National Synchrotron Light Source II

Improvement of EPICS Software Deployment at NSLS-II

ICALEPCS 2019

U.S. DEPARTMENT OF ENERGY | Office of Science

BROOKHAVEN NATIONAL LABORATORY

Anton A. Derbenev, 8 October 2019

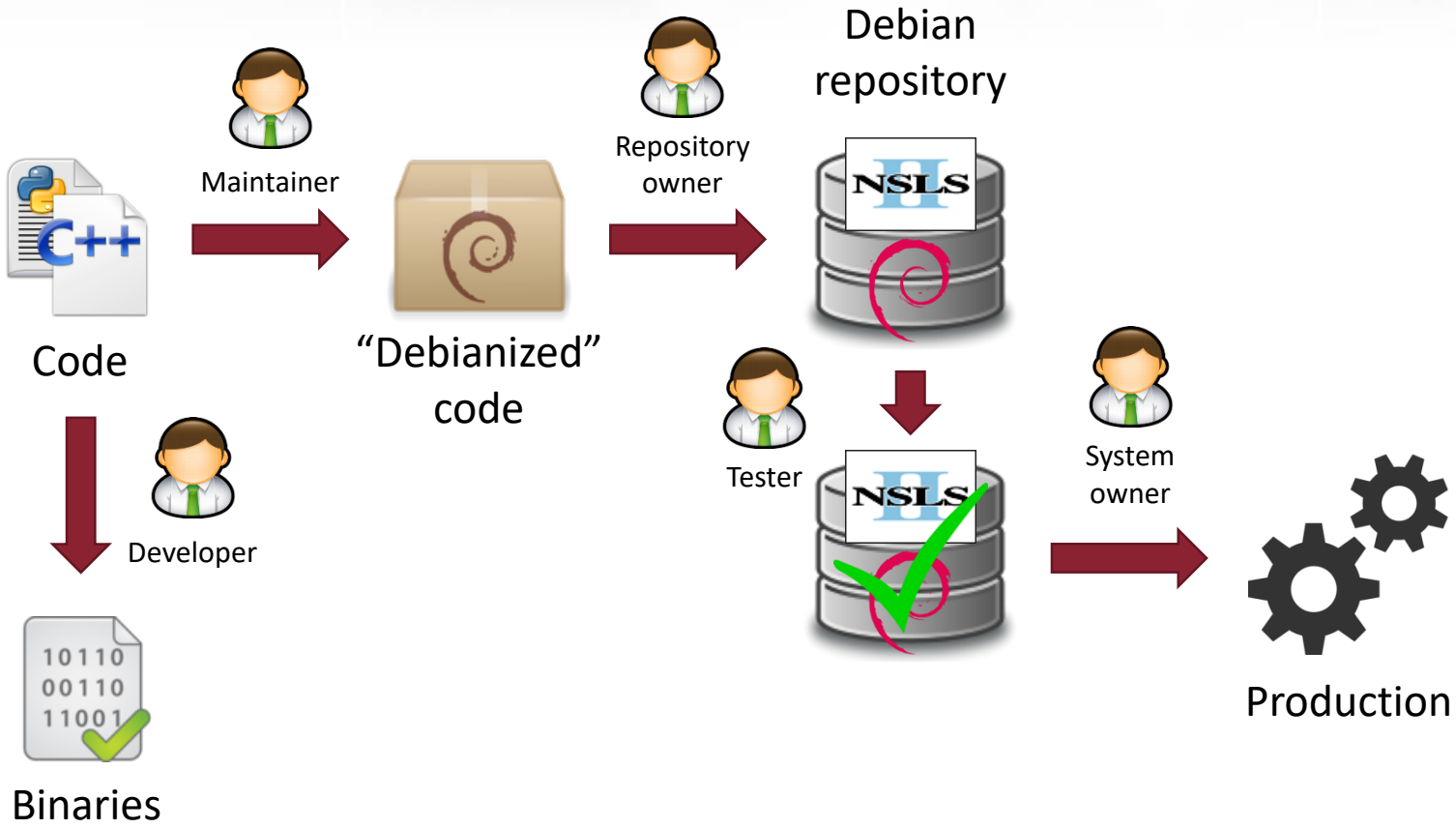
Presentation Plan

- Overview of NSLS-II software delivery
- Key considerations for delivery system design
- A note on solution support
- Utilized and explored approaches
- Future plans
- Conclusion

NSLS-II Software Delivery Overview

- NSLS-II uses Experimental Physics and Industrial Control System (EPICS)
- EPICS base, modules available as Debian packages from NSLS-II repo
  - Debian 8, 9, and 10 (untested) published
  - “Debianized” sources come from **epicsdeb** on GitHub (no EPICS 7 packaging)
  - Delivered to IOC systems by puppet
- Manual builds for newer software (EPICS 7, AreaDetector 3.7…)
- Using GitLab and Mercurial for code and configurations
- IOCs deployed manually, sysv-rc-softioc (procServ) for run control, logs…
- Services deployed manually, configurations come from central repository
- Central repository for OPIs, automatically synced to workstations

EPICS Base and Modules Delivery



- May look simple, but many aspects are covered: building, distribution, updates, versioning, etc.
- Doesn't answer all questions: coordination is required to ensure that environment is both stable and up to date

Key Considerations - I

- There are many ways to organize software delivery
- For evaluation, important characteristics were identified
- Considerations for the solution itself:
  - *Scalability* of the same approach across 28+ beamlines (isolated subnets?)
  - *Maintainability* of involved tools, services, infrastructure (docker engine?)
  - *Accessibility* for software developers and maintainers (trainings?)
  - *Accommodation of unique cases* to leave behind as few as possible (legacy?)
  - *Support of different OSs* to cover what's used in the system (Windows?)

Key Considerations - II

- Most requirements come from software specifics
- Considerations for applications which are deployed:
  - *Support of persistence* to retain local runtime changes (IOC autosave?)
  - *Release/staging function* for rollbacks on immediate failure (segfault on run?)
  - *Versioning* for mitigating instability (occasional crashes?)
  - *In-place production modifications* for urgent fixes (hardware IP change?)
  - *Preliminary testing* to not endanger the production (java update?)
  - *Replication of app instances* to reuse common code/binaries (same driver?)
  - *Ease of instance recovery* for severe failures (server malfunction?)

Special Note – Solution Support

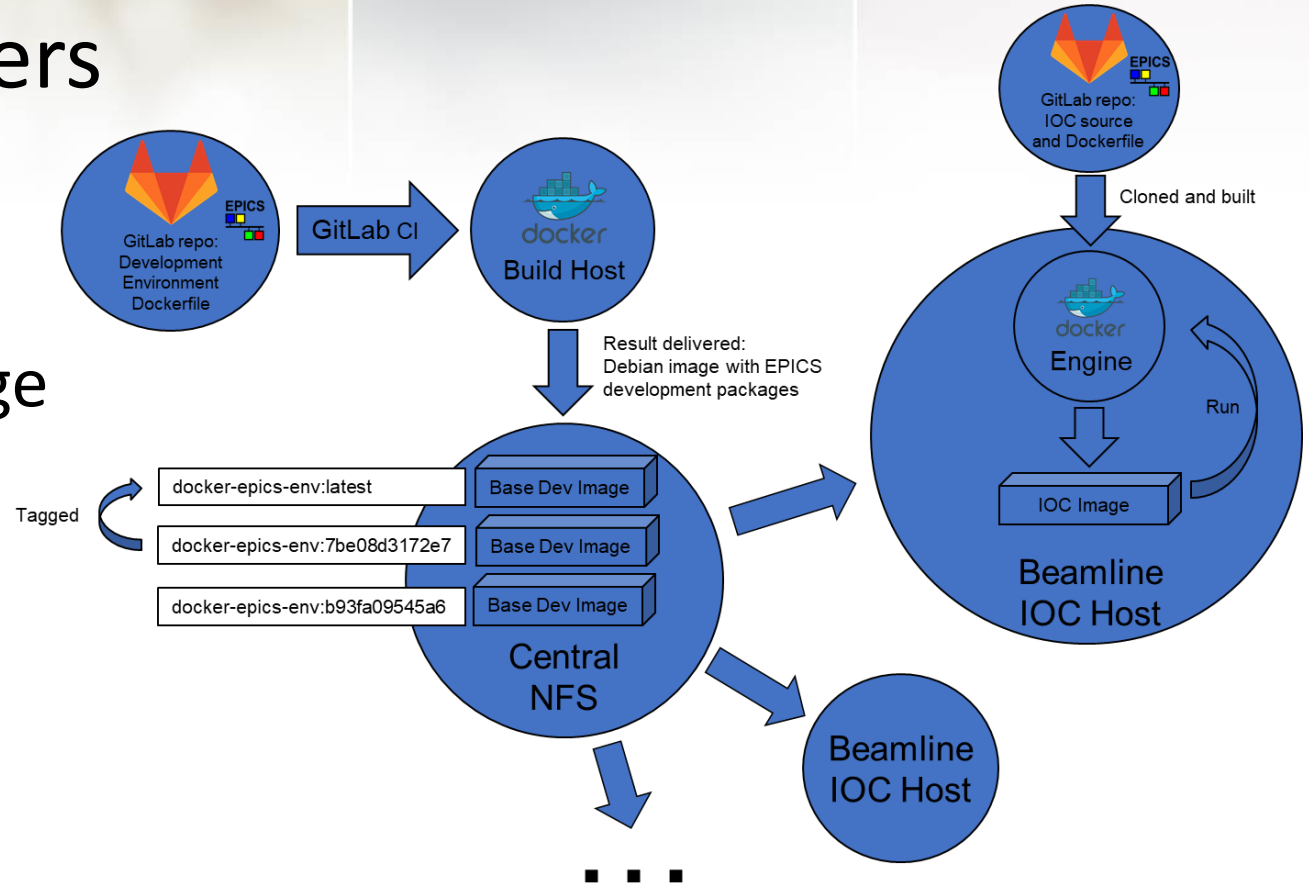
- With higher abstraction comes more sophisticated technology
- The delivery solution itself will evolve, not only applications
- Support needs are not one-time but a persistent liability:
  - Configuring and updating infrastructure, pipeline, tools
  - Introducing fixes and requested features
  - Performing hardware support and upgrade
  - Maintaining virtualization/containerization
  - Training and documentation upkeep

Utilized and Explored Approaches

- Manual delivery with version control
  - SSH to machine, clone the code and build in-place
- Manual delivery with binary bundles
  - Pre-built binaries put on NFS, location mounted on servers
- Orchestrated delivery on demand
  - Ansible-based toolkit to perform checkout, build, and deployment (SNACK)
- CI(/CD) pipelines
  - Jenkins for EPICS services test & build
- VMs/containers
  - VMs are commonality, Docker containers as a proof of concept

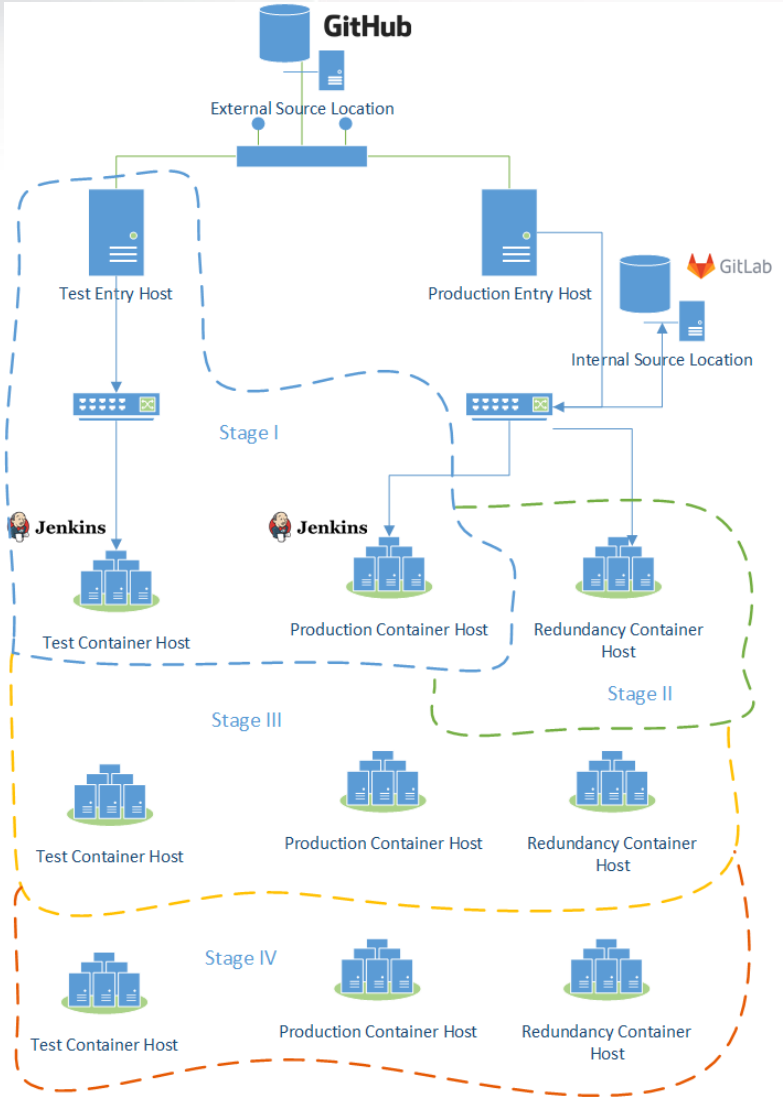
Sample – Docker Containers

- EPICS image based on Debian
- IOC image based on EPICS image
- Use volumes for persistence
- Use host networking for CA
- Docker features for tags, run control, logging, monitoring…
- A foundation to go higher!



Apps in Containers on VMs on Servers

- Switch to containers with known and exact configuration
  - Leave legacy behind
  - Easy to re-deploy and recover
  - Perform testing in isolation from production
- Use modern CI/CD tools
  - Modern solutions are easier in support
  - Users get a well-defined toolkit
  - Can abstract further (compose, swarm, Kubernetes…)
- De-couple from infrastructure considerations
  - Not influenced by external changes, e.g. dependencies, OS, and environment configuration
  - Agnostic to hardware



Future Plans

- Invest in hardware, software, and expertise for CI/CD in general
  - “Container host” machines are standing by for Docker etc. experiments
  - Planning to actually use Jenkins/GitLab/etc. for production delivery
- Specifically focus on deployment and update for tools and services
  - Deliver CS-Studio, Olog, Alarm, Phoebus…
  - Take dependencies under control
- Specifically focus on delivering detector software
  - AreaDetector building, binaries distribution, and IOCs deployment
  - Try to keep it abstract enough to apply to system IOCs in general
- Plan on future delivery for development environments – EPICS [7]
  - Put more effort and pressure to Debian packaging?
  - Use different OS or delivery mechanism?
  - Go really virtual/containers?
  - Combination of that?

Conclusion

- New ways of delivery are needed to keep up with systems evolution
  - Legacy pressure grows every day
- There are many considerations to be aware of, all with support needs
  - Should prioritize requirements and not over-design
- We are looking forward to bringing new tools and approaches
  - Investing in new hardware for CI/CD pipelines, and to try containers

Thanks for your attention!
Questions?