Andreas Wilke and Folker Meyer

COMMON
WORKFLOW
LANGUAGE

Brief CWL intro with some AWE specifics

Original slides from
Michael R. Crusoe, CWL Community Engineer

Michael R. Crusoe, who is this guy?

From Phoenix, Arizona (Sonoran Desert), USA

Studied at Arizona State: Comp. Sci.; time in industry as a developer & system administrator (Google, others); returned to ASU & received B.S. in Microbiology.

Co-author of an identity card **standard** for use by seafarers; accompanying ILO convention ratified by 30+ countries

Introduced to bioinformatics via Anolis (lizard) genome assembly and analysis (Kenro Kusumi, Arizona State)

Returned to software engineering as a Research Software Engineer for k-h-mer project (C. Titus Brown, Michigan State, then U. of California, Davis)

Now based out of Vilnius, Lithuania

“Workflows”

We use the word “workflows” as a shorthand for:

the collection of computer applications, scripts, and code used in **computational data analysis**

- how the applications are configured
- and how the data flows between them

(primarily in a research/scientific context)

Why use a workflow management system?

Features **can** include:

separation of concerns: focus on the science being done first; then optimize execution later
automatic job execution: start a complicated analysis involving many pieces with a single command
scaling (across nodes, clusters, and possibly continents)
automatically generated graphical user interfaces
(example: Galaxy)
How was this file made? (**automatic provenance tracking**)

EBI's metagenomics workflow scripts -> CWL

https://www.ebi.ac.uk/metagenomics/pipelines/3.0

9522 lines of Python, BASH, and Perl code (data analysis workflows logic mixed with operational details

converted into

2560 lines of CWL descriptions

https://github.com/ProteinsWebTeam/ebi-metagenomics-cwl

(Lines of code counts via https://github.com/AlDanial/cloc#Stable)

EBI's metagenomics -> CWL project

Courtesy EMBL-EBI Metagenomics, visualization from

https://view.commonwl.org/workflows/github.com/ProteinsWebTeam/ebi-metagenomics-cwl/blob/master/workflows/rna-selector.cwl

Existing computational research workflow systems

Over 200 different workflow systems

Existing Workflow systems

John Pellman edited this page 14 hours ago · 162 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/
3. Galaxy http://galaxyproject.org/
4. SHIWA https://www.shiwa-workflow.eu/
5. Oozie https://oozie.apache.org/
6. DNANexus https://wiki.dnanexus.com/API-Specification-v1.0.0/IO-and-Run-Specifications# https://wiki.dnanexus.com/API-Specification-v1.0.0/Workflows-and-Analyses#
7. BioDT http://www.biodatomics.com/
8. Agave http://agaveapi.co/live-docs/
9. DiscoveryEnvironment http://www.iplantcollaborative.org/ci/discovery-environment
10. Wings http://www.wings-workflows.org/

https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems

Why have a standard?

- Standards create a surface for collaboration that promote innovation
- Research frequently dip in and out of different but interoperability is not a basic feature.
- Funders, journals, and other sources of incentives prefer standards over proprietary or single-source approaches

Common Workflow Language v1.0

- Common format for tool & workflow execution
- Community based standards effort, not a specific software package; **Very extensible**
- 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) and shared research computing clusters with locall installed software

Participating Organizations

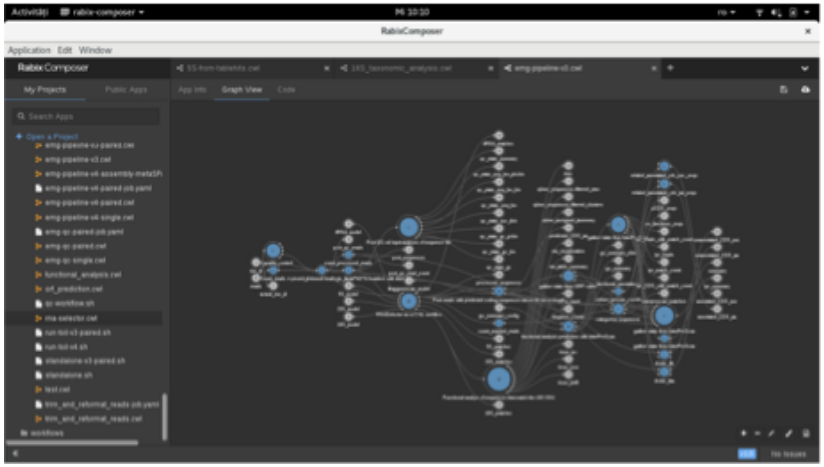
Why use the Common Workflow Language?

Develop your pipeline on your local computer (optionally with Docker)

Execute on your research cluster or in the cloud

Deliver to users via workbenches like Arvados, Rabix, Toil, and soon Galaxy & Taverna.

Rabix Composer: an Open Source GUI for CWL



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**

The CWL model

CWL tool descriptions turn POSIX† command-line data analysis tools into functions

- well defined and named inputs & outputs
- typed

These inputs and outputs are connected into “data flow” style workflows

†The reference CWL runner runs on Microsoft Windows using Docker software containers

Compute Resource matchmaking with CWL

CWL tool descriptions can self describe the “shape” of the computation

- # of cores
- memory needs
- temporary and output storage estimations

This uses fixed values, or can be computed prior to scheduling based upon the input data & its metadata

http://www.commonwl.org/v1.0/CommandLineTool.html#Runtime_environment

Matching jobs to workers in AWE

- AWE servers support multiple queues
- Clients subscribe to work in specific queues
  `--group=docker` (docker in our example)

Example:
`--group=highmen` (job requiring >128GB RAM per node)
`--group=default` (jobs under <128GB RAM)

- Create a specific queue for a worker with e.g. database preloaded into memory or FPGA accelerator
- Workers in queues have capabilities (asterisk in our example)
  `--supported_apps=*`
  - Capabilities can overlap between queues
  - Authentication token per group

Data locality with CWL

Input and output files are modeled in CWL as rich object with identifier (URI/IRI) and other metadata.

Platforms that understand CWL can use these identifiers to send compute to where or near the location of data.

(This is compatible with the “data lakes” proposals from yesterday)

In combination with the resource matchmaking this can conversely result in data being sent to specialized compute as configured by the operator (or machine learning)

Community Based Standards development

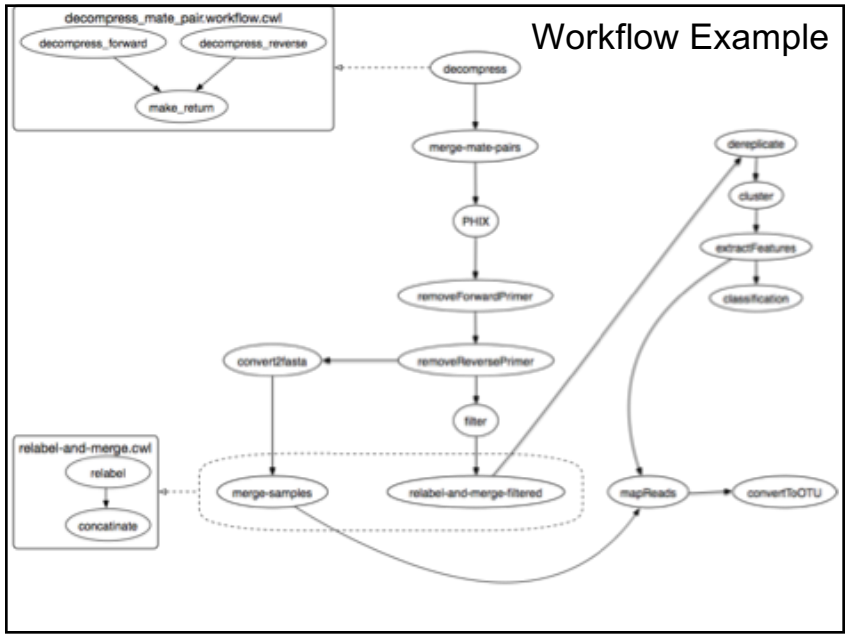
Different model than traditional nation-based or regulatory approach

We adopted the Open-Stand.org Modern Paradigm for Standards: Cooperation, Adherence to Principles (Due process, Broad consensus, Transparency, Balance, Openness), Collective Empowerment, (Free) Availability, Voluntary Adoption

CWL - Main Concepts

- Reusable **tool** description
- Reusable **workflow** description
- Specific workflow input (**job**) to execute on defined data

All descriptions either in YAML or JSON format



Example: samtools-sort.cwl

File type & metadata

```
class: CommandLineTool
cwlVersion: v1.0
doc: find lines with pattern in file
```

Runtime environment

```
hints:
  DockerRequirement:
    dockerPull: quay.io/grep:latest
```

Input parameters

```
inputs:
  pattern:
    type: string
    inputBinding: {position: 0}
  file_to_search:
    type: File
    inputBinding: {position: 1}
```

Executable

```
baseCommand: [grep]
```

Output parameters

```
outputs:
  results: stdout
```

File type & metadata

```
class: CommandLineTool
cwlVersion: v1.0
doc: Some comment
```

- Identify as a CommandLineTool object
- Core spec includes simple comments

Runtime Environment

```
hints:
  DockerRequirement:
    dockerPull: quay.io/grep:latest
```

- Define the execution environment of the tool
- “requirements” 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

Input parameters

```
inputs:
  pattern:
    type: string
    inputBinding: {position: 0}
  file_to_search:
    type: File
    inputBinding: {position: 1}
```

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

Command Line Building

```
"pattern"          : "MAM"
"file_to_searc"    : {
    "class"        : "File"
    "path"         : "./models.xml"
}
```

```
inputs:
  pattern:
    type: string
    inputBinding: {position: 0}
  file_to_searc:
    type: File
    inputBinding: {position: 1}
```

```
baseCommand: ['grep']
```

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

```
["grep", "MAM", "./models.xml"]
```

Output parameters

```
outputs:
  results: stdout
  format: tsv
```

- Specify name & type of output parameters
- In this example, capture the STDOUT stream from "grep" and tag it as being tsv formatted.

Workflows

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

Example: grep & count

```
class: Workflow
cwlVersion: v1.0
```

```
requirements:
  - class: ScatterFeatureRequirement
```

```
inputs:
  pattern: string
  infiles: File[]
```

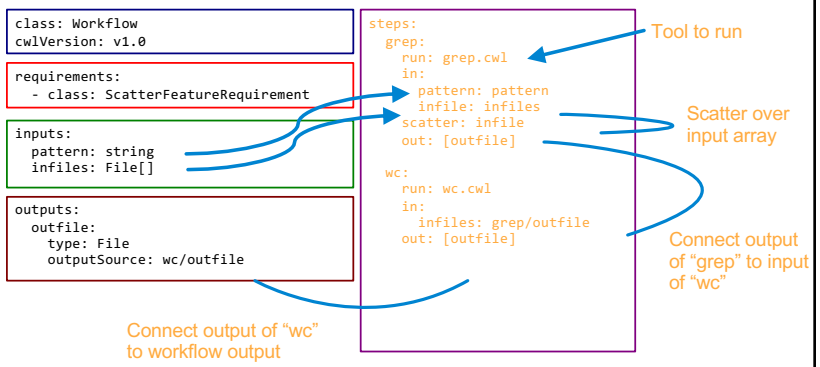
```
outputs:
  outfile:
    type: File
    outputSource: wc/outfile
```

```
steps:
  grep:
    run: grep.cwl
    in:
      pattern: pattern
      infile: infiles
    scatter: infile
    out: [outfile]

  wc:
    run: wc.cwl
    in:
      infiles: grep/outfile
    out: [outfile]
```

Source file: https://github.com/common-workflow-language/workflows/blob/2855f2c3ea875128ff62101295897d8d11d99b94/workflows/presentation-demo/grep-and-count.cwl

Example: grep & count

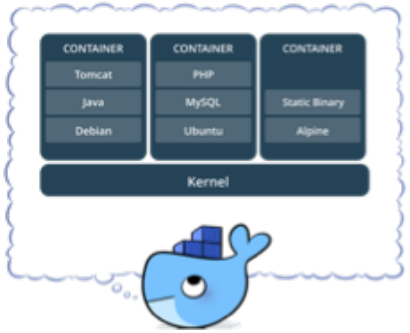


Container

A container image is

- lightweight
- stand-alone
- executable

package of a piece of software that includes everything needed to run it



How to use:

1. Get container (piece of software with execution environment)

   ```
   docker pull CONTAINER_NAME
   ```

2. Execute container

   ```
   docker run CONTAINER_NAME
   ```

Running test within container

Running a version of mam_box_coag verification code:

SOP:

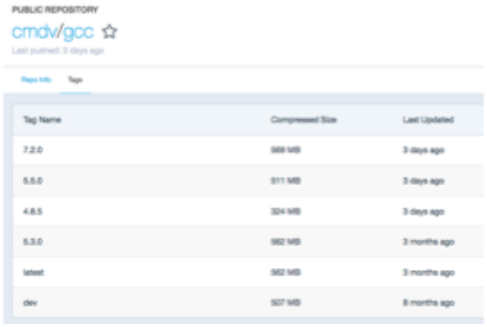
1. `git clone --recursive git@github.com:ACME-Climate/CMDV-Verification.git`
2. `cd CMDV-Verification`
3. `git checkout wilke/mam_gcc`
4. **`docker run -ti -v `pwd`:/CMDV --rm cmdv/gcc:5.3.0 bash`**
5. `cd /CMDV/tests/mam/mam_box_coag/src`
6. `bash ./coag.csh`

https://docs.google.com/document/d/1MQ1IZGNo5Ub01Is3fuyxt3MHjWitTkybzPojzc3X6U4/edit?usp=sharing

Container use cases:

1. Running prepackaged software on any hardware
2. Using as predefined development/execution environment for “own” code

We provide a couple of pre build containers at https://hub.docker.com/u/cmdv/



PS: There is a transition path from Docker to Singularity

ResearchObject.org standard overview