

Lab: R in DSXL

Mar 1st, 2018

*Author: Elena Lowery elowery@us.ibm.com
Updated By: Sidney Phoon yfphoon@us.ibm.com*

Table of contents	
Overview	1
Required software, access, and files	1
Part 1: Create a DSX Project and run R notebook and Shiny	1
Part 2: Publish Shiny	4
Part 3: Check Status of Shiny Server	6

Overview

In this lab you will work with various R assets in DSX Local.

Required software, access, and files

- To complete this lab, you will need access to a DSX Local cluster.

Part 1: Create a DSX Project and run R notebook and Shiny

- Log in to a **DSX Local cluster**.
- If you have not already created the DSX_Local_Workshop project do so now.
- Switch to the **Notebooks** tab in **Assets**. Open the *DriverClassification* notebook.

NAME	STATUS	ENVIRONMENT	TOOL	LAST MODIFIED
DriverClassification	○	Jupyter with Python 2.7, Scala 2.11, R 3.4.3, Spark 2.0.2	JUPYTER	06-24-2018
TelcoChurn_Zepplin		Zeppelin with Anaconda2, Python 2.7	ZEPPELIN	06-24-2018
CreditCardDefault_SKLearn		Jupyter with Python 2.7, Scala 2.11, R 3.4.3, Spark 2.0.2	JUPYTER	06-24-2018
TelcoChurn_SparkML		Jupyter with Python 2.7, Scala 2.11, R 3.4.3, Spark 2.0.2	JUPYTER	06-24-2018

- Work through the notebook.

Note: notice that in the notebook we save the model into the RStudio directory. We use this approach because when we open RStudio, the model is displayed in the UI, which makes it easier for integration with the Shiny application.

In the sample notebook we overwrite the file each time we run the "save code", but additional code can be added to add a version number to the model name.

At this time DSX doesn't provide any additional "model management features" for R models.

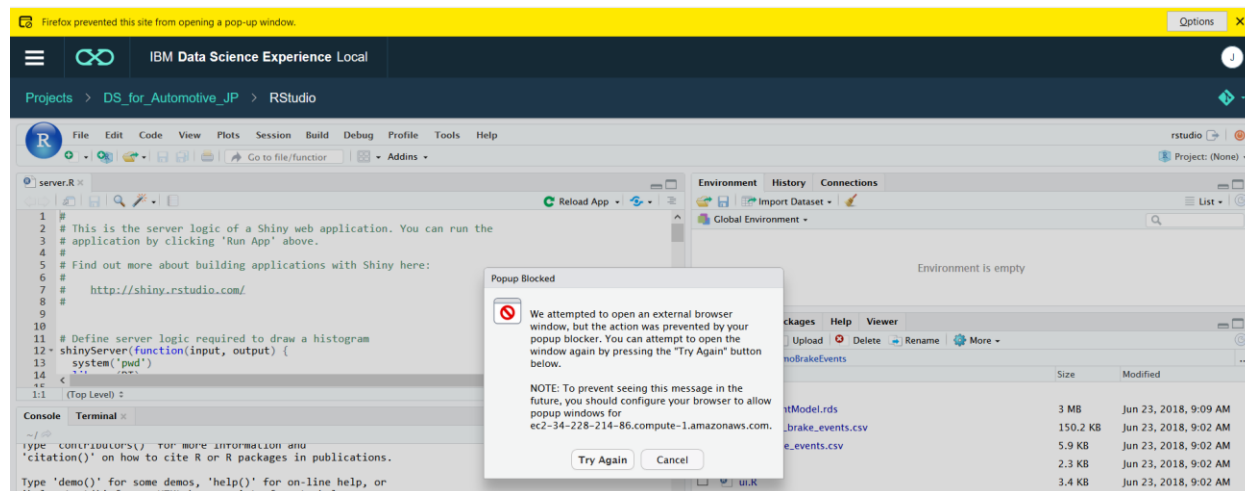
5. Open **RStudio** from the Project view.
6. If `serverR` file is not already loaded in the top left window, click on the `demoBreakEvents` folder in the file explorer (right bottom window), then click on `serverR`.

After the application is loaded, click the **Run App** button.

```

1 #
2 # This is the server logic of a Shiny web application. You can run the
3 # application by clicking 'Run App' above.
4 #
5 # Find out more about building applications with Shiny here:
6 #
7 #   http://shiny.rstudio.com/
8 #
9 #
10 #
11 # Define server logic required to draw a histogram
12 shinyServer(function(input, output) {
13
14   library(shiny)
15   library(data.table)
16   library(ggplot2)
17   library(randomForest)
18
19   ## load data and model
20   newEventsDF <- read.csv("new_brake_events.csv")
21   oldEventsDF <- read.csv("historical_brake_events.csv")
22
23   brakeEventModel <- readRDS(file = "brakeEventModel.rds")
24
25   ## Render plot and conditional statement for new points
26   output$plot <- renderPlot({
1:1 (Top Level)
  
```

7. **Allow popups** or select **Try Again** to bring up the application.



8. Test the application - both the Explore and Model tabs.

Analyze Recent Brake Events








Part 2: Publish Shiny

In this section we will publish the Shiny application.

1. Navigate back to the **Project** view and click **Publish** from the ellipses next to the Shiny application (*demoBreakEvents*) in **RStudio**. Note: you may have to **view all** in order to see the application (it be will of type SHINY)

RStudio view all (8) ⊕ open RStudio

NAME	TYPE	LAST MODIFIED	
 spark_mtcars	R	01-08-2018	⋮
 spark_kernel_basic	R	01-08-2018	⋮
 spark_flights	R	01-08-2018	⋮
 readme	TXT	01-08-2018	⋮
 demoBreakEvents	SHINY	01-08-2018	⋮

2. If you get an error during publish, check with the lab instructor.
3. Provide *Published name* (try to make it unique, for example, add your initials) and select *Published content visibility* that you would like to test. Save the permalink to the Shiny app. Click **Publish**.

Note: All Shiny applications in DSX are published to a shared Shiny server. At this time there is only 1 instance of the Shiny server in DSX (i.e. it's not configured for HA). However, because the Shiny server is deployed as a pod, Kubernetes will monitor its status and restart it, when needed.

Shiny app name
demoBreakEvents

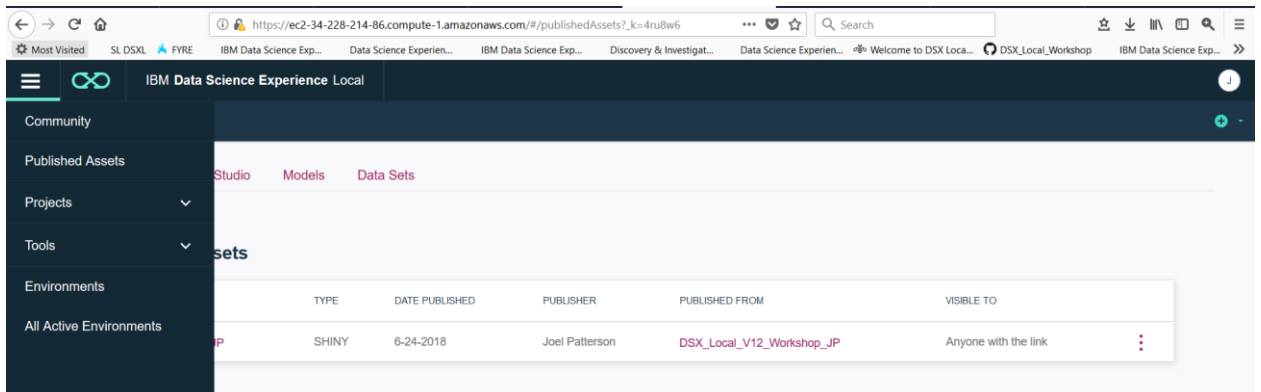
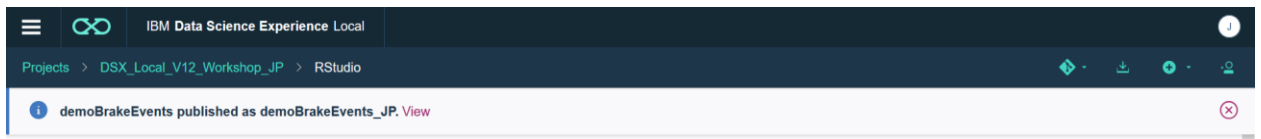
Published name *
ShinyApp_EL ✓

Description

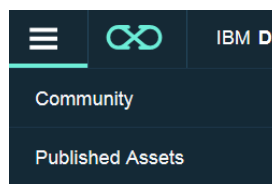
Published content visibility
☐ Anyone with the link
☒ Any authenticated user
☐ Restricted to collaborators in the selected project
 DSX_Local_Workshop ▼

Permalink to read-only published shiny app
https://9.30.230.119/#/publishedAssets/view/dsx-publish-view-shiny-all-auth%2F%5E%5Eall_auth%2Frstudio%2FShinyApp_EL%2F 

- Click on the status message (**View**) or navigate to **Published Assets** from the main DSX menu. Note: it will not show in Published Assets project tab unless you published exclusive to the project.



- Verify that the application works.
- Depending on the “scope” that you published to, the Shiny application can be accessed through different views in DSX.
 - If selected *anyone with the link* or *any authenticated user*, then you can view it in the **Published Assets** accessed through the main DSX menu (in the top left corner)



- If you published it to a specific project, then it will show up in the **Published Assets** tab of the project you published **to** (not published **from**).
- If published globally make sure the link works (try this by starting a different browser and trying the link there).

Part 3: Check Status of Shiny Server

At this time the Shiny Server is deployed as a pod in DSX, and the only way to view status is by looking at the status of the pod. You must have admin privileges to perform these functions.

There is only 1 instance of Shiny Server, but Kubernetes monitors pod status, and the pod should be automatically restarted if it goes down.

1. Switch to the **IBM Data Platform Manager** view
2. Click on Pods. Scroll down to locate the *r-publish...* pod. This is the Shiny Server.

r-publish-content-3361764361-dnphb	✓	dsx1-jan07-0-storage-1.fyre.ibm.com
------------------------------------	---	-------------------------------------

3. If the pod is not running, you can redeploy it from this UI or via Kubernetes commands in ssh.

Display all pods: `kubectl get pods --all-namespaces`

Delete pod: `kubectl delete pod <pod name> -n <namespace>`

Force delete for a pod (if it's stuck in "terminating" state):

`kubectl delete pod <pod name> -n <namespace> --grace-period=0`