ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA042

ASCI: A COMPUTE PLATFORM FOR RESEARCHERS AT THE AUSTRALIAN SYNCHROTRON

J. Marcou, R. Bosworth, R. Clarken, P. Martin, A. Moll
Australian Synchrotron – ANSTO, Melbourne, Australia

Abstract

The volume and quality of scientific data produced at the Australian Synchrotron continues to grow rapidly due to advancements in detectors, motion control and automation. This means it is critical that researchers have access to computing infrastructure that enables them to efficiently process and extract insight from their data. To facilitate this, we have developed a compute platform to enable researchers to analyse their data in real time while at the beamline as well as post-experiment by logging in remotely. This system, named ASCI, provides a convenient web-based interface to launch Linux desktops running inside Docker containers on high-performance compute hardware. Each session has the user's data mounted and is preconfigured with the software required for their experiment.

INTRODUCTION

ASCI consists of a cluster of high performance compute nodes and a number supporting applications. These include an application for launching and managing instances (asci-api), a web interface (asci-webui) and a proxy server for relaying connections (asci-proxy).

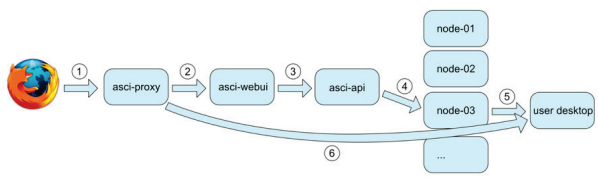

Figure 1: Overview of launching an ASCI session.

The sequence of events involved in creating and connecting to an ASCI desktop session is illustrated in Fig. 1. Users first log in to the web interface and select an environment appropriate for processing their data. The webui sends a request for a new instance of that environment type to the asci-api. The asci-api selects the best compute node to launch the instance on based upon the requirements of the environment and the load on the cluster. Once it has picked a node, the asci-api launches a Docker container based upon the requested environment. The user is then presented with an icon representing the running session and they can connect to this desktop from their web browser.

When the user initiates a connection, a VNC session is created inside the Docker instance with a one-time password. This password is used to launch a NoVNC connection in the user's browser and the user is presented with their desktop and can commence analysing their data.

DOCKER AND ASCI ENVIRONMENTS

Docker containers are a technology for creating isolated process environments on Linux [1]. We chose this technology for the ASCI user environments because they deliver almost identical performance compared with running applications on the bare-metal operating system, while enabling multiple users to simultaneously utilise the node. Docker also enables us to create predefined environments, tailored with the applications required for different types of experiments. Each environment is based on a Docker image which is defined by a text file outlining how to prepare the desktop. These image recipes support inheritance, enabling us to have a base image with installs the ASCI infrastructure applications and then child images with the specialised scientific software for the different experiments, as shown in Fig. 2.

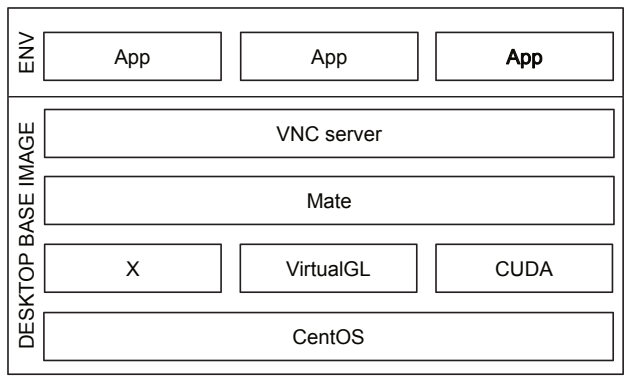

Figure 2: Components of an ASCI Environment.

WEB INTERFACE

A goal of the ASCI project was to allow users to connect to desktop sessions through their standard browser rather than requiring users run a specialised application such as a VNC client. This greatly lowers the barrier to entry to using the system and allows users to access it from any operating system.

We built the web interface using Flask for the server and React for generating the front-end. For rendering the desktops in the browser, we utilise NoVNC [2] which delivers a VNC connection over WebSockets. This results in a responsive interface that runs on all platforms, including mobile and tablet. The appearance of an ASCI Desktop connected over NoVNC is shown in Fig. 3.

VIRTUALGL

In order to provide GPU hardware acceleration to multiple ASCI instances on one node, we need to use a modification

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THPHA042

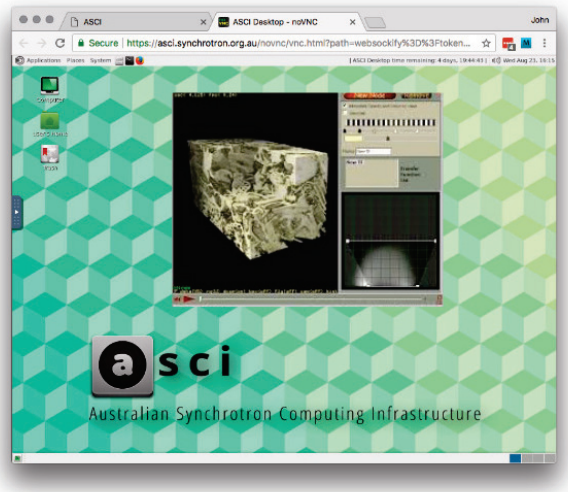

Figure 3: ASCI sessions deliver the familiar Mate desktop environment as well scientific software applications.

of the traditional X architecture. Usually, the X server has direct access to the GPU hardware and this allows graphical application to execute OpenGL instructions. The challenge when running multiple desktops on a single node is that multiple X servers cannot share direct access to the same GPU hardware.

To address this, we run a single X server directly on the node; this is known as the 3DX server. Every ASCI instance then runs its own internal 2DX server which handles graphical applications, such as the Mate desktop environment. When applications make use of the GPU, we launch them with environment variables which causes them to load VirtualGL libraries in place of the standard OpenGL libraries. The VirtualGL libraries will catch and forward all OpenGL instructions to the 3DX server which then executes them on the GPU [3].

DEPLOYMENT

Every component of the ASCI system runs inside its own Docker container. This enables us to precisely define the environment of each application, such as the operating system and dependencies, and to easily reproduce the applications on different hosts. It also means when a developer tests applications on their local machine, they are doing so in the same environment as it will run in production. To facilitate deploying updates we created an application called Autobuild which receives notifications from our Bitbucket server whenever a tag is added to an application's git repository. When Autobuild sees a new tag it clones the code from Bitbucket and uses Docker to build an image for the application based on a Docker file in the repository. The built image is then pushed to our internal Docker registry ready for deployment.

To provision machines to run the Docker containers we use Terraform [4]. This enables us to define, in code, a recipe for creating Virtual Machines on our VMware vSphere cluster and automatically installing and configuring the operating system. For bare-metal systems, such as the compute nodes, Terraform configures a service called Matchbox [5] to enable network booting (PXE) based on the machine's motherboard UUID.

Every machine in the ASCI system runs CoreOS Container Linux which is optimized for container environments and configurable with manifest files. On bare-metal systems, the operating system is loaded in-memory, reducing the state footprint of the system.

MONITORING

To monitor ASCI we use a collection of open source tools known as the Elastic Stack [6]. This includes a database, Elastic Search, for capturing logs and metrics, and the front-end website, Kibana, for viewing logs and creating dashboards. To harvest logs we have the applications inside the Docker containers log to standard out and configure Docker to forward the logs to journald. A utility called Journalbeat then collects the logs and sends them to an Elastic Search pipeline based on the source of the log. The pipeline parses the log and ingests the output into Elastic Search. For alerting, we have an application called ElastAlert monitor the Elastic Search database and trigger a Slack notification based on certain rules. This enables us to be instantly alerted whenever an error occurs or in the case of unusual user behaviour on the website which may be indicative of an attack on the system.

CONCLUSION

ASCI is now in use at the Australian Synchrotron for processing data being produced at the Medical Imaging, X-ray Fluorescence Microscopy and Micro-crystallography beamlines. The simple web interface and tailored environments provide an easy and intuitive platform for users to process their data and the automated build systems allow fast and painless deployment of updates. Future upgrades to the system will include supporting alternative interfaces to the environments, such as Jupyter Notebooks, and integrating a batch job submission system to distribute processing tasks across multiple nodes.

REFERENCES

[1] Docker, https://www.docker.com

[2] NoVNC, http://novnc.com

[3] VirtualGL, https://www.virtualgl.org

[4] Hashicorp Terraform, https://www.terraform.io

[5] Matchbox, https://coreos.com/matchbox

[6] Elastic Stack, https://www.elastic.co