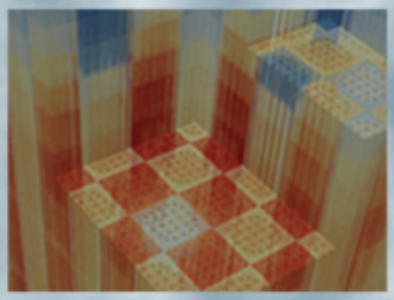
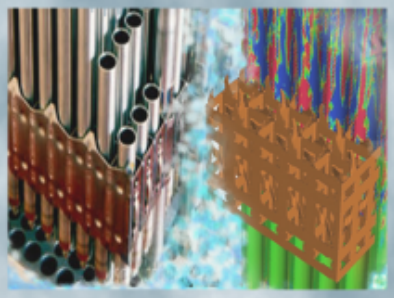
CASL-U-2016-1194-000





Docker binary release for VERA

Damien Lebrun-Grandie, ORNL
Kevin Clarno, ORNL
Mark Baird, ORNL

September 29, 2016

U.S. DEPARTMENT OF ENERGY | Nuclear Energy

REVISION LOG

Revision	Date	Affected Pages	Revision Description
0	09/29/2016	All	Initial Release

Document pages that are:

Export Controlled _____

IP/Proprietary/NDA Controlled_____

Sensitive Controlled_____

Requested Distribution:

To:

Copy

CONTENTS

ACRONYMS

AWS Amazon Web Services
CASL Consortium for Advanced Simulation of Light Water Reactors
CTF COBRA-TF subchannel thermal-hydraulics code
DOE US Department of Energy
DTK Data Transfer Kit
INL Idaho National Laboratory
LWR light water reactor
MCNP Monte Carlo N-Particle transport code developed and maintained by Los Alamos National Laboratory
MPACT Michigan PArallel Characteristics Transport code
MOOSE Multiphysics Object-Oriented Simulation Environment developed by Idaho National Laboratory
ORNL Oak Ridge National Laboratory
RSICC Radiation Safety Information Computational Center
SCALE Comprehensive modeling and simulation suite for nuclear safety analysis and design developed and maintained by Oak Ridge National Laboratory
VERA Virtual Environment for Reactor Applications

1. INTRODUCTION

Over the past few years there has been a persistent desire to have an executable only distribution of VERA for several reasons related to export control and usability. The Radiation Safety Information Computational Center (RSICC) have moved to executable-only releases of SCALE and MCNP unless there is a clear need for the source code to better control the distribution; it is expected that this will be requested of VERA at some point as well. Because the export control approval process for foreign nationals is faster with the executable-only releases, this would significantly benefit the CASL education program to get VERA into the hands of students and streamline the CASL Institute planning process. In addition, many users do not want or need to worry about underlying hardware and software requirements to simply run the code.

With SCALE and MCNP, only serial versions of the code are released as executables, because hardware differences make a simple binary release difficult to achieve strong parallel performance. Therefore, RSICC has also made a "Secure Cloud Server" option available to run these codes in parallel when an individual might otherwise be denied access, but are at a US university or collaborating with US organizations.

This reports documents the completion of CASL milestone L3:PHI.INF.P13.02 to evaluate "Non-source VERA-CS release options." It was created in the middle of fiscal year 2016 (FY16) to evaluate an initial capability to develop and distribute an executable-only version of VERA using two related avenues: use of the "Secure Cloud Server" and creation of "Docker container" for VERA. The specific description states that this milestone will "demonstrate one pathway, using the Dockers tool, and provide an in-depth exploration of a few other options to define work required in FY17. This will include discussions with INL to ensure that pathways for CASL are inline with, or at least not perpendicular to, INLs plans for MOOSE and/or Bison."

The execution plan included steps to:

- create an initial Docker container for the primary VERA code (VERA-CS),
- discuss this plan with INL to ensure that our executable-only pathway is not perpendicular to their efforts, and to
- evaluate the "Secure Cloud Server" option, with the stretch goal to
- create a Docker container that included more of VERA: Shift, Bison, and Tiamat.

The following sections detail the production of a lightweight Docker image with a working installation of all of the VERA simulation components. Following several email threads with INL, it was clear that they were not interested in executable-only releases at all and are supportive of our efforts to use both Docker and a secure cloud.

In the last section of this report, we detail the use of the RSICC Inspect cluster as a secure cloud and discussion of the potential for the AmazonCloud.

2. VERA BASE IMAGE WITH THE DEVELOPMENT ENVIRONMENT AND TPLS

This section describes the creation of the base image for VERA that has the development environment and third-party libraries in place.

2.1 Building the VERA base image from a `Dockerfile`

A `Dockerfile` was used to specify instructions to create the VERA base image. It is a text document that contains all the commands a user would call to assemble the image. It has been pushed to the CASL/vera_tlps repository on GitHub and is located in the `docker/` directory in the VERA TPLs source. Below is how the `Dockerfile` looks like:

```
FROM ubuntu:16.04

RUN apt-get update && apt-get install -y
        build-essential \
        gfortran \
        mpich \
        cmake \
        git \
        [...]

ENV VERA_DEV_ENV_BASE=/tools/vera \
    VERA_SCRATCH_DIR=[...]

RUN mkdir -p ${VERA_BASE_DIR} && \
    cd ${VERA_BASE_DIR} && \
    git clone https://github.com/CASL/vera_tpls && \
    [...]
```

As can be seen from it, the VERA image is built on top of the official image for Ubuntu 16.04 Xenial which is a LTS release. The `apt-get` package management tool is used to install up-to-date compilers with MPI wrappers and other required tools such as CMake or Git. The usual VERA environment variables are declared and the TPLs are installed. (Note: It is safe to override some of the Directory Env Vars such as `VERA_DIR` or `VERA_INSTALL_DIR` later on when extending the VERA base development image.)

2.2 Issues related to building with GCC-5.X

The native package manager system was used to install the development environment instead of running the `install_dev_tools.py` script from TriBITS as prescribed by the VERA installation guide. As a result, the generic tools and compilers come with versions that are more recent than what a typical VERA Dev Env install would have. The table below summarizes the versions currently installed on the image as of September 2016:

Tool	version
GCC	5.4.0
MPICH	3.2
CMake	3.5.1
Git	2.7.4
Python	2.7.12
Perl	5.22.1

There was no major issue to install the VERA TPLs with this somewhat unusual toolset. Few adjustements to the CMake build system were needed and two of the actual TPLs required patches, namely:

- Boost version 1.55.0 (released in November 2013 - Bug is fixed in 1.56.0)
- HDF5 version 1.8.10 (released in November 2012 - Latest release of the 1.8.x Series is 1.8.17)

2.3 Pulling the VERA base image from Docker Hub

The `docker build` command assembles the image from the `Dockerfile`:

```
$ git clone https://github.com/CASL/vera_tpls
$ cd vera_tpls/docker
$ docker build -t <image name> .
```

The build has not been automated yet but the image `dalg24/vera_tpls` may directly be pulled from a public repository on Docker Hub, by doing:

```
$ docker pull dalg24/vera_tpls
```

3. USING THE VERA BASE IMAGE

This section contains reference information on how to use the base image that meets the prerequisites to launch a container and configure, build, test, install VERA within that container.

3.1 Mounting the VERA source into the container

By default, Docker limits the size of its images and containers to 10 GB. The Docker daemon will throw an error if an existing base device becomes larger. This is problematic since cloning the VERA source from `casl-dev` and calling the script `clone_vera_repos.py` produces over 12 GB of data as of today, and a shared release build will add about 8-9 GB to it! Surely, it is possible to increase that default limit but better options are available to get the VERA source into the Docker container.
Additionally, pulling the source from a running container would require adding SSH credentials registered with the Gitolite system that is used to manage the VERA Git repositories inside that container which is not necessarily a wise idea.
A powerful way of managing data inside containers with Docker are data volumes. In particular, Docker let the user mount a directory from the Docker engine's host into a container using the `-v` flag.
To launch a container from the VERA base image, one might want to do:

```
$ CONTAINER_VERA_INSTALL_DIR=/tools/vera/installs/`date +%Y-%m-%d`; \
  CONTAINER_VERA_DIR=/scratch/vera_source; \
  docker run -t -i \
    --name <give your container a name> \
    -e VERA_INSTALL_DIR=$CONTAINER_VERA_INSTALL_DIR \
    -e VERA_DIRA=$CONTAINER_VERA_DIR \
    -v <path to VERA on the host machine>:$CONTAINER_VERA_DIR \
    dalg24/vera_tpls bash
root@af8bae53bdd3:/# # start a Bash session inside the container
```

This will mount the local VERA source tree into the container at `/scratch/vera_source` and launches an interactive Bash shell inside the container as the `root` user.

3.2 Build and test VERA inside the container

Inside the running container, one would then run these commands:

```
root@af8bae53bdd3:/# cd $VERA_BUILD_DIR && ./do_configure.sh
```

The `VERA_BUILD_DIR` environment variable is already defined and a working configure script has been copied into that directory. Then, one would do `make -j<N>` and `ctest -j<N>` as one would usually do.

3.3 Commit a new image with the VERA components

After finalizing the VERA install, if one wish to commit these changes into a new image, one could use:

```
root@af8bae53bdd3:/# rm -f $VERA_BUILD_DIR # clean up disk space
root@af8bae53bdd3:/# exit # exit the container
$ docker commit <container name> <choose a name for the new image>
```

Alternatively, one can save your VERA image into an `.tar.gz` archive using:

```
$ docker export <container name> | gzip > <tarball name>
```

To turn the archive back into a Docker image on another system, one would do:

```
$ gunzip -c <archive name> | docker import - <image name>
$ docker run -it --rm <image name> bash
```

(Note: The image that is produced does not have the VERA source on it. Neither the `docker commit` nor the `docker export` command will include any data from the volumes that have been mounted inside the container.)

4. AUTOMATED NIGHTLY BUILD WITH DASHBOARD SUBMISSION

This section gives details on the setup for a VERA nightly automated build within a Docker container.

4.1 Run a Cron job within the Docker container

The job is scheduled to run nightly using Cron. The `crontab` file runs a CTest script that has been added to the `cmake/ctest/dirvers/docker/` folder of the VERA repository. The Cron job is placed within the container rather than on the host system which makes it easier to deploy. All one need to do is launch the VERA CI container which is pre-configured to run the build nightly.

4.2 Use Docker Compose to run VERA CI container

Docker Compose provides a convenient way to run the VERA CI container. The testing environment is defined in a Compose file which looks like:

```
version: '2'
services:
  vera_ci:
    container_name: casl_vera_ci
    restart: always
    build:
      context: .
      dockerfile: Dockerfile
    image: casl_vera_ci
    env_file: .env
    environment:
      - TERM=xterm
      - TZ=America/New_York
    volumes:
      - <VERA source>:$VERA_DIR
      - <SSH private key>:/root/.ssh/id_rsa:ro
```

One would clone VERA on the host system and specify the path in the `volumes` section of the `docker-compose.yml` file in order to mount it into the container. One must also provide the path to a SSH private key on the host machine (needed to be able to pull repos from `casl-dev` when updating the source).
The `Dockerfile` listed in the Compose file contains the set of instructions to augment the VERA base image in order to be able to run the nightly build. Should the VERA base image be updated or the setup for the Cron job change, then one would be responsible for calling the `docker-compose build` command to produce a new image before redeploying the VERA CI container.
To start the automated build service, all one need to do is:

```
$ cd $VERA_DIR/cmake/ctest/drivers/docker
$ docker-compose up -d
```

5. POTENTIAL USE OF THE AMAZON GOVCLOUD

The AWS GovCloud is an isolated region of the Amazon Web Services designed to address the specific regulatory needs of US governement agencies and organizations. This section discuss the deployment of containers on the Amazon Elastic Compute Cloud (EC2) as well as the hosting of image repositories.

5.1 Deploy containers on the cloud

Amazon EC2 provides the compute capacity in the cloud. It comes with a container management service (Amazon ECS that supports Docker containers and makes it relatively easy to run and manage Docker-enabled applications across a cluster of Amazon EC2 instances. It eliminates the need to install and operate the Docker engine on Amazon EC2 instances. Instead, one defines tasks through a declarative JSON template, where one specifies what image to derive the container from, memory and CPU requirements, data volumes, etc.
One has full control to create and terminate containers. One could then consider running a SSH server within the VERA container and allow the user to connect remotely. The advantage is that the user does not even have to know anything about Docker. As far as he is concerned, he has been granted access to a machine that has a working installation of the VERA simulation components.

5.2 Store and distribute images with the VERA components

Turning container images into tarball archive and transfering them between hosts over the network is cumbersome. It is not a practicable long-term solution for distributing VERA images. Docker Registry is the native approach for storing and distributing Docker images.
Amazon EC2 Container Registry (ECR) is an option that would make it easy to store, manage, and deploy Docker images with the VERA components. It is well integrated with the Amazon EC2 Container Service and would greatly simplify the workflow. This way, CASL would not have to host and manage its images. Amazon ECR offers a fine-grained access control of each repository which is not supported by the standard free-of-charge Docker Registry. (The commercial version Docker Trusted Registry does.)

6. VERA EXECUTABLES ON INSPECT

6.1 Utilizing the InSPECT Cluster for the CASL Institute Training

In support of the CASL Institute training at ORNL in June 2016, RSICC's InSPECT cluster was setup to allow students to access the VERA executables without providing export controlled source code to the students. InSPECT is an 80 node, 2560 core cluster setup by RSICC, from funding provided by the Department of Energy, to house export controlled software and allow access for foreign nationals who would not normally have been granted access to the software. As stipulated in the cyber security plan for InSPECT, no export controlled source code is allowed on the machine, where a user could potentially access the source code. This led to the need to compile the VERA executables on a separate LINUX system and port over the executables to InSPECT.

To accomplish this, the VERA environment was installed on a similar machine as InSPECT, REMUS.ornl.gov with RHEL6, following the VERA Installation Guide. Next VERA was configured and built with shared libraries enabled to avoid the need to create an exact directory structure in InSPECT to mirror REMUS. Once the executables were created, their build was verified by running the "Continuous Integration" testing. Upon successful completion of all continuous integration tests the install directories and third party library files were copied to InSPECT using SCP (Secure Copy). The GNU GCC compilers and MPICH wrappers were built on the InSPECT cluster to allow for parallel execution if the VERA binary files.

To verify the installation of Compilers, Wrappers and VERA on InSPECT, selected inputs for MPACT and COBRA-TF were ran to verify the outputs. Also all library links in the dynamically linked executables were verified to be properly directed to the appropriate tpl and system libraries on InSPECT using the "ldd" command.

6.2 User interaction with VERA executables on InSPECT

Users were assigned a specific account on InSPECT and RSA token for login. To properly run their practicum and in-class projects in InSPECT, scripts were created to allow the student to submit their input files anonymously to a system directory, which in turn would pass the inputs, by a cronjob script, to the system Batch Submission System. Upon completion, the input and output files were return to the user's home directory.