Statistics in the Amazon Cloud with R

Louis J. M. Aslett (louis.aslett@durham.ac.uk)

Department of Mathematical Sciences
Durham University

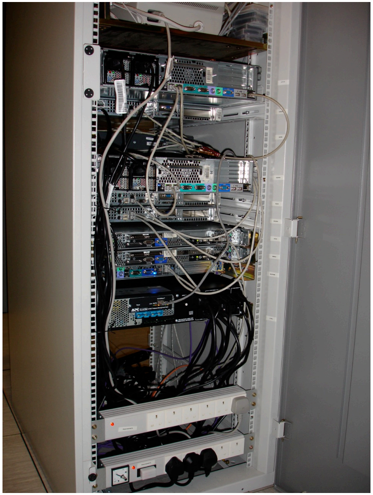
& The Alan Turing Institute

CASI 2019

15th May 2019

Introduction

“The old days …”



Redbus Interhouse, Harbour Exchange Square, London Docklands, 24 March 2005

R in the cloud

"Cloud" ... an online service which allows users to create and destroy virtual servers remotely without having to worry about initial hardware and OS installation and where billing is in very small increments (e.g. hours).

There are several cloud providers, including:

- Amazon EC2
  - http://aws.amazon.com/
- Windows Azure VMs
  - http://www.windowsazure.com/
- Google Compute Engine
  - http://cloud.google.com/
- Digital Ocean
  - http://www.digitalocean.com/
- Rackspace Cloud Servers
  - http://www.rackspace.co.uk/

Why R in the cloud?

- Long analyses
- Collaboration
- Powerful server machines, with access to GPUs
- Instant large clusters
- Online data sources
- Full environment with C/C++/Fortran, $\LaTeX$+ Sweave, Git + Subversion ready-to-go
- Access to the power of Linux without having to dedicate your own machine to running it
- Convenient in the era of mostly laptops

Why Amazon Web Services (AWS)?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for HPC in statistics, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).

Why Amazon Web Services (AWS)?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for HPC in statistics, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.

Why Amazon Web Services (AWS)?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for HPC in statistics, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.

Why Amazon Web Services (AWS)?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for HPC in statistics, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.
- A free tier of instances so everyone can try it without cost.

Why Amazon Web Services (AWS)?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for HPC in statistics, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.
- A free tier of instances so everyone can try it without cost.
- Everything from micro instances to the current state of the art in HPC, including machines with nVidia GPUs for CUDA support. (See also public benchmarks)

Why Amazon Web Services (AWS)?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for HPC in statistics, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.
- A free tier of instances so everyone can try it without cost.
- Everything from micro instances to the current state of the art in HPC, including machines with nVidia GPUs for CUDA support. (See also public benchmarks)
- A 'stock-market' for unused compute capacity, enabling heavily discounted compute jobs.

Some cons to AWS

There are some cons to AWS, though again these could be addressed in future:

- There are no guarantees of availability: Amazon explicitly intend users to design for instance failure in the use cases. There have been high profile outages in the US East Coast data centre
- *High* barrier to learning

AWS Background

Amazon Web Services (AWS)

Launched 2006. By December 2014, 1,400,000 servers operating. Since then, roughly doubled in scale, now with over 64 data centres across 14 countries:

- Dublin, Ireland
- London, UK
- Paris, France
- Frankfurt, Germany
- Stockholm, Sweden
- Montreal, Canada
- North Virginia, USA
- Ohio, USA
- Oregon, USA
- Northern California, USA
- São Paulo, Brazil
- Singapore, Republic of Singapore
- Tokyo, Japan
- Hong Kong
- Seoul, South Korea
- Sydney, Australia
- Mumbai, India

Some of the relevant AWS ecosystem

- EC2 (Elastic Compute Cloud)
  - is the service which enables launching virtual servers
- EBS (Elastic Block Storage)
  - persistent block level storage for use with EC2
- VPC (Virtual Private Cloud)
  - for creation of virtual LANs within AWS for private networking
- S3 (Simple Storage Service)
  - for resiliant storage of data independent of instances
- RDS (Relational Database Server) / DynamoDB
  - managed SQL and NoSQL databases
- SQS (Simple Queuing Service)
  - highly scalable and reliable atomic messaging service

EC2 Jargon

- Instance
    - a virtual server running on EC2
- Volume
    - a cloud ‘hard drive’ which is attached to an instance
- AMI (Amazon Machine Image)
    - a bundle of operating system and pre-loaded applications to boot on an instance
- Stop -vs- terminate
    - *stop*: similar to powering down a computer. Cease paying by the hour, just pay by the month for EBS volume ($0.10/GB/mo). Start and instance boots same machine.
    - *terminate*: power down and destroy all data.
- On-demand/Spot instance
    - *on demand*: immediately available, fixed price per hour
    - *spot*: ephemeral instance, price follows Amazon ‘stock-market’

Demo: Amazon Web Services Console

Demo (tour of interface)

Interacting with EC2

- Web interface, https://console.aws.amazon.com/
- REST API accessible from an array of languages/platforms
  - Android, iOS, *nix, Mac, Windows
  - Javascript, Java, .NET, Node.js, PHP, Python, Ruby, Go, shell
  - interestingly, no official R API yet! However, http://cloudyr.github.io/ looks promising.
  - Official command line is a Python package https://aws.amazon.com/cli/

Interacting with EC2 : official CLI

```
$ pip install aws
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]:
```

Access Key ID and AWS Secret Access Key can be setup at Name > My Security Credentials > Access Keys.

The RStudio AMI

Why?

So …

- AWS enables rapid launching of instances from a set of different operating systems.

Why?

So …

- AWS enables rapid launching of instances from a set of different operating systems.
- However, they’re bare bones systems
  - it’s up to you to setup up the system as you want it
  - not hard for the technically literate … but definitely *time consuming*

Why?

So ...

- AWS enables rapid launching of instances from a set of different operating systems.
- However, they're bare bones systems
  - it's up to you to setup up the system as you want it
  - not hard for the technically literate ... but definitely *time consuming*
- Any AWS user can create AMIs of systems so that they can boot directly to a system they have previously setup
  - an AMI is like a bare metal image of a system which can be restored to any new instance.

Why?

So ...

- AWS enables rapid launching of instances from a set of different operating systems.
- However, they're bare bones systems
  - it's up to you to setup up the system as you want it
  - not hard for the technically literate ... but definitely *time consuming*
- Any AWS user can create AMIs of systems so that they can boot directly to a system they have previously setup
  - an AMI is like a bare metal image of a system which can be restored to any new instance.
- The AMIs I provide is a public share of the pre-setup R system I use and have integrated user feedback and requests.

See http://www.louisaslett.com/RStudio_AMI/

What’s included?

- 20GB EBS image on SSD backed storage

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)
- Git & Subversion support

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full $\LaTeX$ distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)
- Git & Subversion support
- Probabilistic programming languages Stan and JAGS (Greta supported) for MCMC sampling

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full $\LaTeX$ distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)
- Git & Subversion support
- Probabilistic programming languages Stan and JAGS (Greta supported) for MCMC sampling
- CUDA and cuDNN for Tensorflow/Keras support on GPUs

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)
- Git & Subversion support
- Probabilistic programming languages Stan and JAGS (Greta supported) for MCMC sampling
- CUDA and cuDNN for Tensorflow/Keras support on GPUs
- Optimised BLAS for accelerated matrix operations

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)
- Git & Subversion support
- Probabilistic programming languages Stan and JAGS (Greta supported) for MCMC sampling
- CUDA and cuDNN for Tensorflow/Keras support on GPUs
- Optimised BLAS for accelerated matrix operations
- Shiny server for public deployment of Shiny apps

What's included?

- 20GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Dropbox integration
- Full $\LaTeX$ distribution, supporting RMarkdown and Sweave
- GSL and CURL
- GDAL for GIS packages
- Database support (ODBC)
- Git & Subversion support
- Probabilistic programming languages Stan and JAGS (Greta supported) for MCMC sampling
- CUDA and cuDNN for Tensorflow/Keras support on GPUs
- Optimised BLAS for accelerated matrix operations
- Shiny server for public deployment of Shiny apps
- Arbitrary precision arithmetic and number theory libraries
    - GMP, MPFR, FLINT

What's not included?

- Every R package (obviously!). But, fast AWS based mirror is automatically set for quick installation of new pacakges.
- Easy cluster setup (yet … some things perhaps harder to automate well)

Most important thing to remember …

… is to **allow port 80 through the AWS EC2 firewall!**

This is done by setting up your security group on instance launch.

Demo

Demo (including launch, Dropbox, RMarkdown, Stan, stop/start)

Advanced usage

Passwordless SSH

By default password is the instance ID. Can avoid needing this with a public key, which can be assigned to any number of instances.

1. Generate keypair in AWS > EC2 > Network & Security > Key Pairs
2. Save locally to `key.pem` file
3. Select keypair on instance launch
4. Connect using ssh:

```
ssh -i key.pem ubuntu@<IP address>
```

NB: passwordless SSH is to `ubuntu` user, password based SSH is to `rstudio` user.

Connecting securely

By default the connection is unsecured … automatic SSL cert *not* likely to come soon!

Connecting securely

By default the connection is unsecured … automatic SSL cert *not* likely to come soon!

Instead, recommend creating an SSH tunnel.

```
sudo ssh -L80:localhost:80 rstudio@<IP address>
```

OR

```
sudo ssh -i key.pem -L80:localhost:80 ubuntu@<IP address>
```

Connect to http://localhost/ in browser: all traffic encrypted and sent over SSH tunnel.

Windows users can create tunnel using PuTTY.

Useful system tools

It is easy to access the command line via the Terminal tab, or over SSH.

- `htop` — for monitoring system utilisation
- `sudo` — to act as administrator
- `sudo apt-get install ...` — to install system packages
  - eg `sudo apt-get install texlive-xetex` for XeLaTeX
- `df -h /` — view current disk usage
- `nvidia-smi -l 5` — view current GPU utilisation (only GPU instances)

Expanding storage

Can expand the instance root volume to over 16000GB online!

1. Goto AWS > EC2 > Elastic Block Store > Volumes
2. Actions > Modify Volume
3. Choose new size, click 'Modify'
4. Login to RStudio > Terminal tab
5. Check:

```
sudo file -s /dev/nvme?n*
sudo file -s /dev/xvd*
```

1. Depending which command found a file:
   - i. first (`nvme?n*`) command:
     - a. `sudo growpart /dev/nvme0n1 1`
     - b. `sudo resize2fs /dev/nvme0n1p1`
   - ii. second (`xvd*`) command:
     - a. `sudo growpart /dev/xvda 1`
     - b. `sudo resize2fs /dev/xvda1`

Spot instances

Amazon run a ‘stock market’ for unused capacity. Prices often close to bare power costs.

- Set a ‘maximum bid’ price;
- If market price is below, your request is filled;
- Pay fluctuating market price, upto your maximum bid;
- If market price exceeds maximum bid, you may be terminated without warning!

Spot instances

Amazon run a 'stock market' for unused capacity. Prices often close to bare power costs.

- Set a 'maximum bid' price;
- If market price is below, your request is filled;
- Pay fluctuating market price, upto your maximum bid;
- If market price exceeds maximum bid, you may be terminated without warning!

Savings can be very significant. Finding the price not so easy, see spot price checking script at this Github Gist:

https://gist.github.com/louisaslett/ee69c7988210ad3f0871a1cba050771f

Cluster computing (I)

Say my private RSA key is in k.pem. First, upload to the instance, then make the correct permissions for SSH:

```
system("chmod 600 k.pem")
```

Now, launch R across the cluster with one command:

```
library("parallel")
cl <- makePSOCKcluster(
  rep(c("localhost", "10.0.0.58", "10.0.4.249"), 36),
  rshcmd="ssh -i k.pem -o StrictHostKeyChecking=no",
  user="ubuntu",
  master=system("hostname --all-ip-addresses", TRUE))
```

where 10.0.0.58 etc are the VPC private IP addresses ...

Cluster computing (II)

How to get all the IP addresses easily? Amazon CLI!

```
aws --region eu-north-1 ec2 describe-instances \
  --output text | grep ^INSTANCES | cut -f 15
```

Cluster computing (II)

How to get all the IP addresses easily? Amazon CLI!

```
aws --region eu-north-1 ec2 describe-instances \
  --output text | grep ^INSTANCES | cut -f 15
```

Ensure that all the packages you need are installed on all the nodes (see example script), then:

- use `clusterEvalQ()`, `clusterApply()` or `clusterApplyLB()` to launch parallel jobs.
- or optionally hook into the foreach framework, for easy to use `%dopar%` construct:

```
library("doParallel")
registerDoParallel(cl)
```

Cluster computing (III)

For example, fitting a random forest on large data is then trivial:

```
library("randomForest")
fit <- foreach(nt=1:length(cl), .combine=combine) %dopar% {
  randomForest(resp ~ ., data = myDat, ntree = 500)
}
```

Cluster computing (III)

For example, fitting a random forest on large data is then trivial:

```
library("randomForest")
fit <- foreach(nt=1:length(cl), .combine=combine) %dopar% {
  randomForest(resp ~ ., data = myDat, ntree = 500)
}
```

Advanced users can also make use of built-in OpenMPI tools for manual low level cluster programming.

H2O cluster

H2O is an excellent implementation of some machine learning methods ... can easily create a big cluster of machines.

- Cox proportional hazards
- Deep learning (multilayer perceptron)
- Random forests
- Gradient boosting machine
- GLMs
- K-means
- Naïve Bayes
- PCA
- XGboost

H2O – getting started

```
install.packages("h2o")


library("h2o")
```

I can start on my own machine for testing and modest sized problems:

```
h2o.init(max_mem_size = "12G")
```

Or, I can connect to a single server/cluster which is already running:

```
h2o.init(ip = "123.123.123.123", port = "12345")
```

H2O – loading data

Copy data from R to H2O

```
higgs.h2o <- as.h2o(higgs)
```

Load data direct from CSV

```
higgs.h2o <- h2o.importFile("HIGGS.csv", "higgs")
```

Load data direct from SQL (harder)

```
java -cp <h2o_jar_path>:<jdbc_driver_jar_path> water.H2OApp
```

```
db_url <- "jdbc:mysql://1.1.1.1:33/MyDatabase?&useSSL=false"
table <- "HIGGS"
username <- "myuser"
password <- "abc123"
higgs.h2o <- h2o.import_sql_table(connection_url,
                                  table, username,
                                  password)
```

H2O – Random Forest

We can now fit a random forest (for example)

```
fit <- h2o.randomForest(y = ”signal”,
                        training_frame = higgs.h2o,
                        ntrees = 1000)
plot(fit)
h2o.saveModel(fit, ”higgs_rf.h2o”)
```

H2O – Cluster

Demo (scripts available, connect to H2O Flow using ...)

```
ssh -i k.pem -L54321:localhost:54321 ubuntu@<IP address>
```

Tensorflow/Keras

Standard CPU version can be installed, but launching a GPU instance will enable use of CUDA/cuDNN which are pre-installed. Simply setup `tensorflow/keras` in R ...

```r
# Tensorflow
install.packages("tensorflow")
library("tensorflow")
install_tensorflow(version = "gpu")

# Keras
install.packages("keras")
library("keras")
install_keras(tensorflow = "gpu")
```

Demo (scripts available, selecting GPU instances)

Spark / EMR

Java 8 enables:

- Spark server to run locally on the instance (for development);
- and can be integrated with Elastic Map Reduce Amazon clusters for computing in earnest
  - see https://aws.amazon.com/blogs/big-data/launch-an-edge-node-for-amazon-emr-to-run-rstudio/

Case Study

Encrypted statistical machine learning

Modern cryptographic techniques promise to allow privacy to be preserved whilst still allowing computation to be performed, unlike the usual encryption schemes such as AES.

However, these so-called *homomorphic encryption* schemes are currently very computationally demanding and restrictive in the types of computation which can be performed.

Recent work[1] has shown that tailored methods can be built which are competitive with unencrypted machine learning methods, but computational demand is high.

[1] Aslett, L. J. M., Esperança, P. M. and Holmes, C. C. (2015), Encrypted statistical machine learning: new privacy preserving methods. arXiv:1508.06845 [stat.ML].

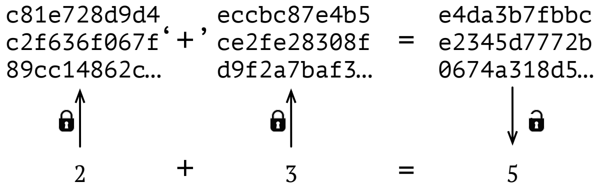
Homomorphic encryption

Definition (Homomorphic encryption scheme)

An encryption scheme is said to be *homomorphic* if there is a set of operations $\circ \in \mathcal{F}_M$ acting in message space (such as addition) that have corresponding operations $\diamond \in \mathcal{F}_C$ acting in cipher text space satisfying the property:

$$\texttt{Dec}(k_s, \texttt{Enc}(k_p, m_1) \diamond \texttt{Enc}(k_p, m_2)) = m_1 \circ m_2 \quad \forall\, m_1, m_2 \in M$$

A scheme is *fully homomorphic* if $\mathcal{F}_M = \{+, \times\}$ and an arbitrary number of such operations are possible. Cartoon version:



Fitting a Completely Random Forest (CRF) encrypted

To show the techniques of the paper are practical required showing you can fit a CRF in a reasonable time (while you have lunch, say) and without extreme cost (don't want to write a grant application just for hardware to run it).

Enter Amazon Web Services …

As a proof of concept, we encrypted the Wisonsin breast cancer prognosis data set locally, resulting in 13.8GB of encrypted data.

The AWS run

- The spot pricing script earlier was used to determine the cheapest region spot prices for the c3.8xlarge instances
    - 32 cores (Intel Xeon E5-2680 v2), 60GB memory.

The AWS run

- The spot pricing script earlier was used to determine the cheapest region spot prices for the c3.8xlarge instances
    - 32 cores (Intel Xeon E5-2680 v2), 60GB memory.
- The 13.8GB of encrypted data was uploaded to an S3 bucket in Dublin, Ireland and from there copied to São Paulo, Brazil. It was split into 18 ‘shards’ before upload.

The AWS run

- The spot pricing script earlier was used to determine the cheapest region spot prices for the c3.8xlarge instances
  - 32 cores (Intel Xeon E5-2680 v2), 60GB memory.
- The 13.8GB of encrypted data was uploaded to an S3 bucket in Dublin, Ireland and from there copied to São Paulo, Brazil. It was split into 18 ‘shards’ before upload.
- 18 spot requests in each region $\implies 1,152$ CPU cores.

The AWS run

- The spot pricing script earlier was used to determine the cheapest region spot prices for the c3.8xlarge instances
  - 32 cores (Intel Xeon E5-2680 v2), 60GB memory.
- The 13.8GB of encrypted data was uploaded to an S3 bucket in Dublin, Ireland and from there copied to São Paulo, Brazil. It was split into 18 'shards' before upload.
- 18 spot requests in each region $\implies 1,152$ CPU cores.
- Each (slighly modified) RStudio AMI instance fitted 50 trees to 1 of the shards.

The AWS run

- The spot pricing script earlier was used to determine the cheapest region spot prices for the c3.8xlarge instances
    - 32 cores (Intel Xeon E5-2680 v2), 60GB memory.
- The 13.8GB of encrypted data was uploaded to an S3 bucket in Dublin, Ireland and from there copied to São Paulo, Brazil. It was split into 18 'shards' before upload.
- 18 spot requests in each region $\implies 1,152$ CPU cores.
- Each (slighly modified) RStudio AMI instance fitted 50 trees to 1 of the shards.
- Upon completion, encrypted subtree uploaded to S3 again.

The AWS run

- The spot pricing script earlier was used to determine the cheapest region spot prices for the c3.8xlarge instances
  - 32 cores (Intel Xeon E5-2680 v2), 60GB memory.
- The 13.8GB of encrypted data was uploaded to an S3 bucket in Dublin, Ireland and from there copied to São Paulo, Brazil. It was split into 18 ‘shards’ before upload.
- 18 spot requests in each region $\implies 1,152$ CPU cores.
- Each (slighly modified) RStudio AMI instance fitted 50 trees to 1 of the shards.
- Upon completion, encrypted subtree uploaded to S3 again.
- Total cost: $23.86 ($\approx$ €21)

Conclusion

- The opportunity cost to maintain cryptographic security is 2 hours of time and $23.86.
- To buy 36 servers with 32 cores of Intel Xeon Ivy Bridge and 60GB RAM would be prohibitive for the occasional encrypted fit.
- The RStudio AMI reduced the time to working substantially.

Future plans

Future plans

- more built in cluster tools
    - avoid need to install tools on nodes
    - helper functions to install supporting packages
    - helper function to launch PSOCK clusters
    - auto deployment of H2O cluster
- Migration tools to upgrade when new AMIs released
- Tools to add multiple user accounts and manage server resources
- *Your wish here!*

Feedback

Please give feedback – I would love ideas for what else would make the AMIs more useful!

There is no paper currently, so please use a standard software citation if you use them:

Aslett, L. J. M. (2012), *RStudio AMIs for Amazon EC2 cloud computing*. AMI ID ami-ae05a1d9. **URL:** http://www.louisaslett.com/RStudio_AMI/

and simply replace the AMI ID with the version used for reproducibility.

Thanks