COMMON WORKFLOW LANGUAGE

Common Workflow Language v1.0: what it means for the Galaxy Community

Adapted from Peter Amstutz’s presentation, licensed CC-BY-SA
Existing Workflow systems

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/
11. KNIME https://www.knime.org/
12. make, rake, drake, ant, scons & many others. Software development relies heavily on tools to manage workflows related to compiling and packaging applications. For the most part these are file based and usually run on a single node, usually supporting parallel steps (make -i) and in some cases able to dispatch build steps to other machines (https://code.google.com/p/distcc/)
   https://github.com/Factual/drake
13. Snakemake https://bitbucket.org/johanneskoester/snakefile
14. BPipe http://bpipe.org
15. Rufus https://code.google.com/p/rufus/
17. Luigi http://github.com/spotify/luigi
18. SciLuigi. Helper library built on top of Luigi to ease development of Scientific workflows in Luigi:
   http://github.com/camil/SciLuigi
21. SeqWare Workflows are written in Java and executed using the Oozie Workflow Engine on
Hadoop or SGE clusters. Uses Zip64 files to group the workflow definition file, workflow itself,
sample settings, and data dependencies in a single file that can be exchanged between
SeqWare users or archived. https://seqware.github.io/ https://seqware.github.io/docs/6-pipeline/
23. Pegasus http://pegasus.isi.edu/
   http://bioinformatics.oxfordjournals.org/content/early/2014/07/24/bioinformatics.382682.full [paper]
   Cosmos2: https://github.com/LFM-HMS/COSMOS2 http://cosmos.hms.harvard.edu/COSMOS2/
31. flowr (R-based) http://docs.flowr.space/ https://github.com/sahilseth/flowr
32. Mistral https://github.com/arteria-project
   https://wiki.openstack.org/wiki/Mistral#What_is_Mistral?3f
   https://wiki.openstack.org/wiki/Mistral/DSLv2
33. nipype http://nipype.org/nipype/
34. End of Day https://github.com/joestubbs/endofday
35. BioDSL https://github.com/maasha/BioDSL
37. OmicsPipe: uses Ruffus http://sulab.scripps.edu/omicspipe/
38. Ensembl Hive https://github.com/Ensembl/ensembl-hive
39. QuickNGS http://bifacility.uni-koeln.de/quickngs/web
41. Chipster http://chipster.csc.fi/
42. The Genome Modeling System https://github.com/chooseggs
   http://www.cuneiform-lang.org/
   http://webapp.cabgrid.res.in/biocomp/Anvaya/ANVAYA_Main.html#HOWTO_INSTALL_ANVAYA
45. Makeflow http://ccl.cse.nd.edu/software/makeflow/
46. Airavata http://airavata.apache.org/
Existing computational research workflow systems

Adapted from Peter Amstutz's presentation, licensed CC-BY-SA

54. Job Description Language. The Job Description Language, JDL, is a high-level, user-oriented language based on Condor classified advertisements for describing jobs and aggregates of jobs such as Direct Acyclic Graphs and Collections. https://adms.cern.ch/file/59069/UIWMS-JDL.pdf


57. Kronos https://github.com/jtaghiyar/kronos


59. YesWorkflow http://yesworkflow.org


62. NGLess https://github.com/uispedro/ngless

63. pypipegraph https://github.com/TyberiusPrime/pypipegraph

64. Cromwell https://github.com/broadinstitute/cromwell


66. sushi https://github.com/uzh/sushi

Why have a standard?

- Standards create a surface for collaboration
- Funders, journals, and other sources of incentives prefer standards over proprietary approaches
- Research dip in and out of Galaxy all the time, might as well make it easier to do so
Common Workflow Language

- Common format for bioinformatics (and more!) tool & workflow execution
- Community based standards effort, not a specific software package
- Defined with a schema, specification & test suite
- Designed for shared-nothing clusters, academic clusters, cloud environments, and local execution
- Supports the use of containers (e.g. Docker)
Why use the Common Workflow Language?

Develop your pipeline on your laptop (optionally with Docker)

Execute on your research cluster or in the cloud

Deliver to users via Workbench like Arvados, Rabix, and soon Galaxy & Taverna.
Implementations

- cwltool (reference implementation)
- Rabix from Seven Bridges Genomics
- Arvados from Curoverse
- Galaxy
- Toil (cloud or SLURM clusters)
- Airflow (SciDAP)
- Google Pipeline Engine cwl-tool wrapper
Design principles

- Low barrier to entry for implementers
- Support tooling such as generators, GUIs, converters
- Allow extensions, but must be well marked
- Be part of linked data ecosystem
- Be pragmatic
LINKED DATA & CWL

- Hyperlinks are common currency
- Use RDF ontologies for metadata
- Support SPARQL to query
- “Data model not a file format”

Example: can use the EDAM ontology (ELIXIR-DK) to specify file formats and reason about them: FASTQ Sanger encoding is a type of FASTQ file, ...
Early Adopters

(US) National Cancer Institute Cloud Pilots (Seven Bridges Genomics, Institute for Systems Biology)

Cincinnati Children’s Hospital Medical Research Center (Andrey Kartashov & Artem Barski)

bcbio: Validated, scalable, community developed variant calling, RNA-seq and small RNA analysis (docs) (Brad Chapman et al.)

Duke University, Center for Genomic and Computational Biology: GENOMICS OF GENE REGULATION project (BOSC ‘16 abstract) (Dan Leehr et al.)

NCI DREAM SMC–RNA Challenge (Kyle Ellrott et al.)

Adapted from Peter Amstutz’s presentation, licensed CC-BY-SA
Early Adopter: bcbio

runs locally, on Microsoft Azure (via Curoverse), or on any other CWL compatible platform

https://bcbio-nextgen.readthedocs.io

Adapted from Peter Amstutz’s presentation
Example: samtools sort

```yaml
class: CommandLineTool
cwlVersion: draft-3
description: Sort by chromosomal coordinates

requirements:
- class: DockerRequirement
dockerPull: scidap/samtools:v1.2-216-gdfcc67f

inputs:
- id: input
type: File
inputBinding:
  position: 1

- id: output_name
type: string
inputBinding:
  position: 2

outputs:
- id: output
type: File
outputBinding:
  glob: $(inputs.output_name)

baseCommand: [samtools, sort]
```

Adapted from Peter Amstutz's presentation, licensed CC-BY-SA
Identify as a CommandLineTool object

Core spec includes simple comments

Metadata about tool extensible to arbitrary RDF vocabularies, e.g.
  - Biotools & EDAM
  - Dublin Core Terms (DCT)
  - Description of a Project (DOAP)

GA4GH Tool Registry project will develop best practices for metadata & attribution
Runtime environment

requirements:
- class: DockerRequirement
dockerPull: scidap/samtools:v1.2-216-gdffc67f

- Define the execution environment of the tool
- Must be fulfilled or an error
- "hints" are soft requirements (express preference but not an error if not satisfied)
- Also used to enable optional CWL features
  - Mechanism for defining extensions
## Example: samtools sort

<table>
<thead>
<tr>
<th><strong>class:</strong> CommandLineTool</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cwlVersion:</strong> draft-3</td>
</tr>
<tr>
<td><strong>description:</strong> Sort by chromosomal coordinates</td>
</tr>
</tbody>
</table>

**requirements:**
- **class:** DockerRequirement
  - **dockerPull:** scidap/samtools:v1.2-216-gdffc67f

**inputs:**
- **id:** input
  - **type:** File
    - **inputBinding:**
      - **position:** 1

- **id:** output_name
  - **type:** string
    - **inputBinding:**
      - **position:** 2

**outputs:**
- **id:** output
  - **type:** File
    - **outputBinding:**
      - **glob:** $(inputs.output_name)

**baseCommand:** [samtools, sort]
Input parameters

inputs:
- id: input
type: File
inputBinding:
  position: 1

- id: output_name
type: string
inputBinding:
  position: 2

- Specify name & type of input parameters
- Based on the Apache Avro type system
- null, boolean, int, string, float, array, record
- "inputBinding": describes how to turn parameter value into actual command line argument
Output parameters

```
outputs:
  - id: output
    type: File
    outputBinding:
      glob: $(inputs.output_name)
```

- Specify name & type of output parameters
- “outputBinding”: describes how to capture the output of the tool and fill in the value of the parameter
  - In this example, search the designated output directory for the file named in the “output_name” parameter
Command Line Building

- Associate input values with parameters
- Apply input bindings to generate strings
- Sort by “position”
- Prefix “base command”

```json
{
  "input": {
    "class": "File",
    "path": "/files/input.bam"
  },
  "output_name": "output.bam"
}
```

```
inputs:
- id: input
type: File
  inputBinding:
    position: 1
- id: output_name
type: string
  inputBinding:
    position: 2

baseCommand: [samtools, sort]
```

```
[“samtools”, “sort”, “/files/input.bam”, “output.bam”]
```
Workflows

- Specify data dependencies between steps
- Scatter/gather on steps
- Can nest workflows in steps
- Still working on:
  - Conditionals & looping

Adapted from Peter Amstutz’s presentation, licensed CC-BY-SA
Example: grep & count

```
class: Workflow
cwlVersion: draft-3

requirements:
- class: ScatterFeatureRequirement

inputs:
- id: pattern
type: string
- id: infile
type: {type: array, items: File}

outputs:
- id: outfile
type: File
  source: "#wc/outfile"

steps:
- id: grep
  run: {"@import": grep.cwl.yaml}
  inputs:
    - id: pattern
      source: "#pattern"
    - id: infile
      source: "#infile"
    scatter: "#grep/infile"
  outputs:
    - id: outfile

- id: wc
  run: {"@import": wc.cwl.yaml}
  inputs:
    - id: infile
      source: "#grep/outfile"
  outputs:
    - id: outfile
```

Tool to run

Scatter over input array

Connect output of "grep" to input of "wc"

Connect output of "wc" to workflow output

Adapted from Peter Amstutz's presentation, licensed CC-BY-SA
Thanks!

http://commonwl.org

Adapted from Peter Amstutz's presentation, licensed CC-BY-SA