Big Data in Economics

Lecture 13: Google Compute Engine (Part I)

Grant R. McDermott

University of Oregon | EC 510

Contents

Today is the first of two lectures on Google Compute Engine. In these two lectures, I'm going to show you how to run R and RStudio Server on virtual machines (VMs) up on the cloud. This means that you'll be able to conduct your analysis in (almost) exactly the same user environment as you're used to, but now with the full power of cloud-based computation at your disposal. Trust me, it will be awesome.

Requirements

Create an account on Google Cloud Platform (free)

These next instructions are important, so please read carefully.

1. Sign up for a 12-month ($300 credit) free trial with Google Cloud Platform (GCP). This requires an existing Google/Gmail account.[1] During the course of sign-up, you'll be prompted to enter credit card details for billing purposes. Don't worry, you won't be charged unless/until you actively request continued access to GCP after your free trial ends. But a billable project ID is required before gaining access to the platform.
2. Download and follow the installation instructions for the Google Cloud SDK command line utility, `gcloud`. This is how we'll connect to GCP from our local computers via the shell.

Introduction

To the cloud!

Thus far in the course, we've spent quite a lot of time learning how to code efficiently. We've covered topics like functional programming, caching, parallel programming, and so on. All of these tools will help you make the most of the computational resources at your disposal. However, there's a limit to how far they can take you. At some point, datasets become too big, simulations become too complex, and regressions take too damn long to run to run on your laptop. The only solution beyond this point is ~~more power~~ MOAR POWA.

[1] If you have multiple Gmail accounts, please pick one and stick to it consistently whenever you are prompted to authenticate a new GCP service API.

The easiest and cheapest way to access more computational power these days is through the cloud.[2] While there are a number of excellent cloud service providers, I'm going to focus on **Google Cloud Platform** (**GCP**).[3] GCP offers a range of incredibly useful services — some of which we'll cover in later lectures — and the 12-month free trial makes an ideal entry point for learning about cloud computation.

The particular GCP product that we're going to use today is **Google Compute Engine** (**GCE**). GCE is a service that allows users to launch so-called *virtual machines* on demand in the cloud (i.e. on Google's data centers). There's a lot more that I can say — and will say later — about the benefits can bring to us. But right now, you may well be asking yourself: "What is a virtual machine and why do I need one anyway?"

So, let's take a step back and quickly clear up some terminology.

Virtual machines (VMs)

A virtual machine (VM) is just an emulation of a computer running inside another (bigger) computer. It can potentially perform all and more of the operations that your physical laptop or desktop does. It might even share many of the same properties, from operating system to internal architecture. The key advantage of a VM from our perspective is that very powerful machines can be "spun up" in the cloud almost effortlessly and then deployed to tackle jobs that are beyond the capabilities of your local computer. Got a big dataset that requires too much memory to analyse on your old laptop? Load it into a high-powered VM. Got some code that takes an age to run? Fire up a VM and let it chug away without consuming any local resources. Or, better yet, write code that runs in parallel and then spin up a VM with lots of cores to get the analysis done in a fraction of the time. All you need is a working internet connection and a web browser.

Now, with that background knowledge in mind, GCE delivers high-performance, rapidly scalable VMs. A new VM can be deployed or shut down within seconds, while existing VMs can easily be ramped up or down depending on a project's needs (cores added, RAM added, etc.) In my experience, most people would be hard-pressed to spent more than a couple of dollars a month using GCE once their free trial is over. This is especially true for researchers or data scientists who only need to fire up a VM, or VM cluster, occasionally for the most computationally-intensive part of a project, and then can easily switch it off when it is not being used.

[2] While the cloud is not the only game in town, it offers a variety benefits that, in my view, make it a no-brainer for most people: economies of scale make it much cheaper; maintenance and depreciation worries are taken care of; access does not hinge on institutional affiliation or faculty status; cloud providers offer a host of other useful services; etc.

[3] Alternatives to GCP include AWS and Digital Ocean. RStudio recently launched its own cloud service too: RStudio Cloud is more narrowly focused, but is great for teaching and is (currently) free to use. The good news is that these are all great options and the general principles of cloud computing carry over very easily. So use whatever you feel comfortable with.

Disclaimer: While I very much stand by the above paragraph, it is ultimately *your* responsibility to keep track of your billing and utilisation rates. Take a look at GCP's Pricing Calculator to see how much you can expect to be charged for a particular machine and level of usage. You can even set a budget and create usage alerts if you want to be extra cautious.

Roadmap

Our goal for the next two lectures is to set up a VM (or cluster of VMs) on GCE. What's more, we want to install R and RStudio (Server) on these VMs, so that we can interact with them in exactly the same way that we're used to on our own computers. I'm going to show you two approaches:

1. Manually configure GCE with RStudio Server (today)
2. Automate with **googleComputeEngineR** and friends (next lecture)

Both approaches have their merits, but I think it's important to start with the manual configuration so that you get a good understanding of what's happening underneath the hood. Let's get started.

Install R and RStudio Server on GCE

Note: It's possible to complete nearly all of the steps in this section via the GCE browser console. However, we'll stick to using the shell, because that will make it easier to document our steps.

Confirm that you have installed `gcloud` correctly

You'll need to choose an operating system (OS) for your VM, as well as its designated zone. Let's quickly look at the available options, since this will also be a good time to confirm that you correctly installed the `gcloud` command-line interface. Open up your shell and enter (without the $ command prompt):

```
$ gcloud compute images list
$ gcloud compute zones list
```

> **Tip:** If you get an error message with the above commands, try re-running them with `sudo` at the beginning. If this works for you, then you will need to append "sudo" to the other shell commands in this lecture.

You'll know that everything is working properly if these these commands return a large range of options. If you get an error, please try reinstalling `gcloud` again before continuing.

Create a VM

The key shell command for creating your VM is **`gcloud compute instances create`**. You can specify the type of machine that you want and a range of other options by using the appropriate flags. Let me first show you an example of the command and then walk through my (somewhat arbitrary) choices in more detail. Note that I am going to call my VM instance "my-vm", but you can call it whatever you want.

```
$ gcloud compute instances create my-vm --image-family ubuntu-2004-lts --image-project ubuntu-os-cloud  --machin
```

Here is a breakdown of the command and a quick explanation of my choices.

- `gcloud compute instances create my-vm`: Create a new VM called "my-vm". Yes, I am very creative.
- `--image-family ubuntu-2004-lts --image-project ubuntu-os-cloud`: Use Ubuntu 20.04 as the underlying operating system.
- `--machine-type n1-standard-8`: I've elected to go with the "N1 Standard 8" option, which means that I'm getting 8 CPUs and 30GB RAM. However, you can choose from a range of machine/memory/pricing options. (Assuming a monthly usage rate of 20 hours, this particular VM will only cost about $7.60 a month to maintain once our free trial ends.) You needn't worry too much about these initial specs now. New VMs are very easy to create and discard once you get the hang of it. It's also very simple to change the specs of an already-created VM. GCE will even suggest cheaper specifications if it thinks that you aren't using your resources efficiently down the line.

- `--zone us-west1-a`: My preferred zone. The zone choice shouldn't really matter, although you'll be prompted to choose one if you forget to include this flag. As a general rule, I advise picking whatever's closest to you.[4]

Assuming that you ran the above command (perhaps changing the zone to one nearest you), you should see something like the following:

```
Created [https://www.googleapis.com/compute/v1/projects/YOUR-PROJECT/zones/YOUR-ZONE/instances/YOUR-VM].
NAME   ZONE        MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP    EXTERNAL_IP   STATUS
my-vm  us-west1-a  n1-standard-8               10.138.15.222  34.105.81.92  RUNNING
```

Write down the External IP address, as we'll need it for running RStudio Server later.[5]

Allow RStudio's port (8787) via a firewall rule RStudio Server runs on port 8787 of an associated IP address. Because Google Cloud by default blocks external traffic on GCE VMs for security reasons, we first need to enable the 8787 port via a firewall rule. The following command creates a firewall rule (which I'm calling "rstudio") that does exactly this.

```
$ gcloud compute firewall-rules create rstudio --allow=tcp:8787
```

Note that these firewall rules work on a per-project basis, so you should only have to create it one it one time. [6]

Logging in

Congratulations: Set-up for your GCE VM instance is already complete.

(Easy, wasn't it?)

The next step is to log in via **SSH** (i.e. **S**ecure **Sh**ell). This is a simple matter of providing your VM's name and zone. If you forget to specify the zone or haven't assigned a default, you'll be prompted.

```
$ gcloud compute ssh my-vm --zone us-west1-a
```

IMPORTANT: Upon logging into a GCE instance via SSH for the first time, you will be prompted to generate a key passphrase. Needless to say, you should *make a note of this passphrase* for future long-ins. Your passphrase will be required for all future remote log-ins to Google Cloud projects via `gcloud` and SSH from your local computer. This includes additional VMs that you create under the same project account.

Passphrase successfully created and entered, you should now be connected to your VM via SSH. That is, you should see something like the following, where "grant" and "my-vm" will obviously be replaced by your own username and VM hostname.

```
grant@my-vm:~$
```

> **Tip:** For the remainder of this lecture, I'll explicitly use the full command line prompt for any shell instance connected to the VM via SHH (i.e. `grant@my-vm:~$`). This is so that you don't get confused if we need to switch back to running commands in local-only shell instances (i.e. $ only).

Next, we'll install R.

Install R on your VM

You can find the full set of instructions and recommendations for installing R on Ubuntu here. Or you can just follow my choices below, which should cover everything that you need. Note that you should be running these commands directly in the shell that is connected to your VM.

[4] You can also set the default zone so that you don't need to specify it every time. See here.

[5] This IP address is "ephemeral" in the sense that it is only uniquely assigned to your VM while it is running continuously. This shouldn't create any significant problems, but if you prefer a static (i.e. non-ephemeral) IP address that is always going to be associated with a particular VM instance, then this is easily done. See here.

[6] While I don't cover it in this tutorial, anyone looking to install and run Jupyter Notebooks on their VM could simply amend the above command to Jupyter's default port of 8888.

```
grant@my-vm:~$ sudo sh -c 'echo "deb https://cloud.r-project.org/bin/linux/ubuntu focal-cran40/" >> /etc/apt/sou
grant@my-vm:~$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E08
grant@my-vm:~$ sudo apt update && sudo apt upgrade -y
grant@my-vm:~$ sudo apt install -y r-base r-base-dev
```

Aside: Those apt commands are referring to the Aptitude package management system. Think of it like of Homebrew for Ubuntu (and other Debian-based Linux distributions).

R is now ready to go on our VM directly from the shell.[7] However, we'd obviously prefer to use the awesome IDE interface provided by RStudio (Server). So that's what we'll install and configure next, making sure that we can run RStudio Server on our VM via a web browser from our local computer.

Install and configure RStudio Server

Download RStudio Server on your VM You should check what the latest available version of Rstudio Server is here. At the time of writing, the following is what you need:

```
grant@my-vm:~$ sudo apt install gdebi-core
grant@my-vm:~$ wget https://download2.rstudio.org/server/bionic/amd64/rstudio-server-1.3.959-amd64.deb
grant@my-vm:~$ sudo gdebi rstudio-server-1.3.959-amd64.deb ## Hit "y" when prompted
```

Add a user Now that you're connected to your VM, you might notice that you never actually logged in as a specific user. (More discussion here.) This doesn't matter for most applications, but RStudio Server specifically requires a username/password combination. So we must first create a new user and give them a password before continuing. For example, we can create a new user called "elvis" like so:

```
grant@my-vm:~$ sudo adduser elvis
```

You will then be prompted to specify a user password (and confirm various bits of biographical information which you can ignore). An optional, but recommended step is to add your new user to the sudo group. We'll cover this in more depth later in the tutorial, but being part of the sudo group will allow Elvis to temporarily invoke superuser priviledges when needed.

```
grant@my-vm:~$ sudo usermod -aG sudo elvis
# grant@my-vm:~$ su - elvis ## Log in as elvis on SSH (optional)
```

> **Tip:** Once created, you can now log into a user's account on the VM directly via SSH, e.g. `gcloud compute ssh elvis@my-vm --zone us-west1-a`

Navigate to the RStudio Server instance in your browser You are now ready to open up RStudio Server by navigating to the default 8787 port of your VM's External IP address. (You remember writing this down earlier, right?) If you forgot to write the IP address down, don't worry: You can find it by logging into your Google Cloud console and looking at your VM instances, or by opening up a new shell window (**not** the one currently connected to your VM) and typing:

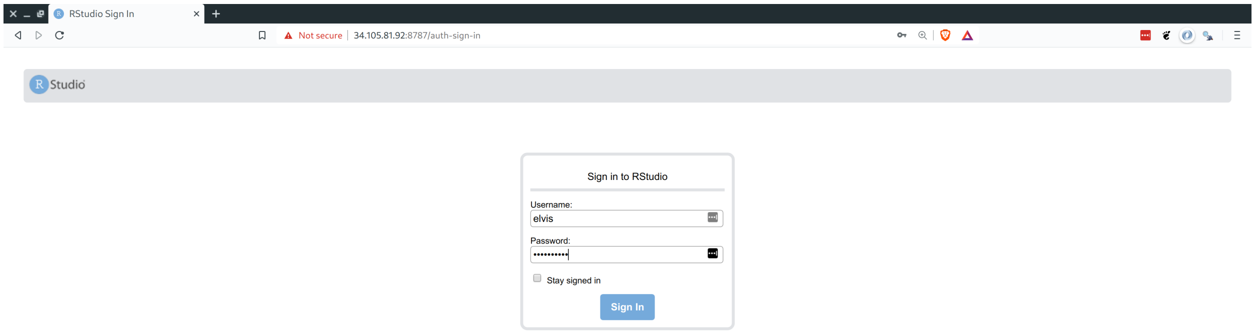
```
$ gcloud compute instances describe my-vm | grep 'natIP'
```

Either way, once you have the address, open up your preferred web browser and navigate to:

http://EXTERNAL-IP-ADDRESS:8787

You will be presented with the following web page. Log in using the username/password that you created earlier.

[7] Enter "R" into your shell window to confirm for yourself. If you do, make sure to quit afterwards by typing in "q()".



Sign in to RStudio

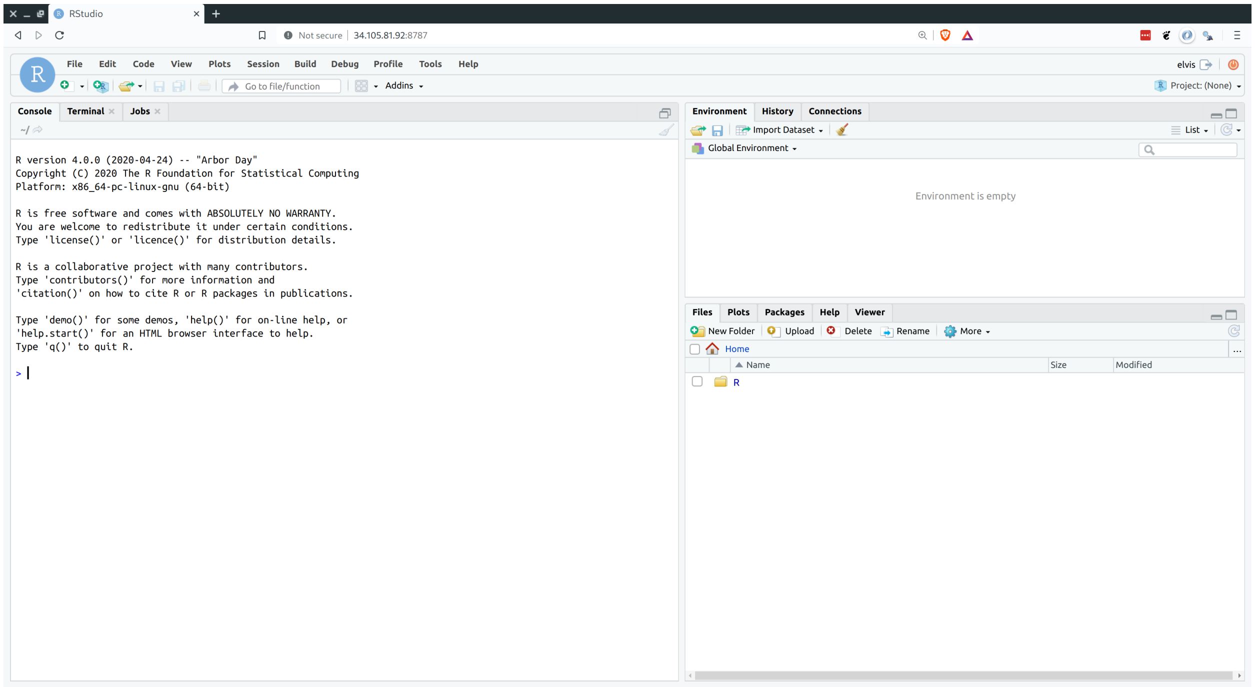
Username: elvis

Password: ••••••••••

□ Stay signed in

Sign In

And we're all set. Here is RStudio Server running on my laptop via Google Chrome.



```
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-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.

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.

> 
```

> **Tip:** Hit F11 to go full screen in your browser. The server version of RStudio is then virtually indistinguishable from the desktop version.

Stopping and (re)starting your VM instance

Stopping and (re)starting your VM instance is a highly advisable, since you don't want to get billed for times when you aren't using it. In a new shell window (not the one currently synced to your VM instance):

```
$ gcloud compute instances stop my-vm
$ gcloud compute instances start my-vm
```

Summary

Contratulations! You now have a fully-integrated VM running R and RStudio whenever you need it. Assuming that you have gone through the initial setup, here's the **tl;dr** summary of how to deploy an existing VM with RStudio Server:

1) Start up your VM instance.

```
$ gcloud compute instances start YOUR-VM-INSTANCE-NAME
```

2) Take note of the External IP address for step 3 below.

```
$ gcloud compute instances describe YOUR-VM-INSTANCE-NAME | grep 'natIP'
```

3) Open up a web browser and navigate to RStudio Server on your VM. Enter your username and password as needed.
   http://EXTERNAL-IP-ADDRESS:8787

4) Log-in via SSH. (Optional)

```
$ gcloud compute ssh YOUR-VM-INSTANCE-NAME
```

5) Stop your VM.

```
$ gcloud compute instances stop YOUR-VM-INSTANCE-NAME
```

And, remember, if you really want to avoid the command line, then you can always go through the GCE browser console.

Getting the most out of R on your GCE setup

You have already completed all of the steps that you'll need for high-performance computing in the cloud. Any VM that you create on GCE using the above methods will be ready to go with RStudio Server whenever you want it. However, there are still a few more tweaks and tips that we can use to really improve our user experience and reduce complications when interacting with these VMs from our local computers. The rest of this tutorial covers my main tips and recommendations.

Keep your system up to date

First things first: Remember to keep your VM up to date, just like you would a normal computer. I recommend that you run the following two commands regularly:

```
grant@my-vm:~$ sudo apt update
grant@my-vm:~$ sudo apt upgrade
```

You can also update the `gcloud` utility components on your local computer (i.e. not your VM) with the following command:

```
$ gcloud components update
```

Installing R packages on Linux

So far we've only installed base R on our VM. We'd almost assuredly like to install some additional packages on it too (tidyverse, data.table, whatever). Now, you can certainly install packages the regular way that you're used to, e.g. clicking the "Packages" tab in RStudio and then searching for your library of choice. However, R package installation on a Linux system comes with a few additional considerations that I want to highlight here.

The first thing to note is that, unlike on Mac and Windows, R packages on Linux do not come as precompiled binaries by default. Instead, they must be compiled from source. I don't want to spend too much time on what it means to a "binary" vs "compiled from source". But a not-unreasonable-analogy is that a binary is like a cake that we bought from the supermarket, while compiling from source is like us baking the cake at home. Why would you want to buy the cake from a supermarket? Well, it's quicker and you might not have all the ingredients at home. Why would you make the cake yourself? Well, you might have better quality ingredients and know slightly better recipe.

Metaphors aside, the upshot is that there are two ways to install R packages on Linux. Both have their merits, so I'll let you decide which you prefer. (It's fine to combine them too.)

Option 1: Install R packages from source (i.e. `install.packages()` & co.) The first option is just to use the regular install method from within R that you're used to: `install.packages()`, `pacman::p_load()`, etc. However, as noted above, this will install from source. In addition to taking quite a bit longer (especially for first-time installs), a number of important R packages require external Linux libraries that must be installed separately on your VM first. For an Ubuntu, we can install these external packages with the following commands.

1) For the "tidyverse" suite of packages:

```
grant@my-vm:~$ sudo apt install -y libcurl4-openssl-dev libssl-dev libxml2-dev
```

2) For the main spatial libraries (sf, sp, rgeos, etc.):

```
grant@my-vm:~$ sudo add-apt-repository -y ppa:ubuntugis/ubuntugis-unstable
grant@my-vm:~$ sudo apt update && apt upgrade -y
grant@my-vm:~$ sudo apt install -y libgeos-dev libproj-dev libgdal-dev libudunits2-dev
```

You should now be fine to install these packages directly from within R/Rstudio as per usual with `install.packages()` and co.

Again, this will take longer than you're probably used to because everything has to be compiled, so my final two tips to greatly speed things up are: i) make sure your packages are installed in parallel by adding `options(Ncpus=parallel::detectCores())` to your `~./Rprofile` file, and/or ii) cache your packages using ccache.

Option 2: Install R package binaries on Ubuntu As I keep saying, installing R packages on a Linux system can come as a bit of shock for first-time users. It takes much longer than Mac or Windows because everything has to compiled from source. The good news is that pre-compiled binaries *are* available on Ubuntu thanks to Mark Rutter. This means we can greatly speed up package installation and reduce other complications related to external dependencies. More information here, but the short version is that you can open up your terminal and try out the following.

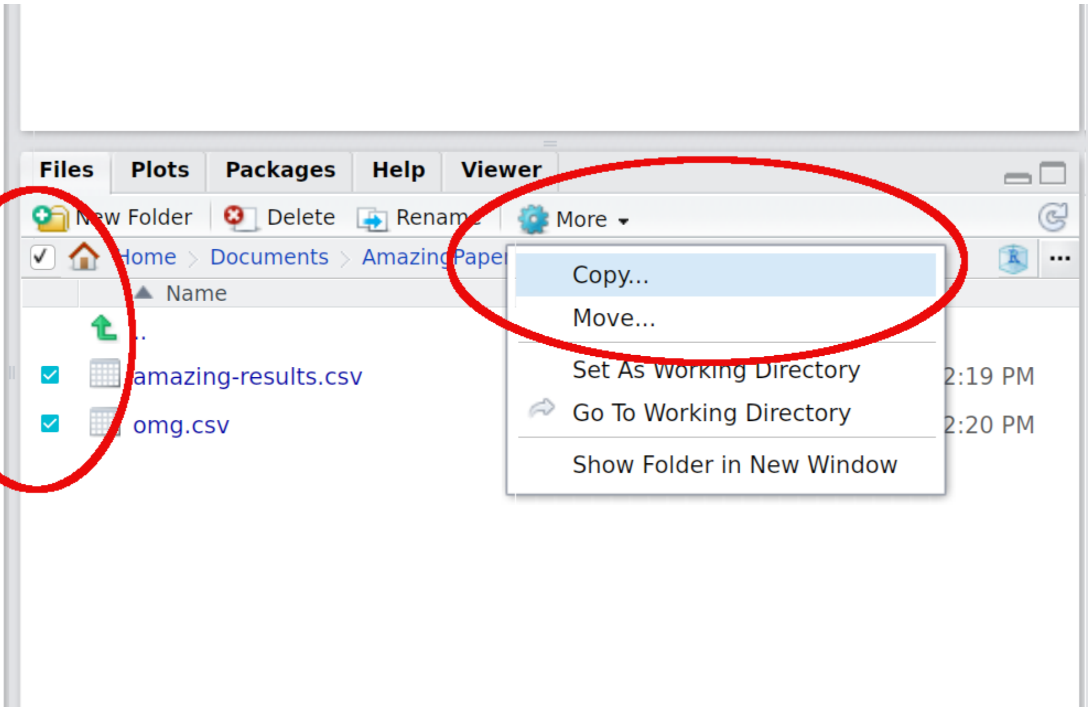
```
#  Add the PPA(s)
# grant@my-vm:~$ sudo add-apt-repository -y ppa:marutter/rrutter4.0 ## Another option for base R (we've already
grant@my-vm:~$ sudo add-apt-repository -y ppa:c2d4u.team/c2d4u4.0+
grant@my-vm:~$ sudo apt update

# List all of the available binaries (press 'q' to exit)
grant@my-vm:~$ apt-cache search r-cran- | sort | less

# Or, to see if a specific package is available:
grant@my-vm:~$ apt-cache search r-cran | grep “tidy”

# Install your package(s) of choice, e.g.:
grant@my-vm:~$ sudo apt install -y r-cran-tidyverse r-cran-rstan
```

Note that this approach requires you to install packages from your terminal using Aptitude (i.e. Ubuntu's built-in package manager that I mentioned earlier in the lecture). So, not the traditional `install.packages()` command that you're used to from within R. On the plus side, this means that Aptitude will take responsibility for updating your packages (assuming you run `$ sudo apt upgrade` occasionally) and all the packages will still be available/viewable to you from inside R regardless.

Transfer and sync files between your VM and your local computer

You have three main options.

1. Manually transfer files directly from RStudio Server This is arguably the simplest option and works well for copying files from your VM to your local computer. However, I can't guarantee that it will work as well going the other way; you may need to adjust some user privileges first.



2. Manually transfer files and folders using the command line or SCP Manually transferring files or folders across systems is done fairly easily using the command line. Note that this next code chunk would be run in a new shell instance (i.e. not the one connected to your VM via SSH).

```
$ gcloud compute scp my-vm:/home/elvis/Papers/MyAwesomePaper/amazingresults.csv ~/local-directory/amazingresults
```

It's also possible to transfer files using your regular desktop file browser thanks to SCP. (On Linux and Mac OSX at least. Windows users first need to install a program call WinSCP.) See here.

> **Tip:** The file browser-based SCP solution is much more efficient when you have assigned a static IP address to your VM instance — otherwise you have to set it up each time you restart your VM instance and are assigned a new ephemeral IP address — so I'd advise doing that first.

3. Sync with GitHub or other cloud service This is my own preferred option. Ubuntu, like all virtually Linux distros, comes with Git preinstalled. You should thus be able to sync your results across systems using Git(Hub) in the usual fashion. I tend to use the command line for all my Git operations (committing, pulling, pushing, etc.) and this works exactly as expected once you've SSH'd into your VM. However, Rstudio Server's built-in Git UI also works well and comes with some nice added functionality (highlighted diff. sections and so forth).

While I haven't tried it myself, you should also be able to install Box, Dropbox or Google Drive on your VM and sync across systems that way. If you go this route, then I'd advise installing these programs as sub-directories of the user's "home" directory. Even then you may run into problems related to user permissions. However, just follow the instructions for linking to the hypothetical "TeamProject" folder that I describe below (except that you must obviously point towards the relevant Box/Dropbox/GDrive folder location instead) and you should be fine.

> **Tip:** Remember that your VM lives on a server and doesn't have the usual graphical interface — including installation utilities — of a normal desktop. You'll thus need to follow command line installation instructions for these programs. Make sure you scroll down to the relevant sections of the links that I have provided above.

Last, but not least, Google themselves encourage data synchronisation on GCE VMs using another product within their Cloud Platform, i.e. Google Storage. This is especially useful for really big data files and folders, but beyond the scope of this lecture. (If you're interested in learning more, see here and here.)

Further resources

In the next lecture, I'll build on today's material by showing you how to automate a lot of steps with the **googleComputeEngineR** package and related tools. In the meantime, here are some further resources that you might find useful.

- I recommend consulting the official GCE documentation if you ever get stuck. There's loads of useful advice and extra tips for getting the most out of your VM setup, including ways to integrate your system with other GCP products like Storage, BigQuery, etc.
- Other useful links include the RStudio Server documentation, and the Linux Journey guide for anyone who wants to learn more about Linux (yes, you!).

Bonus material

Install the Intel Math Kernel Library (MKL) or OpenBLAS/LAPACK

As we discussed in the previous lecture on parallel programming, R ships with its own BLAS/LAPACK libraries by default. While this default works well enough, you can get *significant* speedups by switching to more optimized libraries such as the Intel Math Kernel Library (MKL) or OpenBLAS. The former is slightly faster according to the benchmark tests that I've seen, but was historically harder to install. However, thanks to Dirk Eddelbuettel, this is now very easily done:

```
grant@my-vm:~$ git clone https://github.com/eddelbuettel/mkl4deb.git
grant@my-vm:~$ sudo bash mkl4deb/script.sh
```

Wait for the script to finish running. Once it's done, your R session should automatically be configured to use MKL by default. You can check yourself by opening up R and checking the `sessionInfo()` output, which should return something like:

```
Matrix products: default
BLAS/LAPACK: /opt/intel/compilers_and_libraries_2018.2.199/linux/mkl/lib/intel64_lin/libmkl_rt.so
```

(Note: Dirk's script only works for Ubuntu and other Debian-based Linux distros. If you decided to spin up a different OS for your VM than we did in this tutorial, then you are probably better off installing OpenBLAS.)

Share files and libraries between multiple users on the same VM

The default configuration that I have described above works perfectly well in cases where you are a single user and don't venture outside of your home directory (and its sub directories). Indeed, you can just add new folders within this user's home directory using standard Linux commands and you will be able to access these from within RStudio Server when you log in as that user.

However, there's a slight wrinkle in cases where you want to share information between *multiple* users on the same VM. (Which may well be necessary on a big group project.) In particular, RStudio Server is only going to be able to look for files in each individual user's home directory (e.g. `/home/elvis`.) Similarly, by default on Linux, the R libraries that one user installs won't necessarily be available to other users.

The reason has to do with user permissions; since Elvis is not an automatic "superuser", RStudio Server doesn't know that he is allowed to access other users' files and packages in our VM, and vice versa. Thankfully, there's a fairly easy workaround, involving standard Linux commands for adding user and group privileges (see these slides from our shell lecture). Here's an example solution that should cover most cases:

Share files across users Let's say that Elvis is working on a joint project together with a colleague called Priscilla. (Although, some say they are more than colleagues…) They have decided to keep all of their shared analysis in a new directory called `TeamProject`, located within Elvis's home directory. Start by creating this new shared directory:

```
$ grant@my-vm:~$ sudo mkdir /home/elvis/TeamProject
```

Presumably, a real-life Priscilla would already have a user profile at this point. But let's quickly create one too for our fictional version.

```
$ grant@my-vm:~$ sudo adduser priscilla
```

Next, we create a user group. I'm going to call it "projectgrp", but as you wish. The group setup is useful because once we assign a set of permissions to a group, any members of that group will automatically receive those permissions too. With that in mind, we should add Elvis and Priscilla to "projectgrp" once it is created:

```
grant@my-vm:~$ sudo groupadd projectgrp
grant@my-vm:~$ sudo gpasswd -a elvis projectgrp
grant@my-vm:~$ sudo gpasswd -a priscilla projectgrp
```

Now we can set the necessary ownership permissions to the shared `TeamProject` directory. First, we use the `chown` command to assign ownership of this directory to a default user (in this case, "elvis") and the other "projectgrp" members. Second, we use the `chmod 770` command to grant them all read, write and execute access to the directory. In both both cases, we'll use the `-R` flag to recursively set permissions to all children directories of `TeamProject/` too.

```
grant@my-vm:~$ sudo chown -R elvis:projectgrp /home/elvis/TeamProject
grant@my-vm:~$ sudo chmod -R 770 /home/elvis/TeamProject
```

The next two commands are optional, but advised if Priscilla is only going to be working on this VM through the `TeamProject` directory. First, you can change her primary group ID to "projectgrp", so that all the files she creates are automatically assigned to that group:

```
grant@my-vm:~$ sudo usermod -g projectgrp priscilla
```

Second, you can add a symbolic link to the `TeamProject` directory in Priscilla's home directory, so that it is immediately visible when she logs into RStudio Server. (Making sure that you switch to her account before running this command):

```
grant@my-vm:~$ sudo su - priscilla
priscilla@my-vm:~$ ln -s /home/elvis/TeamProject /home/priscilla/TeamProject
priscilla@my-vm:~$ exit
```

Share R libraries (packages) across users Sharing R libraries across users is less critical than being able to share files. However, it's still annoying having to install, say, the **tidyverse** when your colleague has already installed it under her user account. Luckily, the solution to this annoyance very closely mimics the solution to file sharing that we've just seen above: We're going to set a default system-wide R library path and give all of our users access to that library via a group. For convenience I'm just going to contine with the "projectgrp" group that we created above. However, you could also create a new group (say, "rusers"), add individual users to it, and proceed that way if you wanted to.

The first thing to do is determine where your exisiting system-wide R library is located. Open up your R console and type (without the ">" prompt):

```
> .libPaths()
```

This will likely return several library paths. The system-wide library path should hopefully be pretty obvious (e.g. no usernames) and will probably be one of `/usr/lib/R/library` or `/usr/local/lib/R/site-library`. In my case, it was the former, but adjust as necessary.

Once we have determined the location of our system-wide library directory, we can recursively assign read, write and execute permissions to it for all members of our group. Here, I'm actually using the parent directory (i.e. `.../R` rather than `.../R/library`), but it should work regardless. Go back to your shell and type:

```
grant@my-vm:~$ sudo chown elvis:projectgrp -R /usr/lib/R/ ## Get location by typing ".libPaths()" in your R cons
grant@my-vm:~$ sudo chmod -R 775 R/
```

Once that's done, tell R to make this shared library path the default for your user, by adding it to their `~/.Renviron` file:

```
grant@my-vm:~$ su - elvis
elvis@my-vm:~$ sudo echo 'export PATH="R_LIBS_USER=/usr/lib/R/library"' >> ~/.Renviron
```

The R packages that Elvis installs should now be immediately available to Priscilla and vice versa.

> **Tip:** If you've already installed some packages in a local (i.e. this-user-only) library path before creating the system-wide setup, you can just move them across with the 'mv' command. Something like the following should work, but you'll need to check the appropriate paths yourself: `elvis@my-vm:~$ sudo mv "/home/elvis/R/x86_64-pc-linux-gnu-library/3.5/*" /usr/lib/R/library`.