Everware toolkit. Supporting reproducible science and challenge-driven education.

A Ustyuzhanin[1,2], **T Head**[3], **I Babuschkin**[4], **A Tiunov**[2]

[1]National Research University Higher School of Economics
[2]Yandex School of Data Analysis
[3]Wild Tree Technologies
[4]Manchester University

E-mail: andrey.u@gmail.com

Abstract. Modern science clearly demands for a higher level of reproducibility and collaboration. To make research fully reproducible one has to take care of several aspects: research protocol description, data access, environment preservation, workflow pipeline, and analysis script preservation. Version control systems like git help with the workflow and analysis scripts part. Virtualization techniques like Docker or Vagrant can help deal with environments. Jupyter notebooks are a powerful platform for conducting research in a collaborative manner. We present project Everware that seamlessly integrates git repository management systems such as Github or Gitlab, Docker and Jupyter helping with a) sharing results of real research and b) boosts education activities. With the help of Everware one can not only share the final artifacts of research but all the depth of the research process. This been shown to be extremely helpful during organization of several data analysis hackathons and machine learning schools. Using Everware participants could start from an existing solution instead of starting from scratch. They could start contributing immediately. Everware allows its users to make use of their own computational resources to run the workflows they are interested in, which leads to higher scalability of the toolkit.

1. Introduction

The demand for reproducibility in scientific research is gradually increasing nowadays. It could be seen by number of re-research cases, in which people are spending considerable amount of efforts to replicate results obtained by others before. Popular cases of such replication efforts can be found in biology [1], psychology [2] and other branches of the science [3]. One of the main outcome of such research is that fraction of the results that coincide with original published result is considerably lower than expected. “More than 70% of researchers have tried and failed to reproduce another scientist’s experiments, and more than half have failed to reproduce their own experiments” - those are some of the telling figures that emerged from Nature’s survey of 1,576 researchers who took a brief online questionnaire on reproducibility in research [3].

Another aspect of scientific efforts that is closely related to research and hence to the reproducibility is so-called “challenge driven education”. Examples of that kind of efforts include hackathons, interactive workshops, seminars, master classes and other kinds of outreach activities. These hackathons usually suppose that participants a) not equally well-familiar with the problem they are supposed to tackle, but b) they should start from same baseline solution that should be easily accessible and understandable for everyone participating. Also results participants obtain during the event should also be accessible and reproducible at least for the organisers and perhaps to all the participants.

For the sake of technical-level discussion and without touching much of the social reasons of research irreproducibility, let us consider the case of computational experiment. It is usually takes great deal of efforts for every experiment, but it starts as data has been collected and ends at the point as experimental results firmly confirm/refute on of the hypothesis under the consideration. Even improving reproducibility of that part of the experiment may lead to much better understanding of the particular results by the community. Namely accessibility and repeatability of the analysis pipelines of successful researchers can improve the following aspects: better mentoring/supervision, more careful within-lab validation, simplified external-lab validation, give incentive for better practice and robust experiment design. From educational point of view such transparency would widen access to the best practices and facilitate better teaching. The aspects mentioned above are highly correlated with the reproducibility boost factors mentioned in survey [3].

2. Reproducibility key components

Let us consider the main aspects of the computational research that lead to successful reproducibility of the results. It usually starts from rather basic level of understanding of *assumptions* that are effective in the field of the research. For example to reproduce results in the field of high energy physics you have to understand basic concepts of event, likelihood fit, decay, mother/daughter particles and so on. As soon as one is comfortable with those definitions and assumptions, he/she can follow definition of the data and understand columns/relationships between different entities. Another item included into assumptions would be description of hypothesis being evaluated. Accessibility of the *dataset* and its metadata is the second degree of the reproducibility. Then follows description of the *computational environment* that should be created/restored for the experiment to run. Depending on the level of detalization such description may contain just name of the interpreter and operating system, or can be as detailed as list of required modules, system libraries, configuration files along with exact specification of its versions. Some guidelines demand nothing less than virtual machine with all the modules pre-installed. Next component to follow is the *code of the computational analysis* that reads data, performs necessary transformations and computes final analysis numbers that lead to certain conclusions about hypothesis being considered within the analysis. In some cases the analysis performed by the code is split into various steps: data profiling, feature engineering, simulation, etc. Such steps may form a certain *workflow* that researcher should be able to trigger by a single command. Description of the workflow can be stored in a simple `Makefile`, or could be more sophisticated. Some of the best-practices of software engineering [4] advise that to stay on the safe side developing complicated piece of software (which is not uncommon to modern computational experiments), one should iteratively invest in *automated cross-checks* (which in software engineering are called unit tests). Such checks are not something necessary, but might be a good candidate for generation of intermediate report of the experiment that helps researchers to understand overall status of the analysis. The final component of the computational experiment would be the conclusion supported by *concrete number/figures* that should be generatable by executing the certain command/workflow to produce the results from the data (or links to that data).

Just to summarize computational experiment ingredients:

- assumptions;
- data, metadata;
- environment + resources (CPU/GPU);

- code/scripts;about re-usable science, it allows people to jump right into your research code. Lets you launch *Jupyter* notebooks from a git repository with a click of a button.
- workflows;
- automated intermediate results checks;
- final results (datasets, figures).

3. Ideal-world picture

So the overall picture of the reproducible analysis would include the following stages:

- preparation of the environment, specification;
- preparation of the dataset for the analysis;
- preparation of the software repository for storing the code-related items;
- iterative writing/testing analysis code; updating environment specification if necessary;
- automating running of complicated workflows, e.g. as `Makefile`;
- automating checking of the intermediate results;
- as scientific results has been confirmed by the code, one can publish along the paper:
- the repository with the specification, code, workflow, etc;
- the dataset (or some derivative that is required to finish some of the final steps);
- if needed, it is also possible to publish environment snapshot;
- so other members of scientific community interested in the following-up the research can bind repository, datasets and environment together.

As the final set of publishing steps is completed, and one is looking for reproduction of the results, play with the code, etc. Usually the latter step involves a lot of hassle but it is a good place for Everware to come in. Everware (`https://github.com/everware/everware`) is about re-usable science, it allows people to jump right into your research code. Lets you launch your analysis code from a git repository with a click of a button. Namely, Everware can read environment specification from the code repository, create the environment, put all the code/workflow inside this environment and allow user to interact with it. Of course, some additional assumptions should be taken into account and we are describing it in the following sections.

4. Everware-compatible repository

The suggested approach for storing components mentioned in the previous section heavily relies on modern version control system, namely git. Indeed, configuration of the research components can be stored in the form of text files that easily fit into git storage system with all powerful features git provides.

Table 1: Computational research components that could be stored in git repository.

component	**repository entity**
assumptions	`README.md` (and other documentation media)
environment	`Dockerfile`
code/scripts	source code
workflows	`Makefile`, `snakefile`, etc
automated intermediate results check	`circle.yml` (continuous integration config file)
final results	`html`, `pdf`, `png`, etc.

One specific approach we find quite comfortable for storing environment configurations relies on Docker [5]. By means of using virtualization capabilities of modern operating systems, Docker allows for creation of a virtual machine in very light-weight manner compared to traditional approaches. `Dockerfile` is essentially the set of instructions for creating environment that the code should be able to run at. It has been shown by Boettiger [6] that Docker has enough capabilities for purposes of computational research reproducibility: it supports versioning of the environment, it is quite easy to run on local machine (notebook) or on remote server, also you can share images of your environment via centralized docker image storage (docker hub).

The source code of the research can be stored in variety of ways. But there is one quite prominent approach for doing research in interactive way. Initially the project was called IPython notebooks [7], now it is transformed into multi-language system called Jupyter [8]. Such notebooks contain mixture of the source code, rich comments and output in the form of text, tables, figures or even animation. To execute such notebook one has to start Jupyter server and connect to it through the web browser. The environment in which server starts should have all the libraries installed for the code to run. The user interacting with the notebook can re-evaluate all the code and theoretically can get the same output. Jupyter ecosystems nowadays becomes increasingly poplar since it includes plenty of tools that make analysis review considerably simplier to follow (think of it as a shift from the regular to *literate programming*[9]) and it has quite vibrant community to support all those tools.

Jupyter notebooks (or simply 'notebooks') can be executed in batch mode by means of Jupyter nbconvert [10] or 3-rd party modules like runipy [11]. So for the cases when research workflow consists of several steps, it would be quite natural to store commands of those steps either in `Makefile`-like file or in simple `.sh` script file. Such practice might simplify testing of the results through the development of the research.

So one of the possible way of structuring computational research repository would be the following:

- README.md with general information and references;
- `Dockerfile` that contains instructions for the environment configuration;
- Jupyter notebooks with all the code;
- `Makefile` (or `.sh` script) to rule them overall workflow (optional);
- `circle.yml` with instructions to run the workflow at CI systems (optional).

There has been tremendous amount of efforts spent on advocating writing tests for software engineering[4]. Such tests basically help to trace progress of the development and help to monitor the quality of the software. Continuous integration (CI) systems is the de-facto standard for execution of the tests. Those systems could be hosted on-site or publicly available. Usually they differ by capabilities (i.e. support for running code under different operating systems or services support (like Docker)) and configuration approach, but essentially they are quite similar. Examples of the CI system with great capabilities and straightforward configuration include CircleIC [12] and Travis. So borrowing those best practices from software engineering world to scientific research could help . Although it is not strictly necessary for Everware to proceed.

Here are several examples of the repositories that are structured this way: `https://github.com/openml/study_example`, `https://github.com/yandexdataschool/comet-example-ci`, `https://github.com/yandexdataschool/neurohack_dec_2016`.

The latter repository is a preparation for a data analysis hackathon on neural sciences. As mentioned before requirements for reproducibility of an experimental research are similar to requirements for an educational challenge. Indeed a) participants have to bootstrap themselves into problems given, so basic assumptions has to be specified in some way, b) due to limited amount of time for the challenge, participants have to start with already prepared environment,

c) source code has to be executable, understandable and changeable, d) workflows and automated test may help participants to understand complications and constraints of the challenge. On the following picture there is a flow of actions for preparation of the repository for publishing a research (a) and steps required by publisher/organizer of educational challenge and its participants (b).

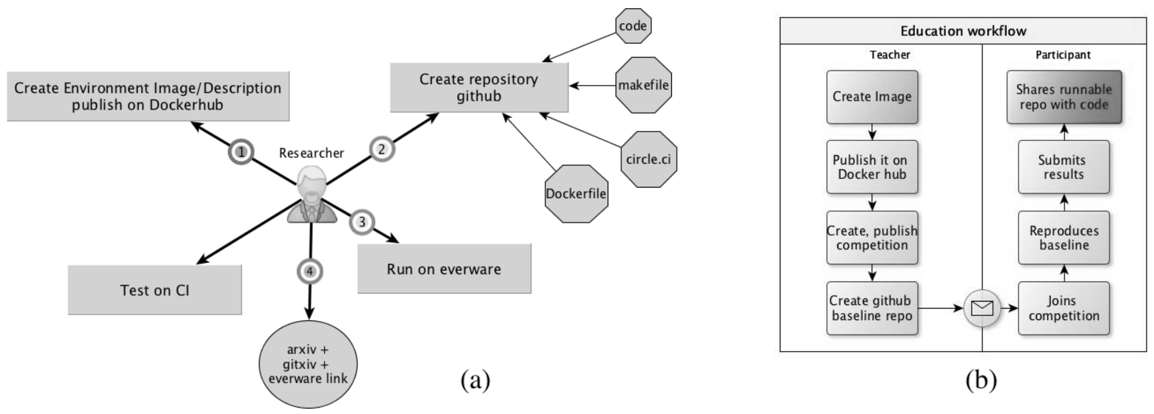

Figure 1: Repository preparation for a) research publishing b) educational challenge

Those steps are pretty well aligned with ideal-world picture presented in the section above, although impose some practical restrictions we'll discuss in the following section.

5. Architecture

Everware is based on JupyterHub[13] component. JupyterHub is a multi-user server that manages and proxies multiple instances of the single-user Jupyter notebook server. Actually Everware enhances Jupyterhub with git authentication and repository cloning functionality. Also Everware enhances spawner component such that it creates custom Docker image from the `Dockerfile` in the repository. Upon successful image creation it is spawned on available Docker cluster. Overall picture is presented on the fig. 2.

Such architecture is pretty simplistic and at the same time flexible enough to support for extensions we'll talk about in the next section.

6. Discussion

Everware provides a powerful mechanism to support reproducibility and to our knowledge it has been used for several research environments and educational challenge setups. As a demonstration stand there is a computational cluster donated by Yandex (`https://everware.rep.school.yandex.net`). It has been used rather extensively for educational and research activities:

- Machine Learning in High Energy Physics summer school 2016 (github link);
- YSDA course on Machine learning at Imperial College London 2016 (github link);
- "Flavour of physics" Kaggle competition, organized in 2015 (github link);
- Machine Learning course at University of Eindhoven (github link);
- LHCb open data masterclass (github link);
- YSDA course on Machine Learning at Imperial College London 2017 (github link);
- Hachathon on Neural Sciences organized by Yandex, HSE (github link);

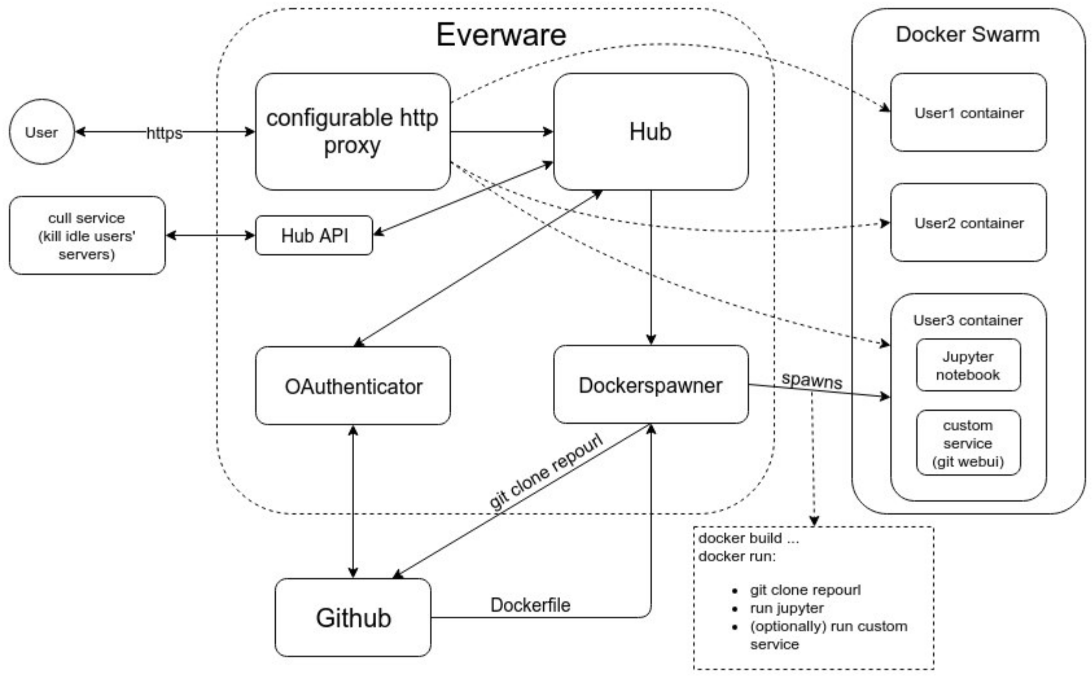

Figure 2: Everware architecture.

- Electro-magnetic showers detection (github link), meta-machine learning research (github link), reinforcement learning (github link).

Based on our experience such approach would stimulate open research by the following aspects:

- make easier supervision/mentoring;
- make easier within-lab validation;
- widen access to the best practices;
- simplify cross-lab validation;
- provide good incentive for formal reproduction;
- is a good thing for industry career track development.

It was mentioned before that those aspects are pretty well aligned with reproducibility boosting factors mentioned in Nature survey [3]. As a cost one would have to pay for those benefits we see the following burden:

- learning a bit of (open-sourced) technology;
- re-organize internal research process;
- inner barrier for openness;
- higher incentive for mindless *borrowing*;
- divergence/potential learning curves (promotes users to create unique environments).

6.1. Open questions, project roadmap

Current limitations of the proposed approach are:

- dependence on Jupyter computational model. This nowadays may be a rather strict suggestion;
- no access to private data sources;
- bottleneck for resource-hungry (either RAM or CPU or disk) analyses - i.e. the system doesn't scales itself with increased demand for computationlly-intensive tasks.

To address those limitations further development of the project would include the following aspects:

- Bring Your Own Resources (BYOR) model. This model would allow to improve a bit complication with resource-hungry analyses, since user would be able to point to his own resources (Amazon, CERN, whatever) and instantiate experiment execution there;
- Jupyter kernel inside separate docker container. This would allow for easier integration of user repositories since it won't require any special things like running Jupyter inside the container and it would be much easier to guess required environment configuration;
- support for custom web interface (e.g. for R shine, or git-webui[14]). This approach would allow for having several services accessible to user from the same container and in case of git-webui simplify interaction with versioning system a lot;
- storage of time-limited user certificates (or proxies) inside docker container during instantiation;
- support automatic capture of the research environment (integration with ReproZip[15]);
- add support for container execution customization, like specification of input file sources or additional container(s) that has to be started along with the main one.

7. Conclusion

In this paper we presented open-source Everware project that is aimed at helping in reproducibility of scientific research and educational activities. The main scope of the project is computational part of the experiment. From the first look the project may seem a bit of a burden on shoulders of researcher but our own experience shows that this price is negligible compared to the reward in the long-term. To help one to get started there are hints and guidelines readily available at: `https://github.com/everware/everware/wiki/Getting-Started`

References

[1] Prinz F, Schlange T and Asadullah K 2011 *Nature reviews Drug discovery* **10** 712–712
[2] Collaboration O S *et al.* 2015 *Science* **349** ISSN 0036-8075 URL `http://science.sciencemag.org/content/349/6251/aac4716`
[3] Baker M 2016 **533**(7604) URL `http://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970`
[4] Kolawa, A; Huizinga, D 2007 *Automated Defect Prevention: Best Practices in Software Management* (Wiley-IEEE Computer Society Press) ISBN 0470042125
[5] *Docker documentation* URL `https://docs.docker.com`
[6] Boettiger C 2014 *CoRR* **abs/1410.0846** URL `http://arxiv.org/abs/1410.0846`
[7] Pérez F and Granger B E 2007 *Computing in Science and Engineering* **9** 21–29 ISSN 1521-9615 URL `http://ipython.org`
[8] *Jupyter documentation* URL `https://jupyter.readthedocs.io/en/latest/content-quickstart.html`
[9] Knuth D E 1984 *The Computer Journal* **27** 97–111
[10] *nbconvert: Convert Notebooks to other formats* URL `http://nbconvert.readthedocs.io/en/latest/index.html`
[11] *runipy: run IPython as a script* URL `https://github.com/paulgb/runipy`
[12] *CircleCI documentation* URL `https://circleci.com/docs/`
[13] *Multi-user server for Jupyter notebooks* URL `https://jupyterhub.readthedocs.io/`

[14] *A standalone local web based user interface for git repositories* URL `https://github.com/alberthier/git-webui`

[15] F Chirigati, R Rampin, D Shasha, and J Freire 2016 *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD)* 2085–2088