ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04BC004

IMPROVING CONTROL SYSTEM SOFTWARE DEPLOYMENT AT MAX IV

B. Bertrand[*], Á. Freitas, A. F. Joubert, MAX IV, Lund, Sweden
J.T. Kowalczyk, S2Innovation, Kraków, Poland

Abstract

The control systems of large research facilities like synchrotrons are composed of many different hardware and software parts. Deploying and maintaining such systems require proper workflows and tools. MAX IV has been using Ansible to manage and deploy its full control system, both software and infrastructure, for many years with great success. We detail further improvements: defining Tango devices as configuration, and automated deployment of specific packages when tagging Gitlab repos. We have now adopted conda as our primary packaging tool instead of the Red Hat Package Manager (RPM). This allows us to keep up with the rapidly changing Python ecosystem, while at the same time decoupling Operating System upgrades from the control system software. For better management, we have developed a Prometheus-based tool that reports on the installed versions of each package on each machine. This paper will describe our workflow and discuss the benefits and drawbacks of our approach.

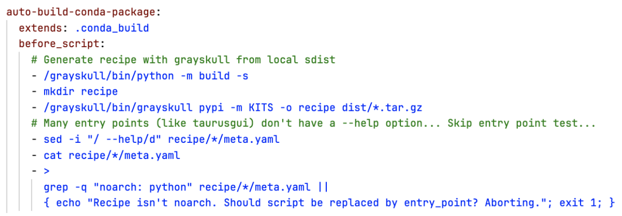
INTRODUCTION

The MAX IV synchrotron radiation facility in Lund, Sweden started user operations in 2016. It consists of a short-pulse facility and two storage rings with 16 beamlines. The control system has more than 500 virtual and physical machines to configure and maintain, including 24k Tango devices with 134k configurable properties. This requires automation, and the Software group has been using Ansible [1] to manage and deploy the full control system, both software and infrastructure, for 10 years [2]. We previously described [3] how we started moving away from RPM packages, tightly coupled to the Operating System version, to conda [4]. We now have deployments on many of our beamlines using solely conda. We discuss how the packages are created, and the pros and cons of our approach. Next, we describe how our Ansible deployment has changed to better support our common workflows. Finally, we report on a new monitoring tool we use to keep track of the actual state of the deployed code, and why it might differ from the Ansible configuration.

PACKAGE MANAGEMENT

Python

When investigating replacing RPM with conda, we started by creating a conda recipe in the source repository using a cookiecutter [5] template. Having to create a recipe in every repository made the adoption and transition quite slow. We developed a GitLab CI pipeline to automatically build conda packages using Grayskull [6]. Grayskull is an automatic conda recipe generator. It could create recipes for Python packages available on PyPI and from GitHub repositories. We added support for local source distribution [7]. Figure 1 details how the GitLab job creates a sdist package to then automatically generate the conda recipe.



```
auto-build-conda-package:
  extends: .conda_build
  before_script:
    # Generate recipe with grayskull from local sdist
    - /grayskull/bin/python -m build -s
    - mkdir recipe
    - /grayskull/bin/grayskull pypi -m KITS -o recipe dist/*.tar.gz
    # Many entry points (like taurusgui) don't have a --help option... Skip entry point test...
    - sed -i "/ --help/d" recipe/*/meta.yaml
    - cat recipe/*/meta.yaml
    - >
      grep -q "noarch: python" recipe/*/meta.yaml ||
      { echo "Recipe isn't noarch. Should script be replaced by entry_point? Aborting."; exit 1; }
```

Figure 1: auto-build-conda-package job.

If the recipe is not *noarch*, the job fails. For pure python package, this could be due to defining a script instead of an entry point, which should be fixed by the developer. For packages requiring compilation, a recipe has to be created manually. This is more for safety as grayskull is capable of generating such recipes. As we only have very few such repositories, this is not an issue. When an entry point is detected, grayskull automatically adds a test by running it with the `--help` flag. Unfortunately, applications based on taurus [8] do not support this flag. We remove this test but it would be better to keep it. The pipeline could be improved to keep it if taurus is not in the dependencies. This new pipeline allowed us to create conda packages very easily for all our internal repositories and made the transition from RPM to conda possible.

C++

We work mostly with Python but also have some C++ Tango device servers. Those had a dependency at build time on Makefiles from Pogo [9], the Tango code generator. To ease the compilation with conda, we migrated the build system to CMake [10]. This was a manual process but was not a huge task as we do not have that many C++ repositories and the *CMakeLists.txt* to create is quite similar between projects. Once a project can be compiled with CMake, and without any Pogo dependency, creating the conda recipe is quite straightforward and similar to how upstream projects like TangoDatabase and TangoTest are built.

Benefits

Moving from RPM to conda allowed us to separate the deployment from the operating system packaging and the system Python version. We could migrate from CentOS 7 to Rocky Linux 8 deploying exactly the same conda packages. Without this change, we would have had to rebuild all our

[*] benjamin.bertrand@maxiv.lu.se

RPMs for Rocky. We are now far less dependent on the OS and switching to a non-RPM based distribution is possible. Another advantage is that we have more freedom regarding all our dependencies and Python itself. We could move to Python 3.9 with conda when still on CentOS 7 while we were using 3.6 with RPM. We have been testing against 3.11 for some time in our CI pipelines and are in the process to switch the default version. Yet another advantage, is that developers can create conda environments on their development machines (running Linux, macOS, or Windows) and develop locally in an environment nearly identical to production. This speeds up development. Before people used docker containers, which was more cumbersome to setup and didn't really work for graphical applications.

For all modules available in a public repository, we try to submit a recipe to conda-forge [11]. There is great infrastructure in place making it easy to keep packages up to date. This benefits the whole community. For internal repositories specific to MAX IV, or for packages that cannot be redistributed due to the license (like Basler pylon Camera SDK), we use our own conda server based on Quetz [12].

Constraints

One remaining annoying issue is our pipeline execution time. Conda-build [13], or even *conda mambabuild*, when using boa [14] is quite slow. Our *auto-build-conda-package* CI job easily takes 5 minutes for a pure Python Tango Device Server, while the *build-pypi-package* job takes 30 seconds. To be fair, *python -m build* will happily create a package even if some of the defined dependencies do not exist. Another job is needed to check if the package can be installed and to test it. Conda-build will both create the package and run some tests after installing it in a new clean environment. So two jobs in one. But the slow part is not the testing one, it is the parsing of the recipe. The conda-build format has become quite complex over the years, requiring some recursive parsing and solving. The community is aware of the problem and some discussions are on-going to introduce a new format [15]. There is even a tool for this proposed format: rattler-build [16]. It is written in Rust, does not have any dependencies on conda-build or Python, and works as a standalone binary. First tests are quite impressive. Building a package with *rattler-build* took only 23 seconds running locally, while *conda mambabuild* took 3 minutes and 18 seconds. This includes the package creation, installation and testing in both cases. For simple recipes, using *boa convert* works well to create the required recipe from the file generated by grayskull. The tool is still in early development, so we will continue to do more testing. For our use case, it looks very promising.

It is worth mentioning wheel2conda [17], an experimental tool to convert pure Python wheels to conda packages. The idea was interesting but the project was abandoned in 2018 and needed development. We opted to stay with the officially supported tools. Since then, a new tool based on the same concept was created: python-wheel-to-conda-package [18]. We have not tested it yet. It may merit investigation as such a tool would be very fast but would require an extra step to test the resulting package.

DEPLOYMENT

Generic Conda Environment

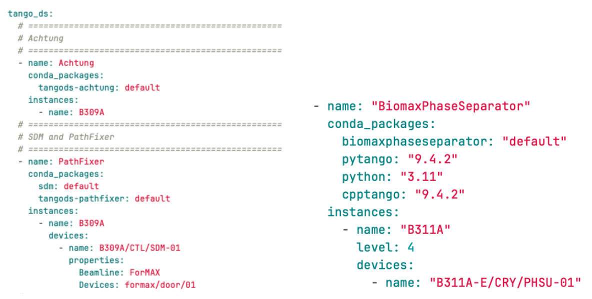
Two years ago, we detailed how we can create conda environments using the *conda_envs* variable in our inventory. The Ansible role was improved. It is now possible to define desktop menus for each environment. The creation of the environment was also optimized by using the *conda* module instead of *conda_env* if only conda packages are used (i.e. not pip dependencies), skipping the creation of an *environment.yml* file.

Tango Device Servers

Sardana used to be deployed using a specific role. We created a generic one to deploy Tango device servers and could replace the sardana role. The ans_maxiv_role_tango_ds role uses the *tango_ds* variable to define a list of Tango device servers. It relies on three different custom Ansible modules.

- The *tango_config* module registers and configures tango devices.
- The *conda* module deploys tango servers via conda.
- The *tango_starter* module ensures that the device servers are started or restarted (in case of change).

This role gives us an elegant way to define the Tango Device Servers to deploy using a nested list of servers, instances and devices, including the packages to install, as seen in Fig. 2a. The role ensures that an instance is restarted if the version changes. As we use mostly Python, the role will automatically add the default Python, cppTango and PyTango versions from the inventory in the environment of each server. This helps to keep the same version deployed everywhere. It is possible to specify a different version to test only in one environment before deploying globally (Fig. 2b).



```
tango_ds:
  # ==================================================
  # Achtung
  # ==================================================
  - name: Achtung
    conda_packages:
      tangods-achtung: default
    instances:
      - name: B309A
  # ==================================================
  # SDM and PathFixer
  # ==================================================
  - name: PathFixer
    conda_packages:
      sdm: default
      tangods-pathfixer: default
    instances:
      - name: B309A
        devices:
          - name: B309A/CTL/SDM-01
            properties:
              Beamline: ForMAX
              Devices: formax/door/01
```

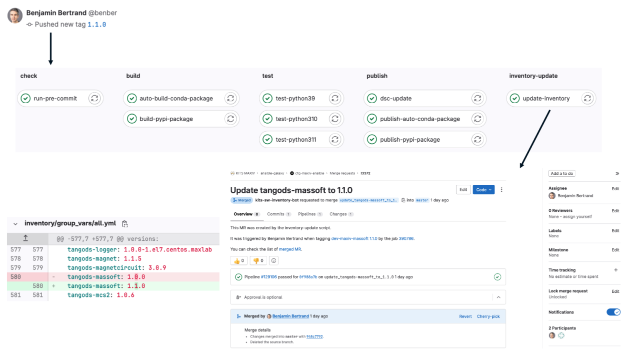
```
- name: "BiomaxPhaseSeparator"
  conda_packages:
    biomaxphaseseparator: "default"
    pytango: "9.4.2"
    python: "3.11"
    cpptango: "9.4.2"
  instances:
    - name: "B311A"
      level: 4
      devices:
        - name: "B311A-E/CRY/PHSU-01"
```

(a) Standard definition. (b) Overwrite default versions.

Figure 2: Tango Device Servers definition

Continuous Deployment

The Ansible inventory defines the location where each package should be deployed, as well as the version to be used. The default version for each package is kept in a *versions* dictionary in the Ansible *all* group. When tagging a repository, a GitLab CI job will automatically create a merge request in the inventory to update the default version of that package (Fig 3). The developer who tagged is assigned to that merge request. He can merge it himself or approve it and set a milestone so it is merged and deployed later.



inventory/group_vars/all.yml

```
@@ -577,7 +577,7 @@ versions:
    tangods-logger: 1.0.0-1.el7.centos.maxlab
    tangods-magnet: 1.1.5
    tangods-magnetcircuit: 3.0.9
-   tangods-massoft: 1.0.0
+   tangods-massoft: 1.1.0
    tangods-mcs2: 1.0.6
```

Update tangods-massoft to 1.1.0

kits-sw-inventory-bot requested to merge update_tangods-massoft_to_1… into master 1 day ago

This MR was created by the inventory-update script.

It was triggered by Benjamin Bertrand when tagging dev-maxiv-massoft 1.1.0 by the job 390786.

You can check the list of merged MR.

Pipeline #129106 passed for 0f988a7b on update_tangods-massoft_to_1.1.0 1 day ago

Merged by Benjamin Bertrand 1 day ago

Merge details
- Changes merged into master with 948c7792.
- Deleted the source branch.

Figure 3: Automatic inventory update.

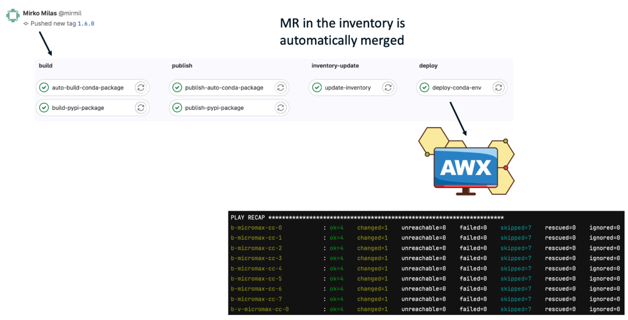
It's even possible to setup continuous deployment by setting extra variables in the *.gitlab-ci.yml* file to specify the conda environment and hosts to update (Fig 4).

```
include:
  - project: kits-maxiv/cfg-maxiv-gitlabci
    file: /.python-ci.yml

variables:
  DEPLOY_CONDA_ENV_NAME: "ctmicromaxsynoptic"
  DEPLOY_HOSTS: "micromax-cc"
```

Figure 4: GitLab CI file for continuous deployment.

In that case, the inventory update MR is merged automatically and an extra CI job is run to trigger the Ansible deployment by using AWX [19] API as shown in Fig 5.



```
PLAY RECAP *********************************************************************
b-micromax-cc-0            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-1            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-2            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-3            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-4            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-5            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-6            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-micromax-cc-7            : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
b-v-micromax-cc-0          : ok=4    changed=1    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0
```

Figure 5: Continuous deployment pipeline.

This is very useful for applications maintained by the beamline staff themselves, like the synoptic for example. It gives them the possibility to deploy their software using our Ansible workflow, just by tagging their repository.

Monday Deployment

The continuous deployment can't be generalized to most applications as we don't want to change software during operations. It is thus crucial to deploy regularly to ensure that the Ansible inventory matches what is really installed on all machines. A deployment crew is in charge to merge approved merge requests with the proper milestone and run the Ansible deploy playbook on all beamlines every Monday (a maintenance day). This is to avoid any divergence and to make sure a bug fix is deployed everywhere. This process also helped us to keep the inventory clean and gave us better confidence in the deployment system, avoiding developers being afraid of breaking something by running Ansible.

MONITORING

The Ansible inventory is very valuable but does not give us the full picture. Some software could have been installed manually, or using Ansible but removed from the inventory and still be present on a machine.

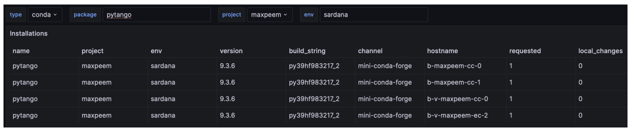
Prometheus

To keep better track of what is deployed at any time, we developed a script to collect packages data from yum and conda. We use Prometheus [20] to monitor Linux hosts via the standard node_exporter and even have a tango_exporter to gather Tango specific information. The packages_export script is run by the crontab every 30 minutes and creates a *packages.prom* file inside the directory monitored by the node_exporter textfile collector. Figure 6 shows the information saved for both a conda package and RPM.

```
package_conda{env="sardana",name="pytango",platform="linux-64",
version="9.3.6",build_string="py39h609f8c2_0",channel="mini-conda-forge",
requested="1",local_changes="0"} 1
package_rpm{name="tango-java",platform="x86_64",version="9.3.5-24.el8.maxlab",
source="maxiv-public",local_changes="0" } 1
```

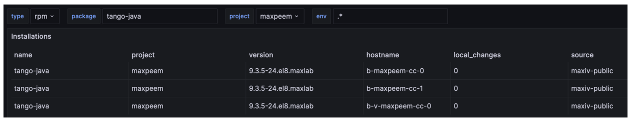
Figure 6: packages.prom file.

We store data for all RPMs coming from our internal repository and for all conda packages installed in a global environment by Ansible. Figures 7 and 8 show how we can search for any package (conda or RPM) via a grafana dashboard.



name	project	env	version	build_string	channel	hostname	requested	local_changes
pytango	maxpeem	sardana	9.3.6	py39hf983217_2	mini-conda-forge	b-maxpeem-cc-0	1	0
pytango	maxpeem	sardana	9.3.6	py39hf983217_2	mini-conda-forge	b-maxpeem-cc-1	1	0
pytango	maxpeem	sardana	9.3.6	py39hf983217_2	mini-conda-forge	b-v-maxpeem-cc-0	1	0
pytango	maxpeem	sardana	9.3.6	py39hf983217_2	mini-conda-forge	b-v-maxpeem-ec-2	1	0

Figure 7: Conda package dashboard.



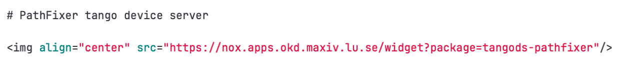
name	project	version	hostname	local_changes	source
tango-java	maxpeem	9.3.5-24.el8.maxlab	b-maxpeem-cc-0	0	maxiv-public
tango-java	maxpeem	9.3.5-24.el8.maxlab	b-maxpeem-cc-1	0	maxiv-public
tango-java	maxpeem	9.3.5-24.el8.maxlab	b-v-maxpeem-cc-0	0	maxiv-public

Figure 8: RPM package dashboard.

Sorting per version is very useful to find if a package is outdated on some hosts. It is also possible to see if a package was locally modified. For debugging, the recommended way is to create a new environment and install the package in editable mode. But avoiding developers modifying code directly in an installed package for quick tests has proven difficult. At least, we now have a way to detect it. We have been thinking about sending notifications about altered packages. This check was implemented by comparing expected and computed sha256 checksums on files. Note that, since conda 23.7.0, the *conda doctor* command implements this health check.

Nox

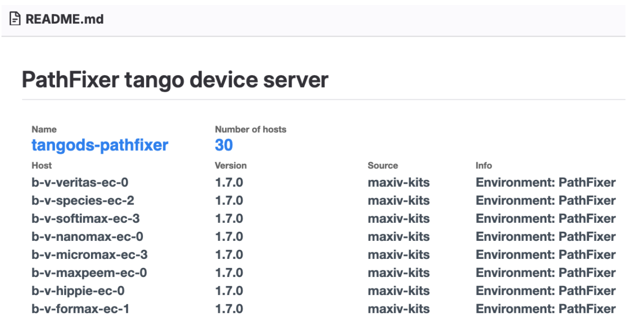
When working on a repository, knowing where and which version of a package is deployed is quite useful. The information can be found in the Ansible inventory or using the previously grafana dashboard, but requires context switching. We developed a small web application, named *Nox* to create widgets that can be integrated in a repository README file. Nox uses the Prometheus information stored in a VictoriaMetrics database. Figure 9 shows how to add a widget to a README in markdown with just one line using the *img* tag. Only the package name needs to be passed to the url. This is rendered as in Fig. 10.



```
# PathFixer tango device server

<img align="center" src="https://nox.apps.okd.maxiv.lu.se/widget?package=tangods-pathfixer"/>
```

Figure 9: Nox widget definition.



README.md

PathFixer tango device server

Name	Number of hosts		
tangods-pathfixer	30		
Host	**Version**	**Source**	**Info**
b-v-veritas-ec-0	1.7.0	maxiv-kits	Environment: PathFixer
b-v-species-ec-2	1.7.0	maxiv-kits	Environment: PathFixer
b-v-softimax-ec-3	1.7.0	maxiv-kits	Environment: PathFixer
b-v-nanomax-ec-0	1.7.0	maxiv-kits	Environment: PathFixer
b-v-micromax-ec-3	1.7.0	maxiv-kits	Environment: PathFixer
b-v-maxpeem-ec-0	1.7.0	maxiv-kits	Environment: PathFixer
b-v-hippie-ec-0	1.7.0	maxiv-kits	Environment: PathFixer
b-v-formax-ec-1	1.7.0	maxiv-kits	Environment: PathFixer

Figure 10: Nox widget rendered.

We added this line to most repositories README in our GitLab instance, making this dynamic information easily available. This is now part of our cookiecutter template when creating a new project.

CONCLUSION

We managed to automatically create conda packages in our GitLab CI pipeline, without having to write a recipe, and are confident we can improve its speed in the near future. Conda allows us to deploy C++ and Python software without being tied to the Operating System, which helps production deployment as well as developers working on their local machines. Contributing to the public conda-forge channel has benefited the Tango Controls and scientific software community. In the last two years, we also improved our Ansible workflow with an easier way to define Tango Device Servers in our inventory. This ensures the necessary applications, and only those, are automatically restarted during deployment. We adopted a regular deployment process to improve our confidence in the state of the deployed software, and our ability to deploy to clean machines. On the monitoring side, we developed tooling to provide us with an accurate picture of what is deployed where. By integrating it with the source repositories, developers can see at a glance which beamlines are using a package and may be affected by planned changes. Ansible has served us well — we expect this to continue, and that our processes will continue to improve.

REFERENCES

[1] Ansible documentation, `https://docs.ansible.com`

[2] V. H. Hardion *et al.*, "Configuration Management of the Control System", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper THPPC013, pp. 1114–1117.

[3] B. Bertrand, Á. Freitas, and V. Hardion, "Control System Management and Deployment at MAX IV", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 819–823. `doi:10.18429/JACoW-ICALEPCS2021-THBL01`

[4] Conda documentation, `https://docs.conda.io`

[5] Cookiecutter, `https://cookiecutter.readthedocs.io`

[6] Grayskull, `https://github.com/conda/grayskull`

[7] Grayskull PR 282, `https://github.com/conda/grayskull/pull/282`

[8] Taurus Project, `https://taurus-scada.org`

[9] Pogo, `https://gitlab.com/tango-controls/pogo`

[10] CMake, `https://cmake.org`

[11] Conda-Forge Community, "The conda-forge Project: Community-based Software Distribution Built on the conda Package Format and Ecosystem", *Zenodo*, 2015. `doi:10.5281/zenodo.4774216`

[12] Quetz, `https://quetz.readthedocs.io`

[13] Conda-build, `https://docs.conda.io/projects/conda-build`

[14] Boa, `https://boa-build.readthedocs.io`

[15] A new YAML based format for "conda-build" files, `https://github.com/conda-incubator/ceps/pull/54`

[16] rattler-build, `https://github.com/prefix-dev/rattler-build`

[17] wheel2conda, `https://github.com/takluyver/wheel2conda`

[18] python-wheel-to-conda-package, `https://github.com/tibdex/python-wheel-to-conda-package`

[19] Ansible AWX, `https://github.com/ansible/awx`

[20] Prometheus, `https://prometheus.io`