A Reproducible Tutorial on Reproducibility in Database Systems Research

Tim Fischer
tim.fischer@uni-tuebingen.de
Eberhard Karls Universität
Tübingen, Germany

Denis Hirn
denis.hirn@uni-tuebingen.de
Eberhard Karls Universität
Tübingen, Germany

Gökhan Kul
gkul@umassd.edu
University of Massachusetts
Dartmouth, USA

ABSTRACT

Reproducibility is a key aspect of the scientific method, and it is essential for building trust in the results of research. This tutorial aims to provide concrete guidance on how to leverage **containerized reproducibility** using Docker for database systems research.

In this tutorial, we present a step-by-step guide on how to prepare a Docker-based artifact for an experiment. We will cover topics such as Dockerfiles, Docker images, Docker Compose, automation using Python, Bash, and Make, and also artifact documentation and packaging best practices. The tutorial itself is a reproducible artifact, and we provide a public GitHub repository with all the code and examples used in the tutorial. This repository can serve as a starting point to prepare artifacts for experiments and publications.

PVLDB Reference Format:
Tim Fischer, Denis Hirn, and Gökhan Kul. A Reproducible Tutorial on Reproducibility in Database Systems Research. PVLDB, 17(12): 4221 - 4224, 2024.
doi:10.14778/3685800.3685840
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/db-reproducibility/template.

1 INTRODUCTION

The *credibility crisis* [9, 11] in computer science research is emerging as a major concern. The lack of reproducibility undermines the credibility of the field as a whole. In database systems research, the problem of reproducibility is particularly acute due to the complexity of the experimental setups and the wide range of parameters that can affect results [14]. However, it is a key aspect of the scientific method, and it is essential for building trust in the results of research [10, 13].

The DB community has recognized the importance of reproducibility and has started to take action [4, 5, 12]. However, the term *reproducibility* is often used in a very broad sense, encompassing a wide range of practices and techniques. For the purposes of this tutorial, we define reproducibility as being able to reach to the same conclusions through experimental repetitions of author's original artifacts by independent researchers. Therefore, **the key to reproducible experiments is to design them with reproducibility in mind, not as an afterthought.**

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685840

This tutorial aims to provide concrete guidance on how to leverage **containerized reproducibility** using Docker, with a strong focus on the practical aspects of artifact preparation. We propose a 90-min tutorial in which we will cover the following topics:

- Introduction to *repeatability* and *reproducibility* for experimental setups (5 mins).
- Brief introduction to containerization technologies (5 mins).
- Preparation of a Docker-based artifact for experiments and publications fit for the submission to the reproducibility efforts of PVLDB or SIGMOD (30 mins).
- Automation using Python, Bash, and Make (20 mins).
- Multi-container setups using Docker Compose (10 mins).
- Artifact documentation and packaging best practices (10 mins).

Target Audience and Learning Objectives. The tutorial is designed to be beginner friendly. It is aimed at researchers who are new to reproducibility efforts and interested in learning how to prepare artifacts for their experiments. We will provide hands-on examples and practical advice on how to prepare artifacts for experiments and publications. Participants will learn how to prepare a Docker-based artifact for an experiment. We will provide a step-by-step guide on how to create a Dockerfile, build a Docker image, and run an experiment inside a container. The tutorial will be interactive, and we will encourage participants to ask questions and share their experiences.

We provide a public GitHub repository with all the code and examples used in the tutorial. This repository is prepared in the style of a research artifact that could be submitted to reproducibility efforts, *making this tutorial itself a reproducible artifact.* This repository can also serve as a starting point for authors who want to prepare artifacts for their own experiments. During the tutorial, participants will be able to follow along and run the examples on their own machine.

2 CONTAINERIZED REPRODUCIBILITY

Both VLDB [4] and SIGMOD's [5] reproducibility efforts claim that *"Ideally, reproducibility should be close to zero effort"*. In practice, however, achieving reproducibility can be challenging due to the complexity and the wide range of experimental setups. This can be a problem for authors replicating their own experiments, for reviewers, and—most importantly—for our fellow colleagues trying to reproduce and build upon them.

Using containerization technologies such as Docker [1] is one way to simplify the process of reproducing experiments [7]. In this section, we will give a brief introduction to containerization technologies and explain how they help to achieve the goal of *"reproducibility should be close to zero effort"*.

Docker is a platform to develop, deploy and run applications in *containers*. Containers are lightweight, self-contained, executable packages that contain everything needed to run an application, including code, runtime, system tools, libraries, and settings. Containers are isolated from each other and from the host system, making them a great tool for reproducibility. Wrapping an experiment in a container ensures that an experiment will run the same on any machine with Docker installed. This eliminates the need to manually install dependencies and configure the environment, which can be a major source of error when trying to reproduce an experiment. Also, containers are portable, which means you can easily share them with others and run them on different machines. This makes it easier for reviewers to reproduce experiments and verify results, as they can simply download the container and run it on their own machine. No additional environment setup is required. Note that Docker does not simplify hardware requirements. If an experiment requires a specific hardware setup, it must still be provided by the user.

The following sections provide a taste of the concrete and actionable information attendees will take home.

2.1 Dockerfiles and Docker Images

Docker *images* are the building blocks of containers and are specified in a so-called *Dockerfile*, which contains a series of commands that build the image. An image is a read-only template for creating a container. Each Dockerfile specifies a base image, *e.g.*, a Linux distribution such as Ubuntu or Alpine. But there are also images for database systems, programming languages, and other software. One repository that provides pre-built images is *Docker Hub* [3].

However, the pre-built images on Docker Hub may not always be suitable for an experiment. In this case, it is necessary to create a custom Docker image. Figure 1 shows an example Dockerfile for building and running DuckDB [15]. The Dockerfile specifies an Ubuntu base image, installs the necessary dependencies, clones the DuckDB repository, builds the system, and defines the entrypoint for running DuckDB. The `docker build` command is used to compile a Dockerfile into an image. The resulting image can then be run with the `docker run` command to create a container.

It is also possible to use local files in the Docker image. This can be useful for including data files, configuration files, or scripts that are needed for the experiment. The **`COPY`** command can be used to copy files from the host system into the image. However, big files should not be included in the image, as this would make it unnecessarily large. Instead, it is better to generate the files during startup of the container, or download them from a remote location. The resulting pre-compiled Docker image could be the artifact that is shared with others to reproduce the experiment.

Note that the Dockerfile in itself already contains a lot of documentation about a part of the experiment. The layer of abstraction that a Dockerfile provides makes it easier to change database versions or experiment parameters later on.

2.2 Docker Compose

For more complex experiments that require multiple containers to be run together, Docker Compose [2] can be used. This tool simplifies the definition and execution of multi-container Docker applications. It uses a `docker-compose.yml` file to define the services that make up the application, and then starts all the services with a single command.

```dockerfile
# Use Ubuntu as the base image
FROM ubuntu:22.04 AS duckdb
# Install the necessary dependencies
RUN apt-get update && apt-get install -y \
    build-essential cmake git
# Clone the DuckDB repository
RUN git clone --depth 1 --branch v0.10.1 \
        https://github.com/duckdb/duckdb.git
# Build DuckDB
WORKDIR /duckdb
RUN make release -j
# Add the DuckDB binary to the PATH
ENV PATH="/duckdb/build/release:${PATH}"
# Define the entrypoint
ENTRYPOINT ["duckdb"]
```

Figure 1: Sample Docker file to build and run DuckDB.

The example in Figure 2 defines two services, one for PostgreSQL using the official image from Docker Hub, and one for DuckDB using the custom image that was built with the Dockerfile from Figure 1.

```yaml
services:
  postgres:
    image: postgres:16
  duckdb:
    image: duckdb
    build:
      context: .
      dockerfile: Dockerfile
```

Figure 2: Example `docker-compose.yml` file for running DuckDB and PostgreSQL.

2.3 Automate as Much as Possible

Docker images and Docker Compose provide the environment to run an experiment. However, starting the containers, running the experiment, and collecting the results is still a manual process. This is where automation comes in. We recommend to automate as much of an experiment as possible.

Automation Script. One way to simplify the process of reproducing an experiment is to provide a single script that automates the experiment. This script should take care of executing all the necessary commands, collecting the results in standardized formats (*e.g.*, CSV files), and finally generating the plots and tables that are presented in the paper. It is also possible to automatically generate the paper PDF from the newly measured results. Creating these figures and tables is crucial because it allows reviewers to verify the results and compare them to the results presented in the paper.

This script should also be run inside the container. That way, it can in principle be written in any language. However, we recommend using common languages such as Python or Bash because these are widely used and easy to understand.

Make. Besides the automation script, it is good practice to have a Makefile that automates the startup of the containers and the initial execution of the automation script. This takes out any guesswork for reviewers. The Makefile should also include targets for cleaning up the environment after the experiment has been run (see Figure 3 for an example).

```
run: # Run the experiments
  @echo "Setup environment to run experiments"
  @docker compose up -d
  @docker compose exec postgres bash
      "run-experiments.sh"
clean: # Clean up the environment
  docker compose down -v --rmi local
```

Figure 3: Sample Makefile for running SQL experiments.

Auditability. Automation scripts should not be *black boxes*. They should be well-documented and easy to understand, so that reviewers can see how the experiment was conducted. One way to achieve this is to keep the user interface simple and provide detailed comments in the script that explain what each step does. But also, during the execution of the script, it should print out information about what it is doing, so that the user can follow along and see what is happening. This is important for transparency and auditability.

If the script fails, the user should be able to easily identify the problem and fix it. Ideally, the script should be designed in such a way that it can be run incrementally, so that the user can restart it at any point without having to start from the beginning. This is especially important for long-running experiments, where it would be problematic to start from scratch every time the script fails. The script should also be designed to be idempotent, so that it can be run multiple times without causing any side effects.

Experimental Separation. Figure 4 shows an example Bash script that runs a series of SQL experiments and measures the time it takes to run each experiment. The script writes the results to a CSV file, which can then be used to generate plots and tables for the paper. Adding error handling and retry mechanisms to the script would make it more robust and reliable, but is omitted here for brevity.

Experiments should ideally be standalone scripts that can run independently. This makes it easier to run individual experiments, debug, and troubleshoot. Each experiment should be self-contained and not dependent on the state of others. In Figure 4, Line 3, the script assumes that all experiments are stored in a folder called `experiments`. This makes it easy to add new experiments by simply adding a new SQL file.

3 ARTIFACT DOCUMENTATION

The final step in preparing the artifact is to document it properly. This can be done in several ways, depending on the guidelines of the venue where the artifact will be submitted. In general, it is

```
1  #!/bin/bash
2  echo "experiment, time" > results.csv
3  for EXPERIMENT in experiments/*.sql; do
4    echo "Running experiment $EXPERIMENT"
5    # Run the experiment and measure the time
6    OUTPUT=`psql -c "\timing on" -f "$EXPERIMENT"`
7    # Extract the time from the output
8    TIME=`echo "$OUTPUT" | grep -o -E "Time: \S+"
9                          | tail -n1 | sed 's/Time: //'`
10   # Print the time and write it to the results file
11   echo "Time: $TIME"
12   echo "$EXPERIMENT, $TIME" >> results.csv
13 done
```

Figure 4: Sample Bash script for running a series of SQL experiments.

a good idea to provide a README file that explains the steps to reproduce the experiment in a tutorial-style format (see Figure 5 for an example). This README file should contain the following information:

Overview. A brief description of the experiment and the purpose of the artifact.

Software Requirements. Explicitly list the software that is required to run the artifact. This should include the operating system, programming languages, libraries, and any other software that is needed.

Hardware Requirements. Information about the approximate hardware that is required to run the experiment. This should include the amount of memory and CPU that is needed, as well as any specialized hardware that is required.

Time Requirements. An estimate of the time it takes to run the experiment. This is important for reviewers to plan their time accordingly. If the experiment is very time-consuming, it may be necessary to provide a simplified version that can be run in a reasonable amount of time.

Cleanup. Instructions on how to cleanup any resources that were created during the experiment.

An example of our README file and the GitHub repository can be found in the link given below[1].

4 CHALLENGES AND OPEN QUESTIONS

Containerization is a powerful tool for reproducibility, but it is not a silver bullet. There are cases where containerization may not be viable or may not be the best solution. For example, if your experiment requires access to specialized hardware, or software that is not available in a container, then containerization may not be a good fit. Also, if your experiment is very resource-intensive and requires a lot of memory or CPU, then providing a container that can run the experiment may not suffice to reproduce the results.

In these cases, it may be necessary to coordinate with the chairs of the reproducibility committee to find an alternative solution. For example, you may be able to provide remote access to the hardware or software that is required for the experiment. Alternatively, a

[1] `https://github.com/db-reproducibility/template`

```
# Overview
This repository contains the artifacts for [...]

# Software Requirements
The artifact requires Docker and Docker Compose.

# Hardware Requirements
At least 4GB of memory and 2 CPU cores are required.

# Time Requirements
* Getting Started: 5 minutes
* Running the experiments: 15 minutes
[...]

# Reproducibility
To reproduce the experiments, follow these steps:
[...]

# Cleanup
To cleanup, run the following commands:
[...]
```

Figure 5: Example README file for an experiment.

simplified version of the experiment that can be run even on *commodity hardware* may be sufficient to verify the claims from the publication. If applicable, we believe that the latter one should be the preferred solution, because reproducibility is not a one-time event. It is a continuous process that allows others to build on your work and extend it.

These *outliers* should be the exception rather than the rule. In the community, there is currently no clear guideline on how to handle these cases. We believe that there should be discussions to establish guidelines on how to handle such cases. However, for the vast majority of experiments, containerization is a great tool for reproducibility, and we encourage authors to use it along with appropriate documentation whenever possible.

5 PRESENTERS

We will be able to shed a light on reproducibility from a variety of angles: as co-chairs of the reproducibility effort of a major database conference (PVLDB), as reviewers in these reproducibility efforts, and as authors and submitters of research items that were reviewed.

Tim Fischer and Denis Hirn are PhD students at the University of Tübingen working on database systems research under the supervision of Torsten Grust. They have experience with containerized reproducibility and have used Docker extensively in their own research to prepare artifacts for experiments that were successfully tested for reproducibility. Their research group has won the ACM SIGMOD 2021 Reproducibility Award. Denis Hirn has also been involved in the SIGMOD reproducibility committee since 2022 as a reviewer. Also, his own research has been published with artifacts that were prepared using Docker.

Gökhan Kul serves as the Co-Chair of the Reproducibility Committee of VLDB since 2023, after serving in the VLDB and SIGMOD reproducibility committees as a reviewer since 2016. He is an Assistant Professor at the University of Massachusetts Dartmouth, and his research focuses on database and software security fields.

6 RELATED WORK

There are a number of previous works that have covered related topics. For example, at ICDE 2021 Mauerer and Scherzinger [12] have presented a tutorial on long-term reproducibility and best practices for preparing artifacts, also using Docker. Bajpai et al. [6] presents a beginner friendly introduction to reproducibility, including best practices for documentation, data collection, and tool recommendations. Chirigati et al. [8] present ReproZip, a tool for automatically creating reproducible experiments from existing ones. ReproZip captures the environment of an experiment and packages so can be shared. This is compatible with our tutorial, but ReproZip is more focused on capturing the environment of an existing experiment, while we focus on preparing artifacts from scratch.

REFERENCES

[1] [n.d.]. Docker. https://www.docker.com/. Accessed: 2024-04-01.

[2] [n.d.]. Docker Compose. https://docs.docker.com/compose/. Accessed: 2024-04-01.

[3] [n.d.]. Docker Hub. https://hub.docker.com/. Accessed: 2024-04-01.

[4] [n.d.]. PVLDB Reproducibility. https://vldb.org/pvldb/reproducibility/. Accessed: 2024-04-01.

[5] [n.d.]. SIGMOD Reproducibility ARI. https://reproducibility.sigmod.org/. Accessed: 2024-04-01.

[6] Vaibhav Bajpai, Anna Brunstrom, Anja Feldmann, Wolfgang Kellerer, Aiko Pras, Henning Schulzrinne, Georgios Smaragdakis, Matthias Wählisch, and Klaus Wehrle. 2019. The Dagstuhl beginners guide to reproducibility for experimental networking research. *ACM SIGCOMM* 49, 1 (2019), 24–30.

[7] Carl Boettiger. 2015. An introduction to Docker for reproducible research. *SIGOPS Oper. Syst. Rev.* 49, 1 (jan 2015), 71–79. https://doi.org/10.1145/2723872.2723882

[8] F. Chirigati, R. Rampin, D. Shasha, and J. Freire. 2016. ReproZip: Computational Reproducibility With Ease. In *ACM SIGMOD*. 2085–2088. https://doi.org/10.1145/2882903.2899401

[9] David L Donoho, Arian Maleki, Inam Ur Rahman, Morteza Shahram, and Victoria Stodden. 2008. Reproducible research in computational harmonic analysis. *Computing in Science & Engineering* 11, 1 (2008), 8–18.

[10] Yolanda G., C. H. David, I. Demir, B. T. Essawy, R. W. Fulweiler, J. L. Goodall, L. Karlstrom, H. Lee, H. J. Mills, J. Oh, Ss A. Pierce, A. Pope, M. W. Tzeng, S. R. Villamizar, and X. Yu. 2016. Toward the Geoscience Paper of the Future: Best practices for documenting and sharing research from data to software to provenance. *Earth and Space Science* 3, 10 (2016). https://doi.org/10.1002/2015EA000136

[11] Benjamin Haibe-Kains, George Alexandru Adam, Ahmed Hosny, Farnoosh Khodakarami, Massive Analysis Quality Control (MAQC) Society Board of Directors, et al. 2020. Transparency and reproducibility in artificial intelligence. *Nature* 586, 7829 (2020), E14–E16.

[12] W. Mauerer and S. Scherzinger. 2021. Nullius in Verba: Reproducibility for Database Systems Research, Revisited. In *IEEE 37th ICDE*. https://doi.org/10.1109/ICDE51399.2021.00270

[13] T. Neumann. [n.d.]. Experiments Hurt the Review Process. https://databasearchitects.blogspot.com/2014/09/experiments-hurt-review-process.html. ([n. d.]). Accessed: 2024-04-01.

[14] M. Pawlik, T. Hütter, D. Kocher, W. Mann, and N. Augsten. 2019. A link is not enough–reproducibility of data. *Datenbank-Spektrum* 19 (2019), 107–115.

[15] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*. 1981–1984.