

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



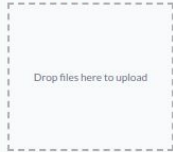
Add Compute

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



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

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
- MDLITE
- MULTICLUSTER_PARSL_DEMO
Multicluster 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

14 active:1 requested:1 stopped

AWS_SFG_CLUSTER (AWS Slurm Cluster - project: 62559eee995ce90043d6219a)

11 active:0 requested:0 stopped

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

Google Cloud

GCPSLURMV2GPU (GCP slurm test - project: cg-cloudmgmt)

OCIELASTIC (Oracle elastic) 1 core/active worker

SFG_V1_TEST

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 : gcpslurm2
    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 left sidebar shows a 'Workflows' section with a search bar and a grid of workflow thumbnails. One thumbnail, labeled 'TRAIN_SHIP_F...', is highlighted with a red dashed box and the number '1'. The main content area shows the configuration for the 'TRAIN_SHIP_FINDER' workflow. It is divided into two sections: 'Data Generation / Transformation (Xpatch / Matlab)' and 'Model Training (Neural Network)'. The first section contains fields for 'Input image directory' (pw://pw/storage/ships-in-satellite-imagery/shipsnet), 'Output image directory' (pw://[cwd]/shipsnet), 'Number of extra samples' (1000), 'Amplitude of noise' (150), 'Rotation range' (359), 'Maximum brightness shift' (1), 'ZCA whitening' (False), 'Horizontal flip' (True), and 'Vertical flip' (True). The second section has an 'Execute' button highlighted with a red dashed box and the number '3'. Below the configuration, there is a description of the workflow: 'Train Ship Finder', followed by instructions to take the 'shipsnet' dataset and run two tasks: 'Data generation' and 'Model training'. A 'Resources' section mentions modifying the 'executors.json' file. The right-hand pane shows a file explorer with folders like '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"}'
```

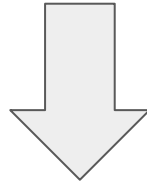
Goals for Parsl workflows:

- 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:

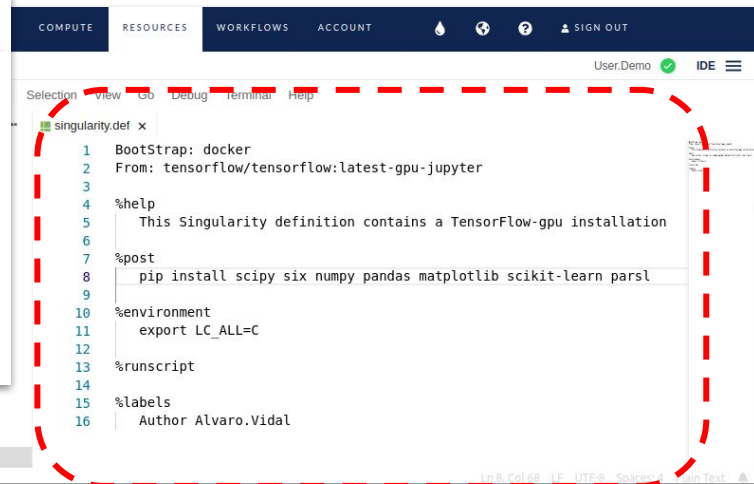
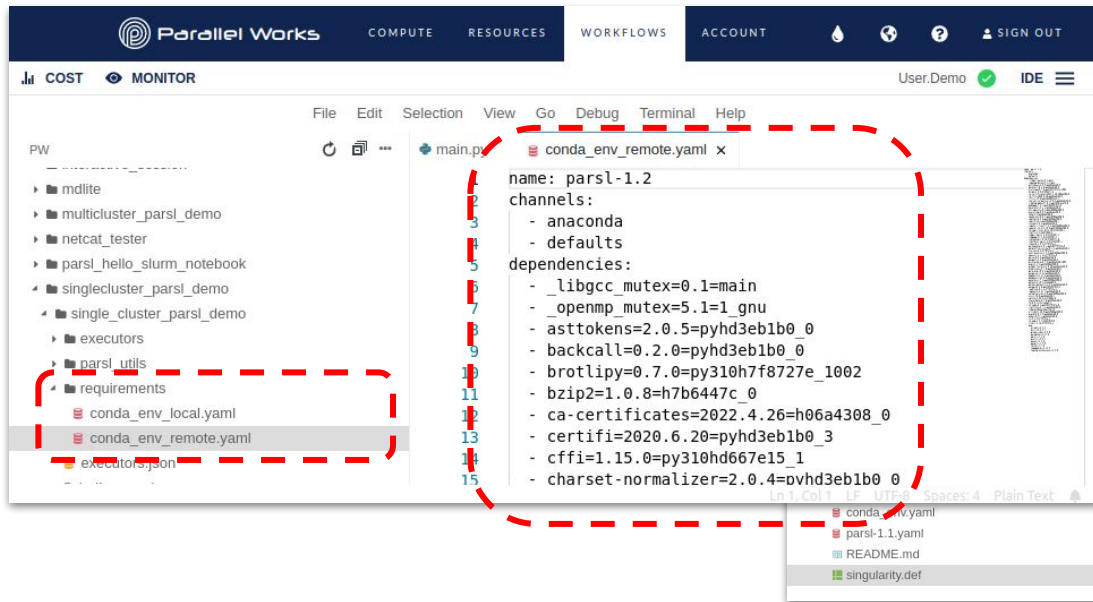
- **JSON configuration file**
- **PW API** to get pool information by pool name:
 - IP addresses and user name of the controller nodes
 - Available worker ports
- **SSHChannel** to connect to the controller nodes
- Run in **controller** nodes: **LocalProvider**
- Run in **compute** nodes: **SlurmProvider**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

JSON IS LOADED AND USED TO DEFINE THE PARSL CONFIGURATION

```
108
109
110
111
112
113
114
115
116
117
118
119
config = Config(
    executors = [
        HighThroughputExecutor(
            worker_ports = ((int(exec_conf['myexecutor_1']['WORKER_PORT_1']), int(exec_conf['myexecutor_1']['WORKER_PORT_2'])),
            label = 'myexecutor_1',
            worker_debug = True,
            cores_per_worker = float(exec_conf['myexecutor_1']['CORES_PER_WORKER']),
            worker_logdir_root = exec_conf['myexecutor_1']['WORKER_LOGDIR_ROOT'],
            provider = LocalProvider(
                worker_init = 'source {conda_sh}; conda activate {conda_env}; cd {run_dir}'.format(
                    conda_sh = os.path.join(exec_conf['myexecutor_1']['CONDA_DIR'], 'etc/profile.d/conda.sh'),
                    conda_env = exec_conf['myexecutor_1']['CONDA_ENV'],
                    run_dir = exec_conf['myexecutor_1']['RUN_DIR']
                ),
                channel = SSHChannel(
                    hostname = exec_conf['myexecutor_1']['HOST_IP'],
                    username = exec_conf['myexecutor_1']['HOST_USER'],
                    script_dir = exec_conf['myexecutor_1']['SSH_CHANNEL_SCRIPT_DIR'],
                    key_filename = '/home/{PW_USER}/.ssh/pw_id_rsa'.format(PW_USER = os.environ['PW_USER'])
                )
            )
        ],
        monitoring = MonitoringHub(
            hub_address = address_by_hostname(),
            resource_monitoring_interval = 5
        )
    )
print('Loading Parsl Config', flush = True)
parsl.load(config)
```

```
objects (1)
  myexecutor_1 (17)
    pool: gpsstamv2
    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 workflow configuration details for 'myexecutor_1'. The configuration is shown in a tree view under 'object {1}' and 'myexecutor_1 {13}'. The configuration includes 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 managing the Python environment.

Options:

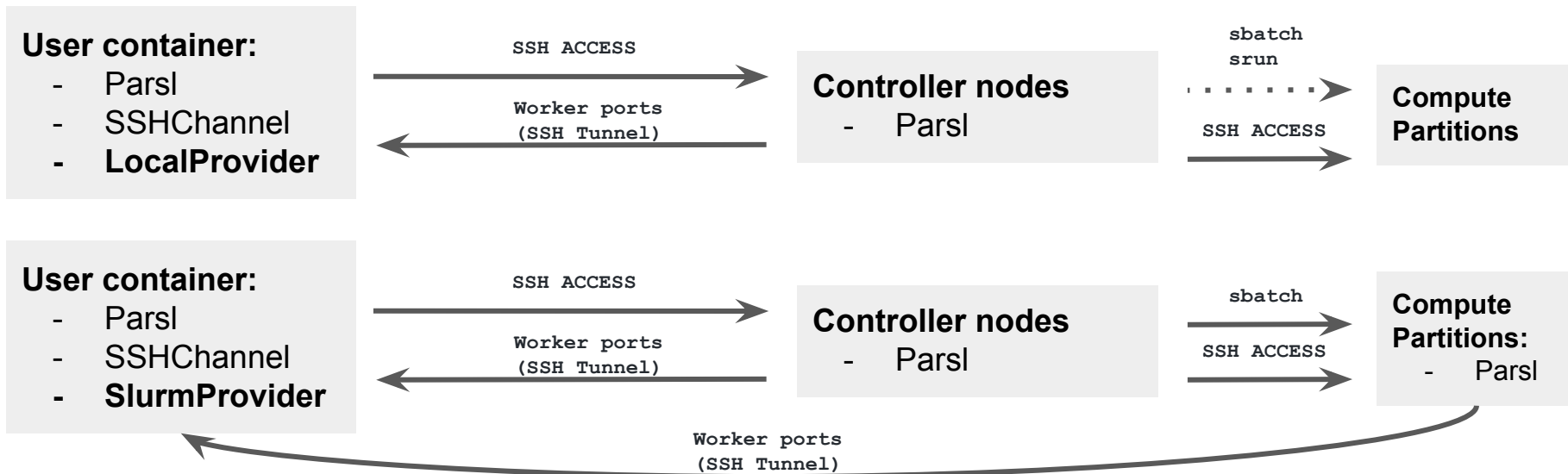
1. Point executor to different python environment
2. Install the Python environment manually once
3. Workflow wrapper installs python environment at runtime (INSTALL_CONDA=TRUE)

Python environment

Optional to install the Python environment at runtime

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:

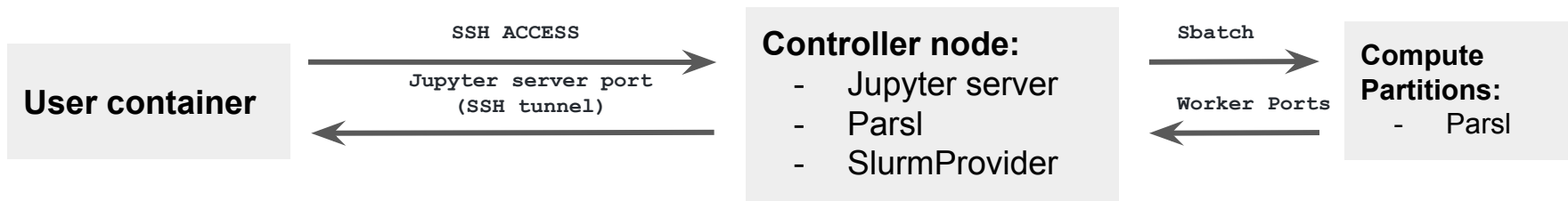
- Connect from the user container to the jupyter server
- Automate server launch

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 'Path to conda.sh' is set to '/contrib/User.Demo/miniconda3/etc/profile.d/conda.sh'. The 'Conda environment' is set to 'base'. The 'Password for notebook access' is 'EnterPassword'. The 'Desktop Host' section has 'Partition' selected. The 'Advanced Options' section has an 'Execute' button. A file explorer on the right shows the directory structure, with 'jupyter' selected.

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 is a 'Workflows' sidebar with 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 features the 'jupyter' logo and a login form with a 'Password:' label, a masked password input field, and a 'Log in' button. On the right is a file explorer sidebar showing the directory structure: 'PW' containing '57220' which includes 'interactive_session' with files like 'env.sh', 'kill_session.sh', 'kill_tunnels_template.sh', 'kill.sh', 'main.sh', 'params.run', 'pw.conf', 'README.md', 'service.html', 'session-57220.out', and 'session.sh'. The bottom status bar includes a left arrow, 'Status', '© 2022 Parallel Works, Inc. - v3.4.26', 'Feedback', and a right arrow.



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

jupyter

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"/>		parsi-slurm-hello-world.ipynb		seconds ago	12.8 kB
<input type="checkbox"/>		minicondas-latest-Linux-x86_64.sh		7 months ago	75.7 MB

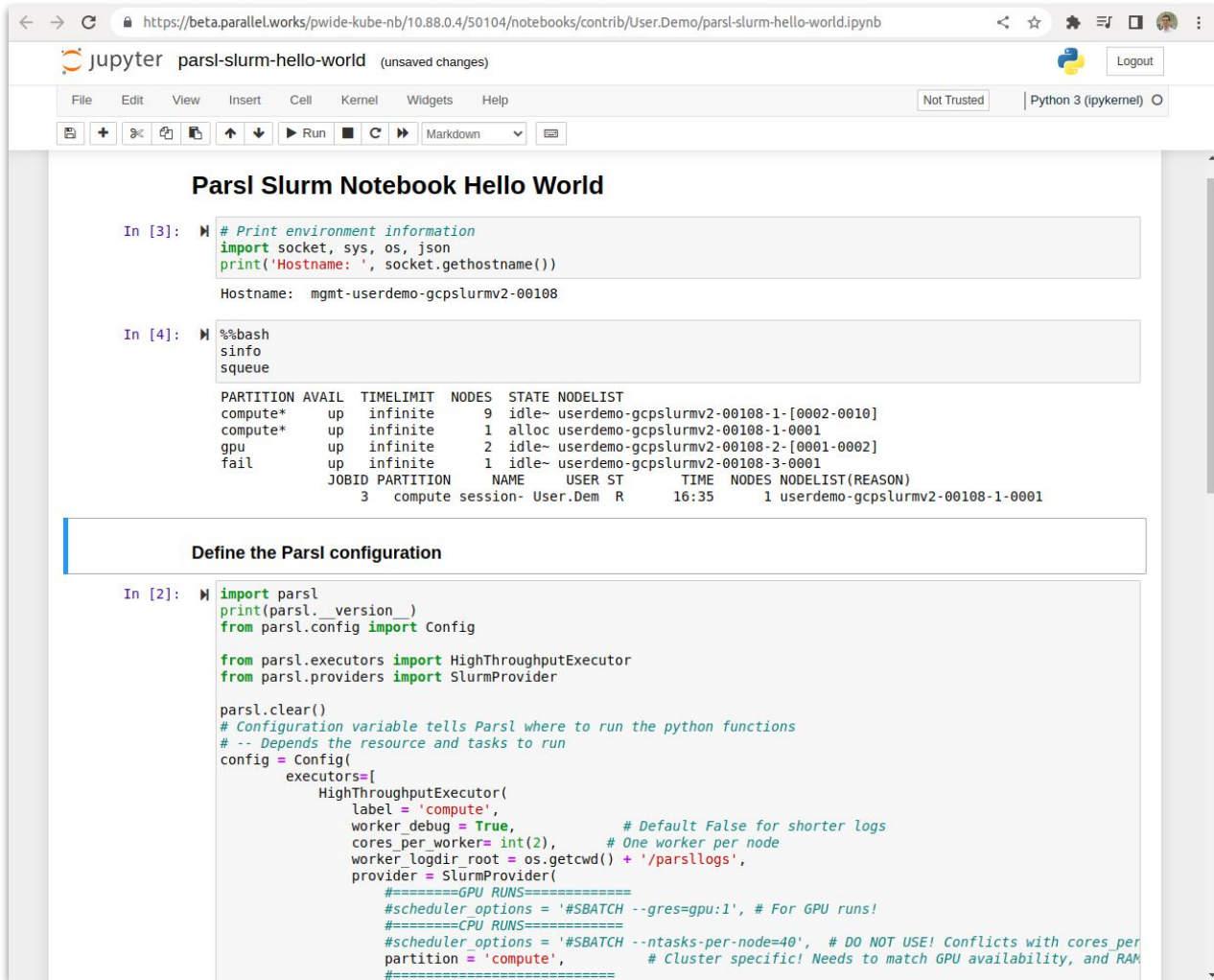
**Parsi
notebook on
the cluster**

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. The 'WORKFLOWS' tab is active. Below the navigation bar, there are sections for 'Workflows' on the left and 'Workflow Monitor' in the center. The 'Workflow Monitor' section contains a table with the following data:

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	⊘	↻	👁

The 'Stop' icon (a red circle with a diagonal slash) for the workflow with ID 57220 is highlighted with a red dashed box. Below the table, there is a 'Resource Monitor' section. On the right side of the dashboard, there is a sidebar with a tree view showing the project structure, including folders like 'archive', 'client', 'jobs', '57211', '57220', 'jupyter-server', 'notebooks', 'rserver', 'services', 'storage', 'workflows', 'converge_runner', 'failover_demo', 'find_ships', 'fv3_ufs_srweather_nb_demo', 'hello_cluster_ssh', 'interactive_session', and 'jupyter'.

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!