Advances in HPC automation - An update on the use of Parsl in Parallel Works

Outline:

1. Parallel Works clusters
2. Parsl workflows (goals and stumbling blocks)
3. Parsl Jupyter notebooks

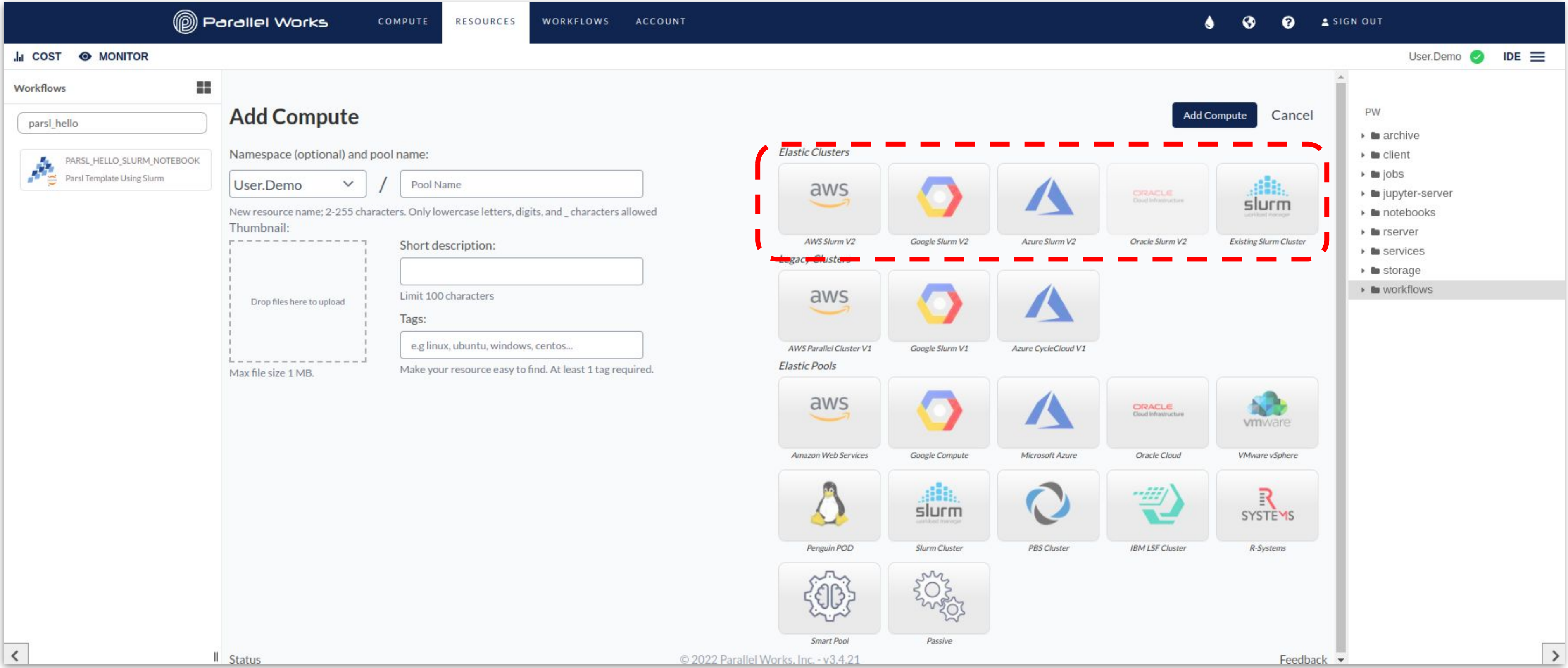
Parallel Works Clusters

Provision HPC SLURM clusters in the cloud:

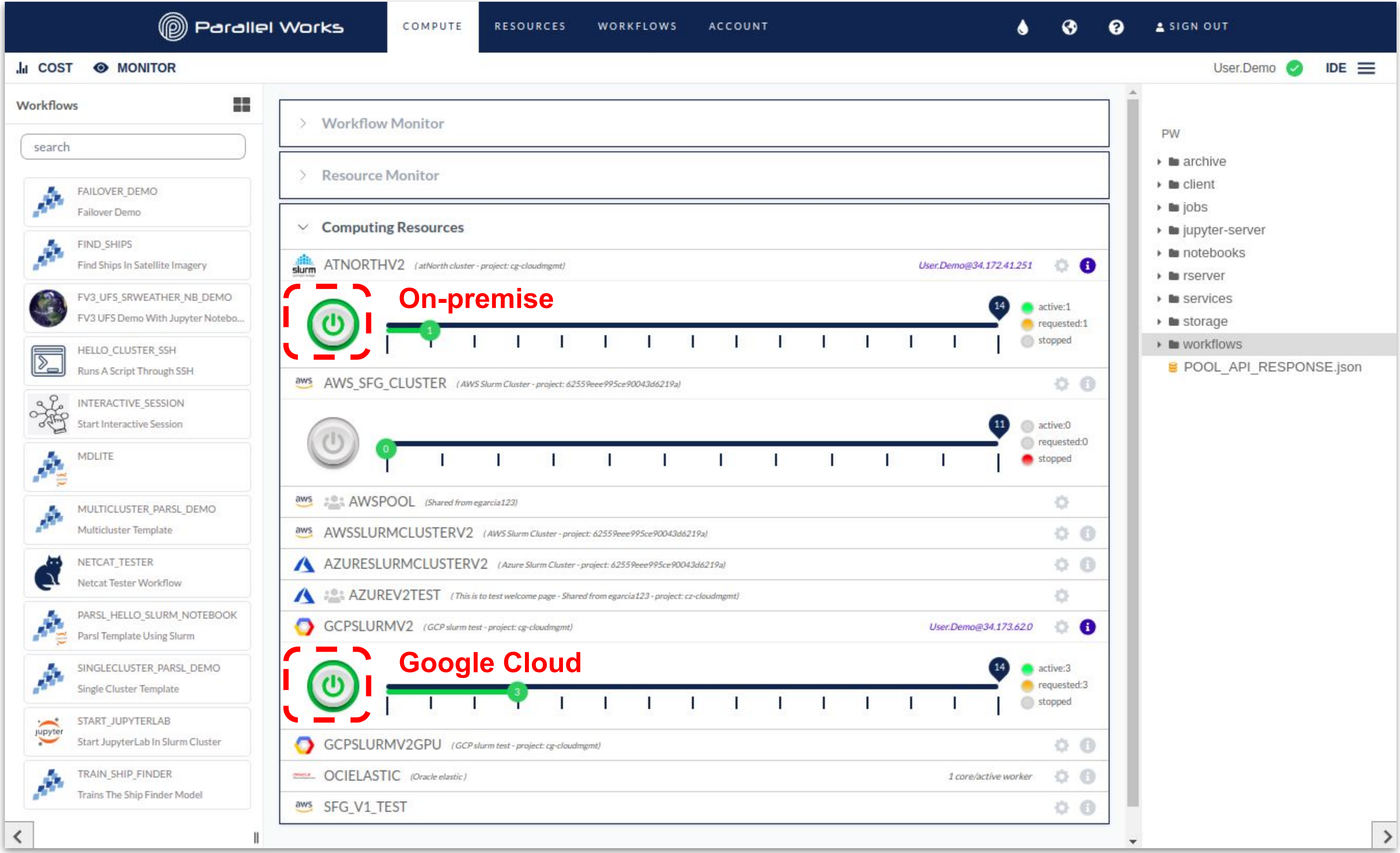
- Same “feel” & performance as on-premise SLURM clusters
- Elastic & highly customizable
- Leverage cloud’s cost: performance & new hardware
- Choice of several clouds

Connect an on-premise SLURM cluster

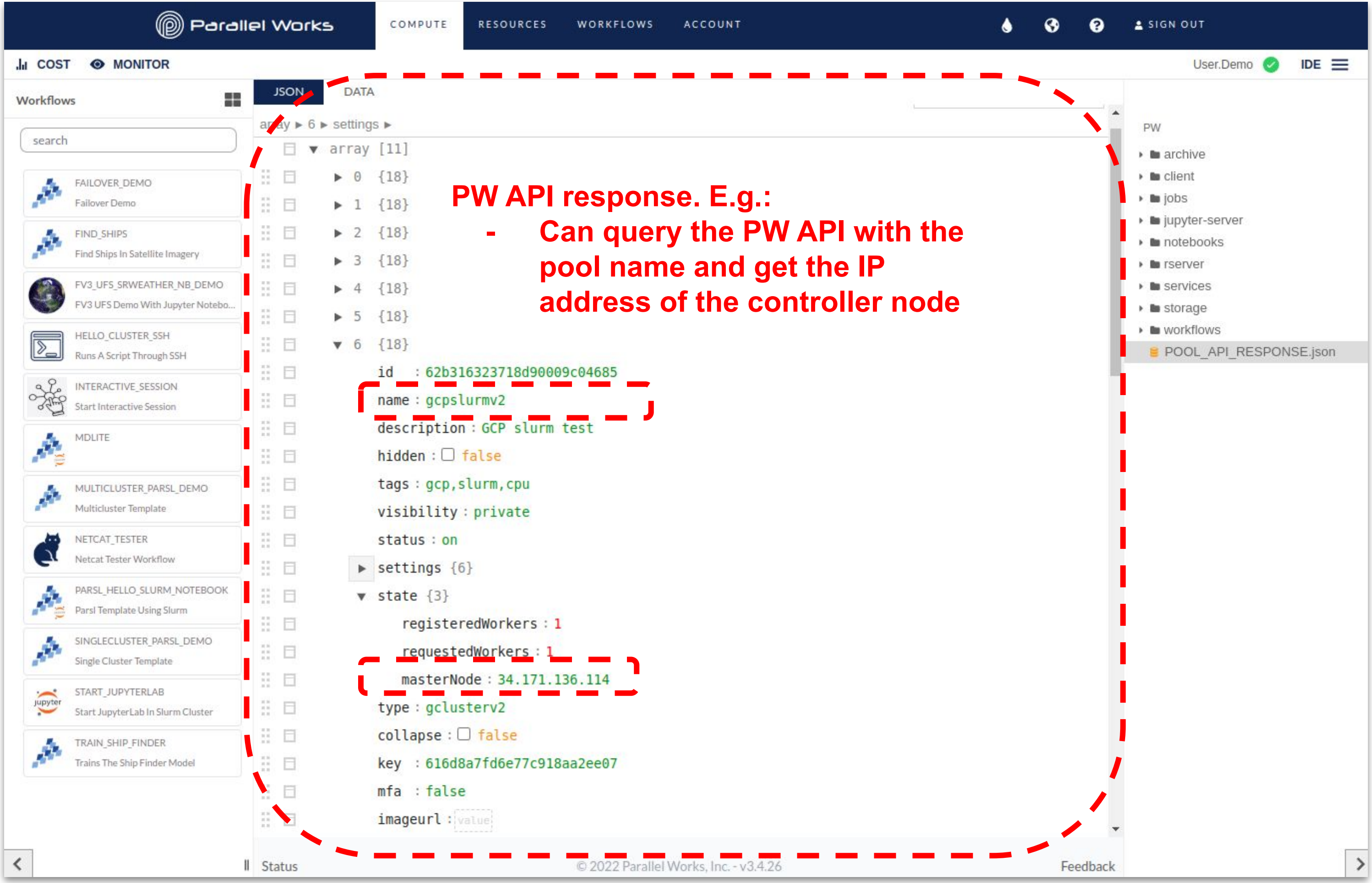
Uniform API
Parsl workflows and notebooks are started the same way in all clusters



To define a cluster simply click on the desired cluster type and fill in the configuration options



Users can activate these clusters with a power button in a uniform way



Uniform API for all clusters and SSH access from the user container to the controller (master) node of the cluster

How can we run Parsl in these clusters?

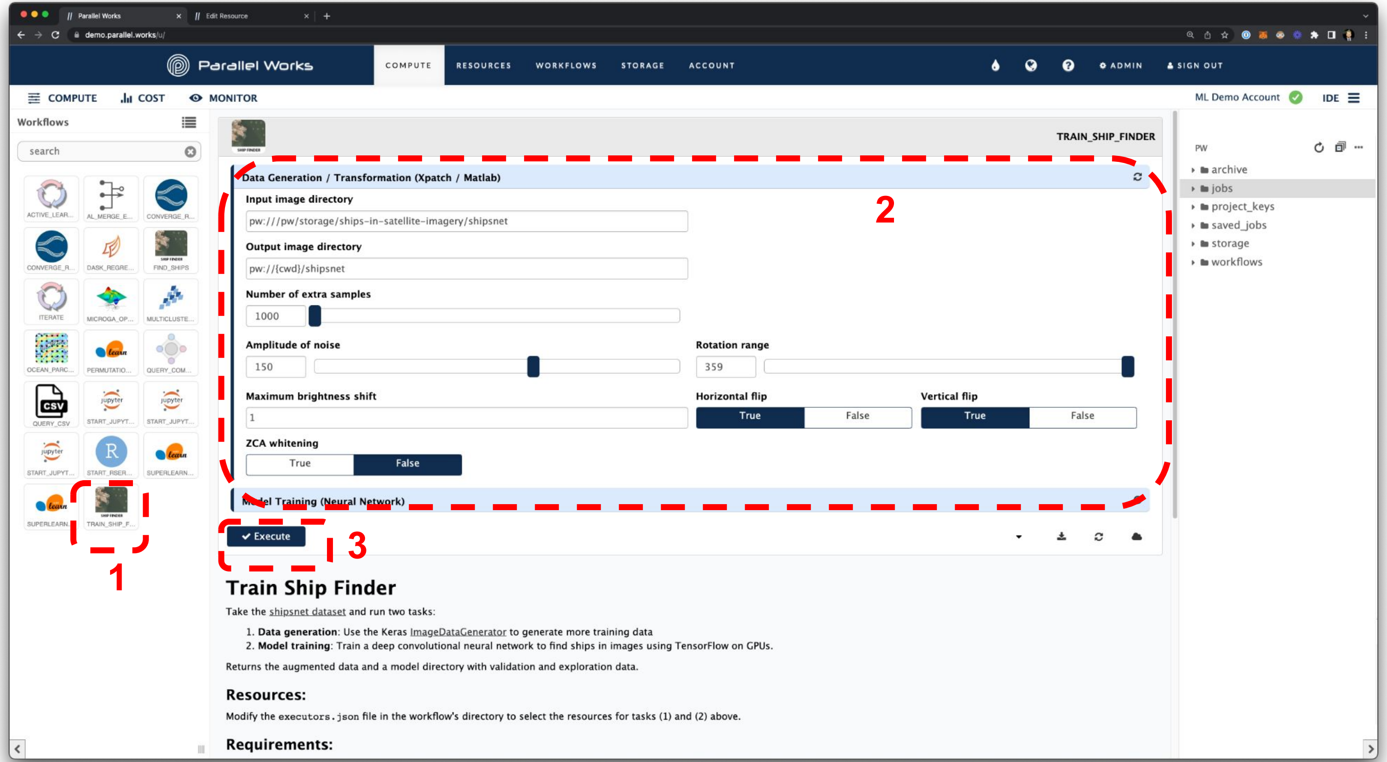
Workflows

Launch workflows in two ways:

1. Web user interface on PW
2. Python PW Client
    - CI/CD use case: GitHub action starts the job (with API key in repo secrets)

Launch workflow using the Web UI

1. Click on workflow thumbnail
2. Enter workflow parameters
3. Click execute → Generates the workflow command and arguments
4. Workflow command and arguments are executed in the user container



Launch workflow using the PW Client

1. Launch workflow with a Python script
2. Automation (e.g.: Github actions)

PW Client use case example

```
import sys
from client import Client
from client_functions import *

pw_user_host = sys.argv[1]
pw_api_key = sys.argv[2]
user = sys.argv[3]
resource_name = sys.argv[4]
wf_name = sys.argv[5]
wf_xml_args = json.loads(sys.argv[6])

c = Client('https://' + pw_user_host, pw_api_key)

start_resource(resource_name, c)
jid, djid = launch_workflow(wf_name, wf_xml_args, user, c)
```

Github action example

```
on: [push]

jobs:
  test-pw-workflow:
    runs-on: ubuntu-latest
    name: test-pw-workflow-beluga
    steps:
      - name: run-workflow-beluga
        id: run-beluga
        uses: parallelworks/test-workflow-action@v5
        with:
          pw-user-host: 'beluga.parallel.works'
          pw-api-key: ${{ secrets.ALVAROVIDALTO_BELUGA_API_KEY }}
          pw-user: 'alvarovidalto'
          resource-pool-names: 'gcpslurmv2'
          workflow-name: 'singlecluster_parsl_demo'
          workflow-parameters: '{"name": "PW_USER"}'
```

Goals for Parsl workflows:

- Moving from a custom modified Parsl to standard Parsl
- Parsl script runs in the user container in Parallel Works (not in the cluster)
- **Run different Parsl apps in different clusters (including on-premise and cloud)**
- Share Parsl workflows with other users

Stumbling blocks:

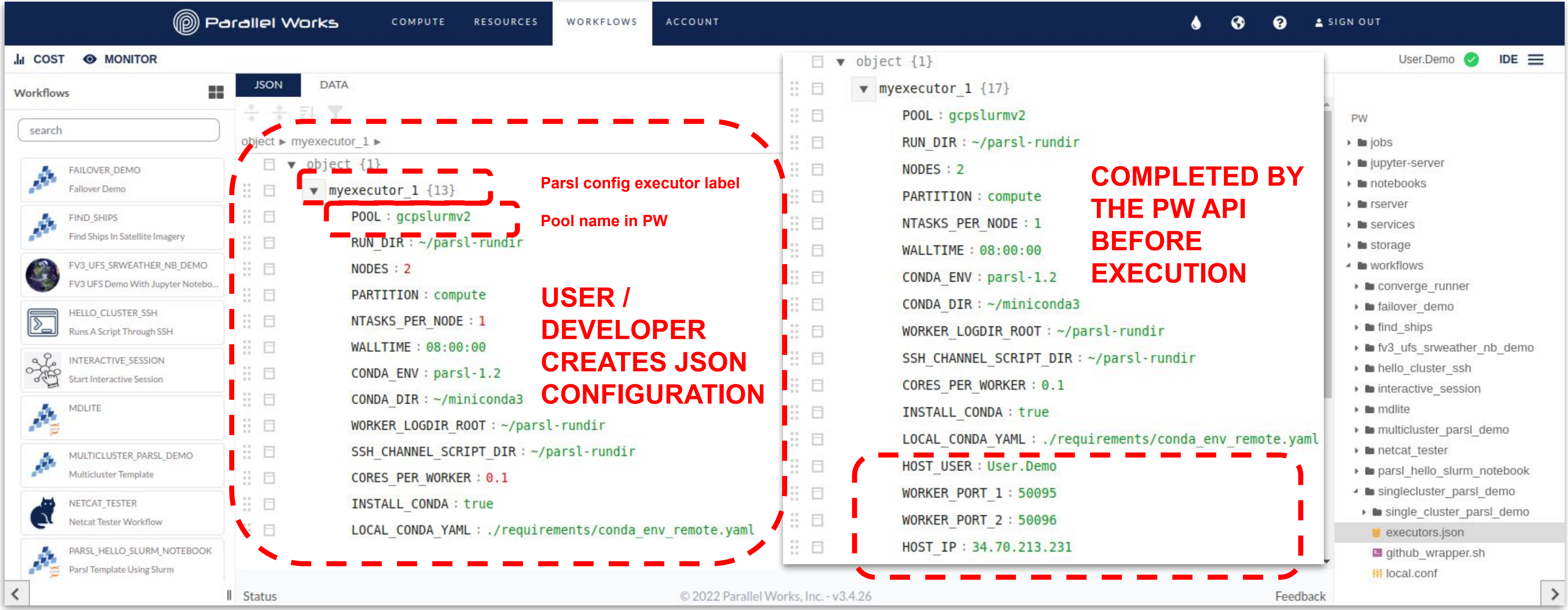
1. Define Parsl configuration for the different resources. Point to PW pools by pool name.
2. Manage python environment in the user container in PW and in the remote resources. Parsl version needs to be compatible. Dependencies.
    - Workflow may run in a different user container (shared) and/or in a different cluster
3. Establish port connections from the workers to the user container
    - User container does not have direct access to worker ports

Dealing with the stumbling blocks

↓

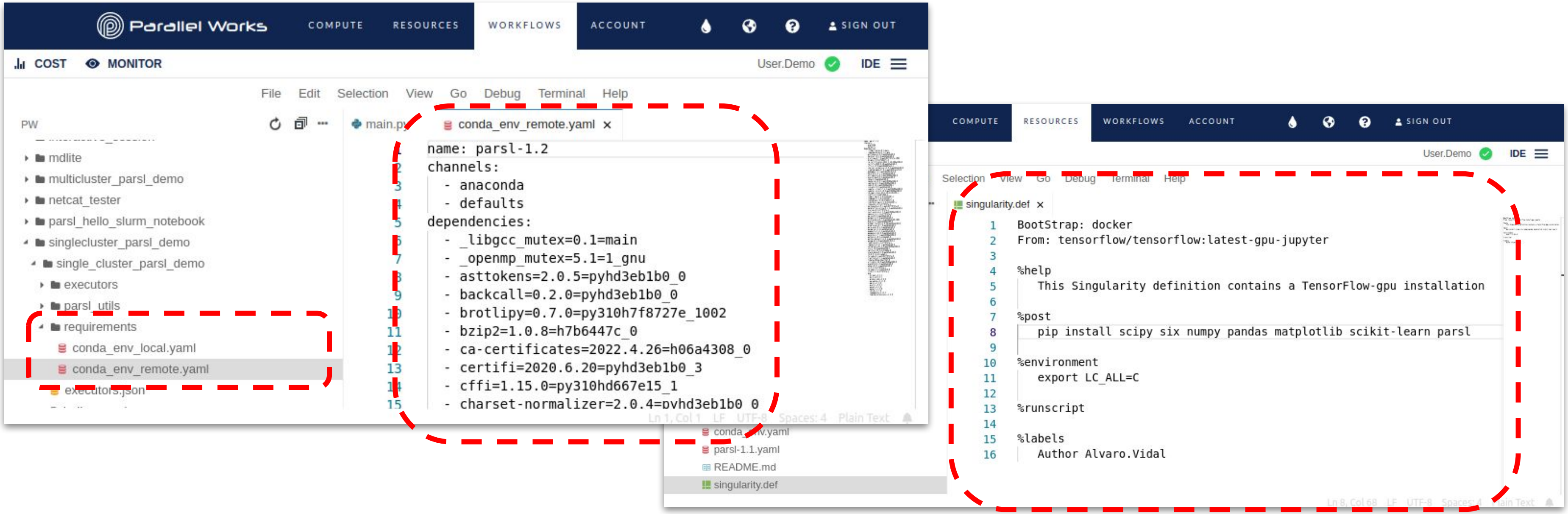
Parsl workflow wrapper

1. Define **Parsl configuration** definition for the different resources:
   - **JSON configuration file**
   - **PW API** to get pool information by pool name:
     - IP addresses and user name of the controller nodes
     - Available worker ports
   - **SSHChannel** to connect to the controller nodes
   - Run in **controller** nodes: **LocalProvider**
   - Run in **compute** nodes: **SlurmProvider** or **LocalProvider + bash_app + srun** (easier to reach ports)

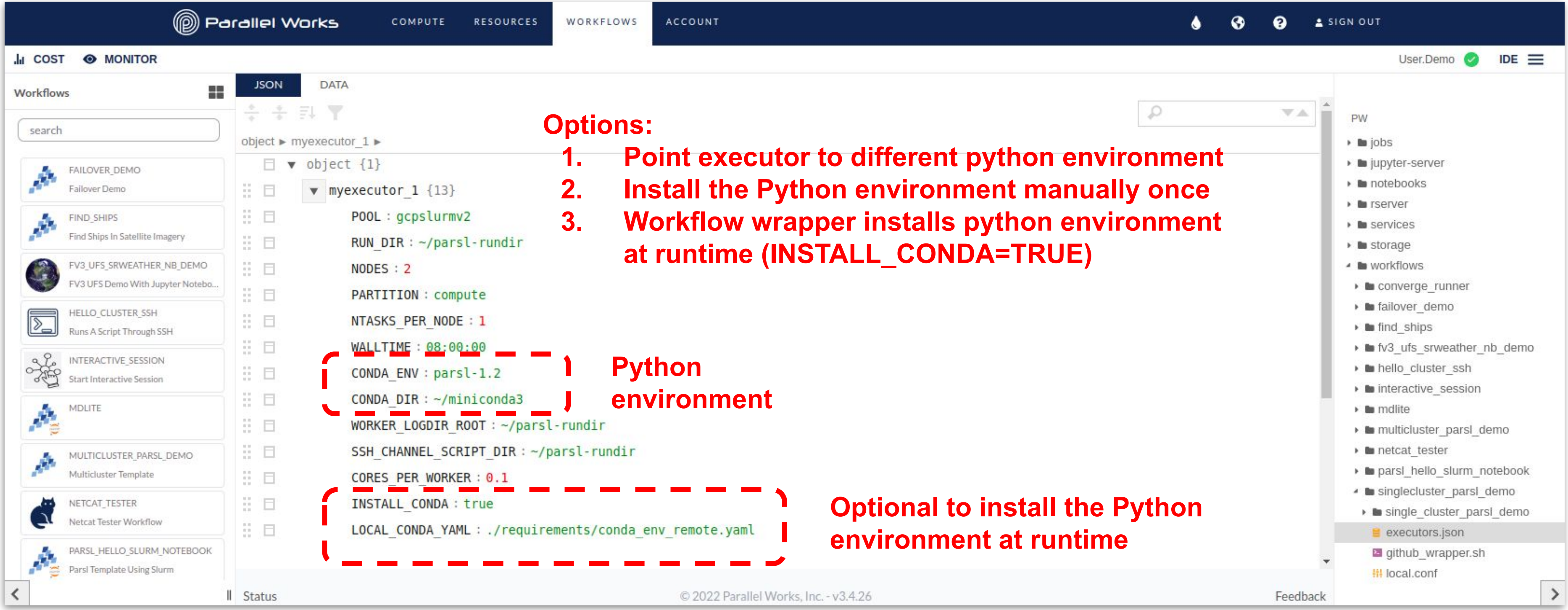


1. Define **Parsl configuration** definition for the different resources:
    - **JSON configuration file**
    - **PW API** to get pool information by pool name:
        - IP addresses and user name of the controller nodes
        - Available worker ports
    - **SSHChannel** to connect to the controller nodes
    - Run in **controller** nodes: **LocalProvider**
    - Run in **compute** nodes: **SlurmProvider** or **LocalProvider + bash_app + srun** (easier to reach ports)

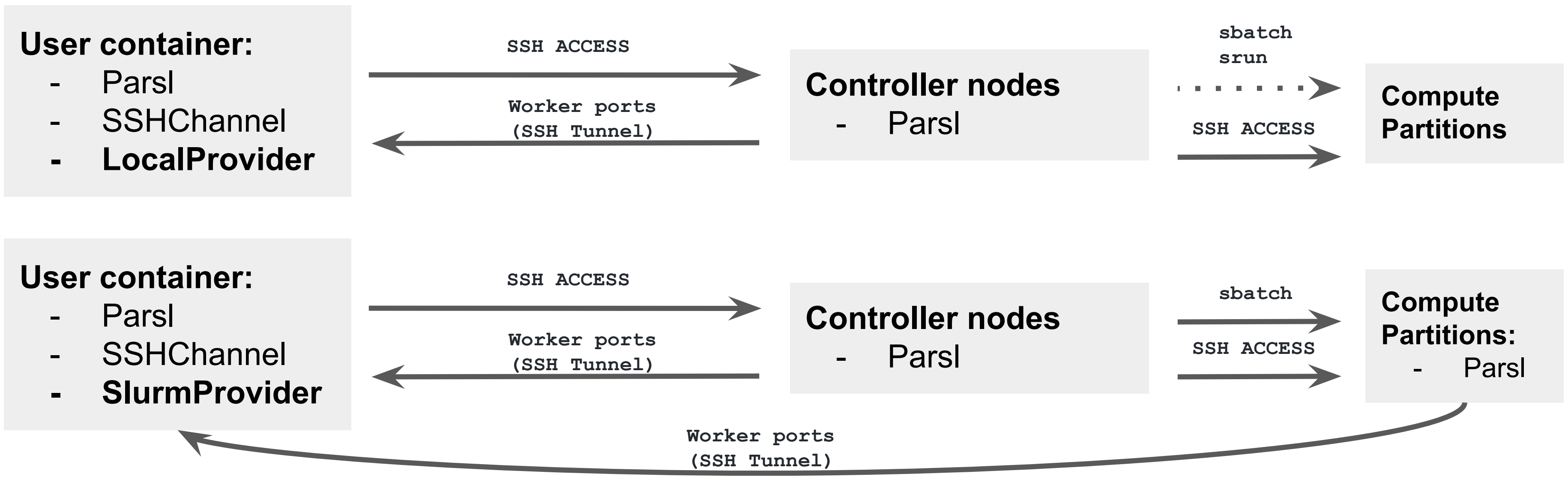
2. Manage **python environment** in the user container in PW and in the remote resources. Parsl version needs to be compatible. Dependencies.
   - Python environment is defined in **YAML or singularity definition files** (better for ML applications)
   - Can choose one per executor and another for the user container
   - Parsl workflow wrapper optionally updates/installs the Python environment from these files



2. Manage **python environment** in the user container in PW and in the remote resources. Parsl version needs to be compatible. Dependencies.
   - Python environment is defined in **YAML or singularity definition files** (better for ML applications)
   - Can choose one per executor and another for the user container
   - Parsl workflow wrapper optionally updates/installs the Python environment from these files



3. Establish **port connections for workers**
   - Parsl workflow wrapper creates SSH tunnels for the worker ports before execution and cleans them after execution
   - Available port numbers are provided by the PW API



Jupyter Notebooks

Goals:

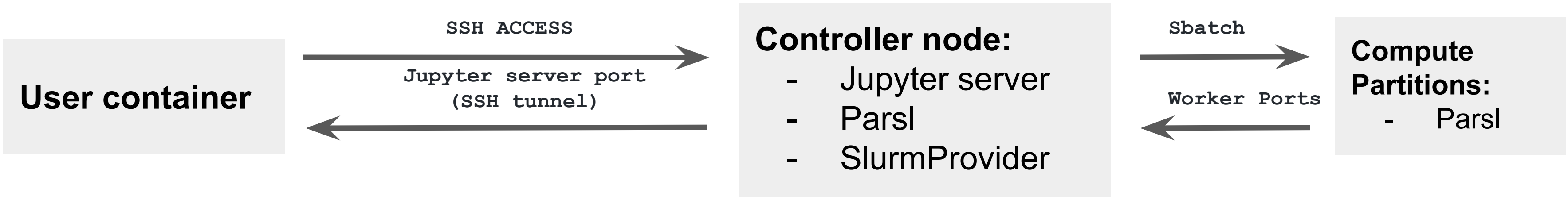
- Connect from the user container to the jupyter server
- Automate server launch

Approach:

- Jupyter server runs in the controller node of a slurm cluster
- Server port is forwarded to the user container → Only the server port is forwarded to the user container!

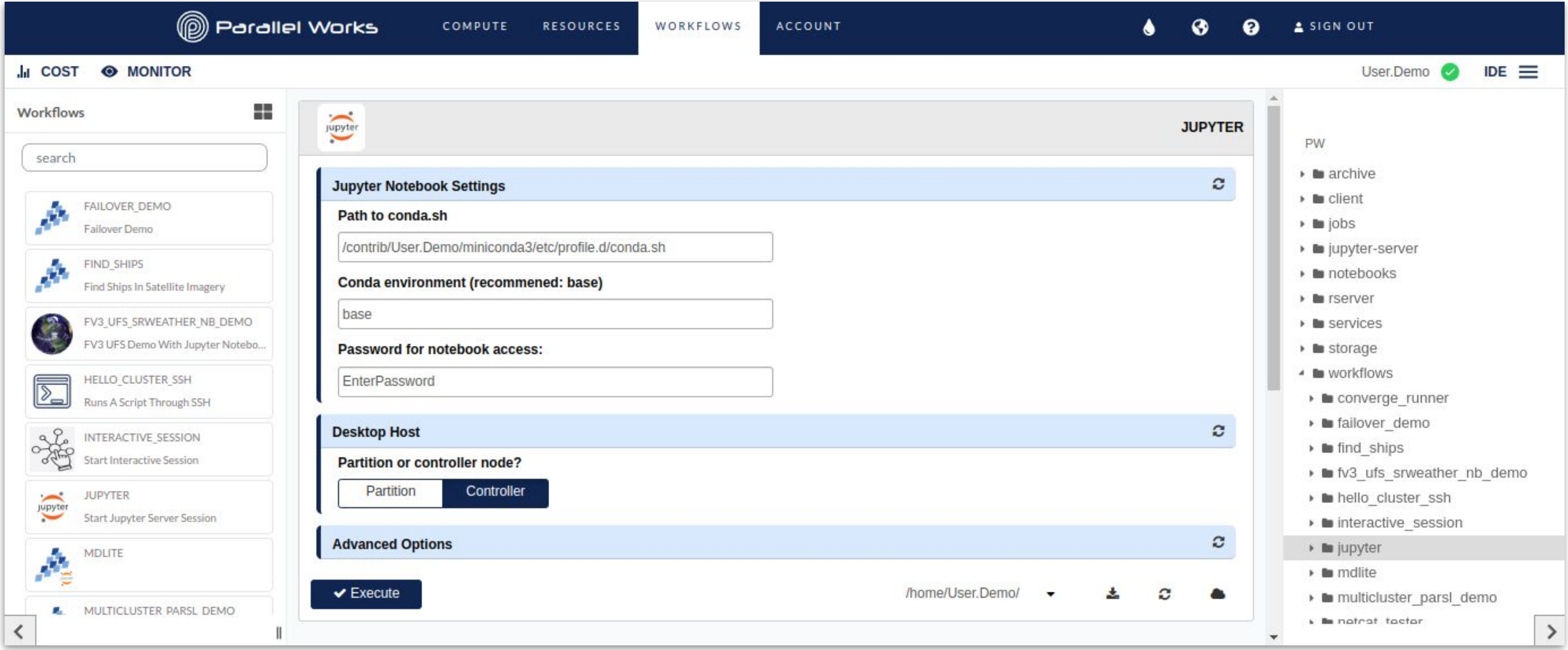
  Limitations:

  - Single cluster (multiple partitions) per Parsl job



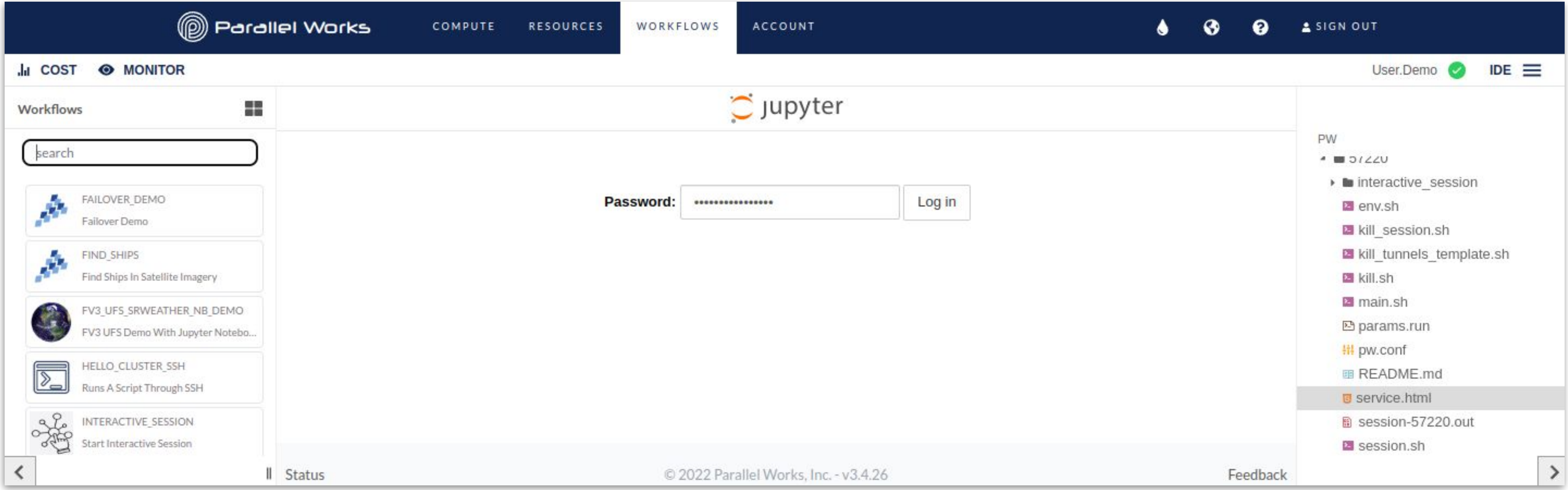
Approach:

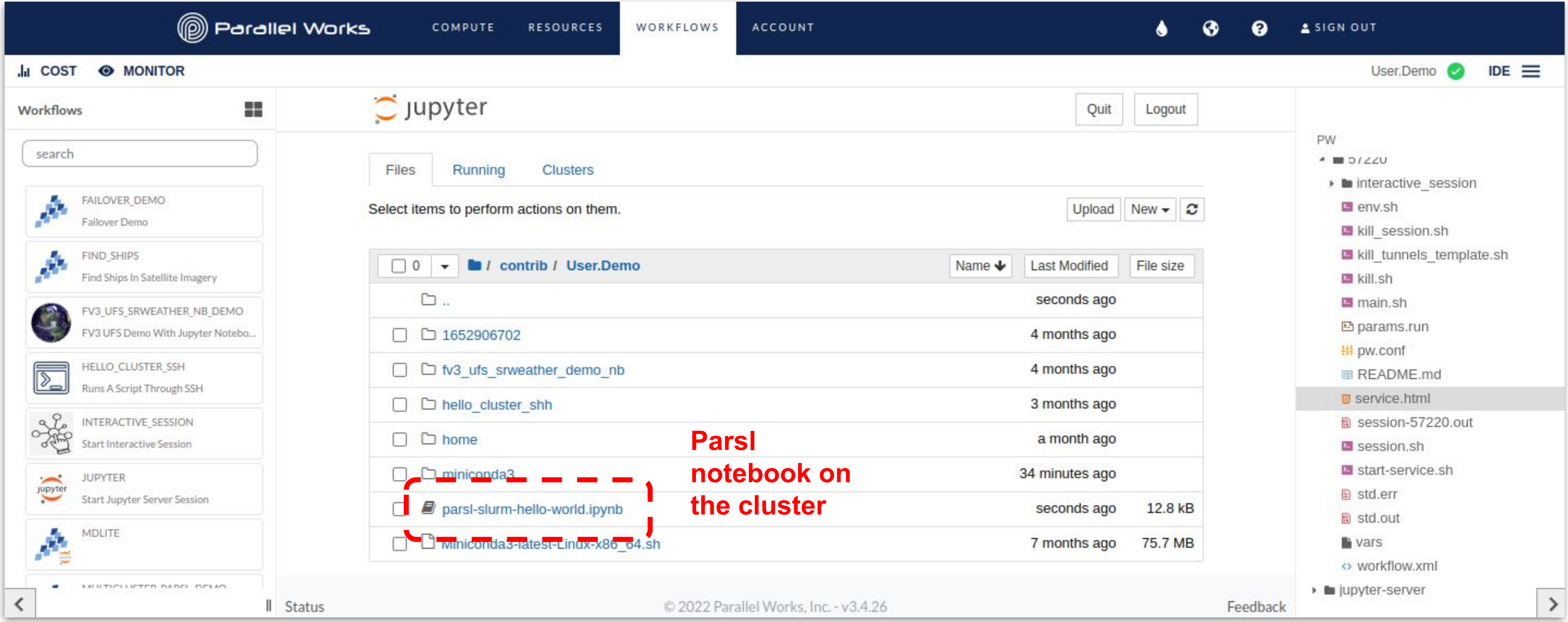
- For automation, the Jupyter server is started by a PW workflow



Approach:

- When the server is ready it pops up in the PW interface
- Enter your password and connect to the server




Parallel Works
COMPUTE
RESOURCES
WORKFLOWS
ACCOUNT
SIGN OUT
COST
MONITOR
User.Demo
IDE
Workflows
search
FAILOVER_DEMO
Failover Demo
FIND_SHIPS
Find Ships In Satellite Imagery
FV3_UFS_SRWEATHER_NB_DEMO
FV3 UFS Demo With Jupyter Notebo...
HELLO_CLUSTER_SSH
Runs A Script Through SSH
INTERACTIVE_SESSION
Start Interactive Session
JUPYTER
Start Jupyter Server Session
MDLITE
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ contrib / User.Demo
Name
Last Modified
File size
.. seconds ago
1652906702 4 months ago
fv3_ufs_srweather_demo_nb 4 months ago
hello_cluster_shh 3 months ago
home a month ago
miniconda3 34 minutes ago
parsl-slurm-hello-world.ipynb seconds ago 12.8 kB
Miniconda3-latest-Linux-x86_64.sh 7 months ago 75.7 MB
Parsl notebook on the cluster
PW
interactive_session
env.sh
kill_session.sh
kill_tunnels_template.sh
kill.sh
main.sh
params.run
pw.conf
README.md
service.html
session-57220.out
session.sh
start-service.sh
std.err
std.out
vars
workflow.xml
jupyter-server
Status
© 2022 Parallel Works, Inc. - v3.4.26
Feedback

Approach:

- Send jobs to different partitions using the SlurmProvider

https://beta.parallel.works/pwide-kube-nb/10.88.0.4/50104/notebooks/contrib/User.Demo/parsl-slurm-hello-world.ipynb

jupyter parsl-slurm-hello-world (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run | Markdown

Parsl Slurm Notebook Hello World

In [3]:

```
# Print environment information
import socket, sys, os, json
print('Hostname: ', socket.gethostname())
```

```
Hostname:  mgmt-userdemo-gcpslurmv2-00108
```

In [4]:

```
%%bash
sinfo
squeue
```

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      9  idle~ userdemo-gcpslurmv2-00108-1-[0002-0010]
compute*     up   infinite      1  alloc userdemo-gcpslurmv2-00108-1-0001
gpu          up   infinite      2  idle~ userdemo-gcpslurmv2-00108-2-[0001-0002]
fail         up   infinite      1  idle~ userdemo-gcpslurmv2-00108-3-0001
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                 3   compute session- User.Dem  R      16:35      1 userdemo-gcpslurmv2-00108-1-0001
```

Define the Parsl configuration

In [2]:

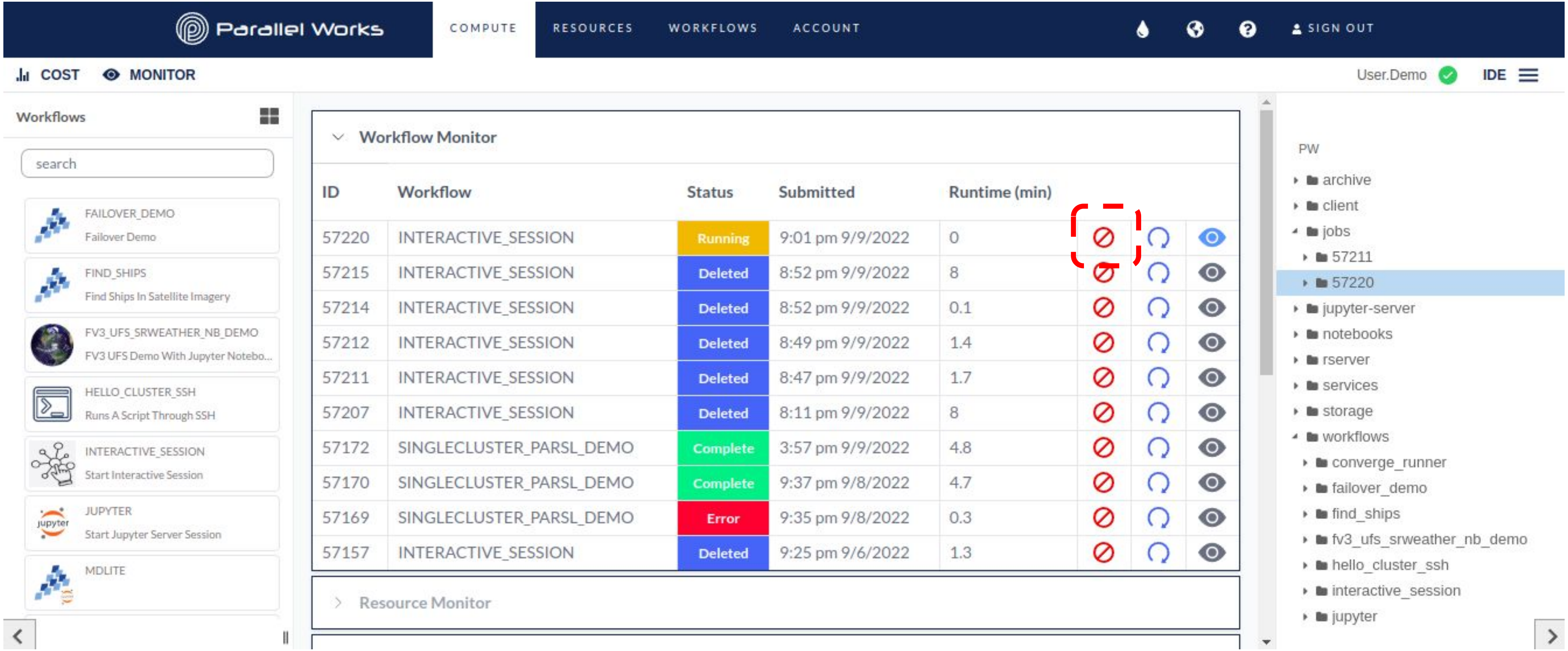
```
import parsl
print(parsl.__version__)
from parsl.config import Config

from parsl.executors import HighThroughputExecutor
from parsl.providers import SlurmProvider

parsl.clear()
# Configuration variable tells Parsl where to run the python functions
# -- Depends the resource and tasks to run
config = Config(
        executors=[
            HighThroughputExecutor(
                label = 'compute',
                worker_debug = True,             # Default False for shorter logs
                cores_per_worker= int(2),      # One worker per node
                worker_logdir_root = os.getcwd() + '/parsllogs',
                provider = SlurmProvider(
                    #========GPU RUNS============
                    #scheduler_options = '#SBATCH --gres=gpu:1', # For GPU runs!
                    #========CPU RUNS============
                    #scheduler_options = '#SBATCH --ntasks-per-node=40',  # DO NOT USE! Conflicts with cores_per
                    partition = 'compute',          # Cluster specific! Needs to match GPU availability, and RAM
                    #===========================
```

Approach:

- Kill jupyter server job



Potential next steps

- Implement failover in Parsl workflows
  - Associate multiple resources with a given Parsl app
  - Resources are ranked; if #1 fails, try #2…
- Streamline the definition of the Parsl configuration through the web UI instead of editing the JSON file

Thank You!