Experiences Following the Popper Convention

Michael A. Sevilla, Ivo Jimenez, Carlos Maltzahn
University of California
{msevilla, ivo.jimenez, carlosm}@ucsc.edu

***Abstract*—Popper is pretty hard.**

I. Introduction

II. Popper-compliant Papers

We have authored five Popper-compliant papers:

- quiho: Automated Performance Regression Testing Using Inferred Resource Utilization Profiles []
- Cudele: An API and Framework for Programmable Consistency and Durability in a Global Namespace [1]
- Malacology: A Programmable Storage System [2]
- Characterizing and Reducing Cross-Platform Performance Using OS-level Virtualization
- Programmable Caches with a Data Management Language & Policy Engine

We use the reader/reviewer sample workflow outlined in [3] and shown in Figure 1. For the visualization component (1), we use Jupyter notebooks. The notebooks themselves are versioned with Git and users interact with local copies by cloning the repository and launching a Jupyter Docker container. The paper is written in LaTeXand built with a Docker container. For the code component (2), both the source code for the system itself and the deploy/experiment code is stored on GitHub. When running experiments, we use Docker containers to isolate libraries and binaries. For the multi-node component (3), we use CloudLab machines and Ansible to script deployment and experiment orchestration. For the data set components (4), we use GitHub to store results files; our inputs and results are small enough that we do no need a larger capacity. GitHub allows files up to 50MB and stores data on S3.

Our experiments start with a baseline; to describe the process, we reference our ceph-popper-template set up on CloudLab. Users setup SSH keys and deploy CloudLab nodes using our CephFS Profile. The profile has the nodes automatically install Docker on bootup using our install script. After the nodes finish booting (i.e. their status on the CloudLab GUI read as READY), users push SSH keys using a convenience script.

The deploy code is based on ceph-ansible, a tool that configures hardware and software for Ceph. We forked the project and made it less dependent on Python. To run an experiment, users log in into the head node and clone the ceph-popper-template. This repository has submodules that point to ceph-ansible and our own custom roles; configuration files for our Ceph setup; and helper scripts written in bash that deploy Ceph and run the benchmarks. An outline of this organization is shown in Figure 2. For more information, see the README. For more information on the baseline and pipelines terminology, read more on the Popper Convention. Then users configure their cluster by specificing IPs and user names in the `hosts` file.

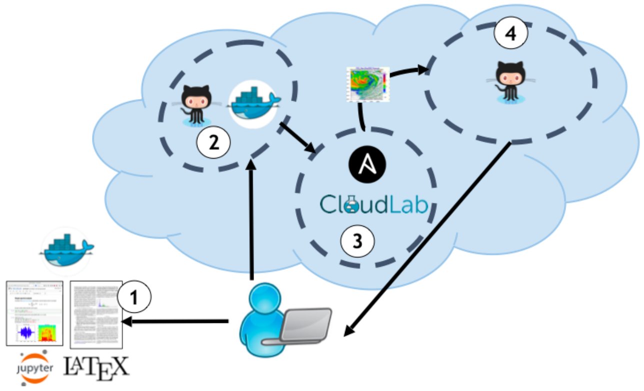

Fig. 1: Illustration of the Popper workflow used in our papers; Figure is adapted from [3]. For (1) - the visualization component - we use Jupyter, LaTeX, and Docker; for (2) - the code component - we use GitHub and Docker; for (3) - the multi-node component - we use Ansible and CloudLab; and for (4) - the data set component - we use GitHub.

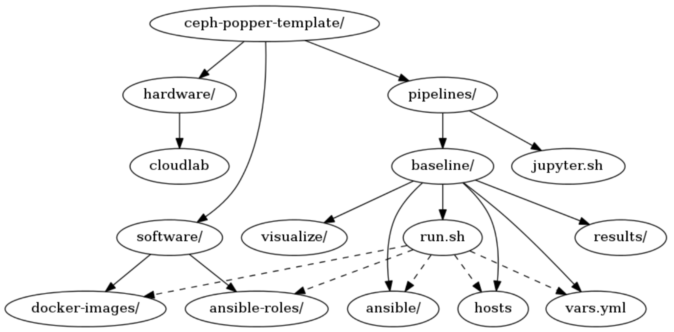

Fig. 2: Organization of our experiment directories. `baseline/` is an example experiment and contains scripts populated with the Popper CLI. The `run.sh` script uses many directories and files to setup and run experiments, as indicated by the dashed lines.

Finally, users specify the Ceph services that should be deployed using the `ansible/` directory. This directory has code for deploying Ceph and its components: `ansible.cfg`, `ceph.yml`, `cleanup.yml`, `group_vars`, `monitor.yml`, and `workloads`. The

*.yml files are Ansible playbooks that start and configure components: ceph.yml starts Ceph, cleanup.yml tears Ceph down, and monitor.yml starts daemons that monitor performance. We separate these components into different playbooks so users can mix and match Ceph services. The workloads directory has scrips for running the baseline benchmarks. The other files and directories are Ansible configuration files used by the playbooks.

Users can change the ansible/ceph.yml to specify which Ceph daemons to launch in the cluster. It uses the hosts file we set up above and is based off the ceph-ansible site file. High level configurations are in the vars.yml. This file is heavily commented. For CloudLab, uncomment all blocks labeled "Uncomment for CloudLab".

To configure the Ceph cluster with the variables in the Ceph configuration file documentation, change the Ansible `group_vars/all` file. We also need to specify the Docker image. These can be specified in each configuration file for the daemon but for simplicity we put everything in the global variable file. Finally, users run the `run.sh` script.

III. Popper Best Practices

To produce Popper-compliant papers, we needed to change our workflows and experimental procedures. We have learned the following lessons:

A. *Reproducibility Must Be a 1st Class Citizen*

Popper-compliance must be observed *throughout* the paper-writing process. Attempts to make published papers Popper-compliant failed for three reasons: (1) there is no incentive from the perspective of a graduate student, (2) it is too hard to remember the experimental workflow, and (3) the artifacts cannot be added to the camera-ready article. This introduces a careful design decision: how does the user decide which results are exploratory and which are complete; we do not want to waste space by saving results that do not contribute to the paper but, at the same time, how do we know when an experiment is worthy of being included in the paper? Discipline must be exercised throughout the paper-writing process to make any experiment that we find useful must be immediately polished to be Popper-compliant.

Another pitfall we face is cross-cluster compatibility. After identifying results as being useful, we usually find inventory files hard-coded throughout the tree, configuration files randomly propagated to different directories, visualization files reliant on data specified with hard-coded paths, and graphs are not created automatically. To address this, we must exercise discipline to separate cluster-specific files from cluster-agnostic files; we do this with `configs_*/` directories in our ceph-popper-template repository.

Finally, organization and documentation must be required throughout the paper-writing process. We fail to do this in early papers and this causes the most work later on. System and experiment deploy code is rarely self-explanatory. At a minimum, each experiment directory must contain a README specifying how to run the jobs. To address this, we recommend relying heavily on DockerHub, GitHub, and collaboration; making things public incentives tidiness and organization. Using internal Docker or Git repositories hampers Popper-compliance and discourages community involvement.

B. *Well-Defined Collaboration Roles*

Collaboration obviously speeds up code development and, As stated in the previous section, collaboration can be used as motivation to keeping Popper-compliant repositories; but collaboration can also hamper the process. Researchers have different workflows and preferences. Committing system and experimental deploy code with different toolkits, documentation, and styles to the same repository results can result in a confusing organizational structure. To address this, we recommend that the first author of the paper (1) produce a style guide, either by committing a couple of experiments or adding detailed READMEs and (2) act as the gatekeeper for all code. We recommend using GitHub because it nicely formats READMEs and supports pull requests, issue trackers, and project boards. Rather than having meetings to agree on experiment organization, we recommend iteratively combining code from collaborators with pull requests so the repository grows organically and quickly in the style approved by the first author.

Another obstacle of collaboration is merge conflicts, especially when editing the paper. We used to assign locks to people, in an *ad hoc* fashion over email; this is not scalable. Instead, we now use Git to manage conflicts. Before issuing a pull request, which notifies the first author, GitHub will notify committers of changes they made in parallel with other changes. Our policy is to force committers to resolve these merge conflicts before bothering the first the author. While we still recommend issuing pull requests to make sure the first author approves of the style, the merge conflicts workflow has been an effective way to avoid annoying the first author with destructive changes.

C. *Maintaining Pointers*

- wary of ephemeral links
- large code bases large code bases (src, deploy)
- different registry hubs (Docker, GitHub)

IV. Community Cooperation

A. *Conference Requirements*

B. *Industry/Laboratory Requirements*

V. Conclusion

References

[1] M. A. Sevilla, I. Jimenez, N. Watkins, S. Finkelstein, J. LeFevre, P. Alvaro, and C. Maltzahn, "Cudele: An API and Framework for Programmable Consistency and Durability in a Global Namespace," in *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS '18, 2018.

[2] M. A. Sevilla, N. Watkins, I. Jimenez, P. Alvaro, S. Finkelstein, J. LeFevre, and C. Maltzahn, "Malacology: A Programmable Storage System," in *Proceedings of the European Conference on Computer Systems*, ser. EuroSys '17, 2017.

[3] I. Jimenez, M. A. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, “The Popper Convention: Making Reproducible Systems Evaluation Practical,” in *Proceedings of the International Parallel and Distributed Processing Symposium Workshop*, ser. IPDPSW ’17.