

Automation of the nsls2forge conda infrastructure

Thomas Hopkins

Undergraduate, Rensselaer Polytechnic Institute, Troy, NY 12180

Maksim Rakitin

NSLS-II, Brookhaven National Laboratory, Upton, NY 11973

Abstract

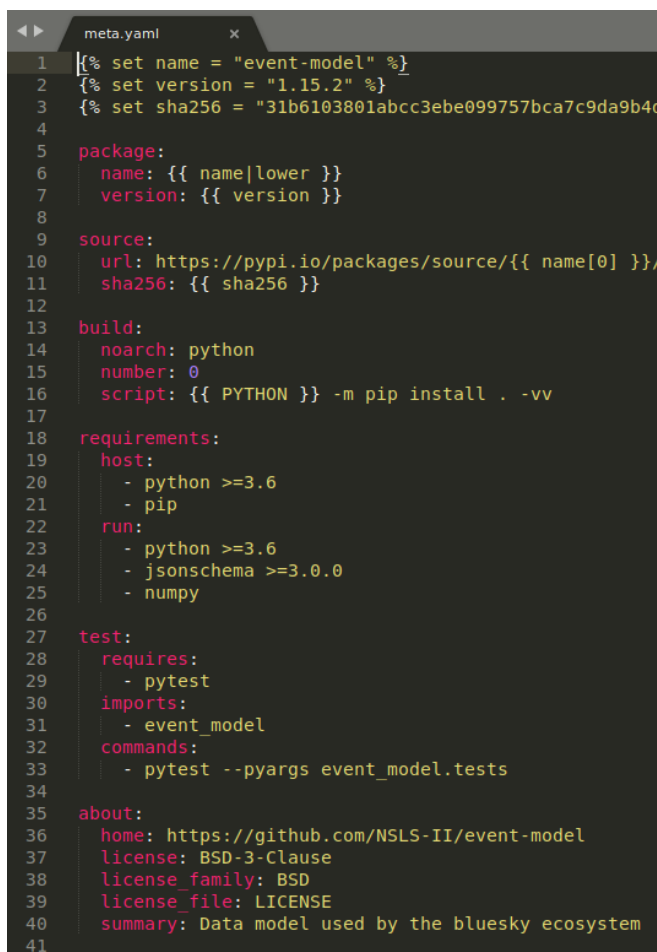
The nsls2forge conda infrastructure at the National Synchrotron Light Source II (NSLS-II) at Brookhaven National Laboratory requires large amounts of automation to be feasible in the long term. Through the use of emerging continuous integration technologies and a collection of software utilities, we have created a bot that will automatically search for and update software packages as they release. This makes newly created or updated software packages created at NSLS-II readily available to install at any beamline workstation, analysis server, or personal computer. The approach we took includes the development of a large dependency graph, the automation of file migrations, and the employment of cloud computing for easy access. A dependency graph is a directed acyclic graph that contains a node for each software package with links highlighting the relationship between packages. This is essential for maintaining a hierarchy of packages which is used by the bot for correct and efficient updating. Migrations are used to change the metadata of a specific package by updating its configuration files. We make use of Microsoft Azure Pipelines cloud services for hosting our bot that executes its job a few times a day. Conda is well known in the scientific Python community for providing software packages and environments. Existing open-source software, by the conda-forge team, provided many key utilities and ideas for the implementation of our bot. By automating the processes of our conda infrastructure at the NSLS-II, we are providing up-to-date, tested, and packaged software at a significantly better rate and volume than before.

I. Introduction

Software environments are the set of facilities, such as operating system, database, and other software tools, that is available to a program when it is being executed. Conda is an open-source, cross-platform, language-agnostic package manager and environment management system. NSLS-II uses Conda to deploy read-only production environments at the beamlines. A software package is an archive containing the Python library files, executables, and list of dependencies needed to run the package. Software packages that are developed and built at NSLS-II may be needed at many different beamlines and analysis servers. The Anaconda Cloud allows us to publish these software packages so we can share our software not only with others at NSLS-II but also around the world.

Most packages that are uploaded to the Anaconda Cloud have a separate “feedstock” repository hosted on the GitHub development platform. Feedstock repositories allow for automatic building and testing of new versions of software packages on many different platforms prior to being published. This is done using continuous integration/continuous deployment (CI/CD) technologies such as Microsoft Azure and TravisCI cloud services. The conda-forge team offers an open-source tool called conda-smithy that re-renders feedstock repositories to create these CI/CD builds for packages based on configuration files found in the repository. Re-rendering is a way to update the configuration files common to all feedstocks when changes to the repository have been made. Most of the time this refers to the CI/CD build configurations. We publish our packages from feedstock repositories to the *nsls2forge* conda channel on the Anaconda Cloud.

Managing feedstock repositories at the nsls2forge requires a lot of manual work. This includes manually editing configuration files (Figure 1), rendering the feedstock using conda-smithy, submitting pull requests, and monitoring build statuses. All these steps are required for each of the 200+ feedstock repositories belonging to nsls2forge.



```
1  {% set name = "event-model" %}
2  {% set version = "1.15.2" %}
3  {% set sha256 = "31b6103801abcc3ebe099757bca7c9da9b4d" %}
4
5  package:
6    name: {{ name|lower }}
7    version: {{ version }}
8
9  source:
10   url: https://pypi.io/packages/source/{{ name[0] }}/{{ name|lower }}/{{ name|lower }}-{{ version }}.tar.gz
11   sha256: {{ sha256 }}
12
13  build:
14   noarch: python
15   number: 0
16   script: {{ PYTHON }} -m pip install . -vv
17
18  requirements:
19   host:
20     - python >=3.6
21     - pip
22   run:
23     - python >=3.6
24     - jsonschema >=3.0.0
25     - numpy
26
27  test:
28   requires:
29     - pytest
30   imports:
31     - event_model
32   commands:
33     - pytest --pyargs event_model.tests
34
35  about:
36   home: https://github.com/NSLS-II/event-model
37   license: BSD-3-Clause
38   license_family: BSD
39   license_file: LICENSE
40   summary: Data model used by the bluesky ecosystem
41
```

Figure 1. The meta YAML configuration file for the event-model-feedstock repository. Notable aspects of this file are the package name, version, source URL, SHA256 hash, requirements, and build architecture. In this case, there is no architecture requirement. This file can be found here <https://github.com/nsls-ii-forge/event-model-feedstock/blob/master/recipe/meta.yaml>.

Software that we write at NSLS-II often depends on other software to build and function correctly. This can be seen most notably when importing different software modules while writing code. For example, writing some software in Python that imports functionality from the NumPy package means that that software depends on NumPy. Software can also require specific

versions of other software. This adds an interesting level of complexity to our problem of automation.

The purpose of this paper is to establish two possible methods of automation for managing software package feedstocks at NSLS-II. We also intend to provide other individuals and organizations that manage feedstock repositories the ability to use and extend these automated methods with ease by making them open-source and portable.

II. Methods

There are two main ways we chose to automate the nsls2forge conda infrastructure. The first is that we created an open-source library of utilities that allows the user to gather important information and make changes to items in feedstock repositories. These utilities can be used by any individual or organization that manages feedstock repositories. The second is a bot that automatically updates configuration files, re-renders, and submits pull requests for feedstocks at the nslls-ii-forge GitHub organization. The bot's configuration is also open-source and can be applied by other individuals or organizations that manage feedstocks with a small amount of modification. The method for the bot was first implemented by the conda-forge team but was not easily adaptable to our conda infrastructure.^{1,2}

We decided to create a suite of utilities that would not only help the end-user perform operations on feedstock repositories, but also be used by the bot to perform the entire process automatically. Other utilities, such as the dashboard (Figure 2) help maintainers see an overview of every feedstock's attributes. We include versions from nslls2forge, Python Package Index (PyPI), Anaconda defaults, conda-forge, and GitHub tags (if they are available) in the dashboard

for each feedstock. This provides an easy way to see how the versions from different sources compare to nsls2forge and if further actions are needed on our part. Table 1 highlights the four main utilities we built.

Name	Build Status	Versions	Downloads
bluesky	Azure Pipelines succeeded health 77%	nsls2forge v1.6.5 pypi v1.6.5 conda not found conda-forge v1.6.5 tag v1.6.5	downloads 4.6k
bluesky-darkframes	Azure Pipelines succeeded health 86%	nsls2forge v0.4.0 pypi v0.4.0 conda not found conda not found tag v0.4.0	downloads 68
bluesky-kafka	Azure Pipelines succeeded health 88%	nsls2forge v0.1.0 pypi v0.2.0 conda not found conda not found tag v0.2.0	downloads 3.8k
boltons	Azure Pipelines succeeded health 90%	nsls2forge v19.1.0 pypi v20.2.0 conda not found conda-forge v20.2.0 tag v20.2.0	downloads 5.3k
bumps	Azure Pipelines succeeded health 72%	nsls2forge v0.7.15 pypi v0.7.16 conda not found conda-forge v0.7.16 tag v0.7.16	downloads 543

Figure 2. Part of the dashboard for feedstock repositories at nsls2forge. Lists build statuses, code health, versions, and number of downloads for every feedstock. Full dashboard can be found here <https://github.com/nsls-ii-forge/project-management>.

Utility	Purpose
all-feedstocks	Performs operations on every feedstock belonging to a GitHub organization
dashboard	Creates a dashboard with build statuses, downloads, versions, and code health for every feedstock at nsls2forge
graph-utils	Performs operations on the dependency graph (build, query, and update)
meta-utils	Performs operations on individual feedstock configuration files

Table 1. Table of utilities built to automate operations at the nsls2forge. These are available via the command line interface. The code can be found here <https://github.com/nsls-ii-forge/nsls2forge-utils>.

When automatically updating software packages with new versions, a problem arises when considering the relationships between packages. To alleviate this problem, we implemented a directed acyclic graph (Figure 3) to represent every feedstock package at nsls2forge along with their dependencies. This method of representing the relationship between packages was developed and implemented by the conda-forge team for similar purposes.^{1,2} We will refer to this graph as the dependency graph.

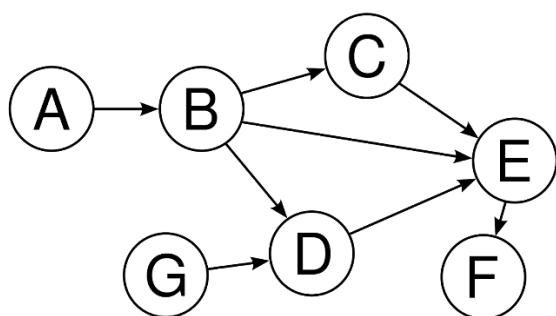


Figure 3. Directed Acyclic Graph (DAG). Useful for representing a complex set of relationships. Nodes A and G are at the top-level. One way to topologically sort this graph is the following ordering: A, G, B, C, D, E, F.

Each node in the graph can be represented by a piece of software. This definition can range from programming languages to software applications and software packages. We store each feedstock along with its raw configurations at each node in the graph. Edges between nodes

are defined as an “is a dependency of” relationship. For example, the package `event-model` imports some functionality from NumPy, therefore, NumPy is a dependency of `event-model`. A directed edge from the node corresponding to NumPy would connect to the node corresponding to `event-model`. A subgraph of the full dependency graph is shown in Figure 4, displaying only `event-model` and its related pieces of software. Sorting the nodes topologically gives us a correct order to update software packages so no dependency issues come up at build time. This is important since software can depend on specific versions of other software.

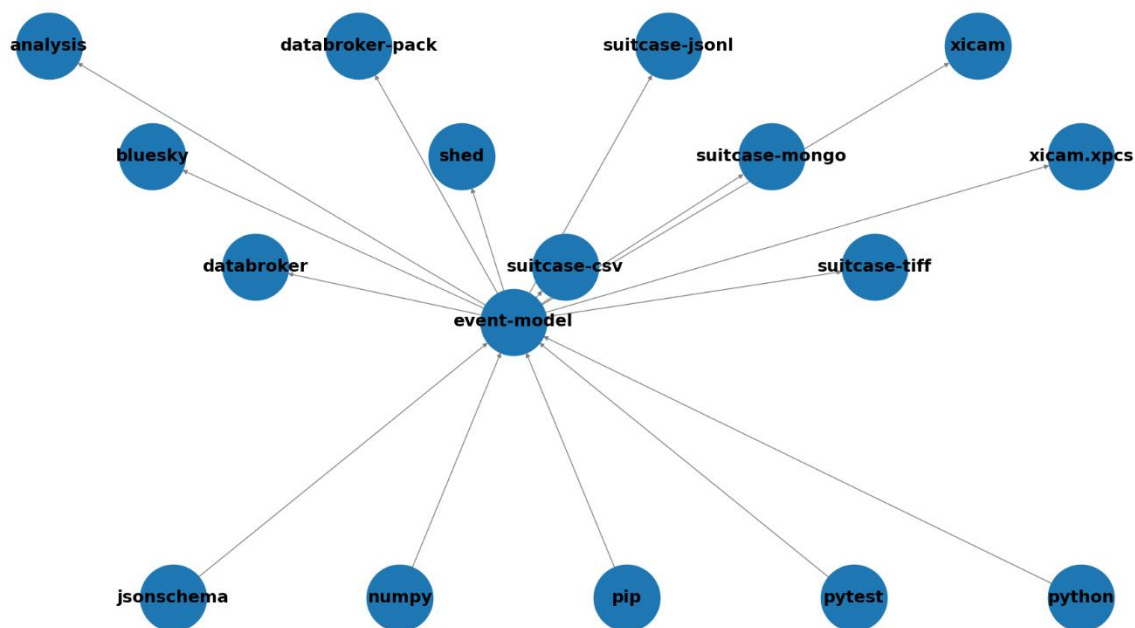


Figure 4. A generated subgraph of our much larger dependency graph. This subgraph shows only the software associated with the `event-model` Python package published at `nsls2forge`. Nodes (in blue) along the bottom are packages that `event-model` depends on to run/build. Nodes along the top are packages that require `event-model` to run/build. This structure for representing package dependencies was developed by the `conda-forge` team.^{1,2} The full graph is hosted at <https://github.com/nsls-ii-forge/auto-tick-graph>.

The bot itself will run in the cloud and use the various prebuilt utilities to automatically update software package feedstocks with new versions as they release. It will go through a cycle of retrieving all feedstock names from nsls2forge, creating/updating the dependency graph, retrieving new versions of packages directly from their sources, topologically sorting packages that require new versions, editing configuration files, and finally submitting pull requests to feedstock repositories on GitHub. This process will occur one to two times per day, submitting a maximum of ten pull requests to GitHub per run.

To retrieve new versions of packages from their sources, we make various API calls to the following sources: PyPI, CRAN, NPM, ROSDistro, GitHub, and any raw URL. PyPI and GitHub are our two most used sources. These API calls check for newest versions of packages that are available. If a new version is found that is greater than the current version of a package, we update the corresponding node in the graph with this new version. The bot will pick up all the nodes with new versions and update them in the correct order.

To edit configuration files with new versions, we use file migrations (Figure 5). These migrations were first developed by the conda-forge team for their bot.^{1,2} They allow us to keep a record of all changes that are made by the bot and update specific portions of configuration files. If anything goes wrong with automatically making changes to configuration files, they can still be manually edited within the same pull request and re-rendered.

1. Simple Additive Version Migrations



Figure 5. Simple version migrator used to update configuration YAML files. Original file on the left, migrator in the middle, and resulting file on the right. Tags are only updated if the version number is greater than the original version number.

III. Discussion

By automating the nsls2forge conda infrastructure, we are providing up-to-date software for NSLS-II beamlines and analysis servers at a significantly better rate and volume than before. Software feedstocks at nsls2forge are now receiving version update requests at a rate of 10 per run of the bot. Since the bot is set to run one to two times per day, it can send requests for up to 20 feedstocks that require version updates in one day. Maintainers at the nsls2forge will still be required to monitor build statuses and approve any changes prior to new versions being released on the Anaconda Cloud. The dashboard (Figure 2) provides a useful overview of these packages along with their build statuses and current versions. The bot adds a small portion of the dashboard to the text of each pull request for the convenience of the maintainers. This contains the package itself as well as its dependencies which require version updates first.

Some future improvements to the utilities and bot are to extend the functionality to update more than just version numbers of packages, retrieve more complex information from the dependency graph, perform file migrations across all feedstock repositories at once, and create an interactive version of the bot that responds to requests for changes in feedstock repositories. The current library of utilities can easily be extended with the additional functionality described above. Since these utilities are open-source, anyone can contribute to improving them. The scope of the interactive version of the bot is much larger and should be an entire project of its own.

IV. Conclusion

We have shown that the current process for maintaining feedstock packages at the `nsls2forge` is unfeasible in the long term. Automation of the `nsls2forge`'s operation helps alleviate most of the manual work in the maintenance process. The library of utilities helps not only the bot but also any user or maintainer work more efficiently in diagnosing issues and performing updates. The dependency graph is a useful representation for recognizing how packages relate to each other and for making decisions when updating packages. The bot provides an end-to-end tool that will update versions of packages automatically. While the changes made by the bot will still need to be reviewed manually, it improves the speed and volume at which packages can be updated and tested at NSLS-II.

V. Acknowledgements

I wish to thank my mentor, Dr. Maksim Rakitin, for his professionalism and generosity during the SULI program. Further, I wish to express my gratitude to the conda-forge and regro teams for providing many key utilities and ideas for the implementation of our bot. This project

was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).

VI. References

¹Wright, C. 2020. The Automation of Conda-Forge. AnacondaCON 2020.

<https://gateway.on24.com/wcc/eh/2336732/lp/2357492/the-automation-of-conda-forge/?campaign=web>

²regro/cf-scripts GitHub repository. Updated 2020. <https://github.com/regro/cf-scripts>