ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THBL01

# CONTROL SYSTEM MANAGEMENT AND DEPLOYMENT AT MAX IV

B. Bertrand*, Á. Freitas, V. Hardion, MAX IV, Lund, Sweden

*Abstract*

The control systems of big research facilities like synchrotron 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 quite some time with great success. All required software (i.e. tango devices, GUIs...) used to be packaged as RPMs (Red Hat Package Manager) making deployment and dependencies management easy. Using RPMs brings many advantages (big community, well tested packages, stability) but also comes with a few drawbacks, mainly the dependency to the release cycle of the Operating System. The Python ecosystem is changing quickly and using recent modules can become challenging with RPMs. We have been investigating conda as an alternative package manager. Conda is a popular open-source package, dependency and environment management system. This paper will describe our workflow and experience working with both package managers.

## INTRODUCTION

The Controls & IT group, also called KITS, is responsible for the whole IT infrastructure at MAX IV. This includes everything from control system hardware and software to data storage, high performance computing, scientific software and information management systems. Within KITS, the Control System Software team manages all the software linked to the control system. With the accelerator and 16 beamlines, this represents more than 330 physical and virtual machines to configure and maintain. Ansible [1] was chosen for its simplicity of use as detailed in CONFIGURATION MANAGEMENT OF THE CONTROL SYSTEM [2] and is a great help to achieve this. The control system is made of many components that often have dependencies with each other: tango devices, controllers, GUIs. Building and being able to deploy each software individually without breaking another part is not straightforward. This requires some tools and is exactly why package managers were designed. One of their role is to keep track of dependencies between packages to ensure coherence and avoid conflicts. Using a package manager makes it easier to distribute, manage and update software.

## PACKAGE MANAGEMENT

### *RPM*

The RPM Package Manager [3] (RPM) is the package management system that runs on Red Hat Enterprise Linux, CentOS, and Fedora. As CentOS is the default Operating System at MAX IV, using RPM to distribute internal software was an obvious choice.

RPM gives us access to a large numbers of high quality packages from the main CentOS repository and others like EPEL [4], the Extra Packages for Enterprise Linux. This provides solid foundation to build on and is one huge advantage of Operating System package managers.

**SPEC file** RPM creation is usually based on a SPEC file [5]. It is the recipe that rpmbuild uses to build an RPM. It contains metadata like the name of the package, version, license, as well as the instructions to build the software from source with all the required dependencies as seen in Fig. 1.

```
Summary:    Tango device for Linkam T96 heater
Name:       tangods-linkamt96
Version:    1.2.0
Release:    1%{?dist}.maxlab
License:    GPL
URL:        http://www.maxiv.lu.se
Source:     %{name}-%{version}.tar.gz
Requires:   lib-maxiv-linkam-t96
Requires:   linkam-sdk
Requires:   lib-maxiv-common-cpp >= 4.0.0
Requires:   libtango9
BuildRequires: lib-maxiv-linkam-t96-devel
BuildRequires: linkam-sdk-devel
BuildRequires: lib-maxiv-common-cpp-devel >= 4.0.0
BuildRequires: libtango9-devel
# for pogo Makefile templates:
BuildRequires: tango-java

%description
Tango device for Linkam T96 heater

%prep
%setup -q

%build
make

%install
[ -z %{buildroot} ] || rm -rf %{buildroot}

# install bins
pushd bin > /dev/null
for f in *; do
    install -D -m755 $f %{buildroot}%{_bindir}/$f
done
popd > /dev/null

%files
%defattr (-,root,root,755)
%{_bindir}/*
```

Figure 1: RPM SPEC file (extract).

C++ projects are packaged using a SPEC file. RPM creation is handled by a GitLab CI [6] pipeline using maxpkg,

* benjamin.bertrand@maxiv.lu.se

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THBL01


a MAX IV plugin to rpkg [7] for managing RPM packaging from a git repository.

**fpm** Most software at MAX IV are written in Python. *setup.py*, even if not recommended anymore [8], was the historical way to package a Python library or application with *setuptools* and is still widely used today. fpm [9] is a command-line program designed to help build packages. It can take different source types (directory, rubygem, python package...) and convert them to a target type, most common being "rpm" and "deb". Using fpm avoids having to write a SPEC file to create RPM. fpm can take the metadata required to build a package from the *setup.py* file.

```
fpm -s python -t rpm \
  --python-bin python3.6 \
  --rpm-dist el7 \
  -p results \
  --python-package-name-prefix python36 \
  --python-setup-py-arguments=--prefix=/usr \
  --no-python-downcase-dependencies \
  ${FPM_FLAGS} \
  --url $CI_PROJECT_URL --verbose .
```

Figure 2: Example of `fpm` command.

Figure 2 shows a typical command to create an RPM from a Python repository. It is automatically run by our GitLab CI pipeline. fpm takes the dependencies defined by *install_requires* in the *setup.py* file and automatically adds the *python36* prefix to match the RPM naming convention. If a Python application had the following *install_requires=["taurus", "PyYAML", "pytango"]*, fpm would create a RPM with the dependencies python36-taurus, python36-PyYAML and python36-pytango. If there is a mismatch between the Python package and RPM name, it is possible to pass extra arguments using the *FPM_FLAGS* variable and overwrite the RPM dependencies, i.e. *FPM_FLAGS: '–no-python-dependencies -d python36-numpy,python36-dateutil'* could be used for a package depending on the *python-dateutil* library.

With the CentOS Project decision to shift its focus from CentOS Linux to CentOS Stream [10] and the uncertainty this created, migration to CentOS 8 was stopped. This forced us to remain on CentOS 7 and that started to create issues. CentOS is known for its stability, which is very important, but can become a problem when recent software are needed. Sardana [11] version 3 was very difficult to install as RPM due to the versions of the dependencies required. This is one of the reason we started to look at alternatives package managers like conda [12].

## *Conda*

Conda is an open-source package, dependency and environment management for any language. It is cross-platform and runs on Windows, macOS and Linux. By using anaconda compilers, binaries created with conda-build can run on any modern Linux distribution. Being OS independent is an interesting feature that makes changing OS or migrating to a new one easier. With RPM, even when upgrading between two major releases of the same distribution, a rebuild of all packages is required. It's not the case with conda. The same package can be installed on CentOS 7 and 8 or even Debian.

Anaconda [13], the company behind conda provides some default channels with a large amount of packages. In the past years, conda-forge [14] became the de facto channel when using conda. Conda-forge is a community-led collection of recipes, build infrastructure and distributions for the conda package manager. It allows developers to automatically build recipe in a clean and repeatable way on Windows, Linux and macOS.

In 2021, the Tango community started to publish tango packages to conda-forge as mentioned in the THE TANGO CONTROLS COLLABORATION STATUS [15]. MAX IV also made available some packages that could be useful to the community like dsconfig or svgsynoptic2. Creating a recipe for conda-forge isn't very difficult but there is a review process done by volunteers that can take some time. For internal software we need a faster way to release packages and we deployed our own internal conda server based on Quetz [16].

Quetz is an open-source conda packages server. We use it to proxy external channels, like conda-forge, allowing conda packages to be installed without internet access. We also have some local channels to store packages created internally.

One complaint that people have about conda is that it can be slow. This is true when using large channels, and we even noticed a quite high memory usage (Fig. 3).

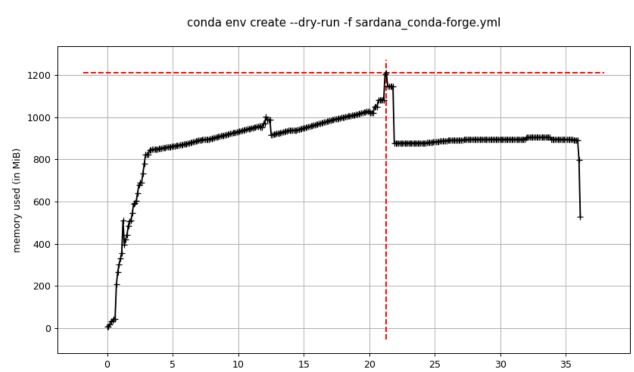


Figure 3: Sardana env creation with conda-forge.

The list of packages available in a channel is stored in a *repodata.json* file. It includes the description of all packages and their dependencies. That file needs to be downloaded and parsed to resolve an environment. The bigger that file, the more work for conda. As packages are never removed from a channel like conda-forge, the repodata.json file keeps growing. It is about 141MB today for the conda-forge linux-64 channel.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THBL01

One way to improve performances is mamba [17], a re-implementation of conda in C++ for maximum efficiency. It was conceived as a drop-in replacement for conda, using libsolv for much faster dependency solving. Mamba is robust and fast (Fig. 4) but not 100% compatible with conda yet, especially for conda-env commands, meaning we couldn't rely on it for all operations.

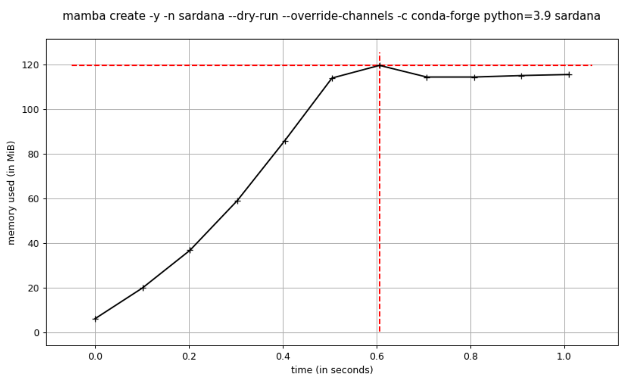


Figure 4: Sardana env creation with mamba.

As conda performances depend on the channel size, we created our own local *mini-conda-forge* channel, a subset of conda-forge. Having a channel with only the packages we are interested in gives a big boost in performances, both in time and memory usage (Fig. 5). Using that channel, instead of conda-forge, solving an environment with sardana decreases the time from over 35 seconds to 1.4 seconds and memory usage from 1.2GB to only 45MB! The result is almost identical to using mamba. Note that those figures don't take into account the download of the repodata file, that was already cached, nor the download and installation of the packages.

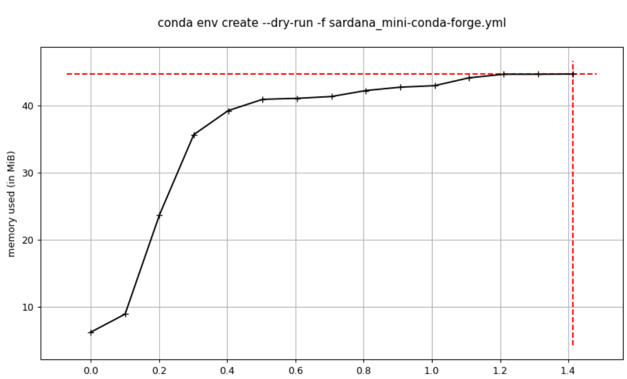


Figure 5: Sardana env creation with mini-conda-forge.

To keep this channel up-to-date automatically, a GitLab CI pipeline is run on schedule every night. It downloads new packages, and their dependencies, from conda-forge and uploads them to mini-conda-forge. The packages to download are based on a list of environments we want to be able to install, that is defined in a text file (Fig. 6). Making new packages available only requires to update this packages specs file.

```
# We want to be able to install conda and mamba
conda
mamba
# Default packages for a sardana environment
python=3.9 sardana bitshuffle matplotlib spyder pytango taurus taurus_pyqtgraph h5py hdf5plugin icepap sockio
# lavue
python=3.9 lavue pytango h5py bitshuffle hdf5plugin
# silx
python=3.9 silx h5py bitshuffle hdf5plugin
# pymca
python=3.9 pymca matplotlib pyqt
```

Figure 6: Example of mini-conda-forge environments list.

Building a conda package requires a recipe, which is defined in a *meta.yaml* file, with the information needed to create the package. This is equivalent to the SPEC file for RPM. Figure 7 is a typical example of a pure Python package on conda-forge.

```
{% set name = "dsconfig" %}
{% set version = "1.6.0" %}

package:
  name: {{ name|lower }}
  version: {{ version }}

source:
  url: https://pypi.io/packages/source/{{ name[0] }}/{{ name }}/dsconfig-{{ version }}.tar.gz
  sha256: e7709b0fa920b9a41460c0613ea2925aed089b8852cc8fa10ed2c3056ef82b56

build:
  number: 0
  noarch: python
  entry_points:
    - xls2json = dsconfig.excel:main
    - json2tango = dsconfig.json2tango:main
  script: {{ PYTHON }} -m pip install . -vv

requirements:
  host:
    - pip
    - python >=3.6
  run:
    - jsonpatch >=1.13
    - jsonschema
    - pytango
    - python >=3.6
    - six
    - xlrd

test:
  imports:
    - dsconfig
    - dsconfig.appending_dict
  commands:
    - pip check
    - xls2json --help
    - json2tango --help
  requires:
    - pip

about:
  home: https://gitlab.com/MaxIV/lib-maxiv-dsconfig
  summary: Library and utilities for Tango device configuration.
  license: GPL-3.0-or-later
  license_file: LICENSE.txt
  description: |
    This is a command line tool for managing configuration of Tango device servers.
    Tango-Controls is a software toolkit for building control systems.
  dev_url: https://gitlab.com/MaxIV/lib-maxiv-dsconfig

extra:
  recipe-maintainers:
    - beenje
```

Figure 7: `conda-forge` recipe.

To package software we develop internally we decided it was easier to make the conda recipe part of the source repository. This removes the need to have a second repository to maintain. Figure 8 shows an example.

conda-build provides a macro to parse the *setup.py* file. It can be used to get the version from that file as well as the runtime dependencies, making it easier to write and maintain the recipe.

Our GitLab CI pipeline template will automatically build and upload the conda package to our Quetz server if it detects a *recipe/meta.yaml* file in a repository.

## ANSIBLE

Ansible was chosen early at MAX IV as the solution to deploy and manage the control system.

### *RPM Deployment*

Ansible has a builtin *yum* module to manage packages with the *yum* package manager. This makes it easy to deploy RPM. For ease of use and consistency, a generic playbook was created to deploy internal software packages with RPM. The *packages_stable* and *packages_testing* variables define the list of packages to be deployed. Those variables are maintained in the Ansible inventory in the proper group or host variables file. The default version for each package is

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THBL01



```
{% set data = load_setup_py_data(setup_file="../setup.py",
  from_recipe_dir=True) %}

package:
  name: tango_exporter
  version: {{ data.get('version') }}

source:
  path: ..

build:
  number: 0
  noarch: python
  script: {{ PYTHON }} -m pip install . -vv
  entry_points:
    - tango_exporter = tango_exporter:main

requirements:
  host:
    - pip
    - python >=3.6
  run:
    - python >=3.6
{% for dep in data['install_requires'] %}
    - {{ dep.lower() }}
{% endfor %}

test:
  imports:
    - tango_exporter
  requires:
    - pip
  commands:
    - pip check
    - tango_exporter --help

about:
  home: https://gitlab.maxiv.lu.se/kits-maxiv/app-maxiv-tangoexporter
  license: GPL-3.0-or-later
  license_file: ../LICENSE.txt
  summary: Prometheus exporter for a Tango control system.
```

Figure 8: `local` recipe.

centralized in the *all* group for consistency (Fig. 9a). Developers are encouraged to use this default version but can also pin it if needed or even use latest (Fig. 9b).

```
versions:
  python-taurus: 4.5.1-7.el7
  python-dsconfig: 1.4.0-1.el7
  python-facadedevice: 1.0.3.dev1-1.el7
  python-fandango: 14.3.0-1.el7
  python36-sdm: 1.6.1
  tangods-pathfixer: 1.5.7
  python-pyicepap: 2.8.1-1.el7
  python36-sherlock: 1.0.4
```

(a) `versions` definition.

```
packages_stable:
  python-taurus: default
  python-dsconfig: default
  python-facadedevice: 0.9.0
  python-fandango: 10.9
  python36-sdm: default
  tangods-pathfixer: default
  python-pyicepap: default
  python36-sherlock: latest
```

(b) `packages` definition.

Figure 9: RPM packages - Ansible inventory.

### *Conda Deployment*

Ansible doesn't have any builtin module to interact with conda. MAX IV has its own modules (Fig. 10) based on the ones developed by ESS [18].

- The *conda* module can install, update or remove conda packages. It works with a list of conda packages.
- The *conda_env* module manages environment using an *environment.yml* file.

Mamba is used by default by the *conda* module but it's currently not compatible with the *conda_env* one.

```
- name: install flask 2.0 and Python 3.9
  conda:
    name:
      - python=3.9
      - flask=2.0
    state: present
    environment: myapp
```

(a) `conda` module.

```
- name: create myenv environment
  conda_env:
    name: myenv
    state: present
    file: /tmp/environment.yml
```

(b) `conda_env` module.

Figure 10: Ansible modules

Our ans_maxiv_role_conda Ansible role installs and configures both conda and mamba. It can also be used to create a list of conda environments by setting the *conda_envs* variable in the inventory. Conda environments are isolated by nature. You usually have to activate one to use it. To be transparent for the users, the role can create wrappers for command line applications. The wrapper is a simple script that activates the environment and runs the command from the env. Wrappers are deployed under `/usr/local/bin`. Users don't have to know that the application they run is installed in a conda environment. Figs. 11 and 12 show how to define such an environment and the resulting wrapper.

```
conda_envs:
  - env_name: silx
    dependencies:
      - python=3.9
      - silx=0.15.1
    wrappers:
      - silx
```

Figure 11: `conda_envs` definition.

```
#!/bin/bash

# Some cli need the env to be activated
source /opt/conda/etc/profile.d/conda.sh
conda activate silx

/opt/conda/envs/silx/bin/silx "$@"
```

Figure 12: `/usr/local/bin/silx` wrapper.

Conda is also used to deploy Sardana version 3. A specific Ansible role and playbook were developed to deploy it based on an environment.yml file. This format allows to define both conda and Python packages (installed with pip). The yaml file is created from a template defined in the role and can be customized, to add extra packages, based on different variables in the inventory. Figure 13 shows an example of such a resulting environment. Wrappers are also installed under `/usr/local/bin` for all sardana commands to make them globally available.

## CONCLUSION

Using a package manager to build, distribute and update software is a requirement in modern software development,

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-THBL01

```
channels: [maxiv-kits, mini-conda-forge]
dependencies:
- pytango=9.3.3
- taurus=4.7.1
- taurus_pyqtgraph
- h5py
- hdf5plugin
- icepap=3.6.2
- sockio=0.15.0
- taurusgui-maxpeemenergy=0.2.2
- python=3.9
- sardana=3.1.2
- bitshuffle
- matplotlib
- spyder
- pytest
- pip
- pip:
    - --trusted-host repo.maxiv.lu.se
    - '-i http://repo.maxiv.lu.se/devpi/maxiv/prod'
    - taurusgui-scangui3==1.1.3
    - taurusgui-quickgui3==2.1.3
name: sardana
```

Figure 13: Sardana environment.

as git is for version control. There are different kinds of package managers that all have their strengths and weaknesses. RPM is by nature integrated with the OS. It's very stable and can be used to create systemd services for example. Conda is OS independent and creates isolated environments, avoiding the risk of dependencies conflicts, and giving more freedom in the packages that can be installed. Ansible, combined with RPM or conda, gives us a reliable and reproducible way to deploy and maintain the control system.

## REFERENCES

[1] Ansible, https://docs.ansible.com/ansible/latest/index.html

[2] V. Hardion *et al.*, "Configuration Management of the Control System", in *Proc. ICALPECS'13*, San Francisco, CA, USA, 2013, paper THPPC013.

[3] RPM Package Manager, https://rpm.org

[4] EPEL, https://docs.fedoraproject.org/en-US/epel/

[5] SPEC file, https://rpm-packaging-guide.github.io/#what-is-a-spec-file

[6] GitLab CI/CD, https://docs.gitlab.com/ee/ci/

[7] rpkg, https://docs.pagure.org/rpkg/index.html

[8] Packaging Python Projects, https://packaging.python.org/tutorials/packaging-projects/#configuring-metadata

[9] fpm, https://fpm.readthedocs.io

[10] CentOS Stream, https://blog.centos.org/2020/12/future-is-centos-stream/

[11] Sardana, https://sardana-controls.org

[12] Conda, https://conda.io

[13] Anaconda, https://www.anaconda.com

[14] 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

[15] A. Götz *et al.*, "The Tango Controls Collaboration Status in 2021", presented at ICALPECS 2021, Shanghai, China, 2021, paper WEAR01, this conference.

[16] Quetz, https://quetz.readthedocs.io

[17] Mamba, https://mamba.readthedocs.io

[18] ESS conda Ansible modules, https://gitlab.esss.lu.se/ics-ansible-galaxy/ics-ans-role-conda