ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA045

PACKAGING AND HIGH AVAILABILITY FOR DISTRIBUTED CONTROL SYSTEMS*

Mauricio Araya†, Leonardo Pizarro and Horst von Brand
Universidad Técnica Federico Santa María, Valparaíso, Chile

Abstract

The ALMA Common Software (ACS) is a distributed framework used for control of astronomical observatories, which is built and deployed using roughly the same tools available at its design stage. Due to a shallow and rigid dependency management, the strong modularity principle of the framework cannot be exploited for packaging, installation and deployment. Moreover, life-cycle control of its components does not comply with standardized system-based mechanisms. These problems are shared by other instrument-based distributed systems. The new high-availability requirements of modern projects, such as the Cherenkov Telescope Array, tend to be implemented as new software features due to these problems, rather than using off-the-shelf and well-tested platform-based technologies. We present a general solution for high availability strongly-based on system services and proper packaging. We use RPM Packaging, oVirt and Docker as the infrastructure managers, Pacemaker as the software resource orchestrator and life-cycle process control through Systemd. A prototype for ACS was developed to handle its services and containers.

INTRODUCTION

The ALMA Common Software (ACS) was designed on the early 2000s, with the objective of providing a basic control and communication codebase for the ALMA observatory [1]. The technologies used for its construction were probably the best technical choices back then, but not all of them have aged well. For instance, ACS relies on the CORBA standard, which regulates almost every aspect of a distributed system using RPC. This was the main research topic in distributed systems at the time, but currently is in its way to obsolescence and is known to have a low performance among similar tools [2]. On the other hand, ACS is still a very powerful and tailored solution for complex array control, with a solid architecture and modular development that allows reusing the code for new projects like the Cherenkov Telescope Array [3]. This imposes new challenges for the framework not only at the software development level (e.g., gradually replacing CORBA interfaces), but also at the construction and deployment levels. In this paper we address this last problem to increase ACS robustness, flexibility and availability using generic solutions from modern, large and distributed software systems.

* Work supported by Centro Científico Tecnológico de Valparaíso (CONICYT FB-0821) and Advanced Center for Electrical and Electronic Engineering (CONICYT FB-0008)

† mauricio.araya@usm.cl

ACS PACKAGING AND DEPLOYMENT

The ACS framework is currently built by using its own construction system based on `make` [4]. It consists in a master `Makefile`, which enters each software module and tool searching for a standard directory structure that have other `Makefiles`, which needs a common base file called `acsMakefile` to operate. This recursive `Makefile` paradigm is widely considered a bad practice, because dependencies are hard to maintain and the construction process is difficult to track and resume. Also, it requires to previously compile other tools known as External Products, with specific versions and patches different from the ones offered by the operating system distribution. Moreover, the construction paths and configurations are setup through environment variables handled by the `bash_profile.acs` file that must be sourced. All this software is finally packaged as a very large monolithic tarball meant to be uncompressed in a very specific OS version.

Besides the direct difficulties at the construction stage, the software distribution produce a steep learning curve for its installation and environment setup, a non-modular deployment of the software packages, a replace-all update method, and no separation between runtime, configuration, development, example and testing packages. A few attempts to update the ACS construction and deployment system have been tested in the past[5]. Among the tools used, two of them stand out: RPM [6] and Docker [7].

RPM Packaging of ACS

RPM is the package manager of multiple operative systems and handles the building, installing, updating and dependency restrictions of the software. It has existed since 1997, and it's core file contains the step-by-step instructions to build the software once, and deploy the binaries and other products ready to use, many times. It also allows to distribute updates with controlled dependencies and versions. One of the resulting products, the Source RPM, allows migration among different architectures almost transparently.

For ACS, we address the packaging problem in a pragmatic fashion, assuming a simple tree-like dependency model, where at the leafs were the components that were dependency-free but needed by the parent node. The ACS-base is divided in three groups: *External Tools*, *Tools* and *Kit*, with External Tools as the lowest group in the tree. With the off-the-shelf notion, each software was searched for in the repositories of Fedora (21 to 26), CentOS and the special repositories SCL, SIG and OpenSUSE, searching for the nearest match regarding the version shipped with ACS.

To group these packages in one place and make them available, an ACS package repository was created at `http://repo.csrg.cl`. Instructions for it usage are available at the project's main page [1]. Source RPMs can also be found if the user wants to port them to another architecture. Also, there are directories grouping the different versions of ACS and the software those versions depend on.

Then, three large RPM packages were built:

- ExtProds: Grouping all RPMs found for replacement and were all the remaining Python software that hadn't an RPM would be installed through Pip. Also the management of each environment variable of the software grouped in this RPM, is done by using files in the `profile.d` directory, partially replacing the use of the bash_profile.acs file.
- ExtJars: Jar libraries not found in any RPM.
- ACE+TAO-ACS: A special RPM was created for ACE+TAO, a software that implements the CORBA standard for C++, because ACS special patches were applied.

These RPMs group all of the software needed to compile ACS source code. The remaining step was to build ACS core from within its own RPM, however this is still not completely reliable, due to pending transformations of the `acsMakefile` into a standard `Makefile`. Regardless of that, the products of a manual compilation of ACS where packaged inside an RPM. This elements where ACSSW and acsdata. The software dependencies or script needed by ACSSW and acsdata, and the respective environment variables, where packaged the same way as we did with ExtProds.

The main improvement is the reduction in time and complexity involving the installation of ACS, avoiding the very time consuming compilation or changing the local OS version to support it. This allows deploying ACS in matter of minutes, handling any update cleanly. Lastly, any architectural differences among hardware — a situation not uncommon to astronomy projects — could save migration time and effort, with the existence of Source RPMs.

ACS Dockerfile

The relatively recently released Docker system (2013) focuses on running isolated processes without the overhead incurred by a complete virtual machine. It uses cgroups, namespace and SELinux as its basic components. The *Dockerfile* is the core building file that defines a container, allowing the resulting container to be shared using the DockerHub platform.

Therefore, using the manual compilation of ACS, a complementary solution was built using Docker containers, which is currently published at the Docker Hub[2], which allows *pull* ACS into a docker platform. This is only available for the ACS 2016.10 version, but an unstable version of ACS 2017.02 can be found at the CSRG's Git Repository[3].

SYSTEM-BASED HIGH-AVAILABILITY FOR ACS

In runtime, ACS is a collection of services for communication (alarm, error, notification channels, etc) and code execution units called components running in different programming language containers[4]. A basic interaction description can be observed on Figure 1, where in case of a failure of any of the components, manual intervention should be needed, involving sometimes the disruption of a ongoing observation. Also, failure or malfunction of hardware or operative system processes would cause interruptions in the data flow.

The lack of a fault-tolerance framework was one of the first weaknesses of the ACS middleware that the CTA consortium has detected. The high-availability support for keeping track and control of the components life-cycle, is being addressed by CTA's Observation Execution System (OES) through `Supervision Trees` (ST). The ST is an idea that can be found in Erlang [8] and consists in the creation of a supervisor process per each running process, known as worker. This model form a tree, starting at the top from the first supervisor process, linked to its worker. Once new workers are created, new supervisor also start, with a tree-graph linking workers to workers, supervisors to workers and supervisors to supervisors. This allows a cascade control of any malfunction. However, it also requires the worker to be "aware" of the existence of the supervisor, adding complexity to the code. The main objective of ST is to handle malfunctioning processes, stopping and/or replacing them, without affecting the rest of the system, in the specific way each component has to be managed.

Before CTA, the high availability (HA) features of distributed systems, such as fault tolerance and life-cycle management, have been historically approached in ACS as software capabilities. This was the only option at the design stage of the framework, because these features were not widely available as system capabilities back then. Nowadays, these features are increasingly common requirements on all sorts of software systems, which now are commonly distributed and diverse. Therefore, solutions for these requirements can be found now as commodity system's capabilities.

There are several off-the-shelf tools to manage HA, with a common pattern of being closer to the Operating System (OS) than the software itself. Between them we found Pacemaker (PCS) [9], virtual machines (VM) and Linux container Orchestrators like the oVirt Manager, Rancher and Kubernetes. As they orchestrate the resources for each VM or Linux container, they need to perform monitoring over the VM/Containers, the underlying hosts and their re-

[1] `https://csrg-utfsm.github.io/`
[2] `https://hub.docker.com/r/leoxdxp/acs/`
[3] `https://github.com/csrg-utfsm/acs-docker`
[4] These are ACS containers, please do not confuse them with Docker containers

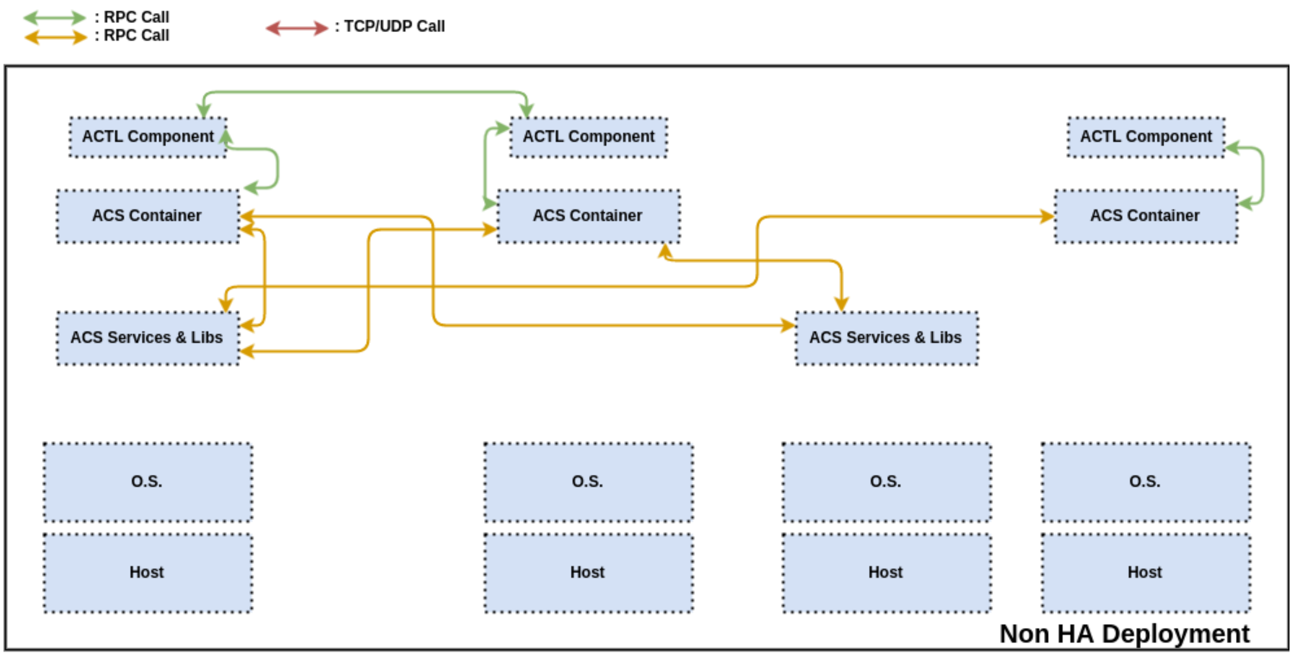

Figure 1: ACS currently, without HA.

sources/availability. Containers are preferred over VMs due to the overhead that VMs cost as they use virtualized hardware. Linux container orchestrators can also arrange a specific start/stop order between different containers, depending on the user configuration.

A HA Proof of Concept for ACS Containers

The fault-tolerance feature under implementation (ST) will work at the components level, not considering the container's life-cycle management. If we understand the container as a process, this challenge is not uncommon, and it can be addressed by tools such as Pacemaker, where each element taking part in a system, such as the network, disks, configurations and processes are handled as a resource. The start/stop execution order can be managed using the Open Cluster Framework[10], to configure those and other instructions (polling/monitoring frequency), and/or using the system daemons. The later is a common extra step, because it allows more detailed monitoring and handling over start/stop sequences.

Using Pacemaker, or at least parts of the HA software such as Heartbeat, enables the use of a widely used tool for a problem that spans along multiple areas, allowing the development of OES to be solely focused on features rather than concerning with corner-cases and/or failure management. A mixed solution, using Rancher to control infrastructure level HA, and Pacemaker to monitor/poll the software that ACS components run could set the base to use ACS without mayor changes in a HA environment, and include new HA tools in the future as shown in Figure 2.

Towards a better understanding of the implications of this mixed solution, we compare the current interactions inside ACS's Services, Containers and Components with our proposal. One of the key problems for HA is the randomness of ports used by ACS components when they communicate with each other. The ports can be fixed to defined segments or narrowed even to one port, by setting configurations on ACS Containers and/or ACS's Configuration Database (CDB). The disadvantage of this would be the necessary coordination to define in each of the dozens of ACS components, which ACS component should use which port. The actual solution for this has been disabling any firewall where ACS is used. Even though, security does not usually hold a high importance in scientific-project context, the use of basic firewalls is a basic good standard of systems administration.

As the operating system perform constant monitoring and polling of all the processes, orchestrating and limiting resources through SysV or Systemd is a commodity, with the only restriction of doing so locally. Therefore, the HA for a distributed set of servers needs to be handled then by a resource manager. In order to provide a first level of redundancy, Rancher was chosen as the infrastructure orchestrator, considering the use of Docker Containers for non-data sensitive ACS elements. For data sensitive ACS elements, as the relational database CDB, the use of virtual machines is recommended, and therefore oVirt Manager was selected.

As an example, the rancher orchestrator can be hosted in a VM in order to provide HA for the Docker container orchestrator through oVirt, simplifying the HA of Rancher. This is done because for a true HA, every component of the solution needs to be highly available. In general, each physical server can be an oVirt and a Docker host at the same time as shown in Figure 3.

The second level of HA, this is, the process-level distributed management can be done by using Pacemaker. The

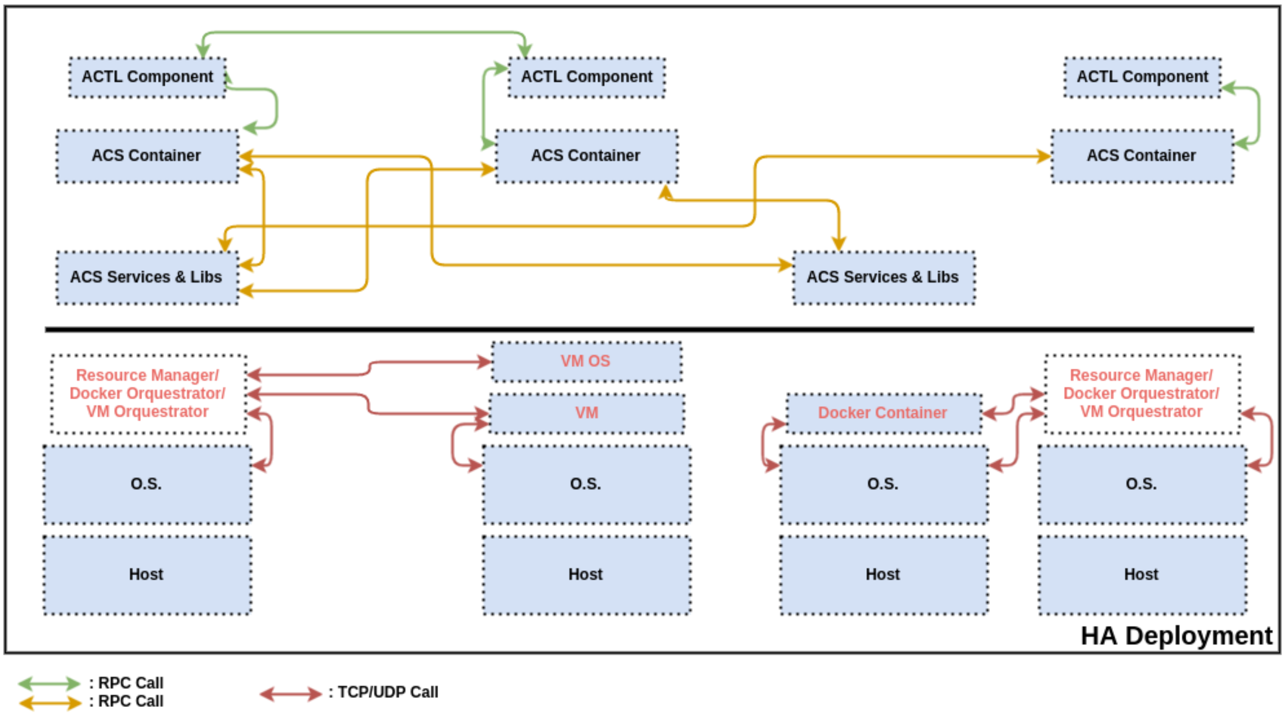

Figure 2: ACS with HA

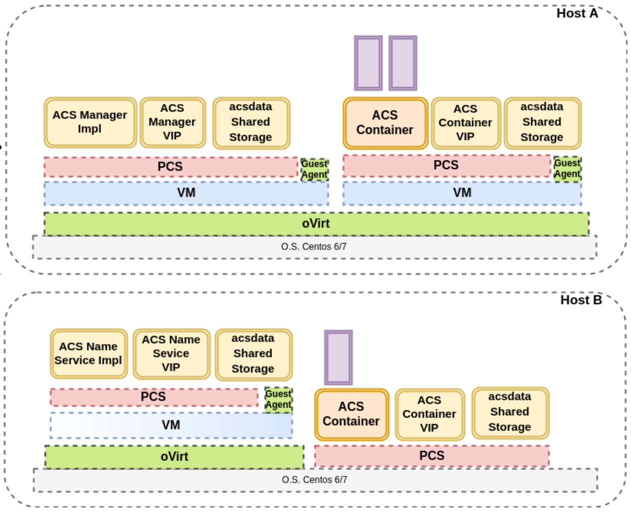

Figure 3: Proof of Concept model for ACS with HA.

configuration of ACS services and ACS Containers, the startup, shutdown and restart sequence, the monitoring period between heartbeats and other details can be done via Pacemakers web interface, which also shows the status of the resources, or via the command line. This level is reinforced with each ACS Service and ACS Container having a Systemd or SysV daemon configuration file. For this proof of concept, two Systemd daemons were created, and they are available within the RPMs. As seen on Figure 2, the coordination among orchestrators and hosts is done over TCP, sometimes using JSON for data serialization, thus involving little load on the network. On the other hand, ACS's communication is done over RPC, with the extra overhead that this implies. Therefore, any HA solution from inside ACS will also have to consider RPC's overhead.

Even if the proposed solution is used only for ACS Services and ACS Containers, it would provide a huge improvement in error management and allow to apply updates without jeopardizing the overall system.

CONCLUSION

Observatories, such as ALMA and CTA, need a building system that can provide continuous software updates without disrupting the production environment and be as cheap in time and human resource as possible. The use of tools such as RPM and/or docker containers to automate deployment, must be considered from the start of the projects in order to reduce the number of man-hours dedicated to this task, specially regarding software testings, were the difference between the testing environment and the production environment, have direct consequences and can delay implementation of new features. In relation to the above, a long term and robust building system arises as the cornerstone for projects that will continuously function for several decades with possible expansions in its resources and systems.

As solutions for High Availability within distributed systems is not only a topic related solely to observatories, the proposed idea can be applied without mayor changes to any other science project. This is specially the case for those that due to the difference between the time of their conception and their implementation, or by other reasons, didn't consider HA or built it in a higher level of the software stack. Also, distributed systems have had a paradigm integration with

HA, providing a new and interesting mix, where software development and system administration areas have merged, changing the perspective on HA and distributed systems.

ACKNOWLEDGMENT

This research was possible due to CONICYT-Chile fundings, specifically through the Basal Project FB-0821 and Basal Project FB-0008.

REFERENCES

[1] G. Chiozzi *et al.*, "The ALMA common software: a developer-friendly CORBA-based framework," in *Advanced Software, Control, and Communication Systems for Astronomy*, H. Lewis and G. Raffi, Eds., vol. 5496, Sep. 2004, pp. 205–218.

[2] A. Dworak, P. Charrue, F. Ehm, W. Sliwinski, and M. Sobczak, "Middleware Trends And Market Leaders 2011," *Conf. Proc.*, vol. C111010, no. CERN-ATS-2011-196, 4p, Oct. 2011.

[3] I. Oya *et al.*, "The software architecture to control the cherenkov telescope array," *Proc.SPIE*, vol. 9913, p. 15, 2016.

[4] S. I. Feldman, "Make – a programm for maintaining computer programs," *Software: Practice & Experience*, vol. 9, no. 4, pp. 255–265, Apr. 1979. DOI: 10.1002/spe.4380090402.

[5] M. Mora *et al.*, "Integrating a university team in the ALMA software development process: a successful model for distributed collaborations," in *Software and Cyberinfrastructure for Astronomy*, vol. 7740, Jul. 2010, p. 77403I. DOI: 10.1117/12.857832.

[6] E. Foster-Johnson, *Red Hat RPM Guide*. Red Hat Press, 2002.

[7] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014, ISSN: 1075-3583. http://dl.acm.org/citation.cfm?id=2600239.2600241

[8] M. Logan, E. Merrit, and Carlsson, *Erlang and OTP in Action*. Manning Publications, 2010.

[9] L. Perkov, N. Pavković, and J. Petrović, "High-availability using open source software," in *2011 Proceedings of the 34th International Convention MIPRO*, May 2011, pp. 167–170.

[10] A. Santos, F. Almeida, and V. Blanco, "The OpenCF: An open source computational framework based on web services technologies," in *International Conference on Parallel Processing and Applied Mathematics*, ser. Lecture Notes in Computer Science, vol. 4967, Springer, 2007, pp. 788–797.