Project Documentation

Enhance MeTA Toolkit and Metapy usability

David McGuire and Jose Cols

{dmcguire,josec5}@illinois.edu

University of Illinois, Urbana-Champaign

CS 410: Text Information Systems

December 7, 2022

Functionality Overview

This project aims to improve existing support for the MeTA and Metapy libraries (Massung et al., 2016). As a result, the functionality offered here is centered on expanding compatibility and portability to these systems, particularly in the context of using them to run MP assignments for CS 410 Text Information Systems.

MeTA

This library's existing functionality is unaltered. The changes are entirely focused on upgrading dependencies and correcting build and Continuous Integration issues caused by four years of neglect. Furthermore, new features were added to support containerized workflows, increasing the library's overall portability and supportability. Specifically, the new functionality created for this repository is:

- Correct the CMake compilation of C++ source code, allowing the construction of programs that rely on MeTA, such as Metapy.
- Add support for containerized workflows based on Docker, such as automatically publishing a Docker Image containing a pre-built version of MeTA. This image enables users to run MeTA binaries or compile C++ programs using MeTA as a dependency, abstracting the complexity of installing and configuring the required tooling.

Metapy

Metapy provides Python bindings for the MeTA toolkit, hence its primary functionality is based on MeTA's capabilities, which, as previously said, remain unmodified. However, there is significant work being accomplished to extend support for newer versions of Python (3.7 - 3.10), fix compilation issues, and make other compatibility improvements aimed at delivering a better user experience. Specifically, these enhancements include:

- Include support for newer and currently maintained Python versions (3.7 - 3.10). This allows users to use Metapy in their modern processes without relying on outdated infrastructure.
- Publish Python wheels from the aforementioned versions for Linux and macOS systems. Given Metapy's reliance on native packages, pre-built wheels for major development platforms are critical to making development easier.
- Modify Jupyter Notebook tutorials to be compatible with Google Colab, which is available to all students as part of the University of Illinois at Urbana-Champaign IT offering. This enhancement allows workflows with a minimum of zero to begin coding friction.
- Add support for Docker-based containerized workflows, such as automatically releasing a Docker Image with a pre-installed version of Metapy. Docker users can now run Python scripts or generate Python packages that rely on Metapy.

Implementation Overview

This section goes into the implementation details of the previously described functionality. Similarly, it is divided by the two repositories established on this project. Table 1 contains all of the implementation changes, as well as links to GitHub Pull Requests that provide more technical details.

MeTA

The ICU reference in the "CMakeLists.txt" file was updated to fix the CMake compilation of the C++ source code. ICU is a collection of C/C++ libraries that provide Unicode

and Globalization support for software applications (The Unicode Consortium, 2022), and it is required for MeTA to support UTF.

In addition, Docker support was introduced in the form of a Dockerfile and a publicly available pre-built Docker image. The Dockerfile provides two additional significant use cases. First, it serves as a starting point for more advanced users who may choose to extend or modify it to include additional capabilities. However, it still abstracts the complexity of installing the necessary tooling and dependencies to compile MeTA. Second, it includes precise configuration instructions that users can utilize to reproduce the build in systems other than Docker.

The Dockerfile is based on Ubuntu 22.04 LTS, which will be supported until 2032 (Ubuntu Foundation, 2022). Similarly, it makes use of the most recent Clang tooling (Clang 14, Libc++ 14, Libc++ ABI 14, and LLD 14) available for this distribution to extend the life of the Docker image. This implementation compiles MeTA as a library and installs it in the directories "/usr/local/lib" and "/usr/local/include", allowing users to reference it in C++ applications. Furthermore, it adds the generated binaries to the system's PATH, allowing all command tools to be invoked globally.

This project uses three GitHub Workflows to ensure the continuous integration of the Docker image on any future changes to the MeTA library or the Ubuntu 22.04 distribution. The first is in charge of creating and publishing the Docker image for ARM64 and AMD64 platforms to DockerHub, a hosted repository service to share container images. This workflow is triggered when new code is merged to the main repository branch (i.e., "submodule/metapy") or when the second workflow activates it. This second workflow runs every day to see if the MeTA base Docker image is out of date. If this is the case, it will start a new build to update the base image and compile MeTA for the new release. Finally, the third workflow runs on every push for every

branch, providing a means to ensure that the changes are working properly. The image created by this step is private and only accessible to repository maintainers in GitHub Packages.

Metapy

The majority of Metapy's implementation developments are focused on creating Python Wheels for multiple platforms (Python 3.7 - 3.10 for Linux and macOS x86_64). This project enhances the existing Travis CI infrastructure with new configurations and commands to fix pipeline failures and introduce compatibility with newer Python versions to achieve this goal. These improvements center on changing Travis CI configuration YAML files (i.e., ".travis.yml") and reworking a number of shell script files, including "build_linux_wheel.sh", "install_linux.sh", "build_linux.sh", "clean_build.sh", and "install_osx.sh". These scripts are in charge of installing the compilation tools and dependencies needed to compile Metapy and package it as a Python Wheel on each operating system. Although the broad strategy to fixing the build is similar across platforms, each has its own set of challenges that demand individual debugging and testing.

In addition, the "pybind11" and "meta" Git submodules were updated. The new MeTA submodule references the MeTA fork generated in this project, which overcomes the toolkit compilation problems. Similarly, pybind11 was updated to version 2.3.0 and used to build new C++ header files to correct a bug in the Python bindings caused by the switch from TLS to TSS in Python 3.7 (Bray, 2016).

Furthermore, the existing Jupyter Notebook tutorials have been updated to work with Google Colaboratory. The new notebooks include a top Python cell that installs the "metapy" package using the wheels published in the project repository. The notebooks also include a quick

link to open the tutorial on Google Colaboratory, allowing any user with a Colab account to run the tutorials without ever having to set up a local Python environment.

Finally, this project includes another Dockerfile that encapsulates a Python environment (version 3.10) pre-installed with Metapy. This Dockerfile inherits from the MeTA Docker image, which already has a pre-built version of the toolkit. This image also includes MeTA as a library, which is needed to compile the Python bindings in order to generate and install the Python wheel. To compile these bindings, the Dockerfile modifies the existing "CMakeLists.txt" file to add a "find_package" command that looks for the pre-installed version of MeTA. A GitHub Workflow, like for the MeTA Docker image, initiates the Dockerfile build whenever the "master" branch is updated. Using Docker Buildx, this build creates an image for the ARM64 and AMD64 platforms, which is published on DockerHub.

Usage Guide

Because metapy, written in Python, is simply a wrapper around MeTA, written in C++, the code that is built is not portable across versions of Python, which may interact with wrapped C++ differently, nor across machine architectures, which expect compiled C++ that is targeted to their specific architecture. The Python community's solution to this conundrum is called the The Wheel Binary Package Format, and this is how metapy is built and distributed. A built package using this standard is called a "wheel". Luckily, the standard package manager for Python, called "pip", works natively with wheel packages, and can either find them on the Python Package Index (PyPI, for short), an API-compatible clone of that site, or, if they're hosted elsewhere, can take a direct http URL to the built artifact. Because PyPI support in Github Packages is outstanding, the current installation is via direct link, and must encode the Python version and

system architecture in the URL; for example, to install Python 3.8 on Linux running on an AMD 64 processor, use the following command:

```
pip install
https://github.com/illinois/metapy/releases/download/v0.2.14/met
apy-0.2.14-cp38-cp38-manylinux_2_24_x86_64.whl
```

Unfortunately, the coverage of Python versions and systems is pretty far from complete; notably, Windows and Python 3.11 are both missing. Luckily, two different virtualization strategies have been applied to limit the need for local use of metapy. The first of these virtualization strategies is Google Colaboratory or Google Colab, for short. Each of the (working) tutorials shipped with metapy has been upgraded to work with Google Colab, and they all now include a link (illustrated in Figure 1), navigable from inside the GitHub repository, to open them in Google Colab, where they may be executed. Because each UIUC student may access Google Colab through their university Google Apps account, it means that students may use these tutorials as a jumping off point for exploration of metapy features.

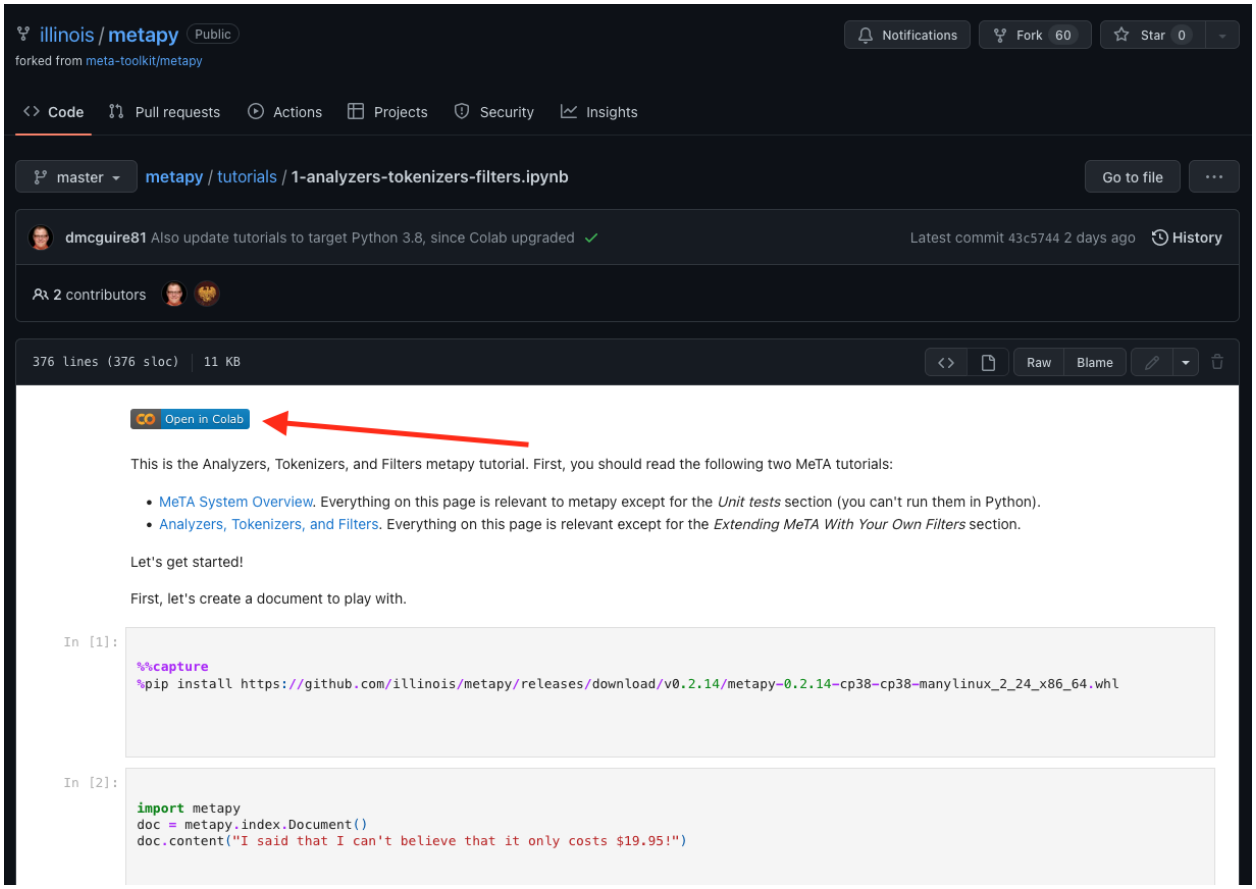
The second virtualization strategy supported is a Docker container, which is hosted on DockerHub. To get started with a Docker container for metapy, use the following commands:

```
docker pull josecols/metapy:0.2.14
docker run -it --rm --name metapy --mount
type=bind,source=$(pwd),target=/app --entrypoint bash
josecols/metapy:0.2.14
```

Additional usage information is included in the "README.md" file for each repository. These contain a "Getting started" section that explains how to run "metapy" on Google Colaboratory using Jupyter Notebooks, Python wheels, and Docker.

Figure 1

A screenshot of the "Open in Colab" link in the tutorials in GitHub.



Member Contributions

Table 1 depicts a summary of all the tasks completed by each team member (intermediate fixes are omitted), grouped by repository. It also includes a link to the GitHub Pull Requests where the work was implemented, which includes additional technical information. Furthermore, both team members contributed to the creation of project documentation and the organization of essential tasks.

Table 1

List of Pull Requests with corresponding author and task description

Author	Repository	Task	Pull Request
David	meta	Fix MeTA's ICU dependency.	https://github.com/illinois/meta/pull/1
Jose	meta	Add Docker support and image builds with continuous integration.	https://github.com/illinois/meta/pull/2
Jose	meta	Add up-to-date checks to base Docker image with continuous integration.	https://github.com/illinois/meta/pull/3
Jose	meta	Add GitHub Packages support for internal Docker images.	https://github.com/illinois/meta/pull/6
David	metapy	Update the MeTA Git submodule reference.	https://github.com/illinois/metapy/pull/1
David	metapy	Fix Travis CI build dependencies for Linux.	https://github.com/illinois/metapy/pull/2
David	metapy	Fix Travis CI build time-outs for macOS.	https://github.com/illinois/metapy/pull/4
David	metapy	Add support for currently-maintained Python versions (3.7 - 3.11).	https://github.com/illinois/metapy/pull/5
David	metapy	Update the pybind11 Git submodule reference.	https://github.com/illinois/metapy/pull/6
David	metapy	Add new Python wheel installation and Google Colab support to Jupyter Notebook tutorials	https://github.com/illinois/metapy/pull/7
Jose	metapy	Add Docker support and image builds with continuous integration.	https://github.com/illinois/metapy/pull/8
Jose	metapy	Add GitHub Packages support for internal Docker images.	https://github.com/illinois/metapy/pull/13
Jose	metapy	Add usage guide to the README file.	https://github.com/illinois/metapy/pull/14/files

References

Sean Massung, Chase Geigle, and ChengXiang Zhai. 2016. MeTA: A Unified Toolkit for Text Retrieval and Analysis. In *Proceedings of ACL-2016 System Demonstrations*, pages 91–96, Berlin, Germany. Association for Computational Linguistics.

Bray, E. M. (2016, December 20). *PEP 539 – A New C-API for Thread-Local Storage in CPython*. Python Enhancement Proposals. Retrieved December 7, 2022, from https://peps.python.org/pep-0539/

Ubuntu Foundation. (2022, October 24). *Releases*. Ubuntu Wiki. Retrieved December 7, 2022, from https://wiki.ubuntu.com/Releases

The Unicode Consortium. (2022). *International Components for Unicode: ICU*. International Components for Unicode: ICU. Retrieved November 13, 2022, from https://icu.unicode.org/home