FLeSSR Usecase 1 - Multi Platform Software development

David Wallom, Matteo Turilli, Chris Williams

Introduction

Within the research community there are a significant number of software development projects that are funded through the research funding agencies, including the Collaborative Computational Projects (which have been running for several years) as well as the new EPSRC Software Sustainability Institute. All of these groups need to be able to build production quality software upon which large communities of end users depend. We have seen within large international projects until recently a wish to maintain control over the projects and development cycle through the limitation of the platforms onto which the software may be deployed in a supported manner. This is pragmatic though has resulted on a number of occasions in scenarios that are not optimal with their being constrained on old or nearly unsupported operating systems and unable to upgrade. Newer large projects have recognised that this is not sustainable in the long term but are no better resourced to provide support to multiple different types of platforms that their researchers may utilize and so must turn to technical solutions.

A prime example of this type of activity is the now formerly established ESFRI project, the Square Kilometer Array [1]. This has a global participation with many loosely connected researchers beyond the groups that are actually constructing the instrument and its software systems. They have decided that they will not limit the platforms on which the SKA software for analysis utilization of data can be run. Therefore, software that is developed by the consortia must be able to be tested during development as both unit and integration tests and on a nightly build environment. This project is not alone within the ESFRI sphere with exemplars that can be found in all of the 5 ESFRI areas that have a similar requirement, from ELIXIR (2) through CLARIN (3) and onto X-FEL(4) etc.

There are a significant number of different documents and schemes that can be used within a project to ensure good software development practice and for this project the requirements that we were trying to answer with this usecase were very simple. Alongside the development of the software compilation environment and the tools to integrate standards software development this

usecase is also a prime example of the skills and experiences at the management of multi instance behaviour and management.

Usecase Description

When supplying a software tool to an end user, the actual coding is only half the battle. Installing a working executable on the users' machine is also a significant challenge – particularly when there are a large number of dependencies. As this can be non-trivial, it is important to test this installation process on a machine that resembles that of the end user, which is very unlikely to have all the products installed that a developer might have on the machine on his desk. Provision of such an environment would be a very useful facility that can seriously reduce application support costs and end-user frustrations.

Developing software for multiple different platforms is often a requirement. The usual reasons for this include: allowing the end user to use their favourite platform, complying with the requirements of their employers IT environment, and reducing the risks of vendor lock-in. In addition, multi-platform development can significantly increase the quality of code, as it requires better design.

Developers need access to all the supported architectures in order to test and debug their code. It is unreasonable for them to be forced to maintain multiple different platforms, as there could be over 20 of these just in popular Linux distributions, though each project still will invest significantly in hardware for each of their developers. This is not just a hardware issue, but also a system administration issue as each platform has to be kept up to date and installed with the software needed for the development. A central service to provide these platforms on demand is clearly a great time and resource saver, allowing developers to concentrate on writing and testing their code.

As well as product releases, the code base is often rebuilt and tested every night to ensure that no bugs have slipped in unnoticed. A service to build and test on all the required platforms is therefore needed.

Usecase system design

There are two different stages to the use case as designed. The first is for the easy provision of an exemplar instance into the cloud with the ability to choose the type of operating system that will be installed onto the system along with dependent software and libraries for the application that is to be tested.

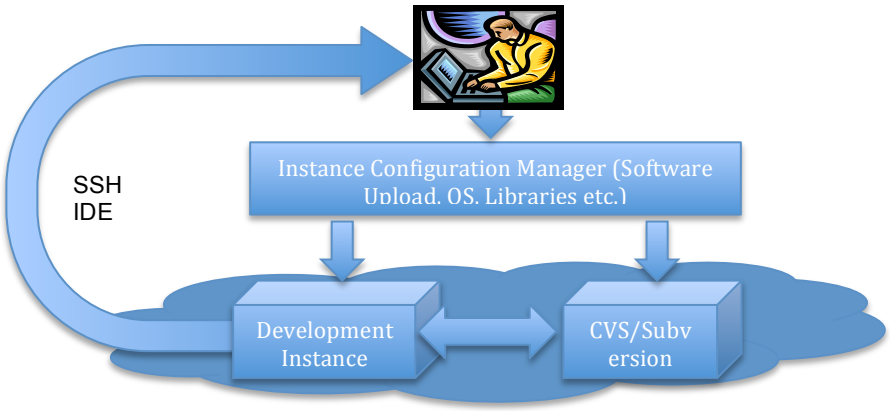

Figure 1; Provision of a single instance for code support and bug investigation

The second is the provision under which the application developer is able to launch - using some form of automated framework such as buildbot or similar - a number of predefined compilation instances using a set of different platforms. After compilation has been performed the application can have unit and integration tests run against it to test functionality.

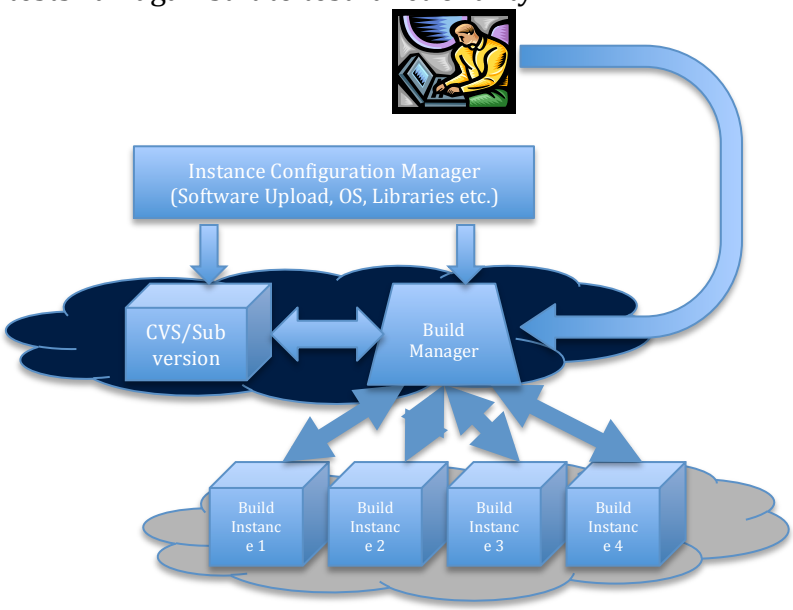

Figure 2; Description of software build operation with compilation instances running on public cloud

The operation of the build system should follow the following operational steps to simplify the interactions between the software developer and the application environment. Within the figure 2 we have shown that the software compilation actually occurs on the hybrid cloud system

The project workflow as first designed is

1) Select Project
   - Select dependant codes
   - Select build platforms, OS and architecture
   - Select project source-code
2) Start Instances
   a. Based on select platform list start instances

b. On running instances download and install dependencies and source-code
c. Start build process
d. Stream standard out and standard error from each build instance

3) Compile and report outputs
e. Pack complete products
f. Report errors and ship complete builds
g. Allow access to error plagued builds directly on instances

Introduction to MPP, the Multi-Platform Publishing tool

MPP is a tool developed within the radio astronomy community to ease the complexity and cost of deploying computer software across multiple-platforms.

Each platform has its own conventions, standards, package managers and packaging formats for deploying software. MPP allows the user to describe a software product in generic terms (e.g this is a binary, this is a library, this is documentation, it requires these dependencies), and will produce a suitable package tailored to each supported platform, that can be easily installed by the user in the normal way native to that platform.

It consists of three Process layers: Build, Test and Publish and is designed to support the entire release and testing processes. Each of these processes can be launched with a single mpp command.

The Build

With a suitable description file, the source code and access to the supported platforms, MPP will create suitable packages containing the build products. These packages are transferred from the platform on which they were built, to a central repository for testing and deployment. Below are two different descriptions files, one for a C++ compilation and other a python build.

```
[project]
name=oskar-simulator
licence=BSD
description=The Oskar SKA Station Beamforming Simulator Backend

[platforms]
ubuntu_8_10-64bit
ubuntu_8_10-32bit
ubuntu_8_04-64bit
ubuntu_8_04-32bit
ubuntu_9_04-64bit
ubuntu_9_04-32bit
openSuse_11_1-32bit
openSuse_11_1-64bit
fedora_11-64bit
fedora_11-32bit
centos_5_3-32bit
centos_5_3-64bit

[install]
/*

[dependencies::build]
cmake
c
c++

[dependencies]
cblas
openmpi
```

```
[project]
description=Log generation Tool
licence=GPL

[description]
purr will watch a directory for any new files. If its a type it can recognise it will analyse the file and generate appropriate images etc in a html format log file.

[platforms]
ubuntu_9_10-64bit
ubuntu_9_10-32bit
ubuntu_9_04-64bit
ubuntu_9_04-32bit
ubuntu_8_10-64bit
ubuntu_8_10-32bit
ubuntu_8_04-64bit
ubuntu_8_04-32bit
openSuse_11_1-64bit
openSuse_11_1-32bit
fedora_11-64bit
fedora_11-32bit
centos_5_3-32bit
centos_5_3-64bit

[dependencies::runtime]
python-qt4
python-imaging
python-pyfits
python-tk
```

```
xml2

[dependencies::runtime]
server-ssh

[build]
cmd=cmake                                   -
DCMAKE_INSTALL_PREFIX=${prefix}/usr    .    &&
make && make install

[build::ubuntu_8_04]
useRepository=oxford_apt:pre-release

[build::openSuse_11_1]
useRepository=oxford_meqtrees_yum:pre-
release
cmd=cmake                                   -
DMPI_COMPILER=${install::lib}/mpi/gcc/openmp
i/bin/mpic++                                -
DCMAKE_INSTALL_PREFIX=${prefix}/usr    .    &&
make && make install

[build::openSuse_11_1-64bit]
cmd=cmake                                   -
DMPI_COMPILER=${install::lib}/mpi/gcc/openmp
i/bin/mpic++                                -
DCMAKE_INCLUDE_PATH=${install::lib}/mpi/gcc/
include                                     -
DCMAKE_LIBRARY_PATH=${install::lib}/mpi/gcc/
openmpi/lib64                               -
DCMAKE_INSTALL_PREFIX=${prefix}/usr    .    &&
make && make install

[build::centos_5_3]
useRepository=oxford_meqtrees_yum:pre-
release
cmd=cmake                                   -
DCMAKE_C_COMPILER=${pack::openmpi::compiler_
cc}                                         -
DMPI_INCLUDE_PATH=${pack::openmpi::include}
-DMPI_LIBRARY=${pack::openmpi::lib}          -
DCMAKE_INSTALL_PREFIX=${prefix}/usr    .    &&
make && make install

[code]
srcDirectory=1.1.1
srcPack=oskar-${version}.tar.bz2
```

```
[install_link::bin]
purr=${install::python_lib}/Purr/Purr/purr.py

[install::python_lib]
purr.pth

[install::python_lib::Purr/Purr]
Purr/*.py

[install::python_lib::Purr/Kittens]
Kittens/*.py

[install::python_lib::Purr/icons/purr]
icons/purr/*.png
icons/purr/*.xpm

[install::python_lib::Purr/Purr/Plugins]
Purr/Plugins/*.py


[install::python_lib::Purr/Purr/Plugins/local_pyc
hart]
Purr/Plugins/local_pychart/*.py

[install::python_lib::Purr/Purr/Plugins/local_pyc
hart/afm]
Purr/Plugins/local_pychart/afm/*.py

[postinstall]
${command::python}          -m          compileall
${install::python_lib}/Purr

[preuninstall]
rm ${install::python_lib}/Purr/Purr/*.pyc
rm ${install::python_lib}/Purr/Kittens/*.pyc
rm ${install::python_lib}/Purr/Purr/Plugins/*.pyc
rm
${install::python_lib}/Purr/Purr/Plugins/local_py
chart/*.pyc

[build]
copyExpand=(purr.pth purr.pth)

[code]
srcDirectory=Purr
srcPack=purr-${version}.tar.bz2
```

Two example build configuration files, one a C++ based application (left) and the other a python application (right), both for the SKA project.

Testing

For each supported platform, it is important to try out the packages built in the previous stage to ensure they work for a typical user. Clean images for each platform are used (i.e. one without any dependencies or development products installed), and any problems with the install are reported back for fixing. Additional post-install testing scripts can also be specified.

Publication

Now we have packages that work, they are ready to be distributed to the wider community. MPP allows you to publish these packages to suitable repositories (again platform specific) from which other users may install the packages on their machines directly. MPP supports multiple level of publishing to allow you to tailor your release process (e.g. a repository for beta-testers, one for supported releases etc.).

MPP and the FLeSSR Project

The work undertaken within the context of the FLeSSR project was to enable MPP to use Cloud resources through the EoverI broker service, eZEEL [5]. Such functionality is a great step forward in being able to offer MPP as a generic

centrally managed service to all software projects. This involved refactoring the MPP code to support multiple instances of any given target architecture. This gives the ability to launch and interact with them through the EoverI service.

Thus a user with suitable credentials, and that have registered with EoverI are now able to perform the Build and Testing steps on platforms hosted on the cloud. For example when the user invokes the build step, MPP will request that a separate virtual machine for each platform type is instantiated. It will then use these machines to generate the packages, shutting them down and freeing the resources on completion.

Usecase utilisation and critique

The SKA consortium has not, due to some problems with the private cloud instance, been using the system heavily. There were teething problems due to slow startup within the build manager of the java virtual machine though this should be infrastructure specific and could be designed to be long lived etc in a future production service. They have though looked at porting the ideas that have been developed through this project onto VMWare ESXi [6] which they already have access to within the hosting department and this use may be taken further as the availability of these type of systems develops. There are though limitations for this solution with ESXi not being able to easily provision multiple instances from a single disk image.

As initially designed the system was setup so that a single user had access to the cloud broker and resources which would have limitations for a production environment with many users. Therefore, further work is required in this area so that multiple users are easily able to share instances between themselves, in a controllable way and based on user attributes. This is currently though a limitation between many of the cloud solutions which are 'all or nothing' in terms or instance and data sharing. With the availability of institutional and UMF cloud systems there is a need to investigate the funding/charging mechanisms for cloud services since projects normally have capital funds but not recurrent costs.

Ongoing work needed for usecase?

Within the originally designed usecase it was envisaged that the software build process would be operated from within a graphical user environment, possibly through a privately located web interface. This feature has not been finally designed into the usecase due to developers that have been using the solution prior to its cloud deployment not feeling that this would be so useful since they historically develop within the command line and text editing capability. A future piece of work though would be to develop a plug in for the popular and open source Eclipse framework such that software builds could be deployed onto the multiple target platforms the developer needs to support quickly and easily.

References

1. SKA - http://www.skatelescope.org
2. ELIXIR - http://www.elixir-europe.org
3. CLARIN - http://www.clarin.eu
4. XFEL – http://xfel.desy.de

5. eZEEL - http://www.eoveri.com/products/ezeel.html
6. VMWare ESXi - http://www.vmware.com/products/vsphere-hypervisor/overview.html