ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP1

INAU: A CUSTOM BUILD-AND-DEPLOY TOOL BASED ON Git

L. Pivetta*, A.I. Bogani†, Elettra Sincrotrone Trieste, Basovizza (TS), Italy

Abstract

Elettra Sincrotrone Trieste is currently operating two light sources, Elettra, a third generation synchrotron, and FERMI, a free electron laser. Control systems are based on a number of diverse systems, such as VME-based front-end computers, small embedded systems, high performance rack-mount servers and control room workstations. Custom device drivers and hard real-time applications have been developed during the years, exploiting the technologies adopted such as RTAI and Adeos/Xenomai, which make a massive update demanding. Modern CI/CD tools are then not available for legacy platforms, and a custom tool, integrating git and a database back-end to build and deploy software components based on release tags has been developed.

INTRODUCTION

At Elettra and FERMI some complex subsystems, such as the global orbit feedback or the hard real-time interfacement of sensors and actuators, have been implemented exploiting the capabilities of GNU/Linux together with hard real-time extensions. Moreover, most of the interfacement of legacy equipment is made by VME based digital and analog I/O boards running on PowerPC single board computers. A number of device drivers and hard real-time applications have been developed in the years, making a complete system upgrade unfeasible. This turns into an effective limitation against adopting and using modern Ci/CD tools and procedures, which are not compatible with legacy systems.

LEGACY SYSTEMS, AND NOT

Different platforms are in use at Elettra and FERMI, depending on generation and the specific integration requirement. Leaving out some oldest systems running Microware OS-9, which are no more developed, and thus, not supported by INAU, several generations of GNU/Linux based single board computers and rack-mount servers are in use in front-end systems:

- Emerson MVME5100 and MVME6100, running Debian 3 (Linux kernel 2.4)
- Emerson MVME7100, running Ubuntu 7.10, (Linux kernel 2.6)
- Supermicro Intel-based rack-mount servers, running Ubuntu 10.04 (Linux kernel 2.6)
- Supermicro Intel-based rack-mount servers, running Ubuntu 14.04 (Linux kernel 3.14)
- Artesyn MVME2500, running Flop (Linux kernel 4.7)
- Jetway NUC, running Flop (Linux kernel 4.7)
- Beaglebone White and Black, running Flop (Linux kernel 3.14)
- Beaglebone AI, running Voltumna (Linux kernel 5.4)
- Altera Sockit, running Voltumna (Linux kernel 5.4)
- Supermicro Intel-based rack-mount servers, running Voltumna (Linux kernel 5.4)
- Dell Intel-based rack-mount servers, running Voltumna (Linux kernel 5.4)

Moreover, each accelerator deserves a central control system cluster, based on Proxmox, running a large number of virtual machines, where all the standard network services, as well as the Tango databases and the high-layer Tango devices run. All these virtual machines run Ubuntu 18.04 LTS. Control room workstations currently run Ubuntu 18.04 LTS as well.

As a side note, Voltumna is also available as virtual machine image, providing an effective approach to a totally reproducible deployment for virtual machine hosts, based on revision-control.

DEVELOPMENT ENVIRONMENT

Depending on the target system, different approaches are used to build the applications. For legacy VME based systems, as well as older intel-based servers, native target compilation/build, based on reference development hosts, is in place. Newer systems, based on FLOP [1] and, more recently, on Voltumna Linux, exploit a dedicated cross compilation environment.

Since 2019 Git is used for revision control of all software components developed in-house. For control systems software, a structure has been defined to easily navigate repositories, and conventions are in place to guarantee a strict consistency within Git repositories.

DEPLOYMENT REQUIREMENTS

Control system integrity and robustness requirements prevent from spreading administrator access rights to all the developers. Instead, system administrators take care of maintaining and servicing the control systems hosts. However, developers want/need flexibility, and not to depend on sysadmin for installing or updating applications and/or specific application configuration files. On the other side, there is the requirement to keep trace of who installed, updated or downgraded what, when and where. This requirement can be easily solved using modern CI/CD tools on reasonably recent hosts, but is unfeasible with old systems. To make this available on legacy platforms a custom build-and-deploy tool has been developed, named INAU which stands for AUtomatic INstallation, in Italian.

* lorenzo.pivetta@elettra.eu
† alessio.bogani@elettra.eu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP1

INAU

INAU is based on two main services: the first, named "builder" takes care of building software components as soon as it's required, the second, named "installer" is in charge of interacting with developers (users) and installing upon request. INAU services share a MySQL database, to store all the configurations as well as the build and installation history, and a dedicated filesystem to keep all the build products.

INAU is mostly written in Python 3, with some small parts based on shell scripts to interact with legacy build hosts that do not provide Python or where the supported Python version is too old. INAU exploits the Python standard library, in particular multiprocessing but also some additional libraries, notably Flask RESTful [2] with SQLAlchemy [3], Python-ldap [4], Paramiko [5], and GitPython [6].

Currently 13 `tables` have been defined and are in use in the SQL back-end:

- *providers*: Git urls
- *repositories*: enabled repositories
- *distributions*: supported distributions
- *architectures*: supported architectures
- *platforms*: supported platforms
- *builders*: reference hosts for building
- *builds*: per-repository build information
- *artifacts*: per-build hash information
- *facilities*: list of deployment facilities
- *servers*: list of deploy servers
- *hosts*: list of deploy hosts
- *installations*: installation history
- *users*: enabled users

The `table` *providers* is used to keep the list of source code stores, e.g. Git servers; at Elettra that is the self hosted Gitlab server, community edition. The *repositories* `table` stores the list of Git repositories, enabled within INAU, where each repository is assigned a unique ID. Table 1 shows the Linux distributions configured in INAU, whilst the architectures available are *ppc*, *i686*, *x86_64* and *cascadelake-64*. Note that not all the combinations *distribution/architecture* are available, and the supported ones are stored in the `table` *platforms*.

Table 1: Supported Linux Distributions

id	name	version
1	Debian	3.0
2	Ubuntu	7.10
3	Ubuntu	10.04
4	Ubuntu	14.04
5	Ubuntu	16.04
6	Ubuntu	18.04
7	Ubuntu	10.04-caen
8	UbuntuDesktop	18.04
9	UbuntuDesktop	10.04
10	UbuntuDesktop	16.04
11	CCD	0.10beta22

The list of reference hosts available for building the software components is stored in `table` *builders*.

INAU design provides support for C++ applications, Python and Bash scripts and application-specific configuration files. Each "product", hereafter referenced as *artifact*, needs to be uniquely identified, and traceable, within the control systems: for each *artifact* INAU computes the hash[1] and stores it in the `table` *artifacts*.

INAU supports selective deployment, where the developer can specify where an artifact has to be installed. The concept of "facility", which at Elettra and FERMI mostly corresponds to the Tango facilities[2] in use, has been introduced. When installing, the usual case is to specify the target facility, which make the `artifact` available to all the hosts in the facility. As a special case, a single host can be specified.

NFS is used to provide shared storage between hosts in Elettra and FERMI control systems respectively, and each *facility* deserves at least one NFS server. The `table` *servers* is used to associate the *platforms* with the relevant NFS server and provide the installation prefix, that enables sharing different types of *artifacts* using the same NFS server.

The `table` *hosts* is used to keep track of the control system hosts. Each host is associated to the *facility* it belongs, the *server* providing the NFS storage and the *platform*.

The `table` *installations* records the installation requests made by the developers, keeping track of the target host, the build id, the type and the request date.

Moreover, in order to use INAU, the developer must be added to the list of authorized users.

For each Gitlab repository exploiting INAU, a web hook, that executes when a new annotated tag is pushed to the repository, has been configured. The builder service, acting as HTTP server, waits for the Gitlab message. When received, the builder service checks-out the specific tag version for all the *platforms* enabled and remotely builds the software component on the reference hosts by means of ssh. For each platform, build output files are kept between builds; when triggered INAU incrementally builds only what required. This speeds up building components also on slower legacy platforms. Whenever successful, the build is stored into the dedicated location.

The INAU installer service is, as well, based on an HTTP server that exports a REST API. Using a simple HTTP client, such as curl, the user can interact with the service. The installer service also supports authentication, based on LDAP. INAU block diagram is shown in Fig. 1.

USING INAU

The user can perform some useful actions, such as get the list of Gitlab repositories enabled in INAU, the list of *hosts* belonging to a *facility* or request some installation history. Moreover, the user can ask the installation of a repository,

[1] Currently sha256 in used

[2] A Tango facility is identified by the Tango database and all the hosts which belong to

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP1

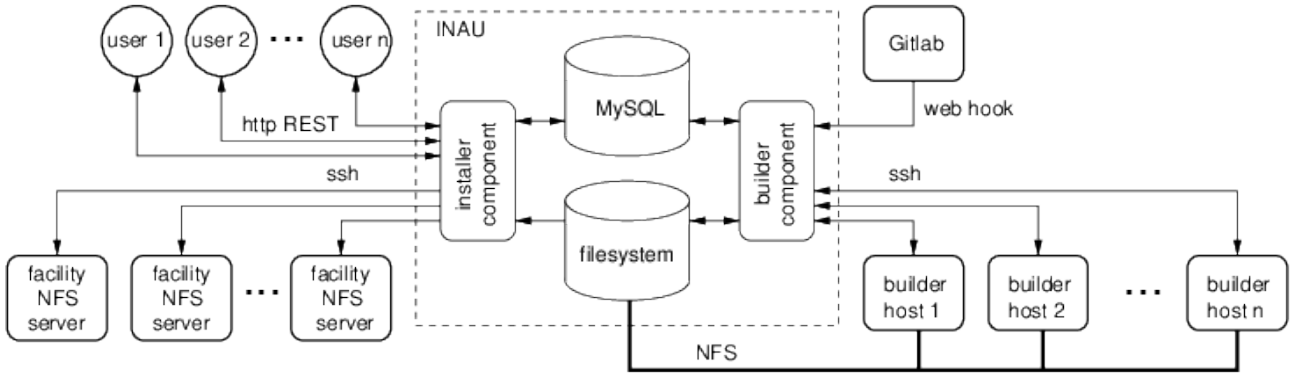

Figure 1: INAU block diagram.

Table 2: Interacting with INAU on the Command Line

```
$ curl https://inau.elettra.eu/v2/cs/repositories ⮐
id   name           provider             distribution  version arch   type      destination
-------------------------------------------------------------------------------------------
3    cs/ds/sfe      ssh://git@gitlab...  Ubuntu        7.10    ppc    cplusplus /bin/
...
1145 cs/gui/miotest ssh://git@gitlab...  UbuntuDesktop 18.04   x86_64 cplusplus /bin/
1148 cs/ds/hipace   ssh://git@gitlab...  Ubuntu        18.04   x86_64 cplusplus /bin/


$ curl https://inau.elettra.eu/v2/cs/facilities/elettra/installations ⮐
host               repository  tag    date                              author
-------------------------------------------------------------------------------
pcl-elettra-cre-01 cs/gui/acdc 1.0.3  Thu, 08 Sep 2022 14:43:39 -0000 lorenzo.pivetta
...
srv-ds-sre-02      cs/ds/4uhv  3.0.10 Wed, 07 Sep 2022 16:27:48 -0000 alessio.bogani


$ curl https://inau.elettra.eu/v2/cs/facilities/padres/installations
-u alessio.bogani -d"repository=cs/ds/4uhv" -d"tag=3.0.8" ⮐
```

specifying the facility, the annotated tag and the username that will be used for authentication.

Some INAU commands and their respective output are listed in Table 2.

CONCLUSION

INAU service is up-and-running since more than three years now, providing developers the autonomy and the flexibility they require, but also guaranteeing a safe approach for system integrity. INAU allows to maintain an accurate and always up-to-date view of applications and application-specific configurations deployed in legacy production systems.

A custom CI/CD service also brings in, as a marginal benefit, a total independence from commercial CI/CD tools that, from time to time, use to change the service policy or introduce proprietary code/plugins.

ACKNOWLEDGEMENTS

Thanks to the Control Systems group members and to the Software for Experiments group members for the useful discussions and suggestions.

REFERENCES

[1] L. Pivetta *et al.*, "FLOP: customizing Yocto project for MVME5xxx PoperPC and Beaglebone ARM", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 958–961. doi:10.18429/JACoW-ICALEPCS2015-WEPGF112

[2] Flask RESTful, https://flask-restful.readthedocs.io/en/latest/

[3] SQLAlchemy, https://www.sqlalchemy.org/

[4] Python-ldap, https://www.python-ldap.org/en/python-ldap-3.4.2/

[5] Paramiko, https://www.paramiko.org/

[6] GitPython, https://gitpython.readthedocs.io/en/stable/