November 24, 2021

OSSI proposal: User friendly deployment of JVM based tools

Caleb Hulbert, John Bogovic and Stephan Saalfeld

Motivation

Thanks to the fast and scalable *n*-dimensional data processing library ImgLib2 (Pietzsch, Preibisch, Tomančák, and Saalfeld 2012), we have developed a number of useful tools for annotation, interactive transformation, and exploration of large image data in Java[1] and the Kotlin programming language.[2] While the Java Virtual Machine (JVM) runs on all major platforms (Linux, MacOS, Windows, Android, and others), end-user friendly deployment of applications remains difficult. The difficulty is not to run the application but to provide (1) an OS native launcher, (2) robust dependency management for operating system (OS) native libraries and components running in the JVM that is compatible with OS native package managers, and (3) frequent robust updates through integration with OS native package managers. Numerous projects address this issue with mixed success:

The Eclipse project[3] provides tools to create OS native launchers and hosts and integrates market repositories to install and update components. Unfortunately, these capabilities are tightly integrated with its Standard Widget Toolkit (SWT) and components are self-managed. Particularly when the Eclipse platform was installed through an OS native package manager, using the internal component manager can lead to conflicts and broken installations.

The Fiji project[4] (Schindelin et al. 2012) provides an OS native launcher for ImageJ and manages components and plugins with an internal package manager. Unfortunately, this package manager depends on write access to the application directory which makes system-wide Fiji installations managed by OS native package managers impossible. Furthermore, the Fiji package manager does not resolve dependency conflicts and shared dependencies managed in different sources can easily lead to version skew and broken installations.

The Conda project[5] aims at solving version skew by managing virtual environments with independent sets of components. Conda, however, focuses on Python-based applications, provides no dedicated support for JVM-internal dependencies, and the

[1] OpenJDK: https://openjdk.java.net/
[2] Kotlin: https://kotlinlang.org/
[3] Eclipse: https://www.eclipse.org/
[4] Fiji: https://fiji.sc/
[5] Conda: https://docs.conda.io/en/latest/

management of components in virtual environments can interact unpredictably with OS native package management.

The Maven project[6] provides a build system that manages versioned dependencies to compile JVM based applications but does not provide support to manage OS native dependencies and has very rudimentary capabilities to run applications.

The JGO project[7] is a Python launcher for JVM based applications that uses Maven to manage JVM-internal dependencies. It can be installed with conda which can be used to manage OS native dependencies.

Several of our applications[8] are distributed with Fiji where maintenance of updates is difficult and fragile because no automatic dependency management exists and individual components interact with a very large environment of dependents and dependencies. Other special purpose tools come without a native launcher and defer to the user (or an IDE such as Eclipse) to resolve and provide dependencies, and to start the appropriate class.

The standalone N5 utilities[9] use an ad-hoc platform dependent installation script for Linux and MacOS that uses Maven to resolve and install JVM-internal dependencies and creates an OS native launch script that sets the appropriate class path for the JVM. This ad-hoc installer is difficult to maintain and has broken repeatedly over the last few years. It also lacks appropriate management for OS native dependencies like the blosc[10] shared libraries, particular for newer versions of MacOS. Installation is possible on Windows by reproducing the steps performed by the installation script and some Windows specific path editing, but an automatic installer has not been developed for lack of time.

The JavaFX application Paintera[11] uses Conda to manage OS native dependencies and JGO to manage JVM-internal dependencies and to provide an OS native launcher. While the application does not interact with Python, it currently requires the installation of a complete Python envorinment to launch JGO. The launcher must be started from the conda terminal which is not user friendly.

Goals and milestones

The goal of this project is to develop a packaging tool chain for tools with JVM and non-JVM dependencies, and to establish standardized distribution channels for these tools at Janelia. The proposed efforts will make installation easier, more reliable for end-users, and thereby reduce long-term software maintenance costs for developers while improving the user experience as well as the visibility and accessibility of the tool. The reproducibility of scientific software will be improved, as software will be less likely to break, e.g. due to version skew. While the initial focus will be tools developed and distributed at Janelia, the results are expected to translate seamlessly to the wider open source community and will simplify the distribution of such tools worldwide.

[6] Maven: https://maven.apache.org/
[7] JGO: https://github.com/scijava/jgo
[8] Saalfeld lab projects: https://github.com/saalfeldlab
[9] N5 utilities: https://github.com/saalfeldlab/n5-utils
[10] Blosc: https://github.com/Blosc/c-blosc
[11] Paintera: https://github.com/saalfeldlab/paintera

Java 14[12] introduced the Java Packaging tool[13] which became an official part of the standard distribution with Java 16.[14] The Java Packaging tool aims at generating OS native packages and launchers for JVM applications that contain a Java runtime environment, JVM dependencies and resources, as well as a native launcher. The tool uses Java's module system to reduce the footprint of the package to the minimal set of modules required to run the application. We do not know how OS native dependencies are handled and if it is possible to share runtimes and dependencies with other packages. However, the jpackage tool is highly configurable and the structure of OS native packages is well documented. We therefore expect that a solution can be built that enables sharing dependencies and using the OS native package manager to update the application.

Exploration

At first, the jpackage tool needs to be explored and understood. Experimental OS native packages for the N5 utitilities should be created and tested. We will have to develop an understanding of what the benefits and shortcomings of bundling a stripped down Java runtime with a fat set of dependencies is, how OS native dependencies can be handled (in the case of N5 utilities we want to support compression with blosc), and how it can be expanded to release individual components as OS native shared libraries and use an OS wide JVM installation.

Consolidation

Building on our understanding of the packaging and release process, we will streamline the release of OS native packages and distribute them through a dedicated Downloads section of the GitHub repositories and through the standard software channels of modern operating systems, e.g. for Ubuntu Linux, we will establish a Janelia PPA. Distributions of the N5 utilities, Paintera, tools for deformable registration and conversion of large datasets (Bogovic, Hanslovsky, Wong, and Saalfeld 2016; Bogovic et al. 2020), and tools to manually proofread and modify intermediate results in the volume reconstruction pipeline for FIB-SEM volumes (Scheffer et al. 2020; Xu et al. 2021) will be generated, tested, and released for Ubuntu Linux, MacOS, and Windows.

Maintenance

The packaging platform and the distribution channels must be maintained to fix bugs, to continuously release upgrades of our tools, and to track the evolution of the Java runtime, the packaging tool chain, and the underlying operating systems. The platform will be documented and opened to the open source community such that the burden of updates and maintenance can be shared.

References

Bogovic, J. A., P. Hanslovsky, A. Wong, and S. Saalfeld (2016). "Robust registration of calcium images by learned contrast synthesis". In: *ISBI*, pp. 1123–1126.

[12]OpenJDK 14: https://openjdk.java.net/projects/jdk/14/
[13]jpackage: https://openjdk.java.net/jeps/392
[14]OpenJDK 16: https://openjdk.java.net/projects/jdk/16/

Bogovic, J. A. et al. (2020). “An unbiased template of the Drosophila brain and ventral nerve cord”. In: *PLOS ONE* 15.12, e0236495.

Pietzsch, T., S. Preibisch, P. Tomančák, and S. Saalfeld (2012). “ImgLib2—generic image processing in Java”. In: *Bioinformatics* 28.22, pp. 3009–3011.

Scheffer, L. K. et al. (2020). “A connectome and analysis of the adult Drosophila central brain”. In: *eLife* 9. Ed. by E. Marder, e57443.

Schindelin, J. et al. (2012). “Fiji: an open-source platform for biological-image analysis”. In: *Nature Methods* 9.7, pp. 676–682.

Xu, C. S. et al. (2021). “An open-access volume electron microscopy atlas of whole cells and tissues”. In: *Nature* 599.7883, pp. 147–151.