**How to build a scaleable computing cluster on AWS and run hundreds or  
thousands of models in a short amount of time. We will completely rely on R and  
open source R packages. This is post 1 out of 2.**

**Introduction**

An ever-increasing number of businesses is moving to the cloud and using  
platforms such as [Amazon Web Services](https://aws.amazon.com/)(AWS)  
for their data infrastructure. This is convenient for  
Data Scientists like myself because this conversion of tools means that my  
knowledge from previous jobs becomes much more applicable to a new role and  
I can hit the ground running.

Lately I have become very excited about the  
[future](https://cran.r-project.org/web/packages/future/vignettes/future-1-overview.html)  
package and how it makes the scaling of computational tasks easy and intuitive.  
The basic idea of the future package is to make your code infrastructure  
independent. Specify your tasks and the future execution plan decides  
how to run the calculations.

I wanted to see what we could do with future and other open source R packages  
such as [aws.ec2](https://rdrr.io/github/cloudyr/aws.ec2/f/README.md) by  
[cloudyR](http://cloudyr.github.io/packages/),  
[ssh](https://ropensci.org/technotes/2018/06/12/ssh-02/)  
by [rOpenSci](https://ropensci.org/),  
[remoter](https://cran.r-project.org/web/packages/remoter/vignettes/remoter.pdf)  
by Drew Schmidt, and last but not least [furrr](https://davisvaughan.github.io/furrr/)  
by Davis Vaughan.

The basic idea:

* use R and AWS to spin up our own cloud compute cluster
* log in to the head node and define a computationally expensive task
* farm this task out to a number of worker nodes in our cluster
* do all of this WITHOUT having to switch between RStudio, RStudioServer, the command line, the AWS console, etc.

Why do I care about the last point? Well, Data Science is a science and should  
rely on the [Scientific Method](https://en.wikipedia.org/wiki/Scientific_method).  
One core component of the Scientific Method is reproducibility, and one of the  
best ways to keep your Data Science workflow reproducible is to write code that  
can run start to finish without any user intervention. This also allows for  
greater applicability in the future because you can re-use your previous data  
product or service in the next project without retracing manual steps.  
Don’t just take my word for it, here is another great Hadley  
Wickham video in which he stresses the same point:

So without further ado, let’s get started implementing that bullet point list!

**Preparation**

There are a few basic requirements that need to be in place:

1. an active AWS account.
2. an Amazon Machine Image ([AMI](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html))  
   with R, remoter, tidyverse, future, and furrr installed.
3. a working ssh key pair on your local machine and the AMI that allows you  
   to ssh into and between your ec2 instances.

Detailed instructions on how to fulfil these basic requirements are beyond the  
scope of this post. You can find more information in the articles linked below.

* [What Is Amazon EC2?](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html)
* [Running R on AWS](https://aws.amazon.com/blogs/big-data/running-r-on-aws/)
* [The CloudyR project](http://cloudyr.github.io/)

**Setup**

Load the required packages. Also make sure your AWS access credentials are set.  
I do this using Sys.setenv. There is other ways but I found that this works  
best for me. We also specify the AMI ID and the instance type (this is a good  
[overview](https://aws.amazon.com/ec2/instance-types/); I am using t2.micro  
here because it is free). If you have any problems with this step, double-  
check that the region set in Sys.setenv matches the region of your AMI.

library(aws.ec2)

library(ssh)

library(remoter)

library(tidyverse)

# set access credentials

aws\_access <- aws.signature::locate\_credentials()

Sys.setenv(

"AWS\_ACCESS\_KEY\_ID" = aws\_access$key,

"AWS\_SECRET\_ACCESS\_KEY" = aws\_access$secret,

"AWS\_DEFAULT\_REGION" = aws\_access$region

)

# set parameters

aws\_ami <- "ami-06485bfe40a86470d"

aws\_describe <- describe\_images(aws\_ami)

aws\_type <- "t2.micro"

Ready for launch!

**Boot and Connect**

We can now fire up our head-node instance.

ec2inst <- run\_instances(

image = aws\_ami,

type = aws\_type)

# wait for boot, then refresh description

Sys.sleep(10)

ec2inst <- describe\_instances(ec2inst)

# get IP address of the instance

ec2inst\_ip <- get\_instance\_public\_ip(ec2inst)

The instance should be running and we can connect to it via ssh in bash.  
That works, but personally I’d prefer to stay in RStudio instead of switching  
to the command line. This is where remoter and ssh come in. We can  
establish an ssh connection straight from our R session and use that to launch  
the remoter::server on our instance. By using the future package to run the  
ssh command we keep our interactive RStudio session free and can subsequently  
use it to connect to the instance with remoter

# ssh connection

username <- system("whoami", intern = TRUE)

con <- ssh\_connect(host = paste(username, ec2ip, sep = "@"))

# helper function for a random temporary password

random\_tmp\_password <- generate\_password()

# CMD string to start remoter::server on instance

r\_cmd\_start\_remoter <- str\_c(

"sudo Rscript -e ",

"'remoter::server(",

"port = 55555, ",

"password = %pwd, ",

"showmsg = TRUE)'",

collapse = "") %>%

str\_replace("%pwd", str\_c('"', random\_tmp\_password, '"'))

# connect and execute

plan(multicore)

x <- future(

ssh\_exec\_wait(

session = con,

command = r\_cmd\_start\_remoter))

remoter::client(

addr = ec2ip,

port = 55555,

password = random\_tmp\_password,

prompt = "remote")

Et Voila! We are connected to our remote head node and can run R code  
in the cloud without ever leaving the comfort of RStudio. And the amazing bit:  
all of this took me about a day to set up from scratch!

I will leave it here for now. In the next post we will dive into the  
details of how to scale up the approach above to create an AWS cloud computing  
cluster. This approach is extremely powerful for embarrassingly parallel  
problems (which are actually not embarrassing at all, I swear!)

As always, I hope it is useful for you. I’d very much appreciate any thoughts,  
comments, and feedback so write me a message below or get in touch via twitter!