Summary

- A single container image is not suitable for implementing workflows associated to complex application testing or validating scientific explorations.
- Popper 2.0 is a lightweight, container-native workflow execution engine. Workflows are written in a minimal specification language. Each step (node) in a workflow (graph) runs in its own container.
- Resource managers and container runtimes are abstracted away from users, allowing them to run the exact same workflow on multiple platforms with zero modifications.

Components

- **CLI**: A command-line interface allows users to run workflows, search for existing actions, generate CI configuration files and visualize workflows.
- **Workflow Runner**: This module parses the workflow representation (.workflow) file and routes the execution of an action with its corresponding plugin.
- **Resource Manager API**: This component deals interfaces with multiple resource managers and their underlying container engines.

Architecture

- **Command Line Interface**
  - **Workflow Runner**
  - **Resource Manager API**

Benefits

- **Tool Agnostic**: Python, R, C++, Rust, etc.; all kinds of scripts can be packaged into a container. The goal is to allow users to express the sequentiality and dependency logic between multiple containers.
- **Domain Agnostic**: Automate software (unit) testing, infrastructure deployment, performance regression, etc.

Challenges

- **Cultural**: Automation-centric, container-native mindset is a radical paradigm shift for users.
- **Security and Stability concerns**: container technologies move relatively quickly. In certain scenarios this lack of stability might represent a significant burden.

Easily Re-execute Workflows

- `$> git clone <workflow-repo> myworkflows`
- `$> cd myworkflows`
- `$> popper run --local machine`
- `$> popper run --manager slurm` on a cluster

Workflow

```
workflow "scc19 challenge" {
  resolves = "generate vtk output"
}

action "build" {
  uses = ".\actions\normalnodes"
  runs = ".\scripts\build.sh"
  env = {
    # Makefiles using NDK
    NORMALMAKEFILE = ".\Makefile\NormalNodes_makefile"
    NDK_BUILD_JOBS = 1
  }
}

action "run quick test" {
  needs = "build"
  uses = ".\actions\normalnodes"
  runs = ["sh", "-c", "cd \./test\/tests\; run\-test\-sh"]
  env = {
    ZENODO\_RECORD\_ID = "433946"
    ZENODO\_OUTPUT\_PATH = ".\./execution"
  }
}

action "download input" {
  needs = ["run quick test", "download input"]
  uses = ".\actions\normalnodes"
  runs = ".\scripts\run.sh"
  labels = ["api"]
  env = {
    MPI\_RUN\_PROCESSES = 1
  };
  # input parameters defined in global file
  INPUT\_DIR = "submodules\Normalized\Demo/"
}

action "validate execution" {
  needs = ["execute"]
  uses = ["actions\bin\validate\master"]
  runs = ".\scripts\validate\sh"
}

action "generate vtk output" {
  needs = ["generate\vtk\output"]
  uses = ["actions\bin\generate\master"]
  runs = ["docker://poppered/visualize:v4.4"]
  # input parameters defined in main file
  args = [".\scripts\visual\Main\m"]
}
```

In a Popper workflow, steps are broken down in the form of steps, each one running in a separate container. The entrypoint to these is usually a script, which represents a low overhead for users that are used to automating workflows using shell scripts.