Supplementary Material

Econometrics at Scale: Spark Up Big Data in Economics

1 Introduction

The supplementary material contains all codes and setup details necessary to replicate the results of the paper "Econometrics at Scale: Spark up Big Data in Economics". Moreover, we hope it is a useful guide for researchers who are not familiar so far with distributed computing solutions. Our set up is built on Amazon Web Services (AWS) and while it would easily translate to another provider (such as Cloudera, Microsoft Azure, Google Cloud Platform, etc ...), the instruction below reference solely to the AWS platform. There is a massive amount of documentation for AWS available online and many user written tutorials describe different applications. In what follows we condensed this information into the minimal steps necessary to get your own data handling and analysis pipeline running on the cloud framework. We benefited greatly both from Amazon's official documentation as well as various resources from third parties. Some of those explained certain steps better than we ever could, so we merely restate them here such that the readers of our paper have all information in one place. Such cases are clearly marked at the beginning of each section.

A word of caution

Please be aware you will be billed by AWS for running computing instances and trying to replicate our setup will require running AWS computing instances. Also, uploading data to AWS may results in cyber-security risks. Make sure that you have the relevant permissions for your data and jurisdiction. This guide comes with absolutely no warranty. You may find some instructions helpful but you use it at your own risk!

The appendix is structured as follows. Section 2 describes how to run spark on your local machine. More specifically, section 2.1 describes `pyspark` and section 2.2 describes `sparklyr`. The next subsection introduces Amazon Web Servies (AWS) and presents how to initialize a minimal setup to reproduce the results of this paper. The last subsection 4 demonstrates how to deploy `pyspark` and `sparklyr` in section 4.1 and 4.2 respectively.

2 Running Spark on your local machine

In this section, we show how to run Spark in local mode from a Jupyter notebook using `pyspark` and from RStudio using `sparklyr`.

2.1 PySpark on Jupyter Notebook

The first step in the installation process is to download the latest Spark release from the official website and to decompress the folder into the home directory of your machine and (of course) have a Python distribution installed. As a prerequisite, make sure to install the latest version of the Java Development Kid 8. Afterwards you need to add the home directory of your local Spark installation to the path variable in your system's environment:

- On **MacOS** you need to edit the `.bash_profile` file, which is stored in your home directory, and add the following statement (You can edit the file using a standard text editor such as for example *nano*. Open the command line interface and after changing into your home directory, type *nano .bash_profile* to edit the file (changes can be saved via *ctrl + x*). Subsequently, the changes need to be activated via command *source .bash_profile*):

```
export PATH=$PATH:/Users/home/spark-2.4.0-bin-hadoop2.7
export SPARK_PATH=/Users/home/spark-2.4.0-bin-hadoop2.7
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
export PYSPARK_PYTHON=python3
alias spark_notebook='source activate your-env; $SPARK_PATH/bin/pyspark --master local[1]'
```

  To start a Spark session, simply execute the following lines in your Jupyter notebook:

```
## in jupyter type:
from pyspark.sql import SparkSession
spark_session = SparkSession.builder.appName('econometrics_at_scale').getOrCreate()
```

- On **Windows** the process is slightly more involved. Download and decompress Spark to a local directory (avoid blanks in the path), e.g. to `C:\opt\spark`. Next, download the *winutils.exe* into the bin folder of your Spark directory (e.g. `C:\opt \spark\spark-2.4.0-bin-hadoop2.7\bin`) and add the relevant environmental variables by running the following commands from the command line interface (You can open the command line interface by pressing *ctrl + R* and type `cmd`, which will open the console):

```
setx SPARK_HOME C:\opt\spark\spark-2.4.0-bin-hadoop2.7
setx HADOOP_HOME C:\opt\spark\spark-2.4.0-bin-hadoop2.7
setx PATH "%PATH%;C:\opt\spark\spark-2.4.0-bin-hadoop2.7\bin"
```

  Close the terminal window and reboot your computer. Restart the terminal and enter `pyspark` which will open a Jupyter notebook. In the Juypter nootebook enter the following codes to create a spark context and test whether you are on the right version:

```
## in Juypter type:
sc = SparkContext.getOrCreate()
sc.version
>> '2.4.0'
```

2.2 Sparklyr on RStudio

If you are using *R*, chances are you are using the RStudio editor, which is free of charge for personal and academic use at the time of writing this paper. The developers of RStudio provide a magnificent introduction to `sparklyr`, which can be found here. In what follows we borrow greatly from those resources, yet focus a bit more on the "social scientist" perspective of data handling and analysis.

To install `sparklyr` simply run the following commands in your R console:

```r
install.packages("sparklyr")
library(sparklyr)
spark_install(version = "2.1.0")
```

Once installed, load the library and create a connection:

```r
library(sparklyr)

# define spark configuration
conf <- spark_config()
   conf$`sparklyr.cores.local` <- 2  # number of CPU cores spark can use
   conf$`sparklyr.shell.driver-memory` <- "8G" # memory size of shell driver
   conf$spark.memory.fraction <- 0.6  # fraction of total computer's memory available to spark

# establish a spark connection
sc <- spark_connect(master = "local",
                    config = conf )
```

The official documentation continues by exploring the well-known flights dataset and we suggest you follow it along. Here however, we would like to continue with an example which might be a bit closer to what economists are used to. Suppose you have a very large dataset and would like to develop your Spark application in local mode (to avoid paying cloud services fees) but the dataset is too large to be read into memory. Since your local spark instance is limited by the physical memory of your computer, we will read in only a fraction of the entire dataset. As an example, we chose the popular HMDA data provided by the *Federal Financial Institutions Examination Council* (FFIEC), which contains loan application data for the US. To develop locally we created a subsample containing the first 10,000 observations from the yearly HMDA datasets. Since it is only a small subsample you can fit it into your computers memory and develop your Spark application locally:

```r
# import small subset to R
df = read.csv("HMDA_subsample.csv")

# copy the R dataframe df to spark using copy_to()
df_spark = copy_to(sc, df)

# or load it directly to spark
hmda_spark= spark_read_csv(sc, name = "hmda_spark", header= TRUE, delimiter = ",",
               path= "HMDA_subsample.csv")
```

Now, let's compute the mean loan amount by year in R using `dplyr` and in Spark usings `sparklyr`:

```r
meanbyyear = hmda %>%
 group_by(as_of_year) %>%
 summarise(mean_loan_amounts_000s =
             mean(loan_amount_000s))

meanbyyear
# A tibble: 10 x 2
   as_of_year mean_loan_amounts_000s
        <int>                  <dbl>
 1       2007                   176.
 2       2008                   201.
 3       2009                   201.
 4       2010                   219.
 5       2011                   224.
 6       2012                   225.
 7       2013                   230.
 8       2014                   242.
 9       2015                   246.
10       2016                   259.
```

```r
meanbyyear = hmda_spark %>%
 group_by(as_of_year) %>%
 summarise(mean_loan_amounts_000s =
             mean(loan_amount_000s)) %>%
 collect()
meanbyyear
# A tibble: 10 x 2
   as_of_year mean_loan_amounts_000s
        <int>                  <dbl>
1        2008                   201.
2        2009                   201.
3        2011                   224.
4        2014                   242.
5        2015                   246.
6        2010                   219.
7        2016                   259.
8        2007                   176.
9        2012                   225.
10       2013                   230.
```

A great feature of `sparklyr` is that you can use `dplyr` syntax for your Spark application. Note how both approaches yield the same result. Also, note that the Spark results are *not* ordered. This is because the `group_by()` command in `sparklyr` distributes the data across executors and collects them back after computing the mean, which does not necessarily preserve the order. This example illustrates the basic data handling pipeline in Spark using `sparklyr`:

1. Establish a Spark connection
2. Load the data into Spark
3. Using `dplyr` syntax, manipulate the Spark dataframe
4. Load the results back to the R environment using `collect()`

Finally, we can disconnect the local Spark interface:

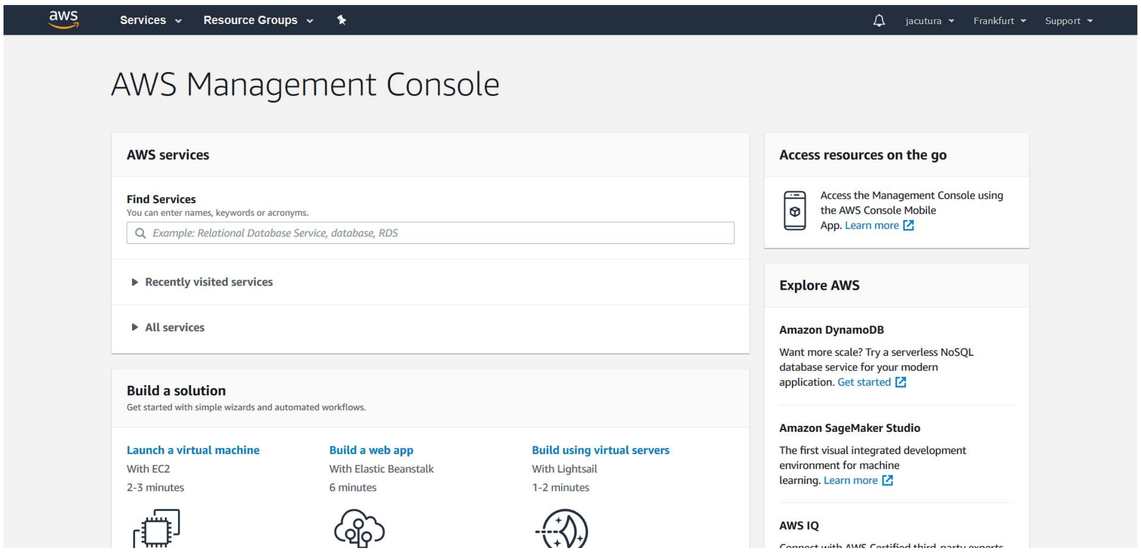
```
spark_disconnect(spark)
```

3 Setting up a cloud computing environment (on AWS)

3.1 Setting up AWS and upload data

In this section we walk through the process of creating an Amazon Web Services (AWS) account. Before we get into the details, note that if you are running computing instances on AWS you will be billed and a credit card is necessary for the setup. Also, note that uploading data to AWS and using computing instances may result in data vulnerabilities. This manual comes with no warranty whatsoever.

You can create your own account at https://aws.amazon.com/. Upon login you will see the AWS Management Console:



AWS Management Console

AWS services

Find Services
You can enter names, keywords or acronyms.
Example: Relational Database Service, database, RDS

Recently visited services

All services

Build a solution
Get started with simple wizards and automated workflows.

Launch a virtual machine
With EC2
2-3 minutes

Build a web app
With Elastic Beanstalk
6 minutes

Build using virtual servers
With Lightsail
1-2 minutes

Access resources on the go

Access the Management Console using the AWS Console Mobile App. Learn more

Explore AWS

Amazon DynamoDB
Want more scale? Try a serverless NoSQL database service for your modern application. Get started

Amazon SageMaker Studio
The first visual integrated development environment for machine learning. Learn more

AWS IQ

As a first step, we will upload data to AWS storage (called "S3"). To this end click *Services* and in the *Storage* section choose *S3*. First, we create a bucket for the project. To do so click *+ Create Bucket*. Choose any (DNS compliant) name.

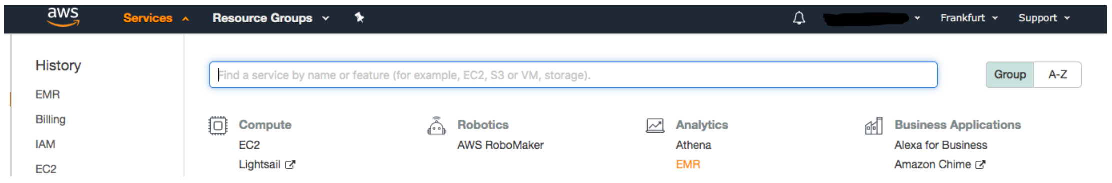
We named our bucket "econometricsatscalebucket" (Note that you cannot use the same name, since all buckets have to be unique on S3.). Under *Configure options* and *Set permissions* you may leave the standard settings.

In the bucket, we create several folders, where we store research data, bootstrap scripts and outputs. Specifically we create 3 folders: "data", "scripts", "output". In order to replicate our analysis in section ?? and ?? upload the HMDA data to your S3 in a subfolder called *data/micro*. For the panel econometrics exercise in section ??, upload our simulated data to a subfolder *data/panel*. For the time series exercise in section ??, upload our simulated time series data to a subfolder called *data/time_series*. Additionally, we need to create subfolders to store outputs from the exercise, namely parquet files (*output/time_series/parquet*), forecasts (*output/time_series/forecasts*) and fitted models (*output/time_series/models*). Also make sure to upload the python modul *fit_model_and_forecast.py* for computing forecasts and saving fitted models to a subfolder *scripts/time_series*. Instructions regarding the necessary bootstrap scripts for sparklyr will be provided at the respective subsections below.

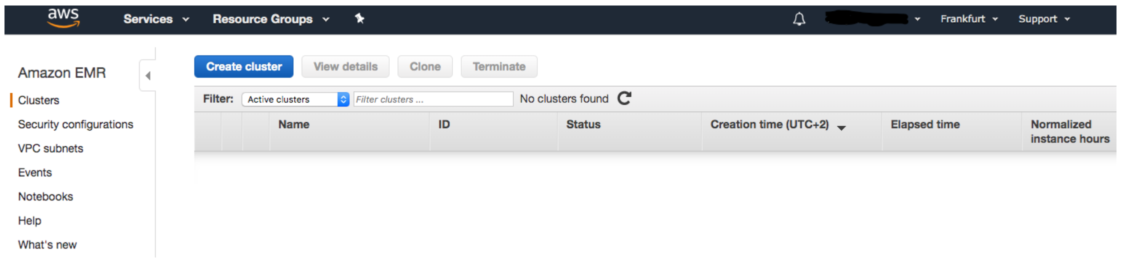
3.2 Creating an *EMR* cluster and installing custom software

In this section we walk through the process of creating a Spark cluster using AWS *EMR* service. As emphazised earlier there are many other cloud vendors which provide similar services, so this section only contains one out of many other existing solutions that may be equally well or possibly even more suited for your application.

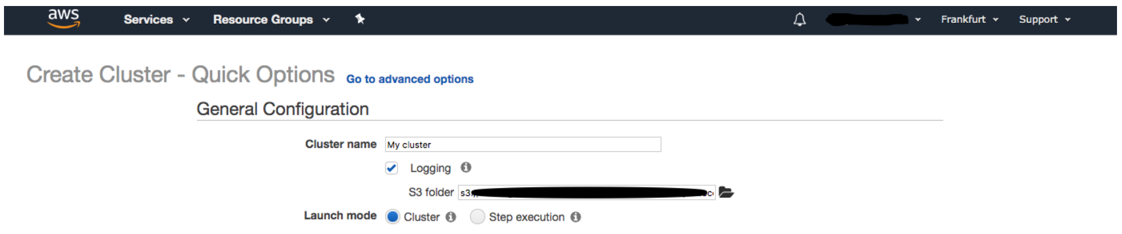
The firs step in creating a cluster is to go to the AWS *Services* menu and select *EMR*:



The *EMR* dashboard will open and provide you with the option to create a cluster as shown below:



By clicking on the *"Create cluster'* button we enter the cluster configuration dashboard, noting that we have actually not yet created a cluster so we do not have to worry about any service charges at this stage. Next we need to configure our cluster to install Spark (at the time of writing this paper the latest version available in *EMR* is 2.4.0) and also Livy which is required for running a Jupyter notebook on the cluster. In order to configure the cluster software, we click on the *"Go to advanced options"* field:



Create Cluster - Quick Options

General Configuration

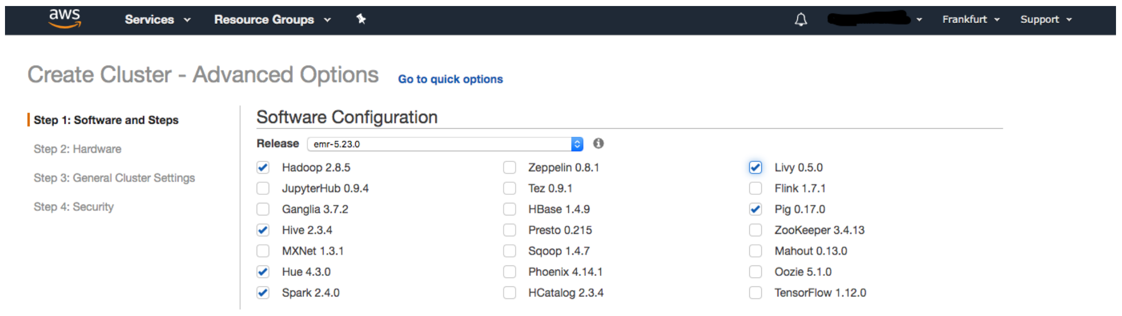
Cluster name: My cluster

✓ Logging

S3 folder: s3[redacted]

Launch mode: ● Cluster ○ Step execution

In *Release* choose the EMR version you want to run. To replicate the finding of this paper, choose "emr-5.23.0". In the *Software Configuration* section, check the Spark and Livy boxes and leave all other configurations at their default values:



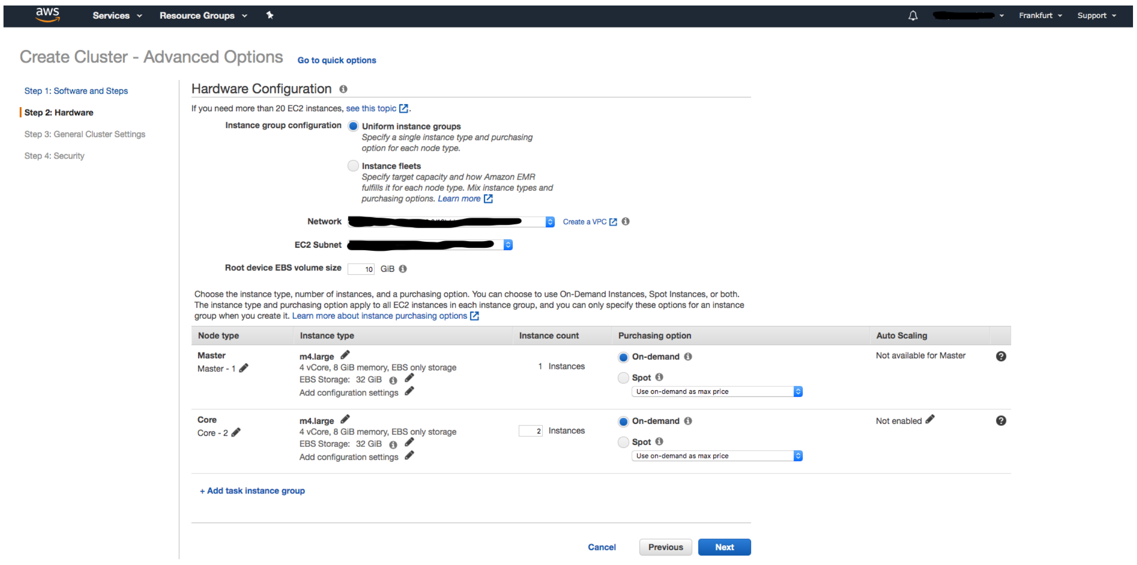
Create Cluster - Advanced Options

Software Configuration

Release: emr-5.23.0

✓ Hadoop 2.8.5	☐ Zeppelin 0.8.1	✓ Livy 0.5.0
☐ JupyterHub 0.9.4	☐ Tez 0.9.1	☐ Flink 1.7.1
☐ Ganglia 3.7.2	☐ HBase 1.4.9	✓ Pig 0.17.0
✓ Hive 2.3.4	☐ Presto 0.215	☐ ZooKeeper 3.4.13
☐ MXNet 1.3.1	☐ Sqoop 1.4.7	☐ Mahout 0.13.0
✓ Hue 4.3.0	☐ Phoenix 4.14.1	☐ Oozie 5.1.0
✓ Spark 2.4.0	☐ HCatalog 2.3.4	☐ TensorFlow 1.12.0

We proceed to configure the hardware settings of the cluster including the instance type and the number of instances. The hardware configuration strongly depends on the resource requirements (and budget considerations) of the specific application. We recommend to start with a small cluster with limited resources to familiarize yourself with the process of deploying and running your locally developed Jupyter notebook or RStudio script. For example, running one master and two core instances of type *m4.large* will give you enough flexibility to deploy and test your application on a small scale, while it will barely cost you more than a few dollars over a couple of hours:



Create Cluster - Advanced Options | Go to quick options

- Step 1: Software and Steps
- **Step 2: Hardware**
- Step 3: General Cluster Settings
- Step 4: Security

Hardware Configuration

If you need more than 20 EC2 instances, see this topic.

Instance group configuration [✓ **Uniform instance groups**] *Specify a single instance type and purchasing option for each node type.*
Instance fleets *Specify target capacity and how Amazon EMR fulfills it for each node type. Mix instance types and purchasing options. Learn more*

Network [redacted] Create a VPC

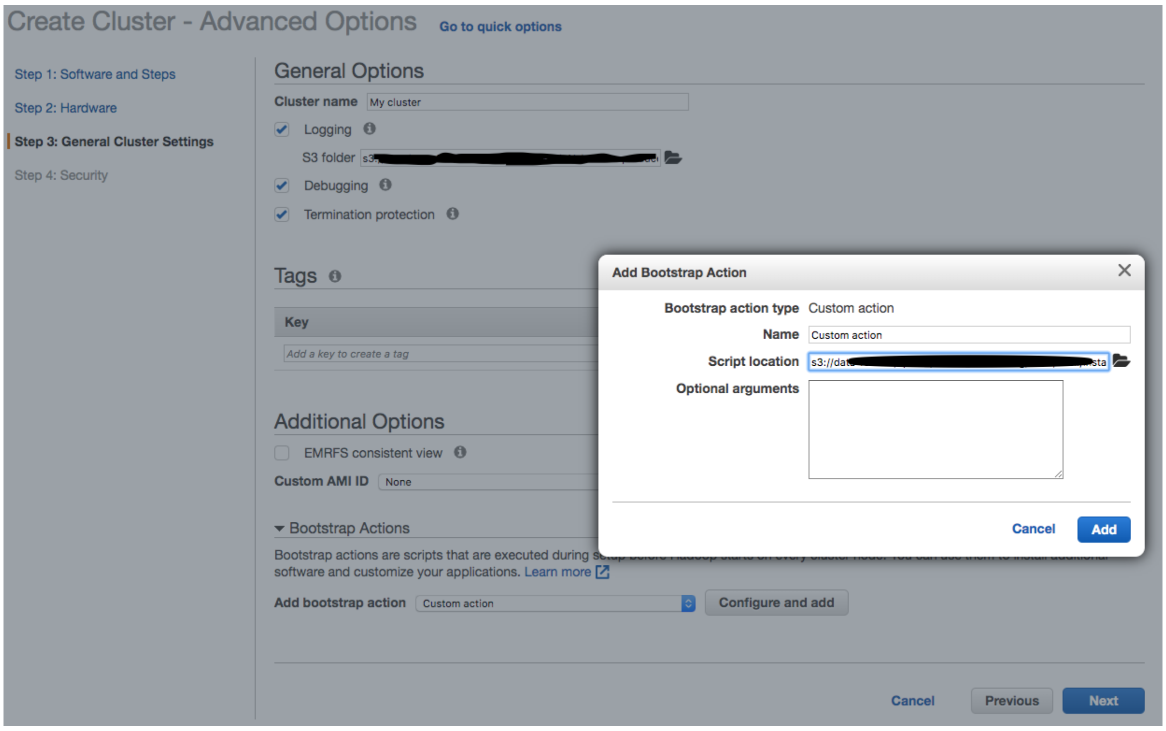
EC2 Subnet [redacted]

Root device EBS volume size 10 GiB

Choose the instance type, number of instances, and a purchasing option. You can choose to use On-Demand Instances, Spot Instances, or both. The instance type and purchasing option apply to all EC2 instances in each instance group, and you can only specify these options for an instance group when you create it. Learn more about instance purchasing options

Node type	Instance type	Instance count	Purchasing option	Auto Scaling
Master Master - 1	**m4.large** 4 vCore, 8 GiB memory, EBS only storage EBS Storage: 32 GiB Add configuration settings	1 Instances	[✓ **On-demand**] **Spot** Use on-demand as max price	Not available for Master
Core Core - 2	**m4.large** 4 vCore, 8 GiB memory, EBS only storage EBS Storage: 32 GiB Add configuration settings	2 Instances	[✓ **On-demand**] **Spot** Use on-demand as max price	Not enabled

Once you have selected your preferred hardware configuration, go to the next section *General Cluster Settings*:



Create Cluster - Advanced Options | Go to quick options

- Step 1: Software and Steps
- Step 2: Hardware
- **Step 3: General Cluster Settings**
- Step 4: Security

General Options

Cluster name My cluster
[✓] Logging
S3 folder [redacted]
[✓] Debugging
[✓] Termination protection

Tags

Key
Add a key to create a tag

Additional Options

[ ] EMRFS consistent view
Custom AMI ID None

Bootstrap Actions

Bootstrap actions are scripts that are executed during s… software and customize your applications. Learn more

Add bootstrap action Custom action | Configure and add

Add Bootstrap Action

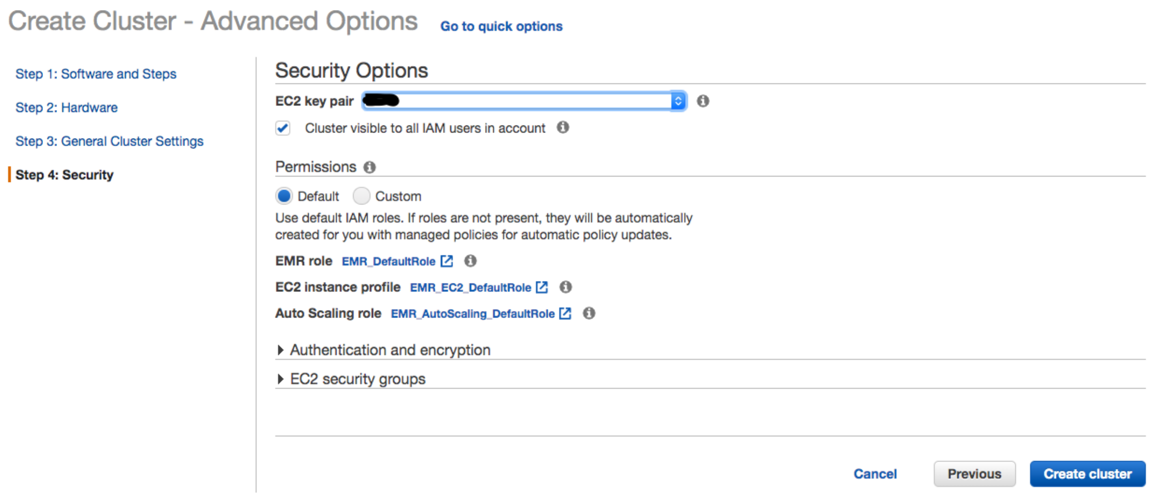
Bootstrap action type Custom action
Name Custom action
Script location [redacted]
Optional arguments

At the bottom of the page there is a *Bootstrap Actions* field where custom actions can be specified to install additional software or to customize the configuration of cluster instances. In essence, bootstrap actions are scripts that run on all nodes after the cluster is launched. For this purpose, a shell script specifying your custom installations must be uploaded to a folder in the S3 bucket. For example, if you want to install `scikit-learn` on all cluster nodes you would upload a shell script with the following content:

```
#!/bin/bash -xe
sudo pip install -U sklearn
```

Add the bootstrap action by selecting *"Custom action"* and, under the *"Configure and add"* button, browse the S3 path to the shell script (no optional arguments needed) and click *"Add"*.

After adding the custom bootstrap action, move on to the last step *Security* where you have to add your previously created EC2 key pair. Finally, you can create the cluster via the *"Create cluster"* button:



Create Cluster - Advanced Options
Go to quick options

Step 1: Software and Steps
Step 2: Hardware
Step 3: General Cluster Settings
Step 4: Security

Security Options

EC2 key pair [redacted]

✓ Cluster visible to all IAM users in account

Permissions

● Default ○ Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role EMR_DefaultRole

EC2 instance profile EMR_EC2_DefaultRole

Auto Scaling role EMR_AutoScaling_DefaultRole

▸ Authentication and encryption

▸ EC2 security groups

4 Running Spark on a cluster

4.1 Running PySpark

This section describes the procedure of creating a Jupyter notebook for running PySpark jobs on an AWS EMR cluster. Note that the notebook will automatically be saved to your S3 bucket so after terminating the cluster you can still use the notebook when you start another cluster at a later stage.

To create a notebook, go to the EMR dashboard, select *Notebooks* and choose *Create notebook*. In the notebook configurations window, you are asked to provide a notebook name and under the *Cluster** option you can either choose to create a new cluster (via *Create cluster*) or alternatively you can attach to an existing running cluster. As mentioned above, you may want to start with a small cluster to familiarize yourself with the notebook workflow on EMR. For example, if you specify three instances, one instance is devoted to the master node and two instances are devoted to the worker nodes. You can also select an S3 location to store your notebook.[1] Finally, you can create the notebook by choosing *Create notebook*. A new view will open showing the configuration details of the notebook. Once the cluster is ready the notebook can be accessed via the *Open* button.

In the notebook, a Spark session is automatically started which can be verified by following the example below:

[1]If you leave the default value a directory will automatically be created in your bucket.

```
data = [1, 2, 3, 4, 5]
sc = spark.sparkContext
dist_data = sc.parallelize(data)
dist_data.collect()
```

4.2 Running sparklyr

In this section we show how to deploy sparklyr on AWS EMR. The general steps are:

1. Configure and upload a bootstrap script
2. launch a cluster using the appropriate bootstrap script
3. Connect remotely to your RStudio on AWS

Configuring and uploading a bootstrap script

For the first step, we suggest to upload a bootstrap script which will install RStudio Server along with sparklyr. One such script is provided by Cosmin Catalin on his GitHub account. For reference, we forked it to our repository and you can download the `install-rstudio-server.sh` bootstrap script here. You can open it using any editor and (if you wish) edit the following parameters:

```
USER="drwho"
PASS="tardis"
SPARK="2.1.1"
```

This allows to change the username and password for the web-login to the AWS RStudio Interface and the spark version running on it. Finally, upload the `install-rstudio-server.sh` to your `S3` storage to any folder (for example into `econometricsatscale\bootstrap\sparklyr`.

Launch a cluster

When launching a cluster as described in 3.2, choose the `install-rstudio-server.sh` as bootstrap script (and leave the additional parameters field blank). Create the cluster. While the cluster is launching you may edit the security group of your master node and enable TCP Port 8787 from Anywhere on your master node in order to allow a remote connect to RStudio Server. To do so, click Security ⇒ Create a security group. As a security group name you can choose anything, similarly for the Description. Upon creation, click "Inbound Rules" in the ribbon, and click "Edit rules" and then "Add Rule". For Type choose "Custom TCP Rule", and set the port range to 8787. As source, choose "Anywhere". Finish by clicking "Save rules".

Connect remotely (from a Windows machine)

As a first step, you will need to install Putty[2], an SSH client, which lets you run sparklyr on AWS from the comfort of your local machine's RStudio interface. Amazons keypair format is not supported by putty, which is why you need to convert it

[2]You can find the latest version here: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

first. To do so, execute `puttygen`, click File ⇒ Conversions ⇒ Import Key and choose the AWS keypair (*".pem" file*) which you downloaded when creating it. Finish by choosing *Save Key*. Now, you can configure PuTTy. How to do so is documented on AWS and merely restated here for reference:

1. Click Category List ⇒ Session and in the Host name field, type "`hadoop@MasterPublicDNS`", where "`MasterPublicDNS`" is your master's node address, for example: "`hadoopec2-###-##-##-###.compute-1.amazonaws.com`"
2. Click Category List ⇒ Session > SSH ⇒ Auth ⇒ Browse and select the `.pkk` file which you generated earlier.
3. Click Category List ⇒ Session > SSH ⇒ Tunnels and in the source port field, type 8157. Leave the Destination field blank and select the `Dynamic` and `Auto` options
4. Choose "Add" and "Open" and choose "Yes" to dismiss the PuTTy security alert.

The AWS console should open. In the AWS consolte type:

```
sudo rstudio-server start
```

and press enter. This will launch RStudio on the AWS master node. Finally, to access RStudio on AWS open a browser and copy and paste:

```
@ec2-###-##-##-###.compute-1.amazonaws.com:8787
```

to your browser. You will be asked to enter the username and password created above and stored in your `install-rstudio-server.sh` file.

Connect remotely (from a Mac)

How to do so is documented on AWS and merely restated here for reference:

1. Open a terminal window. On Max OS X, choose Applications ⇒ Utilities ⇒ Terminal. On other Linux distributions, terminal is typically found at Applications ⇒ Accessories ⇒ Terminal.
2. To establish a connection to the master node, type the following command. Replace "~\mykey.pem" with the location and filename of the private key file (.pem) used to launch the cluster:

   ```
   ssh -i ~/mykey.pem hadoop@ec2-###-##-##-###.compute-1.amazonaws.com
   ```

The AWS console should open. In the AWS console type:

```
sudo rstudio-server start
```

and press enter. This will launch RStudio on the AWS master node. Finally, to access RStudio on AWS open a browser and copy and paste:

```
@ec2-###-##-##-###.compute-1.amazonaws.com:8787
```

to your browser. You will be asked to enter the username and password created above and stored in your `install-rstudio-server.sh` file.