ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA133

# SIRIUS DIAGNOSTICS IOC DEPLOYMENT STRATEGY

L. M. Russo*, LNLS, Campinas, SP, Brazil

*Abstract*

Sirius beam diagnostics group is responsible for specifying, designing and developing IOCs for most of the diagnostics in the Booster, Storage Ring and Transport Lines, such as: Screens, Slits, Scrapers, Beam Position Monitors, Tune Measurement, Beam Profile, Current Measurement, Injection Efficiency and Bunch-by-Bunch Feedback. In order to ease maintenance, improve robustness, repeatability and dependency isolation a set of guidelines and recipes were developed for standardizing the IOC deployment. It is based on two main components: containerization, which isolates the IOC in a well-known environment, and a remote boot strategy for our diagnostics servers, which ensures all hosts boot in the same base operating system image. In this paper, the remote boot strategy, along with its constituent parts, as well as the containerization guidelines will be discussed.

## INTRODUCTION

Sirius [1], like many other particle accelerators and high-energy physics experiments, employs hundreds to thousands of device control system entities. It abstracts the so called *Front-End Controller* (*FEC*) layer, of modern control systems, from the *Client Application* layer and, in the case of a three-tier architecture, *Middle-Layer Services* layer [2, 3].

In the case of facilities based on EPICS [4], like Sirius, *Input-Output Controllers* (IOCs) act as the *FEC* and export its functionalities into *Process Variables* (PVs). As such, *IOCs*, after completing its development cycle (i.e. analysis, design, implementation, test), need to be deployed to the control system in a consistent way, minimizing downtime and coupling to other systems, whereas maximizing the flexibility of changes and rollback strategies in case something fails.

Modern installations typically have thousands of *IOC* instances and hundreds of thousands of *PVs*, in which control, monitoring, archiving and complex interactions with each other take place. Thus, the process of manually compiling, bundling the necessary configuration files and downloading the *IOC* application becomes a burden and an error prone task to the system maintainer. In this sense, the control system environment that has in its foundation principles like consistency and reliability becomes more fragile.

In order to achieve the desired capabilities of such a deployment system, many laboratories and institutes have developed strategies to solve and guide this process employing a myriad of techniques, such as Agile Development, Continuous Integration and Continuous Deployment, and tools, such as *rsync* file synchronizing tool, *Network File System* (NFS), Jenkins, Puppet, Ansible and Containers [5–9].

* lucas.russo@lnls.br

In the next sections, the standards, development cycle and tools chosen by the Sirius beam diagnostics group will be discussed.

## DIAGNOSTICS SYSTEMS

Sirius Diagnostics systems can be summarized by Table 1 extracted from [10]:

Table 1: Summary of Beam Diagnostics Components. LINAC is the Linear Accelerator, LTB is the Linac to Booster Transfer Line, BO is the Booster Ring, BTS is the Booster to Storage Ring and SR is the Storage Ring

| | Linac | LTB | BO | BTS | SR |
|---|---|---|---|---|---|
| DCCT | - | - | 1 | - | 2 |
| FCT | - | 1 | - | 1 | - |
| ICT | 2 | 2 | - | 2 | - |
| Beam Flag | 5 | 6 | 3 | 6 | - |
| Horizontal Slit | - | 1 | - | - | - |
| Vertical Slit | - | 1 | - | - | - |
| Button BPM | - | - | 50 | - | 160 |
| Stripline BPM | 3 | 6 | - | 5 | - |
| Front-End Photon BPM | - | - | - | - | 80 |
| Filling Pattern Monitor | - | - | - | - | 1 |
| Horizontal scraper | - | - | - | - | 1 pair |
| Vertical scraper | - | - | - | - | 1 pair |
| Tune shaker | - | - | 1 | - | 2 |
| Tune pick-up | - | - | 1 | - | 1 |
| Bunch-by-Bunch kicker | - | - | - | - | 2 |
| Bunch-by-Bunch BPM | - | - | - | - | 1 |
| X-ray port | - | - | - | - | 2 |
| Visible light port | - | - | - | - | 1 |
| Streak camera | - | - | - | - | 1 |
| Beam Loss Monitor | - | tbd | tbd | tbd | - |
| Gas Bremsstrahlung Monitor | - | - | - | - | tbd |

Most of the diagnostics presented on Table 1 follow the same development and deployment strategy described in the next section. The exception are the following systems:

- BPM (Button, Stripline, Photon): the system [11] is based on MicroTCA.4 with an x86 AMC CPU board and a CentOS7 operating system. For historical reasons this system, at the time of its inception, did not have the deployment infrastructure available today. As such, management is performed manually with a set of scripts based on parallel *ssh* connections and remote *bash* commands. The plan is to convert it to the herein described deployment strategy.
- Bunch-by-Bunch (Kicker, BPM): the system is based on a custom solution by *Dimtel* [12] with the *IOC* running

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA133



inside the system itself. Currently, there is no plan to change the deployment strategy.

- Streak Camera: the system is based on a commercial solution by Hamamatsu. Currently there is no EPICS IOC planned for the system neither a change in its deployment strategy.

The remainder of the diagnostics employ a set of commercial and in-house developed equipment + *Soft IOCs* running on one of the 8 *Dell PowerEdge R230* diagnostics servers. See [10] for further details.

## DEPLOYMENT PRINCIPLES

The adopted deployment strategy was based on the microservice architecture [13], from which the following principles were used to guide the solution and methods chosen:

- Scalability: applications should be scalable, both in the sense of runtime scalability (e.g., more requests from different clients), and, more importantly, organization scalability, enabling different teams working in parallel with minimal coupling.
- Isolation: applications should be as much isolated as possible from one another, not relying on the host system and relying as little as possible on other services to be functional. Communication must be done through well defined *APIs*.
- Statelessness: applications should not store states and depend on them. This leads to a more complex design and should be avoided.
- Repeatability: applications should always be able to run in the exact same way every time it is deployed or restarted. This ensures the application can be moved from one host to another with minimal effort, enhancing overall system reliability.

## DEPLOYMENT STRATEGY

At Sirius, different groups have taken the task of developing EPICS-based software for the control system, such as: *FECs* for the accelerator; middle-layer services for alarm, monitoring, unit conversion and save-restore systems; high-level applications for beam dynamics and physics processing; beamline controls and experiment automation. As such, development strategies, coding standards, deployment methods, hardware selection and even EPICS modules versions were inherently different from one another, making it difficult to standardize a uniform solution.

In order to tackle the problem, an ongoing set of guidelines and standards is being proposed to help automate the process and to diminish chances of errors. The workflow depicted in Fig. 1 gives an overview of the current software development and deployment status procedures. It was loosely based on [7].

In Fig. 1, three major groups of tasks can be identified: VCS, Standards Checking, Build and Tests (in red); Container Packaging, Container Testing, Container Repository (in blue); Description Update, Refresh Nodes (in yellow). In the next sections these tasks will be described.

### *Building and Testing*

The first four tasks of the workflow defines the basic software development strategy taken by the diagnostics group. All of the developed code is versioned through *git*, either hosted in Github [14] or in the internal Sirius Gitlab [15] instance, following the Gitflow [16] methodology of branching and releasing.

When a developer decides to release a new version of a software, it must check if it adheres to a set of internal rules, guidelines and style guide. This is a manual step of inspecting the code to check for missing dependencies, error in coding styles, system paths and necessary scripts for container packaging. Specifically for EPICS IOCs a set of scripts and support files must be used to serve as the entrypoint for upper management/packaging software, such as *Docker* [17] and *systemd* [18], both supported for Sirius diagnostics IOCs.

Finally, the code is built by using a standard build system, such as: *Make* [19], *CMake* [20] or *Gradle* [21] and tested against the real equipment or infrastructure.

### *Container Generation*

The next group of tasks is related to the image generation to be used in the control system (in this context, *image* always refer to a Docker image to be run as a Docker container). The idea of completely isolating the application from the host system with virtually no coupling between them (with the exception of a modern linux kernel $\geq$ 3.10+ and Docker Engine) was very attractive and suitable to a non-uniform software development culture.

For that matter, a series of reusable Docker images [22–25] were built with the purpose of serving as a basis for other software, specially EPICS IOCs, published on DockerHub [26] and on the internal Sirius *Docker Registry* [27]. Currently, the images are built using a custom script [28] that compiles EPICS base and EPICS modules from source for a variety of Linux distributions, alongside its dependencies. The usage of Debian packages, particularly the NSLS-II EPICS Debian repository [29, 30], is being considered as a replacement of the script. This has the potential of being easier to use, more reliable as it uses the native Debian packaging scheme and closer to the EPICS community.

The diagnostics EPICS IOC images will then inherit from one of the base images (according to its necessity) and build its dependencies and IOC on top of it. In some special cases, applications require additional EPICS modules not available at the base images. Hence, the extra module could either be built alongside the EPICS IOC application in the same image or as an additional image on top of the existing ones. Traditionally, it is preferred to use the later, as it enhances

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA133

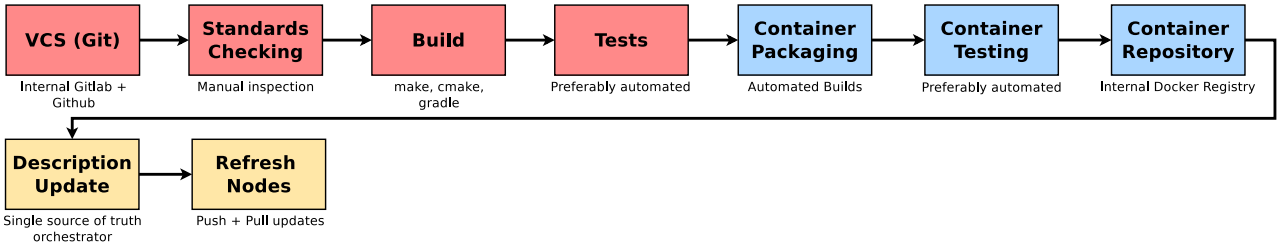


Figure 1: Deployment Workflow. In red (VCS, Standards Checking, Build and Tests), tasks that are related to building and testing a specific control system component. In blue (Container Packaging, Container Testing, Container Repository), tasks that are related to packaging, specifically containers in this case. In yellow (Description Update, Refresh Nodes), tasks that are related to deployment.

decoupling and promotes reusability. The current image hierarchy can be seen in Fig. 2.

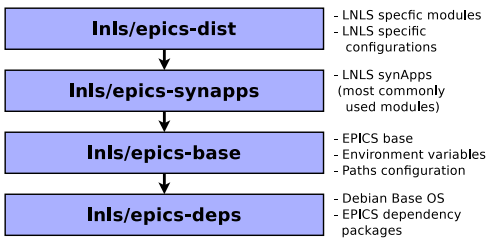


Figure 2: LNLS Docker Images Hierarchy.

The most basic image (*lnls/epics-deps*) contains only a standard Linux distribution, typically Debian, and EPICS Base package dependencies, either in the native Linux distribution package manager or compiled from source, if not available. The decision of not bundling EPICS base alongside its dependencies was taken to enhance the reusability of this image to most of the EPICS base versions. So, in this way, EPICS base versions 3.14, 3.15, 3.16 and 7.0 could all inherit from *lnls/epics-deps* and reuse most of the already installed dependencies.

The EPICS base toolkit, versions 3.14, 3.15, 3.16 and 7.0, is packaged in image *lnls/epics-base* with distinct tags and contains the basic EPICS components and environment variables needed by all EPICS applications.

On top of that, the most commonly used EPICS modules by Sirius diagnostics are bundled inside the *lnls/epics-synapps* and it contains a customized SynApps [9] distribution [31]. It removes unused EPICS drivers and modules, while keeping the most generic ones, such as *asyn*, *alive*, *areaDetector*, *busy*, *calc*, *devIOCStats*, *sequencer*, *motor* and *streamDevice*.

The last image is called *lnls/epics-dist* and it contains specific Sirius EPICS configuration. In the future, Sirius specific modules could be bundled in this image, so all IOCs built from it can benefit.

EPICS modules not contained in the previous hierarchy, such as the Aravis GigE EPICS module [32], are generally bundled in a separated image inheriting from *lnls/epics-dist*. Then, on top of it, the EPICS IOC can be built separately, reusing the image. An example of this usage can be seen in Fig. 3, depicting the Aravis EPICS module and camera EPICS IOC building on top of the Sirius base images.

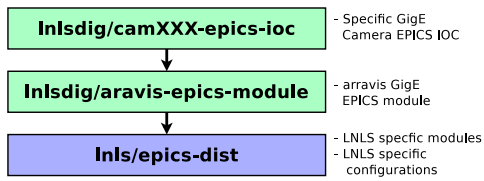


Figure 3: LNLS Camera EPICS IOC and an Aravis EPICS module interaction with the LNLS base images.

Overall, images can grow large in size, as unneeded packages, files and documentation are kept in intermediate *image layers*. On building the described images care was taken to remove all of the generated artifacts in the same layer as they are created or using Docker Multi-Stage [33] builds to copy only the final products. Still, instead of compiling from source, it is deemed best to use the Linux distribution packages, such as the EPICS Debian packages from NSLS-II, to enforce that no intermediate files are kept in the images. This is considered good practice, even though the usage of *OverlayFS* [34] by the Docker engine guarantees that the disk usage of an image will not increase, but will be "shared" or *overlaid* with other images, as long as it uses the same hierarchy. For instance, having two EPICS IOC images inheriting from the same *lnls/epics-dist* image will, roughly speaking, count as the size of just one *lnls/epics-dist* plus the size of the EPICS IOCs images.

Container testing is done to ensure that the software runs exactly the same as the original version, running outside the container environment. No automated infrastructure currently exists for that.

The last task is to push the generated and tested image to an internal container registry, such as Docker Registry, so the internal control system network can fetch the image without relying on foreign network infrastructure.

## *Deployment*

The last group of tasks involves the actual image deployment to the control system, which can be crucial as care must be taken to avoid disrupting other software updates

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA133



and subsystems. Before proceeding, one important Docker detail must be explained.

**Docker Network Isolation** By default, docker containers are run with no network ports published to the host system or the host network. This means that no inbound/outbound communication is allowed. In order to solve this docker offers the *EXPOSE* Dockerfile instruction and the runtime `-p` option for `docker run` command (or equivalent) to actually publish a container port mapped to a certain host port.

This solution, however, only works if a single EPICS IOC is run at the host, mapping the Channel Access (CA) EPICS ports 5064/tcp (CA server connection), 5064/udp (CA search messages), 5065/tcp (CA Repeater process communication), 5065/udp (CA Beacon messages) to the host. If more than one IOC must run in the same host, EPICS will randomly choose an ephemeral port for the CA server connection. However, inside the container, the IOC will always bind to the default ports as it is network isolated from other containers. This makes the task of publishing the network ports beforehand non-trivial for the host, as the IOC will embed the default ports in the protocol messages, but the host will use a distinct mapping. Hence, effectively instructing the clients to use the wrong ports for communication.

In order to overcome these issues, two options were initially evaluated: (i) use a port translator service in the host (similar to a typical NAT service) to dynamically change the port to/from the host/container; (ii) use the `EPICS_CA_SERVER_PORT` environment variable to change the CA server connection port so each IOC would use a specific one defined beforehand. These options were deemed too intricate to maintain, so a simpler solution was adopted based on the ability of docker to use the same network as the host (i.e., without network isolation) by using the docker option `--network=host`. This effectively solved the issue as even with the ephemeral ports, clients could correctly use them. The drawback of this technique is that the network isolation is effectively bypassed, allowing the container to use all of the ports on host, diminishing security. Additionally, the steps described in [35] were necessary to ensure that clients, in either unicast or broadcast CA search message mode, could reach all of the IOCs on the hosts.

**IOC Servers Configuration** On Sirius, diagnostics IOC servers follow the *diskless* approach, relying on a remote boot strategy with NFS mounts for each of the servers' root filesystem, home and EPICS autosave directories. This ensures consistency and repeatability as all of the IOC servers have the exact same configuration and do not rely on anything not available at the read-only root filesystem.

Specifically, the root filesystem, created by a set of scripts available in [36], is intended to be minimal and tightly controlled as it affects all hosts.

After successfully booting into the *rootfs*, a set of *systemd* services, bundled within the rootfs, starts all of the configured docker containers for that particular host along with its runtime parameters. Moreover, hook scripts can further configure the boot process with additional commands.

An ongoing effort is currently in place to evaluate the usage of container orchestration technologies, such as *Docker Swarm* [37]. In this way, instead of distributing docker configuration files to all hosts, a *Swarm Manager* could orchestrate all of the containers from a single place, centralizing all of the configuration on the Manager node. As this is completely independent from the other approach, both can coexist in the same system with any degree of commitment. As of this moment, two services are successfully running in *swarm* mode: Zabbix Agent [38], for gathering statistics about the running operating system, and Portainer Agent [39], for a web-interface overview of the node cluster and all of its running services.

**Remote Boot Procedure** The detailed remote boot procedure and related infrastructure are depicted in Fig. 4

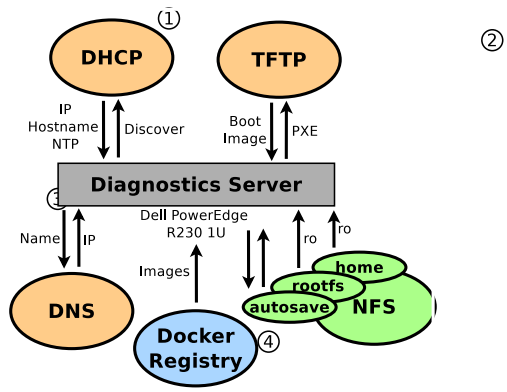


Figure 4: Sirius Diagnostics Remote Boot Strategy. The first step (1) is the IP/Hostname/NTP server assignment to the hosts plus the location of a TFTP server and NFS server for PXE boot. The second step (2) is the requisition of the initial boot image (embedded with the instruction to mount the root filesystem via an NFS server) to the TFTP server. The third step (3) is the mount of the root filesystem, hosts' homes and EPICS autosave directories. The fourth step (4) is the fetching of docker images from an internal docker registry. DNS server is queried in each step to translate domain names to actual IP addresses.

The first step is the DHCP server discover, in which the host tries to get a valid IP address. In this case, the hosts are also configured to receive its hostname, local NTP servers and the location of the TFTP and NFS servers for usage in further steps. The docker configuration files can be seen in [40].

The second step is fetching the initial boot image. This is achieved by loading a minimal *initramfs* image from an TFTP server [41]. After the basic boot steps are completed, the rootfs is mounted via NFS from an NFS server.

The third step is the mounting of the rootfs. Specifically for Sirius, the image is created by [36] and exported by a NFS server configured according to [42]. Along with the

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA133

rootfs, this NFS server also exports the home directories for all hosts, which specifies the docker configuration files for all of the EPICS IOCs to be run and its runtime parameters. Furthermore, an EPICS autosave directory is mounted with read-write permissions, thus enabling EPICS IOCs to perform the save-restore functionality.

The fourth and last step is the actual download of docker images from an internal docker registry. After that, the images are run according to the configuration files, which specifies, among other things, the Sirius EPICS PV prefixes, monitoring port numbers and IOC type.

The DNS server is used by all services to translate domain names to actual IP addresses, so no static IP addresses are used.

## ADVANTAGES AND DRAWBACKS

The overall strategy of employing diskless servers with remote boot brought assurance and convenient auditing capabilities that the running system is exactly the one being exported by the NFS server. Deployment is easilly tested beforehand as the production base system can be trivially and consistently replicated. On the other hand, the necessary infrastructure could be argued to be complex, requiring specialized personnel to maintain it. Nevertheless, as more systems adopt the same deployment strategy the benefits of standardization, flexibility and maintenance outweight the disadvantages over the course of years.

The containerization of IOC applications follows the same principles of the remote boot strategy and it was a natural and modern way to package applications easily and reliably, yet still achieving the wanted microservice architecture principles. Moreover, no measurable performance penalty was perceived by the usage of containers.

One drawback of the container packaging could be pointed out as the complexity of docker tools either for building the image and for running the container. At Sirius, experiments showed that software developers took approximately 2−3 weeks in order to learn docker at the level needed to package IOCs, create new images and debug containers. With a more widespread adoption of containers, this is effectively being absorbed by developers as part of their skill set. Hence, diminishing the impact of adopting container technologies.

## FUTURE WORK

Currently, the docker containers are run in a *pull* scheme, in which the hosts pull the docker images from a registry, using the configuration provided by the exported hosts' homes. This effectively shifts the runtime scalability and repeatability to hosts themselves, diminishing the effectiveness of the overall solution. A more suitable approach would be to concentrate all of the hosts' configuration to a single node, so that it could be better managed, more scalable and monitored. The solution being evaluated is the Docker Swarm tool that brings these advantages.

Additionally, more automation is required to perform automatic standards checking against the developed IOC code, unit and integration tests and continuous integration and deployment. A typical solution for implementing these *DevOps* tasks is the usage of an automation system, such as Jenkins [43], which is also under evaluation.

## CONCLUSION

In this paper, the current state of the EPICS IOC deployment for Sirius diagnostics was described from the software development point of view to the actual container generation and deployment. The difficulties and the solutions found were also discussed, such as the docker network interaction with EPICS dynamic ports. Given the rate of change of the control system, specially during commissioning, environment isolation given by the docker containers and manageability from docker swarm, the diagnostics team is very satisfied with the overall performance and flexibility, despite some minor inconveniences: extra steps in packaging the EPICS IOCs and perceived inability to quickly change the IOC code, as any changes would need to be repackaged in the image.

Moreover, the diskless approach used by the diagnostics servers proved very useful and consistent to provide a repeatable environment to all hosts.

## ACKNOWLEDGEMENT

The author of this paper would like to thank Sergi Puso from ALBA Synchrotron, for kindly presenting the ALBA remote boot and other insightful discussions, and the Docker community for providing an open source environment for containers and containers orchestration tools.

## REFERENCES

[1] A. R. D. Rodrigues *et al.*, "Sirius Status Update", in *Proc. IPAC'19*, Melbourne, Australia, May 2019, pp. 1381–1384. doi:10.18429/JACoW-IPAC2019-TUPGW003

[2] V. Vuppala *et al.*, "Distributed Information Services for Control Systems", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper WECOBA02, pp. 1000–1003.

[3] R. Huhmann *et al.*, "The FAIR Control System - System Architecture and First Implementations", in *Proc. ICALEPCS'13*, San Francisco, CA, USA, Oct. 2013, paper MOPPC097, pp. 328–331.

[4] Experimental Physics and Industrial Control System (EPICS), http://www.aps.anl.gov/epics

[5] I. Arredondo and J. Jugo, "Containerized Control Structure for Accelerators", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 816–821. doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

[6] D. P. Brodrick, T. Brys, T. Korhonen, and J. E. Sparger, "EPICS Architecture for Neutron Instrument Control at the European Spallation Source", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 1043–1047. doi:10.18429/JACoW-ICALEPCS2017-WEBPL01

[7] M. G. Konrad, D. G. Maxwell, and G. Shen, "Continuous Integration and Continuous Delivery at FRIB", in *Proc. PCaPAC'16*, Campinas, Brazil, Oct. 2016, pp. 145–147. doi:10.18429/JACoW-PCAPAC2016-FRITPLC001

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA133



[8] 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.

[9] T. Mooney, "synApps: EPICS-Application Software for Synchrotron Beamlines and Laboratories", in *Proc. PCaPAC'10*, Saskatoon, Canada, Oct. 2010, paper THCOMA02, pp. 106–108.

[10] S. R. Marques, "Beam Diagnostics Systems for Sirius Light Source", in *Proc. IBIC'17*, Grand Rapids, MI, USA, Aug. 2017, pp. 89–93. doi:10.18429/JACoW-IBIC2017-MOPCF09

[11] S. R. Marques, G. B. M. Bruno, L. M. Russo, H. A. Silva, and D. O. Tavares, "Commissioning of the Open Source Sirius BPM Electronics", in *Proc. IBIC'18*, Shanghai, China, Sep. 2018, pp. 196–203. doi:10.18429/JACoW-IBIC2018-TUOC03

[12] Dimtel Products, http://www.dimtel.com/products/index

[13] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead", IEEE Software, vol. 35, no. 3, pp. 24–35, May 2018. doi:10.1109/MS.2018.2141039

[14] Github, https://github.com

[15] Gitlab, https://about.gitlab.com

[16] V. Driessen, "A successful Git branching model", https://nvie.com/posts/a-successful-git-branching-model

[17] Docker, https://www.docker.com

[18] systemd, https://www.freedesktop.org/wiki/Software/systemd

[19] GNU Make, https://www.gnu.org/software/make

[20] CMake, https://cmake.org

[21] Gradle Build Tool, https://gradle.org

[22] LNLS EPICS Deps Docker Image, https://hub.docker.com/r/lnls/epics-deps

[23] LNLS EPICS Base Docker Image, https://hub.docker.com/r/lnls/epics-base

[24] LNLS EPICS SynApps Docker Image, https://hub.docker.com/r/lnls/epics-synapps

[25] LNLS EPICS Distribution Docker Image, https://hub.docker.com/r/lnls/epics-dist

[26] DockerHub, https://hub.docker.com

[27] Docker Registry, https://docs.docker.com/registry

[28] LNLS epics-dev script EPICS Module, https://github.com/lnls-sirius/epics-dev

[29] EPICS Packaging for Debian Linux, https://github.com/epicsdeb

[30] NSLS-II EPICS Debian Repository, https://epicsdeb.bnl.gov/debian

[31] LNLS EPICS SynApps distribution, https://github.com/lnls-dig/support

[32] Aravis EPICS Module, https://github.com/areaDetector/aravisGigE

[33] Docker Multi-Stage Builds, https://docs.docker.com/develop/develop-images/multistage-build

[34] Docker OverlayFS Storage Driver, https://docs.docker.com/storage/storagedriver/overlayfs-driver

[35] How to Make Channel Access Reach Multiple Soft IOCs on a Linux Host, https://wiki-ext.aps.anl.gov/epics/index.php/How_to_Make_Channel_Access_Reach_Multiple_Soft_IOCs_on_a_Linux_Host

[36] Sirius Diagnostics Debian Root Filesystem, https://github.com/lnls-sirius/debian-rootfs

[37] Docker Swarm, https://docs.docker.com/engine/swarm

[38] Zabbix Monitoring Service, https://www.zabbix.com

[39] Portainer Agent, https://portainer.readthedocs.io/en/stable/agent.html

[40] Sirius Docker DHCPD, https://github.com/lnls-sirius/docker-dhcpd-composed

[41] Sirius Docker TFTP, https://github.com/lnls-sirius/docker-tftp-hpa-composed

[42] Sirius Docker NFS Server, https://github.com/lnls-sirius/docker-nfs-server-composed

[43] Jenkins, https://jenkins.io