Popper 2.0: A Container-native Workflow Execution Engine For Testing Complex Applications and Validating Scientific Claims

Jayjeet Chakraborty, **Ivo Jimenez**, Carlos Maltzahn, Arshul Mansoori, Quincy Wofford

CROSS | CENTER FOR RESEARCH IN OPEN SOURCE SOFTWARE
UC SANTA CRUZ | Baskin Engineering UC SANTA CRUZ
better scientific software

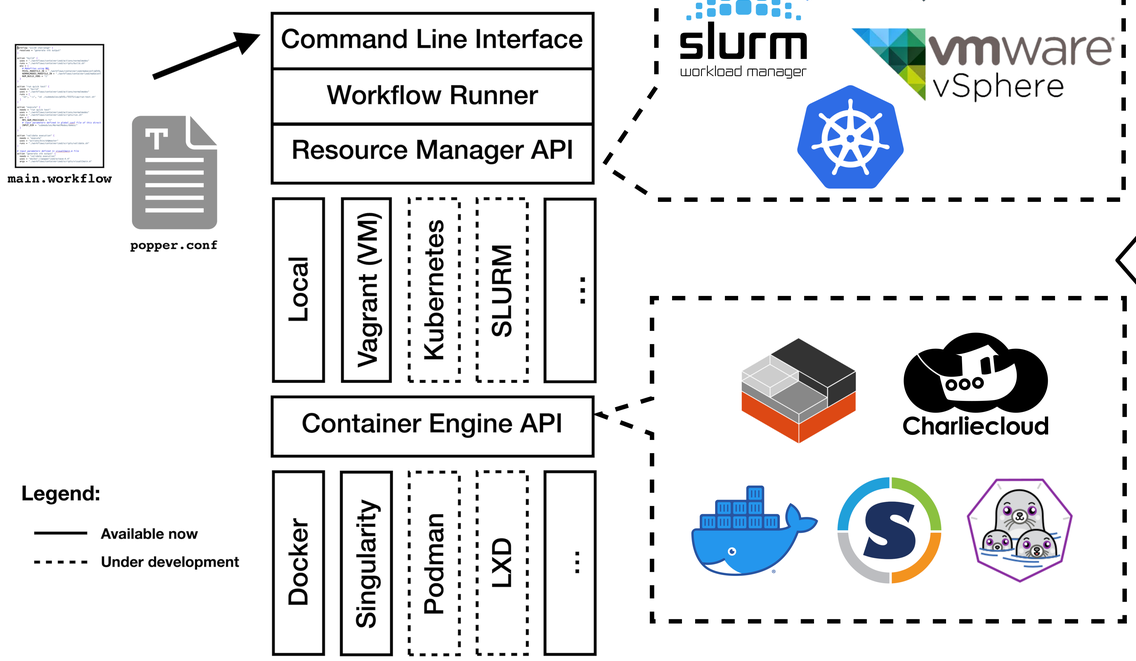
Summary

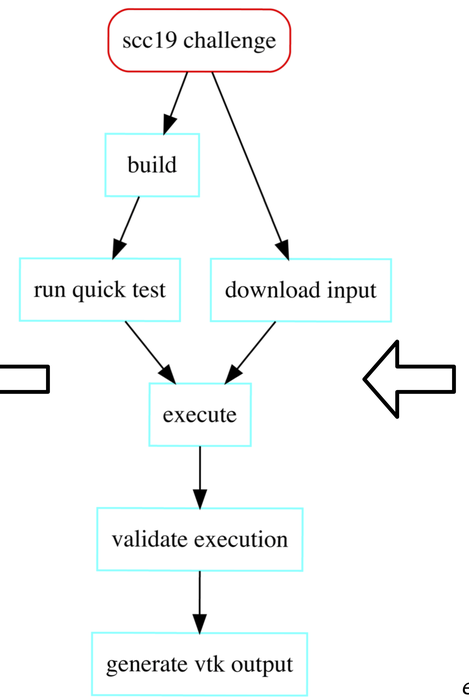
- A single container image is not suitable for implementing workflows associated to complex application testing or validating scientific explorations.
- Popper 2.0 is a lightweight, **container-native** workflow execution engine. Workflows are written in a minimal specification language. Each step (node) 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 underlaying container engines.

Architecture






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

action "build" {
  uses = "./actions/normalmodes"
  runs = "./scripts/build.sh"
  env = {
    # Makefiles using MKL
    PEVSL_MAKEFILE_IN = "./makeconf/pEVSL_mkl.in"
    NORMALMODES_MAKEFILE_IN = "./makeconf/NormalModes_mkl.in"
    NUM_BUILD_JOBS = "1"
  }
}

action "run quick test" {
  needs = "build"
  uses = "./actions/normalmodes"
  runs = ["sh", "-c", "cd ./pEVSL/TESTS/Lap/run-test.sh"]
}

action "download input"{
  uses = "popperized/zenodo/download@master"
  env = {
    ZENODO_RECORD_ID = "439946"
    ZENODO_OUTPUT_PATH = "./execution"
  }
}

action "execute" {
  needs = ["run quick test", "download input"]
  uses = "./actions/normalmodes"
  runs = "./scripts/run.sh"
  labels = ["mpi"]
  env = {
    MPI_NUM_PROCESSES = "1"
    # input parameters defined in global_conf file
    INPUT_DIR = "submodules/NormalModes/demos/"
  }
}

action "validate execution" {
  needs = "execute"
  uses = "actions/bin/sh@master"
  runs = "./scripts/validate.sh"
}

action "generate vtk output" {
  needs = "validate execution"
  uses = "docker://popperized/octave:4.4"
  # input parameters defined in visualCmain.m file
  args = "./scripts/visualCmain.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.

Easily Re-execute Workflows

```
$> git clone <workflow-repo> myworkflows

$> cd myworkflows

$> popper run # local machine

$> popper run --manager slurm # on a cluster
```

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.

Popper
falsifiable.us