HOW TO BUILD AND MAINTAIN A DEVELOPMENT ENVIRONMENT FOR THE DEVELOPMENT OF CONTROLS SOFTWARE APPLICATIONS: AN EXAMPLE OF "INFRASTRUCTURE AS CODE" WITHIN THE PHYSICS ACCELERATOR COMMUNITY

L. Fernandez, R. Andersson, H. Hagenrud, T. Korhonen, R. Mudingay, European Spallation Source, ERIC, Lund, Sweden
B. Zupanc, Cosylab, Ljubljana, Slovenia

Abstract

The Integrated Control System Division (ICS) at the European Spallation Source [1] (ESS) has the mandate to provide all the needed tools to ESS staff, in-kind contributors and consultants spread all over Europe, in order for them to build software for the commissioning and operation of the ESS. This includes EPICS applications, scripting environments, physics simulators and commissioning tools among others. ICS needs to provide support for new releases of the different software components, guaranteeing that the development environment of all the users can be properly updated. ICS needs to guarantee as well that environments can be reproducible and at the same time give the flexibility to users to own and customize their environments. ICS used a new virtualization technology (Vagrant [2]) and a new configuration management system (Ansible [3]) to provide a cutting edge development environment where all the software infrastructure can be described as code and properly stored in a version control system, tagged, tested, versioned and rollbacked if needed.

INTRODUCTION

The construction of the ESS brings an enormous challenge. ESS will be built with the collaboration of many different countries in Europe. And that is not only true for material and hardware, but also for software. Each of these countries, also known as in-kind contributors, will participate in the development of particular components of the accelerator. Regarding software, in-kind contributors will design, develop and test software in their home countries. Later on, that software will be integrated at the ESS. This brings an extraordinary complexity to the software integration of those elements. Software components, frameworks, versions, dependencies, toolkits must be agreed upon by all parties in order to guarantee that all the different pieces will be properly integrated.

After a careful study, the Software Group of the ICS decided to provide all the tools that are needed to develop software for the control systems. ICS guarantees that the tools provided to in-kind contributors would be the ones supported and used in commissioning and operation of the ESS machine. ICS provides such environment in different formats: a physical machine installation and a virtual machine. In both cases the system contains all the libraries, software components and frameworks that are needed to develop software for the ESS Controls Systems.

In order to configure the physical machine or the virtual machine, ICS uses the same source of information: a set of Ansible playbooks describing the configuration of the ICS development environment. The virtualization platform that was chosen was VirtualBox [4] but with Vagrant running on top of it. Vagrant extremely eases the installation, configuration and distribution of the virtual machines (Fig. 1).

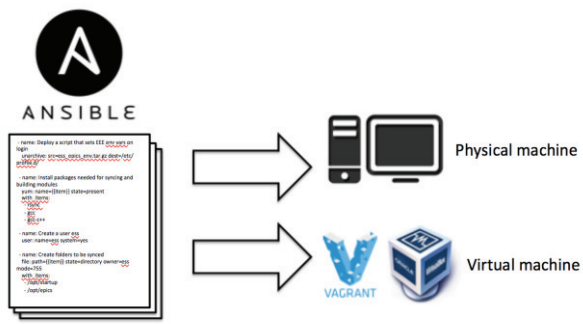

Figure 1: Ansible is used for the configuration of virtual and physical machines.

ANSIBLE

Ansible is an open source configuration management system used to automate the configuration of IT systems, deploy applications and provision software in new and existing systems. Ansible can also be used for software orchestration, where not only the configuration of the systems is important but also the order in which they have to be configured.

The basic unit of Ansible is the playbook. We can see the Ansible playbook as a script where we can describe the configuration of our system.

The main goal of having Ansible is to be able to describe all the infrastructure software of the ICS as code. All the configuration of ICS servers, ICS development machines, all ICS machines in general are described in Ansible playbooks.

There are some other configuration management systems used successfully by the IT, such as: Salt, Puppet or Chef. But, Ansible was chosen among all of them for the following reasons:

ISBN 978-3-95450-147-2

- Extremely low learning curve. It is very easy to write Ansible playbooks. They are written in almost natural language what it is makes very easy the development and the maintenance by non experts. In some cases ICS needs its in-kind contributors to manage and perform small modifications in the Ansible playbooks. The fact that those playbooks are easy to read, write and modify was an important reason to choose Ansible
- Ansible is agent-less. This means that Ansible does not need agents running in the target machines as daemons. Ansible is known for being a push system, where the configuration is pushed from a centralized place into the target machines or nodes. This simplifies the installation and configuration of systems as the only pre-requisites to be fulfilled in the target machines are: ssh and python (2.4 or later). This characteristic was very important in the process of choosing Ansible. In cases when target machines are provided and hosted by in-kind contributors, ICS cannot control them. Being agent-less ICS will not be worried for configuring properly any agent or deamon in those target machines. It simplifies the maintenance and installation of the target machines or nodes.

Having a configuration management system as Ansible allows ICS to describe all the software infrastructure as code (IaC) [5] and to keep such configuration in a version control system such as Git [6]. Also this lets us version the current configuration of ICS systems and therefore keep control of the updates, and even perform rollbacks. ICS uses the Atlassian cloud implementation of Git called Bitbucket [7] (Fig. 2).

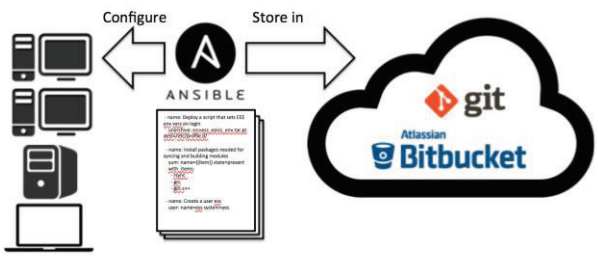

Figure 2: Ansible playbooks used to configure ICS systems are stored and versioned in Git.

VAGRANT

Vagrant provides an easy way to create virtual environments. Vagrant lays on top of virtualization providers such as VirtualBox, VMWare, AWS and others. At ICS Vagrant is used on top of the VirtualBox provider. Using Vagrant, it is possible to configure a virtual environment describing such configuration in a file that is called Vagrantfile. This Vagrantfile configuration file provides all the setup needed to configure the virtual machine properly, including the reference to Ansible playbooks, which install the appropriate versioned software. This way of configuring the virtualization environment spares the end-user of doing the configuration by himself. That also brings a huge benefit when automating the creation and management of virtual machines. On top of all that, being able to describe the virtual machine configuration in a file allows us to keep such configuration in version control. This means that the configuration of the virtual environment can be properly versioned, and previous versions can be restored easily.

In Vagrant the user normally manages the virtual environment with a very simple command line interface. Typical commands are: “vagrant up” to start the virtual machine, “vagrant halt” to stop the virtual machine or “vagrant reload” to reload the configuration from the Vagrantfile (Fig. 3).

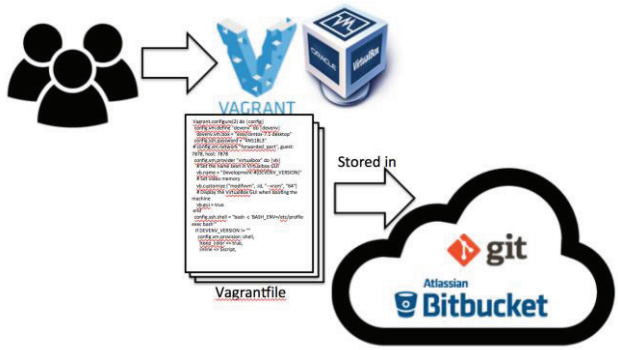

Figure 3: Vagrant uses the playbooks stored in Git to populate the virtual machines.

A key benefit of using virtual machines for the development environment is that they can be treated as disposable environments. If for any reason the environment gets corrupted the virtual machine can be destroyed and a new one created within seconds. That gives the user freedom to work with the environment without being worried to break it. For this to work, it is very important that user’s data and code are kept outside the virtual machine. Vagrant and VirtualBox provide the concept of shared folders. Using shared folders the users can maintain source code and important data in their host environment whereas the code can be also accessible from the virtual machine. Data can also be shared among virtual machines using shared folders.

The user is always able to install new packages in the virtual machine, although we encourage keeping it as neater as possible. In most of the cases, the users just want to modify the virtual machine in order to install their preferred IDE. In that case we encourage users to install the IDE in their host (laptops or desktop computers) and to use share folders to work on the code that later on can be run in the virtual machine. This implies that the users can use their original development environment, with no modifications, and still use any editor or tool.

CONITINUOUS INTEGRATION AND CONTINUOUS TESTING

One big benefit of keeping the entire software infrastructure described as code and stored in a version control system such as Git, is that many activities can be easily automated avoiding manual error prone processes. One of these activities is the testing of new changes. At the ICS an automated system for verification of software configurations has been put in place using Jenkins as the main framework.

Each time there is a change in any of the Ansible playbooks and once that change is committed to the central Git repository, Jenkins [8] triggers a job that verifies that the new playbook runs successfully. Jenkins also verifies that the rest of playbooks that were not modified were not affected by the new changes.

When running tests it is important to setup an environment that is not polluted by previous tests or previous environments or setups. The best practice in these cases is to start a new environment from scratch where Jenkins could test the Ansible playbooks. In order to do that, ICS uses Docker [9] containers. Jenkins spins up a Docker container on demand. Once the Docker container is created Jenkins provisions a fresh environment in the container and verifies that the Ansible playbooks run successfully, provisioning the new container (Fig. 4).

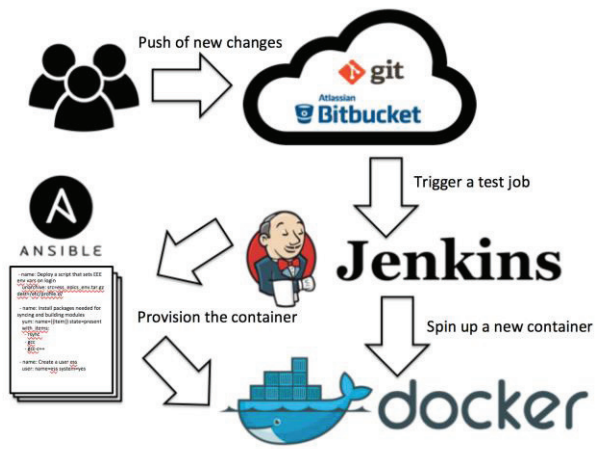

Figure 4: Continuous Integration infrastructure for Ansible playbooks.

Thanks to this current setup, ICS has been able to identify and fix numerous problems before the Ansible playbooks were deployed into production. The use of Jenkins and Docker containers has been a breakthrough in the quality assurance of the ICS software infrastructure.

SOFTWARE COMPONENTS PROVIDED BY THE ICS DEVELOPMENT ENVIRONMENT

The ICS Development Environment needs to provide all the tools needed to develop software for the ESS Control Systems. The following components have been already identified and they are part of the software suite provided by the ICS Development Environment:

- EPICS development environment. This involves EPICS base and modules needed for the development of EPICS low level software.
- CS-Studio [10]. Eclipse-based collection of tools to monitor and operate accelerators. One of the key components of this suite is BOY. BOY is a GUI builder for EPICS applications.
- OpenXAL [11]. It is an open source environment for creating accelerator physics applications. OpenXAL contains a physics model of the ESS machine that lets the developer simulate the ESS accelerator and to communicate at the same time with the physical devices of the machine.
- Jupyter Notebook [12]. Open source interactive data and scientific environment. Jupyter Notebook is currently used by ESS to provide a scripting environment for physics applications. Currently ICS provides support for Python, Julia and R. It is possible to run simulations in OpenXAL and Mantid [13].

CONCLUSION

ICS has successfully built a stable development environment that can be used by in-house personnel, as well as off-site in-kind contributors and consultants. The use of a single development environment will assure that everybody independently of the location will be able to work in the same environment, using the same components, the same versions and the same dependencies. This will facilitate the later integration of all the software developed in different places, once they are installed at the ESS.

REFERENCES

[1] European Spallation Source, https://europeanspallationsource.se/
[2] Vagrant, https://www.vagrantup.com/
[3] Ansible, https://www.ansible.com/
[4] Virtual box, https://www.virtualbox.org/
[5] C. Riley, "Version your infrastructure", 12 Nov. 2015, http://devops.com/2015/11/12/version-your-infrastructure/
[6] Git: https://git-scm.com/
[7] Bitbucket, https://confluence.atlassian.com/bitbucket
[8] Jenkins, https://jenkins.io/
[9] Docker, https://www.docker.com/
[10] CS-Studio, http://controlsystemstudio.org/
[11] OpenXAL, https://github.com/openxal
[12] Jupyter Notebook, http://jupyter.org/
[13] Mantid, simulator for neutron scattering, http://www.mantidproject.org/Main_Page

ISBN 978-3-95450-147-2