Customized Software Environment for Remote Learning: Providing Students a Specialized Learning Experience

Thomas Gerlach, Lukas Schauer, Michael Rademacher, Wolfgang Heiden, and Karl N. Kirschner*

Department of Computer Science, University of Applied Sciences Bonn-Rhein-Sieg, Grantham-Allee 20, 53757 Sankt Augustin, Germany

April 1, 2021

Abstract The Covid-19 pandemic has challenged educators across the world to move their teaching and mentoring from in-person to remote. During nonpandemic semesters at their institutes (e.g. universities), educators can directly provide students the software environment needed to support their learning - either in specialized computer laboratories (e.g. computational chemistry labs) or shared computer spaces. These labs are often supported by staff that maintains the operating systems (OS) and software. But how does one provide a specialized software environment for remote teaching? One solution is to provide students a customized operating system (e.g., Linux) that includes open-source software for supporting your teaching goals. However, such a solution should not require students to install the OS alongside their existing one (i.e. dual/multi-booting) or be used as a complete replacement. Such approaches are risky because of a) the students' possible lack of software expertise, b) the possible disruption of an existing software workflow that is needed in other classes or by other family members, and c) the importance of maintaining a working computer when isolated (e.g. societal restrictions). To illustrate possible solutions, we discuss our approach that used a customized Linux OS and a Docker container in a course that teaches computational chemistry and Python3.

1 Introduction

Due to the Covid-19 pandemic, educators across the world are faced with the challenge of how to create a customized remote learning experience for their students [1, 2, 3]. At tertiary institutions when teaching in physical presence (a.k.a. in-person teaching), educators can achieve this through their labs and computer rooms. For example, an instructor can have a specific program installed onto the university's computers for the student to learn and use. However, when all members of the class are interacting and studying remotely, there are significant hurdles in providing students quality learning experiences - especially concerning laboratories [4, 5].

The competent usage of technology is clearly important for student learning when it is a part of lesson. Such is the situation with remote learning, and includes competence by both the instructor and the students. In context to remote learning, the term ”technology” includes both the hardware (e.g. webcams, microphones) and software. When technology is integrated and into a course – online or in-person – and the participants have positive interaction experience, the students are more likely to be engaged in the course with higher satisfaction [6]. More specific to remote learning, one wants to minimize the student frustration of having to install software from multiple sources, which can involve compiling code and specialized libraries with dependencies. If students struggle with installing the desired software at the course's beginning, requiring the instructor to troubleshoot each problem, this can quickly lead to a lack of confidence in the students and a waste of valuable course time.

*Corresponding author: karl.kirschner@h-brs.de

This paper's goal is to report three possible solutions that enable educators to provide a customized software environments to their remote students. Two solutions provide an entire Linux operating system (OS) environment that includes the course's software. This is achieved by providing an image of the customized OS that can be either installed onto a bootable USB drive, or installed into a virtual machine (VM). The third solution involves the creating of a Docker container. Once formulated, each of these solutions provides a relatively low barrier for the students to obtain the course's software, and thus providing them a unique remote learning experience. The caveat is that these solution require the students to have access to a computer at home.

2 Example Solution Workflows for Image/Container Creation

The solutions devised for our teaching needs requires some knowledge either about VM, Linux OS, or the Docker and Docker Compose software [7] . The examples provided below are intended for educators that might be new to these ideas. As a safety precaution, we recommend educators to work through these solutions by first backing up their computer, or use a noncritical computer (e.g. an unused computer) whose software and data content is unimportant.

In the following, we demonstrate how one can provide software for a molecular modeling course. The software that is used is conveniently found on through the Conda package manager and environment management system. Specifically, OpenMM [8] and AmberTools19 [9] are used for teaching molecular dynamics, Psi4 [10] is used for teaching quantum mechanics and Jupyter Notebook is used for incorporating Python3 programming (e.g. data analysis). The following code assumes you are working with a Linux OS and a command-line interface via a bash terminal.

2.1 Customized Linux OS

The Linux OS solutions (i.e VDI file, Linux image, bootable USB drive) has the advantage that an instructor can provide an entirely customized environment that is identical for all students. This greatly simplifies trouble shooting problems that arise, and provides exact knowledge on what the students have access to. With some existing Linux knowledge, these solutions are relatively easy for an instructor to realize in comparison to the Docker solution presented below. Depending on the competence of the students, they can choose the Linux OS solutions that appeals to them the most.

These solutions also result in large files, primarily due to the core operating system that is included. For example, if one creates an image using Linux Mate and adds OpenMM (includes AmberTools Pytraj), Psi4, Python3 (including matplotlib, scipy, numpy packages), Jupyter Notebooks and removes LibreOffice, the resulting VDI file size is approximately 18 GB. Consequently, this presents a challenge for distributing the image to the students, both regarding where to host the file online and the time needed to transfer the file.

The following sections provides three possible solutions that are based on the idea of providing students a customized OS environment that includes the software need for a given course. These solutions are ordered according to our opinion as the easiest implementation, and in a manner that provides a systematic workflow for realizing all Linux OS-based solutions.

2.1.1 Virtual Machine

VM emulators (e.g. Oracle's VirtualBox [11], QEMU [12]) offers the opportunity to use an OS on an existing operating system through the use of a VM. For illustration, one can create a VM that runs a Linux OS on a computer that is actively running Microsoft Windows. Since the VirtualBox is well documented and straight forward to use, this solution alone provides a good way to remotely providing students software.

1. Install 64-bit Linux OS into a VM by following the VirtualBox instructions.

   We recommend Ubuntu Mate / Linux Mint Mate due to their ease of use, reduced resource usage, and their relatively smaller size.

2. Create a single user (e.g. "student") with a password login.

3. Update system software.

4. Install the software that is required for the course (e.g. miniconda, jupyter-notebook, computational chemistry software).

Once the above workflow is implemented, a Virtual Disk Image (VDI) file will automatically be created whose prefix is the name that was provided during the VM's creation (e.g. vm_name.vdi). The advantages of this solution is that students only need to install a VM emulator onto their computer and then they can use the VDI. This solution is also platform independent and can be implemented on computers running MacOs, Windows and Linux operating systems. One notable disadvantage is that access to the computer's memory is limited, which can cause a performance problem with some software. A second disadvantage is that transferring files from the computer's running OS to the VM can be difficult and result in student frustrations.

2.1.2 Linux Image

Depending on the students' competences, some might want to install the customized OS onto an older unused computer. To enable this, one can convert the VDI file to a Linux image using the following command provided by the VirtualBox:

"vboxmanage clonemedium vm_name.vdi filename.img --format RAW""
where vm_name.vdi is the name of the VDI file and filename.img is the resulting Linux image. One disadvantage of this particular solution is that implementing this on an unused Apple Computers is difficult to achieve and not recommended.

2.1.3 Bootable USB Drive

Both of the above solutions require substantial use of storage space on the student's computer. The less invasive solution is to provide the students a USB drive that can be booted and contains all of the desired software. In order to increase the drive's lifespan, one can reduce the writing of data by the following to modification:

1. Add or change the system's "swappiness=10" in the /etc/sysctl.conf file, and

2. Add "noatime" to /etc/fstab

```
# <file system> <mount point> <type> <options> <dump> <pass>
UUID=be3d875d...48e24  / ext4 noatime,errors=remount-ro 0 1
```

Listing 1. Example of a modified /etc/fstab file that include noatime.

Listing 1 shows an example of a modified Linux /etc/fstab file that includes noatime in the options column. There are several possibilities for installing the customized OS image onto the USB drive. For example, one could use the cross-platform Etcher software [13]. Alternatively, most Linux distributions already enable you to create a bootable USB drive, including the following commmand line example statement:

"sudo dd bs=4M if=path/to/filename.img of=/dev/sdXX conv=fdatasync status=progress"
where sdXX should be replaced with the location of your USB drive. The location can be found using the "lsblk" command line statement (e.g. sdXX = sde1).

Once the student has obtained the USB drive (or created one themselves), they should use the computer's boot menu to choose the USB drive. This menu is often accessed by pressing a specific keyboard key during boot. Alternatively, one can edit the motherboard's BIOS settings to always boot from the USB drive first, and then reboot with the drive inserted.

Ideally, the most recent version of USB drive technology (e.g. version 3.0) should be used in order to maximize the communication speed. Students who have the most recent USB ports on their computer will benefit from this consideration. Due to backwards compatibility, students that have older hardware ports will also be able to use the external drive, but without the speed benefits. As in the above solution, booting Apple hardware using a USB drive that contains a Linux OS is problematic and not recommended.

A possible source of frustration might occur during installation arises from not having enough disk space needed for installing the image. While a product may be advertised to have 32 GB of space, the usable writing space is less (e.g. $\sim$29 GB). Consequently, we recommend that prior to writing the image, one simply inserts the USB drive and verify how much writable space exists on the device.

As a final note, the USB drive is *not* secure. If one simply inserts the USB drive into an already booted computer, then they have access to all of its files. In other words, the protection provided by the student user account and password described above only works if one boots from the USB drive.

2.2 Docker Container

Distributing software using a Docker container is distinctly different than the above solution. Arguably, Docker is the easiest solution for students to implement. It also requires less disk space, and has existing platforms for distributing the resulting container to the students. A disadvantage is that an instructor may find it difficult to create a working container due to the complexity of Docker. However, for MacOS and Window users there are helpful frontends that assist in creating containers. Minimalists examples of a Dockerfile, environment YAML file and docker-compose YAML are provided in Listings 2, 3 and 4 that demonstrate a possible workflow involving computational chemistry software and Jupyter Notebooks.

```
# Choose your desired base image
FROM jupyter/minimal-notebook:latest

# name your environment and choose python 3.x version
#ARG conda_env=compchem_students
# ARG py_ver=3.7

# alternatively, you can comment out the lines above and uncomment those below
# if you'd prefer to use a YAML file present in the docker build context
COPY --chown=${NB_UID}:${NB_GID} environment.yml /home/$NB_USER/tmp/

# https://github.com/ContinuumIO/docker-images/issues/89#issuecomment-566208852
# easy way to extract the name
RUN export conda_env=$(head -1 /home/$NB_USER/tmp/environment.yml | cut -d' ' -f2)

RUN cd /home/$NB_USER/tmp/ && \
    conda env create -p $CONDA_DIR/envs/$conda_env -f environment.yml && \
    conda clean --all -f -y

# create Python 3.x environment and link it to jupyter
RUN $CONDA_DIR/envs/${conda_env}/bin/python -m ipykernel \
    install --user --name=${conda_env} && \
    fix-permissions $CONDA_DIR && \
    fix-permissions /home/$NB_USER

# any additional pip installs can be added by uncommenting the following line
# RUN £CONDA_DIR/envs/£{conda_env}/bin/pip install

# Tell the shell to run the new env at start
RUN echo "conda activate ${conda_env}" >> ~/.bashrc

# Prepend conda environment to path
ENV PATH $CONDA_DIR/envs/${conda_env}/bin:$PATH

# If you want this environment to be the default one, uncomment the following line:
ENV CONDA_DEFAULT_ENV ${conda_env}

# Add additional persistent files needed for course (e.g. example, tutorials)
# This should be one of the later steps to prevent unnecessary rebuilds
COPY --chown=${NB_UID}:${NB_GID} exercise_amber_peptide_water_md/ /home/${NB_USER}/documents

# Workaround to persist course relevant files into the volume
USER root
RUN mkdir '/usr/local/bin/start-notebook.d'
RUN echo "cp -rpn /home/${NB_USER}/documents /home/${NB_USER}/work" > \
  /usr/local/bin/start-notebook.d/copy-documents.sh
RUN chmod +x /usr/local/bin/start-notebook.d/copy-documents.sh
USER $NB_UID
```

Listing 2. Example of a Dockerfile, following the guidelines of Nüst et al. [14]. The file is named ”Dockerfile-method1,” and is referenced in Listing 4. Currently, the shown file is suitable for compiling the docker image from scratch using Listing 2-4.

```
name: compchem_students
channels:
  - defaults
dependencies:
  - ipykernel
  - matplotlib
  - openmm
#  - ambertools
#  - pip
prefix: /mnt/sdc1/miniconda3/envs/compchem_students
```

Listing 3. Example of a environment.yml that install three additional software.

```
version: '3.8'
services:
  compchem-jupyter:
    user: ${UID}:${GID}
    build:
      context: .
      ## The following give you control if using multiple Dockerfiles
      dockerfile: ./Dockerfile-method1
    #image: instructor_name/compchem-jupyter:latest

    container_name: compchem-jupyter

    # The way to interact with the Jupyter
    # in this case open a browser and search for
    # http://127.0.0.1:8888
    # Host:Container
    ports:
    - 8888:8888

    # The persistent folder
    # Host:Container (jovyan is the default within Docker - it is best to use it)
    volumes:
      - ./my_notebooks:/home/jovyan/work

    # restarts the container on crash
    restart: unless-stopped

    environment:
      # Optional - gives the ability to install extensions
      #     (but they will be deleted by the next update)
      - RESTARTABLE=yes

      # Optional if student want to use the new lab version of Jupyter
      - JUPYTER_ENABLE_LAB=yes
```

Listing 4. Example of a docker-compose.yml. Note that "/home/username/" should be replaced by the directory path that docker user.

The example provided in Listing 2-4 will create a container that includes Python3, matplotlib library, Jupyter, and Openmm software. Students can save their work in the container's folder named "work." This folder is linked to another folder named "my_notebooks", which is located within the directory that contains the Listings 2-4 files. This later folder is associated with the computer's OS and is therefore accessible when the container is shutdown.

There are two approaches for enabling the students access to the instructor's docker image. The first is to simply provide them the modified Listing 2-4 files and let the student compile the container. Alternative, a more efficient approach is to host the instructor's image on a free online platforms (e.g. Docker Hub [15]). Once hosted, the students can download the image and run the container on their own computer. In that case, one needs to comment (i.e. add # sign to to statement) the line "dockerfile: ./Dockerfile-method1" in Listing 4, and uncomment the line that follows it. Note, that the student needs to replace the "instructor_name" with the appropriate account name on the online platform. To install the docker image and to start the container, one simply executes the command line "docker-compose up" in a desired folder that you will work from.

3 Discussion

According to a survey performed during this past year, students acknowledge the importance of having the instructor providing an organized content for their on-line learning, which can be achieved, in part, by establishing good learning spaces [5]. Crucial in many courses and laboratories is the student usage of specialized software. A significant usage of software is for data analysis and writing, with a wide range of advance usages that depend on the particular course (e.g. simulations). In order to establish a good remote learning experience, educators should provide software environments that are customized according to the learning goals of their course. To address this we developed several solution that enable students to have a customized software environment that is similar to what they would have when physically at their school.

However, these solutions should not require students to install an OS alongside their existing one (i.e. dual booting) or be used as a complete replacement. Such approaches are risky because a) the students might not be competent in installing a new OS from scratch (e.g. setting up a partition table) or dual-boot system, b) the possibility of disrupting an existing software environment that is needed in other classes or by other family members, and c) the importance of maintaining a working computer when isolated (e.g. when societal restrictions are in place). Therefore, one of our aims in developing the solutions herein is that they require a minimal impact on the student's existing computer. For the virtual machine and docker container solutions, the students only need to install either a VM emulators or the Docker software. Once these are installed, then the instructor provided VDI image or container is loaded. For the bootable USB drive solution, the students need to either choose the USB drive from the computer's boot menu, or adjust the boot priority of their computer's BIOS. While these are low-risk actions, we *strongly* advise the students to create a backup of their computer before applying our workflows.

As in scientific research (e.g. the usage of Docker [16]), the concept of reproducibility is important in teaching. Reproducibility in the context of teaching that involves software requires that the students use the same software versions and libraries as the instructor. This enables the students to reproduce the examples and workflows exactly as they are taught. When the software versions used becomes disjointed, then confusion arises due to the addition, change (e.g. refactoring input flags) or removal (e.g. obsolete functions) of features. This quickly leads to confusion and frustration by the students, resulting in a loss of interest and focus. Furthermore, a uniform software environment significantly simplifies the solving of problems that arise during the course. Of the solutions presented herein, the usage of Docker to update (e.g. correcting a problem, adding features) and redistribute the software environment provides the least amount of effort for the students to update their local version.

In addition to the above aspects of providing a consistent reproducible software environment that has a low-risk implementation, it is important that the solutions include file "persistence." File persistence means that a file created within the VM, Linux OS, bootable USB drive, or Docker container continues to exist after those processes are shutdown. This is the normal working condition when using your computer. The solutions herein include file persistent, allowing students to create and modify files that can be used again when reinvoking the solution (e.g. rebooting the VM). In the case of the Docker container, a few additional step are needed and conveyed to the students to ensure file persistence – for example, setting up a persistent volume in the docker-compose configuration file (see Listing 4).

A final consideration is how to provide customized tutorials, examples and data files. For example, one might want to include data that is obtained from laboratory equipment (e.g. NMR data), enabling the students to do analysis. One can include such files into the solutions presented here. However, using a git repository might be more convenient since it would allow for specific files to be updated quickly and easily.

4 Conclusion

Innovation is an important element for moving society forward. The necessity of physically separating students and instructors has lead to a global awareness of the challenges involved in remote learning. Critical components of 21$^{\text{st}}$ century remote learning – from the first grade to graduate level – are computers, the Internet and software. In many courses, the instructor relies on specific, specialized software that are need for imparting knowledge to their students. When courses are held in-person at school, a uniform software environment is normally provided (e.g. computer labs). Herein we have presented possible solutions for how instructors could provide their students a uniform software environment when teaching remotely. Specifically, a instructor could create a customized Linux OS that can be realized using a VM, installed onto a computer (e.g. using an unused computer), or by creating a bootable USB drive. A fourth, distinct solution is the creation and implementation of a Docker container.

In addition to remote learning, institutions that have spatially separated campuses could also benefit from these proposed solutions. For such institutions, enabling students from one campus to attend courses offered at another campus can pose significant problems due to distance and time constraints. Providing a uniform software environment that can be accessed by diverse (student) computers and campuses facilitates better course cohesion. When executed smoothly, the focus will remain on a course's material, rather than troubleshooting software issues, which improves the students' confidence of the instructor and the institution.

Solutions are never perfect. When using the solutions herein, one issues that might occur is if a student uses a very old hardware. Most modern (i.e. last 10 years) computers have 64-bit processors. If a corresponding 64-bit Linux OS is used, then compatibility issues may arise when being used with a 32-bit processor computer. Furthermore, if a student only has access to an Apple Computer, then they are limited to using the VM solution, which also restricts them in the amount of RAM that accessed by the software. And finally, the use of commercial software presents an unsolved legal problem, and thus we only recommend the use of open-source software in developing the customized software environment.

An added benefit implementing a software environment for remote usage is that the students are not time-restricted for using the software. Access to computers and software at school is often highly controlled, with buildings and computer labs open for limited times during the week. Thus, using the software solution provided herein enable the students to self-pace their learning more, which ideally would maximize their productivity and learning. As present in remote and online learning, educators should consider the disparity in the quality of computer hardware and Internet connectivity that occurs in their student population. Consequently, educators are encouraged to formulate solutions for when students usage of the customize software environment is hampered due to their personal computer resources.

References

[1] O. B. Adedoyin and E. Soykan, "Covid-19 pandemic and online learning: the challenges and opportunities," *Interactive Learning Environments*, pp. 1–13, Published online: 02 Sep 2020. https://doi.org/10.1080/10494820.2020.1813180.

[2] N. Dietrich, K. Kentheswaran, A. Ahmadi, J. Teychené, Y. Bessière, S. Alfenore, S. Laborie, D. Bastoul, K. Loubière, C. Guigui, M. Sperandio, L. Barna, E. Paul, C. Cabassud, A. Liné, and G. Hébrard, "Attempts, successes, and failures of distance learning in the time of covid-19," *Journal of Chemical Education*, vol. 97, no. 9, pp. 2448–2457, 2020.

[3] C. Rapanta, L. Botturi, P. Goodyear, L. Guàrdia, and M. Koole, "Online university teaching during and after the covid-19 crisis: Refocusing teacher presence and learning activity," *Postdigital Science and Education*, vol. 2, no. 3, pp. 923–945, 2020.

[4] P. Li, "Exploring virtual environments in a decentralized lab," *SIGITE Research in IT*, vol. 6, no. 1, pp. 4–10, 2009.

[5] K. A. Jeffery and C. F. Bauer, “Students’ responses to emergency remote online teaching reveal critical factors for all teaching,” *Journal of Chemical Education*, vol. 97, no. 9, pp. 2472–2485, 2020.

[6] C. Roddy, D. L. Amiet, J. Chung, C. Holt, L. Shaw, S. McKenzie, F. Garivaldis, J. M. Lodge, and M. E. Mundy, “Applying best practice online learning, teaching, and support to intensive online environments: An integrative review,” *Frontiers in Education*, vol. 2, p. 59, 2017.

[7] Docker, “Docker,” website accessed on March 30, 2021. https://www.docker.com/.

[8] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, R. P. Wiewiora, B. R. Brooks, and V. S. Pande, “OpenMM 7: Rapid development of high performance algorithms for molecular dynamics,” *PLOS Computational Biology*, vol. 13, no. 7, pp. 1–17, 2017.

[9] D. Case, I. Ben-Shalom, S. Brozell, D. Cerutti, T. Cheatham, III, V. Cruzeiro, T. Darden, R. Duke, D. Ghoreishi, G. Giambasu, T. Giese, M. Gilson, H. Gohlke, A. Goetz, D. Greene, R. Harris, N. Homeyer, Y. Huang, S. Izadi, A. Kovalenko, R. Krasny, T. Kurtzman, T. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, V. Man, D. Mermelstein, K. Merz, Y. Miao, G. Monard, C. Nguyen, H. Nguyen, A. Onufriev, F. Pan, R. Qi, D. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C. Simmerling, J. Smith, J. Swails, R. Walker, J. Wang, H. Wei, L. Wilson, R. Wolf, X. Wu, L. Xiao, Y. Xiong, D. York, P. K. D. Case, R. Betz, W. Botello-Smith, D. Cerutti, T. Cheatham and III, T. Darden, R. Duke, T. Giese, H. Gohlke, A. Goetz, N. Homeyer, S. Izadi, P. Janowski, J. Kaus, A. Kovalenko, T. Lee, S. LeGrand, P. Li, C. Lin, T. Luchko, R. Luo, B. Madej, D. Mermelstein, K. Merz, G. Monard, H. Nguyen, H. Nguyen, I. Omelyan, A. Onufriev, D. Roe, A. Roitberg, C. Sagui, C. Simmerling, J. Swails, R. Walker, J. Wang, R. Wolf, X. Wu, L. Xiao, D. York, and P. Kollman, “AMBER19.” http://ambermd.org, 2019. University of California, San Francisco.

[10] R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, U. Bozkaya, A. Y. Sokolov, R. Di Remigio, R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, “Psi4 1.1: An open-source electronic structure program emphasizing automation, advanced libraries, and interoperability,” *J. Chem. Theory Comput.*, vol. 13, no. 7, pp. 3185–3197, 2017.

[11] Oracle Corporation, “Oracle VM VirtualBox,” website accessed on March 11, 2021. https://www.virtualbox.org.

[12] “QEMU,” website accessed on March 31, 2021. https://www.qemu.org/.

[13] Balena, “Etcher,” website accessed on March 30, 2021. https://www.balena.io/etcher.

[14] D. Nüst, V. Sochat, B. Marwick, S. J. Eglen, T. Head, T. Hirst, and B. D. Evans, “Ten simple rules for writing dockerfiles for reproducible data science,” *PLOS Computational Biology*, vol. 16, pp. 1–24, 11 2020.

[15] Docker, “Docker Hub,” website accessed on March 30, 2021. https://hub.docker.com/.

[16] J. Cito, V. Ferme, and H. C. Gall, “Using docker containers to improve reproducibility in software and web engineering research,” in *Web Engineering* (A. Bozzon, P. Cudre-Maroux, and C. Pautasso, eds.), (Cham), pp. 609–612, Springer International Publishing, 2016.