ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPL006

AUTOMATIC DEPLOYMENT IN A CONTROL SYSTEM ENVIRONMENT*

M. Konrad†, S. Beher, A. Lathrop, D. Maxwell, J. Ryan
Facility for Rare Isotope Beams, Michigan State University, East Lansing, USA

Abstract

Development of many software projects at the Facility of Rare Isotope Beams (FRIB) follows an agile development approach. An important part of this practice is to make new software versions available to users frequently to meet their changing needs during commissioning and to get feedback from them in a timely manner. However, building, testing, packaging, and deploying software manually can be a time-consuming and error-prone process. We will present processes and tools used at FRIB to standardize and automate the required steps. We will also describe our experience upgrading control system computers to a new operating system version as well as to a new EPICS release.

INTRODUCTION

FRIB [1] is a project under cooperative agreement between the US Department of Energy and Michigan State University (MSU). It is under construction on the campus of MSU and will be a new national user facility for nuclear physics. Its driver accelerator is designed to accelerate all stable ions to energies >200 MeV/u with beam power on the target up to 400 kW [2]. Commissioning of the second linac segment is currently underway and the accelerator is planned to support routine user operations in 2022 [3].

FRIB's controls group strives to support commissioning and operation by rolling out bug fixes and new features as fast as possible. To make this happen controls engineers are following principles of agile software development which include iterative, incremental and evolutionary development and a short feedback and adaption cycle. Unfortunately, this approach can be slowed down significantly by the fact that building and deploying control-system software can be a complex and error-prone process that often requires considerable manual work by experts. In the following we will describe how we speed up the build and deployment process for FRIB's controls software by following continuous integration (CI) and continuous delivery (CD) principles.

CONTROL SYSTEM ENVIRONMENT

The vast majority of computers on FRIB's control-system network are based on the x86-64 architecture. This includes workstations, servers, as well as industrial computers in cPCI and MicroTCA form factor. FRIB has standardized on Debian GNU/Linux as operating system for control-system computers. Even hard real-time applications are running on Debian GNU/Linux with a real-time kernel rather than on special real-time operating systems like VxWorks or RTEMS. This heavily standardized environment helps to keep the test matrix small, simplifies the CI infrastructure and allows sophisticated tools developed in the IT industry to be leveraged for software deployment, configuration management as well as for monitoring.

FRIB's controls network is independent from its office network. For security reasons the two networks use separate hardware and a firewall allows only predefined connections between them. In addition to the production control system used to operate the accelerator, FRIB's controls group operates a development control system on a separate network to support development and testing. The development network mimics the architecture of the production network. In particular both networks share the same network topology, run the same IT infrastructure services (DHCP, DNS, storage servers, hypervisors,...) as well as the same control-system services (Alarm Server, Archiver Appliance, Channel Finder,...). A limited number of "control-room" workstations as well as control-system devices like programmable-logic controllers, motor controllers, LLRF controllers etc. are available to support development and testing. In some cases simulation applications are mimicking the behavior of hundreds of devices based on a simplified model [4]. Virtualization is used extensively on both networks to increase availability and flexibility as well as to reduce hardware and maintenance cost. The similarity between both networks allows many bugs to be discovered on the development network before software is deployed to the production network.

CONTINUOUS INTEGRATION

All source code required to build software for FRIB's control system is stored on a central Git [5] repository server on the office network. The revision-control workflow largely follows the Gitflow [6] approach which requires developers to implement new features and bug fixes on feature branches allowing them to work on their feature without the risk of breaking other developer's build. Each repository contains branches for "unstable" and "release" targets. Completed feature branches are merged into an "unstable" branch. Tested software is released by merging from an "unstable" branch into a "release" branch. For both branches software is automatically built, tested and packaged on a Jenkins CI cluster [7]. The resulting Debian packages are pushed into separate Aptly [8] repositories for the development and production environments. See [9] for an in-depth description of FRIB's CI approach.

AUTOMATIC DEPLOYMENT

The Debian package repository as well as the Git repositories are mirrored into the corresponding controls network

* Work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661
† konrad@frib.msu.edu

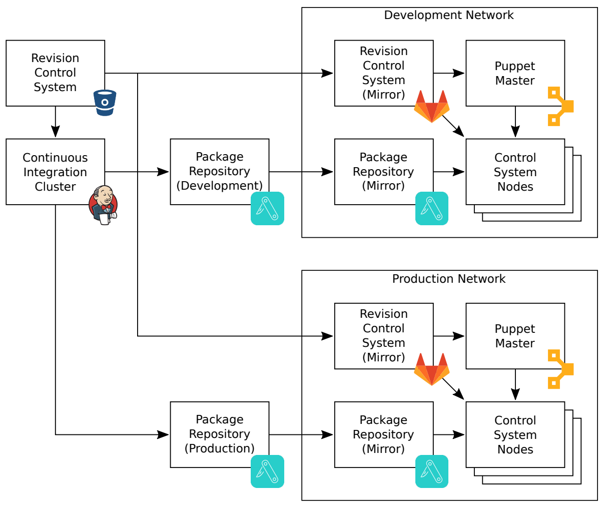
Development Network
Revision Control System
Revision Control System (Mirror)
Puppet Master
Continuous Integration Cluster
Package Repository (Development)
Package Repository (Mirror)
Control System Nodes
Production Network
Revision Control System (Mirror)
Puppet Master
Package Repository (Production)
Package Repository (Mirror)
Control System Nodes

Figure 1: FRIB's development and production controls networks are separate from the office network. Configuration data and software package are mirrored to these networks.

allowing them to operate independently from the office network (see Fig. 1). This ensures control-system machines can be reinstalled even if services on the office network are down.

FRIB relies on the open source versions of Puppet [10] and Ansible [11] for managing the configuration of control-system computers (*nodes*). Both tools use a declarative language to describe the desired state of the target nodes in form of *manifests* (Puppet) and *playbooks* (Ansible). When the tools are run, they inspect the current state of the target node and perform the actions required to reach the desired state specified in the manifest/playbook. In contrast to a simple shell script that for example installs a software package, customizes its configuration files and starts the corresponding service, a well designed Puppet manifest or Ansible playbook is idempotent – applying it multiple times doesn't install the software again, a required line is added to a configuration file only once and services aren't restarted unnecessarily. In addition to avoiding potentially expensive operations this prevents possible side effects like unneeded restarts of services. This property of the tools allows them to run periodically, deploying updates automatically as they become available in Git and the Debian repository while at the same time correcting undesired configuration drift.

Puppet requires an *agent* running on each target node which pulls the desired configuration from a central *master* server. Ansible on the other hand pushes configuration from a central server to the target nodes through SSH. It thus can manage nodes on which an agent cannot be installed.

Both Ansible and Puppet allow code to be bundled up in modules. Modules for common IT administration tasks like managing web servers, database management systems etc. are available as open-source software. FRIB has selected Puppet as the primary configuration management tool for its control system due to the availability of a wide range of Puppet modules that are often offering more flexibility than the modules available for other tools. Other reasons for choosing Puppet were its powerful language, its (compared to Ansible) high performance, its built-in logging capabilities and the fact that Puppet is also used for managing Linux nodes on FRIB's office network allowing for code and expertise to be shared. In the following FRIB's deployment strategy and as well as the tools used will be described for EPICS IOCs and Channel Access gateways. The deployment of other key control system components like archivers and other middle-layer services has been automated as well but is not described in this publication.

EPICS IOCs

The about 300 EPICS IOCs are among the most critical components of FRIB's control system. It is important to ensure they are all running the desired software and configuration while at the same time keeping IOC downtime at a minimum. At FRIB support libraries are built on the CI cluster and get installed as Debian packages whereas IOCs are built from source code on the target machines. This approach has the following advantages:

- Projects containing handcrafted C/C++ code are built in a predictable way on the CI cluster. Compilation errors and failing tests are discovered during development.
- Uninstalling a Debian package removes all contained files. This allows libraries and header files to be removed from system folders cleanly.
- Dependencies between packages ensure that required libraries and tools are installed and compatible.
- Bugs in an IOC's database files can be fixed in the field quickly.
- IOC database development is sped up for devices which are not available on the development network as well as for systems that interact with a large number of devices.

Note that although FRIB's deployment strategy allows database files to be modified in the field, the majority of changes are tested on the development network before they are deployed to the production network. However, the ability to apply a hot-fix in the field can help to reduce accelerator downtime.

A disadvantage of building IOCs on the target nodes is that this causes the deployment to be more complex than installing a package. FRIB has developed the EPICS Soft IOC Puppet module [12] to simplify the required steps of managing IOCs. It provides defaults that make it easy for IOC engineers to follow best practices and facility conventions. At the same time the module allows IOC engineers to deviate from the default if necessary. The following Puppet manifest provides an example for managing an EPICS IOC process with this module:

```
$iocbase = '/usr/local/lib/iocapps'
```

```
class { 'epics_softioc':
  iocbase => $iocbase,
}

package { 'epics-asyn-dev':
  ensure => latest,
}

vcsrepo { "${iocbase}/myioc":
  ensure   => 'latest',
  provider => 'git',
  source   => 'https://git-server/repo.git',
  owner    => 'softioc',
  group    => 'softioc',
}

epics_softioc::ioc { 'myioc':
  ensure     => running,
  enable     => true,
  bootdir    => 'iocBoot/ioclinux-x86_64',
  log_server => 'log.example.com',
  subscribe  => [
    Package['epics-asyn-dev'],
    Vcsrepo["${iocbase}/myioc"],
  ],
}
```

When instantiating the `epics_softioc` class, it has to be pointed to a directory containing the source code of the IOCs running on this node. The class will automatically ensure that libraries and tools required to build and run IOCs are installed. In particular this will install packages providing EPICS Base, procServ [13], the system's native compiler as well as Make. IOC-specific libraries and tools like the Asyn support library can be installed with a `package` resource.

Rather than defining the `iocbase` variable in each node's manifest, it is recommended to define it facility-wide or to look up the directory in Hiera [14] so that engineers can find the IOC directories in the same location on all nodes. The `epics_softioc` class needs to be instantiated once on each node that should run EPICS IOCs. If the roles and profiles design pattern [15] is followed, this would typically be done in a Puppet profile that gets included for each node running IOCs.

The remaining lines in the manifest are describing an IOC instance. They can be repeated if multiple IOC instances are supposed to run on a node. In the example given the `vcsrepo` Puppet module [16] is used to clone a Git repository containing the IOC code into the IOC base directory. Other methods for providing the source code like copying the files using Puppet are supported as well. By default the `epics_softioc::ioc` defined type automatically compiles the IOC code contained in the sub-directory matching the defined type's name. If `ensure => running` is specified, the IOC will be started automatically using the node's init system. In the above example the service will also be enabled which causes it to be started automatically on system boot. `epics_softioc::ioc` automatically configures IOC services to be started after the network has been brought up. On nodes using systemd it can also ensure that other services are running and that certain network drives are mounted before the IOC service is started (e. g. for autosave).

IOCs automatically get started under procServ running an IOC shell script (by default `st.cmd`) in the specified IOC boot directory. By default procServ is configured to restart crashed IOC processes after writing a core file to support debugging. Local users can connect to the shell of a running IOC through telnet or a Unix domain socket. `epics_softioc::ioc` automatically configures logging of all activities on the IOC shell including automatic rotation and compression of old log files. The `log_server` argument causes log messages from the IOC process to be send to the specified log server. If multiple IOC processes should be managed on a node, the `vcsrepo` and `epics_softioc::ioc` resources can be repeated for each IOC. By default the EPICS Soft IOC Puppet module runs IOCs with limited privileges. The required users and groups are created automatically and directory permissions are set appropriately.

Subscribing the `epics_softioc::ioc` resource to the `package` and `vcsrepo` resources automatically triggers a rebuild and a successive restart of the IOC when a new version of the source code is checked out or when a support library is updated. This allows IOCs to be updated in a fully automatic way. If automatic recompilation or automatic IOC restarts aren't desired these features can be disabled.

Defaults provided by the Puppet module can be overridden facility-wide to make it easy for engineers to follow their conventions.

Channel Access Gateway

For security reasons the FRIB controls network is split into multiple sub-networks. Channel Access connections between these networks are only possible through Channel Access gateways [17]. The `epics_gateway` Puppet module [18] has been developed to simplify management of these gateways. Its architecture has some similarities with the EPICS Soft IOC module. Two classes need to be instantiated to bring up a gateway: `epics_gateway` ensures the packages providing the gateway, procServ and the required tools are installed whereas `epics_gateway::gateway` configures and starts instances of the gateway. Multiple gateway instances on the same node are possible to support bidirectional gateways or setups where more than two networks are supposed to be connected through gateways running on the same node. The configuration files of the gateway defining the permissions for access through the gateway are provided as a directory for each gateway instance. Again these directories can be checked out from a revision control system using the `vcsrepo` Puppet module. The Puppet module automatically restarts a gateway instance when a new version of its configuration becomes available to ensure the changes become active.

Firmware Deployment to Embedded Devices

FRIB operates about 350 RF amplifiers that consist of various sub-components each running their own firmware. In total more than 4 000 firmware images need to be managed for these amplifiers. The main controller embedded in each amplifier rack runs Linux on an ARM-based single-board computer. Firmware updates are performed by copying the image to the controller using the Secure Copy Protocol (SCP) and running a command over Secure Shell (SSH) to program the image. Puppet agent is not available on these devices. However, the controllers come with a Python interpreter allowing the firmware update process to be managed using Ansible.

FRIB has developed an Ansible playbook which compares the current firmware versions read out by the IOC with the firmware version available in Git. If an update needs to be performed the playbook copies the relevant firmware images to the controller, executes the corresponding update commands and reboots the controller.

Ansible is also used to manage SSH keys for accessing the controllers and to disable password-based remote logins with the weak default password set by the vendor. This improves IT security by replacing shared passwords by keys that can be removed individually to revoke access for a user.

DEPLOYMENT PROCESS AND EXPERIENCE

Changes are deployed continuously to nodes on the development network. When an engineer merges a new feature to an "unstable" branch it will be built and deployed automatically within a few minutes. This enables engineers to test their code frequently. Releasing to production means merging from an unstable branch to a release branch; again deployment is performed automatically. Branch permissions on the Git repositories prevent direct pushes to release branches and enforce pull requests to be reviewed by another engineer who is familiar with the operations schedule. This effectively prevents deployment at inopportune times.

Commissioning vs. Operation

During the commissioning phase of an accelerator changes typically get deployed in small increments in a "rolling release" pattern. During operations, on the other hand, engineers might need to wait for a maintenance shutdown before they can deploy new features. In the meantime, which at FRIB can be as long as half a year, a large number of changes to various systems might pile up. Some of them might require multiple components to be upgraded at the same time – e. g. a piece of software and the corresponding Puppet code managing its configuration. In some cases software on multiple nodes might need to be updated to ensure compatibility. Interactions between a large number of such changes can be difficult to understand, resulting in higher risk. At FRIB this risk is minimized by integration testing the to-be-released software on the development network. This requires a feature freeze in preparation for a maintenance shutdown. FRIB maintains multiple collections of software to support this approach. Collections targeted at the development network are called `uc1`, `uc2`, `uc3`,... ("unstable software collections") while software collections targeted at the production environment are named `fc1`, `fc2`, `fc3`,... ("FRIB software collections"). For example while an older collection (`fc1`) might be in use on the production network, a new release (`uc2`) might be under test on the development network in preparation for deployment during an upcoming shutdown. At the same time some developers might already have started working on features that are intended for the next software collection half a year later (`uc3`). Once a package has been tested on the development network it can be released to the corresponding FRIB software collection by merging into the corresponding release branch. New software collections are planned to be released during each maintenance shutdown.

Management of Puppet Code

The sum of all Puppet configuration files (manifests, templates, modules, Hiera data etc.) for all nodes on the network is called an *environment*. Puppet supports multiple environments allowing changes to be implemented in a separate environment before integrating them into the production environment. Each node can be pointed to one of these environments. A new environment can thus be tested on a small number of machines before applying the changes to all nodes. Environments provide a way of rolling out changes that affect multiple machines in a coordinated way. FRIB leverages r10k [19] to check out the required modules from a revision-control system to combine them with the manifests for each environment. This allows modules to be developed and revision controlled separately from the rest of the environment. By translating Git branches into environments r10k makes it easy to create and destroy short-lived environments for testing.

Staged Upgrades

At FRIB major changes affecting many machines like upgrading to a new major EPICS release or operating system version are typically performed machine by machine or in small batches to keep the risk low and to simplify scheduling these upgrades. An operating system upgrade generally requires transitioning to a new software collection which has been build for the new operating system and the libraries it ships with (e.g. from `fc1` to `fc2`). This means a slightly different system configuration is required (e. g. pointing to the `fc2` package repository rather than to the `fc1` repository). A new Puppet environment can be created based on the existing environment to test these changes before they are merged into the production environment. Although not strictly necessary, at FRIB nodes are usually reinstalled from scratch in the context of an operating system upgrade. This step ensures that the Puppet configuration is still working correctly with the new operating-system version and guarantees that the node can be reinstalled in case of a hardware failure.

The described approach of upgrading node-by-node allows a mix of multiple operating systems or EPICS versions to be used in parallel on the control-system network. Systems benefiting from the latest operating system/compilers/libraries can thus be upgraded early while the upgrade of other systems is still in preparation. In general we strive to complete these kind of upgrades within a few weeks to avoid the effort of maintaining multiple software collections for a longer time.

Experience

At FRIB it turned out to be difficult to clearly separate management of the controls applications from management of the underlying infrastructure – in particular when custom kernel drivers needed to be loaded, permissions needed to be granted to allow access to hardware or when real-time constraints needed to be met. As a consequence FRIB has decided to follow a DevOps strategy with control-system engineers and IT administrators working as part of the same team. In practice this means that IT administrators and controls engineers both have administrative privileges on nodes running IOCs and that the Puppet code for IOC nodes is maintained collaboratively while manifests dealing with IT infrastructure like authentication etc. are only accessible by IT administrators.

A valuable side effect of automating the deployment of control-system nodes with a configuration management tool is that their manifests/playbooks document all steps in a standardized way which enables engineers unfamiliar with the details of a system to understand its deployment if necessary. At FRIB we found that this transparency encourages team work. Additionally, the ability to execute this "documentation" allows its completeness and correctness to be verified (e. g. by wiping out nodes at a favorable time and rebuilding them from scratch). In a control-system environment where availability is important this can help to keep the mean time to repair low. At FRIB the typical time for running Puppet to rebuild a node from scratch is ≈10 min.

The described Puppet modules can be used in enterprise environments with a Puppet master as well as for deploying services to stand-alone virtual machines on desktop computers (e. g. using Vagrant [20]).

CONCLUSION

FRIB has successfully implemented an infrastructure for CI and automatic deployment based on industry-proven open-source tools like Jenkins, Puppet and Ansible. The strategies and mechanisms in place have boosted deployment speed significantly and thus enabled agile development. The developed Puppet modules simplify deployment of key control-system components like EPICS IOCs, archivers and gateways by increasing abstraction while at the same time preserving flexibility. By providing good defaults they make it easy for engineers to follow best practices and facility standards. Firmware upgrades of embedded devices have been automated successfully on a large scale using Ansible.

The described approach and tools have been applied successfully over several years at FRIB. The built-in flexibility allows to support both the commissioning phase as well as accelerator operations efficiently. Over the last years larger updates like operating system upgrades of all control-system computers have been performed multiple times with minimal downtime. The strategy of testing changes on a development network before deploying them to production has helped significantly to improve the quality of FRIB's software before deploying to production. Overall, FRIB's CI and automatic-deployment infrastructure has proven invaluable in improving repeatability and thus increasing confidence in its control system.

REFERENCES

[1] FRIB, http://www.frib.msu.edu

[2] J. Wei *et al.*, "FRIB Accelerator: Design and Construction Status", in *Proc. 13th Int. Conf. on Heavy Ion Accelerator Technology (HIAT'15)*, Yokohama, Japan, Sep. 2015, paper MOM1I02, pp. 6–10.

[3] J. Wei *et al.*, "The FRIB SC-Linac - Installation and Phased Commissioning", in *Proc. 19th Int. Conf. RF Superconductivity (SRF'19)*, Dresden, Germany, Jun.-Jul. 2019, paper MOFAA3. doi:10.18429/JACoW-SRF2019-MOFAA3, to be published.

[4] M. G. Konrad, M. Davis, and E. Bernal Ruiz, "EPICS Support Module for Efficient UDP Communication with FPGAs", in *Proc. 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19)*, New York, NY, USA, Oct. 2019, paper MOPHA075.

[5] Git Distributed Version Control System, https://git-scm.com

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

[7] Jenkins Automation Server, https://jenkins.io

[8] Aptly Debian Repository Management Tool, https://www.aptly.info

[9] M. G. Konrad, D. G. Maxwell, and G. Shen, "Continuous Integration and Continuous Delivery at FRIB", in *Proc. 11th Int. Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'16)*, Campinas, Brazil, Oct. 2016, pp. 145–147. doi:10.18429/JACoW-PCAPAC2016-FRITPLC001

[10] Puppet Configuration Management Tool, https://puppet.com

[11] Ansible Configuration Management Tool, https://www.ansible.com/.

[12] Puppet Module for managing EPICS Soft IOCs https://forge.puppet.com/mark0n/epics_softioc

[13] procServ Process Server, https://github.com/ralphlange/procServ.

[14] Hiera Hierarchical Key/Value Data Lookup System, https://puppet.com/docs/puppet/latest/hiera.html

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOMPL006

[15] C. Dunn, “Designing Puppet – Roles and Profiles”, `https://www.craigdunn.org/2012/05/239/`.

[16] Puppet Module for managing repositories from version control systems `https://forge.puppet.com/puppetlabs/vcsrepo`

[17] K. Evans, “The EPICS Process Variable Gateway – Version 2”, in *Proc. 10th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS’05)*, Geneva, Switzerland, Oct. 2005, paper P1_033.

[18] Puppet Module for managing EPICS Gateways `https://forge.puppet.com/mark0n/epics_gateway`

[19] r10k Puppet Environment and Module Deployment Tool `https://github.com/puppetlabs/r10k`

[20] Vagrant Tool for building virtual Software Development Environments `https://www.vagrantup.com`