Standardized Computational Workflows at EBRAINS

Sofia Karvounari
Athena Research Center – Technical Coordination team

Standardization

Define in a common standard way:

- Computational workflows: series of **EBRAINS tools** that can be linked/combined to create Directed Acyclic Graphs (DAGs), loops or branches for accomplishing a scientific objective.

- **EBRAINS tools**: scientific simulations or data analysis (data manipulation) pieces of software **packaged** together with libraries, dependencies, binaries.

Workflows and EBRAINS tools will be **executed** via workflow management systems that can lead to **monitoring**, automation of some procedures and recovery of failing steps.

Two worlds..

SCIENTISTS

TECHNICAL EXPERTS

.. linked together!

SCIENTISTS

TECHNICAL EXPERTS

What standardization offer

FOR SCIENTISTS

Automation

Scalability

Accessibility

Portability

FAIRness is everywhere…

F A I R

Findable Accessible Interoperable Reusable

[1] The FAIR Guiding Principles for scientific data management and stewardship

[2] Sharing Fair Research Objects

...Why standard workflows are important?

FOR EBRAINS

FAIR tools + **FAIR data** + **FAIR workflows**

= **FAIR e-Infrastructure**

Let's take a step back

"EBRAINS is a new digital research infrastructure, created by the EU-funded Human Brain Project, that gathers an extensive range of data and tools for brain-related research."

Service categories at EBRAINS

DATA & KNOWLEDGE	ATLASES	SIMULATION
BRAIN-INSPIRED TECHNOLOGIES	MEDICAL DATA ANALYTICS	

[2] https://ebrains.eu/#news_anchor

Current limitations

WORKFLOWS

- No straight forward way to combine tools together
- No way to monitor large jobs that take time
- No way to pause, restart failed tasks, provide logs
- No way to be easily found, accessed and replicated/ reproduced
- Configuration adjustments in order to run in different infrastructures
- No graphs, dependencies, parallelism provided by Jupyter Notebooks

NON – INTERACTIVE TOOLS

- Not defined in an interoperable, standard and easy way to (re-)use

(Standard) Computational Workflows

Definition

Command Line tools

Execution

Monitoring

(Standard) Computational Workflows

Definition

Command Line tools

Execution

Monitoring

Dive in defining

A. Common Workflow Language

B. Specific Use Case written in CWL

C. Rabix composer

A. Common Workflow Language

COMMON WORKFLOW LANGUAGE

WHAT IT IS

Emerging open standard/common way to define analysis workflows and tools

ITS VALUES (ASAP)

- Offers **Scalability** & **Portability**
- Easy **Accessibility**
- Open standard, hence defining workflows and tools become **Automated**

A. Common Workflow Language

COMMON WORKFLOW LANGUAGE

WHY USE

- Common declarative format for **tools** and **workflows**
- Community based standards effort. Extensible
- Support containers (Docker, Singularity)
- Data locality
  - input and output files are modeled as rich objects with URI/IRI + metadata
  - platforms that understand CWL can
    - use URI/IRI to send compute near location of data
    - fetch data from a remote location

[1] Portable workflow creation and deployment with the CWL standards

A. Common Workflow Language

COMMON WORKFLOW LANGUAGE

WHY USE

- Other fields (Bioinformatics, Astrophysics) already see the value
- Compatible with plethora of workflow management systems
- Decouples description from execution of a workflow

[1] Portable workflow creation and deployment with the CWL standards

B. Use Case: Power Spectral Density

WORKFLOW VIA CWL

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

Thanks to Arnau Manasanch (WP2)

B. Use Case: Power Spectral Density
Fetching Tool

WORKFLOW STEP

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

TOOL DEFINITION

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: fetching_data.py
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_fetching_data:latest
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  bucket_id:
    type: string
    inputBinding:
      position: 1
  object_name:
    type: string
    inputBinding:
      position: 2
  token:
    type: string
    inputBinding:
      position: 3
outputs:
  fetched_file:
    type: File
    outputBinding:
      glob: $(inputs.object_name)
```

B. Use Case: Power Spectral Density
Analysis Tool

WORKFLOW STEP

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

TOOL DEFINITION

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: analysis.py
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_analysis:latest
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  input_file:
    type: File
    inputBinding:
      position: 1
  output_file_name:
    type: string
    inputBinding:
      prefix: --output_file
      position: 2
  channels:
    type: int[]
    inputBinding:
      prefix: --channels
      position: 3
outputs:
  output_file:
    type: File
    outputBinding:
      glob: $(inputs.output_file_name)
```

B. Use Case: Power Spectral Density
Visualisation Tool

WORKFLOW STEP

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

TOOL DEFINITION

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: analysis.py
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_analysis:latest
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  input_file:
    type: File
    inputBinding:
      position: 1
  output_file_name:
    type: string
    inputBinding:
      prefix: --output_file
      position: 2
  channels:
    type: int[]
    inputBinding:
      prefix: --channels
      position: 3
outputs:
  output_file:
    type: File
    outputBinding:
      glob: $(inputs.output_file_name)
```

B. Use Case: Power Spectral Density

WORKFLOW INPUTS

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

INPUT PARAMETERS

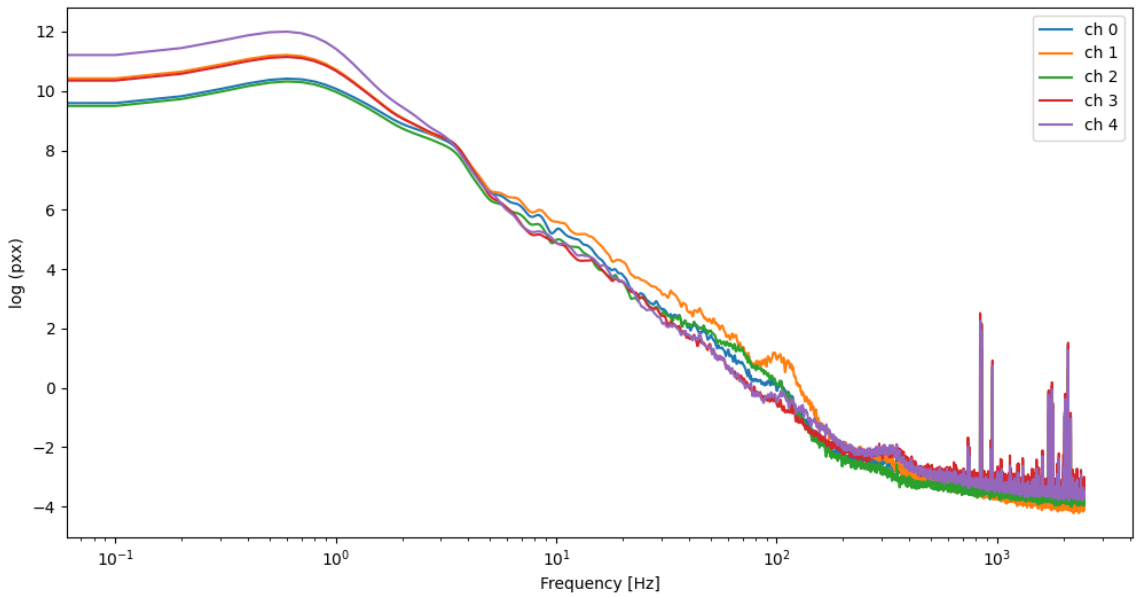
```yaml
bucket_id: 'fetching-data-from-bucket-and-kg'
input_file: 'data_mat/sub-1_chs-32_hem-RH_ana-ISO000_stim-SPN.mat'
channels: [0, 1, 2, 3, 4]
psd_output_file_name: 'psd.json'
output_file_name: 'output.png'
token: 'COPY_TOKEN_HERE'
```

B. Use Case: Power Spectral Density

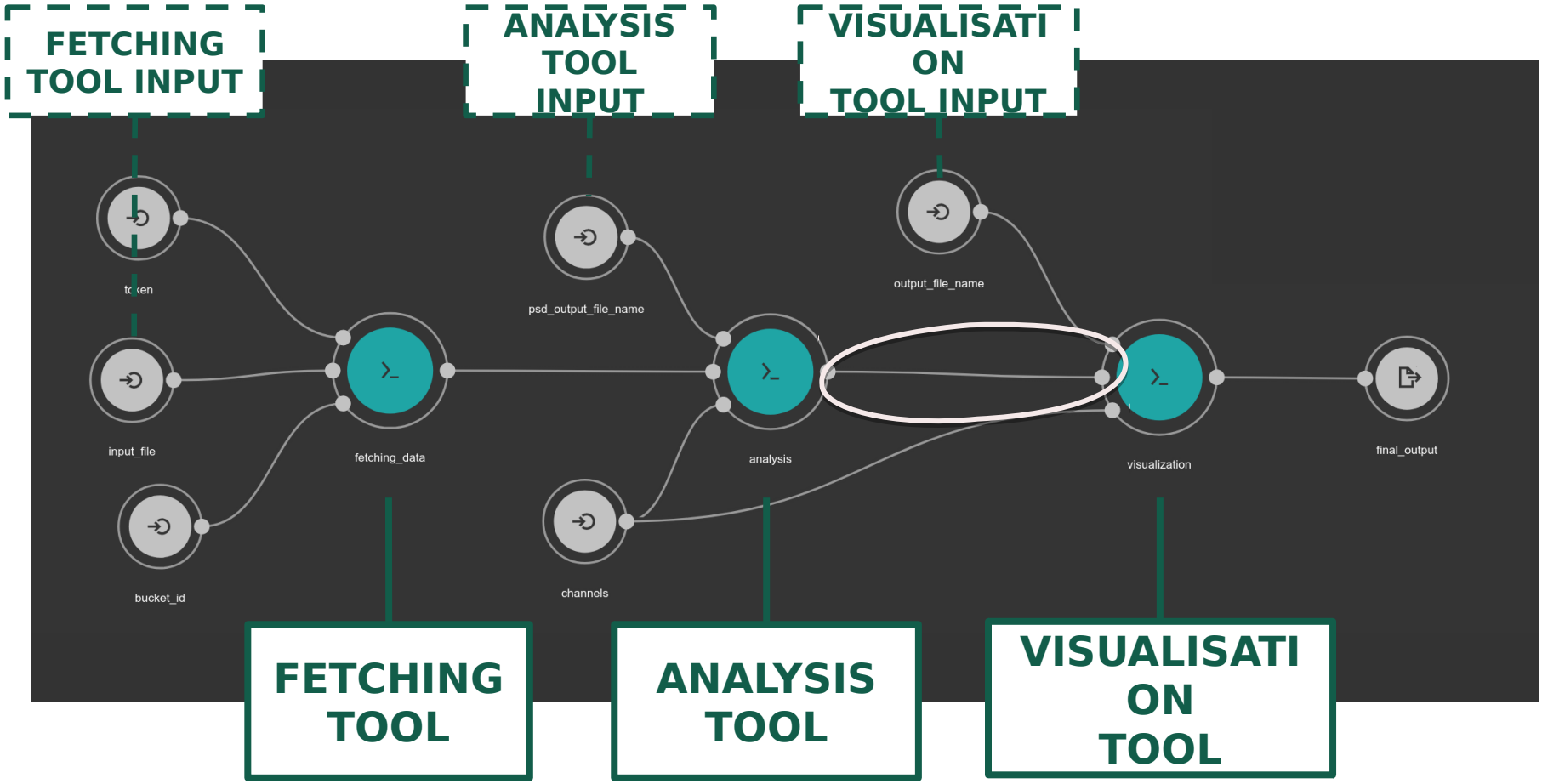
WORKFLOW OUTPUT

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

PLOT



C. Rabix composer



[1] Rabix

(Standard) Computational Workflows

Definition

Command Line tools

Execution

Monitoring

Dive in Command Line tools

A. Types of tools

B. Descriptions via CWL

C. Packaged vs Unpackaged

D. Why Docker?

How a tool can be packaged via Docker?
What else is there?

E. Why Packaged Command Line tools important?

A. Types of tools:

NON-INTERACTIVE

User (possibly) interacts with the program before the runtime

INTERACTIVE

User interacts with the program during the runtime

B. Descriptions via CWL

WORKFLOW

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: Workflow

inputs:
  bucket_id: string
  input_file: string
  channels: int[]
  psd_output_file_name: string
  output_file_name: string
  token: string

outputs:
  final_output:
    type: File
    outputSource: visualization/plot

steps:
  fetching_data:
    run: step1/fetching_data_tool.cwl
    in:
      bucket_id: bucket_id
      object_name: input_file
      token: token
    out: [fetched_file]

  analysis:
    run: step2/analysis_tool.cwl
    in:
      input_file: fetching_data/fetched_file
      output_file_name: psd_output_file_name
      channels: channels
    out: [output_file]

  visualization:
    run: step3/visualization_tool.cwl
    in:
      input_file: analysis/output_file
      output_file_name: output_file_name
      channels: channels
    out: [plot]
```

COMMAND LINE TOOL

```yaml
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs:
  message:
    type: string
    inputBinding:
      position: 1
stdout: output.txt
outputs:
  out:
    type: stdout
```

B. Descriptions via CWL

COMMAND LINE TOOL

```
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: analysis.py
hints:
  [redacted]
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  input_file:
    type: File
    inputBinding:
      position: 1
  output_file_name:
    type: string
    inputBinding:
      prefix: --output_file
      position: 2
  channels:
    type: int[]
    inputBinding:
      prefix: --channels
      position: 3
outputs:
  output_file:
    type: File
    outputBinding:
      glob: $(inputs.output_file_name)
```

C. Packaged vs Unpackaged?

```yaml
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs:
  message:
    type: string
    inputBinding:
      position: 1
stdout: output.txt
outputs:
  out:
    type: stdout
```

- Tool needs to be in $PATH in order for CWL to access it
- All dependencies, libraries must be taken care by the developer manually
- Tool already deployed & ready to be used

C. Packaged vs Unpackaged?

```yaml
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_analysis:latest
```

D. Why Docker?

```yaml
#!/usr/bin/env cwltool

cwlVersion: v1.0
class: CommandLineTool
baseCommand: analysis.py
hints:
  DockerRequirement:
    dockerPull: docker-registry.ebrains.eu/tc/cwl-workflows/psd_workflow_analysis:latest
  ResourceRequirement:
    ramMin: 2048
    outdirMin: 4096
inputs:
  input_file:
    type: File
    inputBinding:
      position: 1
  output_file_name:
    type: string
    inputBinding:
      prefix: --output_file
      position: 2
  channels:
    type: int[]
    inputBinding:
      prefix: --channels
      position: 3
outputs:
  output_file:
    type: File
    outputBinding:
      glob: $(inputs.output_file_name)
```

- CWL works great with Docker containers, input/output files are taking care
- Dependencies, libraries, binaries, code: all packaged
- Docker is supported by other containerized methods like Singularity (HPC)

D. Why Docker?

How can a tool be packaged via Docker?

```
FROM python:3.8-slim

# install dependencies
RUN pip install --no-cache-dir argparse numpy pandas scipy

# copy python script, make executable and add to path
COPY analysis.py /home/tool/analysis.py
RUN chmod +x /home/tool/analysis.py
ENV PATH="/home/tool:$PATH"

CMD [ "/bin/bash" ]

```

1. Create a simple "Dockerfile" with all the commands to install the tool dependencies.

2. "build" (create) the image:

```
docker build -t docker-registry.ebrains.eu/tc/cwl-workflows/analysis -f Dockerfile
```

or "tag" (rename) if it already exists:

```
docker tag old_image_name docker-registry.ebrains.eu/tc/cwl-workflows/analysis
```

3. "push" (upload) the image:

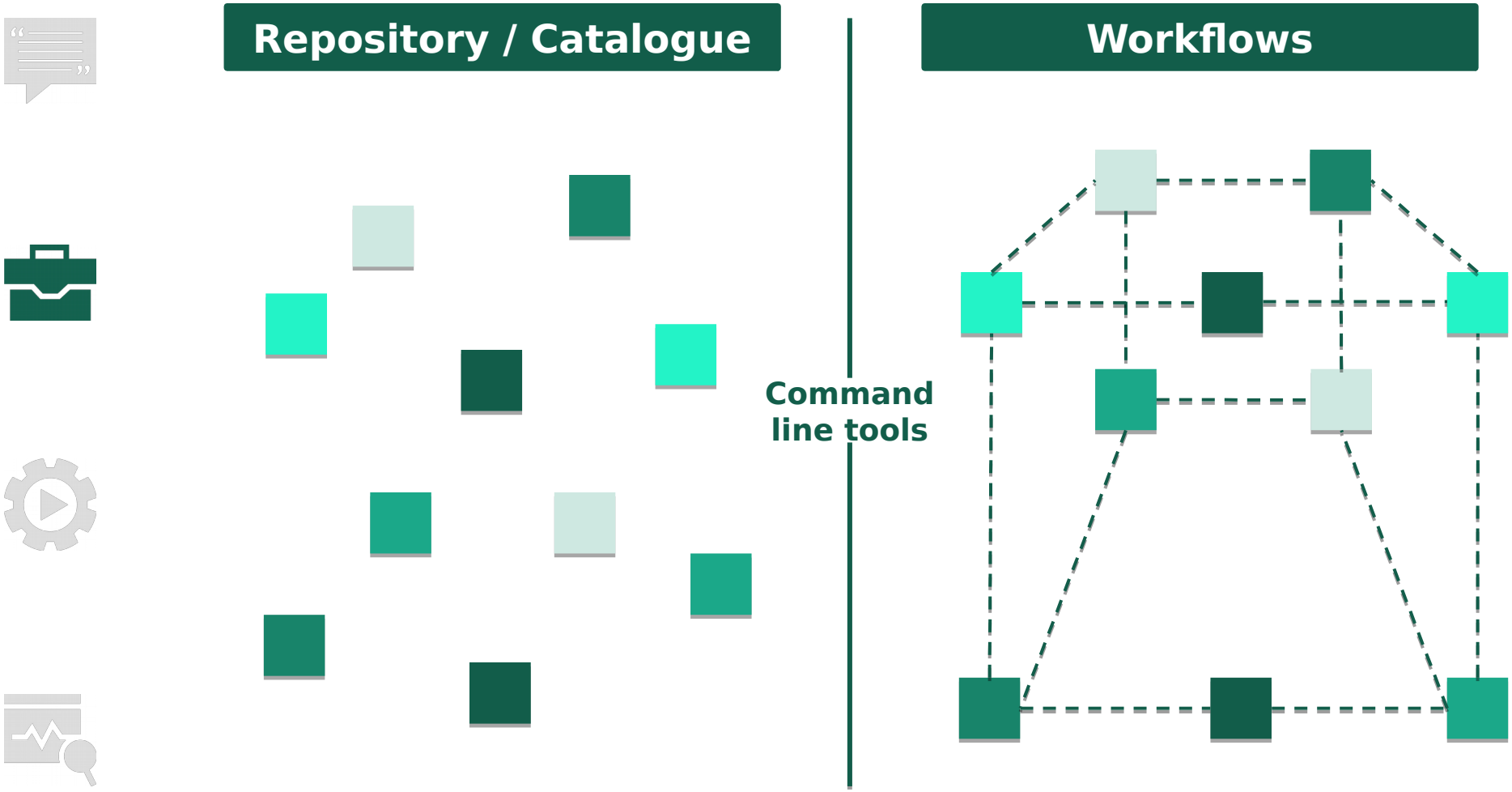
```
docker login docker-registry.ebrains.eu
docker push docker-registry.ebrains.eu/tc/cwl-workflows/analysis
```

D. Why Docker?
What else is there?

FUTURE USE CASE?

- Singularity
- Modules (Spack)
- Automatic installation of packages via IRIs (beta)

E. Why Packaged Command Line tools important?



(Standard) Computational Workflows

Definition

Command Line tools

Execution

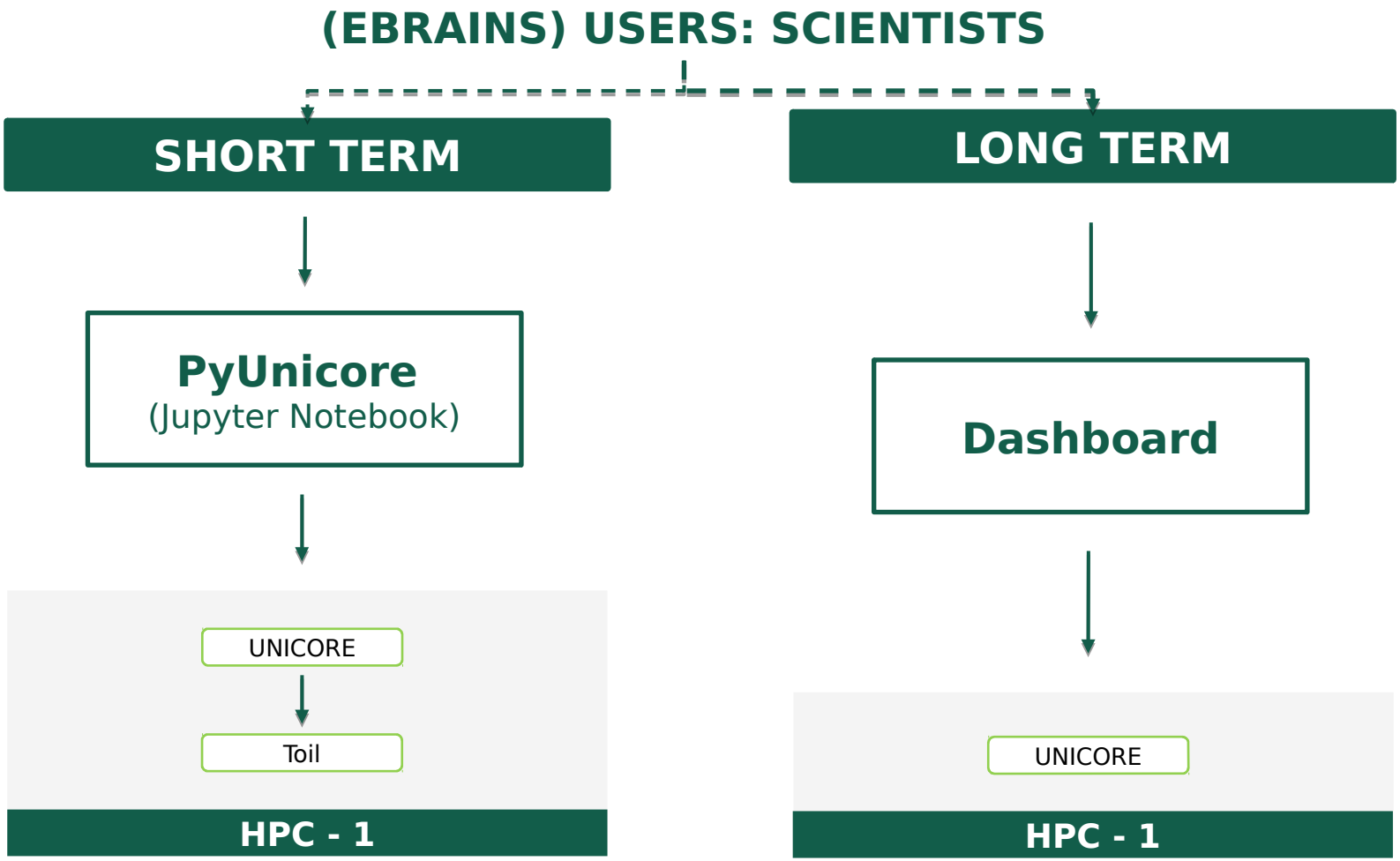
Monitoring

Dive in executing

A. HPC

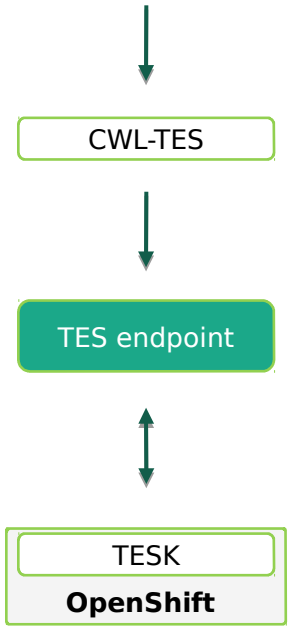
B. OpenShift

A. HPC


(EBRAINS) USERS: SCIENTISTS
SHORT TERM
LONG TERM
PyUnicore
(Jupyter Notebook)
Dashboard
UNICORE
Toil
HPC - 1
UNICORE
HPC - 1

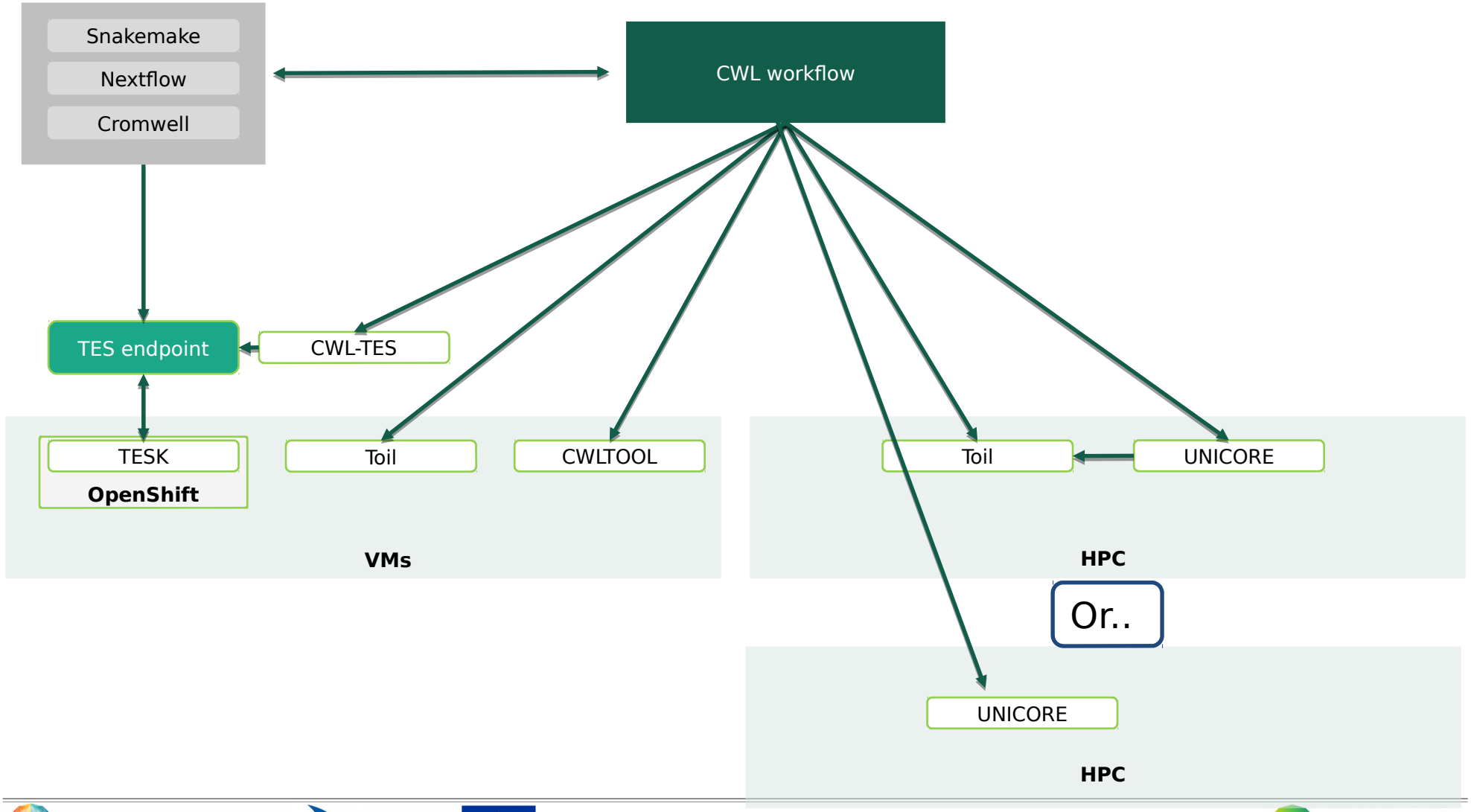
B. OpenShift

(EBRAINS) USERS: SCIENTISTS



In the making...

Another view..



(Standard) Computational Workflows

Definition

Command Line tools

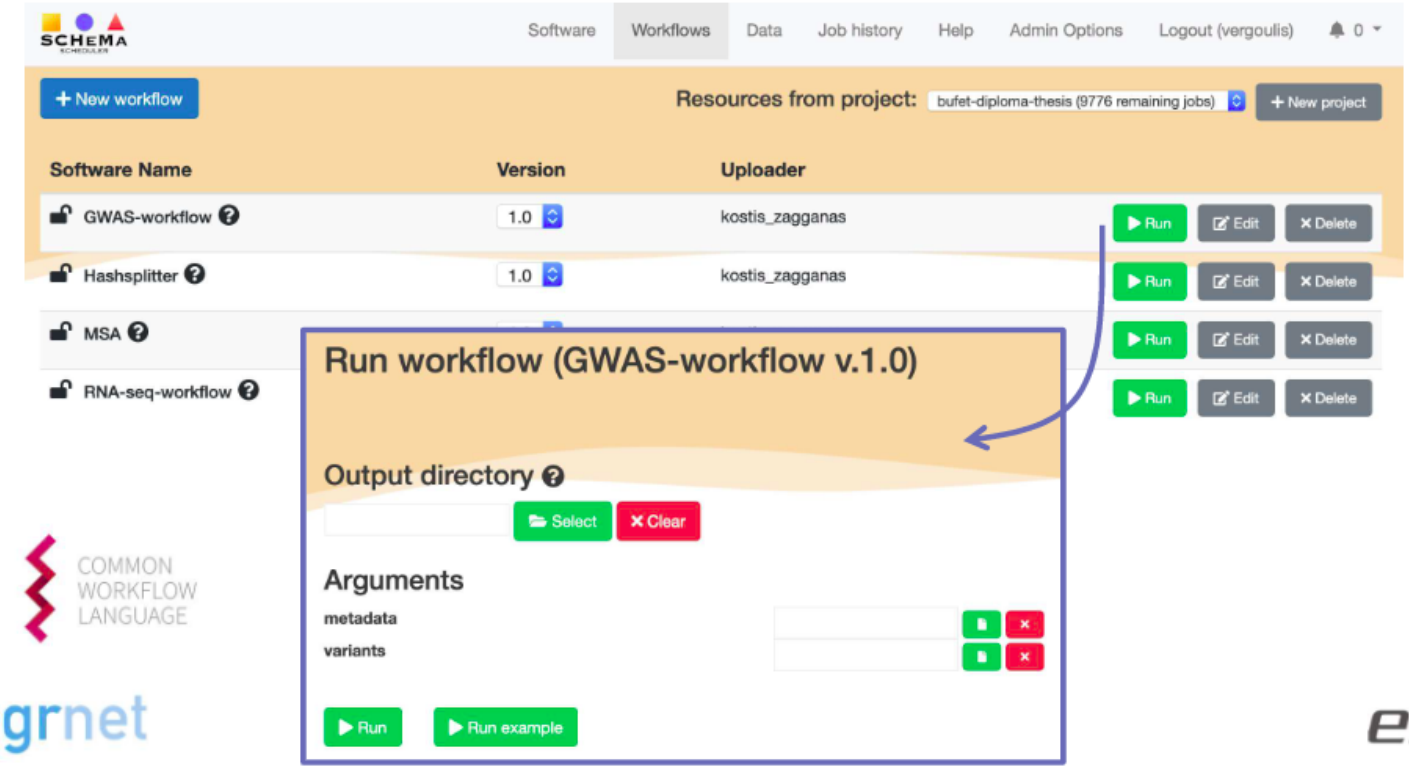
Execution

Monitoring

Dive in monitoring

A. Dashboard from other RIs

Dive in monitoring

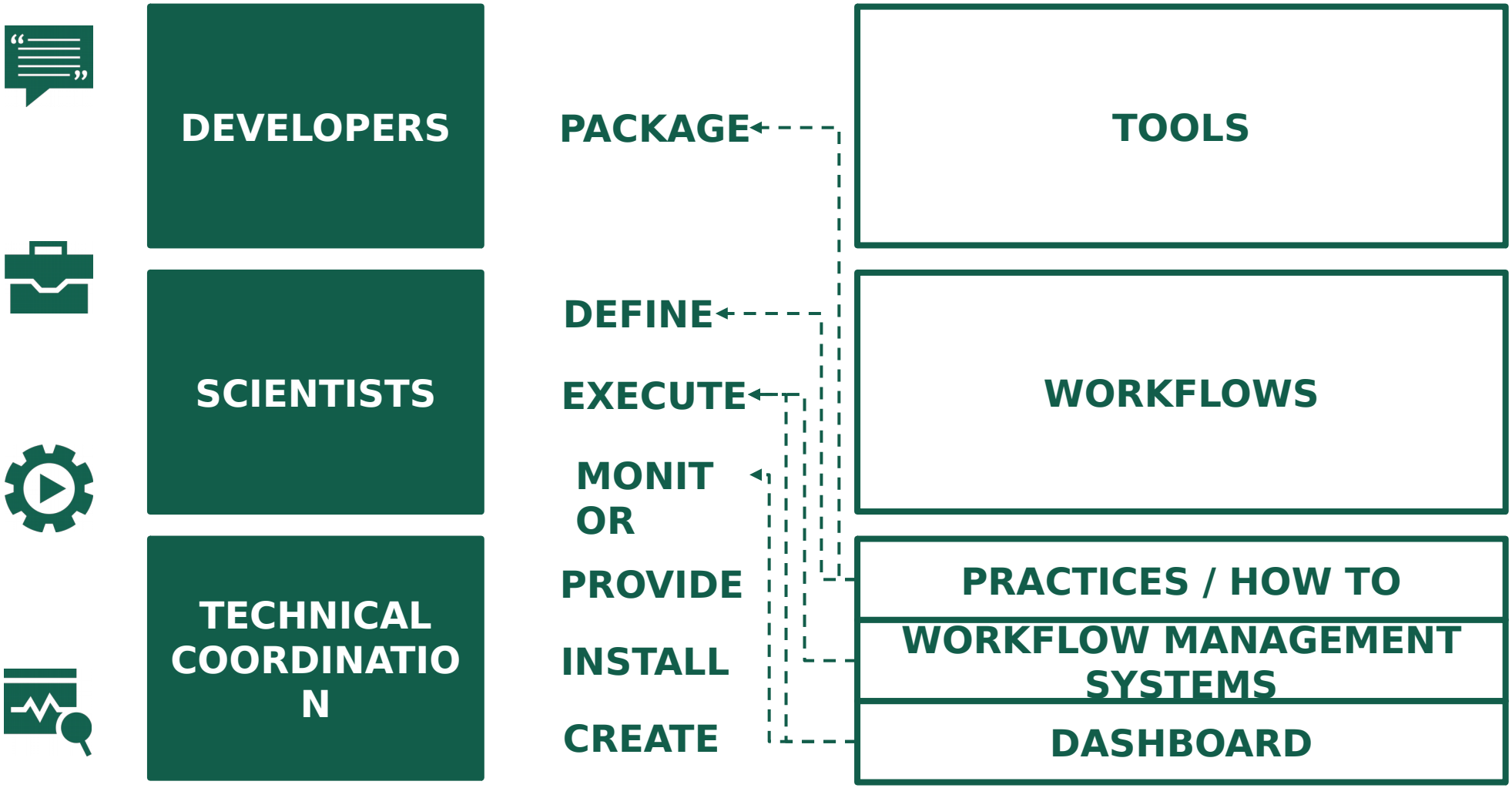


COMMON WORKFLOW LANGUAGE

grnet

elixir GREECE

Our Vision


DEVELOPERS
SCIENTISTS
TECHNICAL COORDINATION
PACKAGE
DEFINE
EXECUTE
MONITOR
PROVIDE
INSTALL
CREATE
TOOLS
WORKFLOWS
PRACTICES / HOW TO
WORKFLOW MANAGEMENT SYSTEMS
DASHBOARD

Our vision(short term)

COMMON WORKFLOW LANGUAGE

SCIENTISTS

docker

DEVELOPERS

DEVELOPERS + TECHNICAL COORDINATION

SCIENTINSTS

Our vision (long term)

WorkflowHub

WorkflowHub | Browse | Help | Search here... | Search | Register | Log in

Workflows

Query
Search here... Go

Created At
Any time

Workflow Type
- Common Workflow Language ×
- Galaxy 72
- Jupyter 14
- Nextflow 14
- Shell Script 9
- Snakemake 3
- More...

Tag
- CWL 9
- covid-19 5
- INDELs 4
- SNPs 4
- Alignment 2
- Amplicon 2
- More...

Submitter
- Ambarish Kumar 4
- Bart Nijsse 4
- Laura Rodriguez-Navas 3
- Jasper Koehorst 3
- Irene Sánchez 3

25 Workflows matching the given criteria: (Clear all filters)
Workflow type: Common Workflow Language ×

Creation date (Ascending)

← Previous 1 2 Next →

Default | Condensed | Table

Virus genome assembly with Unicycler and Spades.

Stable
Virus genome assembly with Unicycler and Spades, The 2 assemblers works in parallel. The graph visualization is made with Bandage. workflow git repository : https://github.com/fjrmoreews/cwl-workflow-SARS-CoV-2/blob/master/Assembly/workflow/assembly-wf-virus.cwl Based on https://github.com/galaxyproject/SARS-CoV-2/blob/master/genomics/2-Assembly/as_wf.png

Download

Type: Common Workflow Language
Creators: None
Submitter: francois moreews

Created: 10th Apr 2020 at 11:45, Last updated: 30th Jun 2020 at 09:05

VIRify

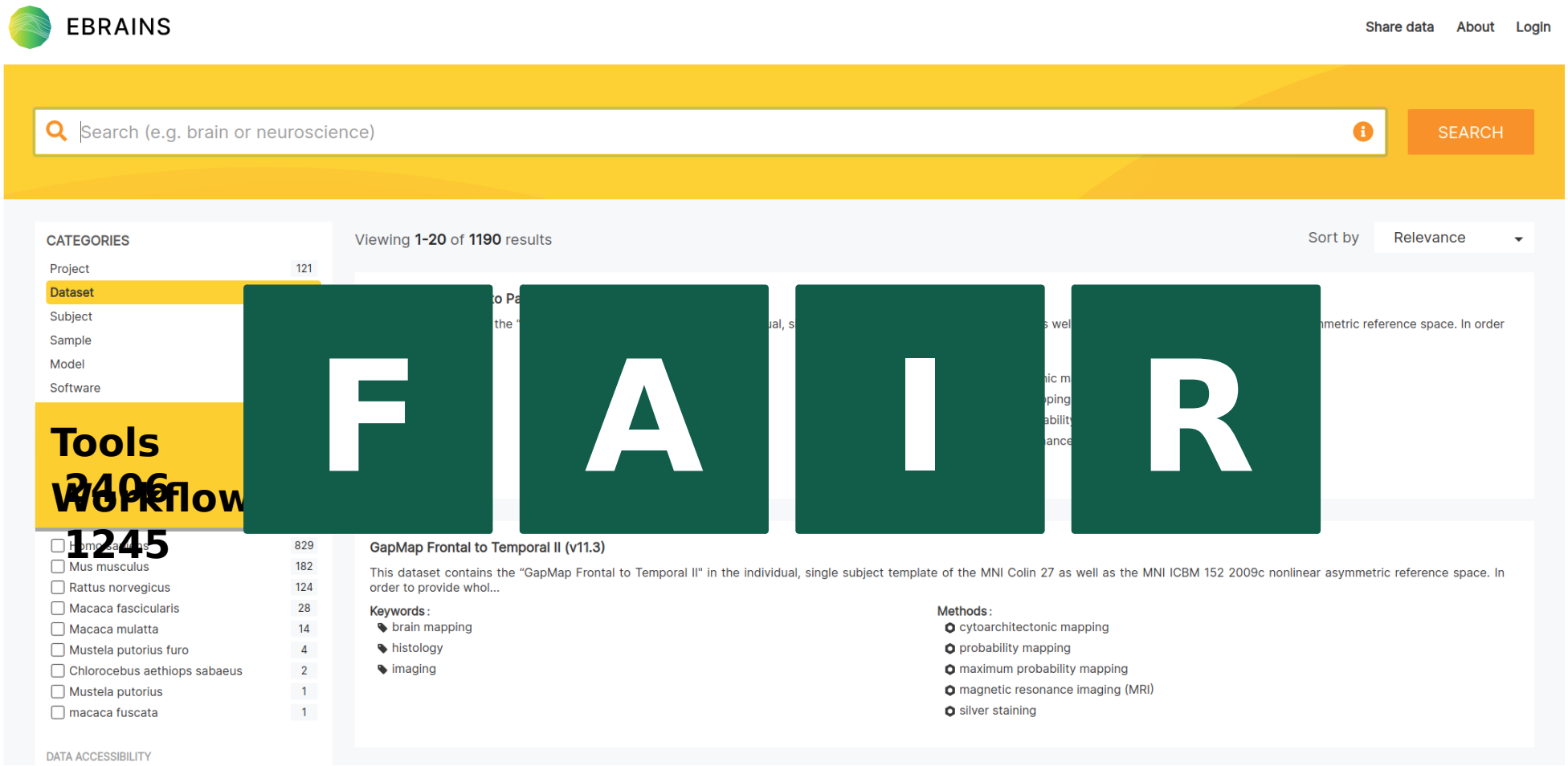
var-PE

Protein MD Setup tutorial using BioExcel Building Blocks (biobb)

[1] https://workflowhub.eu/

Our vision (long term)

A Workflows and Tools Hub



Our vision
Monitoring + Executing

Dashboard

In the making…

Human Brain Project

EBRAINS

Thank You!

www.humanbrainproject.eu	www.ebrains.eu