Practical Reproducible Evaluation of Systems with Popper

Ivo Jimenez, Michael Sevilla, Noah Watkins, Carlos Maltzahn

University of California, Santa Cruz

Independently validating experimental results in the field of computer systems research is a challenging task. Recreating an environment that resembles the one where an experiment was originally executed is a time-consuming endeavour. In this white paper, we present Popper [1], a convention (or protocol) for conducting experiments following a DevOps [2] approach that allows researchers to automate the re-execution and validation of an experiment.

Introduction

Over the last decade software engineering and systems administration communities (also referred to as DevOps) have developed sophisticated techniques and strategies to ensure "software reproducibility", i.e. the reproducibility of software artifacts and their behavior using versioning, dependency management, containerization, orchestration, monitoring, testing and documentation. The key idea behind the Popper Convention is to manage every experiment in computation and data exploration as a software project, using tools and services that are readily available now and enjoy wide popularity. By doing so, scientific explorations become reproducible with the same convenience, efficiency, and scalability as software reproducibility while fully leveraging continuing improvements to these tools and services. Rather than mandating a particular set of tools, the convention only expects components of an experiment to be scripted (see Fig. 1). There are two main goals for Popper:

1. It should be usable in as many research projects as possible, regardless of their domain.
2. It should abstract underlying technologies without requiring a strict set of tools, making it possible to apply it on multiple toolchains.

A DevOps Approach to Carrying Out Experiments

A common generic workflow for experiments with a computational component is the one shown below. Although there are some projects or papers that don't fit this description we focus on this model since it covers a large portion of experiments out there. The implementation and documentation of an experiment, is commonly done in an ad-hoc way (custom bash scripts, storing in local archives, etc.).

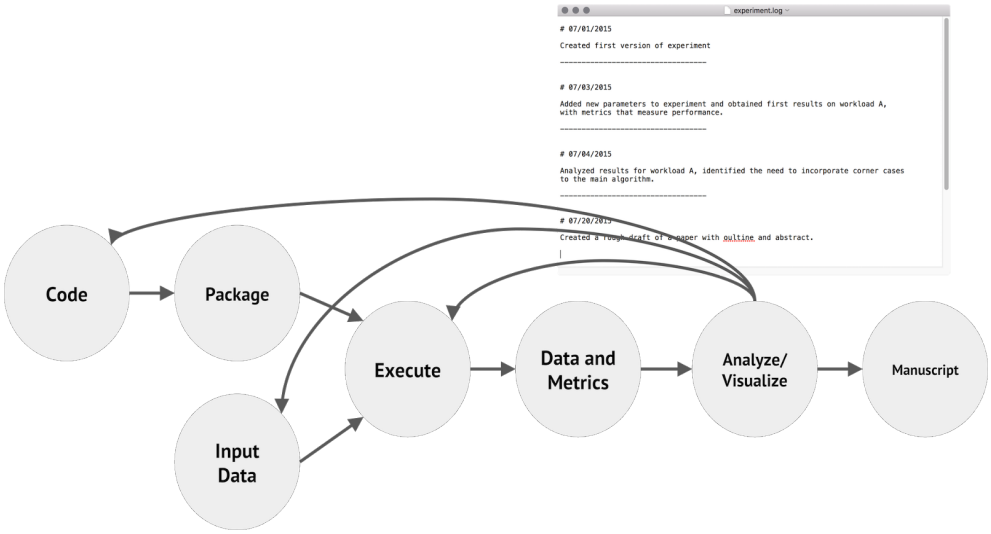

Figure 1: A generic experimentation workflow. The analogy of a lab notebook in experimental sciences is to document an experiment's evolution. This is rarely done and, if done, usually in an ad-hoc way (an actual notebook or a text file).

The idea behind Popper is simple: make an article self-contained by including in a code repository the manuscript along with every experiment's code, orchestration, inputs, parametrization, results and validation. To this end we propose leveraging state-of-the-art technologies and applying a DevOps approach to the "implementation" of an article.

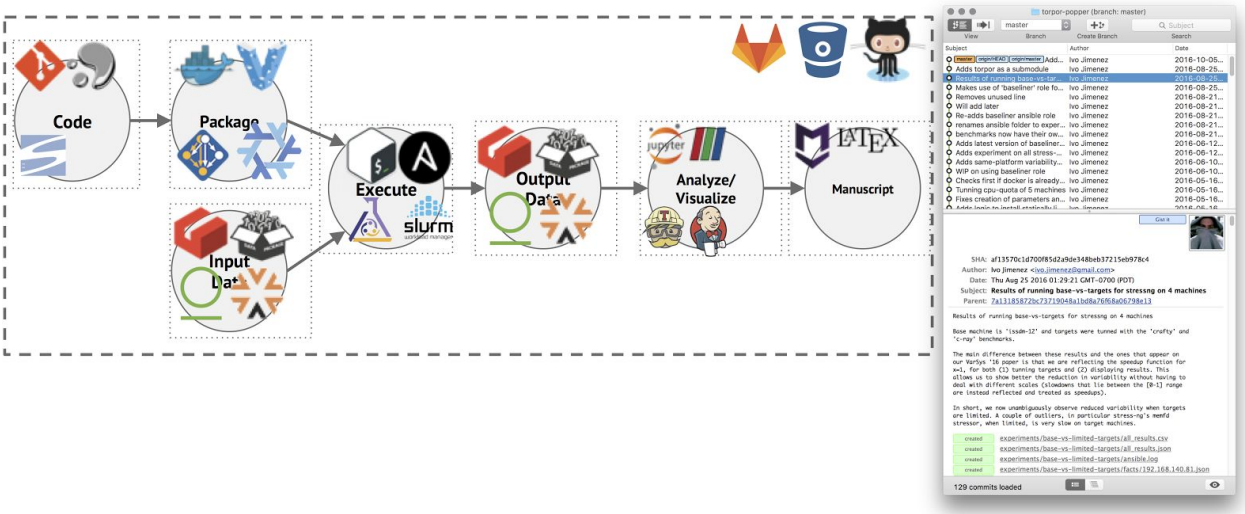

Figure 2: A generic experimentation workflow viewed through a DevOps looking glass. The logos correspond to commonly used tools from the DevOps toolkit. Scripts corresponding to each stage are stored in a version control repository, whose commit log resembles a lab notebook .

Popper maps the components of an experimentation workflow to the engineering best-practices that are commonly applied in open source software projects. Popper is followed by:

1. At each stage of the experimentation process, picking one or more tools from the DevOps toolkit (see below).
2. Creating scripts for each of these tools and store them in a version control repository.
3. Documenting changes to an experiment (as well as results) in the form of commits to this repository.

If, from the inception of an article, a researcher makes use of the DevOps toolbox (e.g., version-control systems, lightweight OS-level virtualization, automated multi-node orchestration, continuous integration and web-based data visualization), then re-executing and validating an experiment becomes practical.

Popper-compliant Experiments

We say that an experiment is Popper-compliant if its code, orchestration, dependencies, results, parameterization and validation are *self-contained*. By self-contained, we mean available in a code repository with dependencies available in artifact and data repositories. If resources are available, we can execute a Popper-compliant (or "popperized") experiment can be executed in its entirety. Additionally, the commit log becomes the lab notebook, which makes the history of changes made to it available to readers, an invaluable tool to learn from others and "stand on the shoulder of giants". A "popperized" experiment also makes it easier to advance the state-of-the-art, since it becomes easier to extend

existing work by applying the same model of development in OSS (fork, make changes, publish new findings).

A list of Popperized experiments is available in the Popper Templates repository. See below for how to use the Popper-CLI tool to easily explore the templates repository and add experiments to a paper repository.

Popper-compliant Tools

While Popper applies to a wide variety of toolchains, it is not universal. We generally require tools to have two basic properties:

1. Referenceable assets. Ability to associate IDs to assets (code, binaries, configuration and data).
2. Scriptability. The tool in question has to be amenable to automation (scriptable). In general, given a high-level, human-readable script (or asset ID), the tool should be able to act upon it.

The notion of Popper-compliance closely resembles the high-level guidelines of the Twelve-Factor App, re-purposed for an academic setting, i.e. we aim for the *Twelve-factor Experiment*.

We maintain a list of "Popperized" experiments at `http://github.com/systemslab/popper`. We also provide a CLI tool for researchers to bootstrap a project that follows the convention, as well as a wiki with guides and examples. Projects that follow the convention can make use of our `http://falsifiable.us` service to automatically validate an experiment.

Listing 1: Interacting with the Popper-CLI tool.

```
$ cd mypaper-repo
$ popper init
-- Initialized Popper repo

$ popper experiment list
-- available templates ---------------
ceph-rados    proteustm  mpip
cloverleaf    gassyfs    zlog
spark-bench   torpor     malacology

$ popper add torpor myexp
-- Added torpor experiment to mypaper-repo

$ popper check myexp
-- SUCCESS - myexp is Popper-compliant
```

Use Case

The following describes a series of steps to bootstrap a data science paper that follows the Popper convention using the Popper-CLI tool. Popper in this scenario is followed so that datasets are properly referenced and analysis scripts used to process data (as well as any output data) are versioned and associated to an article. While in this guide we use LATeX, Docker, dpm and Jupyter, any of these can be swapped for equivalent tools. To learn more about how to use other tools and how the Popper convention is toolchain-agnostic, see here.

Requirements:

- git
- docker
- popper-cli

Initialize a Popper Repository

Our Popper-CLI tool assumes a git repository exists. To create one:

```
mkdir mypaper
cd mypaper
git init
echo "# My Paper Repo" > README.md
git commit -m "First commit of my paper repo."
```

See here for a list of good resources for learning git. Once a git repo exists, we can invoke the popper-cli tool:

```
cd mypaper
popper init
```

The above creates a `.popper.yml` file that contains configuration options for the CLI tool. This file should be committed to the paper repository (git repo we create above). For an explanation on the folder structure of a Popper repo, see here.

Adding a New Experiment

The Popper convention outlines how to make it practical to generate reproducible experiments. As part of our effort, we maintain a list of experiment templates that have been "popperized" (see here for an explanation of what constitutes a Popper-compliant experiment). To see a list of available experiments:

```
popper experiment list
```

In order to add a new experiment, we refer to a template and assign a name to it. The general invocation form is the following:

```
popper experiment add <template> <experiment-name>
```

For example, assume we want to analyze data from an experiment in the area of meteorological sciences (a template created as part of the Big Weather Web project):

```
popper experiment cms-analysis myexperiment
```

This data analysis experiment consists of one dataset and a jupyter notebook. To retrieve the dataset to the local machine:

```
cd experiments/myexperiment

docker run --rm -v `pwd`/datapackages:/datapackages \
  ivotron/dpm install /datapackages/air-temperature
```

NOTE: The above makes use of the dpm tool for managing datapackages. The tool doesn't support `file:///` URLs yet (until this issue gets resolved). In the meantime, to download the dataset from github, replace `/datapackages/air-temperature` with `https://github.com/ivotron/air-temperature`.

To visualize and interact with the data analysis of this experiment:

```
cd experimetns/myexperiment
./visualize
```

The above opens a browser and points it to the notebook. In this example, the dataset used by the notebook resides in the `myexperiment/datapackages/` folder.

For this experiment we assume that input data has been externally generated, i.e. dataset creation is not part of the experiment. Also, the analysis runs on a single machine. Other types of data science projects might involve generating their input datasets and/or process data in a cluster of machines. Popper still can be followed in these scenarios.

Adding More Datasets

Datasets are stored (or referenced) in the `datapackages/` (or `datasets/`) folder of each experiment, with one subfolder for each dataset. For examples datasets see here. To add or reference a new dataset, one has to either provide a URL of the dataset, or inspect a the list of datapackages available in a data repository using the `dpm` tool. Available repositories are `github`, `ckan` and `thredds`.

NOTE: Support for THREDDS is not part of the official `dpm` client yet. Work is being done in this as part of the big weather web project.

Once a dataset URL is available, one can install a package by doing

```
docker run --rm -v `pwd`/datapackages:/datapackages \
  ivotron/dpm install http://motherlode.ucar.edu:8080/thredds/bww/
```

To display the info for a package, use the `info` command of `dpm`. For more info on how to use `dpm` take a look at the official documentation.

Generating Image Files For Reference In Manuscripts

Assume we add a new type of analysis to the notebook and we want to generate an image. For the notebook of our example (xarray-tutorial.ipynb of the `jupyter-bww` experiment), we can generate a file for figure 2 (Line `[45]`). In Jupyter, we add a new cell below the figure and type the following line:

```
plt.savefig('air-temperature.png',bbox_inches='tight', dpi=300)
```

Since the experiment folder is available in the filesystem that Jupyter has available to it, the figure persists even after the Jupyter server exits. To automatically re-execute the analysis and re-generate figures from a notebook, one can use the `run-notebook` script contained in the `jupyter-bww` experiment:

```
cd myexperiment
./run-notebook
```

Documenting the Experiment

After we're done with our experiment, we might want to document it and add a paper. We can use the generic `article` latex template or other more domain-specific one (available here). To display the available templates we do `popper paper list`. In this example we'll use the latex template for articles that appear in the Bulletin of the American meteorological Society (BAMS):

```
popper paper add latex-ametsoc
```

Let's assume we will have a new section in the LATeX file where we describe our experiment. We will make use of the figure that we generated in the previous section. We can make the assumption that the experiments folder is available at the level of the latex file, so we can reference the image directly. For example:

```
\begin{figure}[t]
  \includegraphics{experiments/myexperiment/air-temperature.png}\\
  \caption{Air temperature.}\label{f1}
\end{figure}
```

And to re-generate the PDF containing the new image:

```
cd paper
./build
```

Bibliography

[1] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, R. Arpaci-Dusseau, and A. Arpaci-Dusseau, *Popper: Making Reproducible Systems Performance Evaluation Practical*, UC Santa Cruz, SOE-16-10, 2016.

[2] M. Hüttermann, *DevOps for Developers*, 2012.