REPORT ON D3.1, LEAD BY Université de Versailles Saint-Quentin (UVSQ): Virtual images and containers

ERIK BRAY, LUCA DE FEO

Due on: 01/03/2016 (M6)
Delivered on: 01/03/2016

Progress on and finalization of this deliverable has been tracked publicly on:
`https://github.com/OpenDreamKit/OpenDreamKit/issues/58`

1. Status report

We have created virtual images and containers for most of the relevant components of OpenDreamKit. The following projects have been dealt separately:

- MathHub, (containers hosted at `https://hub.docker.com/r/kwarc/`)
- GAP (containers hosted at `https://hub.docker.com/u/gapsystem/`)
- OOMMF (virtual image hosted at `https://github.com/fangohr/virtualmicromagneti`
- SageMath, PARI/GP, Singular, etc. (containers hosted at `https://hub.docker.com/u/sagemath/`)

We report below on each of these components

1.1. **MathHub.** The MathHub team at Jacobs University maintains two Docker containers for MathHub:

- MathHub (Dockerfile source `https://github.com/KWARC/mathhub_docker`)
- LocalMH (Dockerfile source `https://hub.docker.com/r/kwarc/localmh/`)

The first container is for a MathHub instance, like the one at `https://mathhub.info/`. The second container is for the LocalMH tool, for offline authoring of MathHub installations.

Both containers are manually built. Build automation via Docker Hub's automated build service is a potential improvement that will be studied in a next step.

1.2. **GAP system.** The GAP team maintains two Docker containers:

- gap-container (Automated build from `https://github.com/gap-system/gap-container`)
- gap-docker (Automated build from `https://github.com/gap-system/gap-docker`).

The first is a minimalistic build containing exclusively the core GAP functionality, without any optional GAP packages (user-contributed extensions, redistributed with GAP). The second is a full GAP install, containing all packages redistributed with GAP. Because some of the packages

have non-trivial dependencies on third-party software, gap-docker depends on a gap-docker-base container which provides these dependencies. gap-docker-base is not built automatically because of the time and memory restrictions of DockerHub automated builds.

The GAP Docker container is suggested as one of the alternative ways to install GAP: `http://www.gap-system.org/Download/alternatives.html`.

As a demo application to demonstrate the use of GAP Docker container in the cloud, we run an SCSCP server for the number of groups of order n. This service uses GAP Docker container running on a Microsoft Azure virtual machine. Usage details could be found at `https://github.com/alex-konovalov/gnu`.

GAP is also bundled in the SageMath containers described below.

1.3. **OOMMF.** The Virtual Micromagnetics package is designed to be run through Vagrant provisioned by VirtualBox. The virtual image is hosted at `https://atlas.hashicorp.com/virtualmicromagnetics/boxes/full`.

The Virtual Micromagnetics developers are considering Docker as an alternative provider for the OOMMF component.

1.4. **SageMath.** SageMath bundles most of OpenDreamKit components. The community maintains several Docker containers for use and development of SageMath at `https://hub.docker.com/u/sagemath/`. We describe below the work that has been done in OpenDreamKit regarding those containers.

For SageMath we have created a repository for issue tracking and development of build scripts for SageMath docker images at sagemath/docker-images (not to be confused with sagemath/docker which we have agreed will be superseded by the former). The repository currently includes build recipes for four Docker images:

- sagemath/sagemath - This will always provide a build of the most recent released version of SageMath (or previous versions through the use of image tags).
- sagemath/sagemath-develop - This provides a build of the most recent develop branch of SageMath from the git repository at the time of building. This image will need to be updated automatically on a regular basis via an automated mechanism (more on that later).
- sagemath/sagemath-jupyter - This is the same as the sagemath/sagemath image, but automatically starts the Jupyter notebook with sage when it is run (the base sagemath/sagemath image drops into the sage command line by default).
- sagemath/sagemath-patchbot - This is the same as the sagemath/sagemath-develop image, but automatically runs the sage patchbot by default. There are some currently unresolved (but likely minor) issues with running the patchbot in this container.

We have found Docker containers to be an effective way to distribute a consistent installation of Sage across multiple platforms–Docker support Linux, Mac OSX, and Windows (with some caveats; more on that below). Running Sage in a Docker container provides a consistent experience to the user, regardless what the host OS is, and is generally lighter-weight and more transparent to the user than a full VM. When using the sage command line interface (or any other command line program running in the container) the user interacts with it directly through their host OS's terminal emulator. The container can also run headless, such as when running the notebook, in which case the user can connect their web browser to the notebook server as though running directly on localhost.

We are also exploring the use of Docker on Windows as a way to provide a simple one-click installer for SageMath on Windows as near-term solution to #66 until and unless Sage can be built and run natively on Windows. This in turn has driven improvements to the sagemath Docker images. The installer for Sage would install Docker and its dependencies (if not already installed), and would include the sagemath/sagemath image (or optionally the -develop version)

and launcher shortcuts for starting the command-line and notebook interfaces. This installer is being developed in embray/sage-windows and will have a prototype ready soon.

The current Docker images for SageMath are based on an Ubuntu 15.10 (Wily Werewolf) base image, though it would be possible to reuse most of the existing build scripts to build SageMath images based on other Linux distributions. This could be useful for Sage development, but less interesting to end-users (see sagemath/docker-images#12).

The Docker Hub service for hosting Docker images does provide an Automated Build service that could in principle be used to automatically build new sagemath/sagemath-develop images (as well as sagemath/sagemath when there is a new release). It can monitor changes to the repository storing the image build recipes, and could also be configured to re-build an image upon push to another repository (such as sagemath itself). Leveraging this service would free us of the overhead of maintaining our own build infrastructure for images. However, the Docker Hub Automated Build service does have resource restrictions that may prove too limiting for building SageMath:

- 1 CPU
- 2 GB of RAM
- 30 GB disk space
- 2 hour limit to build time

The disk space is enough, and the RAM is also enough given only 1 CPU is provided. However with only one CPU it may be difficult to get the build time down to 2 hours, but we have not tried this yet. I (@embray) believe there are several actions that can be taken to improve build time of Sage. In particular, a good many of its dependencies are available, with the correct versions, pre-built from Ubuntu's APT repository (excluding packages that require special patches from Sage that affect runtime behavior). Improving the Sage build process to allow more reliance of system packages where possible will significantly cut down build time and be beneficial well beyond the case of building Docker images.

1.4.1. *Caveats:*

- Disk space: The Docker images for SageMath are quite large. Originally the sagemath/sagemath image weighed in at 7.361 GB. It was possible to reduce this to 1.988 GB by relying more on system packages and cleaning up artifacts of the sage build process at the end of the image build. For the sagemath/sagemath-develop image we decided to leave those artifacts intact, so running this image gives access to a sage source tree as it would appear immediately after running `make`. It's possible a few other reductions can be found (such as uninstalling build dependencies once the build is complete), but there's no escaping the fact that these Docker images will be a large download. Fortunately, Docker images can be exported to a `tar` archive (and further compressed with the compression method of one's choice), for distribution on USB drives and the like.
- Connecting to a notebook or other network interface running in the container does, however, require manual setup of port forwarding when running the Docker container. The command line option for this is relatively simple to communicate, however, and wrapper scripts / shortcuts may also be provided to users.
- Although the use of Docker containers provides a more or less uniform experience across host OS's, Docker does not technically run natively on Windows or OSX, as the way it works relies heavily on features specific to the Linux kernel. As such, running Docker on Windows or OSX does actually require running a Linux VM in the first place. The Docker Toolbox, which installs Docker on Windows and OSX, also installs VirtualBox and a very small Linux VM, designed specially for Docker, which includes the bare minimum to run Docker and not much else. Because of its specialized purpose, this VM adds very little overhead compared to a more general purpose VM.

Docker Toolbox includes a program that makes management of this VM easy (docker-machine), and because the VM runs headless users will never see it. The shortcuts installed by Docker Toolbox run a script that ensures the "boot2docker" VM is running, and thus allowing all other `docker` commands to work. So with a few exceptions users never need to be aware of this underlying virtualization layer:

- With Docker it is relatively easy to mount existing directories on the host machines as "volumes" accessible from the Docker container (see "Manage data in containers"). There are a few downsides to this:
  - Although any number of directories can be mounted, they can only be mounted when a container is first created, so one has to think ahead and ensure that all data one will need to access within the container will be accessible (if in doubt, one could mount their entire filesystem I suppose). In principle it is possible to add new mounts from within a container, but not in a way that is friendly to novices (requires using the `mount` command).
  - On Windows and OSX, because of the two layers of virtualization, any local directory that one wants to mount within a Docker container needs to also first be mounted in the boot2docker VM that Docker itself runs in, so it's a two step process and probably hard to explain to novices. On the bright side, boot2docker *automatically* mounts the user's home directory, so this is immediately available. And in many cases all the files a user cares about should be in their home directory, but that is of course often not the case as well. Another possible "bright side" is that VirtualBox includes a command-line interface for controlling VMs–`VBoxManage`. Although not novice-friendly, this does enable writing novice-friendly interfaces for mounting additional directories if they need to, and I (@embray) am working on such interfaces in the Windows installer for Sage (and maybe, hopefully, upstream to Docker as well).
- Networking has the same problem that directory mounting has–it is fairly easy, when starting a new Docker container, to set up which TCP ports should be forwarded from the Docker container to ports on the host machine. However, this is harder to do after a container has already been started, and one might have to start a new container if they don't forward the ports properly. Providing users with shortcuts that do this automatically helps. Windows and OSX have the same problem with ports needing to be forwarded on the boot2docker VM as well, though this can also be managed with `VBoxManage`.
- Probably the biggest hurdle to setting up Docker on Windows and OSX is the requirement for Hardware Assisted Virtualization (HAV) to be enabled on the user's system. This is not so much a limitation of Docker as it is a limitation of VirtualBox–it can't run the boot2docker VM (or any 64-bit VM) without HAV. The default HAV settings vary from system to system, though it is often disabled by default as a security measure. Some systems provide a tool to change this and other BIOS settings from within Windows, but more likely users will have to change this setting from their BIOS configuration menu. This is a support nightmare since so many BIOS menus are different and there is no standard. That said, most BIOS menus have improved substantially in the last decade, so I think helping users with this is less hassle than some other options. But it is certainly discouraging to tell novices "Now to install Sage you have to reboot your computer, hold down ESC, or DEL, or F10, or Ctrl-C, or Fn-Ctrl-Break, etc. etc. and then rummage around in this plain text curses interface until you find something that says something something about"virtualization" or maybe "VT-X" and make sure that's enabled. Then press F10 or F8 or S or etc. etc. to save the settings and reboot. Make sure not to change the time!"

I think it's worth trying this out in at least one workshop to see how bad it is in practice though. The broader Docker community may also have some useful experience with this, as this is currently the *only* way to get Docker on Windows.

A possible workaround is to provide a 32-bit version of boot2docker, which is theoretically possible (though it will run slower *without* HAV). But this is not officially supported by the Docker project, and isn't likely to be (docker/docker#136).