EPJ Web of Conferences **245**, 07029 (2020) https://doi.org/10.1051/epjconf/202024507029
CHEP 2019

High-availability on-site deployment to heterogeneous architectures for Project 8 and ADMX

Benjamin LaRoque[1]
(for the Project 8 and ADMX collaborations)

[1]Pacific Northwest National Lab

Abstract. Project 8 is applying a novel spectroscopy technique to make a precision measurement of the tritium beta-decay spectrum, resulting in either a measurement of or further constraint on the effective mass of the electron antineutrino. ADMX is operating an axion haloscope to scan the mass-coupling parameter space in search of dark matter axions. Both collaborations are executing medium-scale experiments, where stable operations last for three to nine months and the same system is used for development and testing between periods of operation. It is also increasingly common to use low-cost computing elements, such as the Raspberry Pi, to integrate computing and control with custom instrumentation and hardware. This leads to situations where it is necessary to support software deployment to heterogeneous architectures on rapid development cycles while maintaining high availability. Here we present the use of docker containers to standardize packaging and execution of control software for both experiments and the use of kubernetes for management and monitoring of container deployment in an active research and development environment. We also discuss the advantages over more traditional approaches employed by experiments at this scale, such as detached user execution or custom control shell scripts.

1 Introduction

Project 8 [1] and ADMX [2] are medium-sized physics collaborations, each with roughly ten member institutions and fewer than fifty collaborators. Projects at this scale face unique computing challenges stemming from the fact that team size and operational requirements are incompatible with a purely ad hoc approach, but limited resources make dedicated personnel and facilities infeasible.

In recent years, the cloud-computing community has produced a large number of open source tools for managing and deploying software applications, especially those executing in linux containers. While focused on cloud-based deployments, these same technologies can also be leveraged for on-premises deployments. Additionally, the pervasiveness of Raspberry Pies as low-cost and an easily accessible computing system provides an opportunity for standardization with great flexibility. Bringing these two solutions together provides a means of addressing many of the challenges mentioned above.

2 Computing challenges in medium-sized collaborations

As projects grow from very small teams, where ad hoc solutions are sufficient, up to a medium scale there are a number of computing patterns which become problematic. It is useful understand these when developing more deliberate practices.

One common pattern is that a physical server, often a desktop workstation, is initially identified for use developing the controls or acquisition related to a particular experimental or detector subsystem. Once development is completed, it is common to commission with the same hardware, at which point the development system has organically become the production system. As a result, the production system is often a combination of platforms with unmatched hardware, which are not optimal for production use and which may not be well utilized. At the same time, this pattern does not retain an available development platform for future enhancements, testing, or bug fixes.

Software development often mirrors the challenges from the hardware. During initial setup and testing, it is common for users to install dependencies interactively and incrementally as new capabilities are incorporated. Unless very well documented, it can become unclear exactly what is required to repeat the installation procedure in a new location. Further, in single-user systems it is common to run applications interactively, possibly using utilities like screen, tmux, or cron to manage multiple components. This pattern breaks down on multi-server and multi-user systems, where it can quickly become quite difficult to figure out exactly where something is running and how to interact with it.

3 Raspbery Pi as a solution for embedded systems

One solution to the hardware issues described above is the Raspberry Pi line of single-board computers [3]. They are both small and inexpensive, making them easy to integrate into custom electronics systems requiring remote monitoring and control. They also have a large open source community which maintains convenient libraries for programming GPIO pins directly, as well as communicating with attached sensors over a number of protocols including I^2C. An example of such an integrated system is shown in figure 1.

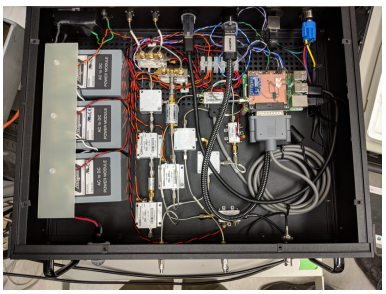

Figure 1. A custom analog electronics system for monitoring and tuning the total power level of radio frequency signals in Project 8. A Raspberry Pi (visible in the upper right) uses an I^2C compatible sensor to monitor power levels and provides digital control logic signals via GPIO to keep the system tuned within a desired range. The entire system is able to be deployed near the relevant hardware systems and the Raspberry Pi communicates over a local area network running the same control software as x86-based servers.

Because they are so cheap, there is not a barrier to purchasing identical boards to support different subsystems. However, the processors on these computers is based on the arm architecture, meaning that binaries compiled for the x86 architecture commonly found in servers and work stations are incompatible, a challenge which is addressed in the following sections.

4 Cloud computing tools: containerization, continuous integration, and orchestration

One fortunate outcome of the growth of cloud-based computing in the tech industry has been the availability of open source projects for deploying, managing, and monitoring applica-

tions. While those tools are often designed with an eye towards deploying highly responsive websites, they are also well suited to on-premises software deployments, especially when high up-time is important. In particular, it is worth discussing containerization, continuous integration, and orchestration as enabling approaches and technologies.

Linux containers provide a means of isolating the execution environment of an application and packaging that exact environment for reuse. There are open standards for defining and storing images [4], and for running those images [5], allowing both open and closed source projects to inter-operate. Because the process of building a functional container image involves installing both an application and its dependencies, the initial act of deploying software in a container creates a record of both how to install the software and the exact execution environment. It also means that containers can be run on servers without the need to first install application-specific dependencies on that server, and that different applications can coexist on a single physical server even if they have mutually-exclusive requirements.

A common way to create a container image is by writing a Dockerfile, which contains a sequence of instructions, and then building a new image which is the result of those instructions [6]. The image produced in such a process conforms to the open standard, but is typically specific to the CPU architecture on which it was built. The specifications also allow creation of an image manifest, which may contain references to multiple images and is typically used to group images which are functionally the same, but which are built to be compatible with different architectures. When running a container manifest, the local container runtime system will select the image which is compatible with the local system, allowing identical commands to be used on dissimilar hardware (for example on both x86 and arm based processors).

Continuous integration is an organizational principle whereby the responsibility for executing rote tasks in the software development cycle is removed from individuals and instead is triggered automatically in response to particular actions, such as merging or tagging in a software version control system. Not only does this reduce the amount of work that individual contributors are responsible for completing, but it also ensures that all required tasks are performed in a consistent and repeatable way. It also allows the set of expected tasks to be expanded without increasing the burden on individual contributors. There are a number of cloud-based providers providing computing resources for running continuous integration workflows against version controlled source code repositories, including Travis CI which is used in this work [7].

Finally, kubernetes is a sophisticated container orchestration system which coordinates the responsibilities of a cluster of servers [8]. It is designed such that the users declare a desired state, such as a set of containerized applications that should be running, and the system utilizes the available resources to both achieve and maintain that state. The user specifies those details which are critical to successful execution, such as required minimal system resources or available attached peripherals, but leaves other details, such as the specific server on which to run, unconstrained. This allows kubernetes to take corrective actions, such as rescheduling applications on different servers, depending on resource utilization and availability. The kubernetes infrastructure monitors the health of all applications and servers and is able to not only restart an individual program that crashes, but also to recover applications which may be running but in an unhealthy or unresponsive state, or to redistribute the entire workload from a server which becomes unavailable. The helm tool provides a standard way to separate the general steps for how an application is run from the details specific to a particular instances of that application [9]. It uses a templating system to combine those into specific desired state manifests which can be given to kubernetes.

5 Achieving higher availability of deployed software

The technologies described in the previous section provide a set of capabilities which the Project 8 and ADMX collaborations leverage to increase the availability and reliability of experimental control software. Prior to using these techniques, each collaboration maintained six to ten servers, including embedded Raspberry Pis, each was administered independently, had software installed locally on the system, and had applications run by users. In order to deploy a bug fix, an administrator would need to identify the server and possibly the user running the impacted application, stop the running application to install the new software, and then redeploy on that same system.

The newly enabled workflow looks quite different. After creating a new bug fix, the developer creates a new tag of the relevant software package in version control, triggering a set of tasks which are executed by Travis-CI. These include possibly running tests, building container images for both x86 and arm processors and a manifest to group them, and publishing the outputs to the hub.docker.com image registry. An example summary of automated tasks is is shown in figure 2, for the case of new features being merged into the develop branch of a particular source code repository. This eliminates the operational disruption required for building and installing new software versions on production servers.

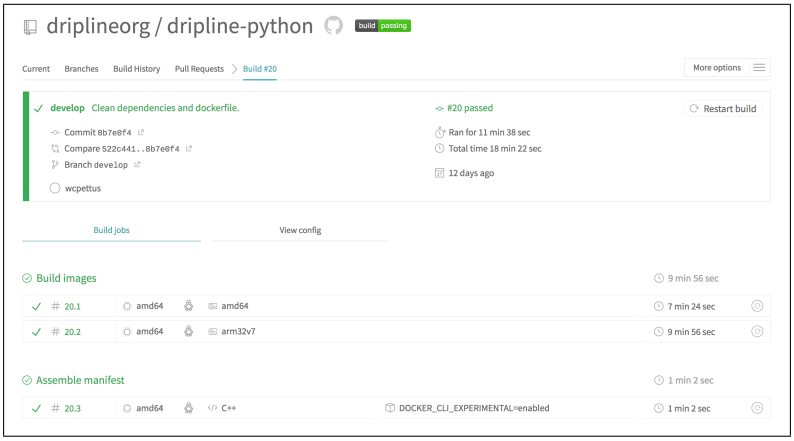

Figure 2. A screen capture of the job summary from travis-ci.com. In this case, a code merge into the develop branch has resulted in building new container images for both x86 and arm architectures, the latter being achieved on native x86 hardware by means of emulation. Those images, along with a manifest referencing both of them, are pushed to the registry hosted at hub.docker.com for storage and distribution to production servers at the experimental site.

Helm charts are used to define how controls software are deployed, including templating the application version that should be used [9]. By updating the relevant helm template and then updating the desired state in kubernetes, the new application version is able to be deployed either to individual instances, or to all applications across all servers. The administrator is able to do this from a central control interface, rather than needing to personally upgrade each server, and the time to upgrade or downgrade between versions is usually dominated by the time it takes for the application to halt and start, rather than time in human effort making the changes.

In addition to centralizing life-cycle management, it is possible to inspect the health and version of all deployed applications, including monitoring console logging, from that same central interface. Further, the kubernetes system supports sophisticated monitoring of the

health of applications, allowing automatic detection of problems which may not result in a crash, such as entering an undesirable state or becoming unresponsive. Just like an actual crash, these undesired states can trigger automatic recovery action. This allows the overall system to be returned to a healthy state, even at times when there is not a person monitoring, and much faster than when active human intervention is required.

6 Results and Conclusions

Project 8 and ADMX are medium-scale experimental physics collaborations requiring more deliberate software management than is common on smaller projects, but without the dedicated personnel of much larger collaborations. In the past, software has been installed in an ad hoc fashion on several independently administered servers. Poor application health was only recoverable by human intervention, and this happened as a result of human monitoring and noticing problems, resulting in substantial down time. Further, installing software fixes had to be done directly on the production system requiring down time for that process.

By embracing the tools developed by the cloud computing community, both projects have seen a significant increase in control software up-time. Many issues, especially sporadic and non-crashing problems are now able to be automatically detected and recovered, significantly reducing outages or periods of low-quality operation. Furthermore, automated build and deployment processes mean that fixes and upgrades are able to be deployed with minimal impact to ongoing operation, rather than requiring prolonged periods of interruption.

Acknowledgements

Project 8, including a portion of this work, is supported by the US DOE Office of Nuclear Physics, the US NSF, the PRISMA+ Cluster of Excellence at the University of Mainz, and internal investments at all collaborating institutions.

The ADMX collaboration gratefully acknowledges support from the US Dept. of Energy, High Energy Physics DE-SC0011665, DE-SC0010280, & DE-AC52-07NA27344. ADMX also receives support from the LLNL and PNNL LDRD programs and R&D support from the Heising-Simons institute.

References

[1] A. Ashtari Esfahani et al. (Project 8 Collaboration), "Determining the neutrino mass with Cyclotron Radiation Emission Spectroscopy - Project 8", *J. Phys. G.* **44** (30 March 2017). (doi:https://doi.org/10.1088/1361-6471/aa5b4f)

[2] T. Braine et al. (ADMX Collaboration), "Extended search for the invisible axion with the axion dark matter experiment" *Phys. Rev. Lett.* **124**, 101303 (11 March 2020). (doi:https://doi.org/10.1103/PhysRevLett.124.101303)

[3] Raspberry Pi Foundation, "Raspberry Pi Documentation" https://www.raspberrypi.org/documentation/

[4] Open Container Initiative, The Linux Foundation, "Image Format Specification" https://github.com/opencontainers/image-spec/blob/master/spec.md.

[5] Open Container Initiative, The Linux Foundation, "Open Container Initiative Runtime Specification" https://github.com/opencontainers/runtime-spec/blob/master/spec.md.

[6] Docker Inc., "Docker Documentation" https://docs.docker.com/.

[7] Travis CI GmbH, "Travis CI User Documentation" https://docs.travis-ci.com/.

[8] The Kubernetes Authors, "Kubernetes Documentation" https://kubernetes.io/docs/home/.

[9] "Helm Documentation" https://helm.sh/docs/.