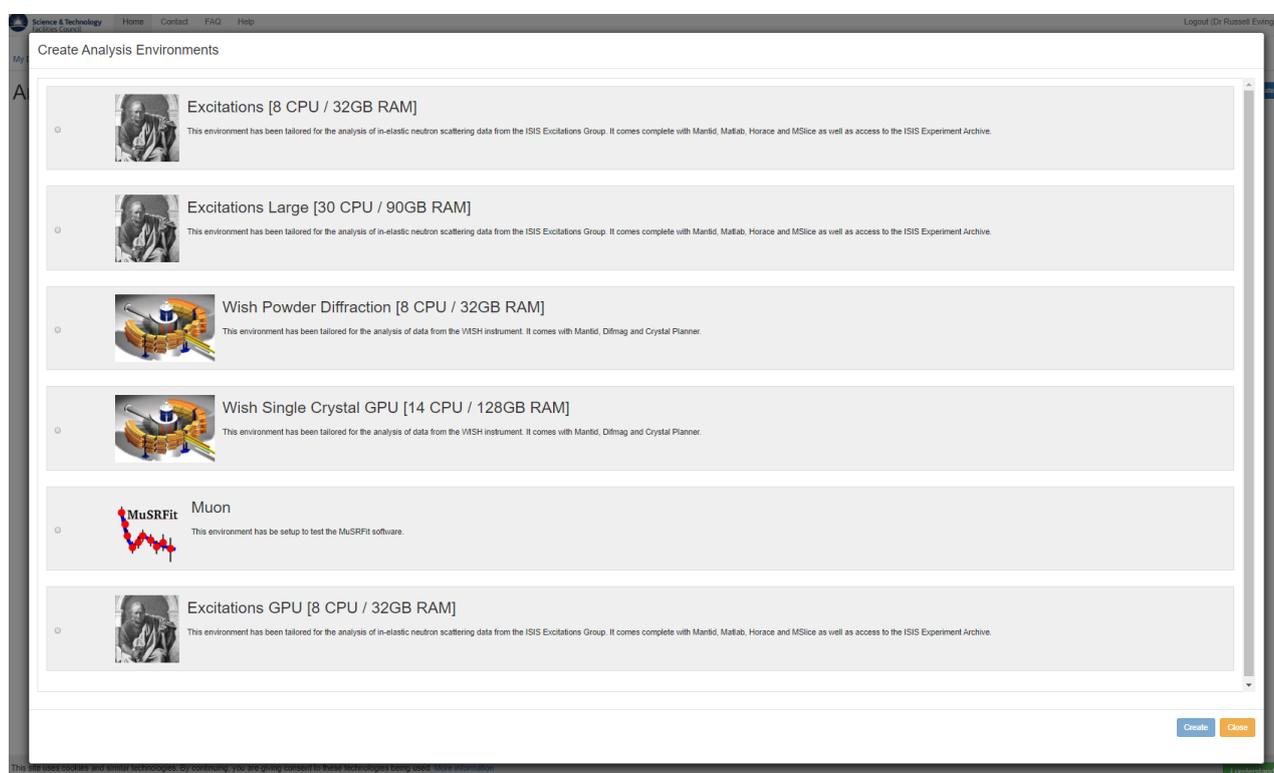


Figure 18 - ISIS Data Analysis as a Service deployment page

4.4 Technical Deployment

In this section we consider the technical details of some remote desktop services, their deployment and configuration.

In the simplest form this would involve a machine with Virtual Network Computing (VNC) server installed accessed via a desktop client. This is demonstrated by the ESS pilot service where remote desktop machines are created on an adhoc basis. Virtual machines are deployed manually via provisioning service (Foreman), and assigned specific roles in a configuration

management tool (Puppet), which then ensures the correct software applications are installed and configured.

ILL's VISA service (architecture shown in Figure 19) provides an example of more developed virtual machine lifecycle and configuration management. Virtual machines are deployed when users request them using OpenStack, a cloud computing platform - Figure 20 shows an instance running within OpenStack. Instead of configuring the machine after provisioning, prebaked images are built every night using a modified version of packer. There are plans to increase the range of images available to cater for specific instrument fields that require different data packages. For example, ISIS DAaaS provides several environments, with different software installed, dependent on the experiment type. They also use OpenStack for provisioning, but additionally use Ansible, a configuration management tool, to further customise the image after deployment.

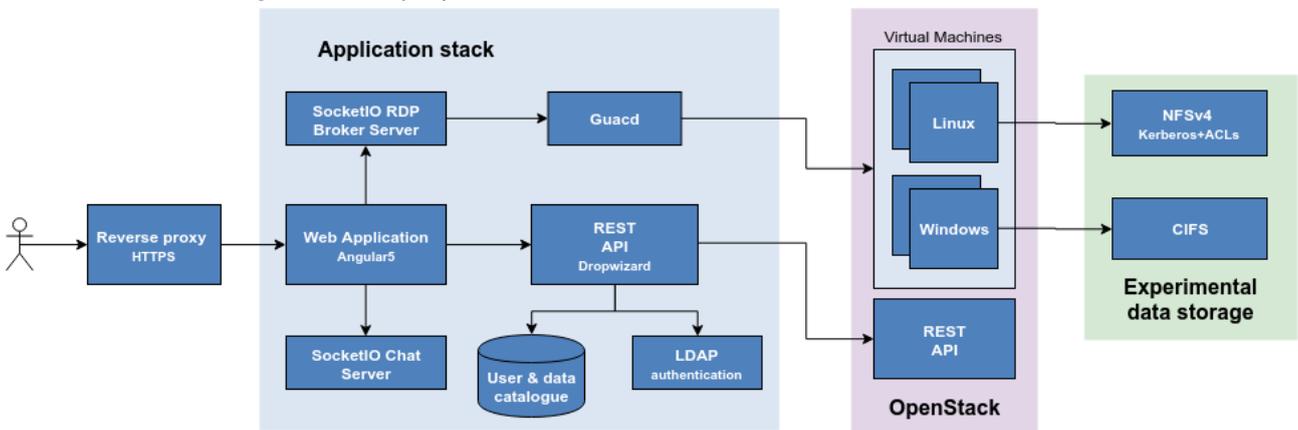


Figure 19 - VISA architecture

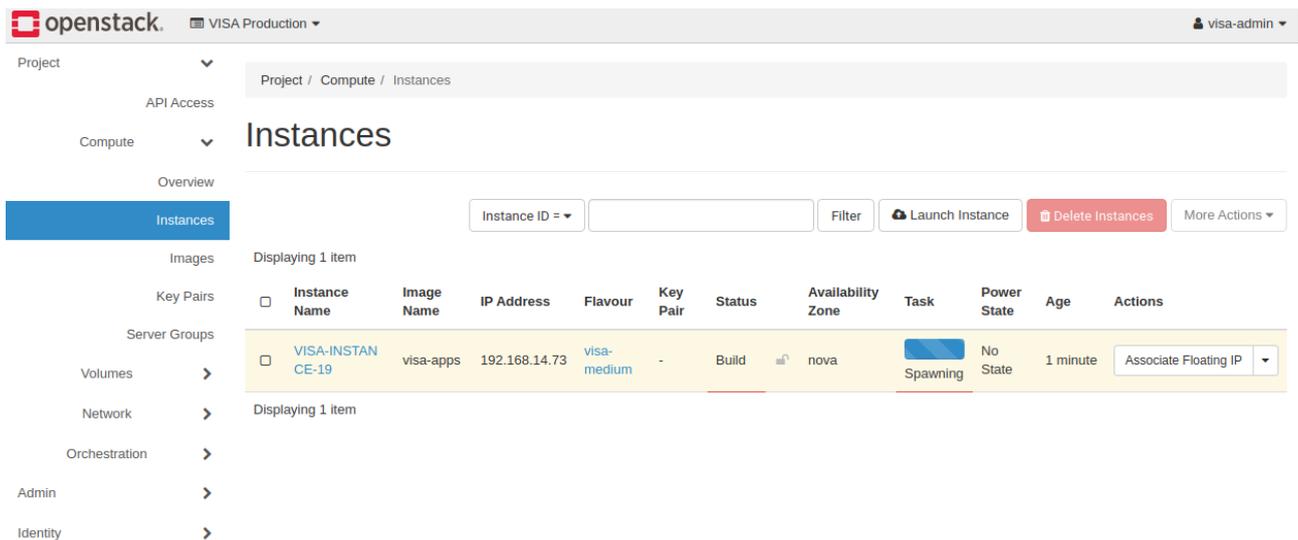


Figure 20 - ILL OpenStack showing VISA image spawning

Several facilities use Apache Guacamole so the user can access the remote desktop service via their web browser. Guacamole is a clientless remote desktop gateway which supports multiple standard remote desktop protocols such as VNC and Remote Desktop Protocol (RDP) (Figure

21).

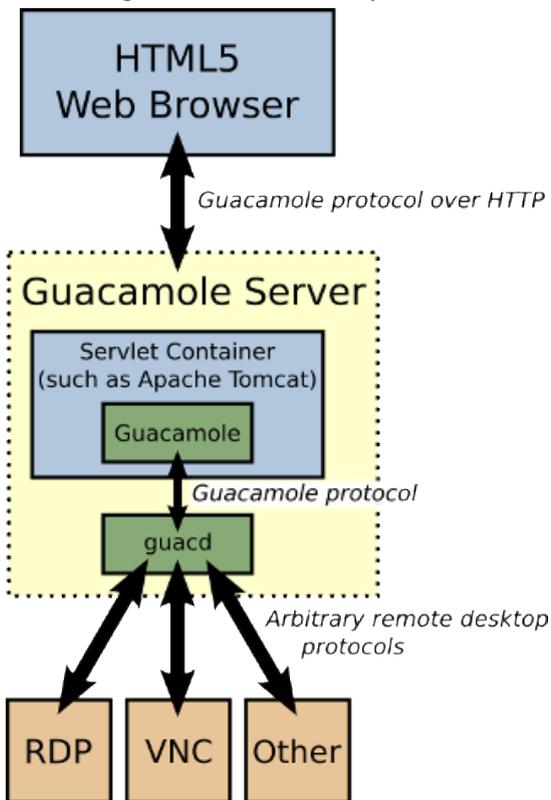
ILL has been using Guacamole and have reported positive feedback from users, in particular direct access to their data, hardware resources, and pre-installed scientific software directly from a web browser.

Whilst Guacamole is effective for the majority of use cases, insufficient performance has been noted with workflows that require intense visualisations.

To mitigate this issue of performance using Guacamole the ILL decided, in the scope of the PaNOSC project, to start developing their own protocol for remote desktop analysis. The new development work has four main objectives:

1. Reduce the time spent processing and the memory footprint on our cloud resources;
2. Reduce the data being sent to the client by using a more efficient protocol;
3. Remove the layers of unnecessary abstraction that leads to a decrease in performance;
4. Aiming to ensure 60 fps but dynamically reducing the frame-rate and increasing the compression ratio if the clients connection is not fast enough

A protocol developed purely for remote desktop analysis on the web should lead to better performance (both on the server and client), less data transfer, reduced load on our cloud resources and also open up the possibilities for applications that require efficient and quick rendering. All of the development will be open-sourced to the wider community.



Remote Desktops

Figure 21 - Apache Guacamole overview

Whilst there are currently a wide variety of methods used between facilities for deployment of resources, it is expected that there will be some convergence toward more homogenous solutions that can work for all participating facilities. For instance, the ESS expects to move

toward using a cloud computing platform such as OpenStack, already used by ILL, or something similar in scope.

This would indicate that one portal is able to manage remote desktop services across multiple facilities, despite minor differences in methodologies.

The integration of these services into EOSC will be handled by collaboration between WP6 (EOSC Integration) and WP4.3 (EOSC integration and common portal for remote data analysis services).

5.0 Towards a Harmonisation of Services

As is clear from the above overview each of the sites offer services at different levels of maturity and are integrated into the local infrastructure on each site. Our approach towards a harmonisation of these services is to develop a common portal that will be deployed at each facility. The common portal will provide abstractions to the various different remote technologies being used at each site.

Each portal instance provides the same user interface and very similar user experience at every site. The portal software will translate user queries, such as search for a data set, or starting the interactive exploration of a data set, into facility specific requests that can be executed on the facility specific computing infrastructure.

Figure 22 illustrates that the portal, indicated by the PaNOSC logo, with its common look and feel (pink box) acts as an adapter to access the required functionality that is implemented in different ways at each facility.

For the integration with EOSC, we could imagine an additional instance of this portal that directs queries to the facilities or a cloud computing resource according to the locality of the data sets. The options are under investigation, and will depend on EOSC developments and the results of Work Package 6.

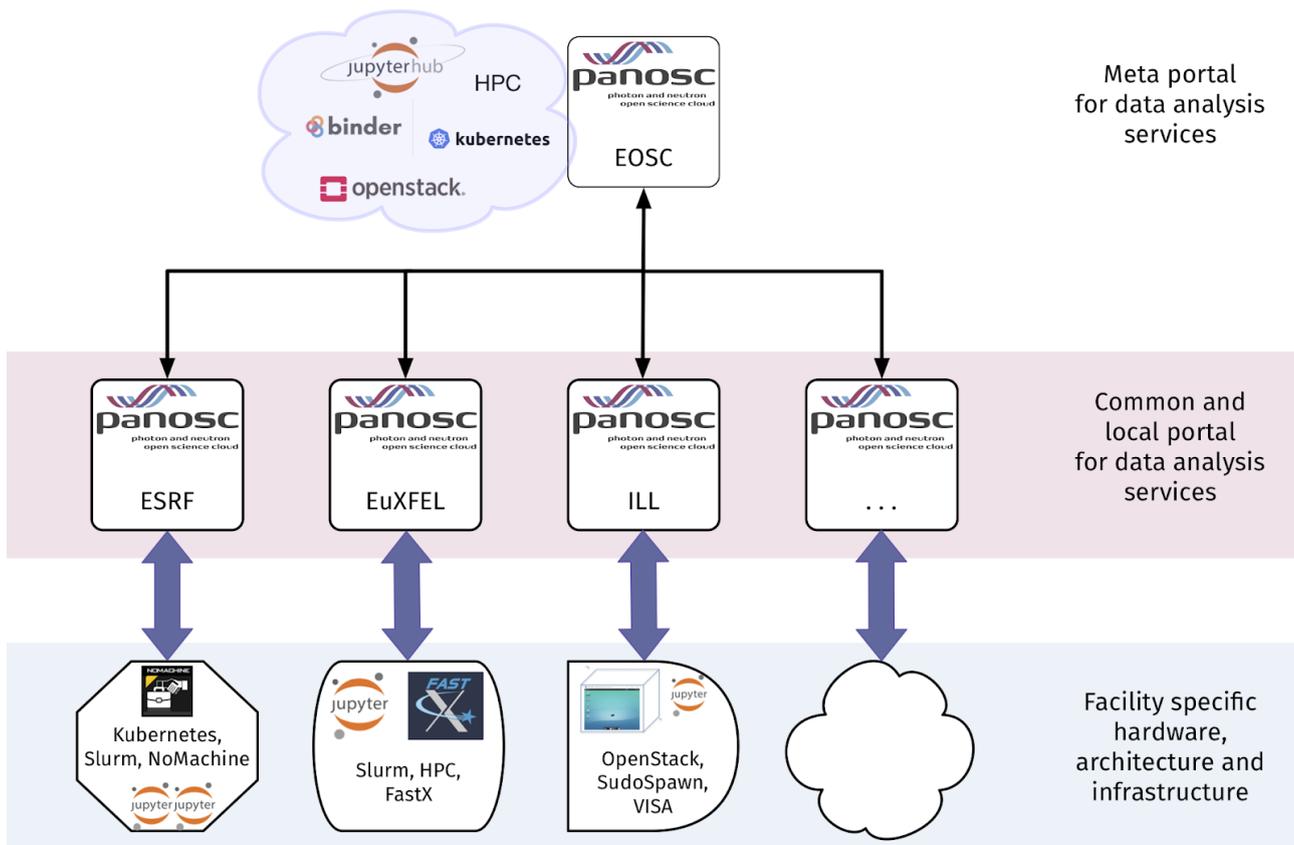


Figure 22 - Each facility (as examples shown are ESRF, EuXFEL and ILL) has established computing infrastructure, visualised through the different shapes, technologies and logos in the light blue box at the bottom. The PaNOSC portal (represented as the PaNOSC logo in a box) in the middle section, acts as a translator from user requests to facility specific compute instructions, while providing a common user experience.

The choice of computing infrastructure and access to data is still governed by each facility but the portal interface and authentication (provided by WP6) will be common across all facilities.

We have defined an architecture specification for the portal [1] after having collected feedback and use cases[2] from all of the partners. The portal is developed with joint input and contributions from WP3.

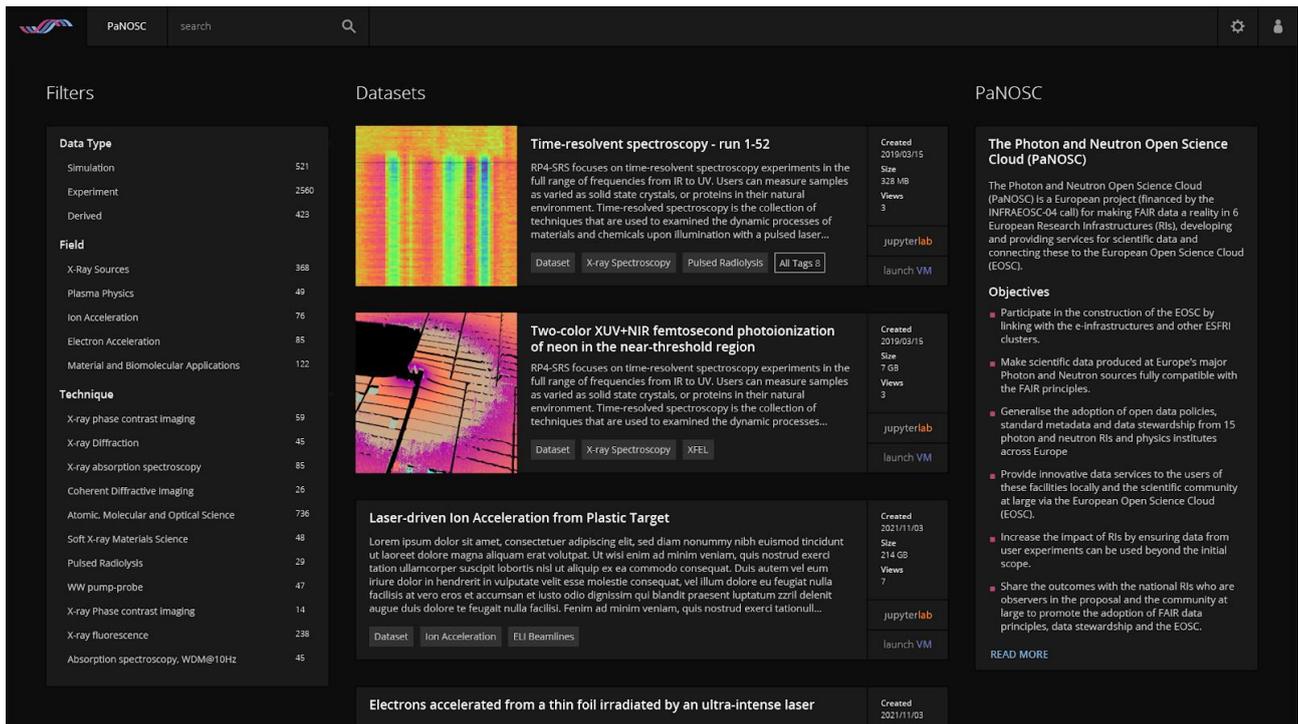


Figure 23 - Mock up of Portal interface

6.0 Summary

We have discussed the status of remote desktop and Jupyter services across the photon and neutron facilities in this project. All facilities now have a remote desktop and Jupyter service available, either in pilot or production status. Facilities are reporting positive user feedback, and in cases where services have been available for longer periods of time they have become a well-established part of the users' normal workflow. Users comment that they find that these services make it easier for them to complete their data analyses, and to share their workflow with others.

Whilst Jupyter services provide easily templated and sharable workflows, remote desktop analysis services allow for access to all software tools, both legacy and GUI-requiring. Together the two services complement each other, and cover the vast majority of use cases.

We have demonstrated prototypes for remote data analysis through the provision of software environments that can be accessed through Jupyter Notebooks and Remote Desktop Graphical interfaces. These provide a user interface (either the Jupyter notebook, or the GUI/UI of the machine to which the remote desktop is connected) with the provision of a software environment (either the container or environment in which the notebook executes, or the virtual machine to which the remote desktop connects). Topics not addressed in this deliverable are the authentication for these services, and data access. Authentication and data transfer are being implemented in WP6 and will be adopted by the services in WP4 as soon as it is operational.

In parallel, efforts are underway to offer remote analysis capabilities, such as remote Jupyter notebooks for EOSC through EGI. The EGI Jupyter service has been tested successfully during an online training for data analysis using X-ray Strain Orientation Calculation Software (<https://kmap.gitlab-pages.esrf.fr/xsocs/>) on 1st April 2019 at the HERCULES European school (<http://hercules-school.eu/>). Nine students participated, with trainers providing positive feedback and indicating interest in using the service again in the future.

The availability of remote data analysis services is rapidly becoming essential with the generalisation of remote experiments and remote access the new mode of operation in the post-COVID-19 phase.

[1] <https://confluence.panosc.eu/display/wp4/Common+Portal+Design>

[2] <https://confluence.panosc.eu/display/wp4/Common+Portal+Use+Cases>