Spotting Black Swans With Ease: The Case for a Practical Reproducibility Platform

Ivo Jimenez and Carlos Maltzahn (UC Santa Cruz)

Abstract

Advances in agile software delivery methodologies and tools (commonly referred to as *DevOps*) have not yet materialized in academic scenarios such as university, industry and government laboratories. In this position paper we make the case for *Black Swan*, a platform for the agile implementation, maintenance and curation of experimentation pipelines by embracing a DevOps approach.

1 Introduction

Reproducibility is the cornerstone of the scientific method. Yet, in computational and data science domains, a gap exists between current practices and the ideal of having every new scientific discovery be *easily* reproducible [1]. Advances in computer science (CS) and software engineering slowly and painfully make their way into these domains—even in CS research itself [2,3], paradoxically.

Popper [4] is an experimentation protocol and CLI tool for implementing scientific exploration pipelines following a DevOps approach. The goal of Popper is to bring the same methods and tools used for the agile delivery of software, known as DevOps [5], to scientists and industry researchers.

Our experience in the past four years developing and evangelizing the use of Popper in multiple scientific domains has allowed us to identify opportunities where open-source software (OSS) can be used to close the existing gap in current research practices. While the Popper protocol is relatively easy to follow (wrap experimentation pipelines in the form of a sequence of Bash scripts), for many practitioners it still represents a big leap between current practices and where OSS development communities are today (automated, portable and versioned software testing pipelines). To this end, in this position paper we make the case for building *Black Swan*[1], a platform for *practical* reproducible research. In a nutshell, Black Swan enables the agile delivery of software created in universities and other research institutions, significantly accelerating technology transfers between research and operational environments.

This paper is organized as follows. Section 2 expands on the need for Black Swan, while Section 3 presents the components of the proposed platform and how they address the current gaps. Section 4 illustrates the utility of the platform by describing use cases where Black Swan would be used. We close by discussing challenges in Section 5.

2 Motivation

Following the Popper experimentation protocol turns out to be a significant paradigm shift for many users

[1] Black swans are typically used to illustrate the concept of falsifiability: the statement "all swans are white" is made falsifiable by defining the condition under which it would be false, i.e. finding one or more non-white swans. In this case, the statement was actually proven false by the discovery of black swans in Australia in the 1600's. We envision *Black Swan* (the platform) to be a tool for researchers to *easily* find black swans in their computational or data science theories (i.e. identify when their claims are false) and, more importantly, allow them to investigate **why**.

that is hard to accept in the context of tight submission for publication deadlines. It begins by defining a pipeline for a scientific exploration, i.e. a sequence of high-level steps that are carried out when executing an experiment or analysis. For example, a data analysis pipeline may consist of four stages: (1) obtain a dataset; (2) pre-process the data; (3) run an analysis on the data; and (4) produce plots.

The contents of a folder containing scripts for a pipeline are shown in Lst. 1 (examples are available on Github). Each stage in a pipeline corresponds to a Bash script, and it codifies what a person would manually type in a terminal otherwise. A stage in a pipeline is a relatively simple list of steps, where each invokes other tools, and passes scripts to them, which are themselves stored in the same repository.

Listing 1 Contents of a pipeline.

```
paper-repo/pipelines/gassyfs
|- README.md
|- baseliner
|  |- config.yml
|- docker
|  |- Dockerfile
|- geni
|  |- cloudlab_request.py
|- results
|  |- output.csv
|- run.sh
|- setup.sh
|- validate.sh
```

There are four key aspects that make Popper pipelines practical. (1) They are version-controlled, which allows readers to understand what was done over time (and why), mimicking a lab notebook; (2) thanks to virtualization technology at the language-, OS- or hardware-level, they are portable and can be easily re-executed in many environments; (3) are automated; and (4) are self-contained.

While implementing a Popper pipeline can be done in many ways,[2] in order for a pipeline to be *Popper Compliant*, it must be a *Self-verifiable Experimentation Pipeline* (SEP) [6]. A SEP is a pipeline that carries out the following high-level tasks in every execution:

1. **Code and data checkout**. Code must reside on a version control system (e.g. Github, Gitlab, etc.). If datasets are used, then they should reside in a dataset management system (Zenodo, CKAN, Datapackages, etc.). The experimentation pipelines must obtain the code/data from these services on every execution.
2. **Setup**. The pipeline should build and deploy the code under test. For example, if a pipeline is using containers or VMs to package their code, the pipeline should build the container/VM images prior to executing them. The goal of this is to verify that all the code and 3rd party dependencies are available at the time a pipeline runs, and that software can be build correctly.
3. **Resource allocation**. If a pipeline requires a cluster or custom hardware, resource allocation must be done as part of the execution of the pipeline. This allocation can be static or dynamic. For example, if an experiment runs on custom hardware, the pipeline can statically allocate (i.e. hardcode IP/hostnames) the machines where the code under study runs (e.g. GPU/FPGA nodes). Alternatively, a pipeline can dynamically allocate nodes on CloudLab [7], Grid500K [8], Chameleon [9], or a public cloud provider. These services typically offer infrastructure automation tools such as Genilib (Cloudlab), Enos [10] (Chameleon), SLURM [11] (HPC centers), or Terraform (AWS, GCP, etc.). All these tools can be used to automate infrastructure-related tasks.
4. **Explicit parametrization**. Parameters that are relevant to the outcome of the pipeline need to be made explicit, for example by creating a `parameters.yml` file.
5. **Environment capture**. Capture information about the runtime environment. For example, hardware description, OS, system packages

[2]For our purposes, any tool that can be invoked on the CLI and can be given a script as input is a so-called "DevOps" tool. For more on other aspects of DevOps tools see [5] and `http://12factor.net`.

(i.e. software installed by system administrators), information about remote services (e.g. version and state of a batch scheduler), etc. Open-source tools such as SOSReport or Facter can aid in aggregating this information.

6. **Results Validation**. Scripts must verify that the output corroborates the claims made on the article. For example, the pipeline might check that the throughput of a system is within an expected confidence interval (e.g. defined with respect to a baseline obtained at runtime), or that a numerical computation is within some expected bounds. This can be done with domain-specific tools [12], or generic data analytics stacks such as Python (Pandas [13]) or R [14].

Since a Popper pipeline is, fundamentally, a list of Bash scripts (with a specific execution order) stored in a version control repository with a pre-defined folder structure, implementing SEPs is an arguably straightforward habit to adopt.[3] However, based on our first-hand experience following the protocol in our laboratory, as well as teaching hands-on tutorials for the past year, we have identified two broad classes of users. On the one hand, there is the user that goes through the learning curve and experiences the benefits of following the protocol, both at the personal level, and when collaborating with others. In words of some of the attendees to the tutorials, "it quickly pays off" to go through the learning process. On the other hand, the fact that a Popper pipeline is implemented *a priori* (i.e. before an article has written), and that creating reusable pipelines implies the use of new tools (such as Docker or Spack), its adoption is seen by some potential new users as a big paradigm shift. The main criticism from this set of people is that "there is no time" for a researcher or student to do things in this "radically new way".

Given the above, we see an opportunity to develop new OSS technology to close the gap for those that still resist to take the leap. The main objective is to create a platform where it is *ridiculously easy* to both, implement and re-execute experimentation pipelines. Our target quantifiable goals are "push-button" repeatability (re-execute an existing experiment) and "no more than 10 minutes" to assemble the skeletal components of a new scientific exploration pipeline (a few clicks on a web-based GUI should suffice).

[3]Based on our first-hand experience, we spend at most the same amount of time implementing a pipeline; what changes is the approach, i.e. we write scripts to automate a pipeline instead of manually executing it. This, in turn, means that the more we re-execute a pipeline, and the more we reuse stages across distinct explorations, the more it pays off.

3 Black Swan

Once implemented, Black Swan will be a community-driven reproducibility platform that enables the agile delivery and validation of scientific insights. There are five main functional components of the platform.

External Service Integrations

One design principle is not to re-invent the wheel. That is, rather than re-implementing functionality found in other services or tools, Black Swan will have a pluggable mechanism so that it can integrate with these. A diagram of basic integrations is shown in Fig. 1. Version control services will store a pipeline's content; input and output will be version-controlled by connecting to dataset management services; CI services will continuously validate the integrity of a pipeline; and a database service will be used to store the history of executions for each pipeline, and as much environmental information as possible.

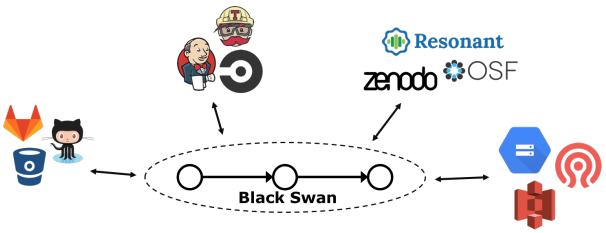

Figure 1: Black Swan will leverage existing services.

Automated Compliance Verification

Black Swan automates the process of verifying that a pipeline complies with the SEP criteria outlined

previously, in a domain-agnostic way. Verification is done at runtime and is based on conventions of what the output of a pipeline looks like. For example, to verify that code and data repositories are being cloned, the output of a pipeline (what's get printed to `stdout`) is inspected to verify that repositories are being contacted (e.g. downloading from Github or Zenodo); the task of infrastructure allocation is expected to generate a `manifest.yml` (or `system.json`) file; explicit parametrization is verified by checking whether a `parameters.yml` file exists; and so on and so forth. By default, a pre-defined list of checks will be supported but the underlying mechanism will be extensible so more items are added in order to verify SEP compliance.

Pipeline Catalog and Pipeline Builder

In order to facilitate the adoption of the DevOps practice, the platform will incorporate a GUI component to allow users to visualize Popper pipelines and their stages. In particular, a *Pipeline Builder* will allow users to "mix and match" stages from an existing catalog of SEP compliant, community-maintained pipelines (Fig. 2). The reusable catalog and the pipeline builder illustrate the importance that community will have in the success of Black Swan as an OSS project. Unless there is a community-wide effort in place, Black Swan will not succeed as a viable project; maintaining a pipeline by a single individual is too overwhelming.

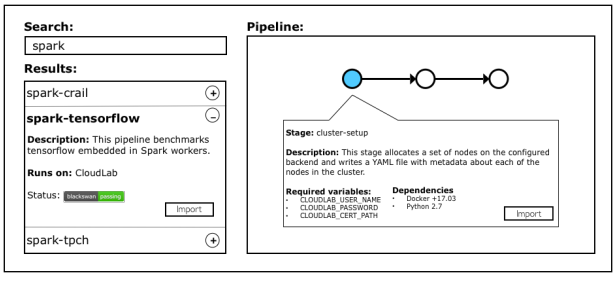

Figure 2: Sketch of the pipeline builder GUI.

Differential Analysis

Whenever the validation stage of a SEP pipeline fails, we have found a black swan, i.e. the domain-specific expectations on results do not hold. Having access to the environmental information on which hardware and software was a pipeline re-executed on, Black Swan will provide facilities to automatically analyze and compare against previous successful executions in order to determine root causes of irreproducibility, or to aid researchers in finding them by incorporating GUI elements to support the visual inspection of differences.

Validation Dashboard

Borrowing ideas from [15], Black Swan will incorporate facilities to quickly visualize the status of a pipeline with respect to domain-specific validations. For failed validations, the dashboard will provide links to trigger differential analysis with respect to previous successful executions, as described above.

4 Use cases

We describe three use cases for which we envision Black Swan to be directly applicable.

Technology Transfer

Organizations with Research and Development (R&D) units such as tech companies and government-funded institutions (DOE labs, NASA, universities) spend significant amounts of resources transferring technology from R&D to production environments. Deploying an instance *Black Swan* internally (or using a public one) would allow organizations to streamline tech (and knowledge) transfers.

Research Curation

In the past decade, institutional libraries have invested a significant amount of resources in the creation of *Data Repositories*. These efforts are aimed at systematically managing the output of research. For example, EU's OpenAIRE initiative is a catalog of software and data containing more than 80 million entries (each being a code repository or dataset). Similar efforts are underway in the US such as the Center for Open Science's Open Science Framework. We see *Black Swan* complementing these efforts, since

Popper enables the curation of research by enabling all these existing repositories to be easily executed and validated over time.

Self-validation of Academic Articles

A growing number of Computer Science conferences and journals incorporate an artifact evaluation process in which authors of an article submit an artifact description[4] (AD) that is tested by a committee, in order to verify that experiments presented in a paper can be re-executed by others. Black Swan can be leveraged to automatically test the reproducibility aspects of articles, assuming the pipeline(s) corresponding to an article are SEP compliant [6].

5 Challenges

Managing changes to code using version-control systems; managing data with dataset management systems; continuously integrating (CI) and deploying (CD) software; all have become standard practices in OSS communities, not because of a fad but because of the quantifiable benefits that following best practices bring. To the contrary, in R&D settings, these practices are seen as a burden. The biggest challenge we face lies in changing the culture within organizations and teams; finding the right incentives so that these users can make the leap and adopt agile practices, so they can enjoy the benefits that come from embracing DevOps.

References

[1] D.L. Donoho, A. Maleki, I.U. Rahman, M. Shahram, and V. Stodden, "Reproducible Research in Computational Harmonic Analysis," *Computing in Science & Engineering*, vol. 11, Jan. 2009, pp. 8–18.

[2] G. Fursin, "Collective Mind: Cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning," *arXiv:1308.2410 [cs, stat]*, Aug. 2013. Available at: http://arxiv.org/abs/1308.2410.

[3] C. Collberg and T.A. Proebsting, "Repeatability in Computer Systems Research," *Communications of the ACM*, vol. 59, Feb. 2016, pp. 62–69.

[4] http://ctuning.org/ae/submission.html

[4] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "The Popper Convention: Making Reproducible Systems Evaluation Practical," *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 1561–1570.

[5] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook*, O'Reilly Media, 2016.

[6] I. Jimenez and C. Maltzahn, "Self-verifiable Experimentation Pipelines," Sep. 2018.

[7] R. Ricci and E. Eide, "Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications,";*login:* vol. 39, pp. 36–38.

[8] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.-G. Talbi, and I. Touche, "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed," *Int. J. High Perform. Comput. Appl.*, vol. 20, Nov. 2006, pp. 481–494.

[9] J. Mambretti, J. Chen, and F. Yeh, "Next Generation Clouds, the Chameleon Cloud Testbed, and Software Defined Networking (SDN)," *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, 2015, pp. 73–79.

[10] R. Cherrueau, D. Pertin, A. Simonet, A. Lebre, and M. Simonin, "Toward a Holistic Framework for Conducting Scientific Evaluations of OpenStack," *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 544–548.

[11] A.B. Yoo, M.A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds., Springer Berlin Heidelberg, 2003, pp. 44–60.

[12] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "I Aver: Providing Declarative Experiment Specifications Facilitates the Evaluation of Computer Systems Research," *TinyToCS*, vol. 4, 2016.

[13] W. McKinney, "Data structures for statistical computing in python," *Proceedings of the 9th Python in Science Conference*, Austin, TX, 2010, pp. 51–56.

[14] R. Core Team, "R: A language and environment for statistical computing," 2013.

[15] I. Jimenez, S. Hamedian, J. Lofstead, C. Maltzahn, K. Mohror, R. Arpaci-Dusseau, A. Arpaci-Dusseau, and R. Ricci, "PopperCI: Automated Reproducibility Validation," *2017 IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT '17)*, Atlanta, GA, USA: 2017.