

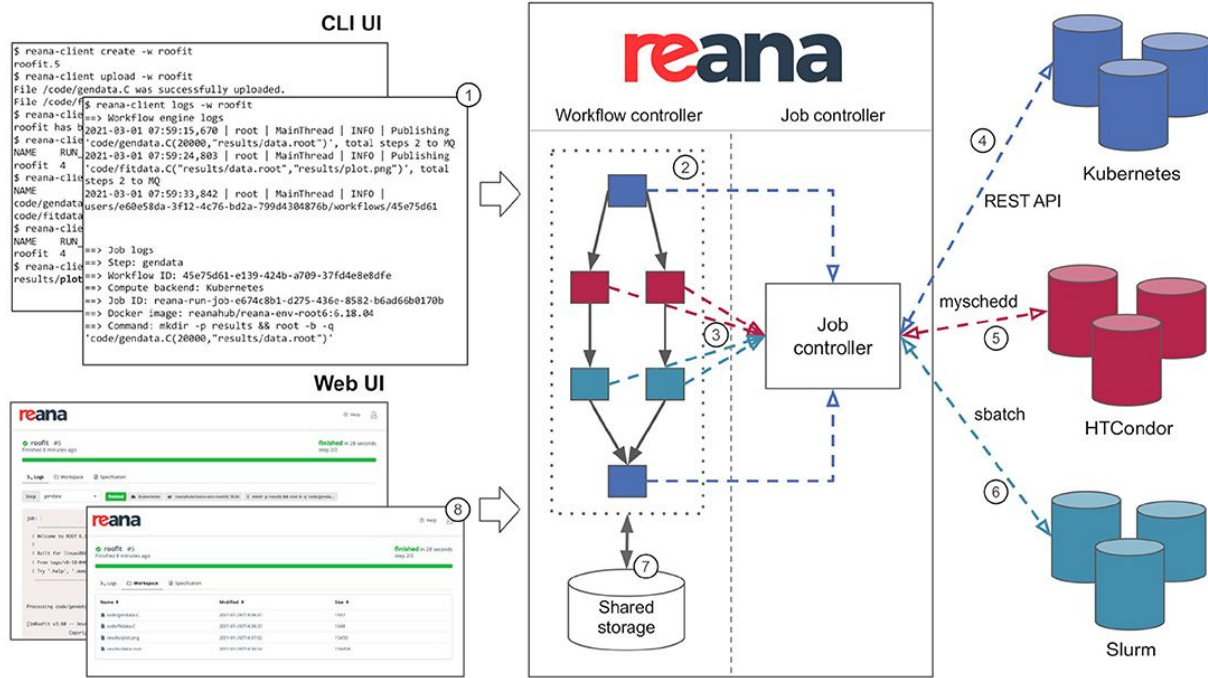


Supporting Dask workloads in the REANA reproducible analysis platform

Alp Tuna (Bogazici University)

Mentor: Dr. Tibor Šimko (CERN)

What is REANA?



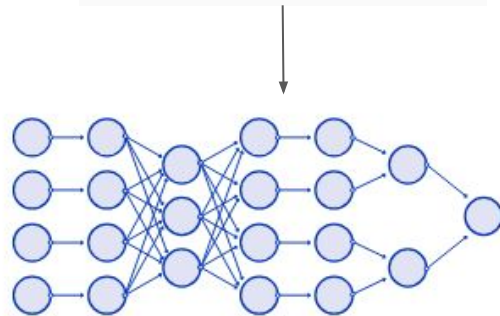
Running declarative workflows (CWL, Snakemake, Yadage) on the cloud

How could we support Dask? dask

- **Dask** cannot be serialized; it constructs its own DAG dynamically.
- Dask clusters should be started at and stopped after workflow execution.
- A reusability stand point of REANA:
 - We need to support any past Dask version that the analyst might have been using.
 - We need to bring any services that the analyst may need for successful workflow execution.

```
import dask.bag as db

# Read large datasets in parallel
lines = db.read_text("s3://mybucket/data/*.json")
records = (lines
           .map(json.loads)
           .filter(lambda d: d["value"] > 0)
           )
df = records.to_dask_dataframe()
```



Dask Task Graph

Defining Dask Workflows in REANA: YAML Specification

```
1  inputs:
2    files:
3      - analysis.py
4  workflow:
5    type: serial
6    resources:
7      dask:
8        image: docker.io/coffeateam/coffea-dask-cc7:0.7.22-py3.10-g7f049
9        number_of_workers: 5
10       single_worker_memory: 8Gi
11  specification:
12    steps:
13      - name: process
14        environment: docker.io/coffeateam/coffea-dask-cc7:0.7.22-py3.10-g7f049
15        commands:
16          - python analysis.py
17  outputs:
18    files:
19      - histogram.png
20  tests:
21    files:
22      - tests/log-messages.feature
23      - tests/workspace-files.feature
```

Example reana.yaml specification

container image for
your Dask workers

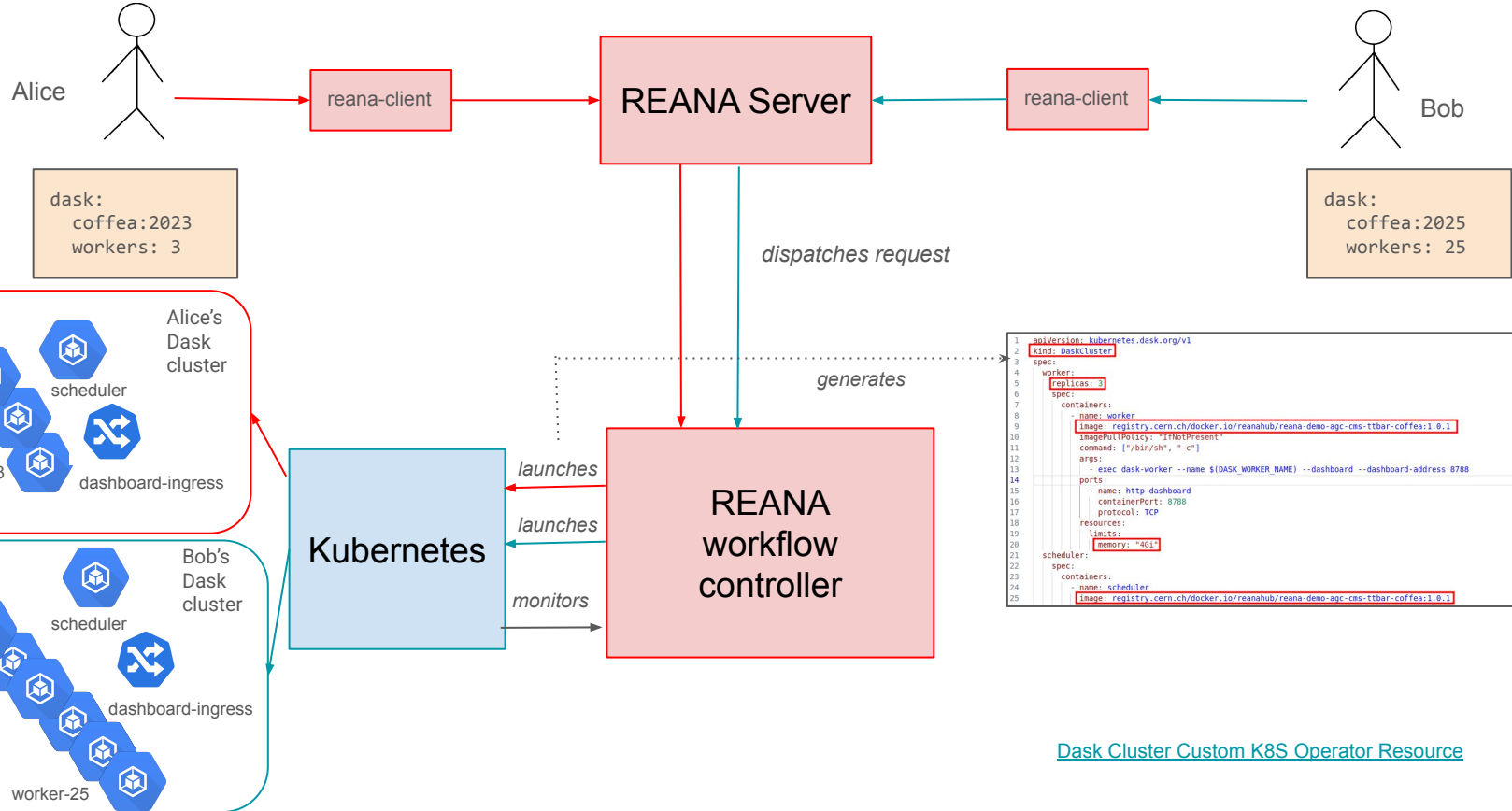
number of workers in your Dask
cluster

memory of each worker in your
Dask cluster



Each researcher can ask
for custom Dask version
dedicated to their
workflow.

Under the Hood: How Dask Works in REANA



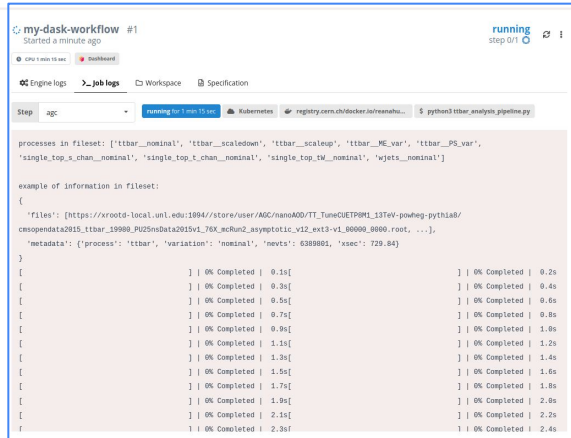
[Dask Cluster Custom K8S Operator Resource](#)

Monitoring & Debugging: Capturing Dask Logs in REANA

Job logs

Scheduler logs

Worker logs

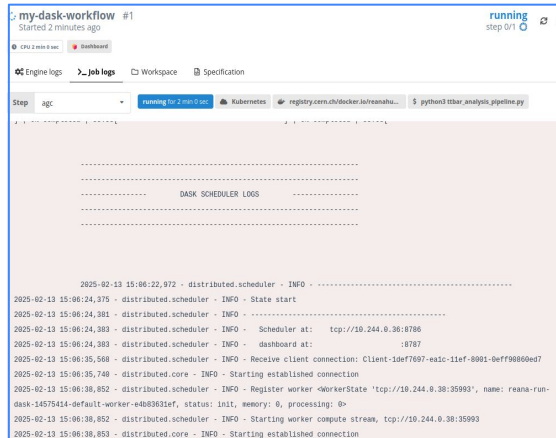


The screenshot shows the 'Job logs' tab for a workflow named 'my-dask-workflow #1'. The workflow is in a 'running' state, having started a minute ago. The logs display the fileset configuration and a table of job completion progress. The fileset includes 'ttbar_nominal', 'ttbar_scaledown', 'ttbar_scaleup', 'ttbar_ME_var', 'ttbar_PS_var', 'single_top_s_chan_nominal', 'single_top_t_chan_nominal', 'single_top_TW_nominal', and 'wjets_nominal'. The table shows progress for each job, with completion percentages ranging from 0.14 to 2.46.

```
processes in fileset: ['ttbar_nominal', 'ttbar_scaledown', 'ttbar_scaleup', 'ttbar_ME_var', 'ttbar_PS_var', 'single_top_s_chan_nominal', 'single_top_t_chan_nominal', 'single_top_TW_nominal', 'wjets_nominal']

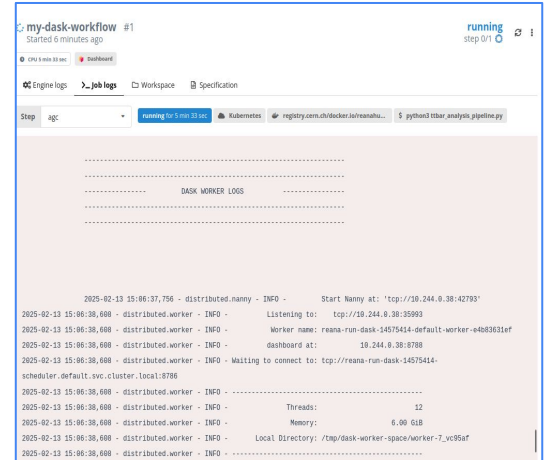
example of information in fileset:
{
  "files": ["https://roots-local.unl.edu:1894//store/user/AGC/nanoAOD/TT_TuneCUETP8M_13TeV-powheg-pythia8/cesopendata2015_ttbar_19960_PU25ndata2015v1_76X_mcRun2_asymptotic_v12_ext3-v1_80960_8800_root..."],
  "metadata": [{"process": "ttbar", "variation": "nominal", "nests": 6389801, "nsec": 729.84}
]}

[ ] | 0% Completed | 0.14 | ] | 0% Completed | 0.26
[ ] | 0% Completed | 0.36 | ] | 0% Completed | 0.48
[ ] | 0% Completed | 0.56 | ] | 0% Completed | 0.68
[ ] | 0% Completed | 0.76 | ] | 0% Completed | 0.88
[ ] | 0% Completed | 0.96 | ] | 0% Completed | 1.08
[ ] | 0% Completed | 1.16 | ] | 0% Completed | 1.28
[ ] | 0% Completed | 1.36 | ] | 0% Completed | 1.48
[ ] | 0% Completed | 1.56 | ] | 0% Completed | 1.68
[ ] | 0% Completed | 1.76 | ] | 0% Completed | 1.88
[ ] | 0% Completed | 1.96 | ] | 0% Completed | 2.08
[ ] | 0% Completed | 2.16 | ] | 0% Completed | 2.26
[ ] | 0% Completed | 2.36 | ] | 0% Completed | 2.46
```



The screenshot shows the 'Scheduler logs' tab for the same workflow. The logs are filtered to show 'DASK SCHEDULER LOGS'. The output shows a series of INFO messages from the distributed scheduler, including state start, client connection details, and worker registration information.

```
2025-02-13 15:06:22,872 - distributed.scheduler - INFO - .....
2025-02-13 15:06:24,375 - distributed.scheduler - INFO - State start
2025-02-13 15:06:24,381 - distributed.scheduler - INFO - .....
2025-02-13 15:06:24,383 - distributed.scheduler - INFO - Scheduler at: tcp://10.244.0.38:8786
2025-02-13 15:06:24,383 - distributed.scheduler - INFO - dashboard at: :8787
2025-02-13 15:06:35,560 - distributed.scheduler - INFO - Receive client connection: Client-1de7f897-ea1c-11ef-8901-0ef798860e07
2025-02-13 15:06:35,740 - distributed.core - INFO - Starting established connection
2025-02-13 15:06:38,852 - distributed.scheduler - INFO - Register worker <WorkerState 'tcp://10.244.0.38:35993', name: reana-run-dask-14575414-default-worker-e4b83631ef, status: INIT, memory: 0, processing: 0>
2025-02-13 15:06:38,852 - distributed.scheduler - INFO - Starting worker compute stream, tcp://10.244.0.38:35993
2025-02-13 15:06:38,853 - distributed.core - INFO - Starting established connection
```



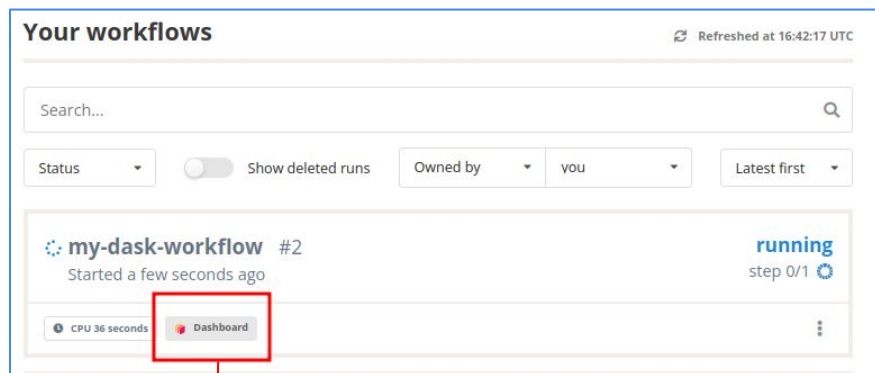
The screenshot shows the 'Worker logs' tab for the workflow. The logs are filtered to show 'DASK WORKER LOGS'. The output shows INFO messages from the distributed worker, including listening information, worker name, dashboard URL, and local directory details.

```
2025-02-13 15:06:37,756 - distributed.nanny - INFO - Start Nanny at: 'tcp://10.244.0.38:42783'
2025-02-13 15:06:38,600 - distributed.worker - INFO - Listening to: tcp://10.244.0.38:35993
2025-02-13 15:06:38,600 - distributed.worker - INFO - Worker name: reana-run-dask-14575414-default-worker-e4b83631ef
2025-02-13 15:06:38,600 - distributed.worker - INFO - dashboard at: 10.244.0.38:8788
2025-02-13 15:06:38,600 - distributed.worker - INFO - Waiting to connect to: tcp://reana-run-dask-14575414-scheduler-default.svc.cluster.local:8786
2025-02-13 15:06:38,600 - distributed.worker - INFO - .....
2025-02-13 15:06:38,600 - distributed.worker - INFO - Threads: 12
2025-02-13 15:06:38,600 - distributed.worker - INFO - Memory: 6.00 GiB
2025-02-13 15:06:38,600 - distributed.worker - INFO - Local Directory: /tmp/dask-worker-space/worker-7_vc96af
2025-02-13 15:06:38,600 - distributed.worker - INFO - .....
```

Logs are captured “live” thanks to [Jelizaveta Lemeševa’s IRIS-HEP fellow project](#) that was presented in October

Visualizing Dask in REANA: The Dashboard

REANA web interface



Your workflows Refreshed at 16:42:17 UTC

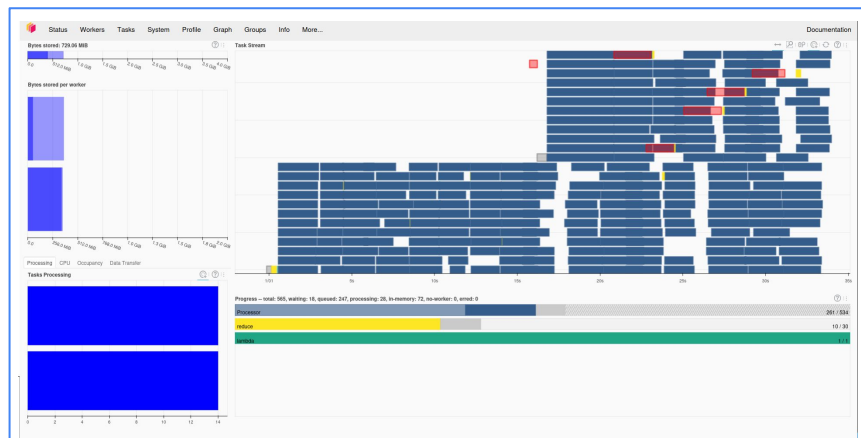
Search...

Status Show deleted runs Owned by you Latest first

my-dask-workflow #2 running
Started a few seconds ago step 0/1

CPU 36 seconds **Dashboard**

Dask dashboard web interface



Putting It All Together: A Complete Workflow Example

1) Define your Dask cluster in “reana.yaml”

```
dask:  
  image: registry.cern.ch/docker.io/reanahub/reana-demo-agc-cms-ttbar-coffea:1.0.1  
  number_of_workers: 3  
  single_worker_memory: 4Gi
```

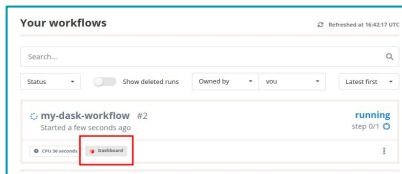
2) Use “DASK_SCHEDULER_URI” environment variable in your analysis

```
import os  
from dask.distributed import Client  
  
DASK_SCHEDULER_URI = os.getenv("DASK_SCHEDULER_URI")  
client = Client(DASK_SCHEDULER_URI)
```



REANA will use this environment variable to connect your job to the dedicated Dask cluster

3) Submit and monitor your Dask workflows



Follow a simple Dask Coffea example



[reana-demo-dask-coffea](https://github.com/reanahub/reana-demo-dask-coffea)

Conclusions



Current Status

- Dask support is implemented in REANA 0.95.0-alpha.1 version that was released on February 6th 2025
- Tested [reana-demo-dask-coffea](#), [calver](#), [AGC CMS ttbar open data analysis](#) examples
- Support for Kerberos and VOMS sidecars to access remote restricted data

What is next?

- Support HTCondor and Slurm backends for Dask workflows (currently only Kubernetes)
- Expose number of threads as an option to users
- Improve monitoring of Dask clusters in case of cluster failures

Thank you for your attention!

Early alpha testers welcome at  [Mattermost](#) :)