Common Workflow Language

Peter Amstutz
Co-founder CWL Project
Arvados project developer
Co-Chair, GA4GH DWG Containers & Workflows task team
<peter.amstutz@curoverse.com>

CUROVERSE™

Common Workflow Language (CWL)

- Common format for bioinformatics tool execution
  - Community based standards effort, not a specific software package
  - Implement CWL in your own workflow engine
  - Defined with a schema, specification & test suite
- Designed for shared-nothing cluster & cloud environments
- Designed for containers (e.g. Docker)

CUROVERSE™

CWL Participating Organizations

- Arvados Project (Curoverse)
- Broad Institute
- Cincinnati Children's Hospital
- Galaxy Project
- Harvard Chan School of Public Health
- Institut Pasteur
- Oregon Health & Science University
- Sanger Institute
- Seven Bridges
- Taverna
- UC Santa Cruz
- UC Davis

CUROVERSE™

Implementations

- cwltool (reference implementation)
- Rabix
- Arvados
- Galaxy
- Parallel Recipes
- Toil
- CancerCollaboratory
- Airflow (SciDAP)

CUROVERSE™

Design principals

- Low barrier to entry for implementers
  - Won't be cross platform if platform developers don't implement it
- Support tooling such as generators, GUIs, converters
- Allow extensions, but must be well marked
- Be part of linked data ecosystem
  - Hyperlinks are common currency
  - Use RDF ontologies for metadata
  - Support SPARQL to query
  - “Data model not a file format”
- Be pragmatic

CUROVERSE™

Tool Model & Execution

- Modeled as user defined function (UDF)
- Typed input / output signature
- Inputs & outputs are fully specified
- Tool executions are isolated from one another & from parent process
- Well defined execution process
  1. Collect & validate inputs
  2. Map input file paths to locations inside container
  3. Build tool command line
  4. Build Docker invocation
  5. Execute
  6. Collect & validate outputs

CUROVERSE™

Runtime environment

- Designated temporary and output directories
  - Only locations tool is allowed to write
  - Not shared with any other tool invocation
  - Initial working directory is output directory
- Rest of file system assumed to be read only (including input files)
- Can specify standard input/output redirection
- Don't propagate parent process environment

Example: samtools sort

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

File type & metadata

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

Runtime environment

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

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

Input parameters

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

Output parameters

```
baseCommand: [samtools, sort]
```

Executable

CUROVERSE™

File type & metadata

```
class: CommandLineTool
cwlVersion: draft-3
description: "Sort by chromosomal coordinates"
biotools:EDAMOperation: edam:operation_2121
dct:creator: Genome Research Ltd.
doap:website: https://samtools.github.io/
```

- 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

CUROVERSE™

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

CUROVERSE™

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 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

CUROVERSE™

Command Line Building

Input object

```
{
  "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]
```

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

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

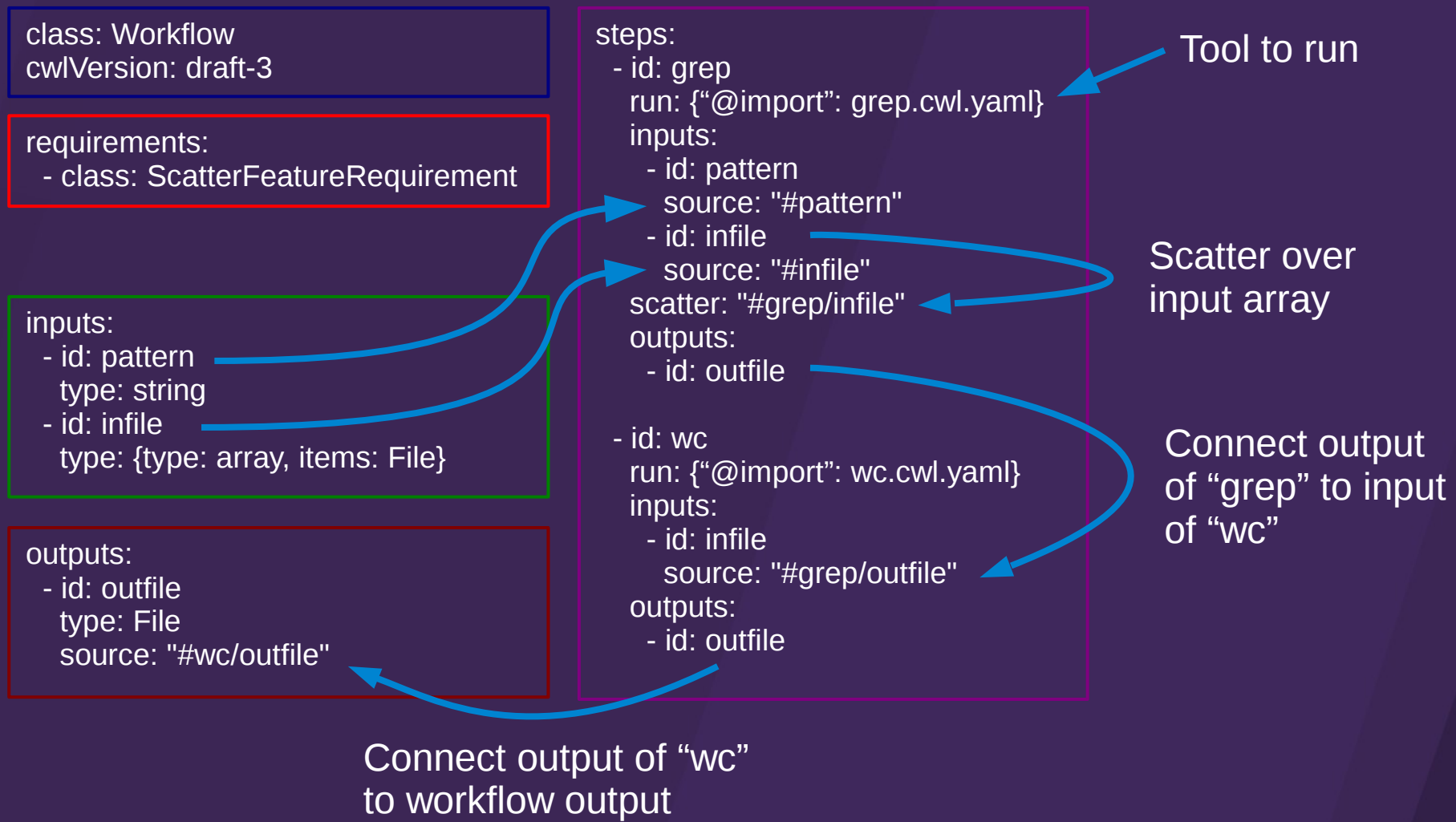
CUROVERSE™

Workflows

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

CUROVERSE™

Example: grep & count



CUROVERSE™

Thanks!

http://commonwl.org