MWSUG 2019 - Paper AL115

Deploying SAS® Viya® to Docker – a practical guide for data scientists

Alan Zablocki, Ph.D., RedMane Technology, Chicago, IL

ABSTRACT

Recently, SAS® has provided the ability to package and deploy SAS® Viya® software in Docker containers. Users with a valid SAS Viya order have a choice between using the Docker image repository hosted by SAS or manually building the Docker image using SAS container recipes, an open source GitHub project. In this paper, we use SAS container recipes to build a customized SAS Viya Docker image with access to a range of licensed SAS products such as SAS® Visual Statistics and SAS® Visual Data Mining and Machine Learning. We guide the reader through the entire process of building and deploying a local SAS Viya Docker image, from configuring local storage and data persistence to adding support for Jupyter Notebook, Python and R.

INTRODUCTION

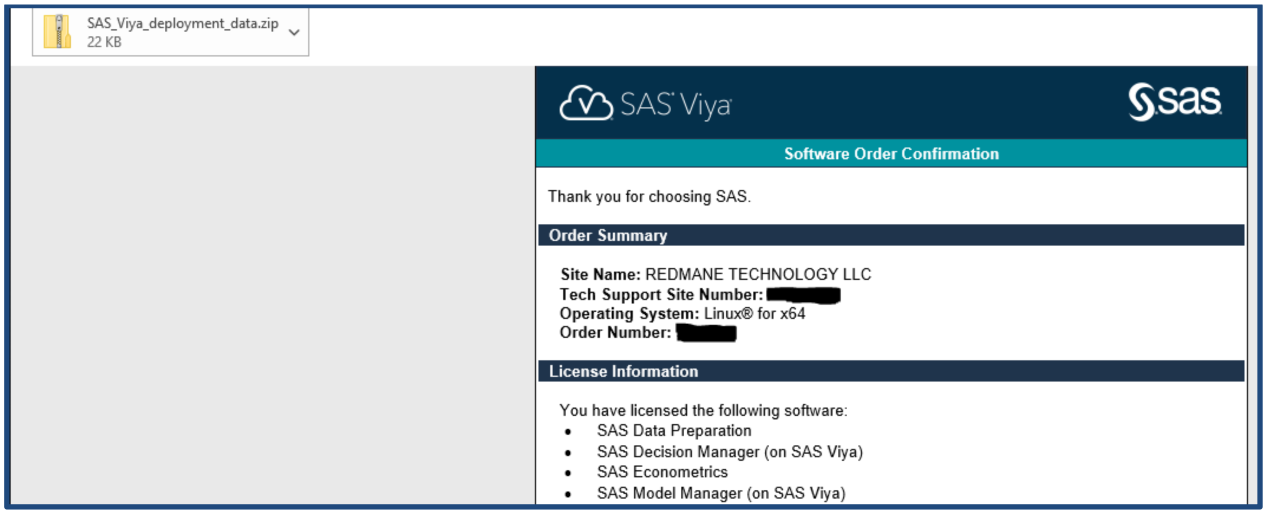
Installing SAS Viya is a lengthy and complicated process. An open source project on GitHub called SAS container recipes (*https://github.com/sassoftware/sas-container-recipes*) makes it easier to start using SAS Viya. This is achieved with Docker, which simplifies creating, deploying and running applications in execution environments called containers. SAS Viya software can be deployed as a single programming-only Docker container, or across many containers, which are then orchestrated (managed) with Kubernetes. Although we mention the full SAS Viya deployment briefly, a complete discussion of multiple container orchestration using Kubernetes is beyond the scope of this paper and will be covered in a future paper. In this paper, we focus our discussion on building a single programming-only Docker image, and running it locally. In our companion paper, “Deploying SAS® Viya® Docker images to the cloud - a step by step guide”, we show how to deploy SAS Viya Docker image using Azure Cloud.

CONFIGURATION

In this section, we describe the system requirements and the set-up process to ensure a successful SAS Viya Docker image deployment.

SOFTWARE ORDER ZIP

Our SAS Software Order is for SAS Viya 3.4 and gives us access to SAS products such as SAS Visual Data Mining and Machine Learning. The Software Order Email (SOE) includes license information as well as a zip file with all the license files (see Figure 1).



Software Order Confirmation

Thank you for choosing SAS.

Order Summary

Site Name: REDMANE TECHNOLOGY LLC
Tech Support Site Number: [redacted]
Operating System: Linux® for x64
Order Number: [redacted]

License Information

You have licensed the following software:

- SAS Data Preparation
- SAS Decision Manager (on SAS Viya)
- SAS Econometrics
- SAS Model Manager (on SAS Viya)

Figure 1: The SOE email with the software order zip file.

The SOE zip contains two key files in the licenses folder: a *.jwt* file and a *.txt* file with the ordered SAS products. If you are missing the *.jwt* file, you should update your order. If you proceed with the build, you will likely encounter an error when parsing the contents of the SOE zip. It may be possible to carry out an installation without the *.jwt* file, but we did not test this. You can refer to this GitHub issue *https://github.com/sassoftware/sas-container-recipes/issues/17* for a possible workaround, but updating the order is the best choice. The SOE zip allows you to build an image and download the software order to create your own software mirror repository.

DOCKER

Building the SAS Viya image with SAS container recipes requires a supported version of Docker-CE (Community Edition). In this paper, we use Docker CE Version 18.09.6 (see Appendix). For installation details and system requirements see *https://docs.docker.com/install/linux/docker-ce/centos/*.

THE LINUX HOST

SAS container recipes support RHEL and CentOS for single or multiple image builds. SUSE Linux is supported for single image builds only. Ubuntu is not supported. You can build the Docker image on a local machine with RedHat or CentOS Linux installed, use a Virtual Machine (VM) in either a Windows or Linux host, or use a VM hosted in the cloud. In this paper, we use a CentOS VM on a Windows10 host. The CentOS VM worked out of the box with Oracle's VirtualBox V5.2.28 (we used CentOS-7-x86_64-DVD-1810.iso).

Hard Disk Space

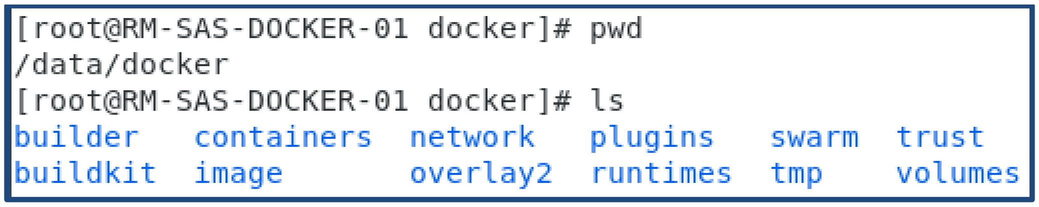
Whether you choose to do a Docker build on a local machine or inside a VM, we recommend that you assign plenty of disk space for the build process. While the amount of space needed for the Docker image(s) will vary depending on your SAS Software Order, we found that our Software Order created a 25GB programming-only SAS Viya Docker image (including the Jupyter Python addon). Our full build, which was contained 32 Docker images, took up 113GB.

The built images are stored in */var/lib/docker*, and therefore the root partition must be large enough to accommodate not only the size of the final image(s), but also any intermediate images and/or downloads that occur. One possible solution is to re-point the */var/lib/docker* directory using a symlink to a larger space such as */home*. We chose a different approach.

We used three separate partitions on two different disks. Our root (/) and */home* partitions were 120GB and 104GB respectively. This was more than enough to store our single image builds and our entire software download (45GB). To perform the full build, we added a new disk and assigned it the */data* partition. We then re-pointed Docker to save images to /data/docker, instead of */var/lib/docker*. To change where Docker saves images, create the file */etc/docker/daemon.json* (if it doesn't exist) as a root user and add the *"data-root"* flag:

```
{
    "data-root": "/data/docker"
}
```

Now all the images will be stored in /data/docker, which has the same folder structure as */var/lib/docker* (see Figure 2). You do not have to delete the contents of */var/lib/docker*. Using the *daemon.json* file allows you to point "*data-root*" back to "*/var/lib/docker*" to launch images that were built previously.



```
[root@RM-SAS-DOCKER-01 docker]# pwd
/data/docker
[root@RM-SAS-DOCKER-01 docker]# ls
builder    containers  network   plugins   swarm  trust
buildkit   image       overlay2  runtimes  tmp    volumes
```

Figure 2: The new "*/var/lib/docker*" on the */data* partition. It has its own folders, and the previously built images are stored in */var/lib/docker*. You can swap between the two locations to run old and new images.

We highly recommend using a mirror repository since there is a limit on the number of order downloads from the SAS servers (we believe the default to be 5). If you are going to use your own mirror repository, you will also need a large amount of disk space for the initial download. By default, the files are saved to the sas_repos folder in the user's */home* directory. Our sas_repos folder was 45GB in size. We cover setting up the mirror repository in a later section.

For a single programming-only image, we recommend creating a VM with a root (/) partition of at least 60GB (about 10GB is taken up by the OS itself) and depending on whether you plan to use a mirror repository, a */home* partition of at least 50GB also.

SAS AND CONTAINERS

There are two ways to deploy SAS Viya with Docker; using a predefined Docker image and using SAS container recipes. SAS container recipes is an open source GitHub project by SAS, which makes building a SAS Viya Docker container much simpler. In this paper, we will build all our containers using the SAS container recipes found at *https://github.com/sassoftware/sas-container-recipes* release version v19m05. See *https://github.com/sassoftware/sas-container-recipes/wiki/Introduction* for more details.

Setting up the SAS container recipes repository

Download the zipped repository from *https://github.com/sassoftware/sas-container-recipes*. In the command line, change to your */home* directory and uncompress the file using:

```
unzip sas-container-recipes-master.zip
```

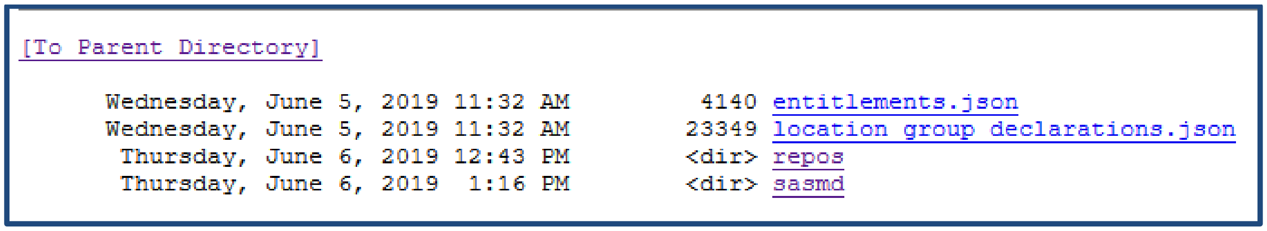
You can also run *git clone* *https://github.com/sassoftware/sas-container-recipes.git*. Then, place the SOE zip inside the folder and leave it unzipped.

Setting up a mirror repository

To setup a mirror repository of your software order, follow the instructions in the SAS Viya 3.4 for Linux Deployment Guide (section: SAS Mirror Manager and the Mirror Repository) accessible here *https://go.documentation.sas.com/?docsetId=dplyml0phy0lax&docsetTarget=p1ilrw734naazfn119i2rqik91r0.htm&docsetVersion=3.4&locale=en*. After downloading and uncompressing the Mirror Manager, issue the following command:

```
./mirrormgr mirror --deployment-data /home/azablocki/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --platform x64-redhat-linux-6 –latest
```

The order repository is then automatically saved inside a sas_repos folder in the user's */home* directory. We strongly recommend hosting the repository on a separate server. We used Internet Information Services (IIS), since our Centos VM is hosted on Windows. Once you upload the data to a remote server, you should be able to view the contents at *http://ip-address/sas_repos* (see Figure 3).



```
[To Parent Directory]

     Wednesday, June 5, 2019 11:32 AM         4140 entitlements.json
     Wednesday, June 5, 2019 11:32 AM        23349 location group declarations.json
      Thursday, June 6, 2019 12:43 PM        <dir> repos
      Thursday, June 6, 2019  1:16 PM        <dir> sasmd
```

Figure 3: Software mirror repository contents.

Before starting a build, be sure to check that the entitlements.json file is accessible either by download or that it can be viewed in the browser. We encountered some errors due to missing MIME types and had to add the MIME types for *.json* and *.odd* files. In addition, the server was not resolving the address for a file with a special character such as the '+' sign. The problematic file was:

sas_repos/repos/shipped/va/104/va-104-x64_redhat_linux_6-yum/Packages/s/sas-cpp-libstdc++6-6.0.95404-20180510.1525974525.x86_64.rpm

To fix this you may have to change options in IIS Request Filtering. If you do not have TLS setup, you will see a warning, but this does not prevent the build from completing. Finally, you may want to use the server IP address instead of a fully qualified domain name (FQDN) in case you encounter Domain Name System (DNS) resolution issues.

BUILDING A SAS VIYA DOCKER IMAGE

In this section, we discuss building the first image to test the environment setup. We then show how to run the basic image locally. Finally, we show in detail how to extend the image by installing various addons. For the purpose of this introductory guide, we will cover the auth-demo addon, and the ide-jupyter-python3 addon, with an additional custom addition for R support.

BUIDLING THE BASIC IMAGE

To build a single programming-only image with the default sasdemo user, and to use the SAS download servers use:

```
./build.sh --type single --zip /home/admin/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --addons "auth-demo"
```

Alternatively, you can add the --mirror-url flag to point to your own hosted software repository. Replace <ip-address> with your IP address:

```
./build.sh --type single --zip /home/admin/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --mirror-url "http://<ip-
address>/sas_repos/" --addons "auth-demo"
```

The build should proceed without any issues and can take up to two hours to complete depending on your machine and the internet connection. Once the build completes, you will see the image tag and instructions on how to run the image. You can use the *docker images* command to see a list of images, their names, tags and their size. For detailed descriptions of the build options, type:

```
./build.sh --help
```

Issues we encountered while doing a basic build

Initially, we ran into two problems that prevented the build from completing. They were:

- Parsing all content from the SOE zip - if your software order is not new, you may find that you do not have all the files that the recipe code expects. You will need to update your order. See this GitHub issue for more details: https://github.com/sassoftware/sas-container-recipes/issues/17.
- Disk space - the */var/lib/docker* directory grows quickly in size as the image is being built. If your local machine or the VM disk runs out of disk space, the build will exit.

DEPLOYING THE BASIC IMAGE

In previous versions of the SAS container recipes, the build returned a command to launch the Docker image. The command was:

```
docker run --detach --rm --env CASENV_CAS_VIRTUAL_HOST=eecb5d45bc1d \
--env CASENV_CAS_VIRTUAL_PORT=8081 --publish-all --publish 8081:80 \
--name sas-viya-single-programming-only --hostname eecb5d45bc1d \
sas-viya-single-programming-only:19.04.0-20190516160342-no-git-sha
```

In the current version of the recipes, there is a new method to deploy the single programming-only SAS Viya image, and it uses a launch bash script. In the *sas-container-recipes-master* folder, create a new directory called *run*. Copy the *example_launchsas.sh* file from the *samples* directory to the newly created *run* directory and rename *to launchsas_authdemo.sh*. This script takes as input the image tag, maps ports, creates the local directories and maps them to the corresponding folders inside the Docker image for persistent storage. We show the contents of the launch script in Figure 4.

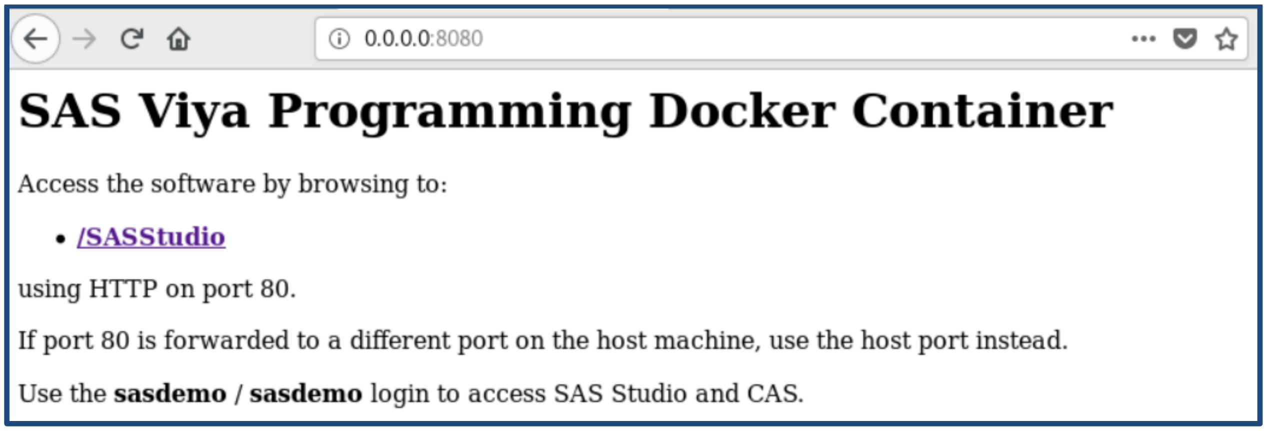
```
#! /bin/bash -e

SAS_CONTAINER_NAME=sas-viya-single-programming-only
IMAGE=$SAS_CONTAINER_NAME:19.05.0-20190611120104-no-git-sha
SAS_HTTP_PORT=8080
SAS_HTTPS_PORT=8443

mkdir -p ${PWD}/sasinside      # Configuration
mkdir -p ${PWD}/sasdemo        # In some cases persist user data
mkdir -p ${PWD}/cas/data       # Persist CAS data
mkdir -p ${PWD}/cas/cache      # Allow for writing temporary files to a different location
mkdir -p ${PWD}/cas/permstore  # Persist CAS permissions
mkdir -p ${PWD}/home  # home dir

run_args="
--name=$SAS_CONTAINER_NAME
--rm
--hostname $SAS_CONTAINER_NAME
--env RUN_MODE=developer
--env CASENV_ADMIN_USER=sasdemo
--env CASENV_CAS_VIRTUAL_HOST=$(hostname -f)
--env CASENV_CAS_VIRTUAL_PORT=${SAS_HTTPS_PORT}
--env CASENV_CASDATADIR=/cas/data
--env CASENV_CASPERMSTORE=/cas/permstore
--publish-all
--publish 5570:5570
--publish ${SAS_HTTP_PORT}:80
--publish ${SAS_HTTPS_PORT}:443
--volume ${PWD}/sasinside:/sasinside
--volume ${PWD}/sasdemo:/data
--volume ${PWD}/home:/home
--volume ${PWD}/cas/data:/cas/data
--volume ${PWD}/cas/cache:/cas/cache
--volume ${PWD}/cas/permstore:/cas/permstore"

# Run in detached mode
docker run --detach ${run_args} $IMAGE "$@"

# For debugging startup, comment out the detached mode command and uncomment the following

#docker run --interactive --tty ${run_args} $IMAGE "$@"
```

Figure 4: The bash script to launch a container with persistent storage.

When you run the launch script, your *run* directory will be populated by the directories: *cas, sasdemo, and sasinside*. You will notice that we also created a *home* directory and mapped it as a mounted volume to the */home* directory in the Docker container. Once you launch the image, you will also have a *home* directory inside the *run* folder. To deploy your Docker image, run:

```
./launchsas.sh
```

You will see a short one-line output. If you launch the image with the interactive mode as in the script, you will see a more verbose output. Use the *docker ps* command to view the running container, its image ID, tag, name, and server address. After a short moment, you should be able to access the image at the addresses in the output, usually 0.0.0.0:8080 as well as localhost:8080 or 127.0.0.1:8080 as shown in Figure 5.


0.0.0.0:8080

SAS Viya Programming Docker Container

Access the software by browsing to:

- /SASStudio

using HTTP on port 80.

If port 80 is forwarded to a different port on the host machine, use the host port instead.

Use the **sasdemo / sasdemo** login to access SAS Studio and CAS.

Figure 5: SAS Viya Docker host page with the link to SAS Studio on port 80.

Click on the SAS Studio link and log into you SAS Viya 3.4 Session in Docker as shown in Figure 6.

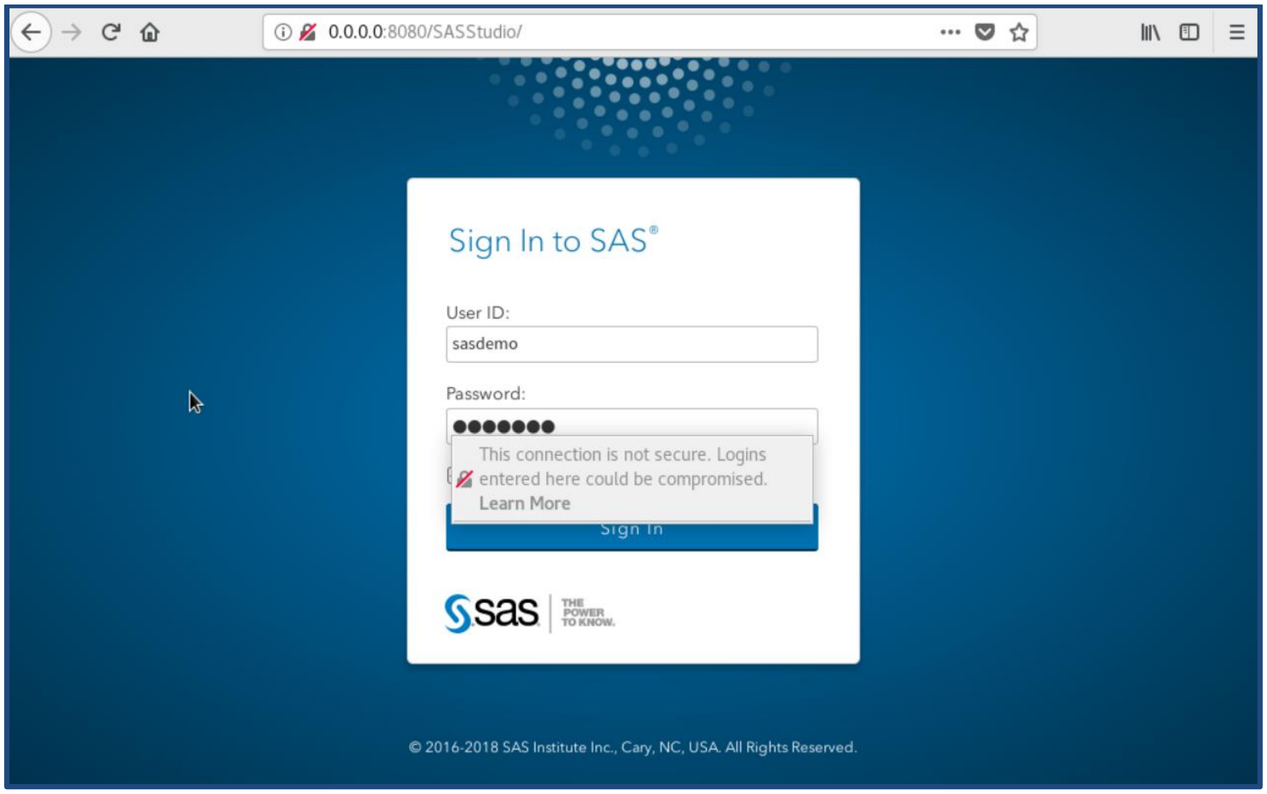
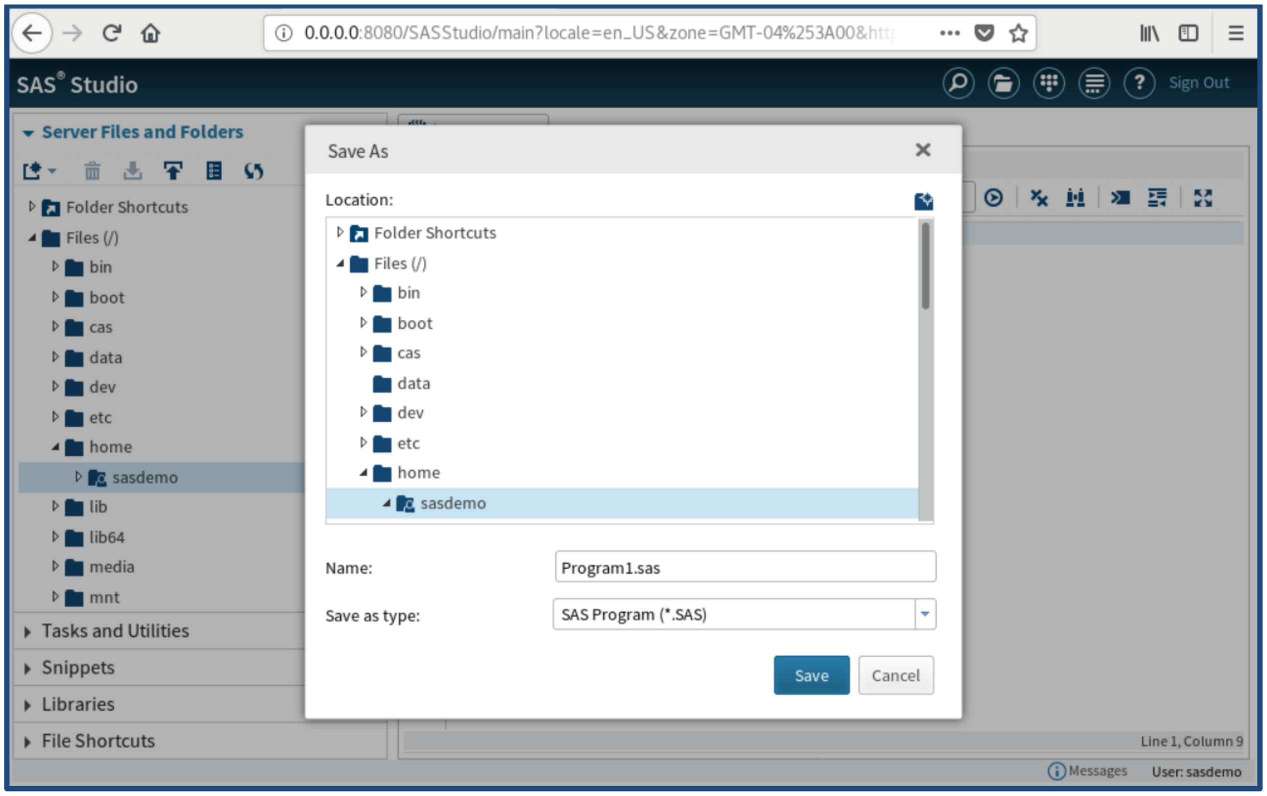

0.0.0.0:8080/SASStudio/
Sign In to SAS®
User ID:
sasdemo
Password:
This connection is not secure. Logins entered here could be compromised.
Learn More
Sign In
sas THE POWER TO KNOW.
© 2016-2018 SAS Institute Inc., Cary, NC, USA. All Rights Reserved.

Figure 6: SAS Studio login screen, with the default user and password.

Once you log in, you are taken to SAS Studio where you can start programming in pure SAS. The various SAS products can be found under Tasks and Utilities.

Testing persistent storage

In the launch script shown in Figure 4, you mapped several volumes between your Docker file system and your local VM. By default, you won't be able to write to all of them as a non-root user. You will be able to write to */home/sasdemo*. In Figure 7, we save a short script to */home/sasdemo* in the Docker container to test persistent storage.

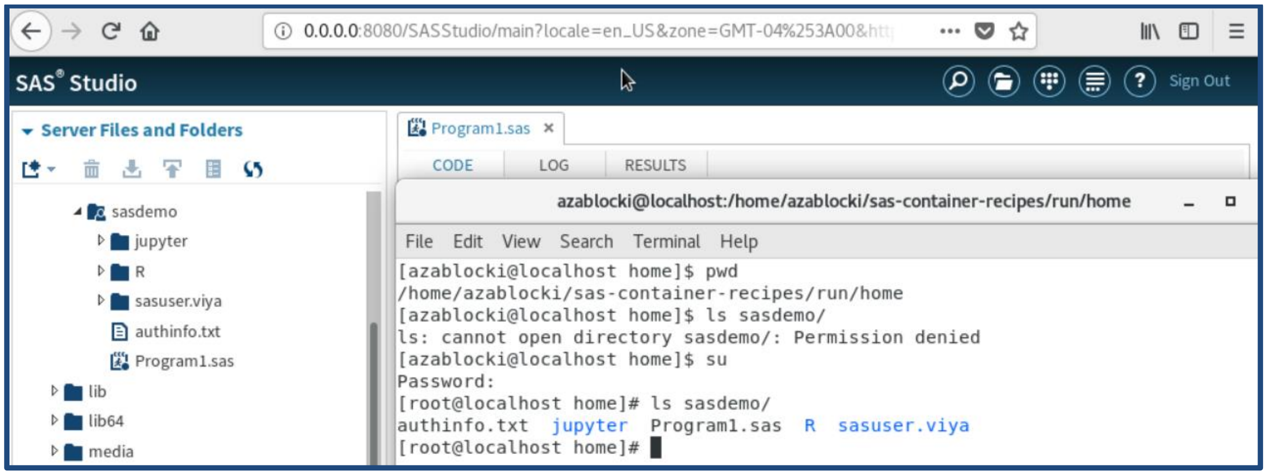


Name: Program1.sas

Save as type: SAS Program (*.SAS)

Figure 7: Saving files in the Docker container under *\/home/sasdemo*.

Since you mapped */home* in the Docker container to your *~/sas-container-recipes/run/home/* folder, you can now view the file in your VM. Navigate to the *~/sas-container-recipes/run/home/* folder (see Figure 8).



```
[azablocki@localhost home]$ pwd
/home/azablocki/sas-container-recipes/run/home
[azablocki@localhost home]$ ls sasdemo/
ls: cannot open directory sasdemo/: Permission denied
[azablocki@localhost home]$ su
Password:
[root@localhost home]# ls sasdemo/
authinfo.txt  jupyter  Program1.sas  R  sasuser.viya
[root@localhost home]# 
```

Figure 8: Access files saved inside the Docker container on your local VM.

If you try to list the contents of */home/sasdemo*, you will not be able to do so, even though the files are there. To see and open the files, you must change to sudo user or root. This is a result of the default configuration used during the build.

ADDING PYTHON AND JUPYTER NOTEBOOK SUPPORT

The SAS container recipes project has an addon for Python and Jupyter Notebook support, and the addon is called ide-jupyter-python3. This addon installs Python3.6, pip, and Jupyter system wide. For more flexibility, a user can choose to modify the Python install in this addon, and use Anaconda to install

Python, pip, and Jupyter. In this paper, we install Python system wide. In a later section, we will show how to modify this addon to add support for R, RStudio server, and the R kernel for Jupyter notebook. To add Python and Jupyter Notebook support to your auth-demo image, use the same command as before, but now provide a space-separated list to the *--addon* flag:

```
./build.sh --type single --zip /home/admin/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --addons "auth-demo ide-jupyter-
python3"
```

If you are using your own mirror repository, add the *--mirror-url* flag as shown below replacing *<ip-address>* with your IP address:

```
./build.sh --type single --zip /home/admin/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --mirror-url "http://ip-
address/sas_repos/" --addons "auth-demo ide-jupyter-python3"
```

The addon build took around 20 minutes on our VM. If you modify the Dockerfile(s) in *addons/ide-jupyter-python3*, the build process will pick up on the changes. We use this method to incrementally add R support by modifying the Dockerfile file in the *addons/ide-jupyter-python3* folder.

INSTALLING R, RSTUDIO SERVER, AND JUPYTER R KERNEL

You can add support for R to your SAS Viya image by modifying the Dockerfile in *addons/ide-jupyter-python3*. Once you install R, you can add RStudio Server or the R Kernel for Jupyter notebook, or both. Here we show how to install both.

We found it useful to test the installation of R, RStudio Server and the R Kernel locally first. We recommend that you do the same on your local machine or in the local VM, to ensure a successful R install during the Docker image build stage. R packages have a lot of dependencies, and sometimes package installations fail when those dependencies are missing or if they cannot be installed. This is often the case when there is a brand new version of R available, when you update your version of R, or if there are missing system libraries (that you have to manually install with yum on RedHat/CentOS or apt-get in Ubuntu/Debian). Note that installing R packages can take a while, which will increase the time it takes to build the Docker image too. For this reason, we recommend you install packages over an Ethernet connection.

INSTALLING R LOCALLY

To install R locally, type *su* in the command line to make sure you are the root user, and run the following commands:

```
yum install -y openssl-devel && \
yum install -y libcurl-devel && \
yum install -y gsl-devel && \
yum install -y cairo-devel && \
yum install -y libssh2-devel && \
yum install -y libxml2-devel && \
yum install -y R
```

The libraries above are often needed to install various analytics and machine learning packages in R. Depending on which packages you want to install during the build, you may need to add a few more libraries to the list above. Once the installation finishes, exit out of root. Test the installation by typing R in the command line to start a new R session.

INSTALLING R STUDIO LOCALLY

To install RStudio Server and/or RStudio, find the version of RStudio you want or get the latest *.rpm* file. Then, run the following commands:

```
wget https://download2.rstudio.org/server/centos6/x86_64/rstudio-server-
rhel-1.2.1335-x86_64.rpm \
&& yum install -y --nogpgcheck rstudio-server-rhel-1.2.1335-x86_64.rpm \
&& rm -f rstudio-server-rhel-1.2.1335-x86_64.rpm \
&& wget https://download1.rstudio.org/desktop/centos7/x86_64/rstudio-
1.2.1335-x86_64.rpm \
&& yum install -y --nogpgcheck rstudio-1.2.1335-x86_64.rpm \
&& rm -f rstudio-1.2.1335-x86_64.rpm
```

You should be able to go to *http://localhost:8787* and sign into RStudio server with your system username and password. If you run into Qt display issues with RStudio, you may need to run *yum install libxkbcommon-x11*. For more information see *https://github.com/rstudio/rstudio/issues/4610*.

INSTALLING R KERNEL FOR JUPYTER LOCALLY

In the previous sections, you installed R and RStudio. If you also have Python, pip, and Jupyter installed locally, you can test the R kernel installation locally. To make R available in Jupyter notebook, install the IRkernel from *https://irkernel.github.io/installation/*. Although you can use devtools to install this kernel, you will need to make sure that you have the right version of CURL installed, in order to successfully install the right version of devtools. In the Appendix, we show how to update CURL, which will allow you to get the correct version of devtools.

In this paper, we do not use devtools to install the R kernel and instead use an R session to install the kernel and other R packages. Since R packages can take a while to install, we provide a minimal list of packages to install. The content of the minimal script called *add_r_kernel_mini.sh* is shown below:

```
#!/bin/Rscript
install.packages(c('rlang','caret','tidyverse', 'dplyr',
'shiny','IRkernel'), repos='http://cran.us.r-project.org')
IRkernel::installspec(user = FALSE)
```

To check if the R kernel installed correctly, launch Jupyter and load the kernel (you may have to use *jupyter-notebook --allow-root*). In the next section, we use this script in the Dockerfile to install the R kernel in our Docker image.

CREATING THE FINAL DOCKER FILE

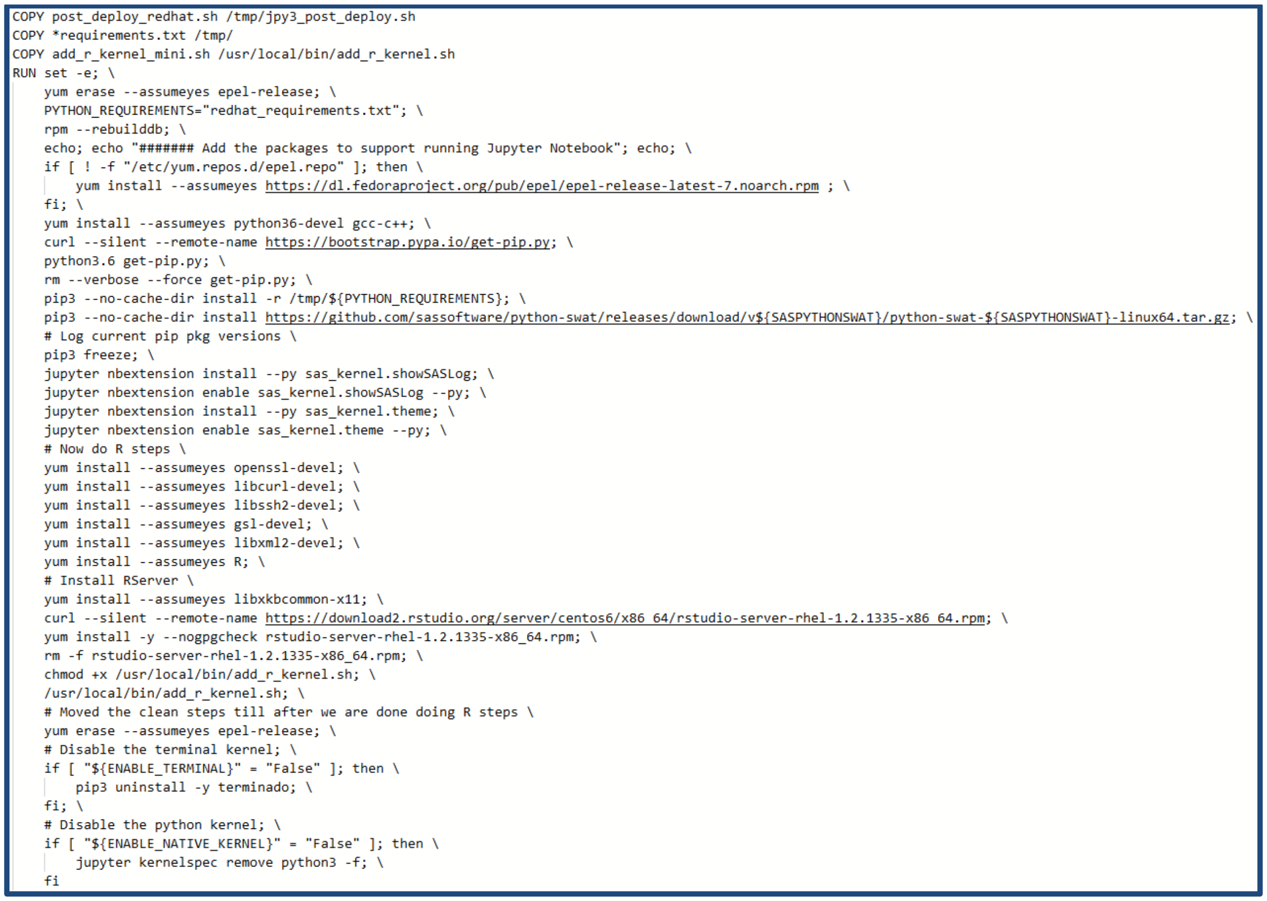
In the previous section, we showed the necessary steps to add R to the SAS Viya Docker image. In this section, we put all the pieces together and modify the default Dockerfile in the ide-jupyter-python3 addon to install R, RStudio server and the R kernel in the Docker image.

Installing R in the Docker image

The installation consists of adding new commands to the Dockerfile as well as enabling the use of bash scripts. Note the use of CURL instead of wget, since wget is not present by default in the Docker image. Follow these steps to install R in your Docker image:

- Inside the addons folder ide-jupyter-python3, create the file *add_r_kernel_mini.sh* with the contents exactly as shown in the previous section.
- Then add the line *COPY add_r_kernel_mini.sh /usr/local/bin/add_r_kernel.sh* right before the line *RUN set -e,*
- Use yum with the flag *--assumeyes* and CURL with the flag *--silent*
- Run *chmod + x* on the bash script and execute the script to install R packages and the R kernel
- Place the line *yum erase --assumeyes epel-release; * after the last R installation step

The final Docker file is shown in Figure 9.



```
COPY post_deploy_redhat.sh /tmp/jpy3_post_deploy.sh
COPY *requirements.txt /tmp/
COPY add_r_kernel_mini.sh /usr/local/bin/add_r_kernel.sh
RUN set -e; \
    yum erase --assumeyes epel-release; \
    PYTHON_REQUIREMENTS="redhat_requirements.txt"; \
    rpm --rebuilddb; \
    echo; echo "####### Add the packages to support running Jupyter Notebook"; echo; \
    if [ ! -f "/etc/yum.repos.d/epel.repo" ]; then \
        yum install --assumeyes https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm ; \
    fi; \
    yum install --assumeyes python36-devel gcc-c++; \
    curl --silent --remote-name https://bootstrap.pypa.io/get-pip.py; \
    python3.6 get-pip.py; \
    rm --verbose --force get-pip.py; \
    pip3 --no-cache-dir install -r /tmp/${PYTHON_REQUIREMENTS}; \
    pip3 --no-cache-dir install https://github.com/sassoftware/python-swat/releases/download/v${SASPYTHONSWAT}/python-swat-${SASPYTHONSWAT}-linux64.tar.gz; \
    # Log current pip pkg versions \
    pip3 freeze; \
    jupyter nbextension install --py sas_kernel.showSASLog; \
    jupyter nbextension enable sas_kernel.showSASLog --py; \
    jupyter nbextension install --py sas_kernel.theme; \
    jupyter nbextension enable sas_kernel.theme --py; \
    # Now do R steps \
    yum install --assumeyes openssl-devel; \
    yum install --assumeyes libcurl-devel; \
    yum install --assumeyes libssh2-devel; \
    yum install --assumeyes gsl-devel; \
    yum install --assumeyes libxml2-devel; \
    yum install --assumeyes R; \
    # Install RServer \
    yum install --assumeyes libxkbcommon-x11; \
    curl --silent --remote-name https://download2.rstudio.org/server/centos6/x86_64/rstudio-server-rhel-1.2.1335-x86_64.rpm; \
    yum install -y --nogpgcheck rstudio-server-rhel-1.2.1335-x86_64.rpm; \
    rm -f rstudio-server-rhel-1.2.1335-x86_64.rpm; \
    chmod +x /usr/local/bin/add_r_kernel.sh; \
    /usr/local/bin/add_r_kernel.sh; \
    # Moved the clean steps till after we are done doing R steps \
    yum erase --assumeyes epel-release; \
    # Disable the terminal kernel; \
    if [ "${ENABLE_TERMINAL}" = "False" ]; then \
        pip3 uninstall -y terminado; \
    fi; \
    # Disable the python kernel; \
    if [ "${ENABLE_NATIVE_KERNEL}" = "False" ]; then \
        jupyter kernelspec remove python3 -f; \
    fi
```

Figure 9: Dockerfile to add Python and R support to the SAS Viya container.

When creating, editing, and saving this Dockerfile in Windows, make sure that you save it with linux/unix line endings, otherwise your build will fail. You can either clean the file with:

```
sed -i -e 's/\r$//' your_Docker_file_plus_R.sh
```

or you can open it in a text editor like gedit and save it with Linux/Unix line endings. The Dockerfile shown in Figure 9 differs significantly from the default one as it does not check for a SUSE vs RedHat platform. We removed this logic from our Dockerfile since we kept encountering build issues with an “unknown platform” error. To make the same changes:

- Inside the addons folder *ide-jupyter-python3*, copy the contents of *post_deploy.sh* to a new file called *post_deploy_redhat.sh* (if your system is RedHat or CentOS)
- In *post_deplot_redhat.sh* change the line *[[-z ${PLATFORM+x}]] && PLATFORM=@PLATFORM@* to *[[-z ${PLATFORM+x}]] && PLATFORM=redhat*
- Remove the following line from the Dockerfile inside the *ide-jupyter-python3* folder *sed -i "s/@PLATFORM@/$PLATFORM/" /tmp/jpy3_post_deploy.sh; *
- Change the line *COPY post_deploy.sh /tmp/jpy3_post_deploy.sh* to *COPY post_deploy_redhat.sh /tmp/jpy3_post_deploy.sh*

BUILDING THE FINAL IMAGE WITH JUPYTER, PYTHON AND R SUPPORT

With the final changes made to the Dockerfile, you can now build an image with added R support. Simply repeat the command you used when adding Jupyter and Python:

```
./build.sh --type single --zip /home/admin/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --addons "auth-demo ide-jupyter-
python3"
```

If you are using your own mirror, add the *--mirror-url* flag replacing <*ip-address*> with your own IP address:

```
./build.sh --type single --zip /home/admin/sas-container-recipes-
master/SAS_Viya_deployment_data.zip --mirror-url "http://ip-
address/sas_repos/" --addons "auth-demo ide-jupyter-python3"
```

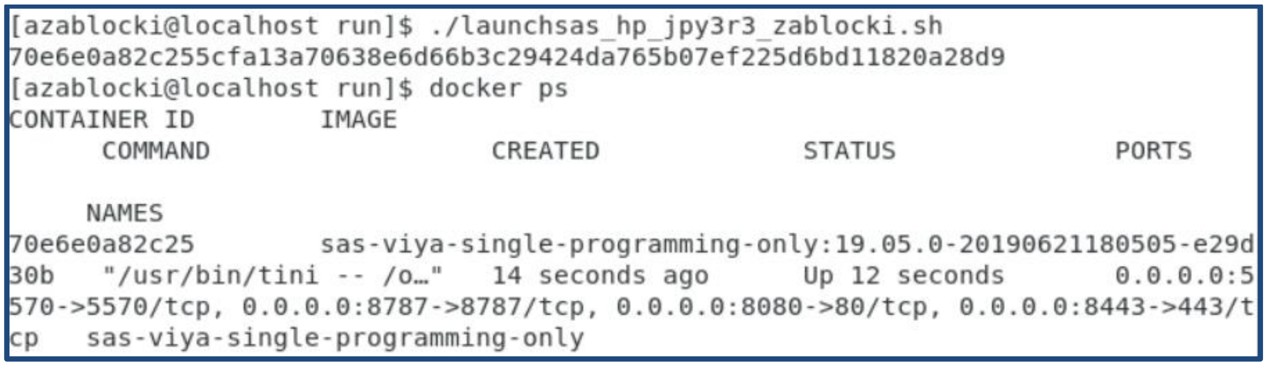
In contrast to the first run when you added Python and Jupyter, this R installation will take much longer (it could take up to an hour depending on your system and internet connection). When the build finishes, make note of the new tag for this build. Create a clone copy of the *launchsas_authdemo.sh* script you used previously and save it as *launchsas_jpy3r3.sh*. Replace the tag in this file with the one corresponding to the newest tag. You will use this script in the next section to test your Python and R Jupyter kernels, and new package installation.

TESTING KERNELS AND NEW PACKAGE INSTALLATION

Now that you have built a new SAS Viya Docker image with support for Python, Jupyter Notebook, and R, you can test its functionality. In this section, we show how to test the persistent storage setup and functionality such as loading libraries, and installing new packages for both Python and R.

STARTING THE CONTAINER

To start the container, run the launch script as shown in Figure 10. Use *docker ps* to confirm that the container is running.



```
[azablocki@localhost run]$ ./launchsas_hp_jpy3r3_zablocki.sh
70e6e0a82c255cfa13a70638e6d66b3c29424da765b07ef225d6bd11820a28d9
[azablocki@localhost run]$ docker ps
CONTAINER ID        IMAGE
      COMMAND                  CREATED             STATUS              PORTS

     NAMES
70e6e0a82c25        sas-viya-single-programming-only:19.05.0-20190621180505-e29d
30b    "/usr/bin/tini -- /o…"   14 seconds ago      Up 12 seconds       0.0.0.0:5
570->5570/tcp, 0.0.0.0:8787->8787/tcp, 0.0.0.0:8080->80/tcp, 0.0.0.0:8443->443/t
cp   sas-viya-single-programming-only
```

Figure 10: Launching the right Docker container, with its own launch script.

The contents of the launch script are the same as the script in Figure 4 with one small difference: the TAG corresponds to the tag for the latest Docker build. It's good practice that for every build you complete, you create a new launchsas.sh script and update it with the tag found in the build output.

TESTING THE PYTHON KERNEL

Navigate to 0.0.0.0:8080/Jupyter. Although port 8888 is the usual Jupyter port, the new version of SAS container recipes now uses port 8080. If you can launch Jupyter, but the kernels are consistently failing to load or they stop, you may be using the wrong port. We found this to be the case when we accessed Jupyter without explicitly stating the port, e.g., using 0.0.0.0/Jupyter instead of 0.0.0.0:8888/Jupyter. If you can launch Jupyter, you should see the file tree along with the option to start a new notebook with three kernel choices; choose Python 3 as shown in Figure 11.

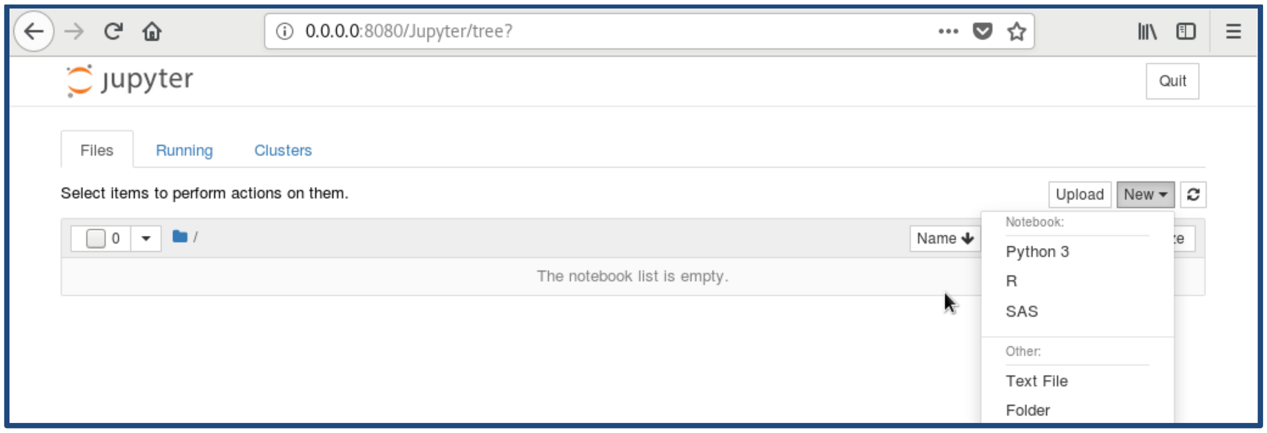
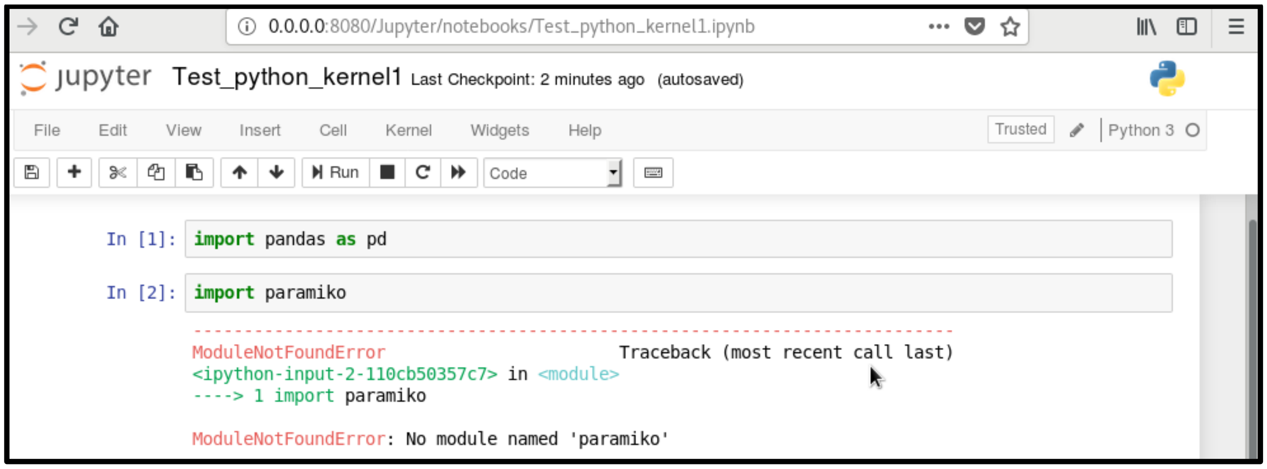

0.0.0.0:8080/Jupyter/tree?
jupyter
Quit
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Notebook:
Python 3
R
SAS
Other:
Text File
Folder
Name
The notebook list is empty.

Figure 11: Running Jupyter on port 8080, with 3 kernels available.

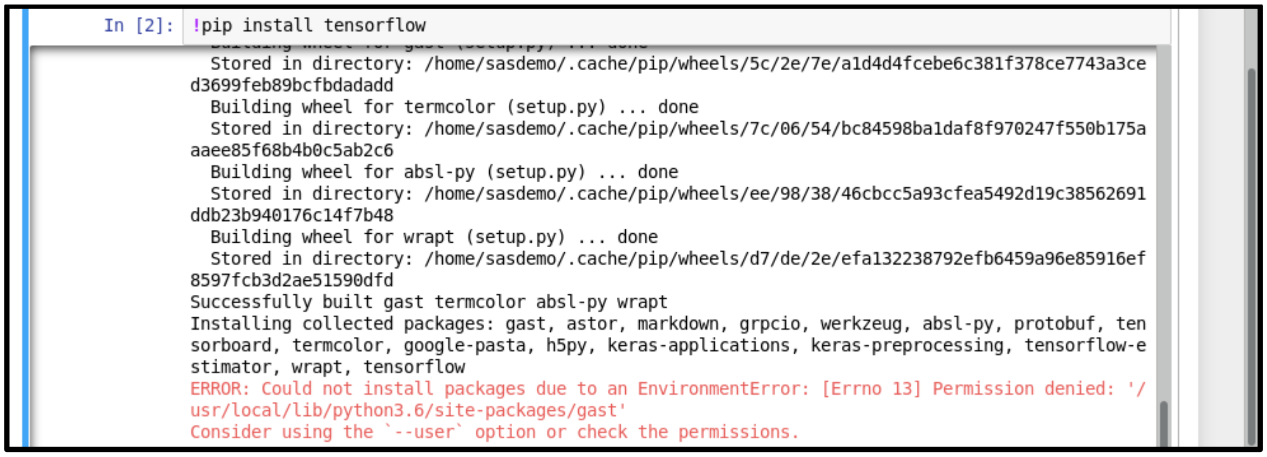
Open a new notebook and import packages that were previously installed at the time of the Docker build. In Figure 12, we load the package pandas and attempt to import the package paramiko.


0.0.0.0:8080/Jupyter/notebooks/Test_python_kernel1.ipynb
jupyter Test_python_kernel1 Last Checkpoint: 2 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [1]: import pandas as pd

In [2]: import paramiko

---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-2-110cb50357c7> in <module>
----> 1 import paramiko

ModuleNotFoundError: No module named 'paramiko'
```

Figure 12: Testing the Jupyter Python kernel and importing packages.

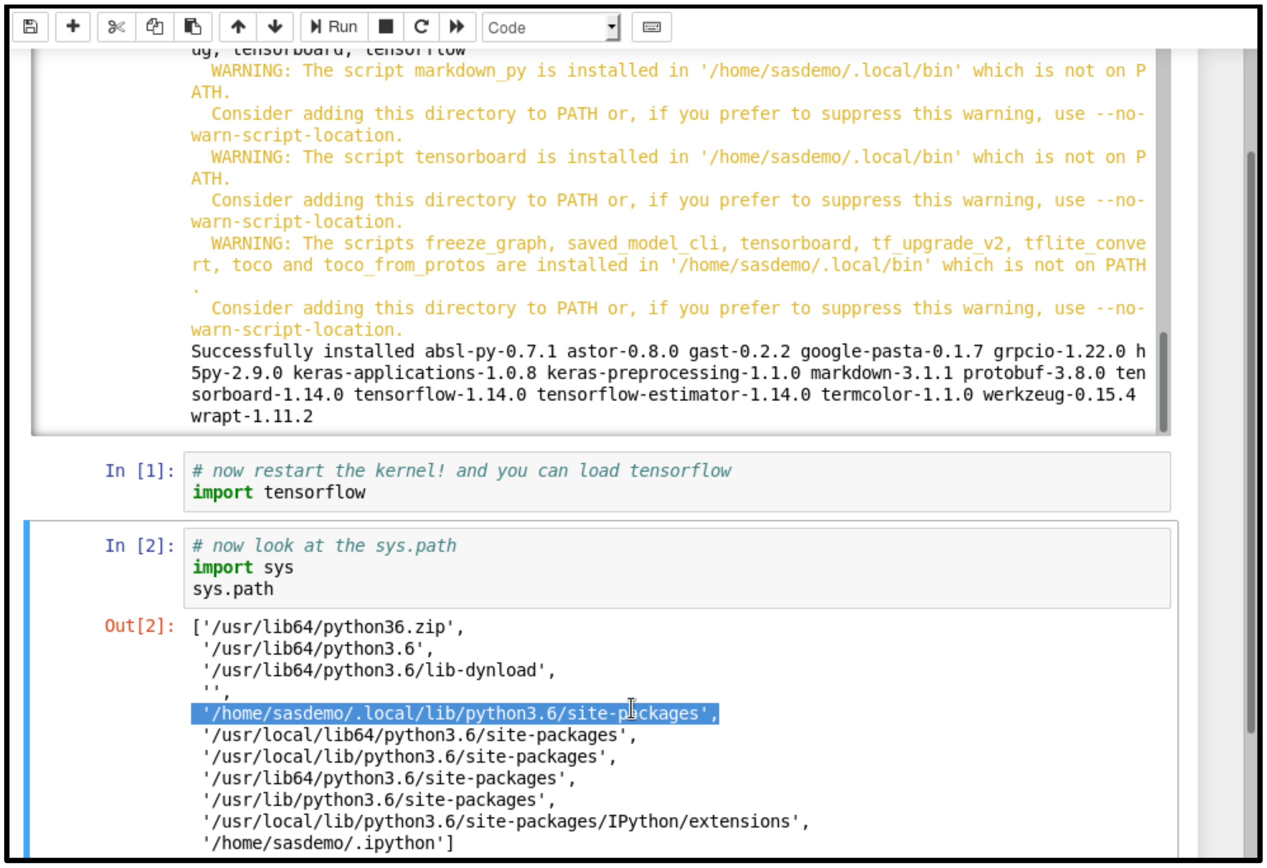
Since paramiko was not installed initially you will have to install it. Install missing packages with the !pip command. If you try to install without the *--user* flag, the install will fail as shown in Figure 13.



```
In [2]: !pip install tensorflow
  Stored in directory: /home/sasdemo/.cache/pip/wheels/5c/2e/7e/a1d4d4fcebe6c381f378ce7743a3ce
d3699feb89bcfbdadadd
  Building wheel for termcolor (setup.py) ... done
  Stored in directory: /home/sasdemo/.cache/pip/wheels/7c/06/54/bc84598ba1daf8f970247f550b175a
aaee85f68b4b0c5ab2c6
  Building wheel for absl-py (setup.py) ... done
  Stored in directory: /home/sasdemo/.cache/pip/wheels/ee/98/38/46cbcc5a93cfea5492d19c38562691
ddb23b940176c14f7b48
  Building wheel for wrapt (setup.py) ... done
  Stored in directory: /home/sasdemo/.cache/pip/wheels/d7/de/2e/efa132238792efb6459a96e85916ef
8597fcb3d2ae51590dfd
Successfully built gast termcolor absl-py wrapt
Installing collected packages: gast, astor, markdown, grpcio, werkzeug, absl-py, protobuf, ten
sorboard, termcolor, google-pasta, h5py, keras-applications, keras-preprocessing, tensorflow-e
stimator, wrapt, tensorflow
ERROR: Could not install packages due to an EnvironmentError: [Errno 13] Permission denied: '/
usr/local/lib/python3.6/site-packages/gast'
Consider using the `--user` option or check the permissions.
```

Figure 13: Installing new Python packages will not work without the --user option.

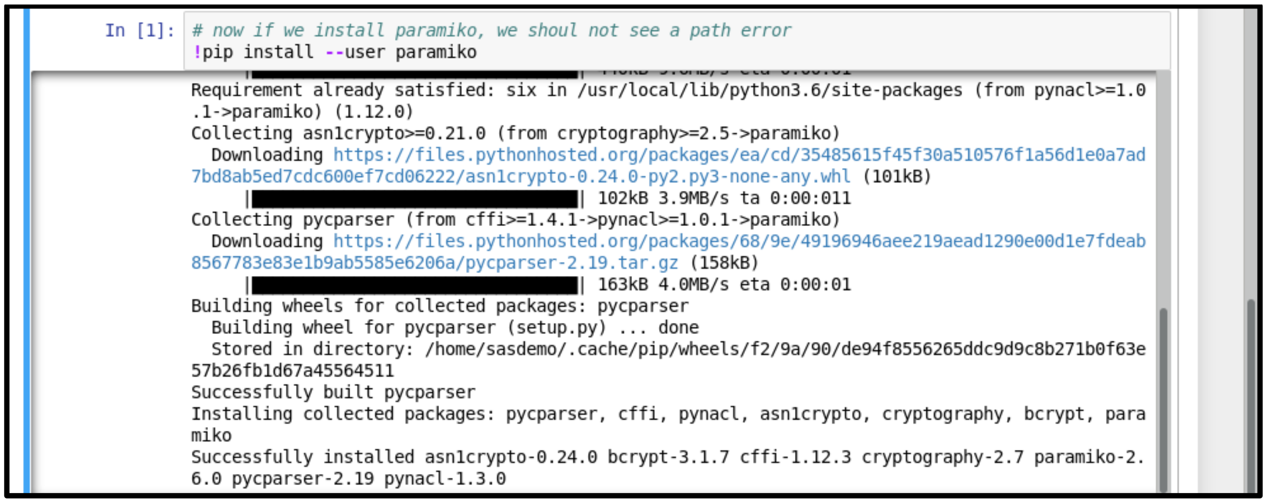
Installing a different package such as TensorFlow with the *--user* flag will now work. To make the package available to the kernel, restart the kernel (see Figure 14).



```
  WARNING: The script markdown_py is installed in '/home/sasdemo/.local/bin' which is not on P
ATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-
warn-script-location.
  WARNING: The script tensorboard is installed in '/home/sasdemo/.local/bin' which is not on P
ATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-
warn-script-location.
  WARNING: The scripts freeze_graph, saved_model_cli, tensorboard, tf_upgrade_v2, tflite_conve
rt, toco and toco_from_protos are installed in '/home/sasdemo/.local/bin' which is not on PATH
.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-
warn-script-location.
Successfully installed absl-py-0.7.1 astor-0.8.0 gast-0.2.2 google-pasta-0.1.7 grpcio-1.22.0 h
5py-2.9.0 keras-applications-1.0.8 keras-preprocessing-1.1.0 markdown-3.1.1 protobuf-3.8.0 ten
sorboard-1.14.0 tensorflow-1.14.0 tensorflow-estimator-1.14.0 termcolor-1.1.0 werkzeug-0.15.4
wrapt-1.11.2

In [1]: # now restart the kernel! and you can load tensorflow
        import tensorflow

In [2]: # now look at the sys.path
        import sys
        sys.path

Out[2]: ['/usr/lib64/python36.zip',
         '/usr/lib64/python3.6',
         '/usr/lib64/python3.6/lib-dynload',
         '',
         '/home/sasdemo/.local/lib/python3.6/site-packages',
         '/usr/local/lib64/python3.6/site-packages',
         '/usr/local/lib/python3.6/site-packages',
         '/usr/lib64/python3.6/site-packages',
         '/usr/lib/python3.6/site-packages',
         '/usr/local/lib/python3.6/site-packages/IPython/extensions',
         '/home/sasdemo/.ipython']
```

Figure 14: Installing a new package and loading a new package after kernel restart. Note that sys.path shows the location where locallly installed packages are stored (when using *pip install --user package-name*).

In Figure 14, you can see that all new packages are saved in the */home/sasdemo/.local* folder. Since you mapped the */home* folder in Docker to a folder in your local VM, any data and newly installed packages will be stored there and will persist. This will also be the case when using this container in the cloud, which we cover in our companion paper, "Deploying SAS® Viya® Docker images to the cloud - a step by step guide". If you try to install another package (for example, paramiko) with the *--user* flag, the package install will now not yield any path warnings (see Figure 15).



```
In [1]: # now if we install paramiko, we shoul not see a path error
        !pip install --user paramiko

Requirement already satisfied: six in /usr/local/lib/python3.6/site-packages (from pynacl>=1.0
.1->paramiko) (1.12.0)
Collecting asn1crypto>=0.21.0 (from cryptography>=2.5->paramiko)
  Downloading https://files.pythonhosted.org/packages/ea/cd/35485615f45f30a510576f1a56d1e0a7ad
7bd8ab5ed7cdc600ef7cd06222/asn1crypto-0.24.0-py2.py3-none-any.whl (101kB)
    |████████████████████████████████| 102kB 3.9MB/s ta 0:00:011
Collecting pycparser (from cffi>=1.4.1->pynacl>=1.0.1->paramiko)
  Downloading https://files.pythonhosted.org/packages/68/9e/49196946aee219aead1290e00d1e7fdeab
8567783e83e1b9ab5585e6206a/pycparser-2.19.tar.gz (158kB)
    |████████████████████████████████| 163kB 4.0MB/s eta 0:00:01
Building wheels for collected packages: pycparser
  Building wheel for pycparser (setup.py) ... done
  Stored in directory: /home/sasdemo/.cache/pip/wheels/f2/9a/90/de94f8556265ddc9d9c8b271b0f63e
57b26fb1d67a45564511
Successfully built pycparser
Installing collected packages: pycparser, cffi, pynacl, asn1crypto, cryptography, bcrypt, para
miko
Successfully installed asn1crypto-0.24.0 bcrypt-3.1.7 cffi-1.12.3 cryptography-2.7 paramiko-2.
6.0 pycparser-2.19 pynacl-1.3.0
```

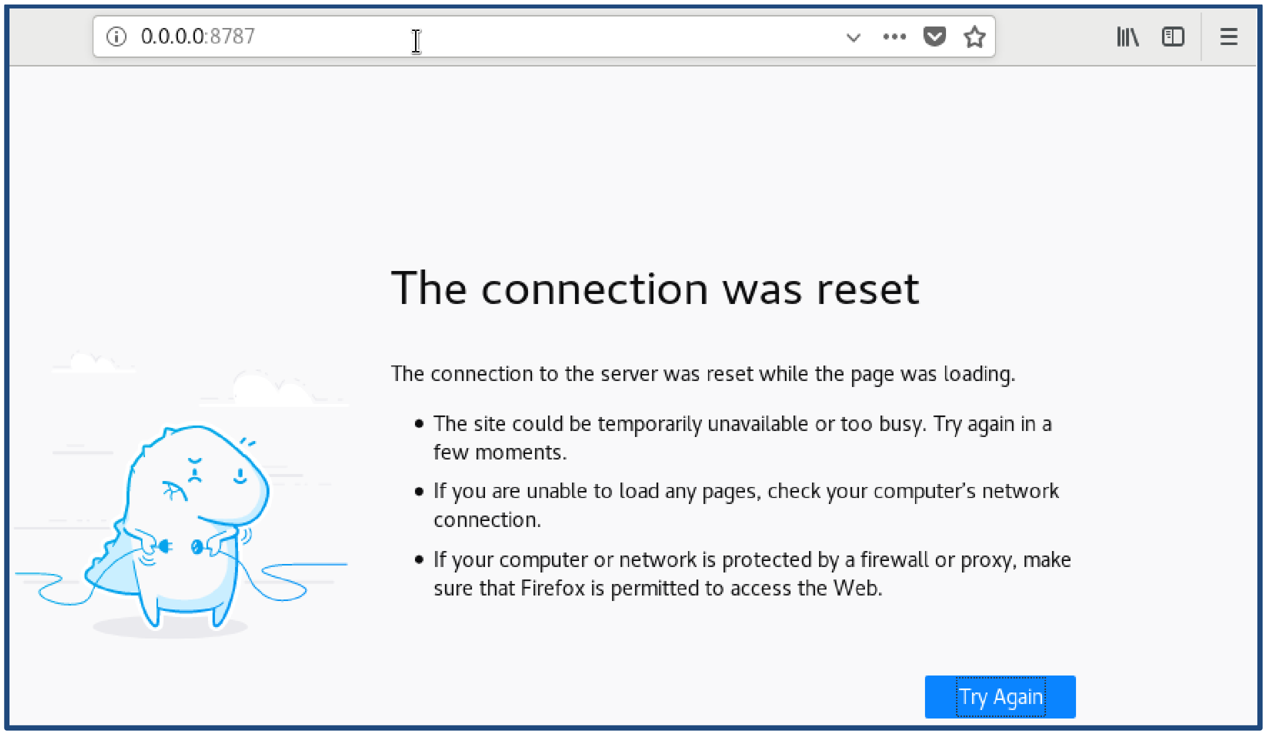
Figure 15: Once you install your first local package (TensorFlow) and restart the kernel, subsequent installs do not return path errors.

TESTING THE R INSTALLATION

There are three different ways to start an R session in your Docker image: using the IRkernel in Jupyter, in a RStudio Server session and in the command line session, when you “exec” into a running Docker container. For more information on R sessions, see *https://stat.ethz.ch/R-manual/R-devel/library/base/html/Startup.html* and *https://support.rstudio.com/hc/en-us/articles/115014830827-Why-is-libPaths-different-in-RStudio-vs-R-*. To test the R installation, you will first test the RStudio server setup. This will show you where RStudio saves locally installed libraries. Then you can test the IRkernel in Jupyter and learn about local library paths in R, and how to configure them to work with your Docker image.

Testing R Studio server

RStudio server runs on port 8787 by default. However, if you navigate to 0.0.0.0:8787, you will not see RStudio, because the server is not running (see Figure 16).



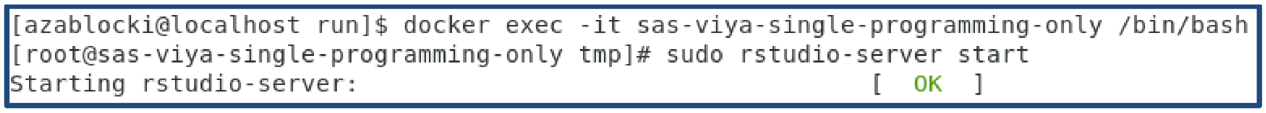
The connection was reset

The connection to the server was reset while the page was loading.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer’s network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Figure 16: To access RStudio server, the server process must be started from inside the container.

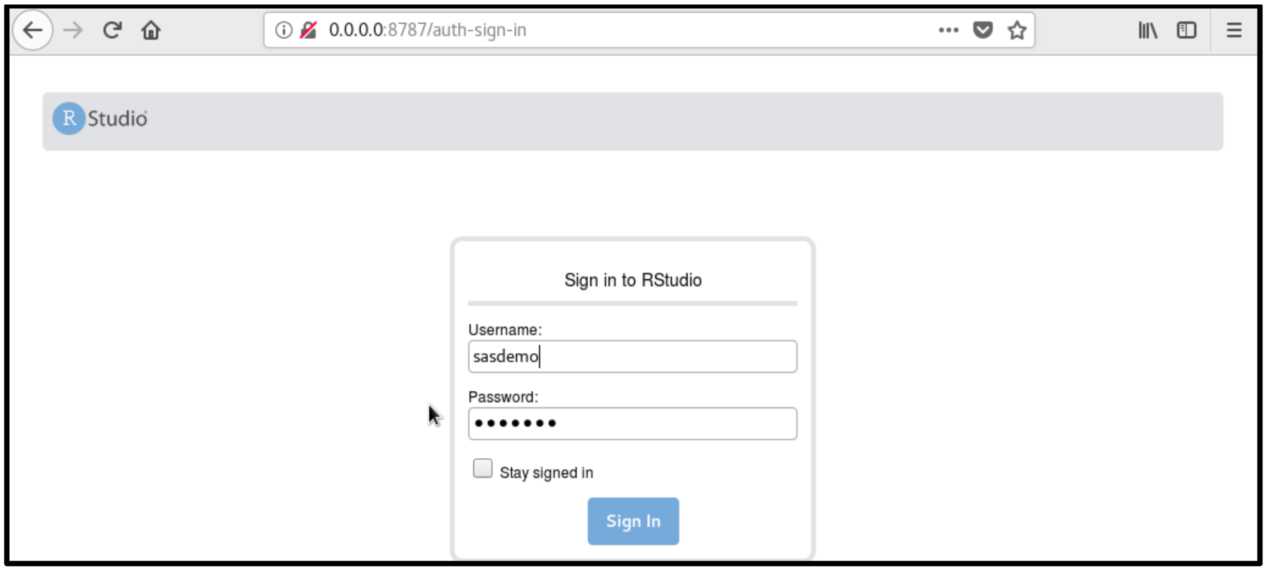
To initiate the server, make sure that Docker is currently running using *sudo systemctl start docker*. Then log into or “exec” into the Docker image and execute the start command as shown in Figure 17.



```
[azablocki@localhost run]$ docker exec -it sas-viya-single-programming-only /bin/bash
[root@sas-viya-single-programming-only tmp]# sudo rstudio-server start
Starting rstudio-server:                                   [  OK  ]
```

Figure 17: Execing into the Docker image and starting the server.

Navigate to 0.0.0.0:8787 (or refresh the page) and a log in screen will appear, where you use the same default username and password as you used to log into a SAS Studio session (see Figure 18).



Sign in to RStudio

Username: sasdemo

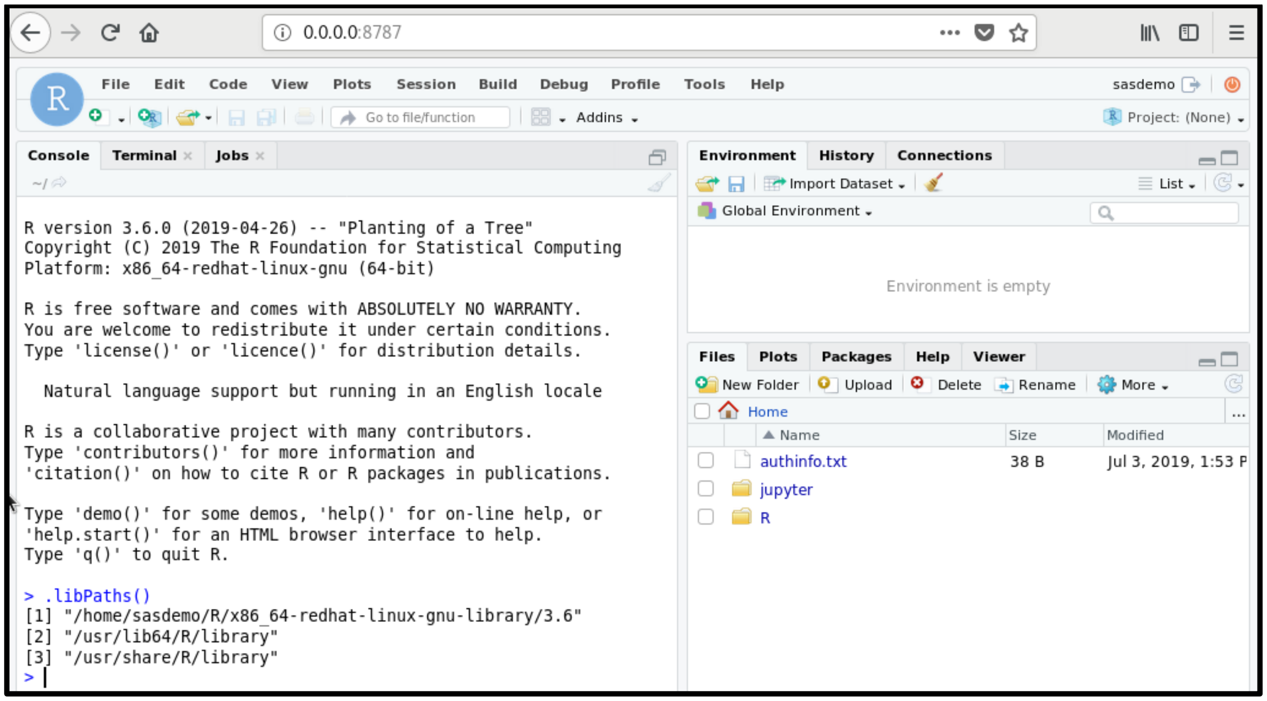
Password: •••••••

Stay signed in

Sign In

Figure 18: With the RStudio server running, you can resolve the sign-in page.

Once you log into RStudio, use the *.libPaths()* command in to show the library paths that are available. Listed first, will be a local path where package installations performed in RStudio will be saved. This is the path that you must add to Jupyter so that you can install new R packages inside a Jupyter Notebook session with an R kernel (see Figure 19).



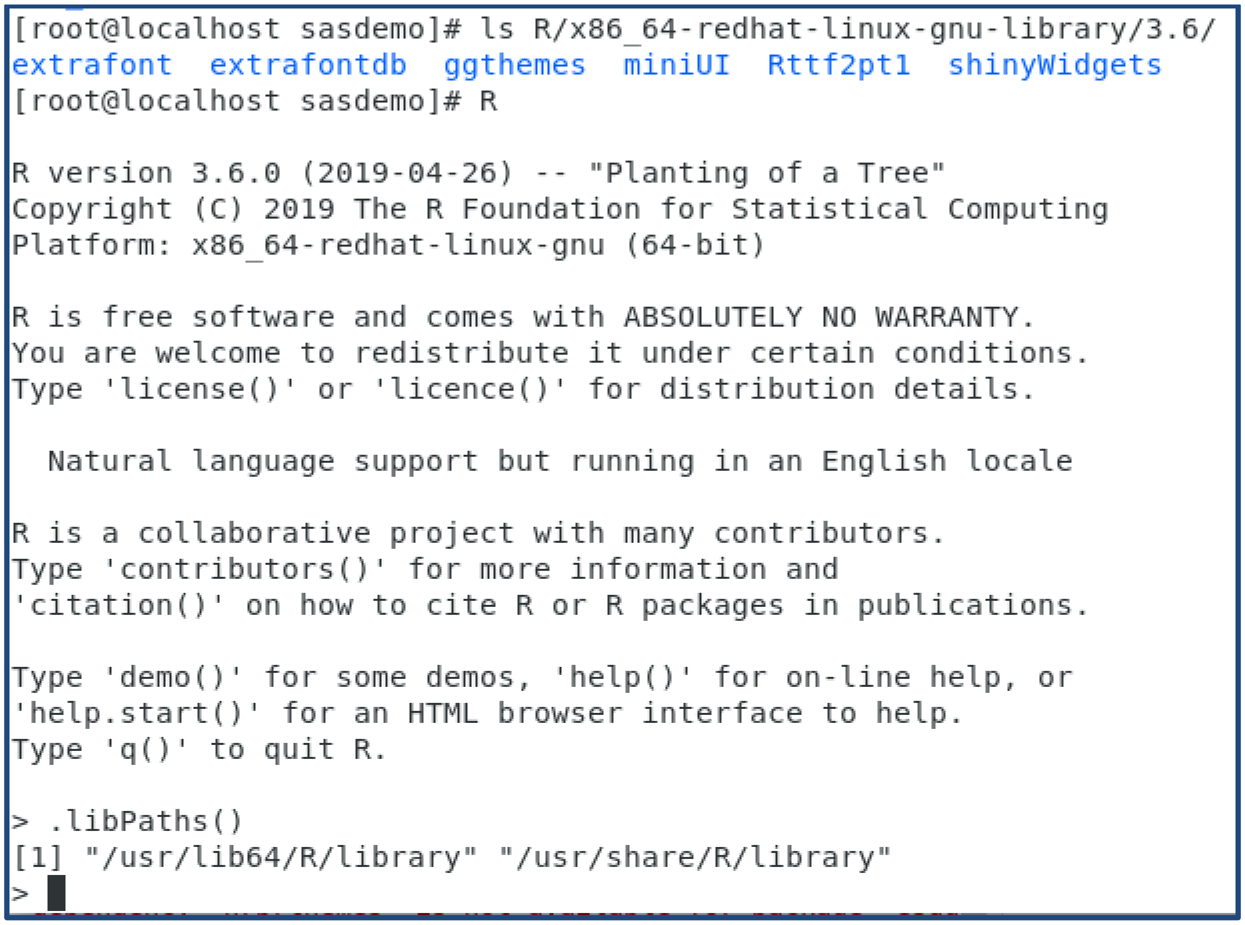
```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> .libPaths()
[1] "/home/sasdemo/R/x86_64-redhat-linux-gnu-library/3.6"
[2] "/usr/lib64/R/library"
[3] "/usr/share/R/library"
>
```

Name	Size	Modified
authinfo.txt	38 B	Jul 3, 2019, 1:53 P
jupyter		
R		

Figure 19: Use *.libPaths()* to show available library paths.

Configuring command line R session library path

If you exec into the Docker container and launch an R session, you will see that the path *"/home/sasdemo/R/x86_64-redhat-linux-gnu-library/3.6"* is missing from the output of the *.libPaths()* command (see Figure 20).



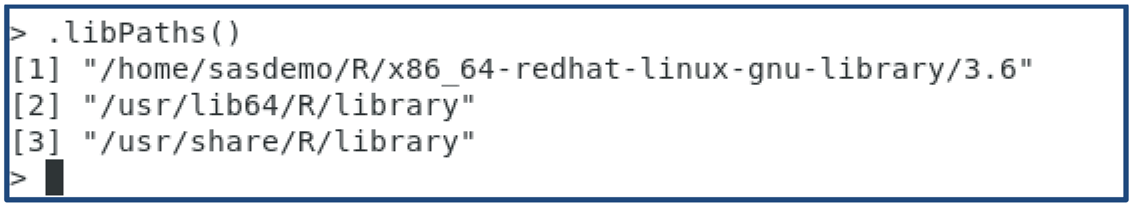
```
[root@localhost sasdemo]# ls R/x86_64-redhat-linux-gnu-library/3.6/
extrafont  extrafontdb  ggthemes  miniUI  Rttf2pt1  shinyWidgets
[root@localhost sasdemo]# R

R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> .libPaths()
[1] "/usr/lib64/R/library" "/usr/share/R/library"
> 
```

Figure 20: Output of *.libPaths()* in an R session launched from the command line.

To make the library path available to Jupyter, you will need to place a *.Rprofile* file in the */home/sasdemo* directory. You can do this from inside a running Docker container, or as a root user in your VM (recall that the contents of */home/sasdemo* are only visible to the root user). In your local VM, navigate to the */home/sasdemo* directory. As a root user, open a file called *.Rprofile*. Enter the following (or your R version equivalent):

```
   .libPaths("/home/sasdemo/R/x86_64-redhat-linux-gnu-library/3.6")
```

Restart the container, exec into the image, and open a new R session. If you use *.libPaths()* to list the available paths, you will now see the newly added path to *R/x86_64-redhat-linux-gnu-library/3.6* (see Figure 21). The one caveat here is that you must start the session in */home/sasdemo*, otherwise the *.Rprofile* file is not used. There are other ways to achieve this, but we do not cover these in this paper.

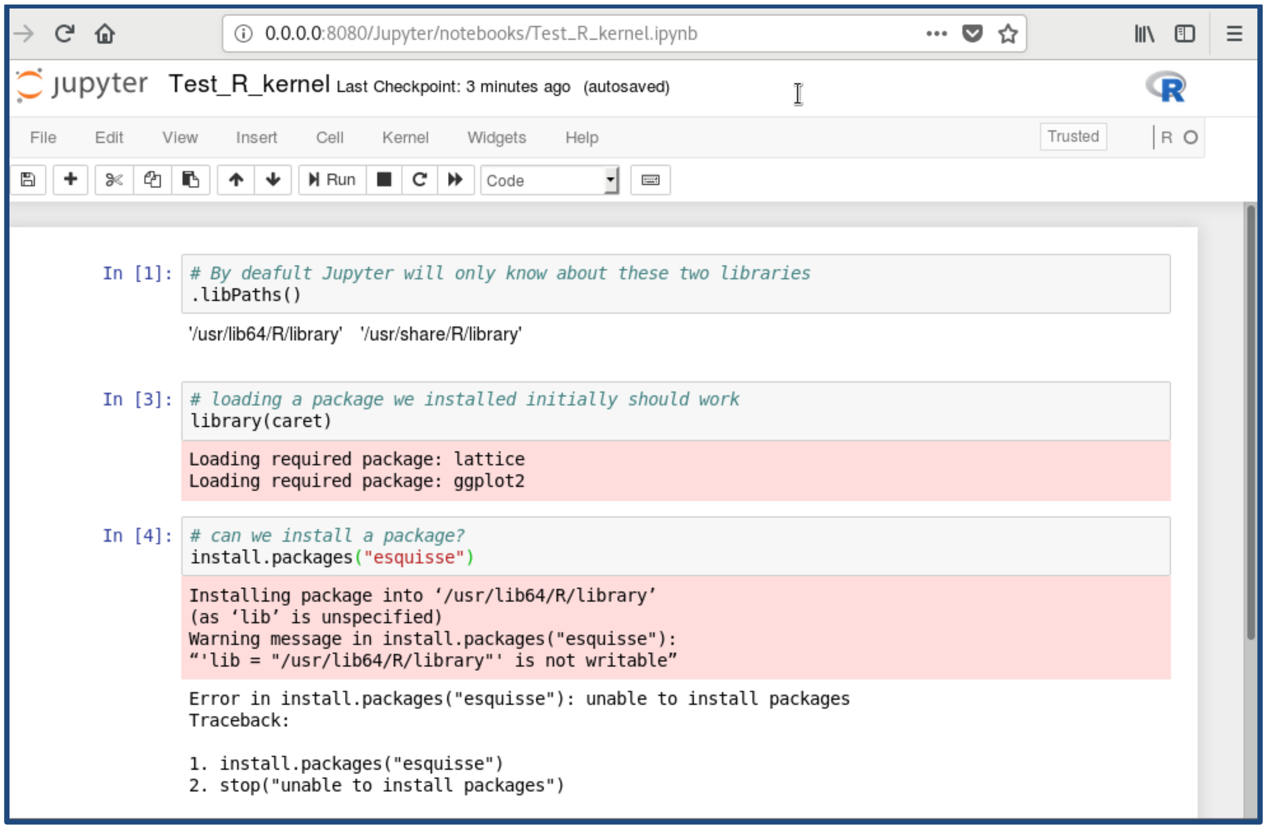


```
> .libPaths()
[1] "/home/sasdemo/R/x86_64-redhat-linux-gnu-library/3.6"
[2] "/usr/lib64/R/library"
[3] "/usr/share/R/library"
> 
```

Figure 21: Once you add *.Rprofile* to your *home/sasdemo* directory, the local R path is visible.

Testing R kernel in Jupyter

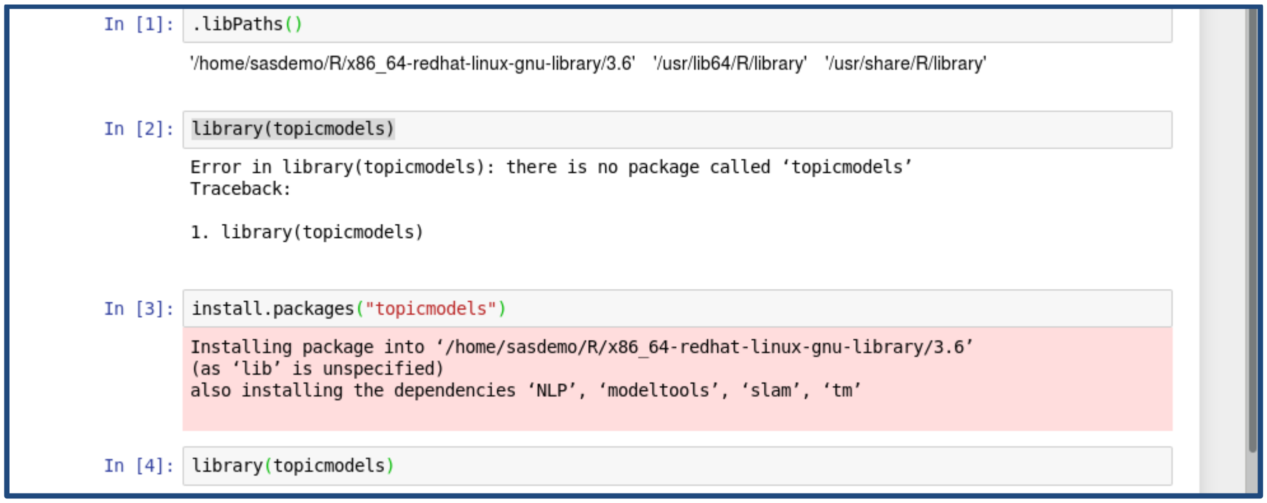
In the previous section, we showed how to set up a local R library path with the *.Rprofile* file. If this file is removed, then you will not be able to install new libraries in a Jupyter session as shown in Figure 22.


0.0.0.0:8080/Jupyter/notebooks/Test_R_kernel.ipynb
jupyter Test_R_kernel Last Checkpoint: 3 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
R
Run
Code

```
In [1]: # By deafult Jupyter will only know about these two libraries
        .libPaths()

        '/usr/lib64/R/library'  '/usr/share/R/library'

In [3]: # loading a package we installed initially should work
        library(caret)

        Loading required package: lattice
        Loading required package: ggplot2

In [4]: # can we install a package?
        install.packages("esquisse")

        Installing package into '/usr/lib64/R/library'
        (as 'lib' is unspecified)
        Warning message in install.packages("esquisse"):
        "'lib = "/usr/lib64/R/library"' is not writable"

        Error in install.packages("esquisse"): unable to install packages
        Traceback:

        1. install.packages("esquisse")
        2. stop("unable to install packages")
```

Figure 22: With the *.Rprofile* file removed, you cannot install new libraries.

When the *.Rprofile* file is removed, you only see */usr/lib64/R* and */usr/share/R*, which are the defaults when you start an R session from the command line. You can import the caret package since it was installed at the time of the build. Installing a new package fails since */usr/lib64/R* is not writable. If you replace the *.Rprofile* file, restart the container, and launch Jupyter Notebook with a R kernel, the command *.libPaths()* shows the full path to *R/x86_64-redhat-linux-gnu-library/3.6* (see Figure 23).



```
In [1]: .libPaths()

        '/home/sasdemo/R/x86_64-redhat-linux-gnu-library/3.6'  '/usr/lib64/R/library'  '/usr/share/R/library'

In [2]: library(topicmodels)

        Error in library(topicmodels): there is no package called 'topicmodels'
        Traceback:

        1. library(topicmodels)

In [3]: install.packages("topicmodels")

        Installing package into '/home/sasdemo/R/x86_64-redhat-linux-gnu-library/3.6'
        (as 'lib' is unspecified)
        also installing the dependencies 'NLP', 'modeltools', 'slam', 'tm'

In [4]: library(topicmodels)
```

Figure 23: With *.Rprofile* present, a Jupyter R kernel session can install new packages to the local path.

As shown in Figure 23, you can now install new R packages. They are saved to the local R library path that will persist, and therefore the packages will be available whenever you start up the container.

CONCLUSION

In this paper, we used the SAS container recipes, an open source GitHub project to build a single programming-only SAS Viya Docker image. We showed how to deploy this image locally with persistent storage, so that code and data can be shared between the Docker image and the local machine or local VM. We also showed how to add R support to the image, and we discussed common issues we experienced during the initial build and subsequent modifications. Finally, we tested Python and R kernels and new package installations. In our companion paper, "Deploying SAS® Viya® Docker images to the cloud - a step by step guide", we show how to deploy this Docker image to the cloud in order to provide a flexible and a fully scalable data science working environment with SAS Viya, Python and R.

APPENDIX

RE-POINTING /VAR/LIB/DOCKER TO A NEW VOLUME

Before configuring the new volume, make sure to stop docker and backup your previous Docker images:

```
rsync -aqxP /var/lib/docker/ /home/admin/backup_var_lib_docker/
```

While you could remove the images with the command *docker image prune -a*, you do not have to, for reasons we give below. Assuming your new volume is visible (when using a disk utility or *fdisk -l*; you can also look inside */etc/fstab*) and the volume is listed under */dev/sdb*, issue the following commands as root to mount the new volume as the /data partition:

```
fdisk /dev/sdb
mkfs.ext4 /dev/sdb1
mkdir /data
mount /dev/sdb1 /data
```

The first command will start the partition process; you will be asked to provide various options. We used *n* for new partition, followed by *p*, for primary partition, and we gave the partition the number *1*. You will then be prompted for values for the first and the last sector. Use the suggested defaults, e.g., 2048 and the upper value for the partition size. Next, edit the file */etc/docker/daemon.json* to point the *data-root* flag to your new directory:

```
{
"data-root" : "/data/docker"
}
```

Finally, change permissions (as root) with:

```
sudo chown root:root /data/docker/ && chmod 701 /data/docker/
```

DOCKER VERSION INFORMATION

The Docker version used in this paper:

```
[admin@RM-SAS-DOCKER-01 ~]$ docker --version
Docker version 18.09.6, build 481bc77156
[admin@RM-SAS-DOCKER-01 ~]$ docker version
Client:
 Version:           18.09.6
 API version:       1.39
 Go version:        go1.10.8
 Git commit:        481bc77156
 Built:             Sat May  4 02:34:58 2019
```

```
  OS/Arch:           linux/amd64
  Experimental:      false
 Server: Docker Engine - Community
  Engine:
   Version:          18.09.6
   API version:      1.39 (minimum version 1.12)
   Go version:       go1.10.8
   Git commit:       481bc77
   Built:            Sat May  4 02:02:43 2019
   OS/Arch:          linux/amd64
   Experimental:     false
```

CENTOS VM INFORMATION

The CentOS version used:

```
[admin@RM-SAS-DOCKER-01 ~]$ uname -a
Linux RM-SAS-DOCKER-01.SASDEV.DMZ 3.10.0-957.21.3.el7.x86_64 #1 SMP Tue Jun
18 16:35:19 UTC 2019 x86_64 GNU/Linux
[admin@RM-SAS-DOCKER-01 ~]$ cat /etc/centos-release
CentOS Linux release 7.6.1810 (Core)
[admin@RM-SAS-DOCKER-01 ~]$ rpm -q centos-release
centos-release-7-6.1810.2.el7.centos.x86_64
```

UPDATING CURL

If you choose to use devtools to install the R kernel, you may experience errors due to the wrong CURL version. To update CURL, find the version you need and run:

```
yum update
wget https://github.com/curl/curl/releases/download/curl-7_65_1/curl-
7.65.1.tar.gz
tar -xvzf curl-7.65.1.tar.gz
cd curl-7.65.1/
./configure
make
make install
curl -V
```

Restart the machine. You should see the newer version and be able to install the right version of devtools. To install the IR kernel, use:

```
devtools::install_github('IRkernel/IRkernel')
```

REFERENCES

SAS Container Recipes https://github.com/sassoftware/sas-container-recipes

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Alan Zablocki
RedMane Technology
alan_zablocki@redmane.com
www.alanzablocki.com