Facing Compute Platform Portability

Challenges with Scientific Workflows

*Experiences from Common Workflow Language*

Stian Soiland-Reyes

eScience lab, The University of Manchester

@soilandreyes

https://orcid.org/0000-0001-9842-9718

*BioExcel/MolSSI symposium, PASC18*

2018-07-03 Basel, CH

This work has been done as part of the BioExcel CoE (www.bioexcel.eu), a project funded by the European Union contract H2020-EINFRA-2015-1-675728.
Automation
- Automate computational aspects
- Repetitive pipelines, sweep campaigns

Scaling – compute cycles
- Make use of computational infrastructure & handle large data

Abstraction – people cycles
- Shield complexity and incompatibilities
- Report, re-use, evolve, share, compare
- Repeat – Tweak - Repeat
- First class commodities

Provenance - reporting
- Capture, report and utilize log and data lineage auto-documentation
- Traceable evolution, audit, transparency
- Compare

Why use workflows?

Findable
Accessible
Interoperable
Reusable
(Reproducible)
https://s.apache.org-existing-workflow-systems
Existing Workflow systems

Michael R. Crusoe edited this page 14 days ago · 186 revisions

Computational Data Analysis Workflow Systems

Permalink: https://s.apache.org/existing-workflow-systems

An incomplete list

Please add new entries at the bottom.

See also: https://github.com/pditommaso/awesome-pipeline

1. Arvados http://arvados.org
2. Taverna http://www.taverna.org.uk/
4. SHIWA https://www.shiwa-workflow.eu/
5. Oozie https://oozie.apache.org/
   https://wiki.dnanexus.com/API-Specification-v1.0.0/Workflows-and-Analyses#
8. Agave http://agaveapi.co/live-docs/
10. Wings http://www.wings-workflows.org/
208. S4M https://bitbucket.org/uqokorn/s4m_base/wiki/Home
211. phpflot https://github.com/phpflo/phpflo
213. Tavaxy: Pattern based workflow system for the bioinformatics domain http://www.tavaxy.org/
214. Ginflow: Decentralised adaptive workflow engine https://ginflow.inria.fr/
cwlVersion: v1.0
class: Workflow
inputs:
  inp: File
  ex: string
outputs:
  classout:
    type: File
    outputSource: compile/classfile
steps:
  untar:
    run: tar-param.cwl
    in:
      tarfile: inp
      extractfile: ex
    out: [example_out]
  compile:
    run: arguments.cwl
    in:
      src: untar/example_out
    out: [classfile]
Which CWL engine runs where?

**cwltool:** Local (Linux, OS X, Windows)

**Arvados:** AWS, GCP, Azure, Slurm

**Toil:** AWS, Azure, GCP, Grid Engine, LSF, Mesos, OpenStack, Slurm, PBS/Torque

**Rabix Bunny:** Local (Linux, OS X), GA4GH TES

**cwl-tes:** Local, GCP, AWS, HTCondor, Grid Engine, PBS/Torque, Slurm

**CWL-Airflow:** Linux, OS X

**REANA:** Kubernetes, CERN OpenStack

**cromwell:** local, HPC, Google, HtCondor

**CWLEXEC:** IBM Spectrum LSF

**XENON:** any Xenon backend: local, ssh, SLURM, Torque, Grid Engine
Task Execution Service (TES)

**Purpose:** A way to send a request to run a CWL or WDL-described Docker-based tool (or other tool) in a remote environment, monitor progress, & retrieve the result.

![Diagram of TES components: Docker Tools, JSON, status, stderr, stdout, file(s), API Standard to Execute (POST new task, GET task status, GET task stderr/stdout), and Cloud-specific Implementation.]

http://ga4gh.cloud/
Abstract

One way to define a workflow is: an analysis task represented by a directed graph describing a sequence of operations that transform an input data set to output. This specification defines the Common Workflow Language (CWL) Workflow description, a vendor-neutral standard for representing workflows intended to be portable across a variety of computing platforms.
Over 5000 CWL Descriptions on GitHub

5,706 code results

IBMSpectrumComputing/cwlexec – cwl_no_class.cwl
Showing the top match  Last indexed on Apr 18
1  cwlversion: v1.0

IBMSpectrumComputing/cwlexec – cwl_unsupported_version.cwl
Showing the top match  Last indexed on Apr 18
1  cwlVersion: v2.0
2  class: CommandLineTool

rekols/recat-emacs – cwl.cwl
Showing the top match  Last indexed on May 3
1  cwlVersion: v1.0
Workflow: lobSTR-workflow.cwl

Workflow Inputs:
- p2
- p1
- rg-sample
- rg-lib
- noise_model
- reference
- output_prefix
- strinfo

Workflow Outputs:
- bam_stats
- bam
- vcf_stats
- vcf
IBM Spectrum LSF Suites

cwlexec

cwlexec implements running CWL (Common Workflow Language) workflows on IBM Spectrum LSF. It is written in Java and tested for Java 8, with the following features:

- Tight integration with IBM Spectrum LSF
- Leverages LSF features (such as native container support)
- Implements CWL draft-3 and v1.0 with a few exceptions (SoftwareRequirement, ExpressionTool, include directive, remote location in File/Directory specification)

https://github.com/IBMSpectrumComputing/cwlexec
cwlexec

- Efficient checking of job completion with maximum parallelism
- Support LSF submission (bsub) options
- Self-healing of workflows
- Docker integration
- Cloud bursting
- Rerun and interruption

https://youtu.be/_jSTZMWtPAY
{  "queue": "high",  "steps": {    "step1": {      "app": "dockerapp"    },    "step2": {      "res_req": "select[type==X86_64] order[ut] rusage[mem=512MB:swp=1GB:tmp=500GB]"    }  }}
Hello!

This guide will introduce you to writing tool wrappers and workflows using the Common Workflow Language (CWL). This guide describes the current stable specific 1.0.

Note: This document is a work in progress. Not all features are covered, yet.

**Prerequisites**

A text editor

A CWL runner. It is recommended to start with the reference implementation. The full list of CWL runners is on the project homepage.

**Schedule**

<table>
<thead>
<tr>
<th>Time</th>
<th>Setup</th>
<th>Download files required for the lesson</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00</td>
<td>1. Introduction</td>
<td>What is Common Workflow Language? Why might I want to learn to use CWL?</td>
</tr>
<tr>
<td>00:00</td>
<td>2. First Example</td>
<td>How do I wrap a simple command line tool?</td>
</tr>
<tr>
<td>00:05</td>
<td>3. Essential Input Parameters</td>
<td>How do I describe inputs to a command? How do I specify the order in which inputs appear in a command?</td>
</tr>
<tr>
<td>00:15</td>
<td>4. Returning Output Files</td>
<td>How do I describe outputs from a command?</td>
</tr>
<tr>
<td>00:25</td>
<td>5. Capturing Standard Output</td>
<td>How do I capture a tool’s standard output stream?</td>
</tr>
</tbody>
</table>
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: Workflow
inputs:
  inp: File
  ex: string

outputs:
  classout:
    type: File
    outputSource: compile/classfile

steps:
  untar:
    run: tar-param.cwl
    in:
      tarfile: inp
      extractfile: ex
    out: [example_out]

  compile:
    run: arguments.cwl
    in:
      src: untar/example_out
    out: [classfile]
cwlVersion: v1.0
class: Workflow
label: EMG QC workflow, (paired end version). Benchmarking with MG-RAST expt.

requirements:
  - class: SubworkflowFeatureRequirement
  - class: SchemaDefRequirement
    types:
      - $import: ../tools/FragGeneScan-model.yaml
      - $import: ../tools/trimmomatic-sliding_window.yaml
      - $import: ../tools/trimmomatic-end_mode.yaml
      - $import: ../tools/trimmomatic-phred.yaml

inputs:
  reads:
    type: File
    format: edam:format_1930 # FASTQ

outputs:
  processed_sequences:
    type: File
    outputSource: clean_fasta_headers/sequences_with_cleaned_headers

steps:
  trim_quality_control:
**Workflow Inputs**

- **reads**
- [Complex Object] 100 "33" 3 3 "SE"

**Workflow Outputs**

- replace problem characters from FASTA headers with dashes
- processed_sequences

**Workflow Diagram**

1. **trim_quality_control**
   - reads1
   - slidingwindow
   - minlenphred
   - leading
   - trailing
   - end_mode

2. **convert_trimmed-reads_to_fasta**
   - fastq
   - sequences

3. **replace problem characters from FASTA headers with dashes**

---

https://w3id.org/cwl/view/git/886df9de6713e06228d2560c40f451155a196383/workflows/emg-qc-single.cwl
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: [tar, xf]
inputs:
  tarfile:
    type: File
    inputBinding:
      position: 1
outputs:
  example_out:
    type: File
    outputBinding:
      glob: hello.txt
Finding the tool

class: CommandLineTool

hints:
  SoftwareRequirement:
    packages:
      samtools:
        version: [ "0.1.19" ]

baseCommand: ["samtools", "index"]
#..

apt-get install samtools=0.1.19*

conda install samtools=0.1.19

module load samtools/0.1.19

https://www.commonwl.org/v1.0/CommandLineTool.html#SoftwarePackage

https://docs.galaxyproject.org/en/latest/admin/dependency_resolvers.html
Package resolution

```xml
<dependency_resolvers>
  <modules modulecmd="/opt/bin/modulecmd" />
  <tool_shed_packages />
  <galaxy_packages />
  <conda />

  <modules modulecmd="/opt/bin/modulecmd" versionless="true" />
  <galaxy_packages versionless="true" />
  <conda versionless="true"/>
</dependency_resolvers>
```

Dependency resolution by **CWLTool** and **Toil**

https://docs.galaxyproject.org/en/latest/admin/dependency_resolvers.html
Where to find command line tools?

- Docker: https://www.docker.com/
- Singularity: http://singularity.lbl.gov/
- Bioconda: https://bioconda.github.io/
- Galaxy Tool Shed: https://toolshed.g2.bx.psu.edu/
cwlVersion: v1.0
class: CommandLineTool
baseCommand: node
hints:
  DockerRequirement:
    dockerPull: mgibio/samtools:1.3.1

https://hub.docker.com/
<table>
<thead>
<tr>
<th>Repository</th>
<th>Stars</th>
<th>Pulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>mgibio/samtools</td>
<td>0</td>
<td>10K+</td>
</tr>
<tr>
<td>kfdrc/samtools</td>
<td>0</td>
<td>8.1K</td>
</tr>
<tr>
<td>maxulysse/samtools</td>
<td>1</td>
<td>5.6K</td>
</tr>
<tr>
<td>comics/samtools</td>
<td>0</td>
<td>2.2K</td>
</tr>
<tr>
<td>z/lskidmore/samtools</td>
<td>0</td>
<td>10K+</td>
</tr>
<tr>
<td>biocontainers/samtools</td>
<td>2</td>
<td>10K+</td>
</tr>
</tbody>
</table>
CONDA
= conda
  + python
  + base packages

MINICONDA
= miniconda
  + 150 high quality packages

ANAconda

BIOCONDA
= anaconda
  + 4000 bioinformatics packages

https://bioconda.github.io/
Let's add some identifiers!

hints:
SoftwareRequirement:
  packages:
    - package: bowtie
      version:
        - '2.2.8'
  specs:
    - https://packages.debian.org/bowtie
    - https://anaconda.org/bioconda/bowtie
    - https://bio.tools/tool/bowtie2/version/2.2.8
    - https://identifiers.org/rrid/RRID:SCR_005476
    - https://hpc.example.edu/modules/bowtie-tbb/2.2

https://www.commonwl.org/v1.0/CommandLineTool.html#SoftwarePackage
Khan et al, CWLProv – Interoperable retrospective provenance capture and its challenges, BOSC 2018

https://doi.org/10.7490/f1000research.1115721.1

https://doi.org/10.5281/zenodo.1208477
prefix wfprov <http://purl.org/wf4ever/wfprov#>
prefix prov <http://www.w3.org/ns/prov#>
prefix wfdesc <http://purl.org/wf4ever/wfdesc#>
prefix wf <https://w3id.org/cwl/view/git/933bf2a1a1c832d88f88f1362755535a9d9f0954/workflow/>
prefix input <app://579c1b74-b328-4da6-80a8-a2ffef2ac9b5/workflow/input.json#>
prefix run <urn:uuid:>
prefix engine <urn:uuid:>
prefix data <urn:hash:sha256:>
default <app://579c1b74-b328-4da6-80a8-a2ffef2ac9b5/>

// Level 1 provenance of workflow run
activity(run:2e1287e0-6dfb-1e7-8acf-0242ac110002, , , [prov:type='wfprov:WorkflowRun'])
wasStartedBy(run:2e1287e0-6dfb-1e7-8acf-0242ac110002, -, -, -, 2017-10-27T14:24)

// The engine is the SoftwareAgent that is executing our Workflow plan
wasAssociatedWith(run:2e1287e0-6dfb-1e7-8acf-0242ac110002, engine:b2210211-8acb)
agent(engine:b2210211-8acb-4d58-bd28-2a36b18d3bd4f, prov:type='prov:SoftwareAgent')

// prov has no term to relate sub-plans - we'll use wfdesc:hasSubProcess
entity(wf:main,[prov:type='wfdesc:Workflow', prov:type='prov:Plan', wfdesc:hasSubProcessOf(wf:main, workflow/packed.cwl)]
entity(wf:main/step1,[prov:type='wfdesc:Process', prov:type='prov:Plan'])
entity(wf:main/step2,[prov:type='wfdesc:Process', prov:type='prov:Plan'])

// First the workflow uses some data; here with a urn:sha:sha256 identifier
used(run:2e1287e0-6dfb-1e7-8acf-0242ac110002, data:5891b5b522d5df086d0ff0b110fb
entity(data:5891b5b522d5df086d0ff0b110fb9d21bbd4fc7163af34d08286a2e846f6be03)
// which we have stored a copy of within the research object
specializationOf(data:58/5891b5b522d5df086d0ff0b110fb9d21bbd4fc7163af34d08286a2e846f6be03)

// Then there was another activity - wfprov:ProcessRun indicating a command line
activity(run:4305467e-6dfb-1e7-885d-0242ac110002, -, -, [prov:type='wfprov:ProcessRun'])
Using provenance to improve performance
Challenges
Determining hardware allocations

Automatic rescale+retry
Configuration or Prediction?
Machine learning from provenance
Scheduling scattered jobs

Can't determine total number of jobs until runtime
..but usually early on you can find the number
Accessing the scheduler

How can tasks influence their own allocations?

Lesson learnt from cloud approach:
Avoid "master" node dependency

Workarounds: Task Service (TES), worker nodes
Moving workflows to Exascale

Don't throw the baby out with the bathwater!

- Automation (*but is is interoperable?*)
- Scalability (*faster! bigger!*)
- Abstraction (*can humans still understand it?*)
- Provenance (*what actually ran?*)

- Findable
- Accessible
- Interoperable
- Reproducible