

# 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

Workflows



parsl\_hello



## Add Compute

Namespace (optional) and pool name:

User.Demo

/

Pool Name

New resource name; 2-255 characters. Only lowercase letters, digits, and \_ characters allowed  
Thumbnail:

Max file size 1 MB.

Short description:

Limit 100 characters

Tags:

e.g linux, ubuntu, windows, centos...

Make your resource easy to find. At least 1 tag required.

Add Compute

Cancel

## Elastic Clusters



AWS Slurm V2



Google Slurm V2



Azure Slurm V2



Oracle Slurm V2



Existing Slurm Cluster

## Elastic Pools



AWS Parallel Cluster V1



Google Slurm V1



Azure CycleCloud V1

## Elastic Pools



Amazon Web Services



Google Compute



Microsoft Azure



Oracle Cloud



VMware vSphere



Penguin POD



Slurm Cluster



PBS Cluster



IBM LSF Cluster



R-Systems



Smart Pool



Passive

PW

- archive
- client
- jobs
- jupyter-server
- notebooks
- rserver
- services
- storage
- workflows

Status

© 2022 Parallel Works, Inc. - v3.4.21

Feedback

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

Parallel Works COMPUTE RESOURCES WORKFLOWS ACCOUNT

Cost MONITOR User.Demo ✓ IDE

Workflows

- 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
- MDLITE
- MULTICLUSTER\_PARSL\_DEMO: Multicuster Template
- NETCAT\_TESTER: Netcat Tester Workflow
- PARSL\_HELLO\_SLURM\_NOTEBOOK: Parsl Template Using Slurm
- SINGLECLUSTER\_PARSL\_DEMO: Single Cluster Template
- START\_JUPYTERLAB: Start JupyterLab In Slurm Cluster
- TRAIN\_SHIP\_FINDER: Trains The Ship Finder Model

Workflow Monitor

Resource Monitor

Computing Resources

- On-premise**
  - ATNORTHV2 (atNorth cluster - project: cg-cloudmgmt) User.Demo@34.172.41.251
    - active: 1
    - requested: 1
    - stopped: 0
- AWS\_SFG\_CLUSTER (AWS Slurm Cluster - project: 62559eee995ce90043d6219a)
  - active: 0
  - requested: 0
  - stopped: 0
- AWSPPOOL (Shared from egarcia123)
- AWSSLURMCLUSTERV2 (AWS Slurm Cluster - project: 62559eee995ce90043d6219a)
- AZURESLURMCLUSTERV2 (Azure Slurm Cluster - project: 62559eee995ce90043d6219a)
- AZUREV2TEST (This is to test welcome page - Shared from egarcia123 - project: cz-cloudmgmt)
- GCPSLURMV2 (GCP slurm test - project: cg-cloudmgmt) User.Demo@34.173.62.0
  - active: 3
  - requested: 3
  - stopped: 0
- GCPSLURMV2GPU (GCP slurm test - project: cg-cloudmgmt)
- OCIELASTIC (Oracle elastic) 1 core/active worker
- SFG\_V1\_TEST

**Google Cloud**

PW

- archive
- client
- jobs
- jupyter-server
- notebooks
- rserver
- services
- storage
- workflows
  - POOL\_API\_RESPONSE.json

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

**PW API response. E.g.:**

- Can query the PW API with the pool name and get the IP address of the controller node

```
array [11]
  0 {18}
  1 {18}
  2 {18}
  3 {18}
  4 {18}
  5 {18}
  6 {18}
    id : 62b316323718d90009c04685
    name : gcpslurmv2
    description : GCP slurm test
    hidden :  false
    tags : gcp,slurm,cpu
    visibility : private
    status : on
    settings {6}
    state {3}
      registeredWorkers : 1
      requestedWorkers : 1
      masterNode : 34.171.136.114
    type : gclusterv2
    collapse :  false
    key : 616d8a7fd6e77c918aa2ee07
    mfa : false
    imageUrl : value
```

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

The screenshot displays the Parallel Works web interface. The top navigation bar includes 'COMPUTE', 'RESOURCES', 'WORKFLOWS', 'STORAGE', and 'ACCOUNT'. The main content area is titled 'Train Ship Finder' and is divided into two sections: 'Data Generation / Transformation (Xpatch / Matlab)' and 'Model Training (Neural Network)'. The 'Data Generation' section contains several input fields and sliders for parameters such as 'Input image directory', 'Output image directory', 'Number of extra samples', 'Amplitude of noise', 'Rotation range', 'Maximum brightness shift', 'Horizontal flip', and 'Vertical flip'. The 'Model Training' section includes a 'ZCA whitening' checkbox and an 'Execute' button. A sidebar on the left shows a grid of workflow thumbnails, with the 'Train Ship Finder' thumbnail highlighted by a red dashed box and the number '1'. The main configuration area is also highlighted by a red dashed box and the number '2'. The 'Execute' button is highlighted by a red dashed box and the number '3'. The right-hand side of the interface shows a file explorer view with a tree structure including 'archive', 'jobs', 'project\_keys', 'saved\_jobs', 'storage', and 'workflows'.

## Launch workflow using the PW Client

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

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

### PW Client use case example

```
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)
```

```
on: [push]
```

### Github action example

```
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: 'gcpslurm2'
          workflow-name: 'singlecluster_parsl_demo'
          workflow-parameters: '{"name": "PW_USER"}'
```

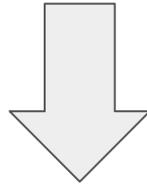
## Goals for Parsl workflows:

- 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

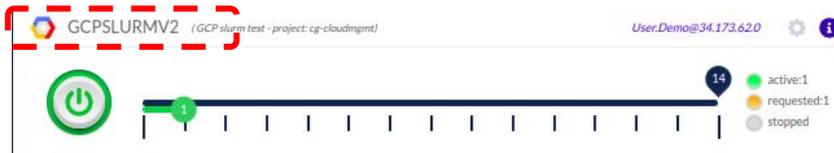
# Dealing with the stumbling blocks



## ParSI 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)



**Parsl config executor label**

**Pool name in PW**

**USER / DEVELOPER CREATES JSON CONFIGURATION**

**COMPLETED BY THE PW API BEFORE EXECUTION**

```
object {1}
  myexecutor_1 {13}
    POOL : gcpslurm2
    RUN_DIR : ~/parsl-rundir
    NODES : 2
    PARTITION : compute
    NTASKS_PER_NODE : 1
    WALLTIME : 08:00:00
    CONDA_ENV : parsl-1.2
    CONDA_DIR : ~/miniconda3
    WORKER_LOGDIR_ROOT : ~/parsl-rundir
    SSH_CHANNEL_SCRIPT_DIR : ~/parsl-rundir
    CORES_PER_WORKER : 0.1
    INSTALL_CONDA : true
    LOCAL_CONDA_YAML : ./requirements/conda_env_remote.yaml
    HOST_USER : User.Demo
    WORKER_PORT_1 : 50095
    WORKER_PORT_2 : 50096
    HOST_IP : 34.70.213.231
```

# 1. Define Parsl configuration definition for the different resources:

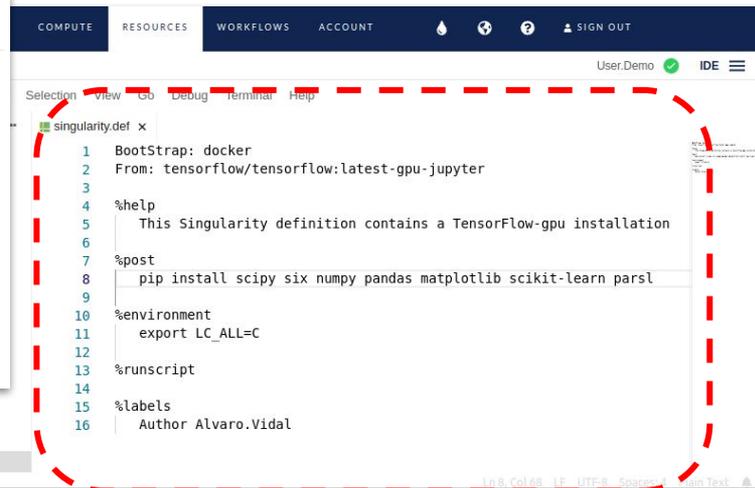
- **JSON configuration file**
- **PW API** to get pool information by pool name:
  - IP addresses and user name of the controller nodes
  - Available worker ports
- **SSHChannel** to connect to the controller nodes
- Run in **controller** nodes: **LocalProvider**
- Run in **compute** nodes: **SlurmProvider** or **LocalProvider + bash\_app + srun** (easier to reach ports)

**JSON IS LOADED AND USED TO DEFINE THE PARSL CONFIGURATION**

```
110 config = Config(
111     executors = [
112         HighThroughputExecutor(
113             worker_ports = ((int(exec_conf['myexecutor_1']['WORKER_PORT_1']), int(exec_conf['myexecutor_1']['WORKER_PORT_2'])),
114             label = 'myexecutor_1',
115             worker_debug = True, # default False for shorter logs
116             cores_per_worker = float(exec_conf['myexecutor_1']['CORES_PER_WORKER']), # One worker per node
117             worker_logdir_root = exec_conf['myexecutor_1']['WORKER_LOGDIR_ROOT'], # os.getcwd() + '/parslloas'
118             provider = LocalProvider(
119                 worker_init = 'source {conda_sh}; conda activate {conda_env}; cd {run_dir}'.format(
120                     conda_sh = os.path.join(exec_conf['myexecutor_1']['CONDA_DIR'], 'etc/profile.d/conda.sh'),
121                     conda_env = exec_conf['myexecutor_1']['CONDA_ENV'],
122                     run_dir = exec_conf['myexecutor_1']['RUN_DIR']
123                 ),
124                 channel = SSHChannel(
125                     hostname = exec_conf['myexecutor_1']['HOST_IP'],
126                     username = exec_conf['myexecutor_1']['HOST_USER'],
127                     script_dir = exec_conf['myexecutor_1']['SSH_CHANNEL_SCRIPT_DIR'], # Full path to a script dir
128                     key_filename = '/home/{PW_USER}/.ssh/pw_id_rsa'.format(PW_USER = os.environ['PW_USER'])
129                 )
130             )
131         ],
132     monitoring = MonitoringHub(
133         hub_address = address_by_hostname(),
134         resource_monitoring_interval = 5
135     )
136 )
137
138 print('Loading Parsl Config', flush = True)
139 parsl.load(config)
```

```
objects (1)
  myexecutor_1 (17)
    pool: gpsstcmv2
    RUN_DIR: ~/parsl-rundir
    NODES: 2
    PARTITION: compute
    NTASKS_PER_NODE: 1
    WALLTIME: 08:00:00
    CONDA_ENV: parsl-1.2
    CONDA_DIR: ~/miniconda3
    WORKER_LOGDIR_ROOT: ~/parsl-rundir
    SSH_CHANNEL_SCRIPT_DIR: ~/parsl-rundir
    CORES_PER_WORKER: 0.1
    INSTALL_CONDA: true
    LOCAL_CONDA_YAML: ./requirements/conda_env_remote.yaml
    HOST_USER: User.Demo
    WORKER_PORT_1: 50095
    WORKER_PORT_2: 50096
    HOST_IP: 34.70.213.231
```

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

The screenshot shows the Parallel Works web interface. The top navigation bar includes 'COMPUTE', 'RESOURCES', 'WORKFLOWS', and 'ACCOUNT'. The main content area displays a workflow configuration for 'myexecutor\_1'. The configuration is shown in a tree view under 'JSON' and 'DATA' tabs. The configuration includes various parameters such as 'POOL', 'RUN\_DIR', 'NODES', 'PARTITION', 'NTASKS\_PER\_NODE', 'WALLTIME', 'CONDA\_ENV', 'CONDA\_DIR', 'WORKER\_LOGDIR\_ROOT', 'SSH\_CHANNEL\_SCRIPT\_DIR', 'CORES\_PER\_WORKER', 'INSTALL\_CONDA', and 'LOCAL\_CONDA\_YAML'. Red dashed boxes highlight the 'CONDA\_ENV' and 'INSTALL\_CONDA' sections. Red text annotations provide options for configuring the Python environment.

**Options:**

1. Point executor to different python environment
2. Install the Python environment manually once
3. Workflow wrapper installs python environment at runtime (**INSTALL\_CONDA=TRUE**)

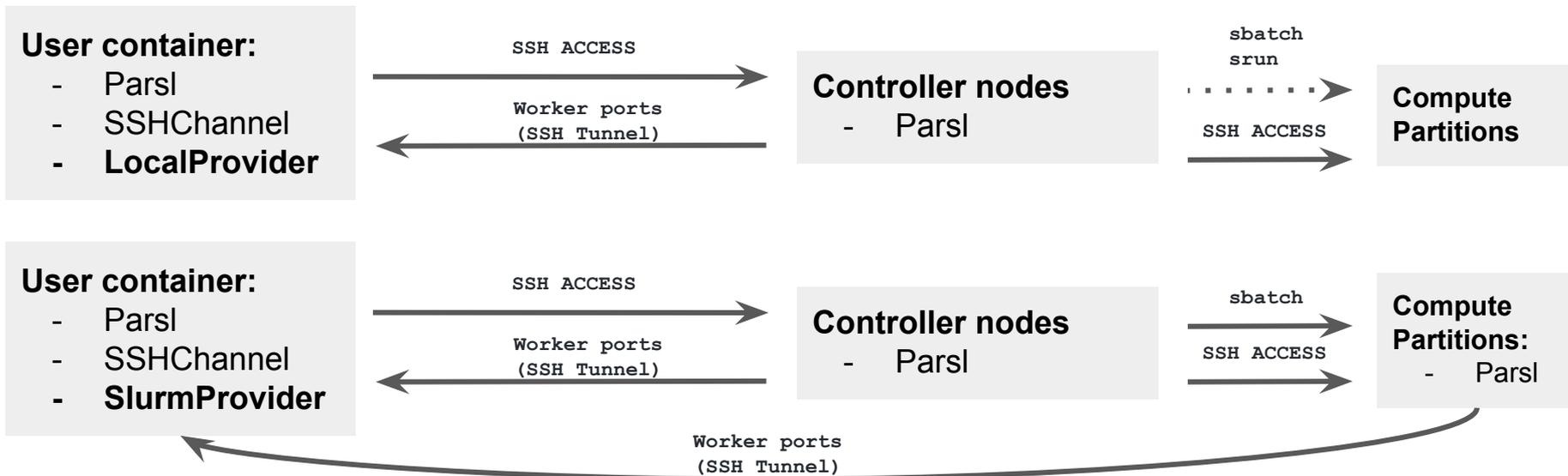
**Python environment**

**Optional to install the Python environment at runtime**

The interface also shows a sidebar with 'Workflows' and 'PW' sections. The 'PW' section lists various workflows and executors, including 'jobs', 'jupyter-server', 'notebooks', 'rserver', 'services', 'storage', 'workflows', 'converge\_runner', 'failover\_demo', 'find\_ships', 'fv3\_ufs\_srweather\_nb\_demo', 'hello\_cluster\_ssh', 'interactive\_session', 'mdlite', 'multiclust\_parsl\_demo', 'netcat\_tester', 'parsl\_hello\_slurm\_notebook', 'singlecluster\_parsl\_demo', 'single\_cluster\_parsl\_demo', 'executors.json', 'github\_wrapper.sh', and 'local.conf'.

### 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:

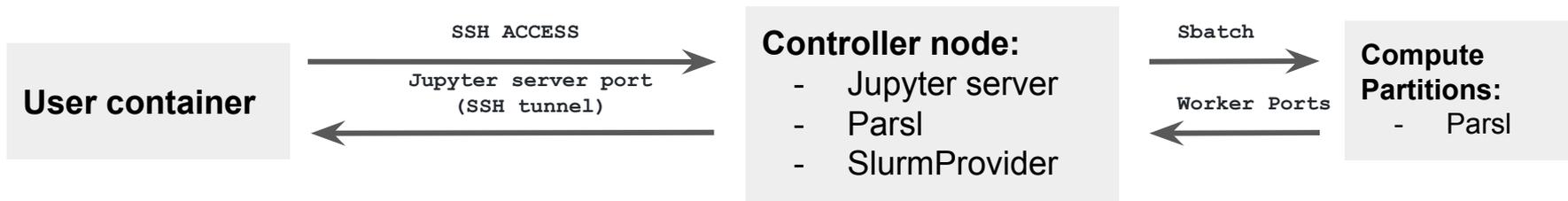
- Connect from the user container to the jupyter server
- Automate server launch

## 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

The screenshot displays the Parallel Works web interface. The top navigation bar includes 'COMPUTE', 'RESOURCES', 'WORKFLOWS', and 'ACCOUNT'. The user is logged in as 'User.Demo'. The main content area is titled 'JUPYTER' and shows the configuration for a Jupyter Notebook session. The configuration is organized into sections: 'Jupyter Notebook Settings', 'Desktop Host', and 'Advanced Options'. The 'Jupyter Notebook Settings' section includes fields for 'Path to conda.sh' (set to '/contrib/User.Demo/miniconda3/etc/profile.d/conda.sh'), 'Conda environment (recommended: base)' (set to 'base'), and 'Password for notebook access:' (set to 'EnterPassword'). The 'Desktop Host' section has a 'Partition or controller node?' dropdown menu with 'Partition' and 'Controller' options. The 'Advanced Options' section is currently empty. At the bottom of the configuration area, there is an 'Execute' button and a file path '/home/User.Demo/'. On the right side, a file explorer shows a directory structure under 'PW', with 'jupyter' selected. The left sidebar shows a list of workflows, including 'FAILOVER\_DEMO', 'FIND\_SHIPS', 'FV3\_UFS\_SRWEATHER\_NB\_DEMO', 'HELLO\_CLUSTER\_SSH', 'INTERACTIVE\_SESSION', 'JUPYTER', 'MDLITE', and 'MULTICLUSTER PARSL DEMO'.

**Parallel Works** COMPUTE RESOURCES WORKFLOWS ACCOUNT

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
- MULTICLUSTER PARSL DEMO

**JUPYTER**

**Jupyter Notebook Settings**

**Path to conda.sh**

/contrib/User.Demo/miniconda3/etc/profile.d/conda.sh

**Conda environment (recommended: base)**

base

**Password for notebook access:**

EnterPassword

**Desktop Host**

**Partition or controller node?**

Partition Controller

**Advanced Options**

✓ Execute /home/User.Demo/

PW

- archive
- client
- jobs
- jupyter-server
- notebooks
- rserver
- services
- storage
- workflows
  - converge\_runner
  - failover\_demo
  - find\_ships
  - fv3\_ufs\_srweather\_nb\_demo
  - hello\_cluster\_ssh
  - interactive\_session
  - jupyter**
  - mdlite
  - multiclust\_parsl\_demo
  - netrat\_tector

## Approach:

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

The screenshot displays the Parallel Works web interface. At the top, a dark blue navigation bar contains the Parallel Works logo and menu items: COMPUTE, RESOURCES, WORKFLOWS, and ACCOUNT. On the right side of this bar are icons for a water drop, a globe, a question mark, and a 'SIGN OUT' button. Below the navigation bar, the main content area is divided into three sections. On the left, a 'Workflows' sidebar features a search bar and a list of workflow cards: '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), and 'INTERACTIVE\_SESSION' (Start Interactive Session). The central area is a Jupyter interface with the 'jupyter' logo at the top. In the center of this area is a login form with the label 'Password:' followed by a text input field containing eight dots and a 'Log in' button. On the right side of the Jupyter interface, a file explorer shows the directory structure for an 'interactive\_session'. The files listed are: 'env.sh', 'kill\_session.sh', 'kill\_tunnels\_template.sh', 'kill.sh', 'main.sh', 'params.run', 'pw.conf', 'README.md', 'service.html' (highlighted), 'session-57220.out', and 'session.sh'. At the bottom of the interface, a light blue footer bar contains the text 'Status', '© 2022 Parallel Works, Inc. - v3.4.26', and 'Feedback'.



## Workflows



search

FAILOVER\_DEMO  
Failover DemoFIND\_SHIPS  
Find Ships In Satellite ImageryFV3\_UFS\_SRWEATHER\_NB\_DEMO  
FV3 UFS Demo With Jupyter Notebo...HELLO\_CLUSTER\_SSH  
Runs A Script Through SSHINTERACTIVE\_SESSION  
Start Interactive SessionJUPYTER  
Start Jupyter Server Session

MDLITE



Quit

Logout

Files Running Clusters

Select items to perform actions on them.

Upload

New



<input type="checkbox"/>	0		Name	Last Modified	File size
			..	seconds ago	
<input type="checkbox"/>		1652906702		4 months ago	
<input type="checkbox"/>		fv3_ufs_srweather_demo_nb		4 months ago	
<input type="checkbox"/>		hello_cluster_ssh		3 months ago	
<input type="checkbox"/>		home		a month ago	
<input type="checkbox"/>		miniconda3		34 minutes ago	
<input type="checkbox"/>		<b>parsi-slurm-hello-world.ipynb</b>		seconds ago	12.8 kB
<input type="checkbox"/>		minicondas-latest-Linux-x86_64.sh		7 months ago	75.7 MB

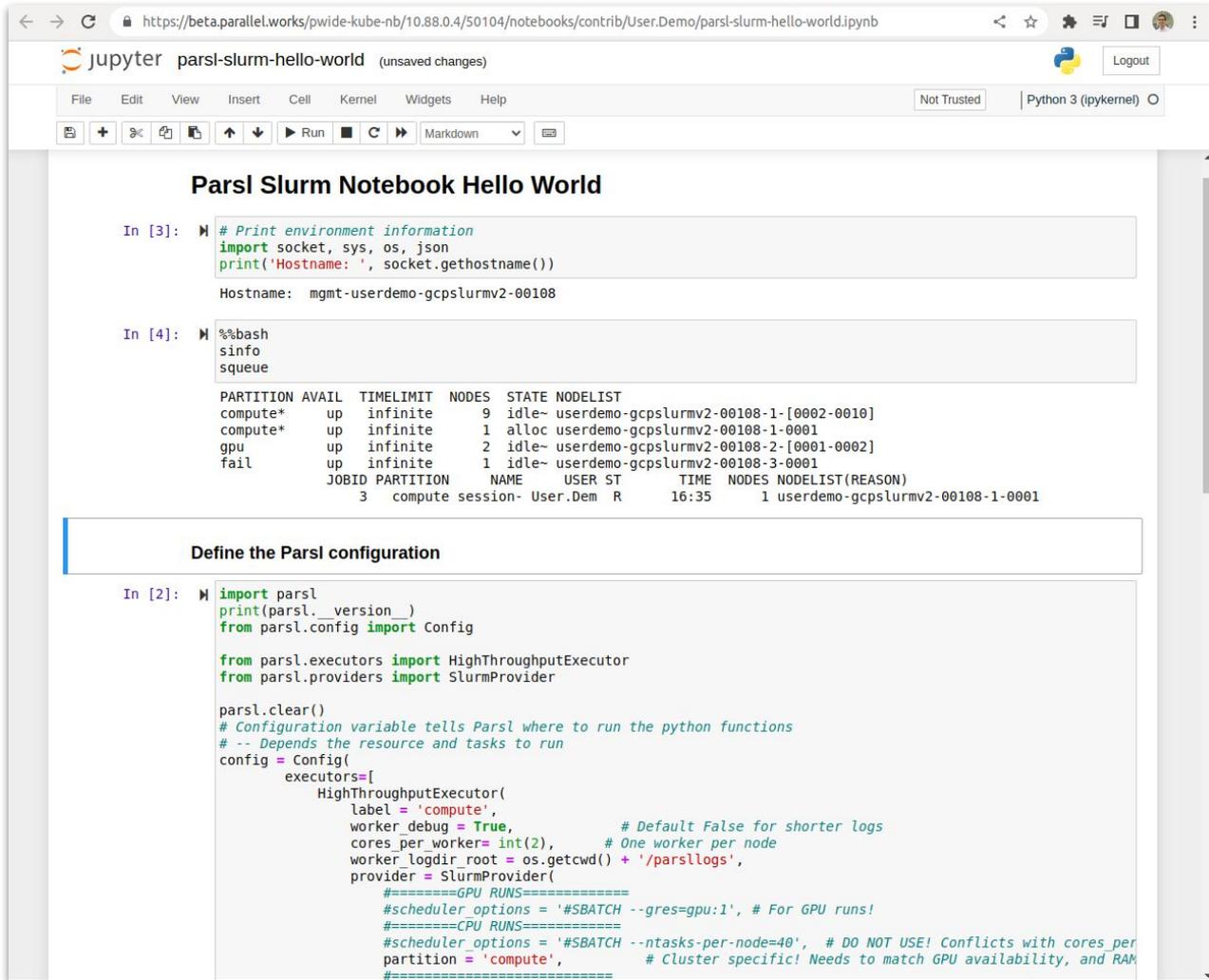
**Parsi  
notebook on  
the cluster**

PW

- 07220
- 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

## Approach:

- Send jobs to different partitions using the SlurmProvider



The screenshot shows a Jupyter Notebook interface with the following content:

### Parsl Slurm Notebook Hello World

```
In [3]: # Print environment information
import socket, sys, os, json
print('Hostname: ', socket.gethostname())

Hostname: mgmt-userdemo-gcpslurm2-00108
```

```
In [4]: %%bash
sinfo
squeue
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
compute*	up	infinite	9	idle-	userdemo-gcpslurm2-00108-1-[0002-0010]
compute*	up	infinite	1	alloc	userdemo-gcpslurm2-00108-1-0001
gpu	up	infinite	2	idle-	userdemo-gcpslurm2-00108-2-[0001-0002]
fail	up	infinite	1	idle-	userdemo-gcpslurm2-00108-3-0001

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
3	compute	session- User.Dem	R		16:35	1	userdemo-gcpslurm2-00108-1-0001

### 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

The screenshot displays the Parallel Works dashboard. At the top, there is a navigation bar with 'COMPUTE', 'RESOURCES', 'WORKFLOWS', and 'ACCOUNT' tabs. Below this, the 'Workflow Monitor' section is active, showing a table of workflow jobs. The table has columns for ID, Workflow, Status, Submitted, and Runtime (min). The workflow with ID 57220 is highlighted in yellow and has a 'Running' status. A red dashed box is drawn around the 'Stop' icon (a red circle with a diagonal line) for this workflow. Other workflows are in 'Deleted', 'Complete', or 'Error' states. On the right side, a sidebar shows a tree view of the workspace structure, including folders like 'archive', 'client', 'jobs', and 'workflows'. The 'jobs' folder is expanded, showing the workflow with ID 57220 selected.

ID	Workflow	Status	Submitted	Runtime (min)	Stop	Refresh	View
57220	INTERACTIVE_SESSION	Running	9:01 pm 9/9/2022	0	⊘	↻	👁
57215	INTERACTIVE_SESSION	Deleted	8:52 pm 9/9/2022	8	⊘	↻	👁
57214	INTERACTIVE_SESSION	Deleted	8:52 pm 9/9/2022	0.1	⊘	↻	👁
57212	INTERACTIVE_SESSION	Deleted	8:49 pm 9/9/2022	1.4	⊘	↻	👁
57211	INTERACTIVE_SESSION	Deleted	8:47 pm 9/9/2022	1.7	⊘	↻	👁
57207	INTERACTIVE_SESSION	Deleted	8:11 pm 9/9/2022	8	⊘	↻	👁
57172	SINGLECLUSTER_PARSL_DEMO	Complete	3:57 pm 9/9/2022	4.8	⊘	↻	👁
57170	SINGLECLUSTER_PARSL_DEMO	Complete	9:37 pm 9/8/2022	4.7	⊘	↻	👁
57169	SINGLECLUSTER_PARSL_DEMO	Error	9:35 pm 9/8/2022	0.3	⊘	↻	👁
57157	INTERACTIVE_SESSION	Deleted	9:25 pm 9/6/2022	1.3	⊘	↻	👁

## 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!**