

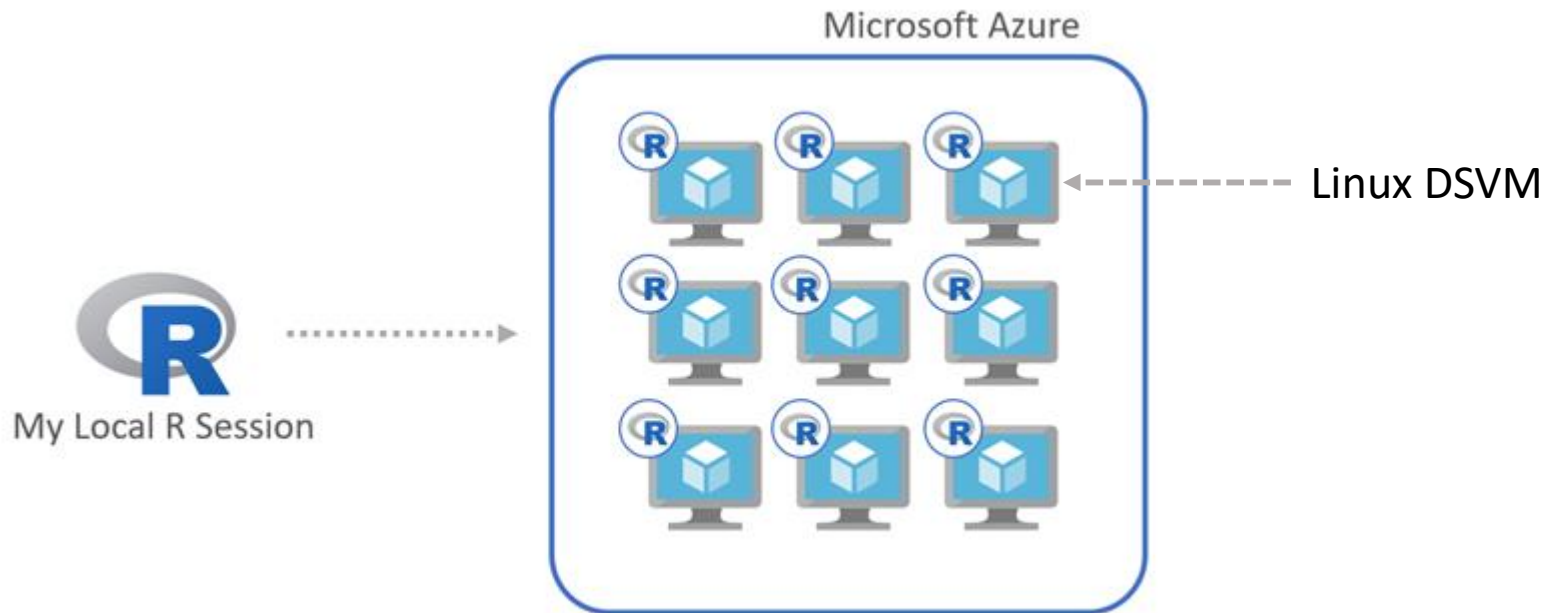


R Parallel Programming in the Azure HPC Batch

Adrian.Fernandez@microsoft.com
@adrianfz10

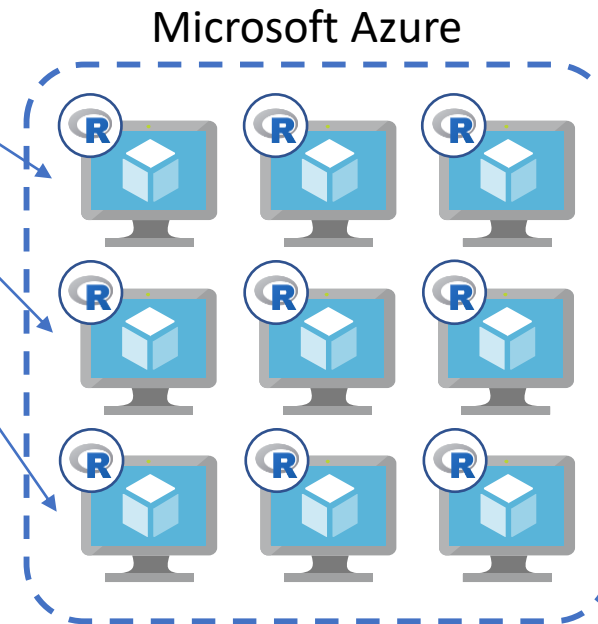
doAzureParallel is... R Package for parallel procs

A simple R package that uses Azure as a parallel-backend for popular open source tools to use –foreach, caret, plyr, etc.



Foreach using doAzureParallel

```
library(doAzureParallel)  
foreach (i = 1:100) %dopar% {  
  myParallelAlgorithm(...)  
}
```



doAzureParallel on Azure Batch

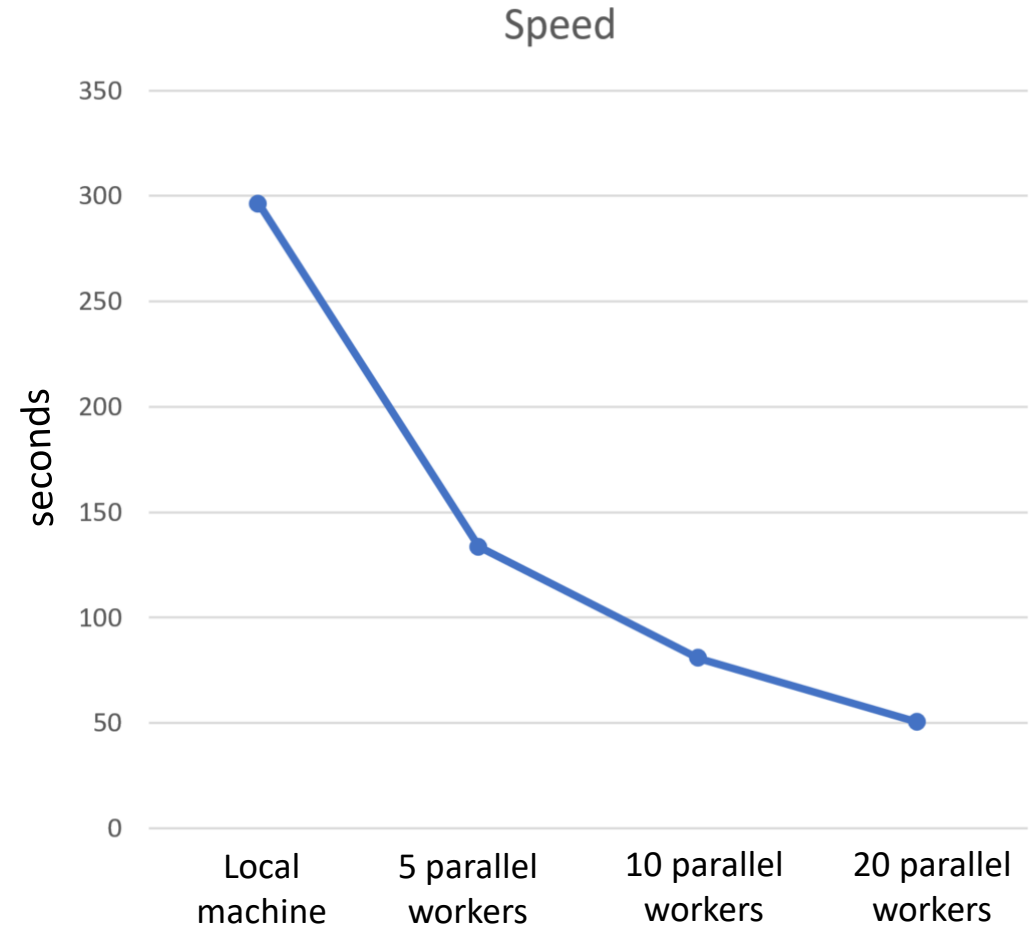
Azure Batch is a platform HPC as a Services that provides easy job scheduling and cluster management, allowing applications or algorithms to run in parallel at scale.

- Capacity on demand; jobs on demand
- Autoscale (more on this later)
- Minimal cluster management (node failure, install, etc)
- Hardware choice – use any VM size
- Pay by the minute
- Cost effective – no charge for using it, you only pay for the VMs
- More cost effective – low priority VMs (more on this later)
- Completely containerized

If you want to run jobs using elastic compute, Batch is a great fit!

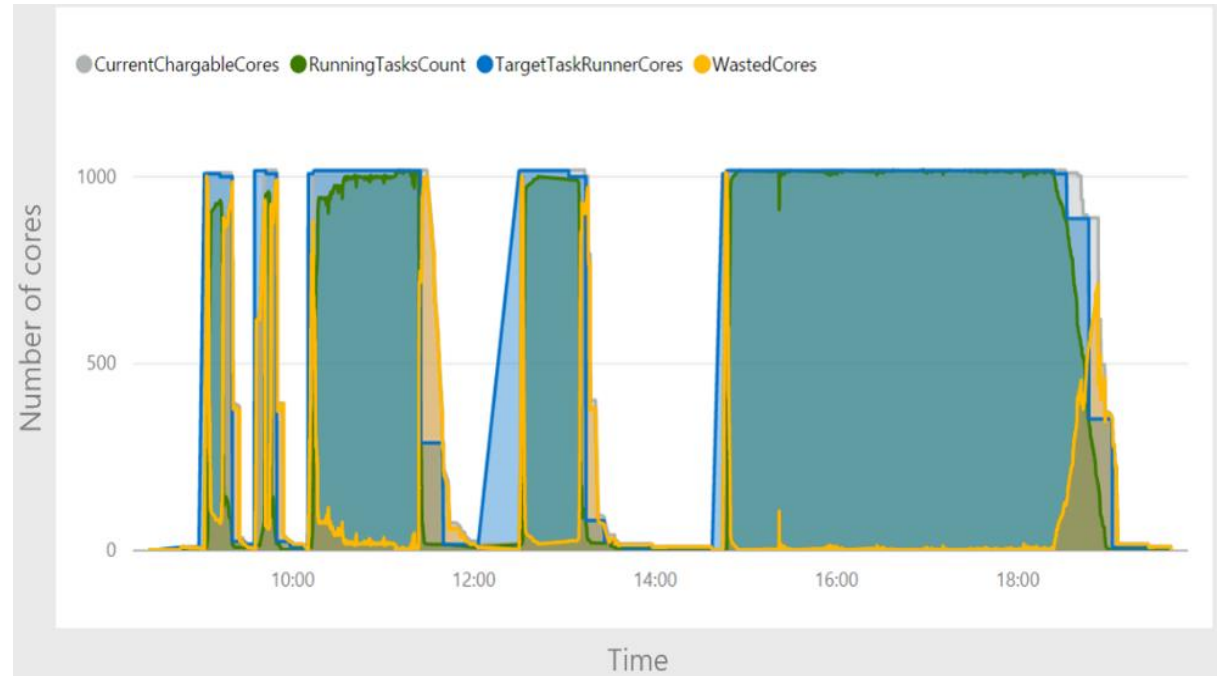
Scale

- From 1 to 10,000 VMs for a cluster
- From 1 to millions of tasks
- Your selection of hardware:
 - General compute VMs (A-Series / D-Series)
 - Memory / storage optimized (G-Series)
 - Compute Optimized (F-Series)
 - GPU enabled (N-Series)
- Results from computing the **Mandelbrot** set when scaling up:



Elastic Compute

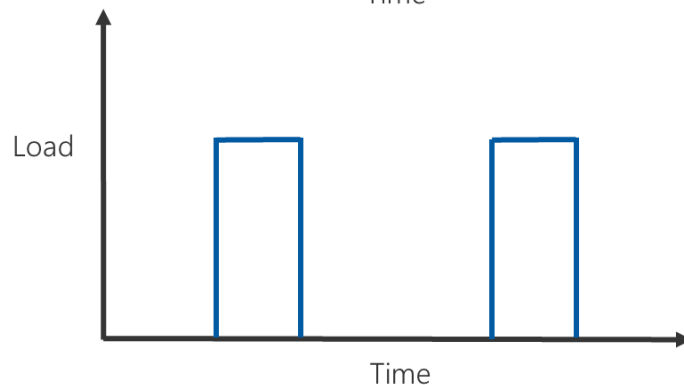
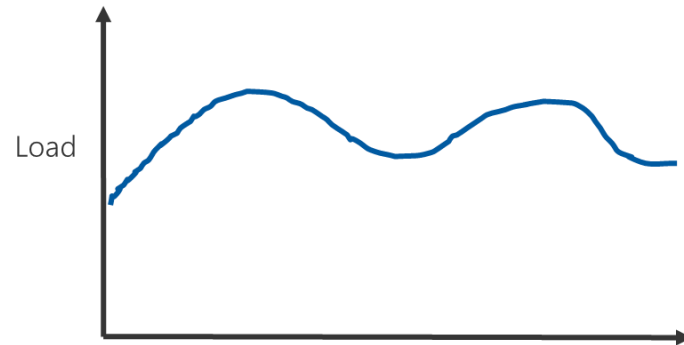
- Compute on-demand
 - Create/delete your cluster as you need
- Autoscaling pool = maximizing cloud elasticity
 - Long running batch jobs / overnight
 - Daily scheduled work – pre-provision cluster so its ready for you at the beginning of the day
 - Bursty work



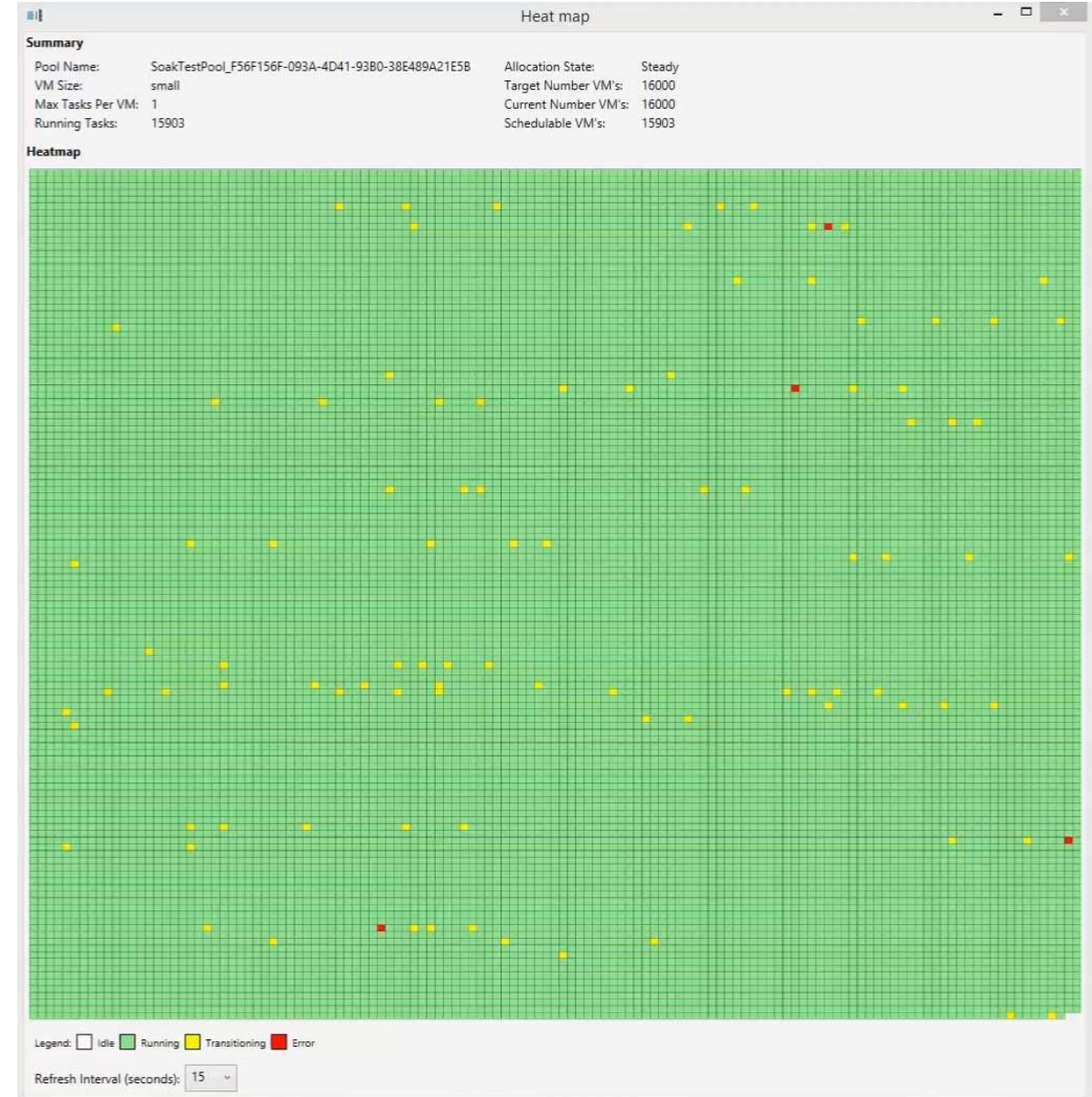
Elasticity & Scale

What would you do with 100,000 cores? –
Big compute at global scale (16,000 cores)

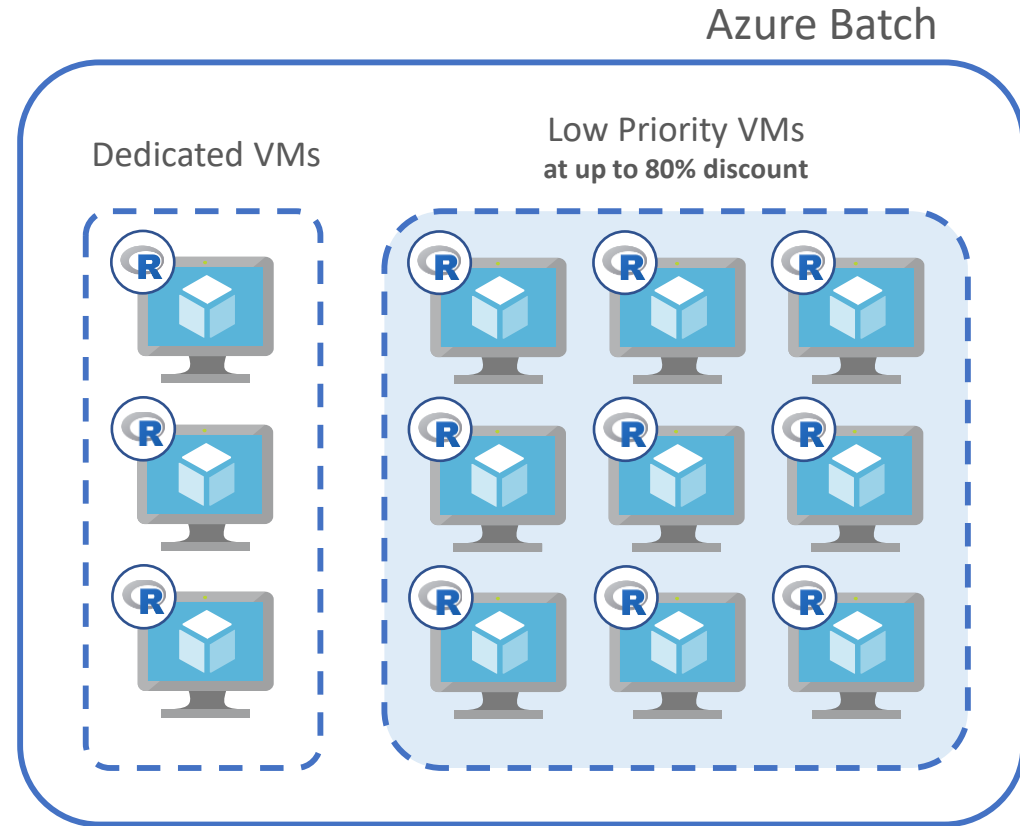
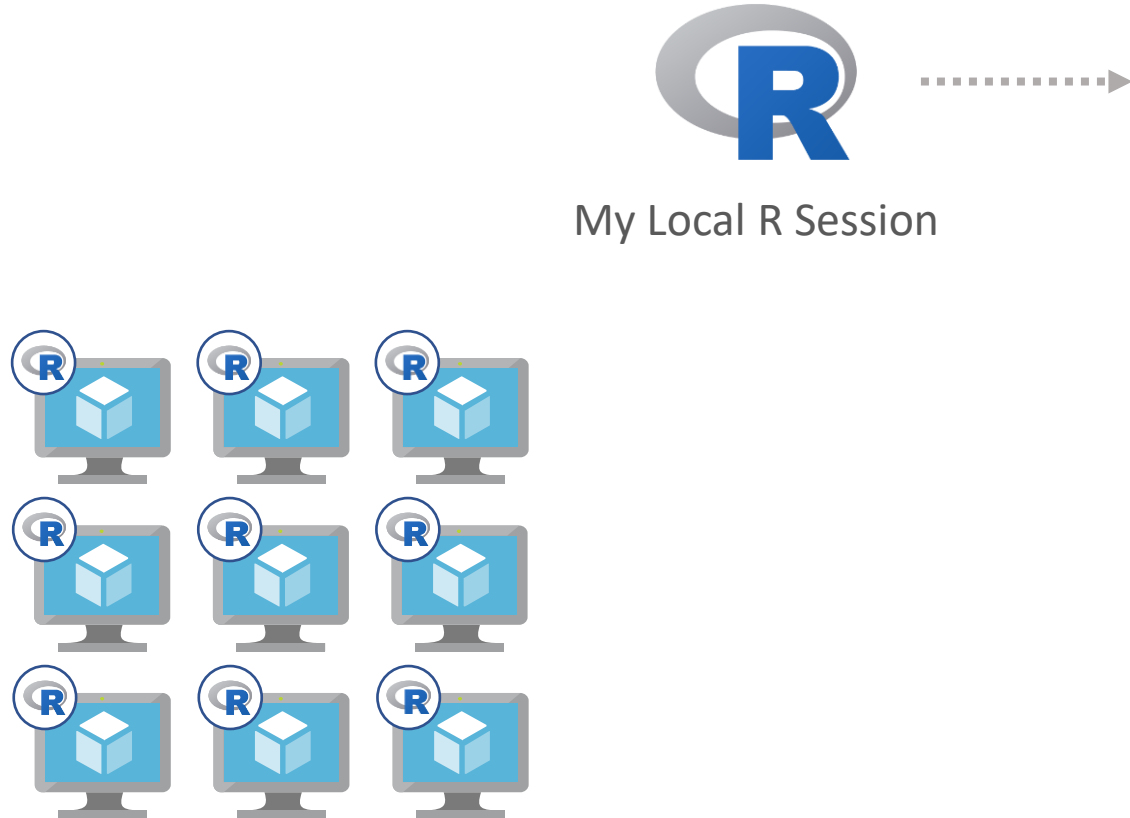
<https://azure.microsoft.com/en-us/blog/what-would-you-do-with-100000-cores-big-compute-at-global-scale/>



<https://blogs.endjin.com/2015/07/spinning-up-16000-a1-virtual-machines-on-azure-batch/>



R + Azure DEMO



Monte Carlo Pricing - HPC Simulation - %dopar%

8-node cluster (standard D2v2: 2 vCPU, 7 Gb)

- specify VM class in `cluster.json`
- specify credentials for Azure Batch and Azure Storage in `credentials.json`

```
# Estimate runtime for 1 million (linear approximation) (1000 x 1000)
1000 * difftime(end_s, start_s, unit = "min")

# Run 1 million simulations with doAzureParallel
# We will run 100 iterations where each iteration executes 10,000 simulations
opt <- list(chunkSize = 20) # optimize runtime. chunking allows us to run multiple iterations

## %dopar% ## AZURE BATCH COMPUTATION
start_p <- Sys.time()
closingPrices_p <- foreach(i = 1:1000, .combine='c', .options.azure = opt) %dopar% {
  replicate(1000, getClosingPrice())
}
end_p <- Sys.time()
```

```
{
  "name": "myAzureBatchPool-HPC",
  "vmSize": "Standard_D2_v2",
  "maxTasksPerNode": 4,
  "poolSize": {
    "dedicatedNodes": {
      "min": 3,
      "max": 5
    },
    "lowPriorityNodes": {
      "min": 5,
      "max": 5
    },
    "autoscaleFormula": "QUEUE"
  },
  "containerImage": "rocker/tidyverse:latest",
  "rPackages": {
    "cran": [],
    "github": []
  }
}
```

45 seconds (more than **5 times faster**) on a warm start

myAzureBatchPool-HPC



Refresh



Add job



Scale



Delete



Overview

General



Properties



Nodes

Settings



Certificates



Start task



Application packages



Scale

Essentials ^

Current cores

16

Dedicated nodes

3

Low-priority nodes

5

Operating System

microsoft-azure-batch ubuntu-server-container 16-04-lts (latest)

VM size

standard_d2_v2

Allocation state

Steady

Summary



Idle
0

Running
8

Creating
0

Starting
0

Rebooting
0

Cross-validation with caret

- Most predictive modeling algorithms have “tuning parameters”
- Example: Boosted Trees
 - Boosting iterations
 - Max Tree Depth
 - Shrinkage
- Parameters affect model performance
- Try ‘em out: **cross-validate**

```
grid <-  
data.frame(  
  nrounds = ...,  
  max_depth = ...,  
  gamma = ...,  
  colsample_bytree = ...,  
  min_child_weight = ...,  
  subsample = ...)  
)
```

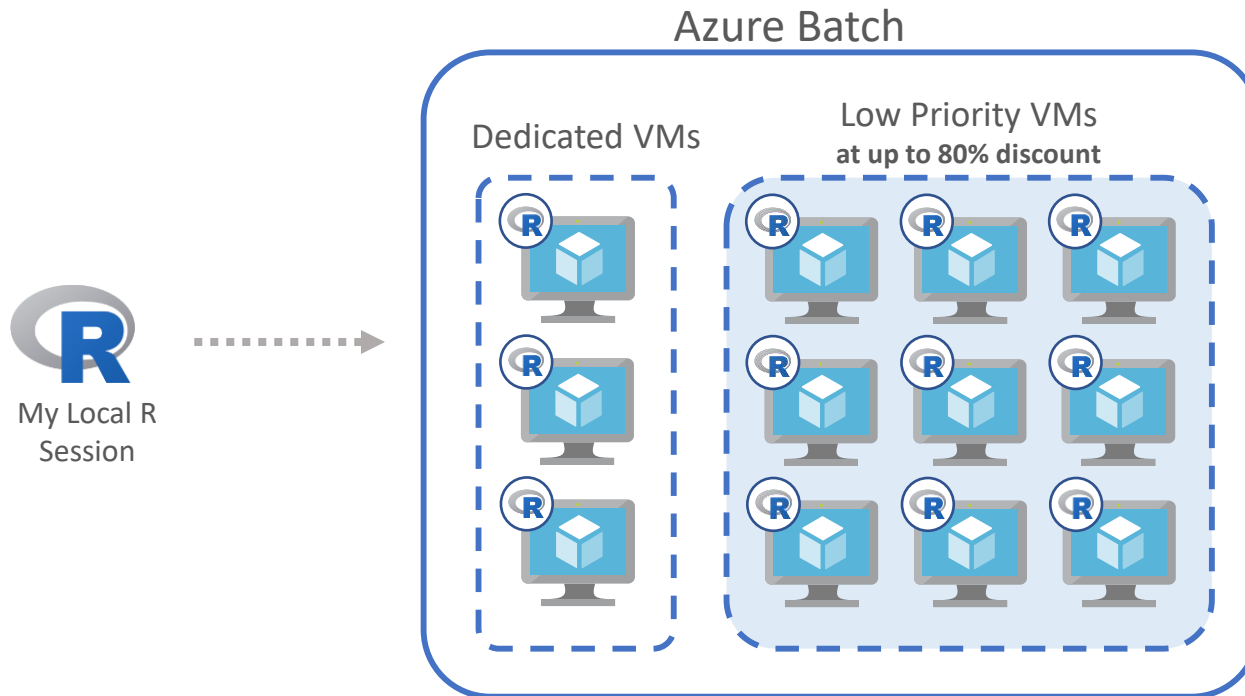
Cross-validation in parallel

- Caret's train function will **automatically** use the registered foreach backend
- Just register your cluster first:
`registerDoAzureParallel(cluster)`
- Handles sending objects, packages to nodes

```
mod <- train(  
  Class ~ .,  
  data = dat,  
  method = "xgbTree",  
  trControl = ctrl,  
  tuneGrid = grid,  
  nthread = 1  
)
```

Low Priority Nodes

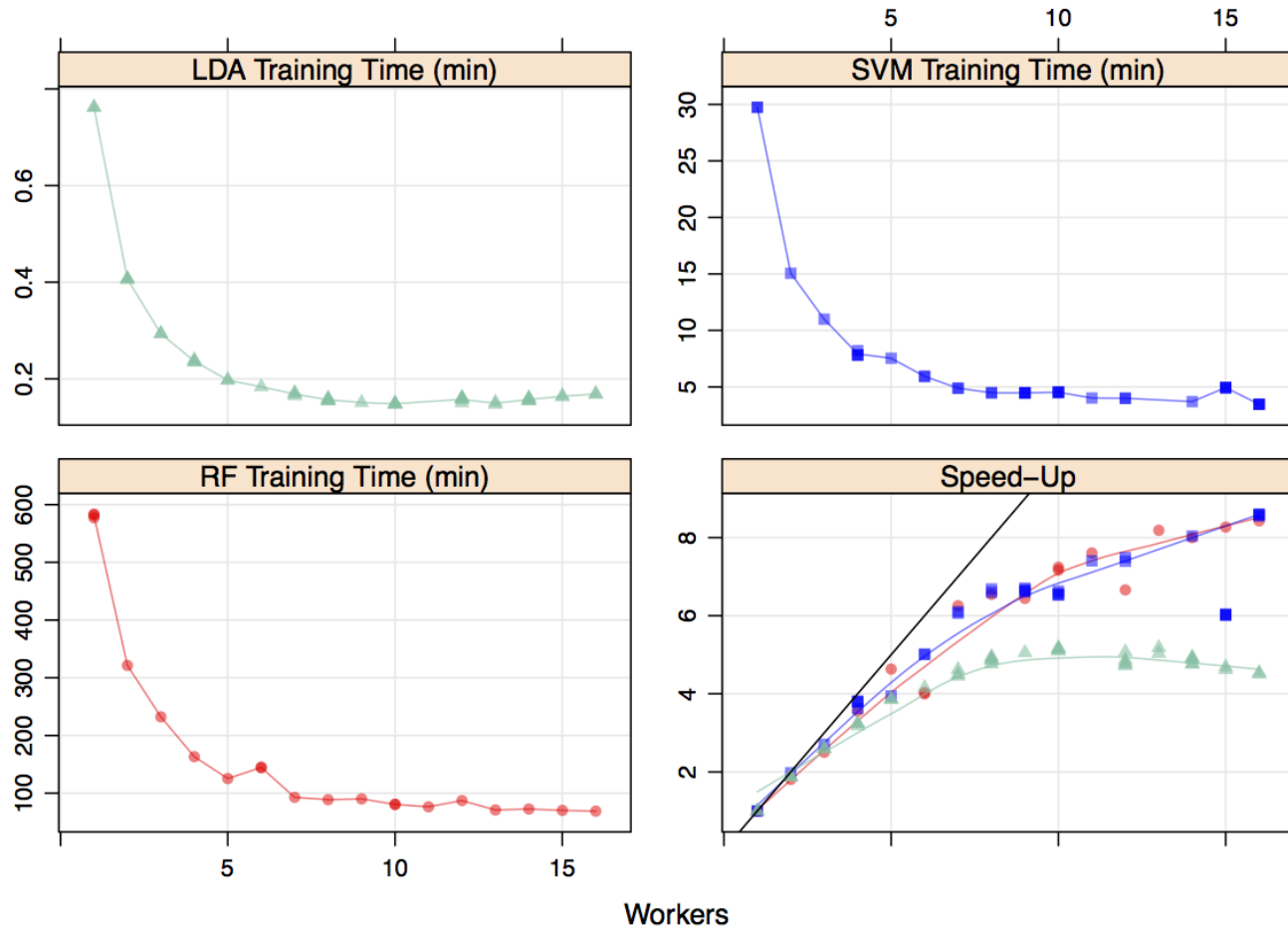
- Low-Priority = (very) Low Costs VMs from surplus capacity
 - up to 80% discount
- Clusters can mix dedicated VMs and low-priority VMs



```
"poolSize": {  
  "dedicatedNodes": {  
    "min": 3,  
    "max": 3  
  },  
  "lowPriorityNodes": {  
    "min": 9,  
    "max": 9  
  },  
}
```

parallel speed up

RF ● SVM ■ LDA ▲



Source: cda.ms/6V

- Max Kuhn benchmarked various hardware and OS for local parallel
 - cda.ms/6V
- Let's see how it works with doAzureParallel

Packages and Containers

- Docker images used to spawn nodes
 - Default: rocker/tidyverse:latest
 - Lots of R packages pre-installed
- But this cross-validation also needs:
 - **xgboost, e1071**
- Easy fix: add to cluster.json

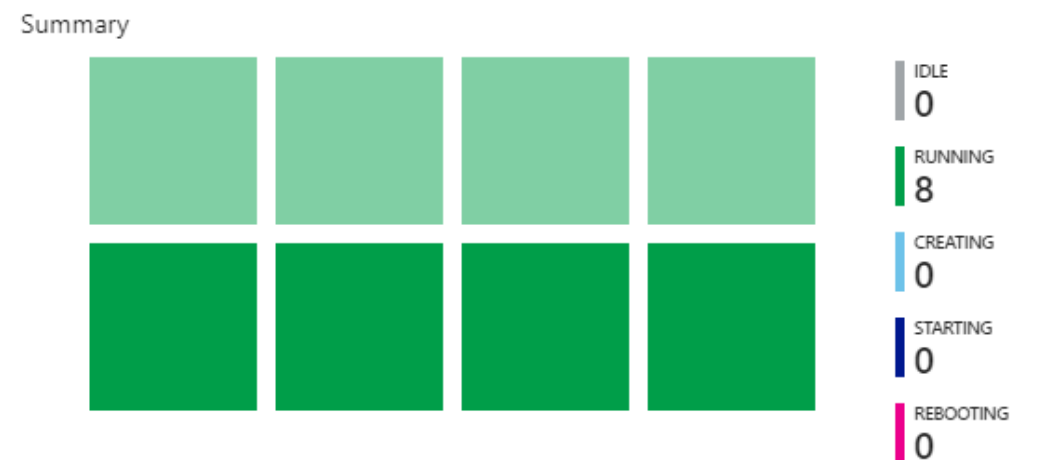
```
{
  "name": "davidsmi8caret",
  "vmSize": "Standard_D2_v2",
  "maxTasksPerNode": 8,
  "poolSize": {
    "dedicatedNodes": {
      "min": 4,
      "max": 4
    },
    "lowPriorityNodes": {
      "min": 4,
      "max": 4
    },
    "autoscaleFormula": "QUEUE"
  },
  "containerImage":
    "rocker/tidyverse:latest",
  "rPackages": {
    "cran": ["xgboost", "e1071"],
    "github": [],
    "bioconductor": []
  },
  "commandLine": []
}
```



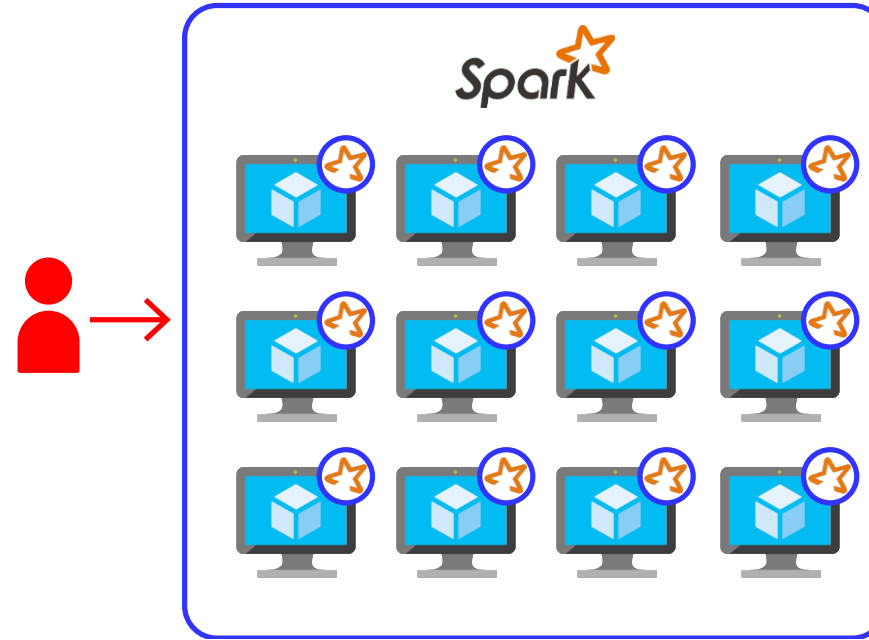
```
=====
Id: job20180126022301
chunkSize: 1
enableCloudCombine: TRUE
packages:
    caret;
errorHandling: stop
wait: TRUE
autoDeleteJob: TRUE
=====
Submitting tasks (1250/1250)
Submitting merge task. . .
Job Preparation Status: Package(s) being installed.....

Waiting for tasks to complete. . .
| Progress: 13.84% (173/1250) | Running: 59 | Queued: 1018 | Completed: 173 | Failed: 0 |
```

Current cores	16	Operating System	Canonical UbuntuServer 16.04-LTS (latest)
Dedicated nodes	4	VM size	standard_d2_v2
Low-priority nodes	4	Allocation state	Steady



MY LAPTOP: 78 minutes
THIS CLUSTER: 16 minutes
(almost 5x faster)



For when it's not complex parallel:

DISTRIBUTED DATA PROCESSING WITH SPARKLYR

What is Spark?

- Distributed data processing engine
 - Store and analyze massive volumes in a robust, scalable cluster
- Successor to Hadoop
 - in-memory engine 100x faster than map-reduce
- Highly extensible, with machine-learning capabilities
 - Supports Scala, Java, Python, R ...
- Managed cloud services available
 - Azure Databricks & HDInsight, AWS EMR, GCP Dataproc
- Largest open-source data project
 - Apache project with 1000+ contributors



R and Spark: Sparklyr

- sparklyr: R interface to Spark
 - open-source R package from RStudio
- Move data between R and Spark
- “References” to Spark Data Frames
 - Familiar R operations, including dplyr syntax
 - Computations offloaded to Spark cluster, and deferred until needed
 - CPU/RAM/Disk consumed in cluster, not by R
- Interfaces to Spark ML algorithms



spark.rstudio.com

Provisioning clusters for Sparklyr

- **Azure:** Command-line interface to provision Spark-ready (and Sparklyr-ready) clusters in Azure Batch
 - www.github.com/azure/aztk
- **AWS:** Configure EMR cluster with software components
 - Guide: spark.rstudio.com/examples/yarn-cluster-emr
- **H2O:** via the **rsparkling** package

dplyr with Sparklyr

Connect to the Spark cluster:

```
library(sparklyr)
cluster_url <- paste0("spark://", system("hostname -i", intern = TRUE), ":7077")
sc <- spark_connect(master = cluster_url)
```

Load in some data:

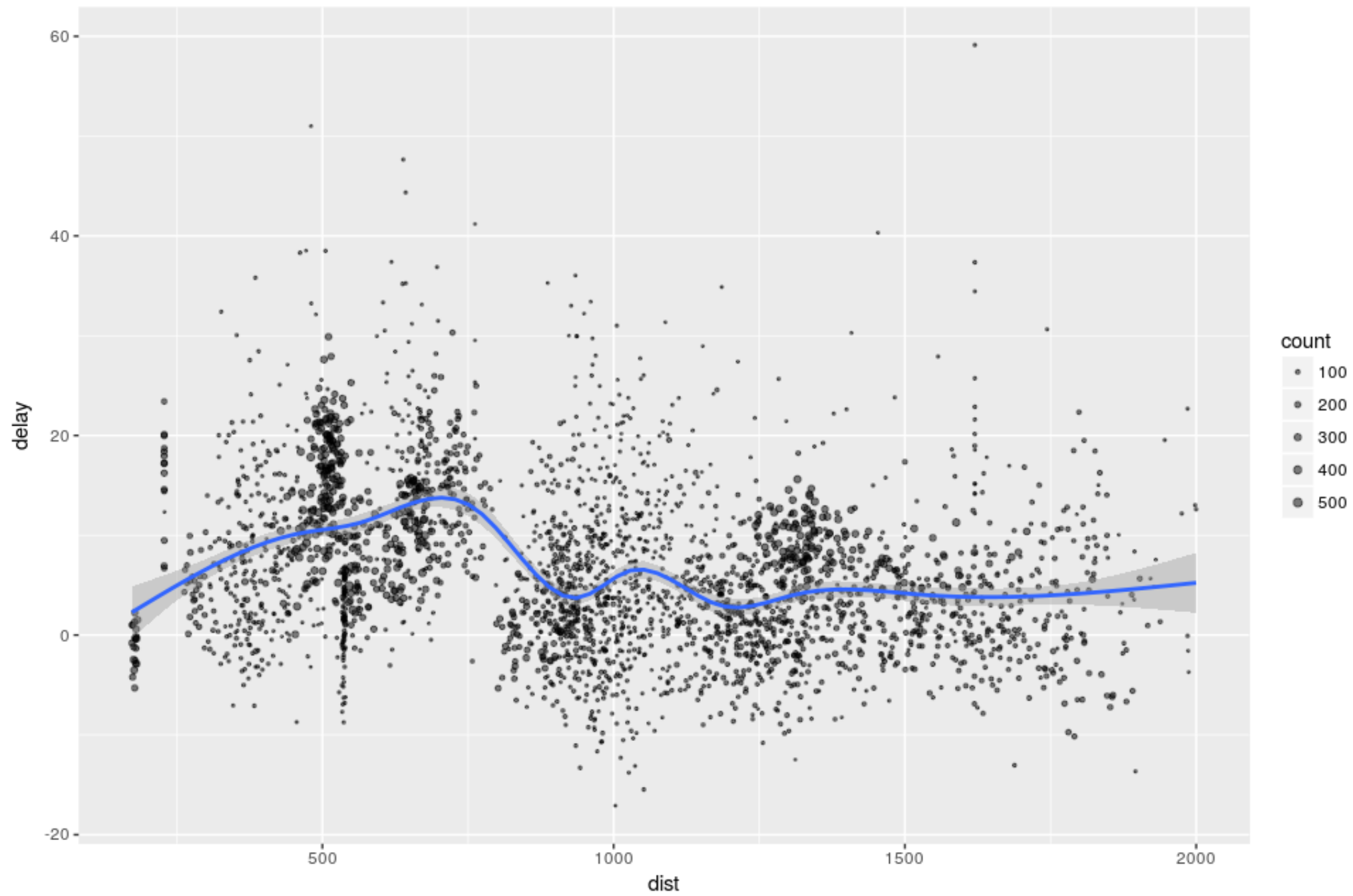
```
library(dplyr)
flights_tbl <- copy_to(sc, nycflights13::flights, "flights")
```

Munge with dplyr:

```
delay <- flights_tbl %>%
  group_by(tailnum) %>%
  summarise(count = n(), dist = mean(distance), delay = mean(arr_delay)) %>%
  filter(count > 20, dist < 2000, !is.na(delay)) %>%
  collect
```

Things to note

- All of the computation take place in the Spark cluster
 - Computations are delayed until you need results
 - Behind the scenes, Spark SQL statements are being written for you
- None of the data comes back to R
 - Until you call `collect`, when it becomes a `tbl`
 - It's only at this point you have to worry about data size
- This is all ordinary `dplyr` syntax



Machine Learning with SparklyR

SparklyR provides R interfaces to Spark's distributed machine learning algorithms (MLlib)

Computations happening in the Spark cluster, not in R

```
> m <- ml_linear_regression(delay ~ dist, data=delay_near)
* No rows dropped by 'na.omit' call
> summary(m)
Call: ml_linear_regression(delay ~ dist, data = delay_near)

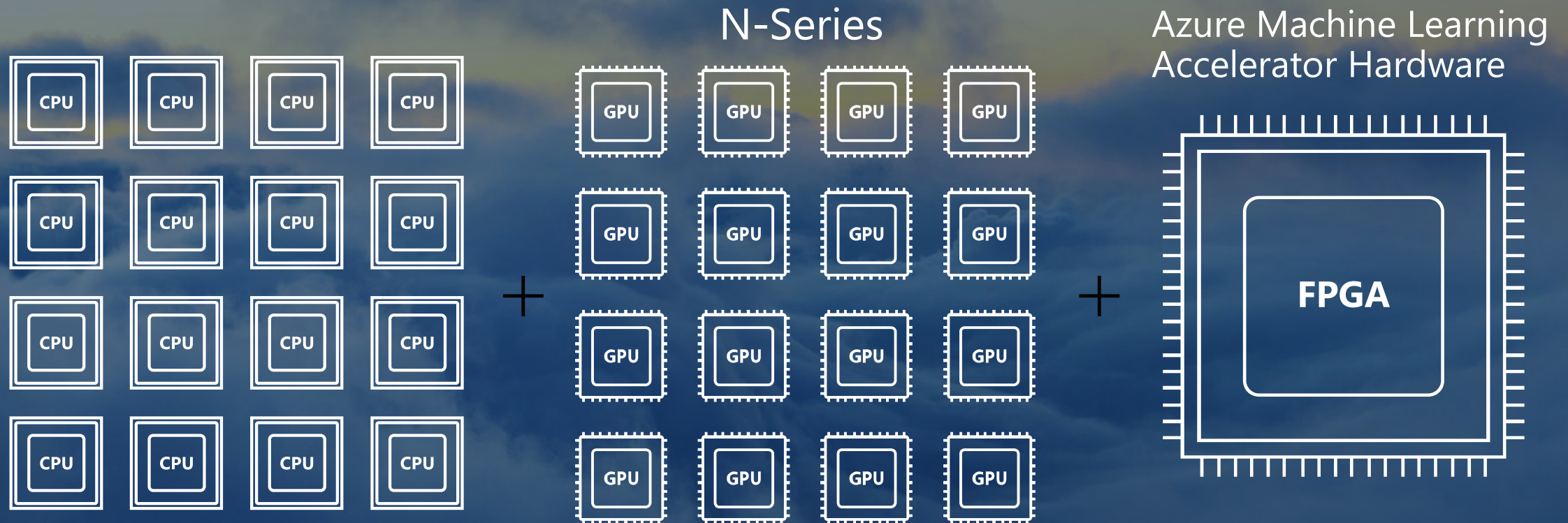
Deviance Residuals::
      Min       1Q   Median       3Q      Max 
-19.9499  -5.8752  -0.7035   5.1867  40.8973 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.6904319   1.0199146   0.677   0.4986
dist         0.0195910   0.0019252  10.176 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

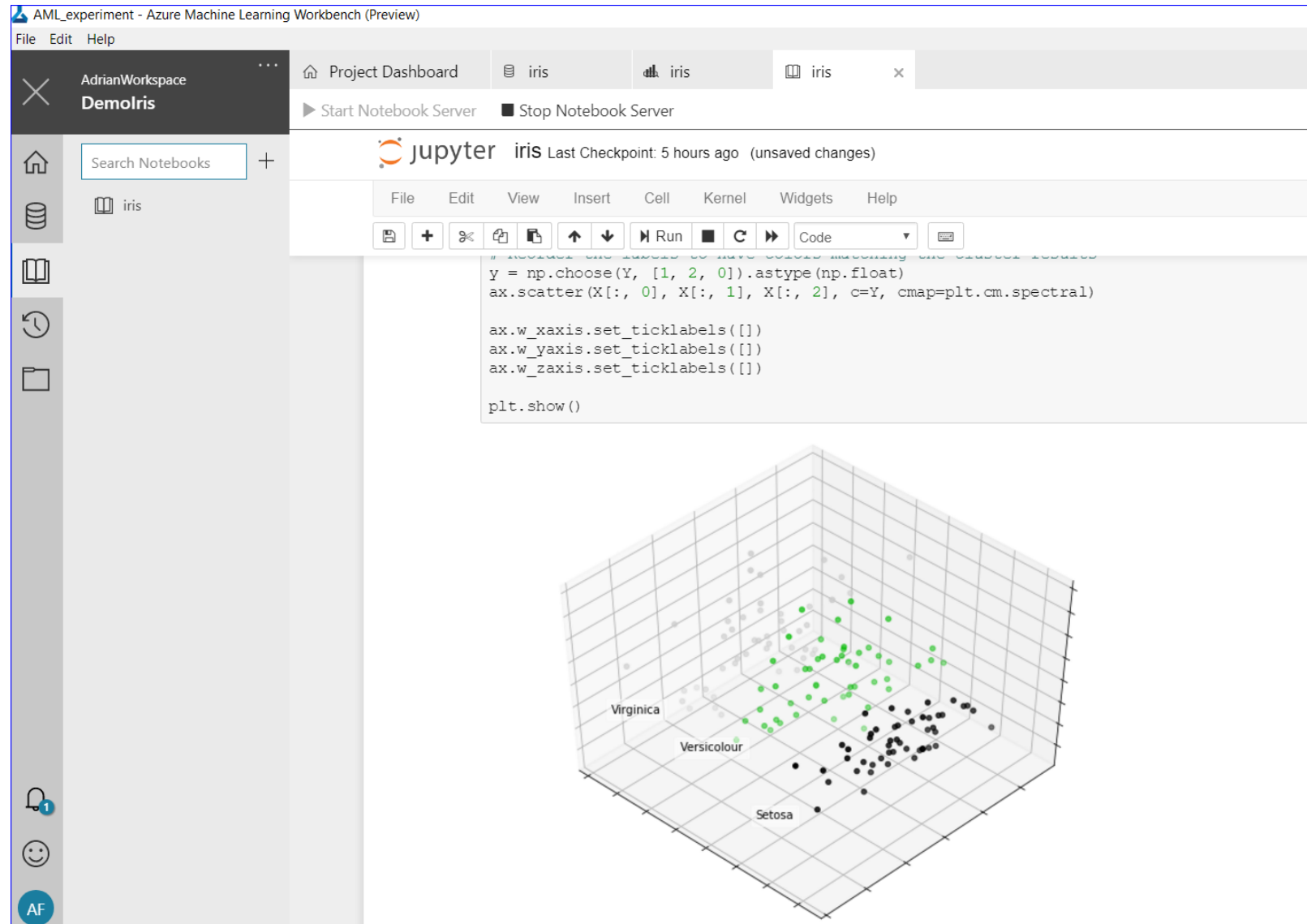
R-Squared: 0.09619
Root Mean Squared Error: 8.075
>
```

Azure

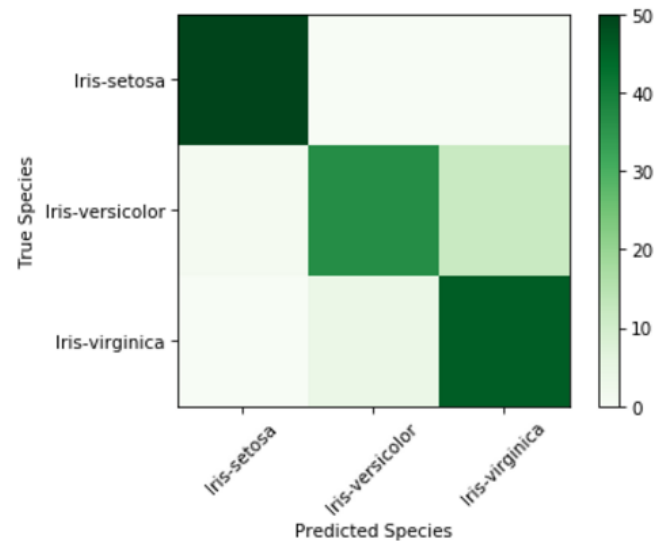
AI Supercomputer



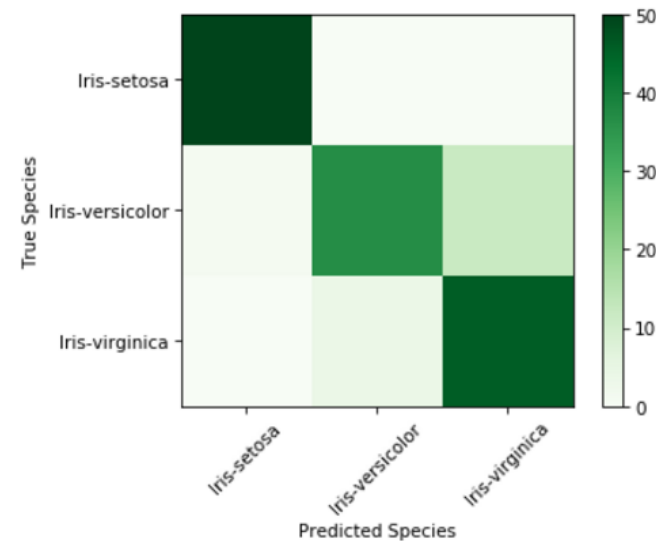
Azure Machine Learning Workbench



outputs/cm.png

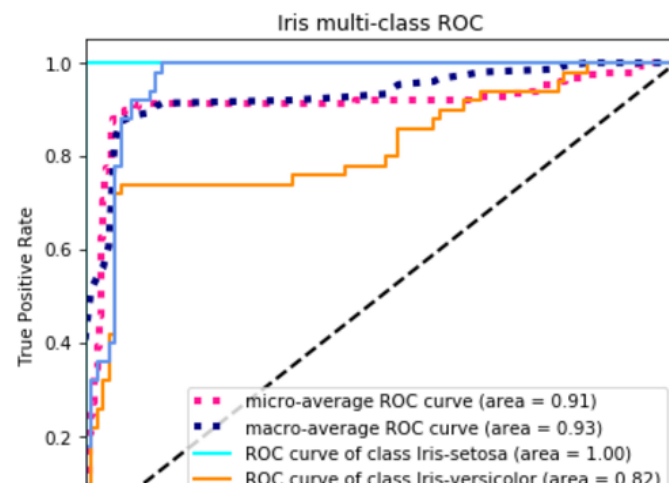


outputs/cm.png

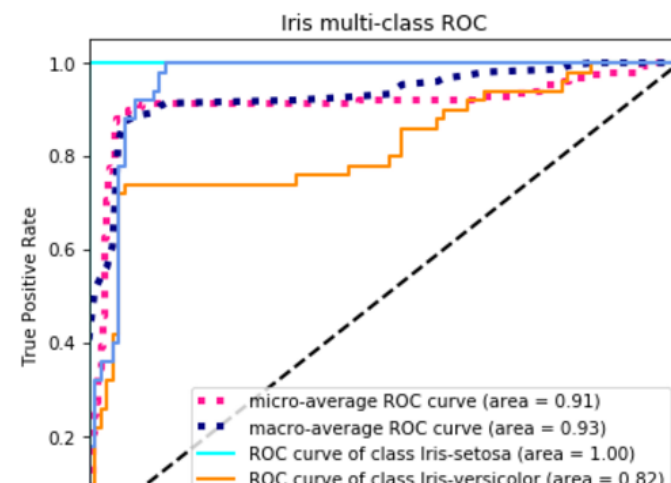


Graphs

outputs/roc.png



outputs/roc.png



Jobs

iris_sklearn.py [13]

✓ Completed

Today at 5:22 PM

plot_graphs.py [9]

✓ Completed

02/02/2018

iris_sklearn.py [6]

✓ Completed

02/02/2018

plot_graphs.py [5]

✓ Completed

02/02/2018

iris_sklearn.py [1]

✓ Completed

02/02/2018

Dismiss All Finished

Azure Machine Learning Studio

The screenshot displays the Azure Machine Learning Studio interface for a workspace named "ktakeda1's Workspace". The main title of the experiment is "Time Series Forecasting", which is currently in "draft" status.

Left Panel (Navigation):

- Search experiment items
- Saved Datasets
- Data Format Conversions
- Data Input and Output
- Data Transformation
- Feature Selection**
 - Filter Based Feature Sele...
 - Fisher Linear Discrimina...
 - Permutation Feature Im...
- Machine Learning**
 - Evaluate
 - Initialize Model
 - Score
 - Train
- OpenCV Library Modules**
 - Image Reader
 - Pre-trained Cascade Ima...
- Python Language Modules**
 - Execute Python Script
- R Language Modules**
- Statistical Functions**
 - Apply Math Operation
 - Descriptive Statistics
 - Elementary Statistics

Center Panel (Workflow):

The workflow diagram illustrates a time series forecasting process:

- Time Series Dataset - Copy**: The initial data source.
- Metadata Editor**: A step to rename the N1725 column to data.
- Split Train Test Split**: The data is partitioned into training and testing sets.
- Parallel Processing**: The workflow branches into five parallel paths, each involving:
 - Execute R Script**: Running different R models (e.g., ARIMA Seasonal, ARIMA Non Seasonal, AVERAGE SEASONAL ETS AN..., ETS Seasonal, ETS Non Seasonal).
 - Join**: Joining the forecast with observed data.
 - Execute R Script**: Generating metrics for each model.
 - Add Rows**: Combining the results from the parallel paths.

Right Panel (Properties):

- Experiment Properties**
 - START TIME: -
 - END TIME: -
 - STATUS CODE: InDraft
 - STATUS DETAILS: None
 - ☐ Disable upgrades
- Summary**
 - Time Series Forecasting with Azure ML using R
- Description**
 - Enter the detailed description for your experiment.
- Quick Help**

Bottom Bar (Actions):

NEW, VIEW RUN HISTORY, SAVE, SAVE AS, DISCARD CHANGES, REFRESH, CANCEL, RUN, PREPARE WEB SERVICE, PUBLISH TO GALLERY, CREATE SCORING EXPERIMENT.

Drag & Drop + Los mejores Algoritmos ML

The screenshot displays the ML Studio interface, which is designed for building machine learning workflows through a drag-and-drop mechanism. The interface is divided into several key sections:

- Top Bar:** Includes the "ML Studio" logo, a search bar, a feedback input field ("Enter feedback here"), the current workspace name "Fraud Detection Workspace", and a menu icon.
- Left Panel (Navigation):** Contains a sidebar with icons for home, machine learning, and evaluation. A search bar is also present. The main list of modules is organized into categories:
 - Machine Learning:**
 - Evaluate:** Cross Validate Model, DEPRECATING Binary Class Metrics Evaluator, Evaluate Model, Recommender Evaluator.
 - Initialize Model:**
 - Classification:** Averaged Perceptron Binary Classification Model, Boosted Decision Tree Binary Classification Model, Logistic Regression Binary Classification Model, Multiclass Logistic Regression Classification Model, Neural Network Binary Classification Model, Neural Network Multiclass Classification Model, One-vs-All Multiclass Classification Model, Random Forest Binary Classification Model, Support Vector Machine Binary Classification Model.
 - Clustering:** (Currently collapsed).
 - Regression:** Bayesian Linear Regression Model, Boosted Decision Tree Regression Model, Linear Regression Model, Neural Network Regression Model.

- Central Canvas:** The main workspace for building the workflow. It features a dashed-line box labeled "Drag Items Here" with an arrow pointing to it. The workflow itself is a vertical sequence of modules connected by arrows:
- Dataset (represented by a cylinder icon).
- Split (represented by a document icon).
- Train (represented by a document icon).
- Model (represented by a flask icon).
- Score (represented by a flask icon).
- Right Panel:** A "Properties" panel for configuring the selected module.
- Bottom Bar:** A toolbar with various actions: NEW, VIEW RUN HISTORY, SAVE, SAVE AS, DISCARD CHANGES, REFRESH, CANCEL, RUN, and PUBLISH WEB SERVICE. A zoom slider and a 1:1 zoom button are also present.

Visual Studio 2017 con R Tools

1-Getting_Started_with_R.R - Microsoft Visual Studio

File Edit View Project Debug Team Tools Test AI Tools R Tools Analyze Window Help

1-Getting_Started_with_R.R

```
scale_y_log10()

### Regression Diagnostics

# It is easy to get regression diagnostic plots. The same plot function that plots points either with a formula

# Look at some model diagnostics.
# check to see Q-Q plot to see linearity which means residuals are normally distributed

par(mfrow = c(2, 2)) # Set up for multiple plots on the same figure.
plot(model, col = "blue")
par(mfrow = c(1, 1)) # Rest plot layout to single plot on a 1x1 grid

### The Model Object

# Finally, let's look at the model object. R packs everything that goes with the model, e.g. the formula and
str(model)
model$coefficients # note this is the same as coef(model)

# Now fit a new model including more columns
model <- lm(log(price) ~ log(carat) + ., data = diamondSample) # Model log of price against all columns

summary(model)
# R-squared = 0.9824, i.e. model explains 98.2% of variance, i.e. a better model than previously
```

R Interactive - Disconnected

```
.. .. - attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
.. .. - attr(*, "names")= chr [1:2] "log(price)" "log(carat)"
- attr(*, "class")= chr "lm"
(Intercept) log(carat)
8.448104 1.676937
>
```

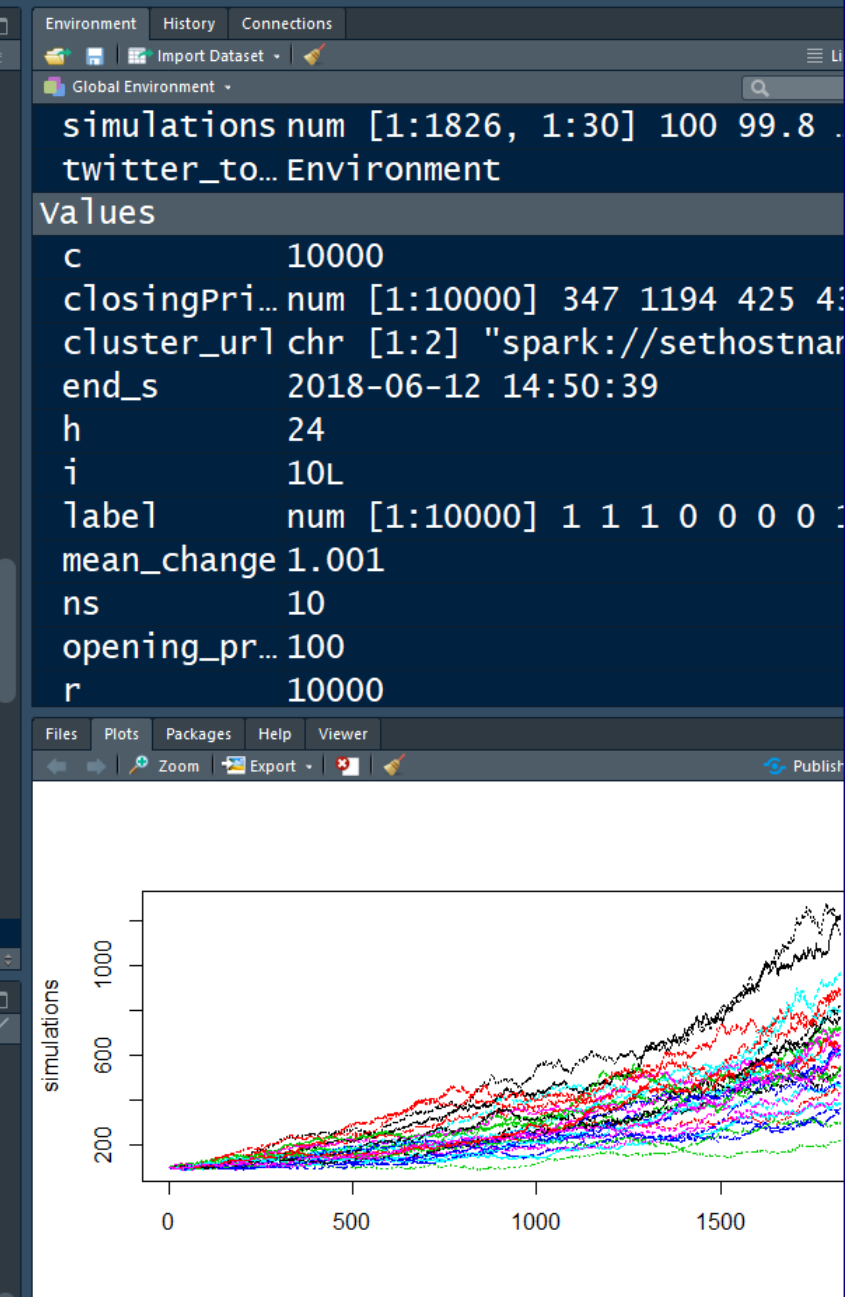
Output R Interactive Node.js Interactive Window

R Plot - Device 3 - Active

The R Plot window displays four diagnostic plots for a linear model:

- Residuals vs Fitted:** A scatter plot of residuals against fitted values. The residuals are centered around zero, indicating a good fit.
- Normal Q-Q:** A plot of standardized residuals against theoretical quantiles. The points follow a straight line, suggesting the residuals are normally distributed.
- Scale-Location:** A plot of the square root of absolute standardized residuals against fitted values. The points are randomly distributed, indicating constant variance.
- Residuals vs Leverage:** A plot of standardized residuals against leverage. A Cook's distance line is shown, and most points are below it, indicating no significant influence of individual observations.

```
70 simulateMovement <- function() {
71   days <- 1825 # ~ 5 years
72   movement <- rnorm(days, mean=mean_change, sd=volatility)
73   path <- cumprod(c(opening_price, movement))
74   return(path)
75 }
76
77 simulations <- replicate(30, simulateMovement())
78 matplot(simulations, type='l') # plots all 30 simulations on a graph
79
80
81
82
83 # Estimate runtime for 10 million (linear approximation)
84 1000 * difftime(end_s, start_s, unit = "min")
85
86 # Run 10 million simulations with doAzureParallel
87
88 # We will run 100 iterations where each iteration executes 100,000 simulations
89 opt <- list(chunkSize = 20) # optimize runtime. Chunking allows us to run multiple
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



CATEGORIES ▾

☒ Solution

☐ Project

☐ Model

☐ Experiment

☐ Machine Learning API

☐ Custom Module

☐ Tutorial

☐ Collection

☐ Notebook

☐ Classroom Training

☐ Video Training

SHOW ▾

☐ Microsoft content only

TAGS ▾

☐ R

☐ Classification

☐ Solution


☐ Linear Regression

You've selected:

Solution ✕


[Clear all](#)




 SOLUTION


Azure Databricks Spark Streaming

Analyzing data streams in real-time using Azure Databricks Spark Streaming, Azure Event Hubs, Cognitive Services Text Analytics, and Time Series Insights.

 560

 92

24 days ago








 SOLUTION

Image Similarity with SQL Server

This solution uses SQL Server 2017 + ML Services with Python to execute a transfer learning algorithm to detect image similarity.

 599

 60

7 days ago





 SOLUTION

Geo AI Data Science Virtual Machine

The Geo AI Data Science Virtual machine (GeoAI DSVM) is a custom Azure VM built on Windows with many popular tools for data science modeling/development, with a focus on Geographic Information Systems.


 1.3K

 162

one month ago







 SOLUTION


Text Classification with SQL Server

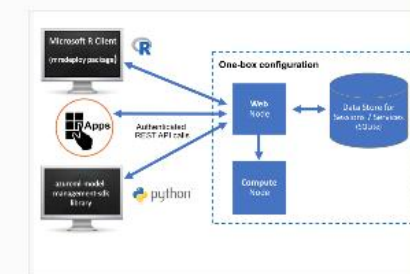
This solution uses SQL Server 2017 + ML Services with R or Python to train a machine learning model to categorize text.

 2K

 184

7 days ago







 SOLUTION


Machine Learning Server Operationalization One-box ...

This solution installs Microsoft Machine Learning Server on Windows VM and configures as One-box to act as a deployment server and to host analytic web services for operationalization.

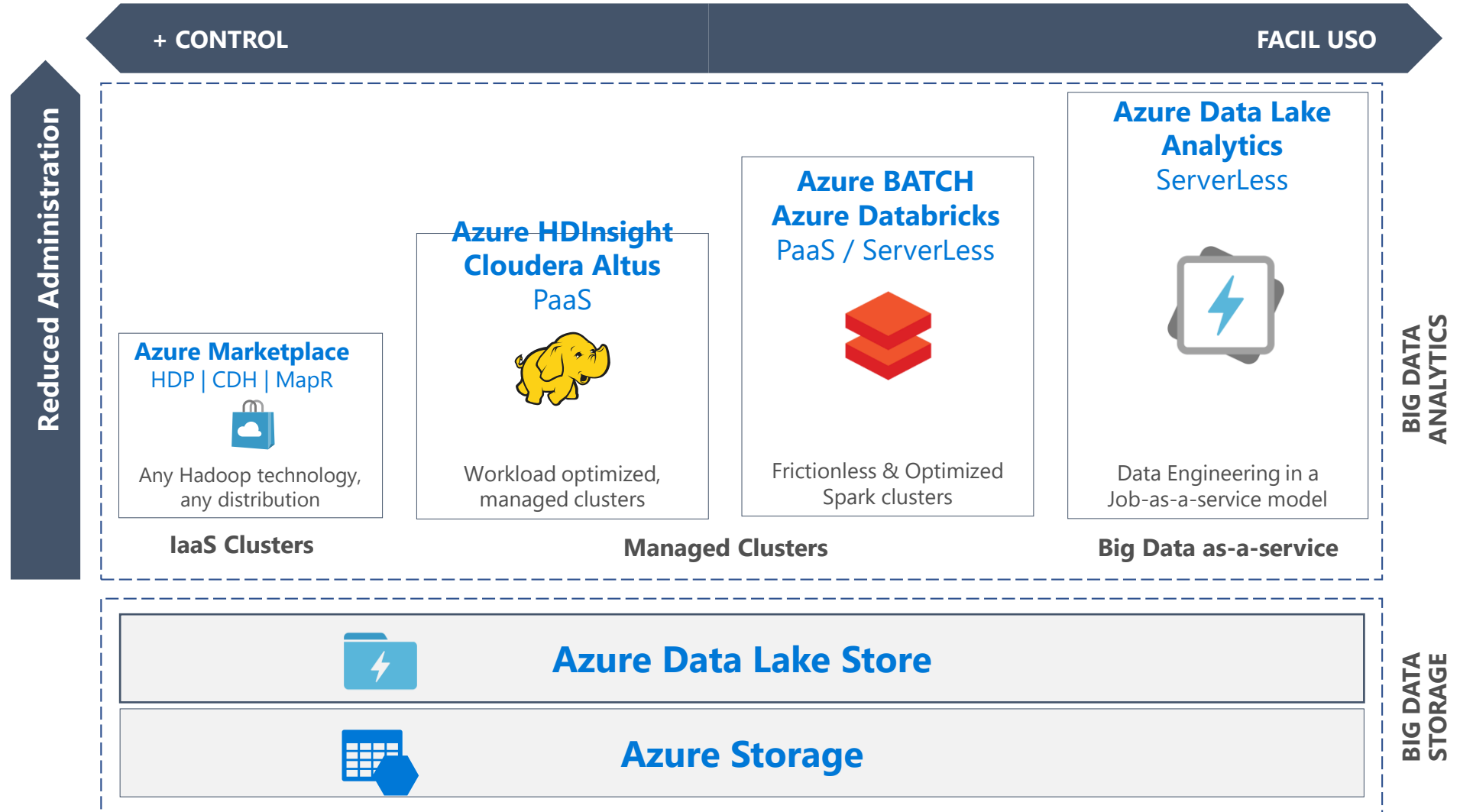
 578

 61

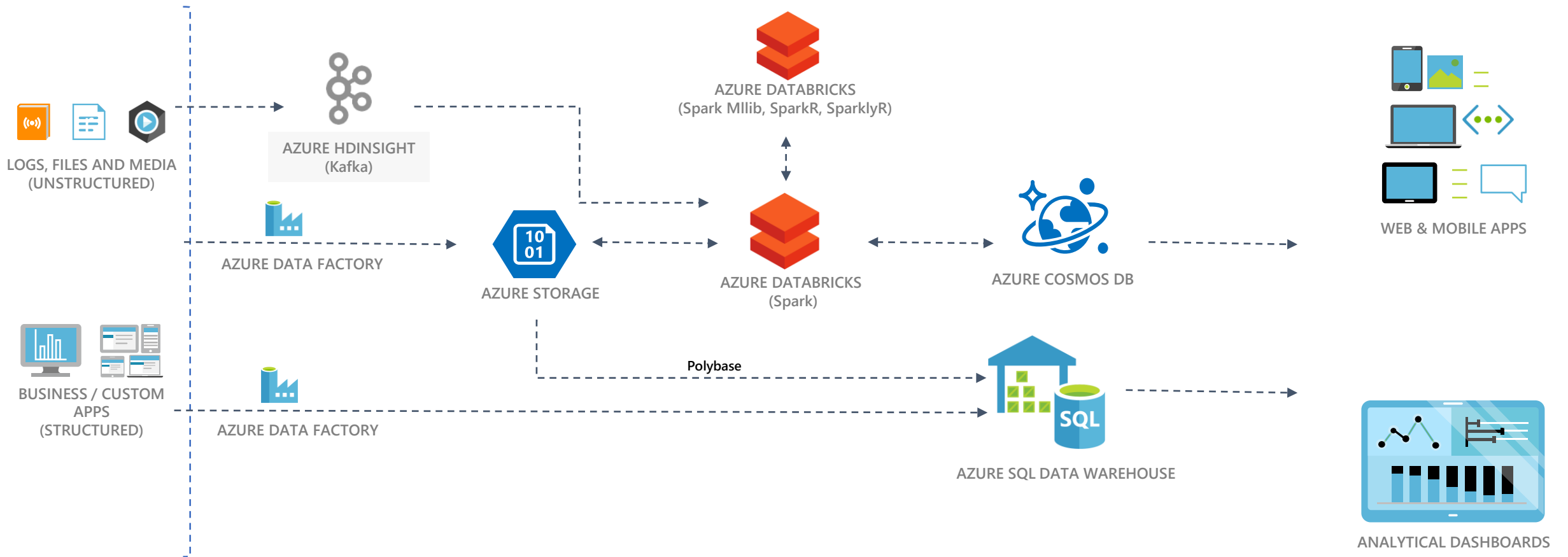
3 months ago



TODAS LAS SOLUCIONES BIG DATA EN AZURE



Big Data Lambda Architecture



In summary

- Embarrassingly parallel (small): foreach + local backend
- Embarrassingly parallel (big): foreach + cluster/multicore backend
- distributed data, not embarrassingly parallel: Spark + sparklyr
- Databricks, cost effective Spark
- GPU + FPGA

Adrián J. Fernández

Technology Solutions Professional
Advanced Analytics & Artificial Intelligence

Adrian.Fernandez@microsoft.com

Microsoft Chile

Av. Vitacura 6844, Vitacura, Santiago, Chile



Microsoft